Jornadas Sarteco

20 - 22 septiembre 2023, Ciudad Real

sarteco

UCLM

AVANCES EN ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

Actas de las Jornadas SARTECO 2023

Editado por:

Arturo González-Escribano

Jesús Barba Romero

Diego R. Llanos

Soledad Escolar Díaz

Julián Caba Jiménez

Fernando Rincón Calle

J S

CIUDAD REAL 2023

Avances en arquitectura y tecnología de computadores

Actas de las Jornadas SARTECO 2023

Ciudad Real, 20 a 22 de Septiembre de 2023

UNIVERSIDAD DE CASTILLA LA MANCHA

Avances en arquitectura y tecnología de computadores
Actas de las Jornadas SARTECO 2023

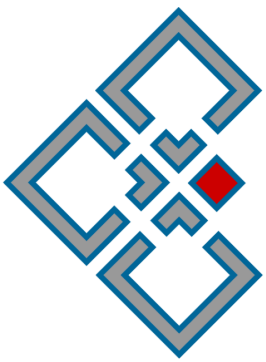
Editores: Arturo Gonzalez-Escribano, Jesús Barba Romero, Diego R. Llanos, Soledad Escolar Díaz, Julián Caba Jiménez, Fernando Rincón Calle

ISBN-13: 978-84-09-54466-0

Ciudad Real, 2023

Entidades organizadoras

sarteco



Universidad de Castilla-La Mancha

Entidades colaboradoras

Ciudad Real
AYUNTAMIENTO

DIPUTACIÓN DE
CIUDAD REAL

inwire

Junta de Comunidades de
Castilla-La Mancha

Entidades patrocinadoras

Empresas SARTECO-PRO

AVNET® SILICA

cojali

INNOVATION & TECHNOLOGY

semidynamics

tecnobit

grupo oesía

Prólogo

La Sociedad de Arquitectura y Tecnología de Computadores (SARTECO) presenta, una vez más las Jornadas SARTECO, las cuales integran las XXXIII Jornadas de Paralelismo (JP2023) y las VII Jornadas de Computación Empotrada y Reconfigurable (JCER2023).

En el marco de las jornadas SARTECO se desarrollan las siguientes actividades:

- Séptima edición del Concurso "Tu Tesis en Tres Minutos"que pretende premiar los mejores trabajos de Tesis recientes en el área.
- Segunda edición del Concurso "Tu Trabajo Fin de Máster en Tres Minutos"que pretende hacer lo propio con los mejores Trabajos Fin de Máster.
- Tres *keynotes* impartidas por Davidd Puron (Barbara IoT), Enrique S. Quintana-Ortí (Universidad Politécnica de Valencia) y Tercera Cervero García (Barcelona Supercomputing Center).
- VI Encuentro Women in SARTECO con la ponencia titulada Çómo y porqué incluir la perspectiva de género en la docencia en ATC", a cargo de María Villarroya Gaudo, profesora Titular de Universidadd y Directora de Secretariado de Internacionalización en la Universidad de Zaragoza. Además, María es también promotora y coordinadora de #10001amigasingenieras y vocal de AMIT-Aragón.
- IV Edición de Sarteco-PRO (Sarteco-PROfesional) donde las empresas tienen la oportunidad de presentar su oferta tecnológica a la comunidad y también las oportunidades laborales a los/las jóvenes investigadores/as que asisten mayoritariamente a las Jornadas. Se fomenta, así, la comunicación entre el sector privado y el académico, contribuyendo a descubrir sinergias y crear nuevas oportunidades de colaboración.

La agenda científica de la edición 2023 de las Jornadas se completa con las presentaciones de un total de 101 artículos en 23 sesiones, distribuidos de la siguiente forma: 29 artículos y 7 sesiones dentro de las JCER y 72 trabajos en 16 sesiones en el ámbito de las JP. De esta forma se consolidan las Jornadas como el evento de referencia a nivel nacional para la comunidad científica y tecnológica en ámbitos como sistemas empotrados, lógica reconfigurable, computación de altas prestaciones y arquitecturas heterogéneas. En los más de 10 años de trayectoria de las Jornadas, éstas se perfilan como un foro indispensable alrededor del cual investigadores, académicos, estudiantes y empresas, comparten y dan a conocer sus trabajos, facilitando colaboraciones y sinergias entre grupos universitarios.

También se consolidan las figuras de asistentes, con 190 inscripciones de más de 30 universidades y centros tecnológicos españoles y de la Universidad Autónoma de Puebla (México). En esta edición, el porcentaje de asistentes mujeres ha superado olgadamente al de anteriores, alcanzando un 15 %, con un total de 28 investigadoras.

Y todo esto no sería posible sin al apoyo de las instituciones y empresas colaboradoras, que generosamente han contribuido al éxito de la organización de esta nueva edición de las Jornadas Sarteco. Por ello, el Comité de Organización desea mostrar su agradecimiento a todas ellas:

- Tecnobit Grupo Oesía.
- Cojali S.L.

- Semidynamics Technology Services, S.L.U.
- AVNET SILICA España
- Universidad de Castilla-La Mancha
- Escuela Superior de Informática
- Departamento de Tecnologías y Sistemas de Información
- Instituto de Tecnologías y Sistemas de Información
- Ayuntamiento de Ciudad Real
- Junta de Comunidades de Castilla-La Mancha

Este año, las Jornadas SARTECO se llevarán a cabo en la hermosa Ciudad Real, en las instalaciones del prestigioso Hotel Guadiana. Para nosotros es emocionnante acogeros durante los tres días que duran las Jornadas que, si bien serán de intenso trabajo, intentaremos disfrutéis de la ciudad y del entorno, con una no menos intensa agenda de eventos sociales.

Desde la organización de estas nuevas Jornadas SARTECO, os deseamos a todos y a todas, un fructífero trabajo y una placentera estancia en Ciudad Real.

Jesús Barba Romero

Comités de coordinación

Comité de Dirección de las Jornadas SARTECO

Francisco Quiles Flor (UCLM, Presidente)
Víctor Viñals Yúfera (UNIZAR, Vicepresidente)
Gracia Ester Martín Garzón (UAL, Secretaria)
Francisco Tirado Fernández (UCM, Presidente de Honor)

Comité Organizador de las Jornadas Sarteco 2023

Jesús Barba Romero (UCLM)
Juan Carlos López López (UCLM)
Félix Jesús Villanueva Molina (UCLM)
Soledad Escolar Díaz (UCLM)
Julián Caba Jiménez (UCLM)
Fernando Rincón Calle (UCLM)
María José Santofimia Romero (UCLM)
José Antonio de la Torre de las Heras (UCLM)
Xavier del Toro García (UCLM)
David Villa Alises (UCLM)
Diego R. Llanos Ferraris (UVa)
Arturo González Escribano (UVa)

Comité T3M2023 y TFM3M2023

Francisco Quiles Flor (UCLM) (Presidente)
Gracia Ester Martín Garzón (UAL) (Secretaria)
Francisco Tirado Fernández (Presidente de Honor)
Dora Blanco Heras (USC)
Vicente J. Blanco Pérez (ULL)

Indice

Arquitecturas heterogéneas y modelos de programación para GPU, FPGA y aceleradores de IA

Aplicaciones de la computación de altas prestaciones

Arquitecturas del subsistema de memoria y almacenamiento secundario

Redes y Comunicaciones

Vehículos aéreos no tripulados

Arquitecturas heterogéneas

Docencia y sistemas ciberfísicos

Ponencias invitadas

Gateway Edge Computing: Inteligencia Artificial ejecutada en dispositivos embebidos

David Puron

Resumen

Según muchos analistas, la capacidad de ejecutar Inteligencia Artificial en dispositivos embebidos será la tendencia tecnológica más transformadora en los próximos 10 años. Esto se llama "Edge-AI" y no solo mejora el rendimiento y el coste del análisis de datos, sino que también es una forma de cumplir con los estándares de ciberseguridad y privacidad más relevantes del mercado. En esta ponencia, David Puron presentará qué es Edge-AI, proyectos reales en los que se usa, cuáles son los tipos de dispositivos "Edge" y sus arquitecturas, así como los retos más comunes que se encuentran las empresas e ingenieros al aplicar esta tecnología.

Biografía

David Purón es CEO de Barbara, Ingeniero de telecomunicaciones con más de 20 años de experiencia en posiciones de ingeniería y dirección ejecutiva. Tras su paso por multinacionales como Telefónica o Huawei, y startups de Silicon Valley, co-fundó Blackphone, proyecto que revolucionó en 2013 el mundo de la ciberseguridad móvil, obteniendo premios del MIT o la revista Time entre otros. David es considerado a día de hoy uno de los nuevos directivos de más renombre en el ecosistema nacional de la tecnología, incluido en la lista Forbes 100 de emprendedores creativos españoles. Actualmente lidera la Barbara, empresa líder en Edge Computing, que recientemente ha recibido una inversión de 2,5 M de euros de Caixa Capital Risc, a través de su fondo Criteria Venture Tech, e Iberdrola, a través de su fondo Perseo.

Una Visión Computacional de las Redes Neuronales

Enrique S. Quintana-Ortí

Resumen

La multiplicación de matrices (GEMM) es un núcleo computacional clave, omnipresente en numerosos ámbitos. Por un lado, muchas aplicaciones científicas requieren la resolución de sistemas lineales de ecuaciones, problemas de mínimos cuadrados y problemas de valores propios. Por razones de portabilidad y rendimiento, estas aplicaciones se construyen sobre la operación GEMM. Por otro lado, las redes neuronales convolucionales para tareas de procesamiento de señales y visión por computador, así como los modelos utilizados en herramientas de aprendizaje profundo como ChatGPT, presentan un coste computacional fuertemente determinado por el rendimiento de GEMM. En esta ponencia, en primer lugar se expondrán los problemas de las instancias actuales de GEMM en bibliotecas para arquitecturas multinúcleo convencionales: rendimiento subóptimo y falta de soporte para tipos de datos orientados a aprendizaje profundo. Partiendo de ese punto, se demostrará cómo pueden superarse estos problemas mediante herramientas para la generación automática de código, junto con un modelo analítico de la configuración de la jerarquía de caché del procesador. Además, se ilustrará que este enfoque se puede aplicar también a arquitecturas más "exóticas", desde aceleradores vectoriales de gama alta y el diseño AIE de Xilinx hasta dispositivos de bajo consumo como procesadores RISC-V y microcontroladores basados en ARM (Arduino).

Biografía

Enrique S. Quintana-Orti se licenció y doctoró en Informática por la Universidad Politécnica de Valencia (UPV), España, en 1992 y 1996, respectivamente. Tras más de 20 años en la Universidad Jaime I de Castellón, España, regresó a la UPV en 2019, donde actualmente es catedrático de Arquitectura de Computadores. Por su investigación, Enrique ha recibido el NVIDIA 2008 Professor Partnership Award, dos premios de la Agencia Espacial Nacional de Estados Unidos (NASA) y el premio a la trayectoria investigadora por la conferencia Euro-Par en 2023. Ha publicado más de 400 artículos en revistas y conferencias internacionales. Actualmente participa en los proyectos de la UE APROPOS (computación aproximada), RED-SEA (redes informáticas a exaescala), eFLOWS4HPC (flujos de trabajo para HPC e IA), Nimble AI (chip neuromórfico para detección y procesamiento) y Metamorpha (micromecanizado a medida con rayos láser). Sus intereses de investigación incluyen la programación paralela, el consumo de energía, la computación con precisión adaptativa, el aprendizaje profundo y el álgebra lineal, así como arquitecturas avanzadas y aceleradores de hardware.

El presente y futuro de la computación... y el papel de las FPGAs

Teresa Cervero García

Resumen

Las FPGAs son unos dispositivos con mucho potencial, pero que parece que no terminan de jugar un papel tan importante como se esperaba. ¿Mala gestión de las expectativas? ¿O un enfoque poco acertado? MEEP (MareNostrum Experimental Exascale Platform) quiere dibujar un nuevo futuro para la supercomputación. ¿Para qué? Con el objetivo de facilitar el trabajo duro a los desarrolladores hardware y software. ¿Cómo? Ofreciendo un Laboratorio Digital, compuesto por un cluster de FPGAs y un conjunto de herramientas para dar soporte: 1) al diseño y desarrollo de IPs, y 2) servir como SDV para apoyar el desarrollo del stack de SW mientras las arquitecturas finales no están disponibles.

Biografía

Teresa G. Cervero nació en Asturias, España, en 1982. Obtuvo los títulos de Máster en Ingeniería de Telecomunicaciones e Ingeniería de Telecomunicaciones Avanzada en 2007 y 2007, respectivamente, y el título de Doctora en 2013, todos de la Universidad de Las Palmas de Gran Canaria (ULPGC). También obtuvo el título de Doctora por la ULPGC en 2013, donde colaboró como miembro de la División de Diseño de Sistemas Integrados (DSI), Instituto de Microelectrónica Aplicada (IUMA) en ULPGC hasta 2015. Posteriormente se incorporó como Directora de Tecnología en Fredcode para formar parte en 2020 del equipo del proyecto MEEP (MareNosturm Exascale Emulation Platform) en el Barcelona Supercomputing Center,donde actualmente continua. Sus intereses de investigación incluyen diseños en tiempo real para aplicaciones multimedia, códecs de video H.264/AVC y SVC, sistemas de imágenes hiperespectrales, diseño basado en síntesis para SoCs y computación reconfigurable.

Parte 1.- XXXIII Jornadas de Paralelismo

Evaluación de Prestaciones

RISC-V for Genome Data Analysis: Opportunities and Challenges

Lorién López-Villellas[1], Esteve Pineda-Sánchez[2], Asaf Badouh[2], Santiago Marco-Sola[3], Pablo Ibáñez[1], Jesús Alastruey-Benedé[1] y Miquel Moretó[2,4]

Abstract— **The RISC-V ISA has gained significant momentum in High-Performance Computing (HPC) research and market due to its open-source nature, fostering collaborative research and innovation. The ever-growing RISC-V-based hardware/software ecosystem has made it an attractive option for HPC application development and production. Within the field of biomedical research, genome data analysis has emerged as a crucial step towards personalized medicine, demanding substantial computational resources and more efficient tools.**

This paper presents a benchmark suite of genome analysis kernels ported to RISC-V and their evaluation on modern RISC-V systems. Our work evaluates the RISC-V toolchain's maturity and the software/hardware ecosystem's readiness for its adoption for genome data analysis. This study aims to provide valuable guidance for researchers and practitioners interested in adopting RISC-V for genome analysis, and provides feedback to the RISC-V community on the challenges that need to be addressed for RISC-V to become an efficient HPC platform.

Keywords— **RISC-V, Genome Analysis, Benchmarking, HPC**

I. Introduction

Over the past years, the RISC-V instruction set architecture (ISA) has gained significant momentum and popularity, largely due to its open nature, which allows for free use, modification, and distribution. This characteristic has made it an affordable and accessible alternative to traditional and proprietary ISAs, like Intel's x86 and ARM. RISC-V's modular design allows for easy customization and scalability, making it adaptable to a wide range of applications and computing devices, from low-power devices to large-scale HPC supercomputers. The RISC-V project has garnered support from industry and academia, helping to drive its development forward. Developing RISC-V microprocessors has become a strategic priority for the European Union, as evidenced by many ongoing initiatives and research projects. To ensure competitiveness, it is crucial to consolidate a strong RISC-V community that can foster research and development of RISC-V technology for HPC-demanding applications, including climate modelling [1], molecular simulations [2], machine learning [3], and computational genomics [4].

[1]Dpto. de Informática e Ingeniería de Sistemas / Instituto de Investigación en Ingeniería de Aragón (I3A), Universidad de Zaragoza, e-mail: {lorien.lopez,imarin,jalastru}@unizar.es.

[2]Barcelona Supercomputing Center, e-mail: {esteve.pineda,asaf.badouh,miquel.moreto}@bsc.es.

[3]Universitat Autònoma de Barcelona, e-mail: santiagomsola@gmail.com.

[4]Universitat Politècnica de Catalunya.

With the advent of sequencing technologies, genome data analysis tools have gained special relevance in biomedical and healthcare research. Moreover, the wide accessibility to sequencing technologies has enabled the development of personalized medicine [5] for the early detection of diseases, such as hereditary cancers (e.g., Lynch syndrome and breast cancer [6]) and other genetic conditions [7]. As sequencing technologies evolve, the cost of sequencing a human genome drops exponentially, and the current sequencing data production is outpacing Moore's Law. This situation has resulted in an increasing computational bottleneck in the genome analysis pipelines [8]. To face this challenge, an increasing number of algorithms [4] and cutting-edge hardware accelerators [9] have been developed to accelerate the genome analysis pipelines.

This paper describes the porting of the GenArchBench benchmark suite to the RISC-V ISA. GenArchBench is a selection of 13 kernels belonging to widely used genomics tools that cover the most important steps of genome sequencing. We evaluate our port on two RISC-V systems to explore the maturity and suitability of RISC-V core designs for executing genomics applications. Our study aims to help other developers working on RISC-V porting and to guide the designers of RISC-V processors by pointing out the identified shortcomings.

This study makes the following contributions:

- We present a porting of the GenArchBench benchmark suite to the RISC-V architecture, exploiting the RISC-V vector extension when possible. The ported suite is publicly available at `https://github.com/LorienLV/_PAPER-JP23_RISC-V_for_Genome_Data_Analysis_Opportunities_and_Challenges`.
- We discuss the challenges and limitations encountered during the RISC-V porting, regarding the development toolchain, the availability of libraries and software for RISC-V, and the support level from the ISA, with particular emphasis on the RISC-V vector extension.
- We analyze the performance of GenArchBench on two different RISC-V platforms (Unmatched by SiFive and Allwinner by Alibaba) compared to an Intel Xeon Skylake-based server baseline for reference.

II. Target Hardware Systems

Our study and experimental evaluation involve two RISC-V platforms. Additionally, we include an

Tabla I: Overview of the machines of the experimental setup.

	Unmatched	**Allwinner**	**SKX**
CPU	Freedom U740	XuanTie C906	Skylake Platinum 8160
ISA	RV64GC+IMAC	RV64GCV	x86_64
VLEN	—	128 bits	128/256/512 bits
# Cores	4 + 1	1	24
Pipeline	in-order	in-order	out-of-order
Commit	2 instr.	1 instr.	4 μOP
Frequency	1.2 GHz	1 GHz	2.1 GHz
L1I	32 KB	32 KB	32 KB
L1D	32 KB	32 KB	32 KB
L2	—	—	1 MB
LLC	2 MB	—	33 MB
Memory	16 GB DDR4	2 GB DDR3	48 GB DDR4

x86_64 machine for comparison to provide baseline performance for a well-established and widely used architecture in HPC environments. The main characteristics of the systems are shown in Table I and are described throughout this section.

A. HiFive Unmatched

The HiFive **Unmatched** from SiFive is a RISC-V Linux development board released in May 2021. It is powered by the SiFive Freedom U740 (FU740) system-on-chip that operates at 1.2 GHz and features a dual-issue in-order 64-bit execution pipeline. It includes four SiFive U74 compute cores, each with a 32 KB instruction cache and a 32 KB data cache. These cores implement the RV64GC instruction set architecture (G and C extensions). The FU740 also includes one SiFive S71 auxiliary core for real-time applications. This core has a 16 KB instruction cache and an 8 KB data tightly integrated memory (DTIM) and implements the RV64IMAC ISA. The FU740 SoC includes a 2 MB coherent banked L2 cache and 16 GB of DDR4 memory operating at 1866 MT/s.

B. Allwinner D1

The **Allwinner** D1 (D1-H) is a system-on-a-chip (SoC) manufactured by Allwinner and released on April 2021. It is equipped with a single Alibaba T-Head XuanTie C906 core running at 1 GHz. This core implements the RISC-V base 64-bit instruction set architecture (ISA) and the G, C, and V extensions (RV64GCV). Additionally, it implements the RISC-V vector ISA v0.7.1, supporting a vector length of up to 128 bits. The core has an in-order single-issue execution pipeline, a 32 KB L1I, and a 32 KB L1D. The SoC is equipped with 2 GB of DDR3 memory.

C. Intel Xeon Skylake

As part of our experimental setup, we include an x86_64 compute node alongside the RISC-V systems previously described, which serves as a performance reference for production environments. This compute node, which we refer to as **SKX**, is powered by an Intel Xeon Skylake Platinum 8160 processor with 24 cores running at a peak clock speed of 2.1 GHz. Each core has a 32 KB L1I and a 32 KB L1D and is capable of retiring up to 4 micro-operations per cycle. The system also includes a 33 MB last-level cache (LLC). The x86_64 CPU supports SSE, AVX2, and AVX-512 single instruction on multiple data (SIMD) extensions, enabling vector operations on 128, 256, and 512 bits registers, respectively.

III. GenArchBench

The GenArchBench benchmark suite compiles a set of representative kernels belonging to widely-used tools and libraries for genome analysis. It includes ten kernels from the GenomicsBench test suite and three additional kernels: the Bit-Parallel Myers algorithm, the Wavefront Alignment algorithm, and a SIMD-accelerated version of Minimap2's chaining implementation (FAST-CHAIN). Table II shows the name, brief description, use-case, application of origin, and characterization of each kernel from GenArchBench.

The GenArchBench kernels cover a broad range of common steps of genome analysis pipelines. The starting point for many standard genome analysis pipelines is the process known as basecalling, i.e., processing the output from the sequencing machines. GenArchBench includes the ABEA kernel (from Nanopolish) and the NN-BASE kernel (from Bonito) as representative basecalling kernels. Once the sequenced DNA bases are known, most genome analyses require locating those sequences into a previously known reference genome by performing the read mapping step. Notable applications for read mapping are BWA-MEM2 or Minimap2. GenArchBench includes the BPM, BSW, WFA, CHAIN, FAST-CHAIN, and FMI kernels from this step. For some studies, there is no reference genome to read map against. In those situations, we need to perform a de-novo assembly of the genome before any read mapping can be performed. GenArchBench includes the DBG (Platypus) and KCNT (Flye) kernels, used within standardized and widely used assembly tools. When the sequenced DNA is located in the reference genome, we can detect variations between the sample and the reference genome. This process, known as variant calling, comprises GenArchBench's kernels NN-VARIANT (Clair3) and PILEUP (Medaka).

Additionally, the GenArchBench suite comprises a set of representative input datasets to execute the aforementioned kernels. It is important to note that genome analysis pipelines generally process large datasets, requiring large memory footprints. Alas, the RISC-V machines of our experimental setup are quite

Tabla II: Kernels included in GenArchBench.

Kernel	Description	Use-Case	Genomics Tool	Charact.
ABEA	Adaptive Banded Signal to Event Alignment	Basecalling	Nanopish	Compute-bound
BPM	Bit-Parallel Myers Alignment	Read mapping	GenArchBench	Compute-bound
BSW	Banded Smith-Waterman	Read mapping	BWA-MEM2	Compute-bound
CHAIN	Seed Chaining	Read mapping	Minimap2	Compute-bound
FAST-CHAIN	SIMD-enabled Seed Chaining	Read mapping	Minimap2	Compute-bound
DBG	De-Bruijn Graph construction	De-novo assembly	Platypus	Compute-bound
FMI	FM-Index	Read mapping	BWA-MEM2	Memory-bound
KCNT	K-mer Counting	De-novo assembly	Flye	Memory-bound
NN-BASE	Neural Network-based Base Calling	Basecalling	Bonito	Compute-bound
NN-VARIANT	Neural Network-based Variant Calling	Variant calling	Clair3	Compute-bound
PILEUP	Pileup Counting	Variant calling	Medaka	Compute-bound
POA	Partial-Order Alignment	De-novo assembly	SPOA	Compute-bound
WFA	Wavefront Alignment Algorithm	Read mapping	AnchorWave	Compute-bound

modest in memory capacity. Thus, we used GenArchBench's reduced input datasets (small inputs) to enable agile development and reasonable execution tests. Furthermore, we generated tiny inputs for the most memory-intensive kernels (FMI, KCNT, and POA) to enable their porting and execution.

IV. Porting GenArchBench to RISC-V

This section presents the outcomes of porting the genomics benchmark suite GenArchBench to RISC-V. We describe the set of kernels successfully ported for RISC-V architectures, the methodology and tools employed, and a primer for accelerating these kernels using RISC-V vector extensions (RVV). Furthermore, we discuss the challenges and limitations encountered during the porting, including the software and toolchain support for RISC-V, the RISC-V ISA suitability for genomics kernels, and other system and hardware caveats.

A. RISC-V GenArchBench Porting

From the initial 13 genomic kernels included in the GenArchBench, we have successfully ported and executed 10 of them. Unfortunately, the lack of RISC-V support from the toolchain, libraries, or hardware has prevented the porting of some kernels. In particular, the lack of RISC-V support from the widely-used Pytorch and Tensorflow libraries prevented the porting of the DL-based kernels (NN-Base and NN-Variant). As will be discussed later, the FAST-CHAIN kernel has been successfully ported to RISC-V by emulating certain vector instructions. However, we have been unsuccessful in executing the vectorized version of the kernel on Allwinner, the only RISC-V machine with support for the vector extension. FAST-CHAIN is a variant of the CHAIN kernel that has been modified to eliminate all heuristics in order to enable vectorization. Accordingly, assessing its scalar performance is not relevant or informative. Thus, Section V does not include data on FAST-CHAIN.

For the porting, we followed a straightforward development approach based on porting, verifying, and optimizing iteratively. We compiled ground-truth results from the input datasets to ensure the ported kernels behave as expected. Moreover, we used an automatic validation system to test all the ported kernels during development.

As a first step, we focused on the scalar adaptation of the kernels with particular emphasis on validating the results. Then, we proceeded to perform executions on scalar cores and gather profiling information. Afterwards, we worked on extending the porting to vector instructions (RVV) to accelerate the kernels' execution on RVV-enabled processors. Compared to the scalar adaptation, the RVV extensions have little software and hardware support, limiting the development to simulated environments using experimental tools.

B. Software and Libraries Support

Due to its recent introduction, RISC-V-based platforms still lack widespread application and library support. As a result, most applications and libraries require recompilation and careful ad-hoc optimizations to exploit the capabilities of current RISC-V processors. In some cases, these libraries lack specific RISC-V support, tampering the efficient porting of high-performance applications. Notable examples in genome sequence data analysis include libraries for common file-format management (like HDF5 and HTSlib libraries) and Deep Learning (DL) frameworks (like Pytorch and TensorFlow).

Nowadays, Deep Learning (DL) libraries and tools have gained widespread adoption in multiple research areas and industrial applications. In particular, DP-based applications for genome analysis rely on the popular Pytorch and TensorFlow libraries. These DL frameworks depict many libraries' dependencies and lack precompiled versions for RISC-V. Moreover, due to the computationally intensive nature of these libraries, straightforward and non-optimized porting to RISC-V could result in significant execution inefficiencies. For that, many efforts from the HPC community aim to enable high-performance execution of DL-libraries on RISC-V.

C. Development Toolchain Support

To address the challenges of porting and optimising widely-used applications and libraries, it is paramount to have mature and robust toolchain support. We have utilised several development tools for the genomics porting to RISC-V; some are stable and upstream available, others experimental and under active development. For clarity, a summary of the tools' support for base and vector RISC-V extensions is shown in Table III.

Tabla III: Summary of tools' support for base and vector RISC-V extensions.

Tool	Base (RV64GC)	Vector (RVV v0.7)
GCC	Yes	No
GDB	Yes	No
Clang	Yes	Partial
llvm-mc	Yes	Partial
Paraver	Yes	Yes
Perf	Partial	No
QEMU VM	Yes	Yes
Valgrind	No	No
Vehave	—	Yes

System infrastructure. For the porting development and evaluation, we performed multiple executions on commercial RISC-V scalar cores running standard Linux distributions; i.e., Ubuntu 20.04 (Unmatched), Fedora 33 (Allwinner), SUSE Linux Enterprise Server 12 SP2 (SKX). In addition, we utilised a QEMU virtual machine to emulate a generic 64-bit RISC-V processor for development, profiling, and RVV emulation purposes. Note that the selected QEMU virtual machine does not natively support RVV extension. We utilised the Vehave emulator for vectorial kernels, which enabled us to debug and functionally validate the ported kernels and perform performance profiling and analysis on an emulated environment.

Compilers. We mainly used GCC v10.3.0 and LLVM-based v12.0.0 compilers to compile libraries and tools. Both compilers used in this study have shown no difficulties to generate scalar binaries. However, RVV and auto-vectorization support is still under development for mainstream compilers. As a result, compiling vector code has been challenging and error-prone in many situations. For that reason, we resort to an extended Clang compiler (BSC-Clang) supporting RVV extensions via auto-vectorization, pragma annotations, and C-Style intrinsics from the EPI project.

Debuggers and profilers. Currently, commonly used tools for profiling and debugging (Valgrind, GNU Debugger (GDB), and Perf) offer limited support for RISC-V. For instance, the RISC-V support disassembling instructions (using RISC-V mnemonics) is a crucial feature when developing applications for RISC-V platforms. Therefore, fully-compliant RISC-V disassembling in GDB debugger is paramount, including instructions on the extended ISA. In that regard, we observe that the llvm-mc tool (Clang toolchain) is more mature, being able to convert opcodes to mnemonics and vice versa. Moreover, we noticed that the widely used Valgrind does not support RISC-V. Due to its importance and usefulness, there is an ongoing effort to port to RV64GC (`https://github.com/petrpavlu/valgrind-riscv64`).

Regarding profilers, like perf tools, we found current RISC-V cores lack numerous standard counters mapped on other architectures. In the case of the RISC-V cores studied in this work, we were only able to measure cycles and instructions in Unmatched. The Allwinner does not have counters.

To provide a better understanding of the kernel's performance and guide the development and optimization on RISC-V platforms, profiling utilities like Paraver have been promptly upgraded. Paraver can be used in conjunction with software emulators like Vehave to produce performance traces and analyse them using a versatile GUI. This tool allows analyzing performance metrics (like instructions and register usage), studying the memory access pattern, displaying annotated timelines with events, and more. These tools allowed us to inspect the critical regions and identify potential performance overheads. Moreover, we were able to investigate vectorial kernels without the need for a real vector machine.

D. ISA support

Ratified elemental extensions of the RISC-V architecture, such as the multiplication/division (M), floating-point (F), and double-precision (D) extensions, have been available for more than a decade and are generally adopted by commercial hardware and mainstream compilers. Most kernels ported on this project rely solely on these basic extensions, allowing its straightforward compilation and execution on any compliant RISC-V core. However, some RISC-V cores lack support for specific instructions from the specification. For example, the Allwinner machine does not support the optional `fence.tso` instruction used by the POA kernel.

Porting the genomic kernels to the RISC-V vector (RVV) extension has proven challenging in many cases. We found that only one kernel could be partially vectorized automatically by the compiler (i.e., WFA). For the kernels BSW and CHAIN, we develop a vectorial implementation based on RVV-intrinsics. For the remaining kernels, we found that the current RVV extension lacks essential instructions used by genomic kernels.

In particular, we found that, unlike AVX-512 (x86_64) or SVE (Arm) ISAs, the well-known count leading zeros (`clz`) and count trailing zeros (`ctz`) instructions lack a vector counterpart in the RVV specification, useful for the CHAIN and WFA kernels. The scalar versions of `clz` and `ctz` are part of the ratified bit manipulation extension (B), and their vector versions are currently part of the cryptography extension proposal. Beyond genomic kernels, `clz` and `ctz` are useful for multiple applications, including square root computation, Huffman decoding, binary logarithm computation, and hash tables indexing. Although their functionality can be emulated by executing other arithmetic instructions, the overall instruction overhead has a non-negligible toll on performance.

Similarly, the RVV specification lacks specific support to reverse the order of the elements within a vector register, a helpful operation for some of the genomic kernels. Vector reverse can be conveniently implemented using the `vid` instruction to compute an index vector and `vrgather` instruction to generate the final result. However, some additional instructions are needed to rearrange the indices produced

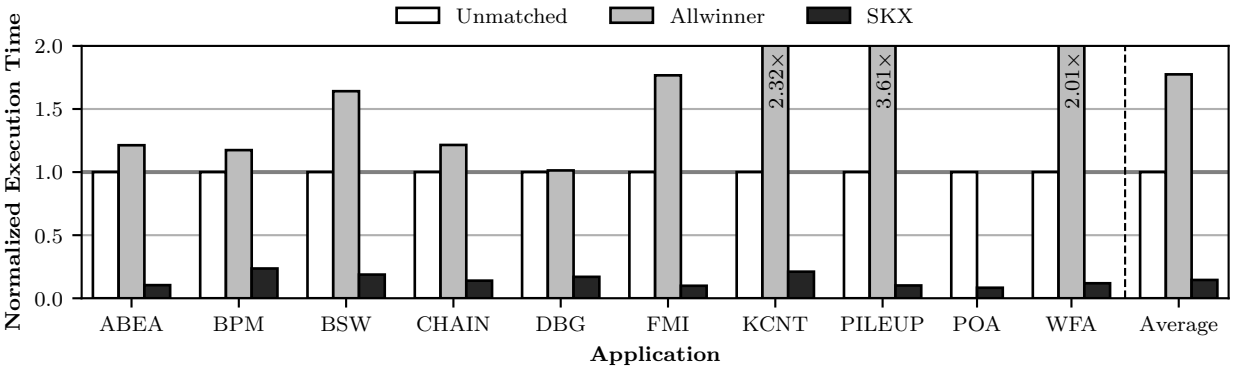


Fig. 1: Single-thread and scalar execution time of GenArchBench's kernels. The results are normalized to the performance on the Unmatched machine.

by `vid`. We claim a more powerful version of the `vid` instruction is possible. If the RVV allowed an instruction similar to `vid`, allowing to specify the starting, ending, and increment values (akin to the functionality of Linux's `seq` command), we could reverse a vector using fewer instructions. Additionally, this new instruction would be useful in a wide range of scenarios, since the current `vid` instruction, usually requires to be paired with additional instructions to compute the desired vector of indices.

In practice, we have encountered issues executing vector kernels on Allwinner, the only RVV-enabled core. Although this processor implements the RVV extension v0.7, it may not execute correctly in certain scenarios. For example, it throws an illegal instruction error when executing the `vmv` instruction, which is used on the kernels BSW, CHAIN, and WFA. Upon further investigation, we found that these kernels used 64-bit element width (SEW=64), and the Allwinner does not support RVV instructions for 64-bits element width, resulting in an illegal instruction exception. Although the kernels could be adapted to use a smaller element width, the compiler emits 64-bit element width by default. Another example is the execution of RVV gather instructions, required by some kernels like the WFA, which also result in a run-time error when executed on Allwinner.

It is important to acknowledge that the RVV specification and cores implementing it are still in the process of being refined and made more robust with future developments. A clear example of this is the specification of the masked instructions, which require using register v0 for the mask operand, although a mask can be stored in any register. Using RVV v0.7, moving a mask register to v0 required several instructions. However, with the introduction of RVV 1.0, this can be achieved with a single instruction. Although the scalar specification and cores prove to be more mature and stable, the RVV specification and hardware are rapidly evolving to be competitive with well-establish vector extensions in the industry like AVX and SVE.

E. Platform Limitations

In the exploration of the maturity and limitations of RISC-V processors to execute genomic applications, we have encountered other technical challenges. Most notably, genomic kernels usually involve processing big amounts of data, requiring large memory footprints. Considering the memory available on current RISC-V platforms, we found that real executions at genome-scale are not feasible as of today. Specifically, Allwinner is with only 2 GB of memory. Considering that representative genomics datasets and databases are usually of the order of tens of gigabytes, we were forced to use reduced inputs for testing the execution of the ported kernels. In some cases, we even had to produce tiny datasets to benchmark the most memory-intensive kernels as described in Section III.

V. Experimental Evaluation

In this section, we evaluate the performance of the two RISC-V machines presented in Section II when executing GenArchBench. First, we present the execution time results for scalar and single-thread executions performed in the RISC-V and SKX machines. Next, we present a scalability analysis for scalar multi-threaded executions on the SiFive Unmatched machine. Due to the challenges and limitations discussed in Section IV, we have not included results for the RVV versions of the kernels.

A. Single-thread scalar execution

Figure 1 shows the execution time of GenArchBench's kernels when performing scalar and single-thread executions on the two RISC-V machines, Unmatched and Allwinner, and the reference x86_64 machine, SKX (see Table I). The results have been normalized to the performance on Unmatched. We selected the Unmatched for normalization reference as it is the only RISC-V machine that can successfully execute the scalar version of all the kernels.

We observe that, compared to the SKX, Unmatched shows slowdowns between 3.8× and 12×, for KCNT and POA, respectively. The average slowdown of Unmatched compared to SKX is 6.6×. This is an expected result considering the characteristics of both processors. In particular, SKX works at a higher frequency (1.75×), has a higher superscalar degree (2×), and implements an out-of-order pipeline while Unmatched implements an in-order core. In addition, SKX has significantly larger L2 and L3 caches

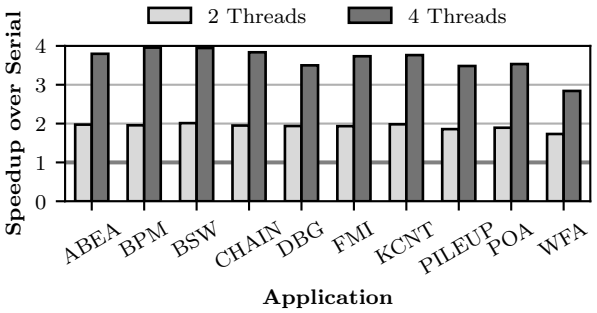


Fig. 2: Speedup over serial execution of GenArchBench's kernels on the Unmatched machine. We show the achieved speedup using 2 and 4 threads.

(16×) and includes hardware prefetchers in L1 and L2 while Unmatched does not implement any prefetcher. These features of their memory hierarchies can make a big difference in some applications.

Moreover, we noticed that Allwinner is consistently slower than Unmatched for all kernels, showing slowdowns ranging between 1.01× and 3.6×. These results are consistent with the characteristics of the machines, as Unmatched operates at a higher frequency (×1.2) and implements a superscalar core (×2). Furthermore, the memory hierarchy may significantly affect the performance of some kernels, considering that Allwinner has only one cache level, divided into L1I and L1D (32 KB each).

It would be noteworthy to perform a more detailed analysis to understand the influence of the memory hierarchy on the results. However, this analysis is not possible due to the lack of the necessary hardware counters in the RISC-V systems.

B. Multi-thread scalar execution

The Unmatched processor includes 4 cores, allowing multithreaded executions using up to 4 threads. Figure 2 shows the speedup of each kernel executing with 2 and 4 threads compared to single-thread execution. Most kernels exhibit near-perfect scalability as the number of threads increases. In particular, 9 of the 10 kernels achieve speedups above 1.85×, using two threads, and above 3.5×, using four threads. The WFA kernel presents the worst scaling behaviour, showing speedups of 1.7× and 2.8× for 2 and 4 threads, respectively.

VI. Related Work

The RISC-V architecture has gained significant traction in recent years. As a result, many efforts have been to extend its toolchain and improve RISC-V core design. Moreover, there are several ongoing European projects targeting the design of RISC-V processors. The European Processor Initiative (EPI) aims to design RISC-V processors for HPC. EPAC is the first chip designed as part of the EPI project, combining several domain-specific accelerators with a dual-issue core and a vector processing unit. The eProcessor project is aimed to deliver the first fully open-source European full-stack ecosystem based on a RISC-V CPU and multiple diverse accelerators. The DRAC project targets to design, verify, implement, and manufacture a RISC-V high-performance processor, incorporating different accelerators with applications to security, autonomous navigation, and genomics [10].

Regarding the RISC-V toolchain, we can find studies comparing different RISC-V compilers, such as the work by Poorhosseini et al. [11], which examines the performance and size of binaries generated by GCC and LLVM, and compares the performance of the compilers themselves. Other studies, such as the one by Adit and Sampson [12], have focused on compiler auto-vectorization. We observe there is an increasing interest in extending the RISC-V support to compilers, such as the JIT compiler [13].

Moreover, we find surveys on the RISC-V specifications [14] and many proposals to speed up critical kernels in the genomics field. For instance, Sargantana [15] is a RISC-V processor with support for a subset of the instructions part of the RISC-V vector extension. Additionally, it includes custom vector instructions to speed up the WFA kernel studied in this project. Furthermore, we find other accelerator proposals for genomics based on the RISC-V ISA [16], [17]. Similarly, we can find works leveraging the RVV extensions to speedup critical kernels in different domains, such as the post-quantum cryptography [18] or deep-learning [19]. Ramírez et al. [20] present a vector benchmark suite for RISC-V. Further, we find several studies benchmarking the performance of HPC applications across different RISC-V cores [21], [22] and different architectures [23].

VII. Conclusions

RISC-V's open nature and unique characteristics make it a highly promising ISA for future computing systems. Despite recent advancements, current RISC-V development lacks critical support from tools, applications, libraries, and hardware. In particular, compilers do not offer full support for some of the latest ratified extensions (including RVV), and development tools such as debuggers and profilers offer limited functionality on RISC-V platforms. Moreover, critical libraries (PyTorch and TensorFlow) lack an efficient port to RISC-V. Additionally, RISC-V machines do not provide reliable hardware support for some ISA extensions and offer limited support for hardware counters. Currently, RISC-V core designs are constrained in resources and not mature enough to compete with established architectures such as x86_64 or Arm.

The extensibility and openness of RISC-V present vast opportunities for developing new domain-specific extensions and accelerators. In this regard, leveraging vector instructions represents a significant leap towards achieving better performance and energy efficiency. Thus, future HPC RISC-V designs require support for the recently ratified vector extension. Notwithstanding the several challenges that need to be addressed, RISC-V has the potential to foster a powerful ecosystem for HPC computing, in

general, and genome data analysis, in particular.

Acknowledgment

We want to thank Roger Ferrer, Kilian Peiro, and Pablo Vizcaino for all of their useful help.

References

[1] Warren M Washington, Lawrence Buja, and Anthony Craig, "The computational future for climate and earth system models: on the path to petaflop and beyond," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1890, pp. 833–846, 2009.

[2] Gerard Martinez-Rosell, Toni Giorgino, Matt J Harvey, and Gianni de Fabritiis, "Drug discovery and molecular dynamics: methods, applications and perspective beyond the second timescale," *Current topics in medicinal chemistry*, vol. 17, no. 23, pp. 2617–2625, 2017.

[3] Marcia Sahaya Louis, Zahra Azad, Leila Delshadtehrani, Suyog Gupta, Pete Warden, Vijay Janapa Reddi, and Ajay Joshi, "Towards deep learning using tensorflow lite on risc-v," in *Third Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2019, vol. 1, p. 6.

[4] Mohammed Alser, Joel Lindegger, Can Firtina, Nour Almadhoun, Haiyu Mao, Gagandeep Singh, Juan Gomez-Luna, and Onur Mutlu, "From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures," *Computational and Structural Biotechnology Journal*, vol. 20, pp. 4579–4599, 2022.

[5] Helen K Brittain, Richard Scott, and Ellen Thomas, "The rise of the genome and personalised medicine," *Clinical Medicine*, vol. 17, no. 6, pp. 545, 2017.

[6] Ridgely Fisk Green, Mary Ari, Katherine Kolor, W David Dotson, Scott Bowen, Nancy Habarta, Juan L Rodriguez, Lisa C Richardson, and Muin J Khoury, "Evaluating the role of public health in implementation of genomics-related recommendations: a case study of hereditary cancers using the cdc science impact framework," *Genetics in Medicine*, vol. 21, no. 1, pp. 28–37, 2019.

[7] Dominique P Germain, Sergey Moiseev, Fernando Suárez-Obando, Faisal Al Ismaili, Huda Al Khawaja, Gheona Altarescu, Fellype C Barreto, Farid Haddoum, Fatemeh Hadipour, Irina Maksimova, et al., "The benefits and challenges of family genetic testing in rare genetic diseases—lessons from fabry disease," *Molecular Genetics & Genomic Medicine*, vol. 9, no. 5, pp. e1666, 2021.

[8] Gaye Lightbody, Valeriia Haberland, Fiona Browne, Laura Taggart, Huiru Zheng, Eileen Parkes, and Jaine K Blayney, "Review of applications of high-throughput sequencing in personalized medicine: barriers and facilitators of future progress in research and clinical application," *Briefings in Bioinformatics*, vol. 20, no. 5, pp. 1795–1811, June 2019.

[9] Tony Robinson, Jim Harkin, and Priyank Shukla, "Hardware acceleration of genomics data analysis: challenges and opportunities," *Bioinformatics*, vol. 37, no. 13, pp. 1785–1795, 05 2021.

[10] Jaume Abella, Calvin Bulla, Guillem Cabo, Francisco J Cazorla, Adrián Cristal, Max Doblas, Roger Figueras, Alberto González, Carles Hernández, César Hernández, et al., "An academic risc-v silicon implementation based on open-source components," in *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*. IEEE, 2020, pp. 1–6.

[11] Mehrdad Poorhosseini, Wolfgang Nebel, and Kim Grüttner, "A compiler comparison in the RISC-v ecosystem," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*. aug 2020, IEEE.

[12] Neil Adit and Adrian Sampson, "Performance left on the table: An evaluation of compiler autovectorization for RISC-v," *IEEE Micro*, vol. 42, no. 5, pp. 41–48, sep 2022.

[13] Quentin Ducasse, Guillermo Polito, Pablo Tesone, Pascal Cotret, and Loïc Lagadec, "Porting a JIT compiler to RISC-v: Challenges and opportunities," in *Proceedings of the 19th International Conference on Managed Programming Languages and Runtimes*. Sept. 2022, ACM.

[14] Enfang Cui, Tianzheng Li, and Qian Wei, "Risc-v instruction set architecture extensions: A survey," *IEEE Access*, vol. 11, pp. 24696–24711, 2023.

[15] Víctor Soria-Pardos, Max Doblas, Guillem López–Paradís, Gerard Candón, Narcís Rodas, Xavier Carril, Pau Fontova–Musté, Neiel Leyva, Santiago Marco-Sola, and Miquel Moretó, "Sargantana: A 1 ghz+ in-order risc-v processor with simd vector extensions in 22nm fd-soi," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 254–261.

[16] Zhongpan Wu, Karim Hammad, Abel Beyene, Yunus Dawji, Ebrahim Ghafar-Zadeh, and Sebastian Magierowski, "An fpga implementation of a portable dna sequencing device based on risc-v," in *2022 20th IEEE Interregional NEWCAS Conference (NEWCAS)*, 2022, pp. 417–420.

[17] Lorenzo Di Tucci, Riyadh Baghdadi, Saman Amarasinghe, and Marco D. Santambrogio, "Salsa: A domain specific architecture for sequence alignment," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 147–150.

[18] S. Pircher, J. Geier, A. Zeh, and D. Mueller-Gritschneder, "Exploring the risc-v vector extension for the classic mceliece post-quantum cryptosystem," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 401–407.

[19] Marco Cococcioni, Federico Rossi, Emanuele Ruffaldi, and Sergio Saponara, "Vectorizing posit operations on RISC-v for faster deep neural networks: experiments and comparison with ARM SVE," *Neural Computing and Applications*, vol. 33, no. 16, pp. 10575–10585, Feb. 2021.

[20] Cristóbal Ramírez, César Alejandro Hernández, Oscar Palomar, Osman Unsal, Marco Antonio Ramírez, and Adrián Cristal, "A risc-v simulator and benchmark suite for designing and evaluating vector architectures," *ACM Trans. Archit. Code Optim.*, vol. 17, no. 4, nov 2020.

[21] Alexander Dörflinger, Mark Albers, Benedikt Kleinbeck, Yejun Guan, Harald Michalik, Raphael Klink, Christopher Blochwitz, Anouar Nechi, and Mladen Berekovic, "A comparative survey of open-source application-class risc-v processor implementations," in *Proceedings of the 18th ACM International Conference on Computing Frontiers*, New York, NY, USA, 2021, CF '21, p. 12–20, Association for Computing Machinery.

[22] Amin Sarihi, Michael A. Schoenfelder, and Abdel-Hameed A. Badawy, "Performance evaluation of an out-of-order risc-v cpu: A spec int 2017 study," in *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2022, pp. 418–419.

[23] Yu Liu, Kejiang Ye, and Cheng-Zhong Xu, "Performance evaluation of various risc processor systems: A case study on arm, mips and risc-v," in *Cloud Computing – CLOUD 2021*, Kejiang Ye and Liang-Jie Zhang, Eds., Cham, 2022, pp. 61–74, Springer International Publishing.

Performance Analysis of a Non-Spatial Agent-Based Model Simulation

Cristina Peralta Quesada[1], Eduardo César Galobardes[1], Andreu Moreno Vendrell[1, 2], Anna Sikora[1]

Abstract— **Agent-Based Modeling and Simulations (ABMS) is a widely extended modeling approach aiming to emulate complex systems by focusing on the actions and interactions of agents within an environment. Complex ABMS applications include massive numbers of agents which behave and interact according to a given set of rules. Nowadays, several parallel ABMS frameworks are available in order to run complex ABMS applications in HPC systems. However, none of these frameworks include by default load balancing mechanisms to handle the load imbalance derived from the dynamic nature of these simulations. Hence, complex ABMS may suffer from bad resource management due to poor communication schemes and/or workload imbalance between processing elements. To deal with these load balance problems, modern literature presents performance enhancement mechanisms for parallel HPC ABMS applications. Nonetheless, most of the solutions for non-spatial ABMS are not applied to any well-known framework.**

In this paper we introduce a Repast HPC implementation of an open-source non-spatial social media platform agent-based model. Repast HPC includes shared network features to facilitate the development of parallel non-spatial agent-based models. However, it does not include agent partitioning mechanisms to distribute agents across computational resources. Distribution is therefore in the hands of the developer. Since a poor distribution of agents would imply an excess of communications between processes, two naive network partitioning methods are developed to understand the effects of naive partitioning in a non-spatial simulation.

Keywords— **Agent Based Modeling and Simulations, non-spatial ABMS, performance analysis, High Performance Computing, Load-Balancing**

I. Introduction

AGENT-BASED Modelling and Simulation (ABMS) [1] is a modelling approach that enables the simulation of complex systems by focusing on modelling the behaviour and interaction of agents within a given environment. Agents are defined as autonomous entities that behave according to predefined rules and have the ability to interact with other agents and their environment. ABMS are classified as discrete event simulations. Thus, they start with a given set of conditions and are expected to run until a certain state is reached or a determined amount of simulation steps have been performed.

Nowadays, ABMS is a popular modelling approach in many fields. We find modern ABMS simulations covering various topics of interest, such as sociological studies on the spread of rumours and the development of echo chambers in social media[2], [3], traffic flow management [4], [5], distribution systems [6][7], economic models for understanding financial markets [8], medicine [9], etc.

The complexity of ABM applications has increased over the years as they evolved to better resemble real-world systems. As a result, the computational resources required for actual ABMS applications now exceed the capabilities of a single workstation. Since the parallel nature of independent agents execution makes these applications suitable for execution on parallel High-Performance Computing (HPC) systems, several parallel ABMS frameworks have been developed to take advantage of HPC platforms. With the increase in complexity that can be achieved with the parallelisation of ABMS, new problems arise. The increasing number of agents and their dynamic nature can lead to an imbalanced workload and communication overhead in complex applications if the agents are not carefully distributed among the processors. In addition, the distribution of agents and communication patterns can change dynamically over the course of the simulation, which should be also considered. None of these problems are solved by default in well-known parallel ABMS frameworks, which do not include dynamic load-balancing mechanisms. This issue is foreseen in literature, where several load-balancing mechanisms are presented, mainly for spatial ABMS applications. However, no suitable solution for dynamic load-balancing has yet been presented for parallel non-spatial ABMS applications developed with well-known ABMS frameworks.

For this reason, in this paper, we present an evaluation of a naive implementation of a non-spatial agent-based model using Repast HPC tools for network models. In doing so, we aim to understand the problems that a non-optimal distribution of agents can cause in parallel non-spatial agent-based models. This work is the first step towards the definition of a load-balancing solution that considers minimising inter-process communication for non-spatial ABMS applications using a well-known parallel ABMS framework.

This paper is organized as follows: Section II presents state-of-the-art examples of non-spatial ABMS applications. Section III presents well-known parallel ABMS frameworks, focusing on Repast HPC and its currently available network features. Section IV presents different load-balancing solutions used until now to improve the performance of parallel

[1]Dpto. de Arquitectura de Computadores y Sistemas Operativos, Universidad Autónoma de Barcelona, e-mail: cristina.peralta,eduardo.cesar,anna.sikora@uab.cat.

[2]Escola Universitària Salesiana de Sarrià (EUSS), 08017 Barcelona, Spain, e-mail: amoreno@euss.cat.

ABMS applications. Section V describes an implementation of the social media model described on [3] in Repast HPC. Section VI shows the performance evaluation of the implemented social media model in Repast HPC. Finally, conclusions are presented on Section VII.

II. Non-spatial Agent-Based Models

Agents in agent-based models are usually represented as entities acting within a spatially explicit field with a defined number of dimensions that have continuous or discrete coordinates. In this case, the relationships between agents are mostly related to the proximity of the agents to other agents in the field. Therefore, spatial proximity is a crucial factor to determine communications between agents. In contrast, non-spatial models avoid the spatial component in simulations by representing agents in a network, where the connections between them are defined by their relationships to each other. Examples of non-spatial agent-based models can be found in many fields [10] in which models are represented as complex or large-scale networks, e.g. social media, social networks, supply chain management, economics, medicine, etc.

As stated in Section I, we are working towards the definition of a performance enhancement mechanism for non-spatial ABMS. Here, we present a brief overview of modern non-spatial agent-based models.

Models of social media platforms are defined as complex networks in which agents represent platform users and are seen as nodes in a network, while connections between agents are usually given by the follower/followee relationships of the users. Nowadays, social media can be modeled as a random network, where nodes are attached randomly to any other existing node, or a Barabási-Albert network[11], where new nodes are more prone to be attached to nodes with a higher number of connections. [2] presents an ABMS application aiming to study the impact of newsfeed curation algorithms on the propagation of information which may lead to echo chamber formation and polarization in social media. It models its social media network as a Barabási-Albert network where at every simulation step, agents may see a number of tweets from the agents they follow based on the curation algorithm applied. In this paper, the authors state that they use a population size of one thousand agents. However, they claim that this population size is not realistic and that their number of agents must be limited due to computational limitations. Therefore, a performance enhancement solution should benefit these kinds of simulations so that they can run with more realistic population sizes. [3], [12] and [13] show other examples of agent-based social media modelling.

Further examples of non-spatial agent-based models of complex networks can be found in modern literature. [14] introduces an ABMS for a supply chain network to test its resilience when a particular supplier or company goes out of business. [15] presents a social network model to assess the influence of certain actors on childrens' health habits. [7] shows an agent-based model for simulating electrical distribution systems implemented in Repast HPC. Agents represent the components within the electrical distribution network and their connections are the links between them. [12] explores belief expansion and opinion dynamics on social networks using agent-based modeling and Bayesian beliefs.

So far, the models discussed in this section are exclusively non-spatial. However, there are alternative modelling approaches that include a mixture of spatial and non-spatial representations. Hence, agents lie within a spatially explicit network that preserves both the spatial properties and the network connections between individuals. [16] and [17] are some examples of these kind of models. Load imbalances derived from the distribution of the networks on these models may be also interesting to evaluate in the future in order to offer a more complete load-balancing solution for parallel ABMS.

III. Parallel Agent-Based Frameworks

Several ABMS frameworks have been developed to support parallel/distributed environments, providing enough computational power to emulate the behaviour of real complex systems. Modern literature presents comprehensive surveys on this topic to facilitate the decision of choosing a framework according to the needs of the developers. [18] and [19] provide a comprehensive review of various ABMS frameworks, including their ease of programming, scalability, programming language, suitability for HPC environments, etc. Further studies on this topic ([20], [19]) evaluate the performance of the currently most popular parallel ABMS frameworks such as FLAME [21], FLAME GPU [22], DMASON [23], EcoLab [24] and Repast HPC [25]. Both papers suggest Repast HPC as the best scaling well-known parallel ABMS framework (although [19] could only evaluate the performance of FLAME and Repast HPC). None of these frameworks include dynamic load balancing mechanisms to handle communication overhead or workload imbalance between processing elements, which can lead to performance degradation as simulation complexity increases. Solutions to this problem have already been presented in the literature (Section IV), but these either mainly target spatial models or have been proposed as new frameworks exclusively for non-spatial agent-based simulations.

Since Repast HPC has been identified as one of the best-performing parallel ABMS framework [20] [19], it is used in this paper to develop a non-spatial ABMS model. Although Repast HPC supports non-spatial agent-based simulations, it lacks documentation on network projection [19] and there are claims of poor performance when simulating complex networks [13]. Other frameworks do not offer advantages for non-spatial agent-based simulations either.

Repast HPC is an ABMS framework based on Repast Symphony [26], aimed to be used in large-

scale distributed HPC platforms. In Repast HPC, models are implemented in C++. Agents are defined as C++ classes whose current state is represented by the values of their class variables, while their behaviour is encoded in their methods. Repast HPC utilizes boost MPI [27] library to manage communication and synchronization between processes. Simulations are run by following a discrete-event scheduler that executes events in the order specified by the developer, with each process synchronised to execute the same simulation step (*tick*) at a given time. At each step, the agents are expected to perform their behaviour in the order specified by the scheduler.

Each Repast HPC simulation consists of a set of agents distributed across different MPI processes. In each process, agents are created and added to one or more contexts. Each context is meant to encapsulate the agents of the process. Agents are added to a context when they are born and removed from it when they move from one process to another or die. Zero or more projections are associated with each context. Projections serve to give agents a relational structure, placing them in a space where they can relate to other agents. For example, a network projection places agents as nodes in a network that can relate to other nodes through edges. Repast HPC offers three different projections: discrete grid, continuous space or network.

In discrete grid and continuous space projections, agents are given discrete and continuous coordinates respectively and placed in a grid terrain. In these projections, agents usually relate to other agents nearby within a certain distance. Repast HPC is responsible for managing synchronisation between processes. To do so, it provides functions to replicate the agents between the neighbouring processes, remove agents which are not needed anymore and keep updated non-local agents and projection information. These types of projections assume that a process needs updated replicas of the agents located at a certain distance from each of the boundaries of its local grid.

In network projections, agents added to the projection are set as nodes in a network (even if no edge is attached to them). Edges between agents represent the relationship between the agents and can hold properties set by the user. In this case, Repast HPC also provides the same functions for projection and agents/edges synchronisation between processes. However, the process of creating non-local agents and inter-process edges is more complex than when using continuous and discrete projections. It is up to the user to determine in which process the agents are placed, to retrieve non-local agents if needed to create an edge between a local and a non-local agent, and to create the edges themselves, apart from keeping the projection and agent information up to date. The allocation of the agents considering their communication scheme and the degree of the agents within a partition has a great impact on the performance of simulations, rendering naive agent partitions unsustainable for complex large-scale simulations. Due to these factors and the additional problem of dynamism in networks, network projection in Repast HPC has been seen as a hard-to-use tool in the past [19], [13].

IV. Load Balancing Solutions

This section presents an overview of the load-balancing solutions proposed for parallel ABMS in modern literature. Most of these solutions target non-spatial ABMS applications, but they are either not applied to well-known parallel ABMS frameworks or are targeted at specific types of agent-based models.

[28] presents a dynamic performance enhancement mechanism for spatially-explicit ABMS applications. An agent system representation is built by clustering agents based on a grid-based clustering algorithm. Empty terrain is ignored in order to minimise the clustering overhead. A weighted hypergraph is then created containing the communication workload and computing of each of the agent clusters. Then a hypergraph load-balancing algorithm is applied using Zoltan [29], which provides a better partition for the simulation. Finally, the agents are migrated to achieve the new proposed optimal partition, while preserving at maximum the current partition in an effort to minimize the migration overhead. We believe that this mechanism could also be applied to non-spatial ABMS as well, but a different approach should be used for agent clustering.

[30] presents a distributed ABMS platform for network simulations that provides synchronisation and load-balancing among other features. This paper acknowledges Repast HPC as one of the most extended parallel ABMS frameworks but does not mention its network projection features. Instead, they describe a *Dynamic Distributed Graph Structure*, which creates a distributed graph between processing elements similar to Repast HPC's Graph structure. They include the possibility to update non-local agents in processes to which they do not belong, and load-balancing functionality based on Zoltan that can be periodically called by the user.

[13] proposes a two-tier partitioning algorithm for large-scale social media simulations on Spark. A SIR[31] model for information diffusion in social networks is defined. Then, a label propagation algorithm is used to detect communities of highly connected agents within the network. A simplified graph of the original network is created, where each community represents a node of the graph. Finally, a graph-cutting algorithm is run to decide the optimal partition of the simulation aiming to provide load balancing while minimising inter-partition communications.

[32] presents a parallelisation of a Social Contact Network (SCN) simulation with Repast HPC. To obtain load balance between highly visited nodes on the network they divide nodes with higher degrees (*hubs*) in different locations and share them indepen-

dently among the simulation processors. This prevents agents from moving to other processors when traveling to a hub node. However, this method is only applicable due to the characteristics of the SCN simulation, where locations may be separated into smaller independent locations. Other kinds of models may not benefit from this approach.

Another performance enhancement mechanism is proposed in [5]. It targets simulations with little communication between agents. Authors claim that if the most important volume of communication is within the agents and their environment, rather than with other agents, it is possible to arbitrarily divide the agents between processing elements into groups of similar size. They call this approach "*agent parallel*" and implement it in Repast HPC. However, they propose a centralised solution that disregards the communication overhead, only ensuring workload balance.

V. Parallel Social Media Platform Model

This section presents the Repast HPC implementation of M_2 model from BigTweet [3], considering its version without countermeasures. M_2 is the second model presented in [3] for studying the spread of information on Twitter and how countermeasures affect the propagation of malicious rumours over the network. In this model, agents are nodes in an undirected Barabasi-Albert [11] network representing the users of the social media platform, and their follower/followee relationship is defined by edges between them. The problem of rumour spreading is treated as an epidemiological SIR model[31] in which agents assume different states depending on their current belief in a rumour:

- *Neutral* agents are susceptible to believing the rumour.
- *Infected* agents believe the rumour and actively propagate information about it.
- *Cured* agents were previously infected but no longer believe nor spread the rumour.
- *Vaccinated* agents fight the rumour by sharing information against it.

Changes between states are governed by three fixed probabilities, *prob_accept_deny*, *prob_deny* and $prob\_infect$.

At the beginning of the simulation, there is an initialisation phase in which agents are naively distributed among the processes, the network connections are created and agent states are initialised. As stated before, agents are nodes in a Barabasi-Albert, which was created previously to the simulation start with networkit [33] and passed to the simulation as an input file. Two naive methods for agent partitioning are defined, Block and Round-Robin. In Block partitioning, the agents for each process are set as an equal-sized block of consecutive nodes of the network. If there is a remainder, it will be handled by the last process. For Round-Robin partitioning, agents are distributed between processes one by one in a cyclic manner.

Once agents are set on their respective processes, connections between them must be established. Inter-process connections require the request of non-local agents by using Repast HPC methods to create replicas of the required agents in each process. Edges can then be placed between local and non-local agents. Since this is an undirected network, it is not necessary to request all the non-local agents and create their edges. In fact, duplicate edges could lead to inconsistencies in the network and are discouraged in Repast HPC. Instead, it is only necessary to request non-local agents and create their edges for half of the adjacency matrix of the graph. The remaining edges and non-local agents are created by Repast HPC when using its projection synchronisation method to synchronise the network between processes as the last step in network creation. All agents are initially set to *Neutral* state when created. A number of them, defined by the input variable *initially.infected* in the file *model.props*, are then set to the *Infected* state before the simulation begins.

After the initialisation phase, the simulation runs in alternating execution and synchronisation phases. In the execution phase, the agents perform their behaviours. First, *Neutral* agents execute their *infectBehaviour* method, drawing an infection probability for each of their *Infected* neighbours. If a drawn probability is lower than $prob\_infect$, the next state of the agent changes to *Infected*. This represents an agent believing in the rumour propagated by an infected neighbour. If the first condition is not met, another probability is drawn and if it is less than *prob_deny*, agent's next state changes to *Vaccinated*, representing the disbelief in the rumour that its neighbour is trying to spread. Second, *Neutral* and *Infected* agents execute their *denierBehaviour* method, searching for *Vaccinated* agents among their neighbours. If a *Vaccinated* neighbour is found, a probability is drawn and if less than *prob_accept_deny*, the agent's state changes. If the agent was previously *Neutral* it turns into *Vaccinated*, representing its belief in the *Vaccinated*'s neighbour information. If it was *Infected*, it turns into *Cured*, dropping its belief in the rumour and stopping spreading information on the topic.

After the execution step, a synchronisation step is performed to update the current state of the non-local agents among the processes. Data regarding the state of the agents is collected at every step of the simulation and each simulation step is supposed to represent an hour of time in the social media platform.

A. Model Validation

For model validation we ran the Repast HPC simulation for the values of $prob\_infect$, *prob_accept_deny* and *prob_deny* from 0.01 to 0.1 incrementing with a step of 0.01 and fixing Repast random seed from 0 to 9 with a step of 1 on each simulation executed. Then we evaluated the best fit for our model as the

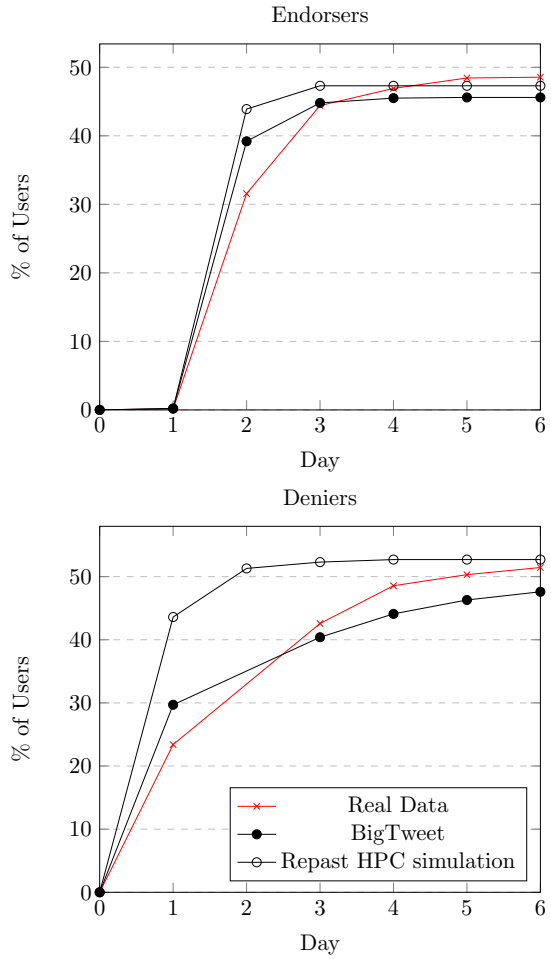


Fig. 1: Model Validation: Real data from Twitter15 Palin dataset, BigTweet simulation output, RepastHPC simulation output.

sum of the differences between the percentage of endorsers and deniers on each day of the simulation as done in [3]. Figure 1 presents the best fitting result of our simulation with *probabilit_accept_deny*=0.01, *probability_deny*=0.01, *probability_infect*=0.02 and *seed*=3 and the best output of BigTweet M_2, which happens to use the same probabilities. Real data shown comes from [34] dataset, available on [3] to be compared with Bigtweet's output. The behaviour of Repast HPC simulation similar to the baseline implementation in BigTweet's M_2 model.

VI. Performance Evaluation

This section presents the performance evaluation of the model described in V and implemented in Repast HPC. The simulations were executed on the cluster of research group HPCA4SE at Universitat Autònoma de Barcelona, using a node with an Intel Xeon Gold 6338 processor with 32 cores, 32K L1, 128K L2, 49162K L3 cache and 500GB of main memory.

A. Workload Distribution

Agent partitioning on non-spatial simulations is not a trivial problem. To achieve a network partition that ensures an even workload while minimising communication, agents must be grouped into tightly connected groups and distributed among processing elements to minimise inter-process communication. Some efforts have been done in the past to distribute agents within non-spatial agent-based simulations (Section IV). However, none of these apply to well-known parallel ABMS frameworks or do not contemplate dynamic networks. Therefore, as Repast HPC does not offer any partitioning methods for agents on non-spatial ABMS, we defined two naive partitionings to distribute the agents among processes (Section V).

This Section evaluates the distribution achieved by the two developed naive agent partitions: Block and Round-Robin. These partitions are not aimed to achieve load-balancing nor minimise communications but are rather used as a first approach to non-spatial ABMS simulation in Repast HPC. All experiments run with 1 million agents on an undirected Barabasi-Albert network, 32 MPI processes, 72 simulation steps, 80 initially infected agents, and probabilities *prob_accept_deny*=0.01, *prob_deny*=0.01, $prob\_infect$=0.2.

Figure 2 shows the distribution of local and non-local agents across processes with one million agents and 32 processes for Block (BLK) and Round-Robin (RR). Both partitioning methods ensure an equal share of local agents in each process, with up to 7.58 and 12.7 times more non-local than local agents in the process with the minimum number of replicas on BLK and RR respectively. Meaning that in the best case scenario, the number of replicas is high for both partitioning methods. It can be seen that the RR approach delivers homogeneous distribution of non-local agents among the processes, whereas with BLK the number of non-local agents decreases exponentially as the number of the process increases. This is due to the way in which this particular network is created and loaded into the simulation. In Barabasi-Albert networks, older nodes tend to have a higher number of neighbours. For this reason, the number of nodes with higher degrees is concentrated in the first entries of the network file, as each line of the file stores a node and its neighbours in the order in which they were created. By dividing them into consecutive blocks, most of the high-degree nodes end up on the first processes of the simulation, whereas when using RR, an equal proportion of these high-degree nodes are distributed among the processes. The high number of replicas in both partitioning methods may raise memory usage as the complexity of the simulation grow. Network partitioning could be improved, as it will be discussed in Section VI-C, to reduce the number of non-local agents.

Figure 3 shows the average time spent on a step for each process during the simulation. In the BLK partitioning, the average time spent on a step in the first process is between 38 and 58 times longer than for the remaining processes, displaying a higher workload imbalance than RR. This is again due to the high-degree agent population inside the first process. A higher degree of an agent in this case implies more

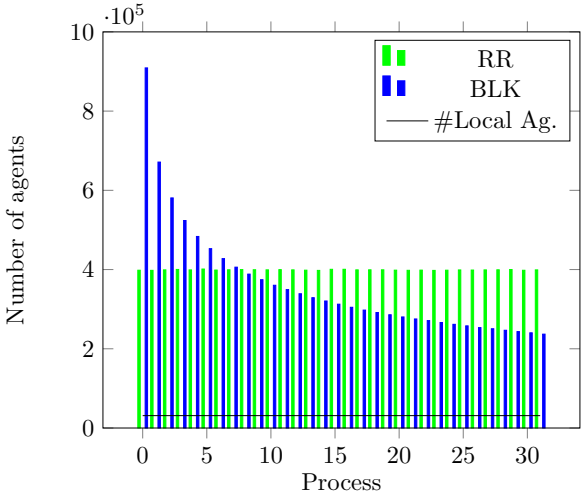


Fig. 2: Number of Local and Non-local agents in each process with a fixed number of 1M agents and 32 processes for BLK and RR partitioning.

workload, as each agent looks for their neighbours' state in order to change its own. On the other hand, RR shows an even distribution of the workload across the processes due to the more even distribution of high-degree nodes discernible in Fig.2.

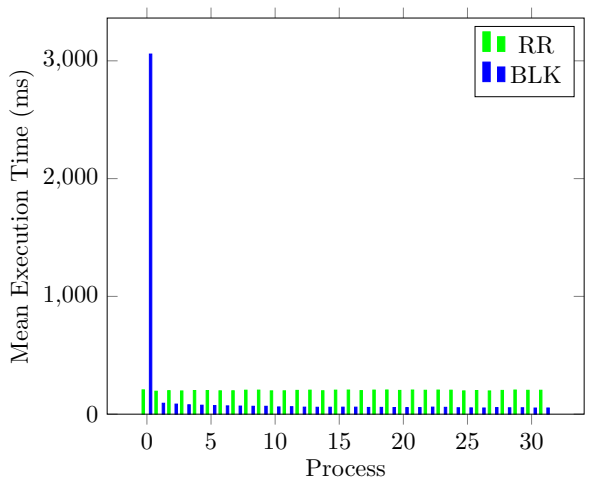
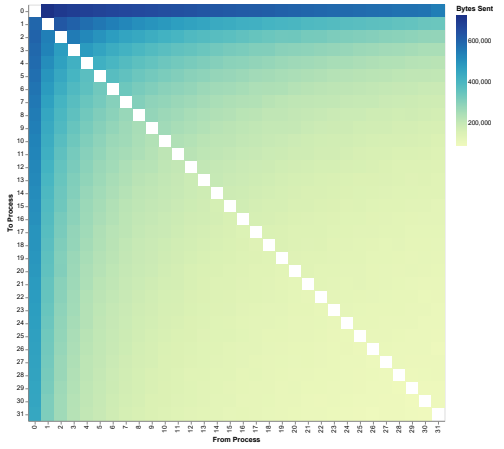


Fig. 3: Mean Execution Time in milliseconds per process for a 1M agent simulation with 32 processes including BLK and RR partitioning.

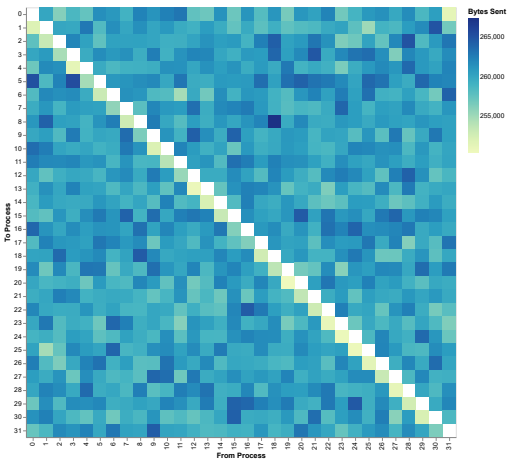
Figure 4 shows the total bytes sent between the processes for the two partitioning methods. In BLK (Fig. 4a), the amount of communications is higher for the lower processes and fades as the number of the process increases. It is important to note that the colour scale for both diagrams is different. The darkest shade on BLK distribution is set to about 800KB, while the darkest shade on RR is about 270KB. In RR (Fig. 4b), the amount of communication between all processes is more even. A further look at the communication between processes is made in Section VI-B.

B. Scalability Tests

This Section presents two scalability tests for the developed model in Repast HPC. First, a strong scalability test, where the performance of the simulation is evaluated when the number of processes



(a) Communication Matrix for BLK distribution



(b) Communication Matrix for RR distribution

Fig. 4: Communication Matrix for (a) BLK and (b) RR partitioning for 1M agent simulation with 32 processes.

varies from 2 to 32 while maintaining a fixed number of 1 million agents. Second, a test where the simulation runs with a fixed number of 32 processes, gradually increasing the number of agents from 200K to 1M. In both tests the network used is an undirected Barabasi-Albert network with a minimum of 10 connections per node. The remaining input parameters of the simulation are fixed to: 80 initially infected agents, 72 simulation steps -equivalent to 3 days of simulation-, $prob\_accept\_deny$=0.01, $prob\_deny$=0.01 and $prob\_infect$=0.02.

Figure 5 shows the results of the strong scalability test, including execution time and total bytes sent for each of the different configurations. For the BLK partitioning the execution time is generally higher than for RR but is closer as more processes are added to the simulation, reaching a speedup of x1,5 for RR against BLK an 32 processes. Regarding the communications on both partitions, in RR the communication volume is higher than in BLK for all configurations. However, this does not seem to hinder the performance of this partitioning method as we still obtain better execution times for RR in any case. There is no speedup achieved against sequential version until 32 processes are used, and then, it only reaches a x2,58 and x1,71 speedup for RR and BLK

versions respectively.

For the two processes version, most of the execution time is spent in agent requests, meaning that the most time expensive part of the simulation is requesting non-local agents in order to create the distributed simulation graph. Execution time for step function is the second most time expensive part on this version at half the time spent on agent requests. On the other hand, for 32 processors the most time expensive function is still agent requests but it is now 7 times faster than for 2 processes version. Execution time for step is also reduced to be less than one third of agents request time. We can conclude then that for small number of processors, agents requests overshade any gain obtained from sharing the workload. While for higher number of processes, the request time is less problematic but it is only possible to appreciate little improvement in performance due to the low computational demands of this model which makes graph initialization and synchronization a higher performance burden than computation.

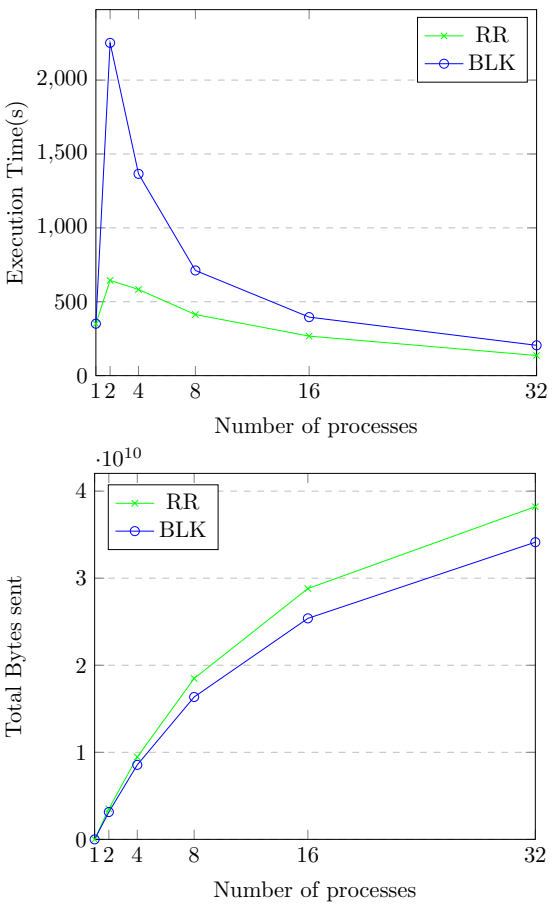


Fig. 5: Strong Scalability test including execution time and total bytes sent for social media model featuring 2 to 32 processes with Round Robin (RR) and Block (BLK) initialization.

Figure 6 shows the results of scaling the number of agents for a fixed number of processors, including execution time and total bytes sent for each of the different configurations. It presents the execution times of BLK and RR partitioning and a third version with the sequential execution time using one process with Repast HPC.

When the number of agents is quintupled, the RR, BLK and sequential versions in this experiment take 4.69, 5.45 and 6.75 more times to run the simulation respectively, with RR being the best scaling partition and also the fastest of the three versions. Communication volume per agent is kept the same for all versions. In this case, the best performance is obtained by RR partition achieving a x2,58 speedup per 32 processes against sequential version. The limitations on the achieved speedup are due to the fact that this model is not very computational demanding plus naive partitions offer no minimisation of non-local replicas, thus most of the execution time is spent on agents requests, graph initialization and synchronization rather than computation.

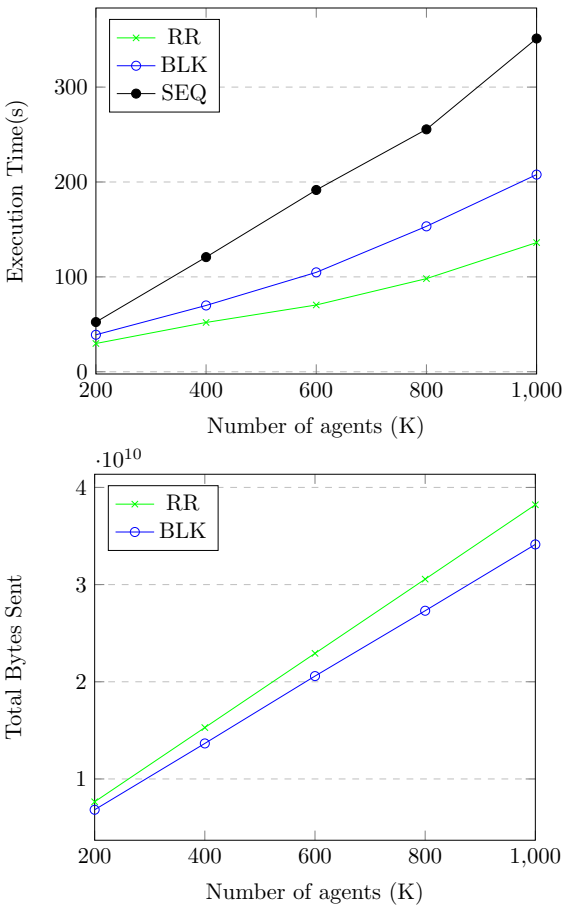


Fig. 6: Growing number of agents, including execution time and total bytes sent for social media model featuring 2K to 1M agents with Sequential time (SEQ - 1 process), Round Robin (RR) and Block (BLK) partitioning.

C. Discussion

As seen on Section VI-A, the number of non-local agents replicated among the shared network is high compared to the number of local agents for both proposed naive partitioning methods. Working towards obtaining an efficient partitioning for dynamic non-spatial agent-based models in parallel ABMS frameworks is the main goal of this project. To do so, a better agent partitioning method should be applied to aggregate highly connected nodes in a single process, helping minimise inter-process communications.

Community detection algorithms [35] such as label-propagation [36] are used in literature for de-

tecting communities of highly connected nodes in a network. Other clustering mechanisms [37] could be explored for non-spatial models with edge-attributed networks in order to find a broarder solution that covers as many network structures as possible.

Once clustered, agent communities or groups could be used to create a better partition of the network, as done in [28] for spatially-explicit simulations. This should help minimise the excess of agent replication among processes and thus lower the communication volume on the simulation while keeping the workload balanced. It is important to consider that when working with dynamic networks, the partitioning mechanism should imply a low overhead as it is periodically executed to keep the network balanced.

VII. Conclusions

In this paper, we have provided an overview of the state-of-the-art in non-spatial ABMS models, parallel ABMS frameworks and load-balancing solutions for non-spatial agent-based models. We have also developed a Repast HPC implementation for a simple social media platform model, including two naive partitioning methods and a brief performance evaluation of the model.

As mentioned in Section III, no modern well-known ABMS framework provide dynamic load-balancing for non-spatial agent-based models. Repast HPC is used in this work as it supports non-spatial ABMS through the use of network projections, plus has proven in the past to be one of the best scaling parallel ABMS frameworks.

To understand how a naive distribution of agents can affect the performance of non-spatial agent-based applications, two naive partitioning methods (BLK and RR) are implemented on a non-spatial agent-based social media platform model (Section V). A series of tests are performed in Section VI with BLK and RR partitionings. Section VI-A studies the workload distribution and communications across processors. Here, BLK has shown poor workload distribution and communication balance between processors. This is due to the way the network is created and loaded into the simulation, which makes the processes with the lowest ranks concentrate the agents with the highest degree. RR achieves better workload distribution across processors by sharing high-degree nodes more evenly across the network. However, both partitions result in a high number of non-local agents required to create the shared graph. This can have a negative impact on the memory usage and inter-process communications when the complexity of the simulation increases.

Section VI-B studies how the model execution time and communication volume evolves when adding more resources or agents. For this model, poor performance is obtained in any case as it is not very computationally demanding. Therefore, any improvement achieved through parallelisation is dimished by synchronisation costs and shared graph distribution. The time spent on synchronisation could be improved by developing a better agent partition method. This topic is further discussed in Section VI-C.

In terms of future work, we are currently working on implementing a more complex non-spatial agent-based model [2] in Repast HPC. We have chosen this model as the authors claim performance limitations that do not allow the model to run with a realistic agent population. Hence, it should benefit from parallelisation in Repast HPC while providing us with a large-scale, complex, non-spatial ABMS to work with. It would be interesting to add dynamism to the model's network to evaluate how this affects to the load balancing and communication within the simulation.

Acknowledgements

This work has been supported by the Ministerio de Ciencia e Innovación MCIN AEI/10.13039/501100011033 under contract PID2020-113614RB-C21 and by the Catalan government under contract 2021 SGR 00574.

References

[1] C. Macal and Michael North, "Agent-based modeling and simulation," *Proceedings of the 2009 Winter Simulation Conference (WSC)*, 12 2009.

[2] Anna Gausen, Wayne Luk, and Ce Guo, "Using agent-based modelling to evaluate the impact of algorithmic curation on social media," *J. Data and Information Quality*, jul 2022.

[3] Emilio Serrano and Carlos A. Iglesias, "Validating viral marketing strategies in twitter via agent-based social simulation," *Expert Systems with Applications*, vol. 50, pp. 140–150, 2016.

[4] Yadong Xu, Wentong Cai, Heiko Aydt, and Michael Lees, "Efficient graph-based dynamic load-balancing for parallel large-scale agent-based traffic simulation," in *Proceedings of the Winter Simulation Conference 2014*, 2014, pp. 3483–3494.

[5] Li Qiang, Xuefeng Guan, Rui Li, and Huayi Wu, "4dsas: A distributed dynamic-data driven simulation and analysis system for massive spatial agent-based modeling," *ISPRS International Journal of Geo-Information*, vol. 5, pp. 42, 03 2016.

[6] Emily Berglund, "Using agent-based modeling for water resources planning and management," *Journal of Water Resources Planning and Management*, vol. 141, pp. 04015025, 05 2015.

[7] Sonja Kolen, Stefan Dähling, Timo Isermann, and A. Monti, "Enabling the analysis of emergent behavior in future electrical distribution systems using agent-based modeling and simulation," *Complexity*, vol. 2018, pp. 1–16, 02 2018.

[8] Shu-Heng Chen, Chia-Ling Chang, and Ye-Rong Du, "Agent-based economic models and econometrics," *The Knowledge Engineering Review*, vol. 27, pp. 187 – 219, 2012.

[9] S. Rikard, Thomas Athey, Anders Nelson, Steven Christiansen, Jia-Jye Lee, Jeffrey Holmes, Shayn Peirce, and Jeffrey Saucerman, "Multiscale coupling of an agent-based model of tissue fibrosis and a logic-based model of intracellular signaling," *Frontiers in Physiology*, vol. 10, pp. 1481, 12 2019.

[10] Manuel Herrera, Marco Pérez-Hernández, Ajith Kumar Parlikad, and Joaquín Izquierdo, "Multi-agent systems and complex networks: Review and applications in systems engineering," *Processes*, 2020.

[11] Soon-Hyung Yook, Hawoong Jeong, and Albert-László Barabási, "Modeling the internet's large-scale topology," *Proceedings of the National Academy of Sciences*, vol. 99, no. 21, pp. 13382–13386, 2002.

[12] Daron Acemoglu and Asuman Ozdaglar, "Opinion dynamics and learning in social networks," *SSRN Electronic Journal*, vol. 1, pp. 3–49, 08 2010.

[13] Bin Chen, Hailiang Chen, Dandan Ning, Mengna Zhu, Roger ai, Xiaogang Qiu, and Weihui Dai, "A two-tier partition algorithm for the optimization of the large-scale simulation of information diffusion in social networks," *Symmetry*, vol. 12, pp. 843, 05 2020.

[14] Kang Zhao, Zhiya Zuo, and Jennifer Blackhurst, "Modelling supply chain adaptation for disruptions: An empirically grounded complex adaptive systems approach," *Journal of Operations Management*, vol. 65, pp. 190–212, 03 2019.

[15] Thabo Van Woudenberg, Bojan Simoski, Eric Fernandes de Mello Araújo, K.E. Bevelander, William Burk, Crystal Smit, Laura Buijs, Michel Klein, and Moniek Buijzen, "Identifying influence agents that promote physical activity through the simulation of social network interventions: Agent-based modeling study," *Journal of Medical Internet Research*, vol. 21, pp. e12914, 08 2019.

[16] Taylor M. Anderson and Suzana Dragićević, "Network-agent based model for simulating the dynamic spatial network structure of complex ecological systems," *Ecological Modelling*, vol. 389, pp. 19–32, 2018.

[17] M. Herrera, J. Izquierdo, R. Pérez-García, and I. Montalvo, "Multi-agent adaptive boosting on semi-supervised water supply clusters," *Advances in Engineering Software*, vol. 50, pp. 131–136, 2012, CIVIL-COMP.

[18] Sameera Abar, Georgios K. Theodoropoulos, Pierre Lemarinier, and Gregory M.P. O'Hare, "Agent based modelling and simulation tools: A review of the state-of-art software," *Computer Science Review*, vol. 24, pp. 13–33, 2017.

[19] Alban Rousset, Bénédicte Herrmann, Christophe Lang, and Laurent Philippe, "A survey on parallel and distributed multi-agent systems for high performance computing simulations," *Computer Science Review*, vol. 22, pp. 27–46, 2016.

[20] Andreu Moreno, Juanjo Rodriguez, Daniel Beltrán, Anna Sikora (Morajko), Josep Jorba Esteve, and Eduardo César, "Designing a benchmark for the performance evaluation of agent-based simulation applications on hpc," *The Journal of Supercomputing*, vol. 75, 03 2019.

[21] Simon Coakley, Marian Gheorghe, Mike Holcombe, Shawn Chin, David Worth, and Chris Greenough, "Exploitation of high performance computing in the flame agent-based simulation framework," in *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, 2012, pp. 538–545.

[22] Paul Richmond, Dawn Walker, Simon Coakley, and Daniela Romano, "High performance cellular level agent-based simulation with FLAME for the GPU," *Briefings in Bioinformatics*, vol. 11, no. 3, pp. 334–347, 02 2010.

[23] Gennaro Cordasco, Rosario De Chiara, Ada Mancuso, Dario Mazzeo, Vittorio Scarano, and Carmine Spagnuolo, "A framework for distributing agent-based simulations," 08 2011, vol. 7155, pp. 460–470.

[24] Russell Standish and Richard Leow, "Ecolab: Agent based modeling for c++ programmers," 02 2004.

[25] Nicholson Collier and Michael North, "Parallel agent-based simulation with repast for high performance computing," *SIMULATION*, vol. 89, no. 10, pp. 1215–1235, 2013.

[26] Michael North, Nicholson Collier, J. Ozik, Eric Tatara, C. Macal, M. Bragen, and Pamela Sydelko, "Complex adaptive systems modeling with repast simphony," *Complex Adaptive Systems Modeling*, vol. 1, 10 2013.

[27] Douglas Gregor and Matthias Troyer, "Boost. mpi," 2006.

[28] Claudio D. Márquez Pérez, *A grid-hypergraph load balancing approach for agent based applications in HPC systems*, Ph.D. thesis, Universitat Autònoma de Barcelona. Departament d'Arquitectura de Computadors i Sistemes Operatius., September 2017.

[29] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine, "The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring," *Scientific Programming*, vol. 20, no. 2, pp. 129–150, 2012.

[30] Paul Breugnot, Bénédicte Herrmann, Christophe Lang, and Laurent Philippe, "A synchronized and dynamic distributed graph structure to allow the native distribution of multi-agent system simulations," in *2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2021, pp. 54–61.

[31] Herbert W. Hethcote, "The mathematics of infectious diseases," *SIAM Review*, vol. 42, no. 4, pp. 599–653, 2000.

[32] Yulin Wu, Wentong Cai, Zengxiang Li, Wen Jun Tan, and Xiangting Hou, "Efficient parallel simulation over large-scale social contact networks," *ACM Transactions on Modeling and Computer Simulation*, vol. 29, pp. 1–25, 04 2019.

[33] Eugenio Angriman, Alexander van der Grinten, Michael Hamann, Henning Meyerhenke, and Manuel Penschuck, *Algorithms for Large-Scale Network Analysis and the NetworKit Toolkit*, pp. 3–20, Springer Nature Switzerland, Cham, 2022.

[34] Vahed Qazvinian, Emily Rosengren, Dragomir R. Radev, and Qiaozhu Mei, "Rumor has it: Identifying misinformation in microblogs," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, Edinburgh, Scotland, UK., July 2011, pp. 1589–1599, Association for Computational Linguistics.

[35] Santo Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010.

[36] Kishore Kothapalli, Sriram V. Pemmaraju, and Vivek Sardeshmukh, "On the analysis of a label propagation algorithm for community detection," in *Distributed Computing and Networking*, Davide Frey, Michel Raynal, Saswati Sarkar, Rudrapatna K. Shyamasundar, and Prasun Sinha, Eds., Berlin, Heidelberg, 2013, pp. 255–269, Springer Berlin Heidelberg.

[37] Petr Chunaev, "Community detection in node-attributed social networks: A survey," *Computer Science Review*, vol. 37, pp. 100286, 08 2020.

[38] Message P Forum, "Mpi: A message-passing interface standard," Tech. Rep., USA, 1994.

[39] "Tool Command Language.," `http://www.tcl.tk/`, [Online; Accessed 4 Dec 2022].

[40] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek, "Cuda, release: 10.2.89," 2020.

[41] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan, "Mason: A multiagent simulation environment," *SIMULATION*, vol. 81, no. 7, pp. 517–527, 2005.

[42] Sameer Shende and Allen Malony, "The tau parallel performance system.," *IJHPCA*, vol. 20, pp. 287–311, 01 2006.

Caracterización de prestaciones, interferencias y análisis de consumo energético en un procesador ARM Thunder X2

Ibai Calero[1], Salvador Petit[1], María E. Gómez[1] y Julio Sahuquillo[1]

Resumen— **En la actualidad, la eficiencia energética es de gran importancia en todo tipo de dispositivos, desde servidores hasta supercomputadores, pasando por ordenadores de escritorio. En este contexto, en los últimos años, los sistemas basados en la arquitectura *ARM*, tradicionalmente relegados a sistemas móviles y empotrados, han ganado una mayor cuota de mercado en nichos copados por procesadores basados en la arquitectura *x86*. Esta irrupción se debe a que los procesadores *ARM* de altas prestaciones ofrecen una mayor eficiencia energética que la competencia.**

El presente trabajo persigue caracterizar comportamientos típicos de las aplicaciones y analizar las correlaciones entre diversas métricas de prestaciones y consumo energético para servir de ayuda en el diseño de nuevas propuestas de planificación centradas en procesadores *ARM*. La principal conclusión del análisis de los resultados indica que las aplicaciones que sufren un menor *slowdown*, como consecuencia de la contención por los recursos compartidos del procesador, son también las que alcanzan una mayor eficiencia energética.

Palabras clave— **Caracterización, prestaciones, consumo energético, ARM, Marvell, ThunderX2, contadores de prestaciones**

I. Introducción

Las transiciones digital y energética son esenciales para lograr una economía sostenible y constituyen un pilar fundamental de la nueva estrategia industrial a nivel global. Ambas, además, están estrechamente relacionadas entre sí en varios aspectos, incluyendo una preocupante paradoja. Por un lado, se necesitan centros de proceso de datos (CPD) cada vez más potentes para apoyar estas transiciones y, por otro lado, el aumento vertiginoso de la cantidad de datos y de la demanda de servicios soportados por dichos sistemas, implican un riesgo de incremento desmesurado del consumo energético. Es por ello que necesitamos desarrollar CPDs que sean mucho más eficientes energéticamente que los existentes.

Uno de los ámbitos donde se pueden lograr mejoras en la eficiencia es a nivel de procesador, lo cual ha adquirido especial relevancia tanto en la industria como en el ámbito académico en los últimos años. Se ha pasado de considerar importante únicamente las prestaciones de un procesador a valorar también su eficiencia.

Tradicionalmente, la importancia de la eficiencia energética se ha centrado en dispositivos alimentados por batería, como los dispositivos móviles, dado que la mejora de la eficiencia energética en estos dispositivos es clave, al ser la duración de la batería una de las características principales a valorar por el consumidor. Sin embargo, en la actualidad, la eficiencia energética es de gran importancia en todo tipo de dispositivos, desde servidores hasta superordenadores, pasando por ordenadores de escritorio.

Entre las opciones para mejorar la eficiencia, por un lado encontramos un incremento del uso de procesadores basados en la arquitectura *ARM*. Estos procesadores han sido asociados tradicionalmente con un consumo y prestaciones más reducidos, limitándose principalmente a ámbitos como el de los dispositivos móviles. Por otro lado, los equipos personales y servidores han utilizado mayoritariamente procesadores basados en la arquitectura *x86*, los cuales suelen tener un mayor consumo y prestaciones, pero una menor eficiencia.

No obstante, esta situación está cambiando a medida que los procesadores *ARM* se vuelven más competitivos en términos de prestaciones. Un ejemplo de esta mejora en las prestaciones se puede ver en el ámbito de los superordenadores. Existen varias clasificaciones para estos, como *TOP500*, que evalúa las prestaciones en base a los PFlop/s de cada superordenador; *Green500*, que se basa en la eficiencia energética; y *HPCG*, que también evalúa las prestaciones, pero a diferencia del *TOP500*, el *benchmark* que utiliza simula una carga más realista, con, entre otras aspectos, un mayor número de accesos a memoria. En la actualidad, el superordenador más potente del mundo según la lista *HPCG* y segundo según *TOP500* es el *Fugaku*, el cual utiliza procesadores *A64FX* basados en *ARM*. Poniendo de manifiesto el potencial de *ARM*, no solo en el bajo consumo, sino también para sistemas en los que se requieren prestaciones elevadas.

El presente trabajo se centra en obtener métricas que puedan ser utilizadas en este tipo de procesadores para planificar las aplicaciones que se ejecutan en un sistema, ya sea maximizando las prestaciones, la eficiencia o logrando un equilibrio entre ambas. Para ello, se realiza un estudio y caracterización del comportamiento de diferentes aplicaciones en un procesador *ARM*. El principal objetivo del estudio es caracterizar comportamientos típicos de las cargas *SPEC CPU* y encontrar correlaciones entre métricas de prestaciones y consumo energético que sean de ayuda para el diseño de nuevas propuestas de planificación centradas en procesadores *ARM*. Los resultados muestran que la eficiencia energética en términos de prestaciones por vatio está altamente ligada a las

[1]DISCA, Universitat Politècnica de València, e-mails: icalqui@etsinf.upv.es, {spetit,megomez,jsahuqui}@disca.upv.es

prestaciones de los núcleos. Así, las aplicaciones que sufren una menor degradación de prestaciones debido a la contención por los recursos compartidos son también las que alcanzan una mayor eficiencia en el consumo energético.

II. Trabajo relacionado

La eficiencia energética de los sistemas computacionales genera cada vez más interés, tanto en la industria como en el ámbito académico. Es por ello que hay un gran número de publicaciones sobre este tema. A continuación se comentan algunas de estas publicaciones.

En [1], el objetivo principal es mejorar tanto las prestaciones como la eficiencia energética mediante la reducción de la contención en el acceso a los recursos compartidos. Para ello, se modifica el planificador de *Linux* para que considere el comportamiento de las aplicaciones y detecte cuál es el recurso crítico en cada momento mediante contadores *hardware*. Con esta información, se obtienen los denominados *activity vectors*, que representan la utilización de los recursos de cada aplicación y son usados por el planificador para elegir las aplicaciones a ejecutar y asignarlas a los núcleos del sistema, minimizando la contención.

En [2] se propone un planificador que trata de prevenir el sobrecalentamiento de núcleos individuales. Mediante la prevención del sobrecalentamiento, se evita la activación de los mecanismos de reducción de frecuencia y prestaciones (*thermal throttling*) implementados en los procesadores actuales. La propuesta se basa en el hecho de que un mayor consumo energético implica una mayor disipación de calor y, por tanto, una mayor temperatura. Teniendo esto en cuenta, se distribuyen las aplicaciones balanceando el consumo energético entre los núcleos, obteniendo un gradiente térmico más uniforme y evitando *hot spots*.

En [3], se propone una técnica para gestionar el consumo energético de forma global que utiliza, entre otros, mecanismos como *DVFS*. El objetivo de la propuesta es obtener las máximas prestaciones para un presupuesto energético determinado. En este trabajo se propone usar modelos basados en aprendizaje por refuerzo para abordar la alta complejidad que implica la gestión de energía en múltiples núcleos.

Enfocado a procesadores *ARM* heterogéneos, en [4] se plantea un planificador centrado en aplicaciones de navegación web que utiliza *DVFS* para regular prestaciones y energía. El planificador propuesto realiza predicciones sobre el tiempo de carga y consumo de energía de una página web considerando diferentes tipos de núcleos y niveles de frecuencia. El objetivo es determinar en qué núcleo se debe planificar la carga de la página web de manera que se cumplan restricciones temporales, mientras que se minimiza el consumo.

A diferencia de la mayoría de las propuestas recientes del estado del arte, centradas en arquitecturas heterogéneas para sistemas de bajo consumo, este trabajo se enfoca a sistemas de altas prestaciones basados en procesadores *ARM* con núcleos homogéneos. Este tipo de procesadores representan una alternativa energéticamente eficiente en el ámbito de la computación de altas prestaciones con respecto a arquitecturas más asentadas como las basadas en procesadores *Intel* o *AMD*. En este contexto, este trabajo representa un primer paso en el estudio de las métricas más adecuadas para caracterizar el comportamiento de las aplicaciones en tiempo de ejecución, con el objetivo de utilizarlas en trabajos subsiguientes para planificar la ejecución de las aplicaciones, mejorando las prestaciones, la eficiencia energética y la equidad en el sistema.

III. Metodología

En esta sección se discute la metodología utilizada realizar la caracterización de las aplicaciones, incluyendo el entorno experimental, las cargas, las métricas empleadas y los ajustes realizados sobre las mediciones del consumo energético.

A. Entorno experimental

El equipo donde se han realizado los experimentos utiliza el sistema operativo *CentOS Linux 7* y el núcleo de *Linux* 4.18.0. Dispone de un total de 64 GB de memoria *RAM* en 4 módulos *DIMM DDR4* 2666 MHz de 16 GB cada uno. El equipo cuenta con 2 procesadores *ThunderX2 CN9975* con 28 núcleos físicos cada uno trabajando a una frecuencia base de 1 GHz y una máxima de 2,5 GHz.

El procesador dispone de tres niveles de caché. El primer nivel es privado por núcleo y tiene una capacidad de 64 KB, dividido en 32 KB para datos y 32 KB para instrucciones. El segundo nivel también es privado y tiene una capacidad de 256 KB para datos e instrucciones. Por último, el tercer nivel es compartido por todos los núcleos del procesador y tiene una capacidad de 32 MB.

Para la realización de los experimentos se ha utilizado el *manager Stratus* [5], el cual facilita la planificación de las aplicaciones y la monitorización de contadores de prestaciones *hardware* por medio de la utilidad *perf* [6].

B. Cargas de trabajo

Aunque se persigue caracterizar todas las aplicaciones del *SPEC* [7] desde el punto de vista del consumo, para la realización del estudio se han seleccionado cuatro *benchmarks* de las *suites SPEC CPU 2006* y *SPEC CPU 2017*, los cuales, basándonos en nuestra experiencia, consideramos que cubren cuatro de los comportamientos más comunes de las aplicaciones del *SPEC* (*core bound*, *memory bound*, *L3 bound* y *extreme L3 bound*).

En concreto, los *benchmarks* utilizados en este estudio son: *calculix, cactuBSSN_r, mcf* y *xalancbmk_r* y sus comportamientos se detallan en la sección IV-A.

Para poder comparar de manera razonable los distintos *benchmarks* entre sí, se ha limitado el número

máximo de instrucciones que cada aplicación puede ejecutar a las que ejecuta una única instancia de esa aplicación durante un minuto a la máxima frecuencia soportada por el procesador (2,5 GHz). Una vez se ha ejecutado este número máximo de instrucciones para una aplicación dada, se miden sus prestaciones y consumo energético totales.

El *manager* ha sido configurado para reiniciar las aplicaciones a medida que van finalizando, hasta que la aplicación más lenta termina. De esta manera, se asegura que todas las aplicaciones sufren un nivel de contención en el sistema similar durante toda su ejecución, independientemente del orden en que finalizan.

C. Ajustes con respecto al consumo estático del procesador

Debido a que las mediciones de los consumos del procesador incluyen el consumo estático, es necesario determinar este consumo para identificar su impacto en el total de la energía global. El consumo estático o *leakage*, también conocido como corrientes de fuga, representa la energía consumida por el simple hecho de que los transistores se encuentren activos. Este consumo se mide cuando el procesador no ejecuta ninguna aplicación.

El porcentaje sobre el consumo global que representa el consumo estático puede llegar a ser relativamente alto, especialmente cuando se ejecuta un número reducido de aplicaciones. Para abordar este problema, se han utilizado las siguientes técnicas:

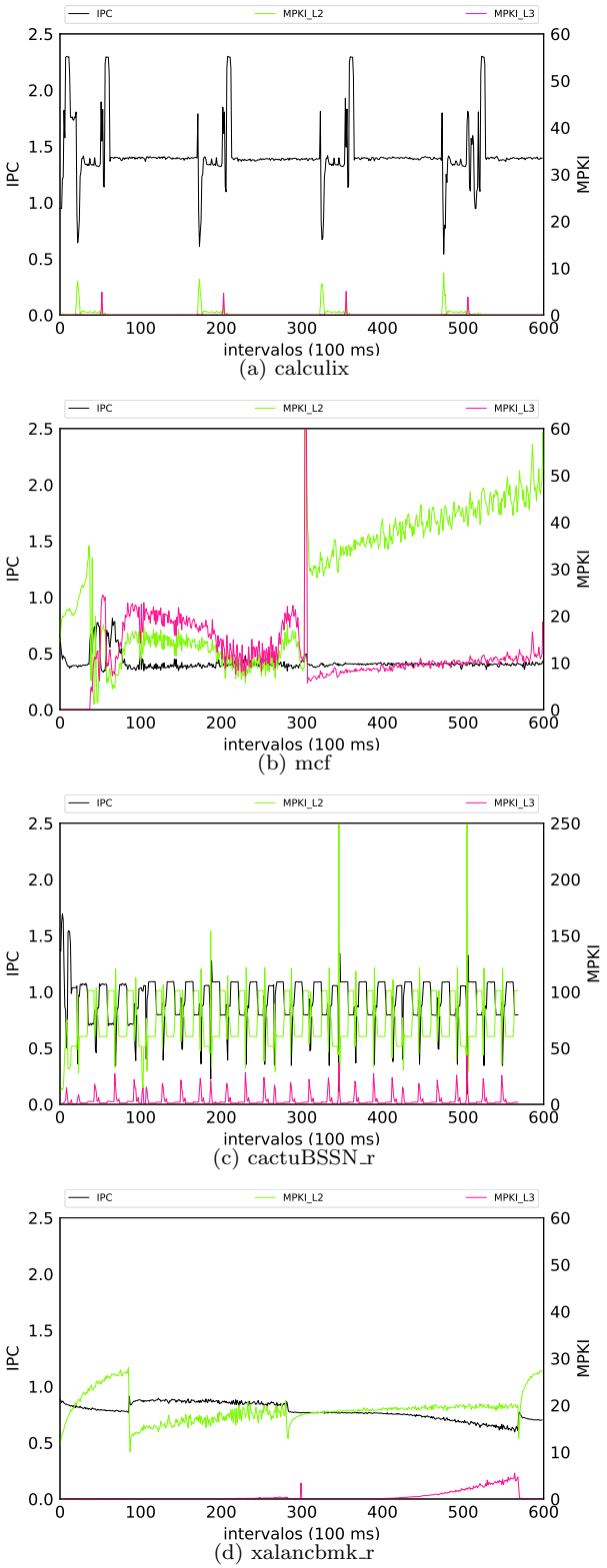
Reducción de la frecuencia. Se ha reducido la frecuencia de los núcleos del sistema que no se encuentran en uso a la mínima permitida (1 GHz), minimizando el consumo estático de estos.

Sustracción del consumo estático. Se ha medido el consumo energético estático, tanto de los núcleos como de la caché, cuando no se ejecuta ninguna carga. Este consumo se ha sustraído de los resultados de consumo energético obtenidos en los experimentos donde se ejecuta una sola aplicación (ejecución individual). De esta manera, se puede determinar con exactitud el consumo dinámico atribuible a la ejecución de una sola aplicación en ejecución individual.

IV. Evaluación de prestaciones y energía de aplicaciones individuales

Como se ha comentado anteriormente, el objetivo de estos experimentos es identificar una serie de métricas que puedan ser usadas para determinar la mejor manera de planificar las aplicaciones, en función de si se pretende maximizar sus prestaciones, la eficiencia energética o alcanzar un equilibrio entre ambos.

Para ello, se estudia y caracteriza, desde el punto de vista del consumo, el comportamiento de los *benchmarks* mencionados en la Sección III-B (*calculix*, *cactuBSSN_r*, *mcf* y *xalancbmk_r*), los cuales, como se ha comentado anteriormente, consideramos que cubren cuatro de los comportamientos más comunes de las aplicaciones de la *suite SPEC*. Estos *benchmarks* se ejecutan de forma individual, variando el número de instancias de cada uno y, por último, cuando se ejecutan en parejas, variando también el número de instancias.



(a) calculix
(b) mcf
(c) cactuBSSN_r
(d) xalancbmk_r

Fig. 1: MPKI e IPC dinámicos

A. Caracterización de aplicaciones: prestaciones

En primer lugar, se evalúan las prestaciones de los *benchmarks* seleccionados utilizando el *IPC* como

métrica. También se analiza la relación entre las prestaciones y la cantidad de accesos a memoria, así como entre las prestaciones y el consumo de los núcleos del procesador.

Para cuantificar los fallos de las distintas cachés, se utiliza la métrica $MPKI_{Li}$ (*cache misses per kilo instructions*) para la caché de nivel i. Esta métrica puede aplicarse a las cachés de todos los niveles y permite conocer, durante la ejecución de una aplicación, la cantidad de fallos de éstas por cada mil instrucciones. Resulta especialmente útil aplicarla a la caché L3, ya que los fallos de este nivel se traducen en accesos a memoria principal, los cuales penalizan significativamente a las prestaciones de las aplicaciones.

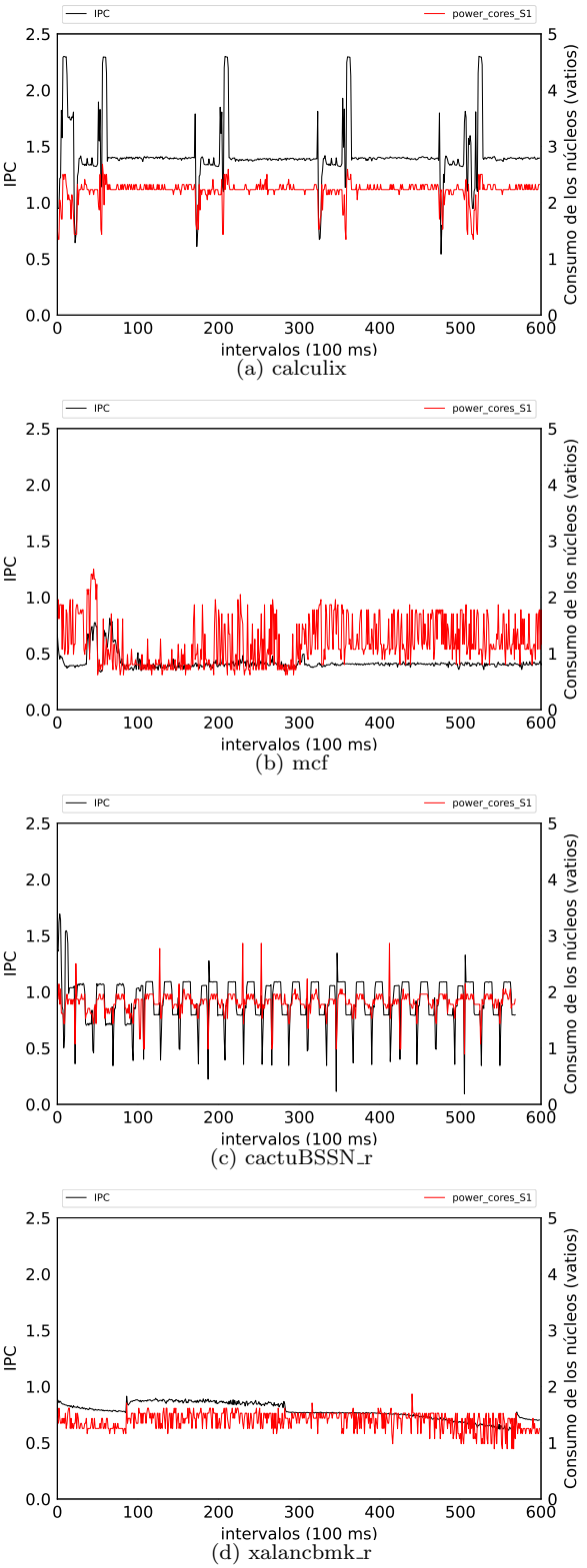
La Figura 1 muestra la evolución del *IPC*, $MPKI_{L2}$ y $MPKI_{L3}$ de cada una de las aplicaciones. Se observan cuatro comportamientos distintos:

CPU bound. Se puede observar cómo *calculix* (Figura 1a), con un *IPC* medio alrededor de 1,4, muestra un comportamiento limitado por la *CPU* (*CPU bound*) en casi toda su ejecución, con pocos accesos a memoria. El *MPKI* de ambas caches estudiadas es prácticamente cero a lo largo de la ejecución, y el de L3 nunca supera el 5 %.

Memory bound. En el extremo opuesto tenemos a la aplicación *mcf* (Figura 1b), que presenta un *IPC* casi constante de 0,4 y muestra un comportamiento limitado por la memoria (*memory bound*) a lo largo de su ejecución. Esto se aprecia especialmente en la segunda parte de su ejecución (desde el intervalo 300 al 600), donde el $MPKI_{L2}$ va creciendo de manera uniforme desde 30 hasta casi 60. El $MPKI_{L3}$ supera el valor 20 y es cercano al de $MPKI_{L2}$ en la primera parte de su ejecución, y en la segunda parte está alrededor de 10.

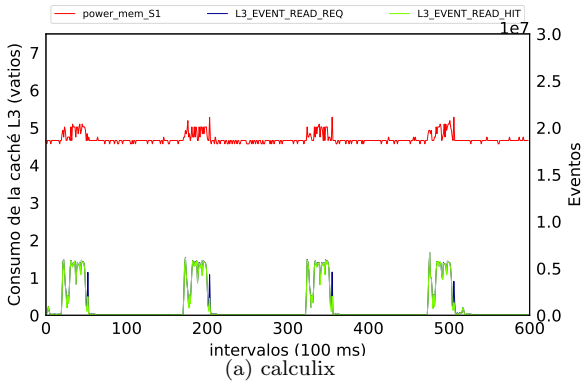
Extreme L3 bound. Este comportamiento es presentado por *cactuBSSN_r* (Figura 1c), que alterna de manera regular un número de accesos alto a la caché L3 con uno muy alto. El bajo supera, la mayoría de las veces, los 50 accesos cada mil instrucciones ejecutadas (5 %). En este caso, el *IPC* se encuentra alrededor de 1,1, pero en los valores muy altos de accesos a la caché L3 (superiores a 100), el *IPC* cae drásticamente a 0,4. Aunque la localidad de la L3 es, en general, alta ($MPKI_{L3} < 7$), esta presenta picos regulares que superan los 20. Esta elevada localidad contribuye a reducir el número de accesos a la memoria principal, los cuales tienen una gran latencia. La menor latencia de la L3 en comparación con la memoria, junto con la ejecución fuera de orden del procesador, disminuye el impacto de tantos accesos a la L3 en las prestaciones. Siendo los fallos de la caché L3, y no los de la L2, los que reducen drásticamente el *IPC*.

L3 bound. *Xalancbmk_r*, mostrada en la (Figura 1d), presenta un comportamiento con un número notable de accesos a la caché L3 (alrededor de 20 por cada mil instrucciones ejecutadas). Durante la primera mitad de su ejecución (desde el intervalo 0 al 300), la mayoría de estos resultan en aciertos ($MPKI_{L3}$ es cercano a 0). A partir del intervalo 450, el $MPKI_{L3}$ empieza a crecer. Este incremento en el número de accesos a memoria principal, con su elevada latencia, perjudica al *IPC* que se reduce de 0,85 a 0,7.

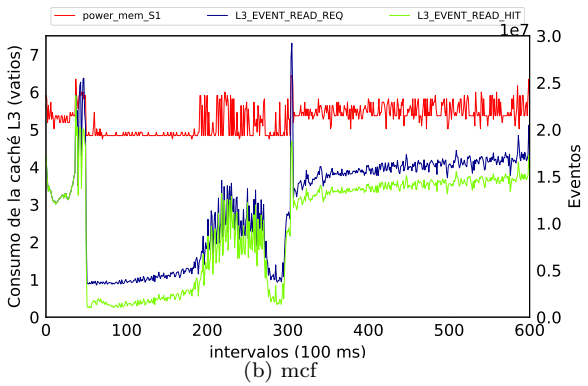


(a) calculix
(b) mcf
(c) cactuBSSN_r
(d) xalancbmk_r

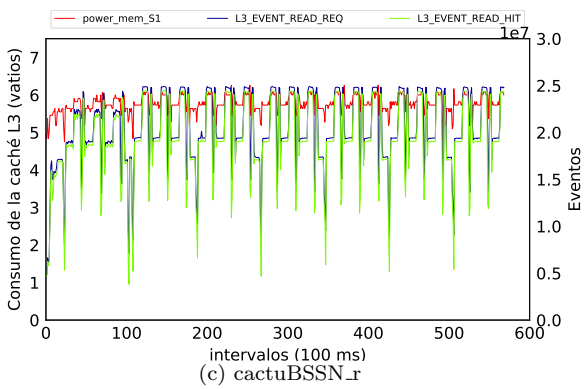
Fig. 2: IPC y consumo dinámico de los núcleos.



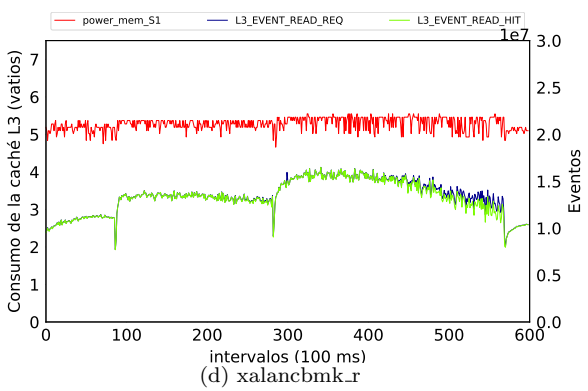
(a) calculix



(b) mcf

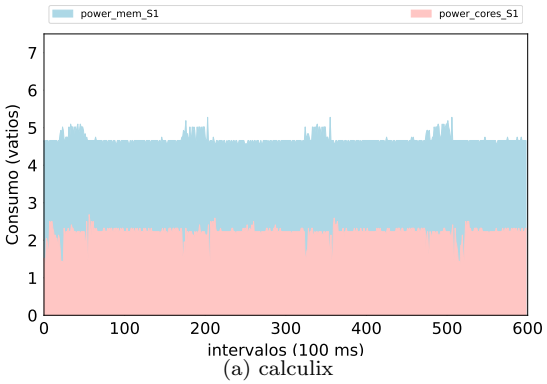


(c) cactuBSSN_r

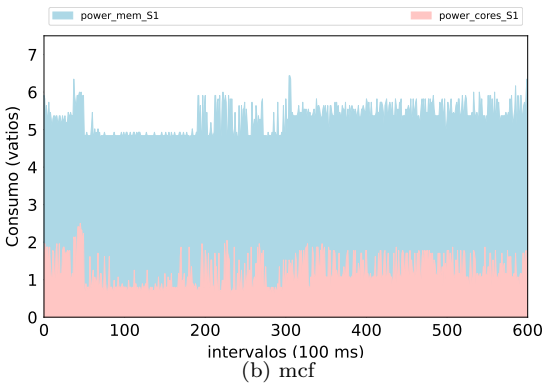


(d) xalancbmk_r

Fig. 3: Consumo y eventos de la caché L3 dinámicos.

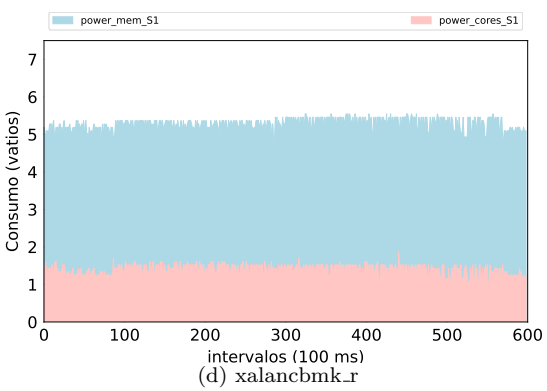


(a) calculix



(b) mcf

(c) cactuBSSN_r



(d) xalancbmk_r

Fig. 4: Consumo dinámico de los núcleos y L3.

B. *Caracterización aplicaciones: consumo vs prestaciones*

Otro aspecto importante es el efecto que las prestaciones de las aplicaciones tienen en el consumo energético del procesador. Las Figuras 2 y 3 muestran, respectivamente, a) la evolución del *IPC* junto al consumo de los núcleos del procesador (*power_cores_S1*, medido en vatios) y b) la evolución de los accesos y aciertos a la caché L3 (número de eventos *L3_EVENT_READ_REQ* y *L3_EVENT_READ_HIT*) junto a su consumo (*power_mem_S1*, también medido en vatios). En esta sección se analiza el comportamiento de estas métricas y su interrelación, centrándose, en primer lugar en el consumo de los núcleos y, en segundo lugar, en el consumo de la caché L3.

Nótese que se han utilizado las estrategias mencionadas en la Sección III-C para evitar que la interpre-

tación y análisis de los resultados de las aplicaciones en ejecución individual se vean afectados por el consumo estático global del procesador.

B.1 Consumo de los núcleos

Como era de esperar, el consumo de los núcleos muestra una relación directa con el *IPC*. Las aplicaciones con un mayor *IPC*, como *calculix* y *cactuBSSN_r* (Figuras 2a y 2c), presentan los mayores consumos (2,2 y 1,8 vatios respectivamente). Además, puede verse el efecto que las variaciones en el *IPC* tienen en el consumo, pues las caídas y picos del *IPC* se reflejan también en este. Por otra parte, en las Figuras 3a y 3c, se pueden observar los accesos y aciertos de estas mismas aplicaciones en la caché L3. Se aprecia que la evolución de los accesos y aciertos es inversamente proporcional a la del *IPC* y al consumo de los núcleos de ambas aplicaciones. Esto se debe a las latencias asociadas a estos accesos.

En cambio, *mcf* y *xalancbmk_r* (Figuras 2b y 2d) presentan un menor *IPC* y consumo (sus consumos medios son 1,2 y 1,4 vatios respectivamente). El consumo de estas aplicaciones es menos estable que el de *calculix* y *cactuBSSN_r*. Esto se debe a su patrón accesos a memoria. En las Figuras 3b y 3d, puede verse cómo los accesos a la caché L3 de *mcf* y *xalancbmk_r* presentan dientes de sierra, los cuales se trasladan al *IPC* y, por lo tanto, al consumo de los núcleos de las aplicaciones.

B.2 Consumo de la caché L3

El consumo dinámico de la caché viene determinado por la cantidad de accesos y aciertos. En cada acceso, se accede al *tag array* para comprobar si el bloque que contiene el dato solicitado se encuentra en la cache. En caso de fallo, se accede a la memoria principal y, en caso de acierto, se accede al *data array*, lo que incrementa el consumo.

Para facilitar el análisis del consumo de la caché y su relación con el consumo de los núcleos, la Figura 4 representa la evolución de ambos. Nótese que la figura presenta los datos de forma *no apilada*. Por ejemplo, durante la mayor parte de la ejecución de *calculix* (Figura 4a) el consumo de los núcleos se encuentra alrededor de los 2,3 vatios, mientras que el de la caché es de aproximadamente 4,7 vatios.

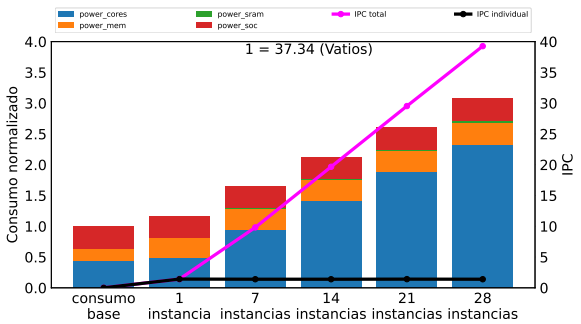
Calculix, al ser una aplicación *CPU bound*, presenta el menor consumo de la caché L3 de las 4 aplicaciones. Esto se debe a que realiza pocos accesos a la L3, como se muestra en la Figura 3a, donde se observan valores cercanos a 0 durante la mayor parte de su ejecución, excepto por algunos picos.

Sin embargo, *mcf* (Figura 4b) presenta un consumo mayor (sobre los 5,3 vatios). Esto es esperable, ya que *mcf* es una aplicación *memory bound*. Esto también se refleja en la cantidad de accesos y aciertos que realiza (ver Figura 3b), que es mucho mayor que la de *calculix*.

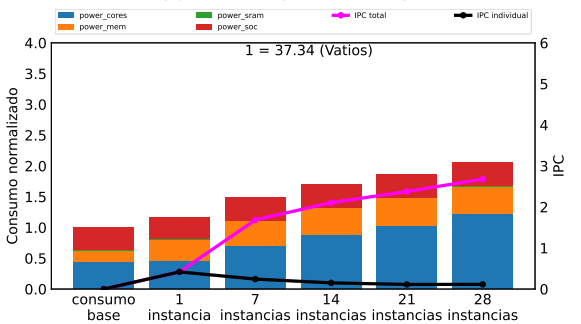
Como se mencionó anteriormente, los accesos a la caché L3 de *mcf* y *xalancbmk_r* presentan un patrón de dientes de sierra. Este patrón se refleja en el consumo de la caché L3 de ambas aplicaciones. Los dientes de sierra son más pronunciados en *mcf*, que también muestra una variación más marcada en su consumo.

CactuBSSN_r (Figura 4c es la aplicación con un mayor consumo de la L3 (cercano a 5,8 vatios). Esto es característico de una aplicación *extreme L3 bound*, como puede verse en la Figura 3c, donde se observa que es la aplicación que accede más a la L3, además de tener un número de aciertos muy elevado (similar al número de accesos).

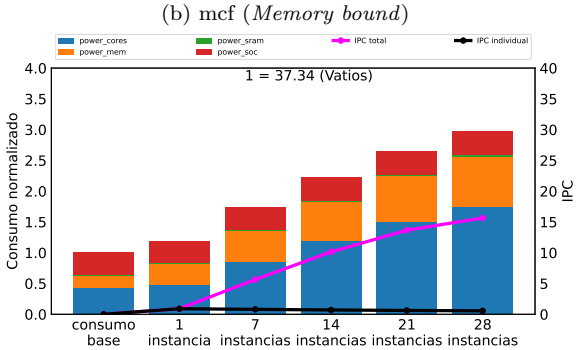
Por último, *xalancbmk_r* (Figura 4d), con un consumo cercano a los 5,3 vatios, es la aplicación con un consumo más estable. Esto ocurre ya que presenta muy pocas variaciones en el número de accesos a la



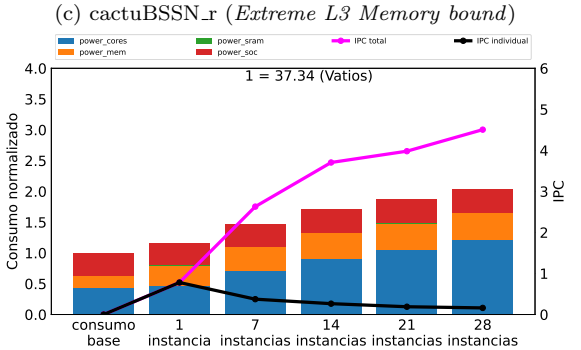
(a) calculix (*CPU bound*)



(b) mcf (*Memory bound*)



(c) cactuBSSN_r (*Extreme L3 Memory bound*)



(d) xalancbmk_r (*L3 Memory bound*)

Fig. 5: Escalabilidad del IPC vs consumo variando el nº de instancias.

caché L3 (Figura 3d).

V. Análisis de interferencias variando el número de instancias por aplicación

A. Prestaciones y consumo

En esta sección se analizan las interferencias entre las aplicaciones en tiempo de ejecución, variando el número de instancias por aplicación hasta llegar a 28, que es el número de núcleos del procesador estudiado. Los experimentos miden los resultados para 1, 7, 14, 21 y 28 instancias. De esta manera, aumentamos el número de núcleos de 7 en 7 a partir de 7, hasta ocupar todos los núcleos, manteniendo también la ejecución individual (1 instancia).

En primer lugar, se caracterizan las interferencias asumiendo que las aplicaciones son del mismo tipo. En este sentido, existen dos opciones: o bien utilizar aplicaciones distintas del mismo tipo, o bien replicar el número de instancias de una misma aplicación. La segunda opción, es la más simple y reproducible, y por tanto más común: utilizar un $\mu benchmark$ y replicarlo tantas veces como sea necesario. En consecuencia, es la elegida en el presente trabajo.

La Figura 5 presenta la variación del *IPC* y del consumo del procesador a medida que aumenta el número de instancias. Dentro del *IPC*, distinguimos entre el de una única instancia y el global, que es la suma del *IPC* de cada una de las instancias que se ejecutan. El consumo está normalizado con respecto al consumo estático base, que es el medido cuando no se ejecuta ninguna aplicación y se distribuye en varias categorías: núcleos (*power_cores*), *SoC* (*power_soc*), caché L3 (*power_mem*) y resto de estructuras de almacenamiento en el procesador (*power_sram*).

En estas figuras, observamos tres patrones de comportamiento según la escalabilidad en prestaciones de las aplicaciones:

Highly-scalable performance. Este comportamiento es el presentado cuando todas aplicaciones son *CPU bound*. En la figura, es el representado por *calculix*. En este tipo de aplicaciones, al ser limitadas por la *CPU*, se genera muy poca contención entre instancias por los recursos de fuera del núcleo (por ej., la caché L3), por lo que el *IPC* por instancia se mantiene, el *IPC* global presenta un crecimiento prácticamente lineal y el consumo crece, también, de manera lineal.
Como cabe esperar, el consumo de los núcleos es también muy elevado, mientras que el consumo de la caché y del *SoC* tienen valores similares, independientemente del número de instancias.

Medium-scalable performance. Este comportamiento es el presentado cuando todas las aplicaciones hacen un uso intensivo de la caché L3 con una alta localidad. El ejemplo lo presenta *cactuBSSN_r* (*extreme L3 bound*), que tiene un *IPC* mayor que todas las aplicaciones *memory bound* pero menor que las *CPU bound*. Estas aplicaciones hacen un acceso intensivo a la L3. La latencia de muchos de estos accesos no puede ser ocultada por la ejecución fuera de orden, pero el hecho de que presenten una alta localidad evita que el *IPC* caiga de manera drástica, consiguiendo alcanzar prestaciones intermedias incluso al aumentar el número de instancias hasta 28.
Respecto al consumo, se aprecia que el de la L3 crece de manera significativa al aumentar el número de aciertos, mientras que el de los núcleos se reduce respecto al de la aplicación *CPU bound*, pasando de 2,3 a 1,8 vatios aproximadamente como consecuencia del menor *IPC*.

Low-scalable performance. Dos tipos de las aplicaciones estudiadas no escalan cuando se incrementa su número. El *IPC* para 28 instancias es inferior a 5 en ambos casos, lo que se traduce en una disminución del consumo, tanto en los núcleos como en la caché L3. Un bajo consumo en la caché indica que o bien se accede poco, o bien que la mayoría de los accesos resultan en fallo. Este comportamiento se presenta cuando la aplicación es *memory bound* (*mcf*) o *L3 bound* con una elevada tasa de fallos.
La caché ayuda a reducir el impacto en las prestaciones de tantos accesos. No obstante, al ejecutar más instancias, estas comienzan a experimentar contención al acceder a la caché, incrementando el número de fallos y por tanto los accesos a memoria principal. Se puede observar cómo, al pasar a siete instancias, el *IPC* individual se reduce drásticamente. Situación que solo empeora a medida que se añaden más instancias. Esto pone de manifiesto la importancia de ejecutar un número reducido de instancias de este tipo de aplicación. Pese a que los vatios consumidos son menos, es necesario considerar que la contención en el acceso y uso de la L3 afecta al tiempo de ejecución de las aplicaciones. Por lo que, al alargarse su tiempo de ejecución, el consumo total de energía puede ser superior al de aplicaciones como *calculix*.

Los resultados ponen de manifiesto la facilidad con la que escalan las aplicaciones limitadas por *CPU* y lo importante que resulta limitar el número de aplicaciones que realizan una gran cantidad de accesos a memoria, pues se perjudican al no ser el sistema capaz de soportar de manera eficiente un elevado *MLP* (*memory level parallelism*). También se observa el elevado peso que tiene el consumo estático sobre el dinámico, especialmente cuando se ejecuta un número reducido de instancias, donde la mayor parte del consumo se debe al estático base y no al procesamiento de la aplicación.

B. Prestaciones por vatio

Para evaluar de manera conjunta las prestaciones y el consumo energético, es necesario utilizar métricas que consideren ambos valores en la misma fórmula. Una métrica típica es las prestaciones por vatio (*performance per watt*), definida como el *IPC* dividido

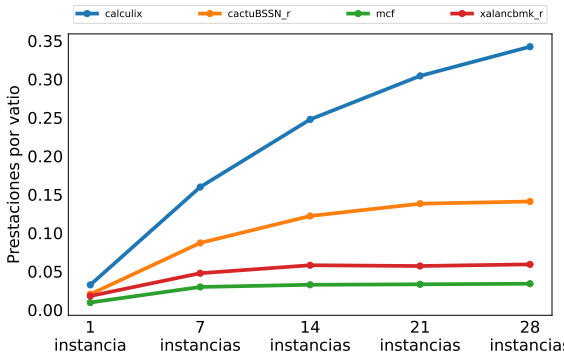


Fig. 6: Prestaciones por vatio variando el número de instancias de las aplicaciones.

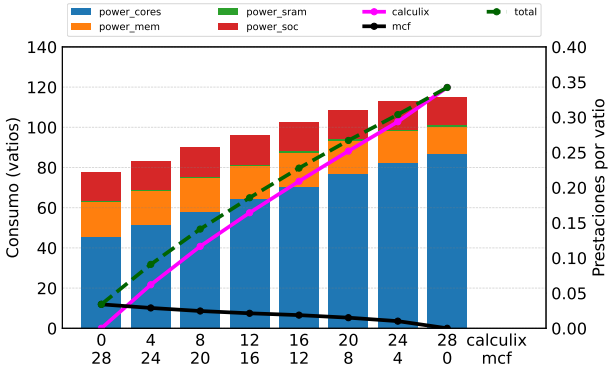
entre el consumo en vatios del procesador. En cierta medida, esta métrica proporciona una estimación de la eficiencia del sistema. Para mejorarla, deben aumentarse las prestaciones para un consumo dado, o reducir el consumo para unas determinadas prestaciones.

La Figura 6 presenta las prestaciones (en términos de *IPC*) por vatio de las distintas instancias y aplicaciones. Los aspectos más destacables son:

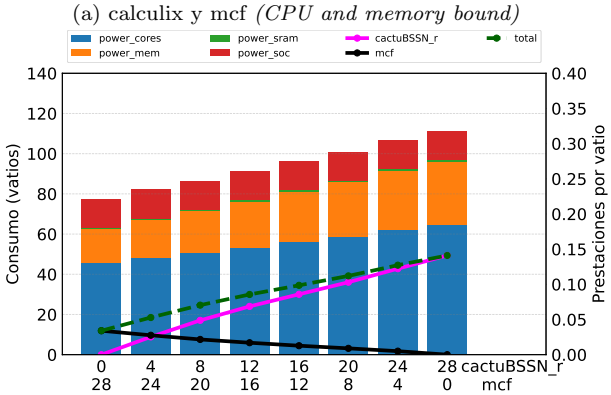
- Las aplicaciones (*calculix*) que presentan *hihgly-scalable performance* siguen siendo las más escalables en prestaciones por vatio. No obstante, se aprecia que las prestaciones por vatio escalan de manera menos pronunciada que si solo se consideran las prestaciones. Esto se debe a que, pese a que las prestaciones mejoran casi linealmente al aumentar el número de instancias, el consumo del procesador se incrementa todavía más.
- Las aplicaciones *extreme L3 bound* que presentan *medium-scalable performance* (*cactuBSSN_r*) son las que tienen una escalabilidad intermedia, aunque las prestaciones disminuyen de manera significativa a partir de 21 instancias por aplicación.
- Las aplicaciones *memory bound* (*mcf*) junto con las *L3 bound* con baja localidad en L3, son también las que presentan peores resultados. Las prestaciones por vatio apenas mejoran de 7 a 14 instancias, para prácticamente estabilizarse a partir de dicho valor.

C. *Combinando aplicaciones con distinto comportamiento*

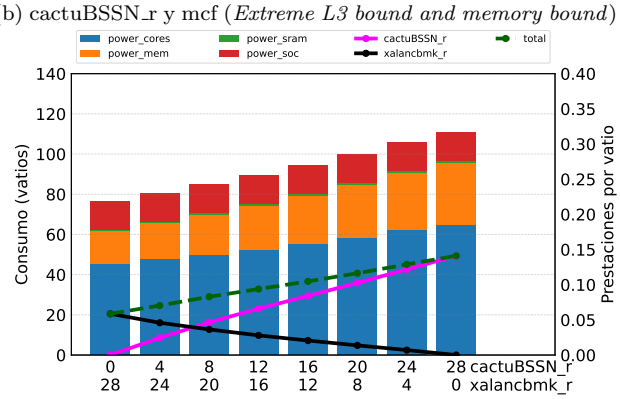
Hasta ahora se ha estudiado el efecto sobre las prestaciones y consumo al ejecutar múltiples instancias de aplicaciones exhibiendo el mismo comportamiento desde el punto de vista de prestaciones. En esta sección se analiza el efecto de mezclar instancias de aplicaciones exhibiendo comportamientos distintos. Para simplificar el estudio, el análisis se restringe a 3 tipos de comportamiento en un mismo experimento. De nuevo, se utilizan las aplicaciones estudiadas como *µbenchmarks* representativos de los diferentes comportamientos. Se persigue identificar el valor óptimo de instancias por aplicación de cada comportamiento, de cara a maximizar la métrica objetivo (por ej., las prestaciones por vatio).



(a) calculix y mcf *(CPU and memory bound)*



(b) cactuBSSN_r y mcf (*Extreme L3 bound and memory bound*)



(c) cactuBSSN_r y xalancbmk_r (*Extreme L3 and L3 bound*)

Fig. 7: Prestaciones por vatio y consumo del sistema variando el número de instancias de las parejas.

En cada experimento, el número de instancias ejecutadas entre dos aplicaciones es igual al número de núcleos (28), distribuidas en diversas combinaciones variando el número de instancias correspondientes a cada aplicación o comportamiento. En concreto, para un número de instancias de un determinado comportamiento 1 ($n1$) y de otro comportamiento 2 ($n2$), donde $n1 + n2 = 28$, se estudia el conjunto de combinaciones ($n1$-$n2$) siguiente: (28-0), (24-4), (20-8), (16-12), (12-16), (8-20), (4-24) y (0-28).

La Figura 7 presenta los valores de prestaciones por vatio y consumo variando el número de instancias de aplicación de cada tipo: *cactuBSSN_r* con *mcf*, *calculix* con *mcf* y *cactuBSSN_r* con *xalancbmk_r*. El consumo se muestra de manera apilada mientras que las prestaciones por vatio se muestran mediante puntos unidos con líneas para estudiar la tendencia.

Se presentan las prestaciones por vatio de cada aplicación y el total.

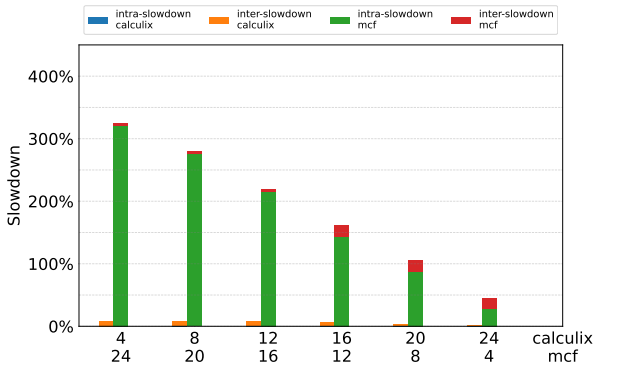
La Figura 7a muestra que las mejores prestaciones por vatio se alcanzan con 28 instancias de *calculix* y 0 de *mcf*. Es decir, cuando todas las aplicaciones son de *CPU bound*. En este caso, el consumo de los núcleos es elevado y el de la cache es más reducido. Debido al alto *IPC* de la aplicación, las prestaciones por vatio son las mayores de todas las combinaciones, alrededor de 0,35 *IPC*/vatio. De manera análoga, en la Figura 7b se observa que las peores prestaciones por vatio se obtienen cuando todas las aplicaciones son *memory bound* (*mcf*). En el caso del consumo de la caché L3, *cactuBSSN_r* (*extreme L3 bound*) es la aplicación que presenta el mayor consumo, llegando a los 30 vatios para 28 instancias. Esto es debido a que esta aplicación es la que más accesos y aciertos realiza en la caché, como se ha mencionado anteriormente. Por último, en la Figura 7b, donde se combinan aplicaciones limitadas por la L3, se muestra un comportamiento similar a la anterior, excepto que las prestaciones por vatio son mayores cuando todas las aplicaciones son limitadas por L3 (*xalancbmk_r*) que cuando todas son limitadas por memoria (*cactuBSSN_r*).

Si miramos el consumo por componente, se observa que los núcleos son los componentes que más consumen del procesador, siendo su consumo mayor que la suma del resto. Este consumo crece con el número de instancias de *calculix (CPU bound)* y de *cactuBSSN_r (extreme L3 bound)* (Figuras 7a y 7b). Estas aplicaciones son, como se ha visto antes, las que presentan los mayores de *IPC*. De hecho, la combinación con 28 instancias de *calculix* presenta el mayor consumo de los núcleos (80 vatios) y de todo el sistema (110 vatios).

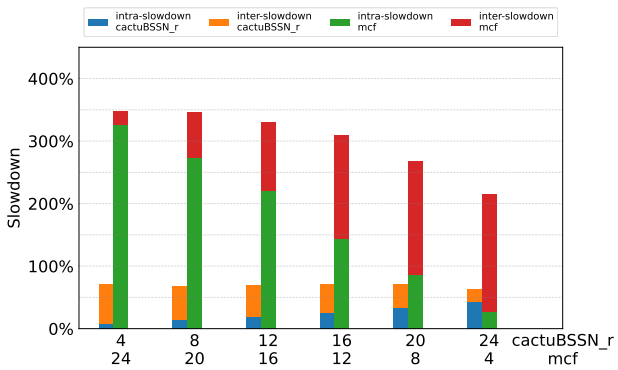
Se puede observar que, para cada combinación, una de las aplicaciones siempre alcanza mejores prestaciones por vatio. Esto provoca que la mejor combinación sea siempre aquella en la que esa aplicación monopolice la *CPU*, perjudicando el avance de la otra aplicación (con unas prestaciones por vatio menores). En consecuencia, se puede concluir que las prestaciones por vatio no son una métrica adecuada, por sí sola, para elegir la mejor combinación si se desea que ambas aplicaciones realicen avances significativos en su ejecución.

Una posible solución consiste en utilizar una nueva métrica que actúe como contrapeso. Con frecuencia se suele utilizar el *slowdown* o degradación de las prestaciones, definida como la pérdida de prestaciones que experimenta una aplicación respecto a si se ejecutase ella sola en el procesador. Esta métrica se suele denominar en la literatura *individual speedup (IS)*. Se suele utilizar de manera directa, por ejemplo, definiendo un *slowdown* máximo, o indirecta. El *IS* se define para una instancia de una aplicación.

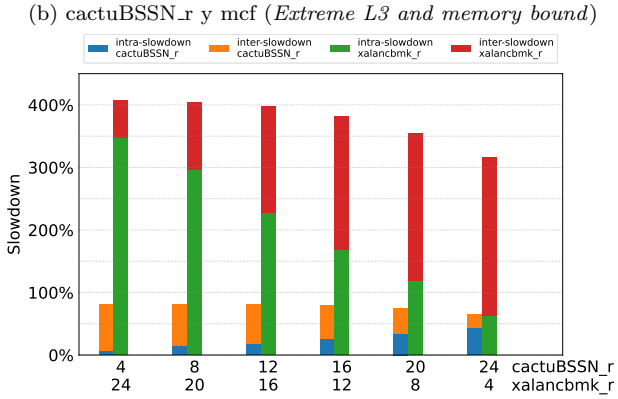
La Figura 8 muestra la degradación de prestaciones para las combinaciones previamente mencionadas. Para calcular la degradación de prestaciones de las aplicaciones, cada instancia se considera como una aplicación independiente, por tanto, debe calcularse el *slowdown* de cada instancia respecto a su ejecución individual. En la figura, la altura de la barra representa la media del *slowdown* (en porcentaje) de las instancias de una aplicación. La barra aparece dividida en dos partes, que muestran la degradación de prestaciones provocada por el resto de instancias de aplicaciones del mismo tipo (*intra-slowdown*) y la causada por las instancias del otro tipo (*inter-slowdown*).



(a) calculix y mcf *(CPU and memory bound)*



(b) cactuBSSN_r y mcf (*Extreme L3 and memory bound*)



(c) cactuBSSN_r y xalancbmk_r (*Extreme L3 and L3 bound*)

Fig. 8: Degradación de las prestaciones variando el número de instancias de cada aplicación

- *CPU and memory intensive*. En la Figura 8a se muestra la degradación para distintas combinaciones de *calculix* y *mcf*. Frente a *calculix*, que se degrada muy poco, *mcf* presenta una degradación mucho mayor, que se incrementa con su número de instancias, llegando al 325 % para 24 instancias de *mcf* y 4 de *calculix*. Esta degradación está provocada mayoritariamente por las otras instancias de *mcf (intra-slowdown)*, ya que

mcf es una aplicación *memory bound* e incrementar el número de instancias de esta aumenta la contención en la memoria.

- *L3 and memory intensive*. En la Figura 8b se muestra la degradación para distintas combinaciones de *cactuBSSN_r* y *mcf*. La tasa de aciertos en la caché L3 de *cactuBSSN_r* es superior a *mcf*, lo que evita una gran degradación. *Mcf* sigue presentando la mayor degradación (cercana al 350 % en el peor caso) y, aunque esta se reduce con su número de instancias, esta reducción es menor que cuando se ejecuta junto a *calculix*. Esto ocurre ya que *cactuBSSN_r*, con su elevado número de accesos a la L3, también interfiere con *mcf*, provocando que su menor degradación supere el 200 %, frente a menos del 50 % cuando se ejecuta con *calculix*.
- *Both L3 intensive*. En la Figura 8c se muestra la degradación para distintas combinaciones de *cactuBSSN_r* y *xalancbmk_r*. En este caso el comportamiento es similar al de *L3 and memory intensive*. La mayor diferencia está en que *xalancbmk_r*, que también está limitada por la L3, sufre mucho más las interferencias provocadas por *cactuBSSN_r*, siendo 300 % la degradación más baja que llega a presentar *xalancbmk_r*.

VI. Conclusiones

Los procesadores de altas prestaciones comerciales de las últimas generaciones continúan evolucionando desde el punto de vista de contadores de consumo energético. Mientras hace apenas unos años el contador de consumo energético medía el consumo global del *processor package*, los procesadores recientes permiten medir distintos componentes del *package*. El objetivo de este trabajo ha sido realizar una primera toma de contacto con estos contadores, y utilizarlos para caracterizar tanto el consumo, como el consumo teniendo en cuenta las prestaciones.

En este trabajo se han caracterizado cuatro comportamientos típicos representativos de las cargas *SPEC CPU*. De los resultados, se pueden extraer las siguientes observaciones y *findings* principales.

- Dado que los núcleos son los que más energía consumen, la mejor combinación en términos de prestaciones por vatio viene dada por aquella que contenga las aplicaciones que alcancen las mayores prestaciones.
- La combinación con mayores prestaciones por vatio coincide con aquella en que ambos tipos de aplicaciones sufren menor *slowdown*.
- Un pequeño número de aplicaciones de memoria intensiva (o L3 intensiva), por ej. 4 en nuestros experimentos, supone un *slowdown* entre el 210 % y el 330 %, lo que inevitablemente lleva a unas pobres prestaciones por vatio. Más aún, este *slowdown* viene dado en más de un 80 % por las escasas aplicaciones de este tipo.

Las observaciones mencionadas conducen a las siguientes conclusiones: un planificador orientado a proporcionar las máximas prestaciones por vatio debe cuidadosamente limitar el número de aplicaciones de memoria intensiva, en la medida de lo posible, con el fin de evitar fuertes *slowdowns* y, en consecuencia, mejorar las prestaciones por vatio.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación junto al FEDER europeo a través de los proyectos TED2021-130233B-C32 y PID2021-123627OB-C51.

Referencias

[1] Andreas Merkel, Jan Stoess, and Frank Bellosa, "Resource-conscious scheduling for energy efficiency on multicore processors," in *Proceedings of the 5th European Conference on Computer Systems*, Paris, France, 2010, Association for Computing Machinery, EuroSys '10, pp. 153–166.

[2] Andreas Merkel and Frank Bellosa, "Balancing power consumption in multiprocessor systems," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 4, pp. 403–414, 2006.

[3] Mwaffaq Otoom, Pedro Trancoso, Hisham Almasaeid, and Mohammad Alzubaidi, "Scalable and dynamic global power management for multicore chips," in *Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures*. Association for Computing Machinery, 2015, pp. 25–30.

[4] Yuhao Zhu and Vijay Janapa Reddi, "High-performance and energy-efficient mobile web browsing on big/little systems," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 13–24.

[5] Lucía Pons, Salvador Petit, Julio Pons, María E. Gómez, Chaoyi Huang, and Julio Sahuquillo, "Stratus: A hardware/software infrastructure for controlled cloud research," in *2023 31th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2023.

[6] "Linux Perf Utility," `https://perf.wiki.kernel.org/`.

[7] "The SPEC organization," `https://www.spec.org/spec/`.

[8] "Linux EAS (Energy-Aware Scheduling)," `https://docs.kernel.org/scheduler/sched-energy.html`.

[9] "Linux kernel documentation - power management: Energy model," `https://docs.kernel.org/power/energy-model.html`, Accessed on May 23, 2023.

[10] Marvell, "MARVELL® ThunderX2® PMU Events (Abridged)," 2019.

[11] Khaled M. Attia, Mostafa A. El-Hosseini, and Hesham A. Ali, "Dynamic power management techniques in multicore architectures: A survey study," *Ain Shams Engineering Journal*, vol. 8, no. 3, pp. 445–456, 2017.

[12] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of energy-cognizant scheduling techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1447–1464, 2013.

[13] Kenzo Van Craeynest, Shoaib Akram, Wim Heirman, Aamer Jaleel, and Lieven Eeckhout, "Fairness-aware scheduling on single-isa heterogeneous multi-cores," in *Proceedings of PACT*, 2013, pp. 177–187.

[14] Vicent Selfa, Julio Sahuquillo, Lieven Eeckhout, Salvador Petit, and María E. Gómez, "Application clustering policies to address system fairness with intel's cache allocation technology," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017, pp. 194–205.

Improving the Performance of Sobol Sequence Generation

Daniel Martínez Davies[1], Santiago Veigas Ramírez[1], J. Daniel García[2], and Javier Fernández Muñoz[2]

Abstract— **Low-discrepancy sequences are particularly useful for efficiently tackling high-dimensional integration which would otherwise require time-consuming and computationally-heavy algorithms for their resolution. Quantitative finance is a field in which such problems are prominent, and execution times are critical. QuantLib is a widely used library that has been designed to undertake quantitative problems and provides an implementation for the low-discrepancy Sobol sequence.**

This paper explores the generation of low-discrepancy Sobol sequences, and presents the performance results obtained after optimizing QuantLib's Sobol generator for improved memory-access patterns and cache usage. To properly assess performance differences, the number of Sobol points and sequence dimensionality have been factored in as parameters of the testing scenarios.

Additionally, NVIDIA's cuRAND Sobol generator has been evaluated in the context of singular task-offloading by means of CUDA's Host API, exploring scenarios where its use could improve or worsen performance, particularly, in applications characterized by the lack of dedicated GPGPU kernels.

Keywords— **Sobol, QuantLib, CUDA, Optimization, Performance.**

I. Introduction

Quasi random numbers are useful for generating uniform sequences. MonteCarlo simulations make use of random sampling for computation and prediction. Though not designed to replicate randomness, the use of quasi random sequences can result in faster convergence, as the generated numbers are more uniform. A popular quasi random generator is the Sobol sequence.

Since these simulations typically involve a large amount of computation, performance is a main concern. Additionally, for accurate results, a massive number of points need to be generated, and therefore algorithms for number generation need to be as fast as possible.

The QuantLib library [1] is widely used for these types of simulations, and provides a fast Sobol generator. Nevertheless, for same cases, the performance on a CPU is not enough, so some have tried to take advantage of the large-scale parallelism offered by GPUs.

[1]Sensia Solutions, e-mail: {dmartinez,sveigas}@sensia-solutions.com.
[2]Computer Science and Engineering Dept., Universidad Carlos III de Madrid, e-mail: {jdgarcia,jfmunoz}@inf.uc3m.es.

II. State of the Art

A. Quasi Random Sequences

Low discrepancy sequences, or quasi random numbers, are useful for generating points that are equidistributed. They are appropriate for cases where pure randomness is not a strict requirement, but rather uniformity, such as certain types of financial problems.

Popular quasi random sequences include both the Halton sequence [2] and the Sobol sequence [3] [4].

Direction integers m are an important feature of Sobol numbers. Primitive polynomials are closely related to these direction integers. For a primitive polynomial of degree d, the first d direction integers can be chosen freely, so long as m_i is an odd integer, $0 < m_i < 2^i$. The chosen free direction integers will affect the uniformity of the sequences[5]. The subsequent values of m are calculated using a recurrence relation that will depend on the primitive polynomial [6].

A vector v can be generated by using the chosen direction integers, where:

$$v_i = m_i/2^i \tag{1}$$

The original Sobol implementation generates a vector x of floating point numbers:

$$x_n = b_1 v_1 \oplus b_2 v_2 \oplus b_3 v_3 ... \tag{2}$$

where $...b_3 b_2 b_1$ is the binary representation of n.

A better implementation in gray code was proposed by Antonov and Saleev [7]:

$$x_{n+1} = x_n \oplus v_c \tag{3}$$

The index c is obtained by finding the position of the rightmost zero bit of n.

Whereas the original method requires several XOR operations, the new proposal only necessitates one, though it requires calculating the value of c.

The resulting vector x contains floating point numbers that are not normalized. A normalization factor can be used to convert into a normalized floating point number. Performance is better for normalized numbers, due to the architecture of computers [8].

Table I: Sobol generation

Sobol Generation					
i	1	2	3	4	5
m_i	1	3	5	3	23
$v_{i(2)}$	0.1	0.11	0.101	0.0011	0.10111
$v_{i(10)}$	0.5	0.75	0.625	0.1875	0.71875

For Sobol sequences, the dimensionality of a problem must be considered. Therefore, for each dimension, a set of direction numbers and its associated primitive polynomial, must be chosen, which will generate a vector v and a sequence x, as shown in Table I. Typical implementations use 32 direction integers per dimension, which will generate up to 2^{32} points per dimension. The first point contains a vector filled with the value 0, though sometimes this first point is discarded [9].

The direction integers of Joe and Kuo are commonly used, as they are open source and have good properties [10] [11]. These direction integers support up to 21,201 dimensions, though with other direction integers it would be possible to increase the number of dimensions.

Having an efficient skip-ahead function is of utmost importance for parallel number generation; without one, a linear dependency between states is created, which in turn becomes a possible source of scalability bottlenecks. The Sobol sequence can be skipped ahead without having to generate intermediate states or values, and therefore it proves to be suitable for GPU or parallel implementations.

Equation 2 can be used to skip ahead to the nth sequence. However, for implementations using gray code, n must first be converted into gray code before applying said formula.

B. Computer Architecture and Performance Impact

On the other hand, for performance, parallelism is very important. In particular, Flynn's Taxonomy explores four different types of parallelism: Single Instruction Single Data(SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD), and Multiple Instruction Multiple Data (MIMD) [12]. For Sobol generation, SIMD will be very important for vectorization on the CPU, and large scale parallelism on the GPU.

Vectorization increases the throughput of a program by applying the same operation to multiple data in a single instruction. x86 and ARM architectures offer assembly instructions for vectorization.

Accessing data is necessary for most computational problems. Certain algorithms require millions of accesses, and storage on the CPU registers is limited. Main memory can be used for storing large amounts of data. However, these memory accesses are slow, and leave the CPU idle without much useful work. As a result, modern day CPUs include cache storage for quicker memory access. They typically have several levels, where each subsequent level has more storage capacity but access times are slower.

When searching for data, the CPU cache is checked first. If not found, the specified and adjacent data is fetched from main memory and stored in cache. This is considered a cache miss. When the data is already stored in cache, it can be quickly accessed, this is considered a cache hit.

Caches take advantage of two types of locality: spatial and temporal locality [13]. Temporal locality states that data that has been used recently, will likely be used again in the near future. Hence, the reason why, when data is retrieved from main memory, it is stored preemptively on the CPU cache.

Spatial locality, on the other hand, indicates that data that is stored close together, is likely to be used again soon. Of course, these are only performance-oriented heuristics; for very specific cases it may make more sense to store other data in cache instead.

Because of the role that cache plays in performance, memory layout is also important. Languages such as C and C++ use a row-major layout for storing vectors and matrices. This means that elements from the same row are stored together in memory, taking advantage of spatial locality. Therefore, due to how modern CPU cache works, it is better to access the elements of a matrix, row by row, as opposed to column by column. This reduces the number of cache misses and improves the overall performance.

III. Problem Description

Random number generation and quasi random number generation is often used for calculating millions of simulations, as the convergence rate can be slow. As a result, performance of the random number generator often becomes a major concern. In many cases, the generation of random sequences can become an important bottleneck. The QuantLib library offers a wide number of generators, including a Sobol generator for quasi random sequences. In order to improve performance, the generation must be profiled with an appropriate tool. In this case, a combination of the Linux perf tool and the Intel VTune Profiler was used.

When analysing the QuantLib generator, the main bottlenecks for generating the Sobol sequences appear to be: XOR functions performed for formula 3 and shown in Figure 1; and the function for normalizing the resulting vector displayed in Figure 2. For the XOR function, accesses to the direction integers are necessary. Since the value of the index c is constant for all dimensions of the same point, the accesses will be very predictable: all access will be from elements of the same column.

Careful observation of Figure 1 reveals that the memory accesses of the direction integer matrix is sub-optimal, as the data is accessed by column instead of by row.

```
1 for (Size k=0; k<dimensionality_; k++)
2     integerSequence_[k] ^= directionIntegers_[k][j];
```

Fig. 1: XOR Operation - Sobol Generator QuantLib

```
1 for (Size k=0; k<dimensionality_; ++k)
2     sequence_.value[k] = v[k] * normalizationFactor_;
```

Fig. 2: Floating-point Normalization - Sobol Generator QuantLib

Nevertheless, this memory access pattern is only done when generating sequences. In the constructor of the QuantLib Sobol class, all of the accesses to

```
for (Size k=0; k<dimensionality_; k++)
    integerSequence_[k] ^= directionIntegers_[j][k];
```

Fig. 3: Xor Optimization Optimized - Sobol Generator QuantLib

the direction integer matrix is done inversely. For these cases, the current memory layout makes sense. However, the constructor is only called once, whereas the function for generating the next sequence may be called millions of times. Therefore, it may be worthwhile to change the memory layout to what is shown in Figure 3.

Another area of interest is to try to vectorize both the XOR operations and the normalization of the floating point number. Nevertheless, both GCC and Clang are capable of auto-vectorizing the code, so the existing implementation does not need to be modified in order to reap the performance benefits of SIMD vectorization.

Another optimization to consider is to modify the algorithm for the normalization of the floating point numbers as seen in Figure 2, since there are other ways to normalize floating point numbers that diverge from the approach used by QuantLib.

One method involves finding the leftmost 1 bit, in order to calculate the correct exponent. The C++ standard library offers certain functions such as std::countl_zero to do so. The original non normalized number will contain the mantissa. The leftmost 1 bit will indicate how many left shifts need to be applied to obtain the correct mantissa. Nevertheless, empirical results showed this alternative to be no faster, and in some cases slower, than the current multiplication. Moreover, std::countl_zero requires C++20. While there are compiler intrinsics that return the same result, they are not portable.

GPUs can be used to improve the performance of compute bound applications, thanks to their large scale parallelism.

Nevertheless, GPU generation for Sobol sequences has to be considered carefully. Direction integers need to be transferred to the GPU. The skip ahead function can be executed on the CPU, generating a vector x for each dimension. However, generating several skip ahead functions and transferring the resulting x vector to the GPU for each core can be expensive. Instead, a faster alternative can be to execute the skip ahead function in parallel in the GPU. Instead of transferring a vector x, only the sequence number has to be sent to the GPU, reducing the amount of data to transfer and increasing parallelism.

Subsequently, each node can generate a stream of numbers in parallel. As the sequence is generated, the result can be stored in the GPU and transferred later.

Nvidia offers their implementation in CUDA for Sobol generation on the GPU.

IV. Experimental Results

The following chapter describes a series performance metrics related to the execution time of the different Sobol generators.

Three Sobol engines have been evaluated: QuantLib's Sobol; NVIDIA's cuRAND Sobol; and a cache-optimized generator, which is based of QuantLib's implementation. For the purposes of these test, the cache-optimized generator will be referred to as 'FastSobol'. This includes the optimization of transposing the direction integer matrix for better performance in the xor operation seen in Figure 3.

NVIDIA's cuRAND Sobol generator was incorporated into the tests in order to additionally measure the impact of offloading Sobol into the GPU by means of the CUDA programming model. A final comparison was done to display the advantages of GPU generation for Sobol sequences, and to understand in which situations a GPU can provide better performance than a CPU.

Both the QuantLib and FastSobol generators, produce double-precision floating-point numbers for their output sequences. CUDA can generate either single or double-precision numbers, both cuRAND generators have been tested and denoted as 'CUDA-32' and 'CUDA-64' respectively.

Table II describes the hardware and software environment used for all of the testing scenarios.

Table II: Hardware & Software testing environment

CPU Configuration	
Model name	AMD EPYC 7763
Cores per socket	64
Threads per core	2
CPU max MHz	3529.052
Cache L1d/i	2 MiB
Cache L2	32 MiB
Cache L3	256 MiB
GPU Configuration	
Model name	NVIDIA A100 SMX4
Memory	80 GB
Stream multiprocessors	108
Software Configuration	
C++ compiler	GCC 11.3
QuantLib	1.30 be7bdfb
NVIDIA Driver	530.30.02
CUDA Toolkit	CUDA 12.1
CUDA Compiler	NVCC 12.1.105
CUDA cuRAND	10.3.2.106

Performance metrics for the different test scenarios are presented below. The following two main parameters were varied during the tests: dimensionality of the Sobol sequences, and the number of quasi-random points to generate per dimension. This way, performance differences can be studied for numerous combinations of Sobol configurations. Joe and Kuo's direction integers were used for every scenario, which limits the number of dimensions to 21,201; however cuRAND imposes a limit of 20,000 dimensions, thus the latter value was used as an upper boundary.

Additionally, in the subsequent figures, some generators have been marked with an asterisk (*) in their names; this denotes that the generated Sobol sequen-

ces have been immediately consumed, as opposed to storing the values in a buffer in memory. The test cases using the generators that lack said marking, better highlight the overheads associated with dynamic memory, and are a better point of comparison with the cuRAND generators since they have similar memory footprints.

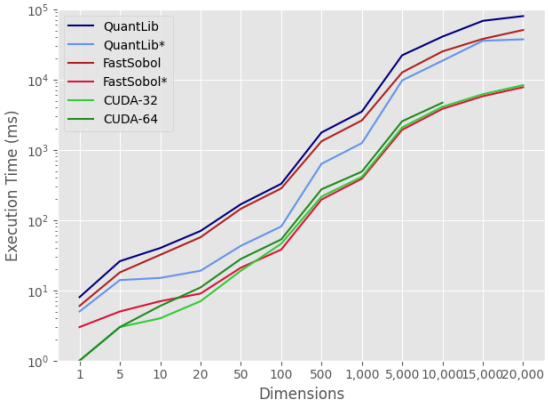


Fig. 4: Sobol performance comparison (2^{20} points)

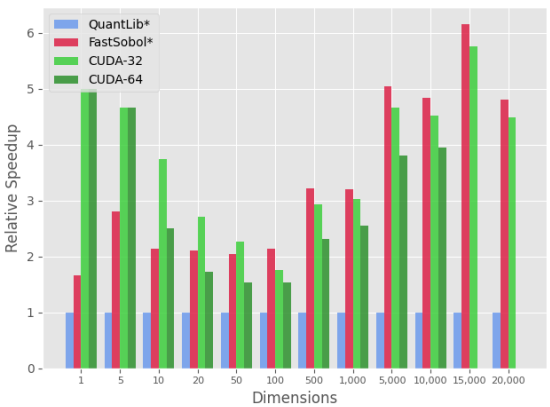


Fig. 5: Sobol relative speedup (2^{20} points)

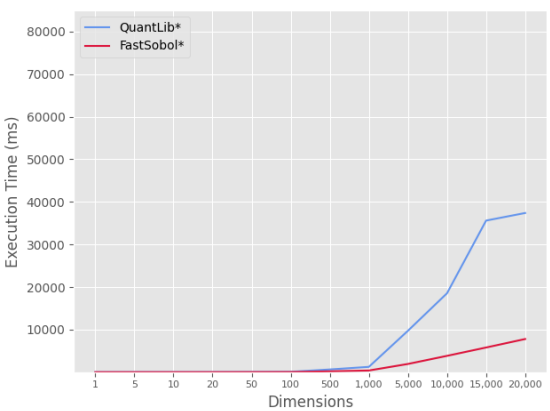


Fig. 6: Sobol fast performance (2^{20} points)

As can be seen in figure 4, the performance of the original QuantLib implementation was improved. The cache optimizations greatly reduced execution times. Similarly, Figure 9 shows the relative speedups normalized with respect to QuanLib's Sobol

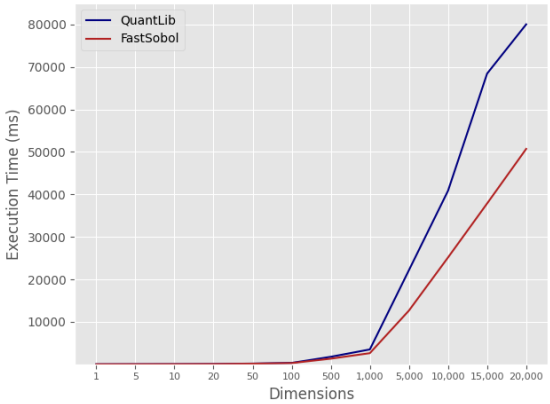


Fig. 7: Sobol slow performance (2^{20} points)

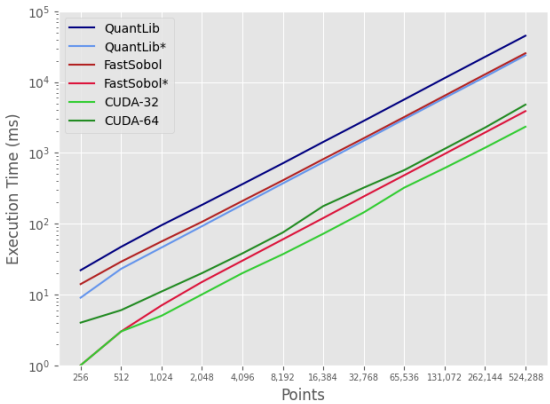


Fig. 8: Sobol performance comparison (20,000 dimensions)

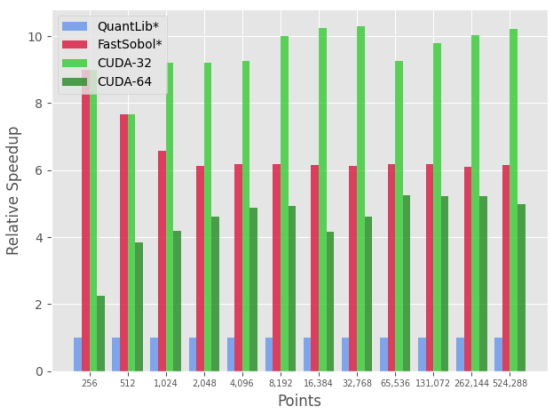


Fig. 9: Sobol relative speedup (20,000 dimensions)

generator. The execution time reductions are more noticeable with increased dimensionality, achieving similar performance to cuRAND's Sobol generator. Nevertheless, it is worth noting that CUDA has to transfer the data from GPU to DRAM.

Figure 6 shows the performance improvements when applying the cache optimization. When no copy is performed, the execution time is greatly reduced, especially for higher dimensions. On the other hand, Figure 7 compares the performance of the two implementations when a copy is made. As can be observed, the performance gain is not as considerable, as a significant portion of the execution time is spent on the data transfer. Therefore, for the generation of points,

it is important to limit data transfers to a minimum to maximize performance.

$$QuantLib_{p\times d} = \left[\begin{array}{c|ccccc} & D_1 & D_2 & D_3 & \cdots & D_d \\ \hline P_0 & q_{01} & q_{02} & q_{03} & \cdots & q_{0d} \\ P_1 & q_{11} & q_{12} & q_{13} & \cdots & q_{1d} \\ P_2 & q_{21} & q_{22} & q_{23} & \cdots & q_{2d} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ P_p & q_{p1} & q_{p2} & q_{p3} & \cdots & q_{pd} \end{array}\right]$$

$$cuRAND_{d\times p} = \left[\begin{array}{c|ccccc} & P_0 & P_1 & P_2 & \cdots & P_p \\ \hline D_1 & q_{10} & q_{11} & q_{12} & \cdots & q_{1p} \\ D_2 & q_{20} & q_{21} & q_{22} & \cdots & q_{2p} \\ D_3 & q_{30} & q_{31} & q_{32} & \cdots & q_{3p} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ D_d & q_{p0} & q_{p1} & q_{p2} & \cdots & q_{dp} \end{array}\right]$$

Fig. 10: QuantLib vs. cuRAND memory layout

The generated quasi-random sequences were verified for each Sobol implementation in order to ensure the correctness of the generated sequences.

It was observed that cuRAND produces its numerical sequences under a different memory layout compared to QuantLib. Figure 10 visually describes the output of the two implementations in terms of matrices. The cuRAND library sorts the output first by point and then by dimension; QuantLib does so in the opposite order. From a matrical standpoint, for cuRAND, each row contains the quasi-random point 'q_p' for a fixed dimension 'd' that corresponds to the row index. Therefore, QuantLib's Sobol generator is cache friendly and better suited for applications that consume the quasi-random sequences in a row-major ordering with respect to the dimensions; cuRAND's output suffers from column-wise access for the previous use-case and performance would significantly degrade.

Offloading the quasi-random number generation to the GPU has multiple overheads related to kernel submission and data transference latencies. To hide such costs, a minimum number of elements must be generated and stored in memory for later use on the CPU. For the previous tests, the batch size was matched to the total number of quasi-random values, as such, only a singular kernel with maximal memory footprint was launched. For some tests, the observed memory usage exceeded 78 GiB, nonetheless memory consumption can be adjusted accordingly to satisfy different hardware constraints.

Comparatively speaking, QuantLib's Sobol generator has a significantly smaller memory footprint. Given that each point has to be iteratively generated, the maximal vector size for a program that immediately consumes the values, would be equal to the number of dimensions. Replicating this behaviour in CUDA yields extremely poor performance, as seen in Figures 11 12. Submitting numerous small GPU kernels not only fails to hide the data transference latencies, but also contribute in the degradation of the overall throughput.

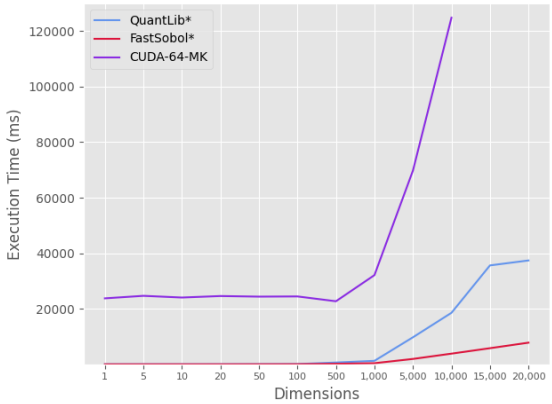


Fig. 11: cuRAND Sobol Micro-Kernels (2^{20} points)

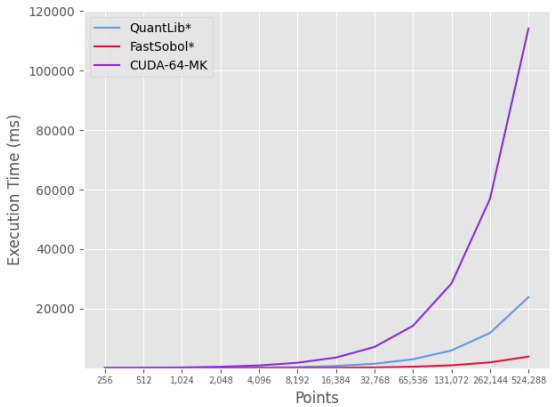


Fig. 12: cuRAND Sobol Micro-Kernels (20,000 dimensions)

Lastly, NVIDIA's cuRAND Sobol engines support the use of custom vectors of direction integers; by default Joe and Kuo's vectors are used [10][11]. It is unclear if cuRAND makes use of look-up tables tailored for the former direction integers in order to improve performance. QuantLib makes no use of such optimization, thus for a custom set of direction integers, execution time between the two implementations may vary significantly with respect to the previously described results; however such hypothesis has not been tested.

V. Conclusions

This paper introduced the basic concepts regarding low-discrepancy Sobol sequences, including alternative state representations such as gray code, and skip-ahead functions to efficiently advance the state of the Sobol generator. Sobol generators are commonly used in fields such as Quantitative Finance, which are subject to strict performance requirements. Consequently, this paper touches on the importance of cache-friendly memory access patterns for high-performance applications.

QuantLib is a commonly used library for financial applications. After profiling with Intel VTune and Linux perf, bottlenecks were identified in the QuantLib's Sobol generator. The former generator was modified to better exploit cache and reduce DRAM accesses. Additionally, the improved gene-

rator was compared against CUDA cuRAND Sobol generator, a parallel GPU implementation of Sobol, in order to better showcase the impact of the performance improvements obtained from the proposed optimizations.

The empirical results obtained demonstrate that the data layout in memory is of upmost importance for performance. Since memory accesses can be very slow, performance can be greatly improved by efficiently using the cache and taking advantage of spatial and temporal locality. Nevertheless, subsequent memory transfers should also be minimized, as they can degrade execution times. The number and speed of memory transfers can affect the overall performance of a given implementation.

Additionally, results showed that QuantLib's Sobol implementation scaled linearly with the number of points and the number of dimensions, even when optimized for better cache utilization. The optimized FastSobol generator nearly approached cuRAND's Sobol execution times. However, it is important to note that FastSobol is a sequential implementation, and cuRAND's Sobol, even though it executes in parallel, suffers from non-negligible memory transfer penalties under the proposed testing scenarios. Nonetheless, the execution time reduction from the cache optimizations is significant, which could be especially useful for multi-process applications; and further performance improvements could be obtained if the Sobol generator were to be parallelized.

Large scale parallelism on GPUs can provide the best performance, however, transferring data back to the CPU is expensive. Ideally, the entire computation would be done at the GPU level, in order to minimise data transfers. Unfortunately, a GPGPU approach would likely require redesigning software components, which may pose development challenges depending on the complexity of the application. Nevertheless, not all algorithms can be parallelized effectively for execution on a GPU. Moreover, profiling and optimizing applications for efficient memory access patterns could prove to be a more time and cost effective alternative, as opposed to applying major refactorizations for GPU computing.

Bibliography

[1] The QuantLib contributors, "QuantLib: a free/open-source library for quantitative finance," .

[2] J. H. Halton, "Algorithm 247: Radical-inverse quasi-random point sequence," *Commun. ACM*, vol. 7, no. 12, pp. 701–702, dec 1964.

[3] Il'ya Meerovich Sobol', "On the distribution of points in a cube and the approximate evaluation of integrals," *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, vol. 7, no. 4, pp. 784–802, 1967.

[4] Il'ya Meerovich Sobol', "On the distribution of points in a cube and the approximate evaluation of integrals," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86–112, 1967.

[5] Ilya M Sobol, "Uniformly distributed sequences with an additional uniform property," *USSR Computational Mathematics and Mathematical Physics*, vol. 16, no. 5, pp. 236–242, 1976.

[6] Paul Bratley and Bennett L. Fox, "Algorithm 659," *ACM Transactions on Mathematical Software*, vol. 14, no. 1, pp. 88–100, 1988.

[7] Ilya A Antonov and VM Saleev, "An economic method of computing lpτ-sequences," *USSR Computational Mathematics and Mathematical Physics*, vol. 19, no. 1, pp. 252–256, 1979.

[8] David Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, 1991.

[9] Art B. Owen, "On dropping the first sobol' point," *Springer Proceedings in Mathematics &; Statistics*, p. 71–86, 2022.

[10] Stephen Joe and Frances Y. Kuo, "Remark on algorithm 659," *ACM Transactions on Mathematical Software*, vol. 29, no. 1, pp. 49–57, 2003.

[11] Stephen Joe and Frances Y. Kuo, "Constructing sobol sequences with better two-dimensional projections," *SIAM Journal on Scientific Computing*, vol. 30, no. 5, pp. 2635–2654, 2008.

[12] M.J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966.

[13] John L Hennessy and David A Patterson, *Computer Architecture: A Quantitative Approach*, The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science & Technology, San Francisco, 2011.

Análisis de algoritmos de clasificación para la detección de texturas en bloques de codificación de vídeo

Javier Ruiz Atencia[1], Otoniel López Granado[1], Manuel Pérez Malumbres[1], Miguel Onofre Martínez-Rach[1] y Héctor Migallón[1]

Resumen— **Este estudio presenta una comparación exhaustiva de distintos algoritmos de clasificación supervisada para la detección de texturas en el contexto de la codificación de vídeo híbrida basada en bloques. Para ello, se ha elaborado un dataset de imágenes extraídas directamente del particionado de bloques de codificadores de vídeo, y se ha clasificado manualmente según su nivel de textura. El estudio emplea el algoritmo Mean Directional Variance (MDV) para extraer la información relativa a la orientación de cada bloque, en forma de varianzas promedio para determinadas pendientes racionales. Este vector de varianzas es procesado posteriormente para obtener una serie de estadísticos que se utilizan como elementos de entrada para entrenar y evaluar cuatro modelos populares de aprendizaje supervisado: Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM) y Supervised Neural Networks (SNN). El objetivo es identificar el algoritmo más eficaz para clasificar con precisión los niveles de textura y utilizar esta información en la codificación de vídeo perceptual.**

Palabras clave— **Algoritmos de clasificación, detección de textura, MDV, codificación perceptual.**

I. Introducción

En la codificación de vídeo perceptual, la información sobre el nivel de textura en los bloques en los que se divide una imagen es esencial para explotar el efecto del enmascaramiento por textura dado por el sistema visual humano (HVS), según el cual, el ruido o error de cuantización es menos perceptibles en regiones con un alto nivel de textura, mientras que son mucho más notables en zonas con baja textura. Por lo tanto, la detección adecuada de la información de textura en la codificación perceptual de vídeo se vuelve especialmente relevante para garantizar un correcto funcionamiento de las técnicas de codificación perceptuales que aprovechan las características del HVS para comprimir en las regiones donde el ojo humano es incapaz de detectar estos errores.

La detección del nivel de textura en la codificación de vídeo perceptual requiere de algoritmos de clasificación robustos y precisos. Estos algoritmos deben ser capaces de identificar de manera confiable tipo de textura presente en cada bloque y asignarle la etiqueta correspondiente. La elección de un algoritmo de clasificación eficiente tiene un impacto directo en el rendimiento de algoritmos de codificación perceptual y, por lo tanto, en la calidad visual del vídeo.

La motivación detrás de este análisis comparativo radica en la necesidad de evaluar y comparar diferentes algoritmos de clasificación para la detección de textura en la codificación de vídeo perceptual. Si bien existen varios enfoques y métodos propuestos en la literatura, es esencial comprender sus fortalezas y limitaciones para seleccionar el algoritmo más adecuado para una aplicación específica. Para llevar a cabo este estudio, se ha creado un dataset de bloques de diferentes tamaños, desde 8x8 hasta 64x64 píxeles, los cuales han sido clasificados manualmente según su nivel de textura.

En este estudio, presentamos un análisis comparativo exhaustivo de diferentes algoritmos de clasificación para la detección de textura en bloques de codificación de vídeo. Nuestro objetivo es evaluar y comparar el rendimiento

[1]Dpto. de Ingeniería de Computadores, Universidad Miguel Hernández de Elche, e-mail: {javier.ruiza,otoniel,mels,hmigallon,mmrach}@umh.es.

de estos algoritmos en términos de precisión y eficiencia. Los resultados obtenidos proporcionarán una guía para futuras investigaciones y desarrollos en el campo de la codificación perceptual de vídeo.

La estructura de este estudio es la siguiente: a lo largo de la SecciónII, se detalla la metodología utilizada, que abarca la elaboración del dataset (II-A), la utilización del algoritmo MDV para determinar el nivel de textura de los bloques (II-B), el análisis comparativo de diferentes algoritmos de clasificación supervisados (II-C), la selección de características (features) más relevantes para estos modelos (II-D), la descripción del proceso de entrenamiento (II-E) y la elección de métricas utilizadas para su evaluación (II-F). Los resultados obtenidos y la discusión asociada se presentan en la Sección III. Finalmente, en la Sección IV, se resumen las conclusiones del estudio y se plantean las líneas futuras de investigación para ampliar este trabajo.

II. Metodología

A. Descripción del dataset

Para preparar el dataset de entrenamiento se han obtenido de forma aleatoria más de 2000 bloques, particionados en tamaños de 8×8, 16×16, 32×32 y 64×64 píxeles, extraídos de las bases de datos ESPL Synthetic Image Database [1], USC-SIPI Image Database [2], TESTIMAGE [3] y Kodak image dataset [4]. Cada bloque es etiquetado de forma manual según su nivel de textura, pudiendo ser bloques de tipo "Plain"(generalmente lisos, con poca actividad espacial), ."Edge"(con un borde o dirección claramente marcado) o "Texture"(con mucha actividad espacial).

Estos tamaños de bloque se han seleccionado por ser los más representativos en el particionado de fotogramas en los principales codificadores de vídeo actuales. Se ha descartado el uso de bloques de tamaño 4×4 ya que con tan pocos píxeles es difícil determinar una clasificación óptima.

La elección del etiquetado en estos tres niveles de textura no se ha escogido solamente porque sea la más utilizada en la literatura a lo largo de los años, como muestran los artículos [5] [6] [7] [8], sino porque tiene implicaciones en el sistema visual humano. En [5], los autores demostraron que los bloques Textura son los que tienen más margen de aplicar cierto error de cuantización sin ser detectado, mientras que a los bloques Plain no se debería aplicar ningún tipo de error adicional. Por último, a los bloques Edge sí se les puede aplicar cierto error de cuantización, pero en menor medida en comparación con los Texture.

El proceso de etiquetado de los bloques se ha llevado a cabo de manera manual, utilizando un entorno gráfico desarrollado en MATLAB. Para cada clasificación, la herramienta muestra el bloque original, así como su versión suavizada con un filtro de mediana. La versión suavizada se ha añadido para ayudar a decidir la clasificación en los casos donde a partir del bloque original es difícil determinar el nivel de textura, como ocurre en la frontera con los bloques Plain.

El conjunto de datos final contiene una variedad de bloques de diferentes tamaños y niveles de textura, lo que proporciona una representación amplia de las características que se encuentran en la codificación de vídeo. En la Tabla I se detalla la distribución de los bloques etiquetados según su etiqueta y tamaño. Se puede observar que la distribución del dataset se encuentra desbalanceada, sin embargo, esto no va a suponer ningún problema, ya que los algoritmos utilizados y las métricas de evaluación tienen esto en cuenta en el proceso de entrenamiento. Este conjunto de datos etiquetado manualmente se utiliza como base para el entrenamiento y evaluación de los algoritmos de clasificación en este estudio comparativo.

Tabla I: Distribución del dataset según etiquetado y tamaño de bloque.

Tamaño	Etiqueta			Total
	Plain	Edge	Texture	
8	128	129	103	360
16	227	165	135	527
32	182	145	157	484
64	180	168	283	631
Total	717	607	678	2002

B. Detección de textura basado en MDV

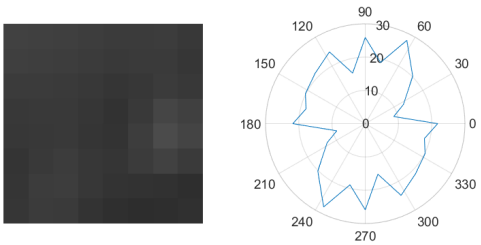
Una vez elaborado el dataset, se ha decidido descartar el uso de redes neuronales convolucionales (CNN) para la creación del modelo de clasificación. Las CNN son ampliamente utilizadas y efectivas para la clasificación de imágenes complejas y de alta resolución, ya que están diseñadas para aprovechar la estructura espacial de las imágenes, utilizando capas de convolución y pooling para extraer características locales y construir una representación jerárquica de la imagen.

Sin embargo, nuestro dataset contiene bloques de baja resolución, cuya información espacial y relación entre los píxeles no son lo suficientemente rica para aprovechar al máximo las capacidades de estas redes neuronales. A esto habría que sumar el coste computacional que supondría añadir esta arquitectura al codificador de vídeo, como en [9], los tiempos de decodificación se ven incrementados en un 11656 %.

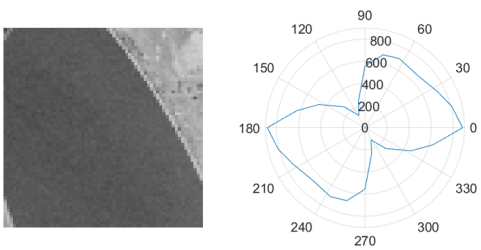
Por tanto, en lugar de utilizar los píxeles de intensidad directamente como entrada para los modelos, se ha optado por una aproximación en la cual se hace uso del algoritmo Mean Directional Variance (MDV) [10] [11] para obtener la orientación del gradiente de los bloques, ya que como veremos a continuación, existe una fuerte relación entre los valores de esta métrica y el nivel de textura de los bloques.

El algoritmo MDV se basa en calcular la varianza direccional media a lo largo de ciertas direcciones espaciales con pendientes racionales. En [11], los autores utilizan un total de 24 pendientes racionales, lo cual mejora la obtención de la direccionalidad dominante, sin embargo, para nuestro propósito no es necesaria tal nivel de precisión, siendo suficiente el cómputo de la MDV utilizando únicamente 12 pendientes. Una de las principales ventajas de trabajar con este algoritmo radica en que, para cada bloque, se obtiene un vector de 12 gradientes, sin importar el tamaño del bloque, lo que simplifica el proceso de entrenamiento y permite el uso de modelos de clasificación más sencillos.

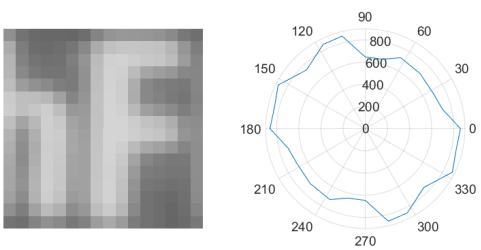
En la Figura 1 se muestran tres ejemplos de bloques clasificados manualmente junto con la representación en diagrama polar de la métrica MDV. A simple vista es fácil observar que existe una fuerte relación entre el nivel de textura de un bloque y su métrica MDV asociada. Para bloques Plain, la forma del diagrama polar puede resultar variable, sin embargo, el valor de la dirección con mayor varianza es relativamente bajo si se compara con los otros dos tipos de bloque. Para los bloques Edge, existe siempre un mínimo (gradiente dominante) en la dirección de la orientación del bloque. Por último, para los bloques Texture, los datos extraídos de la métrica son elevados, careciendo de un mínimo dominante.



(a) bloque de 8 × 8 clasificado como Plain



(b) bloque de 64 × 64 clasificado como Edge



(c) bloque de 16 × 16 clasificado como Texture

Fig. 1: Muestra de bloques etiquetados manualmente (izquierda) y su representación polar de la métrica MDV (derecha).

A partir de esta observación, se han extraído para cada bloque etiquetado manualmente, algunas estadísticas descriptivas del vector de

gradientes, como son el valor mínimo, el máximo, el promedio, el rango o su varianza. Al representar en una gráfica de dispersión, tal y como se muestra en la Figura 2, se puede comprobar que los bloques se agrupan en clústeres muy bien definidos según su nivel de textura. Esto demuestra la observación de la fuerte relación entre el nivel de textura y su métrica MDV anteriormente expuesta, y permite utilizar estos estadísticos como elementos de entrada o características a los diferentes modelos de entrenamiento utilizados en este estudio.

C. Selección de modelos de clasificación supervisado

La elección de los modelos de aprendizaje supervisado utilizados en este estudio comparativo se basa en su relevancia y eficacia demostrada en la detección y clasificación de patrones a partir de variables predictivas. Cada uno de estos modelos presenta características distintas que los hacen adecuados para abordar este problema desde diferentes enfoques.

Decision Tree (DT): El árbol de decisión es un modelo de clasificación intuitivo y fácil de interpretar. Construye un modelo en forma de árbol en el que cada nodo interno representa una característica o atributo, y cada hoja representa una etiqueta de clase. El algoritmo divide de forma recursiva los datos de entrenamiento en función de los valores de las características, con el objetivo de maximizar la separación de las clases en cada paso. Siguiendo las rutas desde la raíz hasta las hojas, el árbol de decisión asigna etiquetas de clase a nuevas instancias según sus valores de características.

Random Forest (RF): Este modelo combina múltiples árboles de decisión, conocidos como bosque, para realizar las predicciones. Cada árbol de decisión se construye utilizando un subconjunto aleatorio de características y muestras de entrenamiento, promoviendo la diversidad y reduciendo el riesgo de sobreajuste. Durante la fase de predicción, se combinan las predicciones individuales de cada árbol para alcanzar una etiqueta de clase final mediante votación mayoritaria o promediado.

Support Vector Machine (SVM): Su objetivo es encontrar un hiperplano óptimo que separe las diferentes clases al maximizar el margen entre ellas. Esto se logra mediante la transformación de los datos de entrada en un espacio de mayor dimensión y la identificación de la frontera de decisión óptima. Asigna puntos de datos a diferentes clases en función de su posición relativa al hiperplano. SVM es particularmente efectivo para manejar conjuntos de datos complejos con relaciones no lineales a través del uso de funciones de kernel.

Supervised Neural Networks (SNN): son modelos inspirados en la estructura y funcionalidad de las redes neuronales biológicas. Estas redes pueden aprender patrones y características complejas de los datos mediante un proceso de entrenamiento con datos etiquetados. Las SNN constan de neuronas artificiales interconectadas y organizadas en capas. El algoritmo aprende ajustando los pesos y sesgos de las neuronas a través de un proceso llamado retro propagación, en el que se minimizan los errores entre las etiquetas de clase predichas y las reales.

Los cuatro algoritmos tienen relevancia en el contexto de la clasificación de datos debido a sus características y capacidades particulares. Los DT y los RF son especialmente adecuados para capturar patrones y reglas basados en características específicas de los bloques de imagen. Las SVM son efectivas en la clasificación de datos linealmente no separables y pueden encontrar fronteras de decisión complejas. Las SNN, por su parte, tienen la capacidad de aprender y capturar características complejas. Teniendo en cuenta la simplicidad y robustez, estos cuatro algoritmos proporcionan una base sólida para comparar y evaluar su rendimiento en nuestro estudio.

D. Extracción y preprocesado de características

En el apartado II-B se ha determinado que el conjunto de características o elementos de entrada a los modelos de aprendizaje a evaluar no sean los píxeles de intensidad de los bloques, sino su vector de gradientes obtenido mediante el algoritmo MDV. A partir de este vector, también se han extraído numerosos estadísticos que también pueden ser utilizados

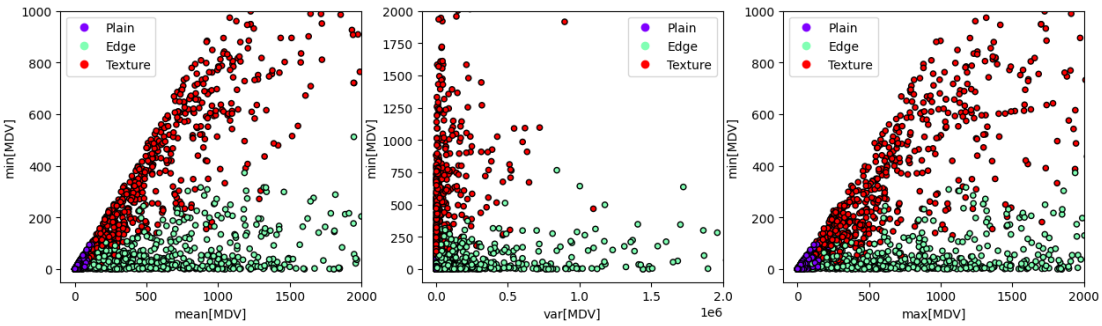


Fig. 2: Diagrama de dispersión de los bloques etiquetados manualmente según algunos de sus estadísticos descriptivos. A la izquierda, el mínimo frente al promedio, en el centro, el mínimo frente a la varianza, y a la derecha, el mínimo frente al máximo.

como características, ya que están fuertemente relacionados con el etiquetado o clasificación de los bloques. Por último, otra característica adicional que se ha tenido en cuenta ha sido la indicación del tamaño del bloque.

Antes de utilizar todo este conjunto de datos de entrada para el entrenamiento y evaluación de los algoritmos de clasificación, en función del tipo de algoritmo, se ha llevado a cabo una serie de pasos de preprocesamiento de los datos para garantizar la calidad y coherencia de los resultados. Los pasos principales involucrados en este preprocesamiento son los siguientes:

División del dataset: Consiste en la separación del conjunto de datos en dos grupos, el grupo de entrenamiento y el grupo de testeo. De esta forma, se evalúa la capacidad de generalización de los modelos de clasificación. La proporción asignada ha sido la de un 80 % para el grupo de entrenamiento y un 20 % para el grupo de testeo. El conjunto de datos se entrada se ha dividido en dos partes, la parte de entrenamiento y la parte de testeo, asegurando que ambos contuvieran una representación equilibrada de bloques Texture, Edge y Plain. Esta división se ha mantenido para los cuatro algoritmos de clasificación evaluados.

Validación cruzada: Debido al tamaño del dataset utilizado, relativamente pequeño, se ha decidido utilizar la estrategia de validación cruzada, concretamente la función *StratifiedShuffleSplit* de la biblioteca Scikit-Learn (sklearn). Esta función divide los datos de entrada en conjuntos de entrenamiento y testeo de forma estratificada y aleatoria. Esto permite validar que los resultados obtenidos sean independientes del conjunto de entrenamiento y testeo, aportando mayor confiabilidad a los resultados obtenidos. En este estudio, se ha seleccionado un total de 5 divisiones para los cuatro algoritmos a evaluar.

Estandarización de los datos: Consiste en la transformación de los datos de entrada, de forma que a cada característica se le resta la media y se escala para conseguir una desviación estándar unitaria. De esta forma, se asegura que las características se encuentren en un rango consistente, lo cual ayuda a evitar sesgos en el entrenamiento de los modelos. Para realizar el estandarizado se ha utilizado la función *StandardScaler* de la biblioteca sklearn. Este paso, a diferencia de los anteriores, se ha aplicado únicamente a los modelos SVM y SNN y no a los modelos DT y RF. Esto es debido a que los algoritmos basados en árboles de decisiones se basan en reglas simples que comparan los valores de las características directamente en cada nodo, no siendo afectados por la escala de estas.

Aumento de datos: Únicamente utilizado en el modelo SNN, consiste en incrementar artificialmente el tamaño del dataset, realizadon diferentes transformaciones a los bloques etiquetados manualmente. Estas transformaciones han consistido en rotar 90, 180 y 270 grados cada bloque, así como aplicar un volteo, lo cual se traduce en la permutación y/o inversión del vector de gradientes, consiguiendo incrementar

7 veces el tamaño del dataset original. Sin embargo, para todas estas transformaciones, las estadísticas descriptivas permanecen constantes, por lo que la aumentación de datos no tiene ningún impacto si se utilizan estos estadísticos como elementos de entrada.

E. Entrenamiento de modelos

La implementación de los diferentes algoritmos de aprendizaje automático se ha llevado a cabo utilizando las bibliotecas de machine learning Scikit-Learn (sklearn) y Keras para Python 3.10. Estas bibliotecas ofrecen una amplia gama de herramientas y funciones para la implementación y evaluación de algoritmos de aprendizaje automático.

Para la búsqueda sistemática de los hiperparámetros óptimos se han empleado diferentes métodos en función del modelo. En el caso de los algoritmos DT, RF y SVM, se ha empleado la clase *GridSearchCV*, que permite explorar diferentes combinaciones de parámetros y seleccionar aquella configuración que maximice el rendimiento del modelo. En la Tabla II se listan los parámetros que se han utilizado para la búsqueda de hiperparámetros óptima.

Tabla II: Listado de los algoritmos de clasificación que utilizan la clase *GridSearchCV* y sus hiperparámetros asociados.

Algoritmo	Hyper parámetros
Decision Tree	(criterion, max_depth)
Random Forest	(criterion, max_depth, max_features, n_estimators)
SVM	(C, kernel='linear', decision_function_shape), (C, kernel='rbf', gamma, decision_function_shape)

En cuanto a las redes neuronales supervisadas, se ha utilizado la librería KerasTuner para la búsqueda de hiperparámetros. KerasTuner proporciona una interfaz sencilla y eficiente para optimizar las redes neuronales supervisadas. Esta herramienta permite definir el espacio de búsqueda de hiperparámetros y automatiza el proceso de ajuste de los modelos, facilitando así la obtención de resultados óptimos.

En el diseño de las SNN para este estudio, se han desarrollado tres modelos diferentes, cada uno de ellos con una configuración específica de capas y dropout. El objetivo principal de esta variación es explorar diferentes arquitecturas y evaluar su rendimiento. Todos estos modelos son compilados utilizando el optimizador Adam, con una función de pérdida denominada çategorical_crossentropy", comúnmente utilizada en problemas de clasificación multi clase, como es el caso de nuestro estudio. Además de la función de pérdida, se definen dos métricas de evaluación adicionales, recall y precision, que proporcionan una medida adicional del rendimiento del modelo durante el entrenamiento y evaluación. En la Sección II-F se detallarán estas métricas.

Con respecto a la arquitectura de los modelos, el primero de ellos está compuesto por una capa de entrada, que recibe los datos de entrada. A continuación, se aplica una técnica de regularización llamada dropout, que aleatoriamente desactiva un cierto porcentaje de las neuronas de la capa anterior durante el entrenamiento. Esto ayuda a prevenir el sobreajuste y mejorar la generalización del modelo. Finalmente, se incluye una capa de salida. El segundo modelo de SNN presenta una arquitectura ligeramente más compleja. Además de la capa de entrada y el dropout inicial, se ha añadido una capa intermedia y otro dropout antes de la capa de salida. El tercer modelo de SNN incorpora otra capa intermedia adicional y otro dropout tras ella.

Cada una de estas capas, así como los ajustes en cuanto a tasa de aprendizaje y tasa de dropout en los tres modelos ha sido sometida a una búsqueda exhaustiva de hiperparámetros utilizando KerasTuner, permitiendo ajustar los parámetros de cada capa, como el número de neuronas, el tipo de función de activación y la tasa de dropout, así como la tasa de aprendizaje del modelo, con el objetivo de encontrar la configuración óptima que maximice el rendimiento del modelo.

Con respecto a los datos de entrada o características, dada la clara segmentación de los bloques dados a partir de las estadísticas descriptivas de los vectores de gradiente visto en la Figura 2, se ha comprobado que utilizar las

estadísticas en lugar del vector de gradientes es más que suficiente para todos los algoritmos evaluados en este estudio. Concretamente, las estadísticas utilizadas han sido las siguientes: mínimo, máximo, promedio y varianza. A estas estadísticas, habría que sumar una característica más, el tamaño del bloque, que incrementa ligeramente los resultados obtenidos.

Aún con todo, para los modelos basados en redes neuronales, se ha decidido probar también la opción de utilizar directamente el vector de gradientes más el tamaño del bloque. Para esta segunda opción, que se identificará como SNN*, se ha aplicado el preprocesado aumento de datos, explicado en la Sección II-D.

F. Evaluación de modelos

Para evaluar el rendimiento de los diferentes algoritmos de aprendizaje automático, así como para maximizar la búsqueda de hiperparámetros en los modelos SNN, se han utilizado las siguientes métricas:

Precision (Precisión): mide la proporción de instancias clasificadas correctamente como positivas entre todas las instancias clasificadas como positivas por el modelo. Es decir, cuántas de las predicciones positivas del modelo son realmente positivas. Se calcula dividiendo el número de verdaderos positivos (TP) entre la suma de los verdaderos positivos y los falsos positivos (FP). Se calcula utilizando la siguiente fórmula: $Precision = TP/(TP + FP)$.

Recall (Exhaustividad): mide la proporción de instancias positivas que fueron correctamente identificadas por el modelo en comparación con todas las instancias positivas reales. Es decir, cuántas de las instancias positivas fueron correctamente detectadas por el modelo. Se calcula dividiendo el número de verdaderos positivos (TP) entre la suma de los verdaderos positivos y los falsos negativos (FN). Se calcula utilizando la siguiente fórmula: $Recall = TP/(TP + FN)$.

F1-score: es una medida que combina la precisión y la exhaustividad en un solo valor. Es la media armónica de la precisión y el recall y proporciona una medida equilibrada del rendimiento del modelo. Se calcula utilizando la siguiente fórmula: $F1 = 2 * (Precision * Recall)/(Precision + Recall)$.

Estas métricas son útiles para verificar y comparar modelos de aprendizaje automático en conjuntos de datos desbalanceados ya que ofrecen una evaluación integral del rendimiento del modelo, considerando tanto los casos positivos como los negativos.

III. Resultados y discusión

Los resultados obtenidos para cada uno de los algoritmos de aprendizaje automático optimizados, con el fin de obtener la mayor puntuación, se presentan en la Tabla III. Estos resultados reflejan el rendimiento de los modelos en términos de su capacidad para clasificar correctamente los bloques en el conjunto de datos. Se puede observar que los algoritmos SVM, SNN (Model 2) y RF muestran los mejores resultados para todas las métricas evaluadas. Estos tres modelos logran un equilibrio destacado entre la precisión y la capacidad para identificar correctamente los casos positivos.

Tabla III: Resultados promedio de los diferentes modelos de aprendizaje automático configurados con los parámetros óptimos encontrados. Los modelos SNN* corresponden con los entrenados con el vector de gradientes como entrada.

Algoritmo	Precisión	Recall	F1 Score
Decision Tree	0.9350	0.9347	0.9347
Random Forest	0.9600	0.9596	0.9596
SVM	0.9624	0.9621	0.9621
SNN (Model 1)	0.9508	0.9501	0.9498
SNN (Model 2)	0.9601	0.9601	0.9600
SNN (Model 3)	0.9579	0.9576	0.9573
SNN* (Model 1)	0.8827	0.8841	0.8822
SNN* (Model 2)	0.9071	0.9080	0.9078
SNN* (Model 3)	0.9261	0.9263	0.9261

Es importante destacar el rendimiento obtenido por el algoritmo de Árbol de Decisión (DT), a pesar de ser inferior en comparación con los otros tres modelos. Su simplicidad permite su uso en aplicaciones en tiempo real, lo cual es una ventaja en ciertos escenarios.

En relación a las redes neuronales supervisadas, se observa que el uso directo del vector de gradientes (modelos SNN*) como características de entrada resulta en un menor rendimiento en comparación con el uso de sus estadísti-

cos descriptivos (modelos SNN). Sin embargo, es importante mencionar que solo se evaluaron arquitecturas de hasta 3 capas. Como se muestra en la tabla, el rendimiento de los modelos SNN* mejora a medida que se agrega una nueva capa oculta a la arquitectura. Es posible que con más capas se obtenga un rendimiento similar a los modelos SNN.

En particular, los modelos SNN muestran los mejores resultados con dos capas ocultas, aunque los resultados con una única capa también son prometedores. Esta arquitectura más sencilla de implementar puede ser preferible en ciertos casos.

En relación a la búsqueda de hiperparámetros óptimos, la Tabla IV muestra los valores seleccionados para cada uno de los algoritmos presentados en este estudio. Para los modelos SNN y SNN*, se presentan únicamente los parámetros óptimos para la arquitectura con mayor rendimiento dentro de su clase.

Tabla IV: Listado de parámetros óptimos encontrados para cada algoritmo de aprendizaje evaluado.

Algoritmo	Parámetros	Valor
Decision Tree	criterion	gini
	max_depth	8
Random Forest	criterion	log_loss
	max_depth	10
	max_features	sqrt
	n_estimators	9
SVM	C 1000	
	kernel	linear
	decision_function_shape	ovr
SNN (Model 2)	units layer 1	12
	dense_activation 1	relu
	droput 1	0
	units layer 2	6
	dense_activation 2	tanh
	droput 2	0.08
	learning_rate	0.01
	epochs	50
SNN* (Model 3)	units layer 1	12
	dense_activation 1	tanh
	droput 1	0.02
	units layer 2	8
	dense_activation 2	sigmoid
	droput 2	0.01
	units layer 3	10
	dense_activation 3	tanh
	droput 3	0.06
	learning_rate	0.01
	epochs	60

IV. Conclusiones

En este estudio, se ha desarrollado un dataset de entrenamiento para la clasificación de bloques según su nivel de textura en el contexto de la codificación de vídeo híbrida basada en bloques. Se ha llevado a cabo un análisis exhaustivo de diversos algoritmos de aprendizaje supervisado con el fin de automatizar la clasificación de los bloques y determinar cuál de ellos ofrece el mejor rendimiento para su aplicación en la codificación de vídeo perceptual.

Los resultados obtenidos han revelado que los algoritmos SVM, Redes Neuronales Supervisadas (SNN) y Random Forest (RF) presentan los mejores resultados en cuanto a las principales métricas evaluadas. Estos modelos han logrado un equilibrio destacado entre precisión y capacidad de clasificación, lo cual los posiciona como opciones prometedoras para la clasificación de bloques en la codificación de vídeo.

Un hallazgo interesante de este estudio ha sido la ventaja de utilizar las estadísticas descriptivas de la métrica MDV para la detección de texturas. La utilización de estas estadísticas, como el mínimo, máximo, promedio y varianza, junto con el tamaño del bloque, ha demostrado ser suficiente para obtener buenos resultados en todos los algoritmos evaluados. Esto permite obtener resultados prometedores utilizando características más simples y con menor costo computacional.

En cuanto a futuras investigaciones, se sugieren varias vías de estudio. En primer lugar, sería interesante realizar una comparativa con otros métodos de clasificación o detección de textura existentes en la literatura, lo que fortalecería los resultados obtenidos en este estudio. Asimismo, se podría aumentar el tamaño del dataset, ya que las 2000 muestras distribuidas entre los diferentes tamaños y tipos de bloques pueden no representar completamente la diversidad de bloques existentes, considerando incluso dimensiones no cuadradas.

Además, se podrían explorar otras métricas descriptivas o características adicionales para mejorar la capacidad de detección de textu-

ras en los bloques de codificación. Por ejemplo, considerar características relacionadas con la distribución espacial de los píxeles o utilizar técnicas más avanzadas de extracción de características podrían brindar información adicional y mejorar la precisión de la clasificación.

Agradecimientos

Esta publicación es parte del proyecto PID2021-123627OB-C55, financiado por MCIN/AEI/10.13039/501100011033/FEDER, UE, y por la Conselleria de Innovación, Universidades, Ciencia y Sociedad Digital de la Comunidad Valenciana con referencia CIAICO/2021/278.

Referencias

[1] Debarati Kundu and Brian L. Evans, "Full-reference visual quality assessment for synthetic images: A subjective study," in *2015 IEEE International Conference on Image Processing (ICIP)*, 2015, pp. 2374–2378.

[2] University of Southern California, Signal and Image Processing Institute, "The USC-SIPI Image Database," .

[3] Nicola Asuni and Andrea Giachetti, "TESTIMAGES: a Large-scale Archive for Testing Visual Devices and Basic Image Processing Algorithms," in *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, Andrea Giachetti, Ed. 2014, The Eurographics Association.

[4] Kodak, "The Kodak Color Image Dataset," .

[5] H.H.Y. Tong and A.N. Venetsanopoulos, "A perceptual model for JPEG applications based on block classification, texture masking, and luminance masking," in *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269)*, Oct 1998, pp. 428–432 vol.3.

[6] X.H. Zhang, W.S. Lin, and P. Xue, "Improved estimation for just-noticeable visual distortion," *Signal Processing*, vol. 85, no. 4, pp. 795 – 808, 2005.

[7] Tao Xiang, Shangwei Guo, and Xiaoguo Li, "Perceptual Visual Security Index Based on Edge and Texture Similarities," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 5, pp. 951–963, 2016.

[8] Dina A. Abdulqader, Mohammed Sadoon Hathal, Basheera M. Mahmmod, Sadiq H. Abdulhussain, and Dhiya Al-Jumeily, "Plain, Edge, and Texture Detection Based on Orthogonal Moment," *IEEE Access*, vol. 10, pp. 114455–114468, 2022.

[9] Chuanmin Jia, Shiqi Wang, Xinfeng Zhang, Shanshe Wang, Jiaying Liu, Shiliang Pu, and Siwei Ma, "Content-Aware Convolutional Neural Network for In-Loop Filtering in High Efficiency Video Coding," *IEEE Transactions on Image Processing*, vol. 28, no. 7, pp. 3343–3356, 2019.

[10] Damián Ruiz-Coll, Gerardo Fernández-Escribano, José Luis Martínez, and Pedro Cuenca, "Fast intra mode decision algorithm based on texture orientation detection in HEVC," *Signal Processing: Image Communication*, vol. 44, pp. 12–28, Mar 2016.

[11] Elena Georgiana Paraschiv, Damián Ruiz-Coll Maria Pantoja, and Gerardo Fernández-Escribano, "Parallelization and improvement of the MDV-SW algorithm for HEVC intra-prediction coding," *The Journal of Supercomputing*, vol. 75, pp. 1150–1162, Mar 2019.

Performance Evaluation of Dragonfly Topology Variants

Javier Navaridas, Jose A. Pascual [1]

Resumen— **Dragonfly is becoming one of the networks of choice for high-performance computer systems as it is designed with current technology trends and offers a sweet spot in terms of cost, simplicity, performance, fault-tolerance and power. In a Dragonfly topology, compute nodes are connected to switches forming groups in such a way that all switches of a group have a direct connection with each other. Then each of the groups have a connection to every other group, resulting in a clique of cliques. This provides a very high connectivity with a very low diameter which are desired characteristics of high performance interconnection networks. However, the way in which these groups is connected, namely the *connection rule*, has not been studied in detail. In this paper, we compare several connection rules under different architectural and workload configurations. We found that in some cases, the connection rule has limited impact, but in other cases it can have a massive impact by providing up to 4 times more throughput. We also analyze the effects of employing Valiant routing which in worst-case scenario can provide orders of magnitude more throughput than the basic dimensional routing. Finally, we measure the effects of a simple congestion control. We found it is never counterproductive, although in many cases the throughput improvement is negligible. However, in some extreme cases, the through improvement can be close to 200 %.**

Palabras clave— **Interconnection networks, Dragonfly topology, Connection rules, Routing algorithms, Performance evaluation, Simulation and modeling.**

I. Introduction

High performance computing systems and datacenters are highly valuable resources which are continuously growing in importance as they form the backbone of large parts of our day-to-day thanks to the advent of cloud computing, but are also pivotal both in business and science contexts. Most current scientific studies rely on modeling and analyzing different natural phenomena and/or technological processes, which often require a huge amount of computing power impossible to attain using regular off-the-shelf computers. For instance, physicists, chemists or pharmaceutical researchers simulate, for different purposes, interactions between huge amounts of molecules. Likewise, in the business context, corporations demand large amounts of computing power in order to use data mining software over large databases, with the objective of extracting knowledge from raw data, and to use that knowledge to their advantage. Obtaining patterns of consumer habits, boosting sales, optimizing costs and profits, estimating stocks or detecting fraudulent behavior are just a few interesting application domains.

[1]Dpto. de Arquitectura y Tecnología de Computadores, Universidad del País Vasco — UPV/EHU

In these large scale computing systems, the interconnection network is a critical component as its performance, fault-tolerance and scalability will limit those of the full system. Most parallel applications will stall if traffic cannot flow between compute nodes rapidly enough and with sufficient data density. For this reason, research on various aspects of interconnection networks is essential to keep growing the availability of global computing resources.

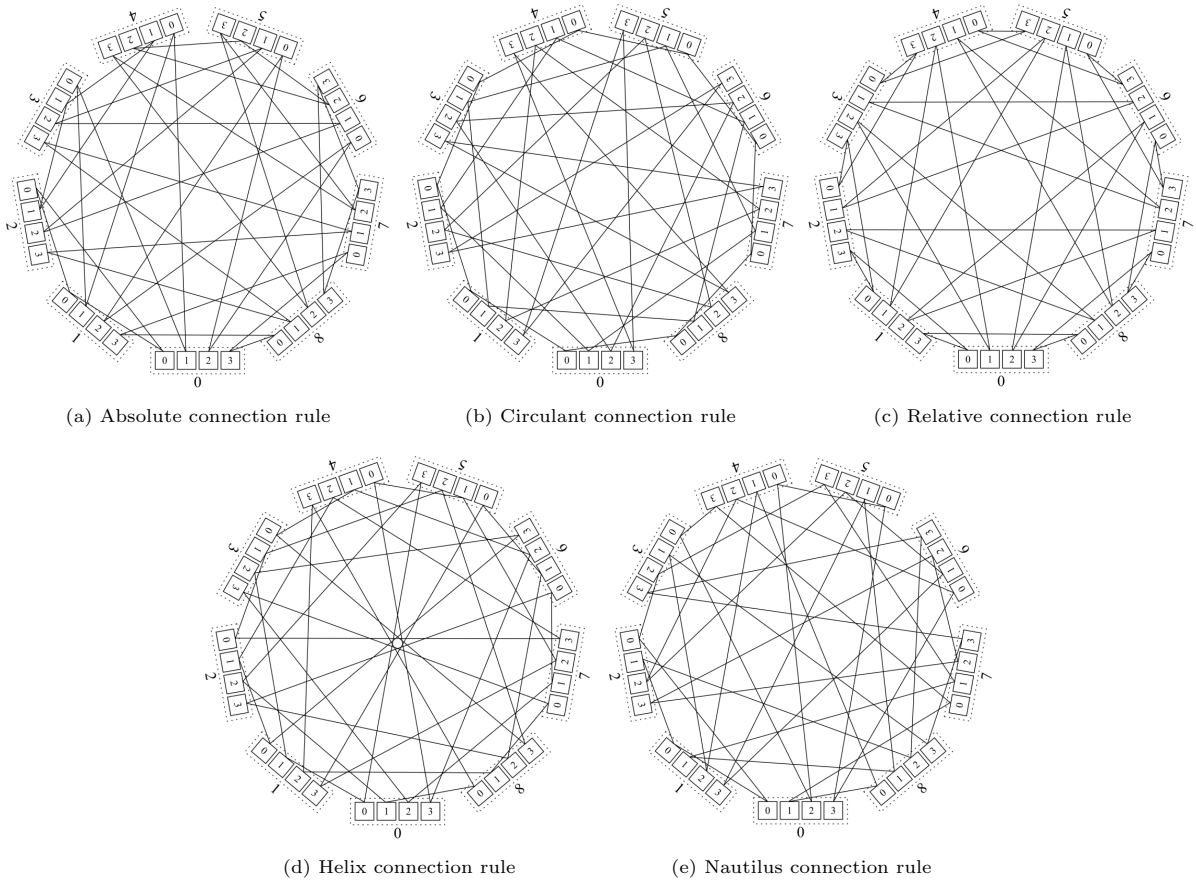
In this regards, we must highlight the dragonfly topology [1] as one of the most promising network architectures in modern data centers and high performance computing systems. Several systems nowadays are based on variants of dragonfly topologies, such as those which rely on Slingshot [2], or many that are built around Infiniband hardware [3], [4]. Dragonfly is a direct, hierarchical topology where endpoints are connected to switches which are organized in fully-interconnected groups. Then, these groups are connected in a fully interconnected manner, in such a way that each group has an inter-group link to every other group.

The Dragonfly topology is defined by three parameters:

1. **p**: the number of ports in each switch which are connected to endpoints.
2. **a**: the number of switches in each group. Therefore **a**-1 ports are devoted to intra-group connections.
3. **h**: the number of ports in each switch that are connected to other groups

However, these parameters only define the number of components, not how they are connected together. To completely defined the network, it is also necessary to define a connection rule which defines how the inter-group links are connected together. Indeed, the very first implementation of the Dragonfly topology [1] did never formally define any connection rule. Connection rules were first introduced by Camarero et al. which defined three such connection rules [5]. Later on, Belka et al. proposed another two variants [6]. Research on Dragonfly systems often does not specify the connection rule but, typically, uses either the relative or absolute arrangement.

Our contribution is to evaluate these five connection rules against each other which, to our knowledge, have never been compared before. In particular, we compare the maximum throughput achieved by three networks of different sizes, ranging from 1K to 16K nodes. Furthermore we look into several router mechanisms, such as the routing algorithm and the effectiveness of congestion control.



(a) Absolute connection rule (b) Circulant connection rule (c) Relative connection rule

(d) Helix connection rule (e) Nautilus connection rule

Fig. 1: Various connection rules for a Dragonfly topology with parameters (p=2, a=4, h=2). (top) Original implementations proposed by Camarero et al. (bottom) Later variants proposed by Belka et al. Boxes represent switches. Dotted rectangles delimit groups. Lines represents inter-group links. Intra-group links and endpoints are not shown for the sake of clarity.

Our results show that, with balanced traffic, the connection rule has little influence on the overall results, as does the activation or not of congestion control. The routing scheme, however, can have a significant impact as the use of Valiant can reduce throughput by up to 30 %. In unbalanced traffic scenarios, the differences between connection rules become more noticeable, with more unbalanced workloads generally having higher impact. In the worst cases, throughput variations can be up to 4-fold. Moreover, in these scenarios congestion control shows its effectiveness with throughput improvements of up to 2.5x. Finally, as is well known within the community, the utilization of Valiant routing can provide *orders of magnitude* higher throughput by exploiting path diversity in pathological cases where all traffic from a group is routed through the same link when using the canonical dimensional routing.

II. Background

Dragonfly [1] is a direct topology which is becoming the go to topology for large-scale systems due to its performance and efficiency. Furthermore, since Top-of-Rack switches are connected directly among themselves, it does not require extra racks to host the core network, as happens with tree-like [7], [8], [9] or Clos-based [10] topologies. One of the limitations of the Dragonfly topology is that it is full of cycles which could be problematic if not dealt with appropriately. However, Dragonfly comes with a specific HPC deadlock-avoidance mechanism [1] or can rely on general-purpose mechanisms [11].

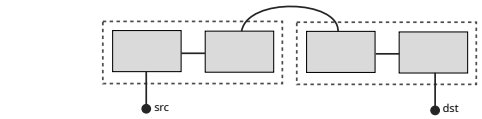
The basic routing scheme for Dragonfly is known as dimensional routing, where a packet is routed within a group to the switch connected to the destination group. And once it arrives to the destination group, it is routed within that group to the destination switch. Using dimensional-routing the longest path is 5 hops: 2 switch-to-node links, 2 intra-group links and 1 inter-group link (see Fig. 2). However, dimensional routing is known to be non-minimal [12] and, even worse, is not capable to deal effectively with unbalanced workloads as inter-group links tend to become a bottleneck. For this reason, it is very common to rely on proxy routing [1], as previously introduced by Valiant [13].

The idea behind proxy routing is to avoid these bottlenecks by distributing traffic throughout the whole network in a way that allows to maximize network utilization. Valiant routing randomly selects a proxy group and sends the packet through that group. Therefore, it has a longest path of 7 hops: 2 switch-to-node links, 3 intra-group links (one within each of the local, the proxy and the destination groups) and 2 inter-group links (from the local to the proxy and from the proxy to the destination).

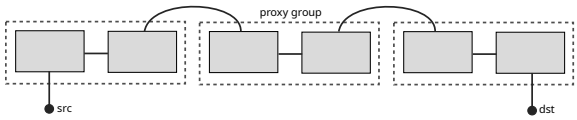
Apart from the canonical routing algorithms, dimensional and Valiant, there exists a body of re-

Tabla I: Dragonfly configurations considered in our study.

Dragonfly	p	a	h	endpoints	groups	switches	switch radix	total links
Small	4	8	4	1,056	33	264	15	2,508
Medium	6	12	6	5,256	73	876	23	12,702
Large	8	16	8	16,512	129	2064	31	40,248



(a) Dimensional routing



(b) Valiant routing

Fig. 2: Worst-case paths in a Dragonfly. *src* and *dst* represent the source and destination servers. Grey boxes represent switches and are assorted together in groups highlighted by the dotted areas. For simplicity only the 2 switches per group that belong to a path are shown.

search on routing techniques for Dragonfly systems. For instance, it was found that the original Valiant algorithm may generate hotspots on intra-group links [14]. This discovery led to the development of routing schemes that use non-minimal paths also within a group [14], [15]. Despite these issues, Valiant continues being the main building block for adaptive routing algorithms [16].

At any rate, when the topology was originally introduced, it did not define any specific rule to connect groups with each other, which is an important part of the network design, as we will see with our analysis later on. Camarero et al. [5] first introduced three simple connection rules: relative, absolute, and circulant, shown in Figure 1. In their analysis, however, they did not found any substantial differences between them. Later, Hastings et al. performed a bisection bandwidth analysis and found significant differences of up to 50 % [17]. Kaplan et al. deepened the understanding of these connection rules by means of a simulation-based evaluation. They found small performance differences in many cases, but some differences as high as 44 % [18].

Later, Belka et al. [6] introduced the helix and nautilus connection rules in an attempt to improve bisection bandwidth. These two variants are shown in the lower row of Figure 1. To our knowledge, there is no further analysis comparing all these connection rules together, so this paper aims at filling this gap by carrying out an independent analysis of all these Dragonfly variants together with a baseline network which defines connections among groups through random permutations.

III. Experimental Set-up

In this section we present the experimental environment used to analyze the performance of the different Dragonfly connection rules. Experiments have been carried out with our in-house developed simulator, INSEE [19]. First we describe the architecture of the system and the different traffic generation models. In addition, we discuss the metrics of interest we focus on in the experiments that have been performed.

Tabla II: Network configuration used in our evaluation.

Parameter	Value
Number of VCs	3
VC allocation	dally (from [1])
Port arbitration	random
Transit queue length	4 packets
Packet length	8 phits
Phit length	64 bits
Link bandwidth	10Gbps
Number of injectors	1
Injection queue length	4 packets

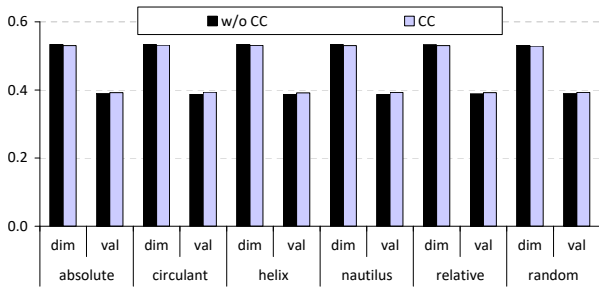
A. Simulation environment

The evaluation has been carried out using INSEE [19], a widely used and tested interconnection network simulator. INSEE uses a highly detailed model of the router at a phit-level and supports a wide variety of traffic models, including synthetic patterns, traces extracted from real applications [20] or synthetically generated traces [21]. In addition, it also implements many different topology/routing/architecture combinations.

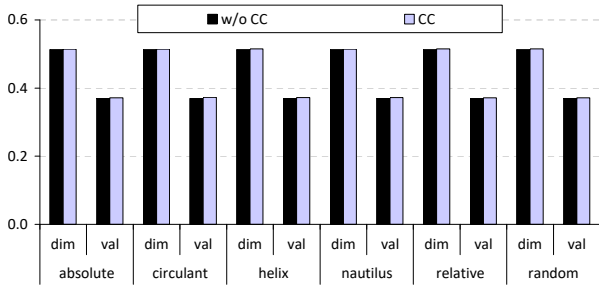
We carried out experiments with three topologies of increasing size, from 1K to 16K nodes, shown in Table I together with their main topological characteristics. As discussed before, we consider 5 different connection rules: absolute, relative and circulant (from [5]) as well as helix and nautilus (from [6]). In addition, as a baseline, we add a sixth configuration where inter-group connections are decided following a random permutation.

In each set of experiments we focus on raw performance by measuring the maximum throughput sustained by each configuration. In particular, our experiments use the following synthetic traffic patterns:

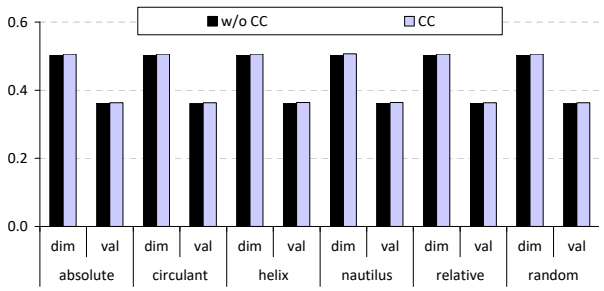
- **Uniform**: This is the traditional pattern in which packet destinations are selected uniformly at random. This generates a balanced distribution of traffic and serves to understand the performance under optimal traffic conditions.
- **Hot region**: Another traditional pattern where destinations are selected randomly, but a gi-



(a) Small system – Dragonfly (p=4,a=8,h=4)

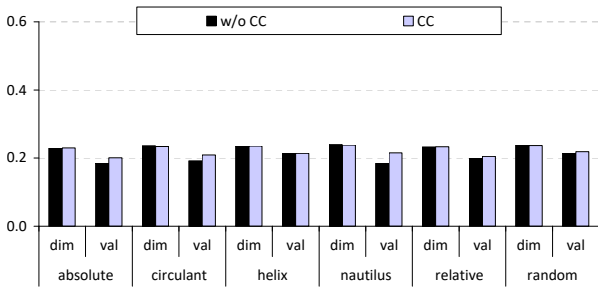


(b) Medium system – Dragonfly (p=6,a=12,h=6)

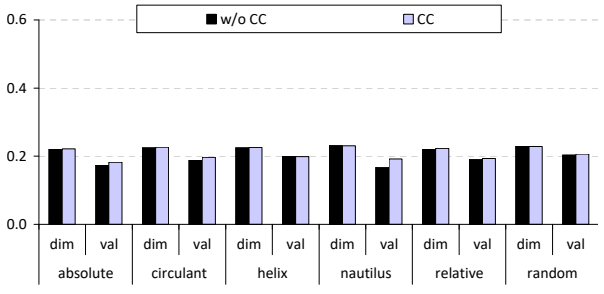


(c) Large system – Dragonfly (p=8,a=16,h=8)

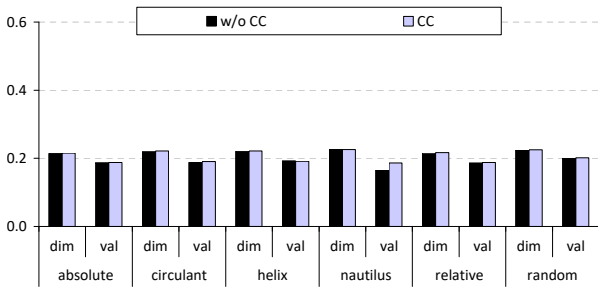
Fig. 3: Maximum throughput – random traffic.



(a) Small system – Dragonfly (p=4,a=8,h=4)



(b) Medium system – Dragonfly (p=6,a=12,h=6)



(c) Large system – Dragonfly (p=8,a=16,h=8)

Fig. 4: Maximum throughput – hot region traffic.

ven group of nodes, the hot region, has a higher probability of being selected as destination, increasing the risk of generating localized congestion. This generates a non-uniform distribution of traffic and serves to understand the performance under unbalanced traffic distributions. In our implementation 25 % of the traffic goes to a 12,5 % of the network.

- **Adversarial**: This is a pathological traffic pattern specifically designed for stressing Dragonfly fabrics. In Adversarial, all the nodes in a group try to use the same output port by sending traffic randomly to the nodes in the next group.
- **Permutation**: Given a source node, the destination node is always the same, and is computed as a permutation of the source node identifier. In this case, we use a shift permutation in a way that the target is a different group. Generating a traffic distribution akin to Adversarial, but without destination contention.

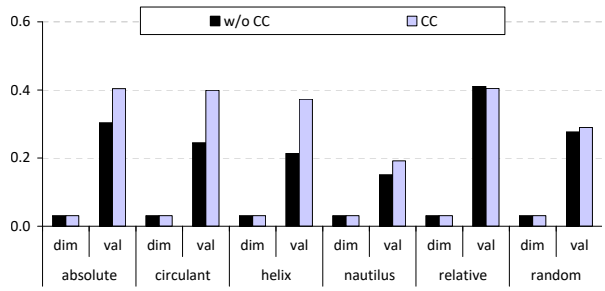
We consider a simple switch architecture with queues located in the input ports. The specific configuration that we use in our experiments is presented in Table II. In addition to these router and endpoint parameters, we consider systems using both Dimensional (dim) and Valiant (val) routing schemes in order to ascertain the effect that the routing can have on performance. Finally, we consider a simple congestion control mechanism (CC) which limits traffic rate by forcing the ports connected to the endpoints to be blocked upon contention with traffic from other ports. This should help to reduce the injection rate before reaching saturation and, therefore, to improve overall performance.

IV. Performance evaluation

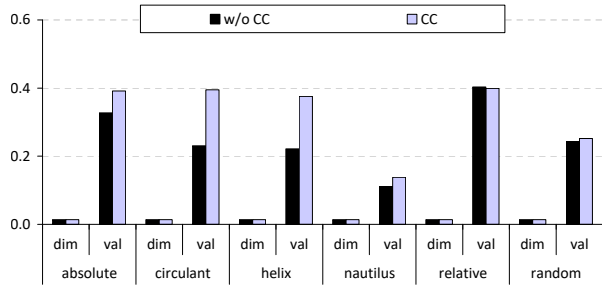
In this section we present and discuss the results of our experimental work. For simplicity we have organized our discussion based on the traffic model employed, in a way that the unbalance increases from each traffic model to the next.

A. Uniform traffic

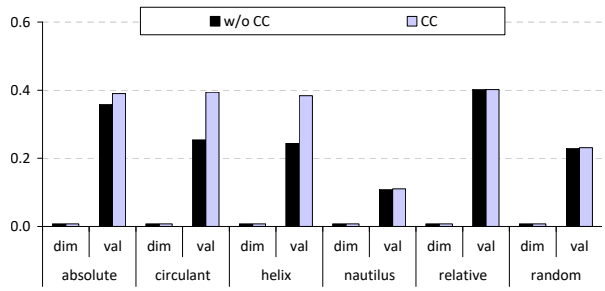
We start our discussion by analyzing the results with Uniform traffic, shown in Figure 3. Uniform traffic is well balanced and should not generate any long-lasting bottleneck. The first thing that we can see is that neither the connection rule nor the utilization of a congestion control mechanism have a visible effect on performance. All connection rules can sustain a similar throughput and congestion control provides a barely perceptible increase in throughput (be-



(a) Small system – Dragonfly (p=4,a=8,h=4)

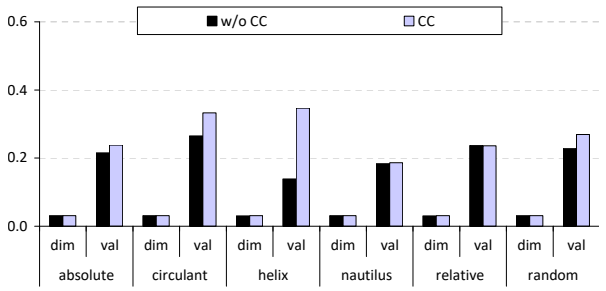


(b) Medium system – Dragonfly (p=6,a=12,h=6)

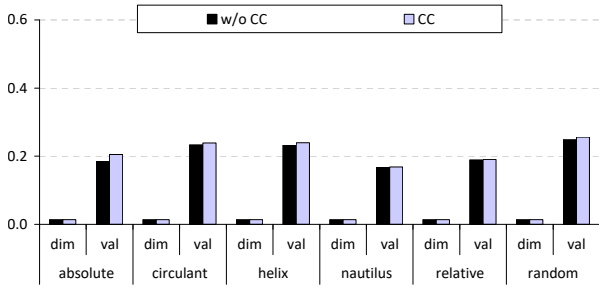


(c) Large system – Dragonfly (p=8,a=16,h=8)

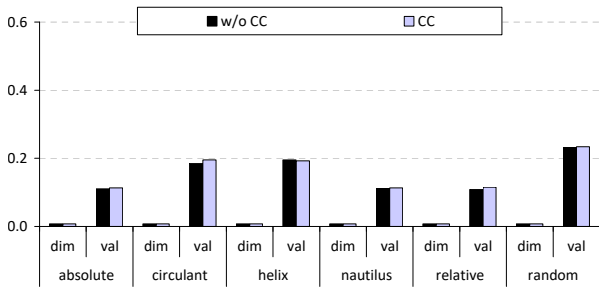
Fig. 5: Maximum throughput – adversarial traffic.



(a) Small system – Dragonfly (p=4,a=8,h=4)



(b) Medium system – Dragonfly (p=6,a=12,h=6)



(c) Large system – Dragonfly (p=8,a=16,h=8)

Fig. 6: Maximum throughput – permutation traffic.

low 1 %). This was somewhat expected because the balanced use of resources means no sustained congestion ever arises. Network size has a slightly higher impact, but still not very significant. From Small to Medium, the throughput is reduced by about 4 %, which is further reduced by another 2 % when moving to Large. Finally, in all cases, relying on Valiant has a substantially detrimental effect (approx. 30 % less throughput) when compared with Dimensional. This is reasonable since Uniform does not generate bottlenecks in the network that can be avoided by distributing traffic by means of proxy routing. However, with Valiant, packets travel a longer distance, so a higher bandwidth consumption is required per packet, which eventually leads to premature saturation of network resources.

B. Hot region traffic

We move now to analyze the results with Hot region traffic, shown in Figure 4. This traffic forces some unbalanced use of resources as portions of the topologies need to handle higher network pressure so they become a bottleneck. This portion corresponds to approximately 4, 9 and 16 groups, respectively, for each of the 3 size configurations.

The first thing that we notice is that with Hot region, the throughout is roughly halved, when compared with Uniform. This is reasonable, since most of the traffic is directed to a small portion of the network, which generates congestion around the hot region which ultimately spreads through the whole network which, in turn, reduces the amount of traffic traveling through other parts of the network as well. This results in reduced traffic flow. We can also start to find now differences between the connection rules, with up to 4 % different between the best and worst performing variants. It is interesting to note that Nautilus, which sustains the most throughput with dimensional routing, is the worst performing with Valiant routing. Also, congestion control is slightly more effective that in the previous case, with a few cases where it can improve throughput by around 10 %. Finally, Valiant routing is still worse than dimensional, but the difference is now much smaller; between 10 and 15 %.

C. Adversarial traffic

Let's discuss now the results with Adversarial traffic, depicted in Figure 5. This traffic forces a highly unbalanced use of resources because all the endpoints in each group are randomly sending traffic to endpoints in the next group. This means that, for dimen-

sional routing, the *unique* link between each group and the next becomes the bottleneck. For this reason, we can see that the load that can be accepted when using Dimensional routing is minuscule, practically tending to zero for the Large configuration, where the 128 endpoints in a group are sharing a single link. Because of this bottleneck, the congestion control mechanism has no effect in this case.

Because of the above, we will focus the discussion on the results with Valiant routing. In this case, we can appreciate substantial differences between connection rules. For instance, nautilus, the worst performing one, can support only between a half and a quarter of the throughput than the other topologies. Of special interest is relative, which is not only the connection rule able to support the most throughput, but it is also the only one that does not require from congestion control in order to achieve this maximum, as it seems to be barely affected by it. This interesting aspect requires of further investigation, but is outside of the scope of this paper. The rest of the connection rules get some benefit from using congestion control, in some cases, like helix and circulant, to a large degree; up to a 75 % more throughput when it is activated.

D. Permutation traffic

We finally analyze the results obtained with Permutation traffic, collected in Figure 6. This traffic enforces an even higher unbalanced use of resources because, now, all the endpoints in each group are sending traffic to different endpoints in the same target group. This means that there is no destination contention which means that the inter-group links need to handle all the traffic and so, become the bottleneck. Again, we can see that dimensional traffic is severely penalized with this traffic model because the link to the target group has to be shared by all endpoints in the local group. Because of that, the obtained throughput is identical as in the previous case and, again, we will focus more on the results with Valiant routing.

Similar to the previous case, we can see considerable differences between connection rules, although not to the same extent as before. It is of particular interest that using a random permutation for setting the connections achieves the best performance for the Medium and Large networks. This suggests that the connection rules that have been introduced until now are not very well suited for this kind of traffic, which opens the door to investigate novel connection rules that are more appropriate for this use case. With regards the congestion control mechanism, it seems to be rather effective in the Small network, but this effectiveness dilutes as we move to larger sizes. Intriguingly, with the Small helix network, the throughput is increased by 2.5× by activating congestion control. This would require further investigation, but since it does not happen with larger networks, we consider this occurrence as an outlier.

V. Conclusions and future work

This paper has compared, for the first time, the five connection rules that have been proposed for the Dragonfly topology. Our study has considered networks of different sizes, with up to tens of thousands of nodes. In addition, we investigate the effect of the routing algorithm as well as the suitability of an injection-limiting congestion control mechanism.

Our results show that, with balanced traffic, networks constructed with all the connection rules offer similar performance, with negligible differences. Furthermore, in these cases congestion control is not very useful but, importantly, it is not harmful either. Routing, on the other hand, has a more significant influence on performance, since Valiant reduces throughput by up to 30 %.

As we moved to increasingly more unbalanced traffic models, the differences between connection rules became more noticeable, with adversarial demonstrating the largest differences between them, with disparities of up to 4 times. Not only that, but congestion control is at its peak with Adversarial traffic, showing throughput increases of up to 75 %. With respect to routing we found that Dimensional routing is orders of magnitude less performant than Valiant.

As future work, we plan to further investigate the two special cases found here: Adversarial traffic with Relative connection rule and Permutation traffic with Random connections, which may lead to develop new connection rules. In addition, we will work on designing new routing algorithms that can improve upon the classic Dimensional and Valiant, as well as adaptive versions that are capable to adjust themselves based on the status of the network, preferably relying only on local information.

Acknowledgments

Authors want to thank David P. Bunde for his support with ensuring that our implementations of Helix and Nautilus were consistent with their design. This work is supported by the Basque Government (projects ELKARTEK21/89 and IT1504-22) and by the Spanish Ministry of Economy and Competitiveness MINECO (PID2019-104966GB-I00). Dr. Javier Navaridas is a Ramón y Cajal fellow from the Spanish Ministry of Science, Innovation and Universities (RYC2018-024829-I).

Referencias

[1] John Kim, William J. Dally, Steve Scott, and Dennis Abts, "Technology-driven, highly-scalable dragonfly topology," in *Procs. of the 35th Annual Intl. Symposium on Computer Architecture*, 2008, ISCA '08, pp. 77–88.

[2] Daniele De Sensi, Salvatore Di Girolamo, Kim H. McMahon, Duncan Roweth, and Torsten Hoefler, "An in-depth analysis of the slingshot interconnect," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–14.

[3] German Maglione-Mathey, Jesus Escudero-Sahuquillo, Pedro Javier Garcia, Francisco J. Quiles, and Eitan Zahavi, "Leveraging infiniband controller to configure deadlock-free routing engines for dragonflies," *Journal of Parallel and Distributed Computing*, vol. 147, pp. 16–33, 2021.

[4] Neil McGlohon, Robert B. Ross, and Christopher D. Carothers, "Evaluation of link failure resilience in multirail dragonfly-class networks through simulation," in *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, New York, NY, USA, 2020, SIGSIM-PADS '20, p. 105–116.

[5] Cristóbal Camarero, Enrique Vallejo, and Ramón Beivide, "Topological characterization of hamming and dragonfly networks and its implications on routing," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, 2014.

[6] Madison Belka, Myra Doubet, Sofia Meyers, Rose L. Momoh, David Rincon-Cruz, and David P. Bunde, "New link arrangements for dragonfly networks," *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, pp. 17–24, 2017.

[7] F. Petrini and M. Vanneschi, "k-ary n-trees: high performance networks for massively parallel architectures," in *Proceedings 11th International Parallel Processing Symposium*, 1997, pp. 87–93.

[8] Javier Navaridas, Jose Miguel-Alonso, Francisco Javier Ridruejo, and Wolfgang Denzel, "Reducing complexity in tree-like computer interconnection networks," *Parallel Computing*, vol. 36, no. 2, pp. 71–85, 2010.

[9] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta, "Vl2: A scalable and flexible data center network," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.

[10] Cristóbal Camarero, Carmen Martínez, and Ramón Beivide, "Random folded clos topologies for datacenter networks," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 193–204.

[11] Jose A. Pascual and Javier Navaridas, "High-performance, low-complexity deadlock avoidance for arbitrary topologies/routings," in *Proceedings of the 2018 International Conference on Supercomputing*, New York, NY, USA, 2018, ICS '18, p. 129–138.

[12] Ryland Curtsinger and David Bunde, "Shortest paths in dragonfly systems," in *2019 International Workshop of High-Perfomance Interconnection Networks in the Exascale and Big-Data Era (HiPNEB)*, 2019, pp. 1–8.

[13] Leslie G. Valiant, "A scheme for fast parallel communication," *SIAM Journal on Computing*, vol. 11, no. 2, pp. 350–361, 1982.

[14] Marina García, Enrique Vallejo, Ramón Beivide, Miguel Odriozola, Cristóbal Camarero, Mateo Valero, Germ'n Rodríguez, Jesús Labarta, and Cyriel Minkenberg, "On-the-fly adaptive routing in high-radix hierarchical networks," in *2012 41st International Conference on Parallel Processing*, 2012, pp. 279–288.

[15] Marina García, Enrique Vallejo, Ramón Beivide, Miguel Odriozola, and Mateo Valero, "Efficient routing mechanisms for dragonfly networks," in *2013 42nd International Conference on Parallel Processing*, 2013, pp. 582–592.

[16] Pablo Fuentes, Enrique Vallejo, Marina García, Ramón Beivide, Germán Rodríguez, Cyriel Minkenberg, and Mateo Valero, "Contention-based nonminimal adaptive routing in high-radix networks," in *2015 IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 103–112.

[17] Emily M. Hastings, David Rincon-Cruz, Marc Spehlmann, Sofia Meyers, Anda Xu, David P. Bunde, and Vitus J. Leung, "Comparing global link arrangements for dragonfly networks," *2015 IEEE International Conference on Cluster Computing*, pp. 361–370, 2015.

[18] Fulya Kaplan, Ozan Tuncer, Vitus J. Leung, Scott K. Hemmert, and Ayse K. Coskun, "Unveiling the interplay between global link arrangements and network management algorithms on dragonfly networks," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 325–334.

[19] Javier Navaridas, Jose Miguel-Alonso, Jose A Pascual, and Francisco J Ridruejo, "Simulating and evaluating interconnection networks with insee," *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 494–515, 2011.

[20] José Miguel-Alonso, Javier Navaridas, and Francisco Javier Ridruejo Perez, "Interconnection network simulation using traces of mpi applications," *International Journal of Parallel Programming*, vol. 37, pp. 153–174, 2009.

[21] Javier Navaridas, José Miguel-Alonso, and Francisco Javier Ridruejo Perez, "On synthesizing workloads emulating mpi applications," *2008 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–8, 2008.

Arquitecturas del procesaddor, multiprocesadores y chips multinúcleo

NATSA: A Near-Data Processing Accelerator for Time Series Analysis

Ivan Fernandez[§] Ricardo Quislant[§] Christina Giannoula[†]
Mohammed Alser[‡] Juan Gómez-Luna[‡] Eladio Gutiérrez[§]
Oscar Plata[§] Onur Mutlu[‡]
[§]*University of Malaga* [†]*National Technical University of Athens* [‡]*ETH Zürich*

Abstract— **Time series analysis is a key technique for extracting and predicting events in domains as diverse as epidemiology, genomics, neuroscience, environmental sciences, economics, and more. *Matrix profile*, the state-of-the-art algorithm to perform time series analysis, computes the most similar subsequence for a given query subsequence within a sliced time series. *Matrix profile* has low arithmetic intensity, but it typically operates on large amounts of time series data. In current computing systems, this data needs to be moved between the off-chip memory units and the on-chip computation units for performing *matrix profile*. This causes a major performance bottleneck as data movement is extremely costly in terms of both execution time and energy.**

In this work, we present *NATSA*, the *first* Near-Data Processing accelerator for time series analysis. The key idea is to exploit modern 3D-stacked High Bandwidth Memory (HBM) to enable efficient and fast specialized *matrix profile* computation near memory, where time series data resides. NATSA provides three key benefits: 1) quickly computing the *matrix profile* for a wide range of applications by building specialized energy-efficient floating-point arithmetic processing units close to HBM, 2) improving the energy efficiency and execution time by reducing the need for data movement over slow and energy-hungry buses between the computation units and the memory units, and 3) analyzing time series data at scale by exploiting low-latency, high-bandwidth, and energy-efficient memory access provided by HBM. Our experimental evaluation shows that NATSA improves performance by up to 14.2× (9.9× on average) and reduces energy by up to 27.2× (19.4× on average), over the state-of-the-art multi-core implementation. NATSA also improves performance by 6.3× and reduces energy by 10.2× over a general-purpose NDP platform with 64 in-order cores.

I. Introduction

A time series is a chronologically ordered set of samples of a real-valued variable that can contain millions of observations. Time series analysis is used to analyze information in a wide variety of domains [1]: epidemiology, genomics, neuroscience, medicine, environmental sciences, economics, and more. Time series analysis includes finding similarities (*motifs* [2]) and anomalies (*discords* [3]) between every two subsequences (i.e., slices of consecutive data points) of the time series [4, 5]. There are two major approaches for motif and discord discovery: approximate and exact algorithms [6]. Approximate algorithms [2] are faster than exact algorithms, but they can provide inaccurate results or limited discord detection, which cannot be tolerated by many applications (e.g., vehicle safety systems [7]). Unlike approximate algorithms, exact algorithms [8] do not yield false positives or discordant dismissals, but can be very time-consuming on large time series data. Thus, *anytime* versions (aka interruptible algorithms) of exact algorithms are proposed to provide approximate solutions quickly [9, 10] and can return a valid result even if execution stops early.

The state-of-the-art exact *anytime* method for motif and discord discovery is *matrix profile* [9], which is based on Euclidean distances and floating-point arithmetic. Fig. 1 depicts a naive example of anomaly detection using *matrix profile*, where the sinusoidal signal has an anomaly between values 250 and 270. The *matrix profile* output of this time series shows low values for the periodic subsequences of it as they are very similar to the other subsequences, and higher values for the anomalies and their neighboring subsequences.

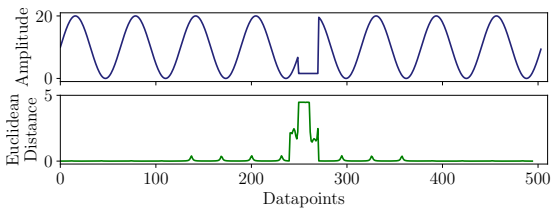


Fig. 1: A time series (upper figure) including anomalies and its *matrix profile* output (lower figure). Anomalies appear as higher Euclidean distance values in the profile.

We evaluate a recent CPU implementation of the *matrix profile* algorithm [10] on a real multi-core machine (Intel Xeon Phi KNL [11]) and observe that its performance is heavily bottlenecked by data movement. In other words, the amount of computation per data access is not enough to hide the memory latency and thus time series analysis is memory-bound. This overhead caused by data movement limits the potential benefits of acceleration efforts that do not alleviate data the movement bottleneck in current time series applications.

Several CPU and GPU implementations of *matrix profile* have been proposed in the literature [9, 10, 12, 13]. However, these acceleration efforts still require transferring the time series data from the main memory to the CPU/GPU cores, leading to the data movement bottleneck. Near-Data Processing (NDP) [14–19] is a promising approach to alleviate data movement by placing processing units close to memory. As a result, NDP solutions have the potential to improve system performance and energy efficiency when they are carefully designed with low-cost and low-overhead near data processing cores for memory-bound applications [20].

Our **goal** in this work is to enable high-performance and energy-efficient time series analysis for a wide range of applications, by minimizing the overheads of data movement. This can enable efficient time series analysis on large-scale sys-

tems as well as embedded and mobile devices, where power consumption is a critical constraint (e.g., heart beat analysis on a mobile medical device to predict a heart attack [21]). **To this end**, we propose *NATSA*, the *first* Near-Data Processing Accelerator for Time Series Analysis. The key idea of NATSA is to exploit modern 3D-stacked High Bandwidth Memory (HBM) [22, 23] along with specialized custom processing units in the logic layer of HBM, to enable energy-efficient and fast *matrix profile* computation near memory, where time series data resides. NATSA supports a wide range of time series applications thanks to *matrix profile*'s generality.

Our evaluation shows that NATSA provides up to 14.2× (9.9× on average) higher performance and up to 27.2× (19.4× on average) lower energy consumption compared to a state-of-the-art multi-core system. NATSA consumes 11.0× and 4.1× less energy over optimized implementations of *matrix profile* on an Intel Xeon Phi KNL [24] and NVIDIA GTX 1050 GPU [12], respectively. NATSA has 9.6× and 1.8× smaller area than these two accelerators, at equivalent performance points. NATSA outperforms a general-purpose NDP platform by 6.3× while consuming 10.2× less energy.

This work makes the following *contributions*:

- We propose *NATSA*, the first near-data processing accelerator for accelerating time series analysis using modern 3D-stacked High Bandwidth Memory (HBM).
- We propose a new workload partitioning scheme that preserves the *anytime* property of the algorithm, while providing load balancing among near-data processing units.
- We perform a detailed analysis of NATSA in terms of both performance and energy consumption. We compare different versions of NATSA (DDR4 [25] and HBM [22]) with four different architectures (8-core CPU, 64-core CPU, GPUs and NDP-CPU) and find that NATSA provides the highest performance and lowest energy consumption.

II. Background

A. Time Series Analysis: The matrix profile

A *time series* T is a sequence of n data points t_i, where $1 \leq i \leq n$, collected over time. A subsequence of T, also called a *window*, is denoted by $T_{i,m}$, where i is the index of the first data point, and m is the number of samples in the subsequence, with $1 \leq i$, and $m \leq n - i$.

The state-of-the-art exact *anytime* method for time series analysis is *matrix profile* [9]. When analyzing a time series, the *profile* is maintained as another time series that represents the most similar neighbor for a particular subsequence of the original time series. The similarity between two subsequences $T_{i,m}$ and $T_{j,m}$ can be calculated using the *z-normalized Euclidean distance*, which is defined as follows.

$$d_{i,j} = \sqrt{2m\left(1 - \frac{Q_{i,j} - m\mu_i\mu_j}{m\sigma_i\sigma_j}\right)} \qquad (1)$$

where $Q_{i,j}$ is the dot product of $T_{i,m}$ and $T_{j,m}$; μ_x and σ_x are the mean and the standard deviation of the points in $T_{x,m}$, respectively. These statistics are computed in $O(n)$ time [26].

Using the distance in Eq. 1, the *matrix profile* algorithm solves the similarity search problem in three steps. First, it builds a symmetric $(n - m + 1) \times (n - m + 1)$ matrix D, called *distance matrix*. Each cell in D, $d_{i,j}$, stores the distance between two subsequences, $T_{i,m}$ and $T_{j,m}$. Second, it creates an array P of size $n - m + 1$, called *profile*. Each cell P_i in P keeps the minimum distance recorded in the i^{th} row of D. Third, it allocates an array I that is of the same size as P, called *profile index*, such that $I_i = j$ if $P_i = d_{i,j}$. This way, P contains the minimum distances between subsequences, while I is the vector of "pointers" to the location of these subsequences within the time series.

Fig. 2 depicts an example of the distance matrix D, the profile P, and the profile index I. The neighboring subsequences of $T_{i,m}$ are highly similar to it (i.e., $d_{i,i+1} \approx 0$) due to overlapping between them. The algorithm excludes these subsequences from the computation to avoid false positives, by defining an exclusion zone for each subsequence. It follows the approach in [10], where the exclusion zone of $T_{i,m}$ is $T_{i,\frac{m}{4}}$ (i.e., ends at $t_{i+\frac{m}{4}}$ of the time series).

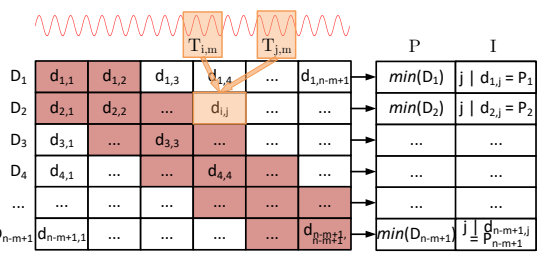


Fig. 2: Example of distance matrix (D), profile (P), and profile index (I). P_i holds the minimum distance calculated in row D_i, and I_i holds the index j of the subsequence that results in that distance. The cells in the exclusion zone are coloured red.

B. The SCRIMP Implementation

The state-of-the-art CPU-based implementation of the *matrix profile* algorithm is SCRIMP [10]. We use an optimized version of SCRIMP [24] as baseline for our work, since it has the best convergence properties and takes advantage of multithreading and vectorization. The key mechanism behind optimized SCRIMP is that the dot product in Eq. 1 can be calculated incrementally in the diagonals of D as follows:

$$Q_{i,j} = Q_{i-1,j-1} - t_{i-1}t_{j-1} + t_{i+m-1}t_{j+m-1} \qquad (2)$$

According to Eq. 2, except for the first dot product, the remaining cells of a diagonal can be calculated using the values from the immediate upper left cells. This fact significantly reduces the number of multiplications and additions needed.

Algorithm 1, optimized SCRIMP [24], exploits both thread-level parallelism and vectorization. First, it precalculates the means and standard deviations of every subsequence of the time series (line 1), and initializes the profile vector (lines 3-4). Second, it computes the diagonals (see Fig. 2) using the loop in line 5. The variable `nDiag` is the number of diagonals of D assigned to each thread. These diagonals can be ordered in the `diag` vector (line 6) a) *randomly*, enabling the *anytime* property of the algorithm, or b) *sequentially*, discarding the *anytime*

property but allowing for optimizations [10] (e.g., exploiting data locality of consecutive diagonals).

Algorithm 1 Optimized SCRIMP [24]

```
1:  μ, σ ← precalculateMeansDevs(T, m);
2:  vectFact ← VECTOR_WIDTH/sizeof(datatype);
3:  for i ← 0 to size(P) − 1 do
4:      P_i ← ∞;
5:  for idx ← tid * nDiag to (tid + 1) * nDiag − 1 do
6:      i ← 0; j ← diag_idx;
7:      q ← dotProduct(T_{i,m}, T_{j,m});          ▷ Vectorized loop
8:      d ← dist(m, q, μ_i, σ_i, μ_j, σ_j);
9:      if d < P_i then P_i ← d; I_i ← j;
10:     if d < P_j then P_j ← d; I_j ← i;
11:     i ← i + 1;
12:     for j ← diag_idx + 1 to size(P) do
13:         for k ← 0 to vectFact − 1 do          ▷ Vectorized loop
14:             qs_k ← t_{i+m−1+k} t_{j+m−1+k} − t_{i−1+k} t_{j−1+k};
15:         qs_0 ← qs_0 + q;
16:         for k ← 1 to vectFact − 1 do
17:             qs_k ← qs_k + qs_{k−1};
18:         q ← qs_{vectFact−1};
19:         for k ← 0 to vectFact − 1 do          ▷ Vectorized loop
20:             ds_k ← dist(m, qs_k, μ_{i+k}, σ_{i+k}, μ_{j+k}, σ_{j+k});
21:             if ds_k < P_{i+k} then P_{i+k} ← ds_k; I_{i+k} ← j + k;
22:             if ds_k < P_{j+k} then P_{j+k} ← ds_k; I_{j+k} ← i + k;
23:         i ← i + vectFact;
```

Note that only P and I are allocated in memory, since storing D can lead to large memory consumption for large series due to the n^2 memory footprint (i.e., the values of D are calculated on the fly, updating P and I when needed). For each diagonal, the algorithm first computes the dot product of the first pair of subsequences in line 7 using the *dotProduct* function, which is vectorized. Second, it calculates the distance according to Eq. 1 (line 8). Third, it checks and replaces the corresponding profile element with the new distance provided that the calculated one is smaller (lines 9-10).

The algorithm addresses the imposed data dependency due to the dot product update between the elements in the diagonal with the following steps: 1) it pre-computes the add terms in Eq. 2 in batches of size *vectFactor* in a vectorized manner (lines 13-14); 2) it adds the previous dot product to the first new one (line 15); 3) it sequentially updates the remaining dot products in the batch (lines 16-17) saving the last one for the next iteration of the diagonal (line 18); 4) it computes the distance as well as the profile update in a vectorized way (lines 19-22). As a result, all loops are fully vectorized except the one in lines 16-17.

C. NDP and 3D-Stacked Memory

Near-Data Processing (NDP) [14–20, 27–48, 48–56] is a promising paradigm to reduce the data movement between CPUs and memory by placing simple general-purpose processors [20, 40, 45] or application-specific accelerators [16, 20, 36, 46, 47, 57] in or close to the logic layer of 3D-stacked memory. Generally, NDP can provide performance benefits for memory-bound applications when they exhibit one or more of the following major properties: 1) requiring higher memory bandwidth than available in the system, 2) being sensitive to memory access latency [58], or 3) performing irregular memory accesses, such that they cannot effectively benefit from cache hierarchy of conventional CPU architectures.

Recent advances in die-stacking technologies have enabled the integration of multiple layers of DRAM arrays in a single package. A 3D-stacked memory consists of several memory dies, one on top of each other, connected using Through-Silicon Vias (TSV) [22, 23]. NDP locates low-power processing units inside the logic layer of 3D-stacked memory, to harness the significantly higher bandwidth and the lower latency provided while consuming less energy. The most prominent 3D-stacked memory technologies are High Bandwidth Memory (HBM) [59] and Hybrid Memory Cube (HMC) [60], but there are several others [61, 62].

III. Motivation

NATSA is motivated by two key observations: First, time series motif and discord discovery are two of the most important analysis primitives for a wide variety of applications. Besides the applications mentioned in Section I, we can find these primitives applied to bioinformatics [63–65], speech processing [66], robotics [67], weather prediction [68], entomology [69], geophysics [70], finance [71], communication engineering [72], and electroencephalography [73].

Second, memory is the main bottleneck in time series analysis. We characterize the performance of a state-of-the-art CPU-based multithreaded and vectorized implementation of SCRIMP, developed in [24]. We run SCRIMP [24] on an Intel Xeon Phi 7210 processor, with 64 cores and 256 hardware threads, using two types of memory (DDR4 and HBM) available in this architecture. In Fig. 3, we present the performance results normalized to 1 thread (lines) and utilized memory bandwidth (bars) of SCRIMP. We observe that, when using DDR4, the performance of SCRIMP does not scale beyond 32 threads, whereas the higher memory bandwidth provided by HBM enables SCRIMP to scale up to 128 threads. This shows that SCRIMP's performance saturates on many-core architectures, because the achievable bandwidth saturates when the number of threads increases. To know the cause for this memory boundedness we perform the next experiment.

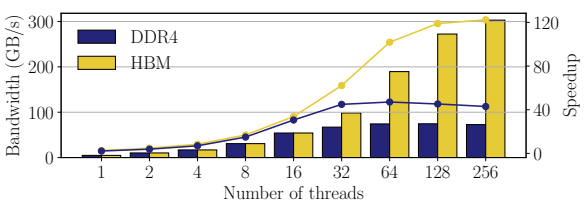


Fig. 3: Memory bandwidth usage (bars) and normalized performance (lines) of a parallel and vectorized version of SCRIMP [24] running on an Intel Xeon Phi 7210.

We perform the roofline analysis as we show in Fig. 4. We observe that the arithmetic intensity of SCRIMP is significantly low. The confirms that the memory boundedness of SCRIMP is due to the low arithmetic intensity of the algorithm, which leads processing cores to be underutilized. Based on all these observations, we conclude that the performance of the state-of-the-art CPU-based implementation of the *matrix profile*, SCRIMP [24], is heavily bottlenecked by available memory bandwidth and

data movement. Our goal is to reduce the data movement bottleneck of SCRIMP by building an NDP accelerator that matches the compute throughput of processing elements with the available memory bandwidth.

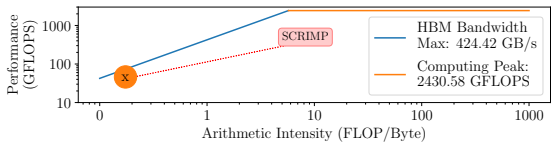


Fig. 4: Roofline analysis of a parallel and vectorized version of SCRIMP [24] running on an Intel Xeon Phi 7210.

IV. NATSA Architecture

Our Near-Data Processing Accelerator for Time Series Analysis, NATSA, is designed to 1) fully exploit the memory access parallelism and high memory bandwidth offered by HBM, and 2) employ the required amount of computing resources to provide a balanced solution. NATSA is built next to the HBM memory and exploits the full HBM bandwidth available. NATSA consists of multiple processing units (PUs) that efficiently compute the diagonals of *matrix profile* in a parallel fashion. The PUs are designed to compute diagonals using a vectorized approach to process a batch of elements of a diagonal at the same time. Each PU includes energy-efficient floating-point units [74], bitwise operators, and registers (See Table **??** in Sect. **??**). Each PU communicates with the HBM memory via a controller connected to one of the 8 memory channels provided by HBM.

A. NATSA Processing Units (PUs)

Each NATSA PU consists of four hardware components: the *Dot Product Unit* (DPU), the *Distance Compute Unit* (DCU), the *Profile Update Unit* (PUU), and the *Dot Product Update Unit* (DPUU), as we show in Fig. 5. We share the floating-point arithmetic operators (e.g., multipliers) among those hardware components to minimize idle cycles and enable reusability. The control unit (❶ in Fig. 5) is a state machine that orchestrates the execution flow of a PU. The multiplexers (❷ in Fig. 5) choose between the output of DPU and DPUU based on a signal from the control unit, so that the DCU can take advantage of Eq. 2, starting from the second element of the diagonal all the way down to the last. We replicate those hardware components to compute different elements of a diagonal in parallel, using the vectorized approach outlined in Section II-B. The diagonal assignment is pre-calculated in the host CPU, which sends the indices of the to-be-computed diagonals to each NATSA PU. Finally, each NATSA PU uses its own 1KB scratchpad memory to temporarily store fixed-size auxiliary data, such as the window size or configuration parameters.

The execution flow through the hardware components of a PU includes the following six steps:

1. **Dot product computation of the first element of the diagonal.** The DPU calculates the dot product between the first pair of subsequences of the diagonal ($T_{i,m}$ and $T_{j,m}$) by using the time series input T, and the window size, m, which is used to signal the end of each subsequence. This hardware component vectorizes the operation and outputs the result, $q_{i,j}$, for the next step.
2. **Euclidean distance computation of the first element of the diagonal**. The DCU computes the first Euclidean distance of each diagonal following Eq. 1, using the dot product computed by the DPU $q_{i,j}$. The values of μ and σ are precomputed by the host CPU in negligible time ($O(n)$ [26]) with respect to the total execution time. This simplifies the design of the PU.
3. **First profile update.** If the Euclidean distance calculated in the DCU, $d_{i,j}$, is lower than that stored in the profile for both subsequences, the PUU updates the profile vector and index vector, PP and II.
4. **Dot product update.** The dot product of the second and successive cells in the diagonal is calculated from the previous cell. It is computed in the DPUU by subtracting the first product and adding the new one to $q_{i,j}$, as shown in Eq. 2. This hardware component is replicated to enable vectorization and is pipelined with the DCU and the PUU.
5. **Second and successive Euclidean distance computations.** The DCU computes again the Euclidean distance, but now it obtains $q_{i,j}$ from the DPUU. The DPUU hardware component is replicated for vectorization of the dot product update calculations.
6. **Second and successive profile updates.** The PUU updates the profile vector and profile index vector, if needed. This hardware component is replicated to perform several updates at a time.

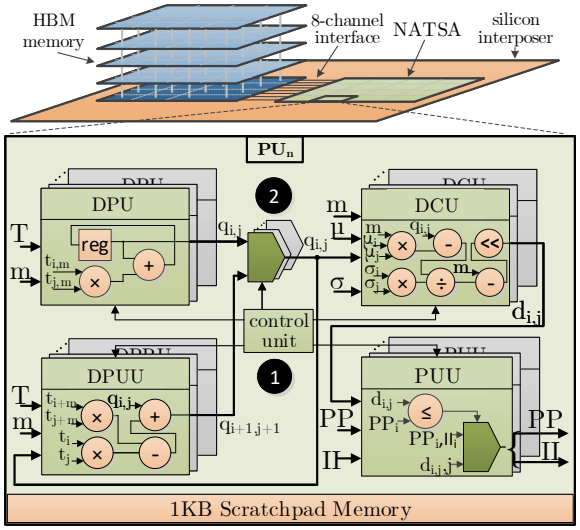


Fig. 5: NATSA design and integration next to HBM memory. NATSA is connected directly to the HBM interface.

B. Workload Partitioning Scheme

Computing the diagonals of the distance matrix may lead to load imbalance among the PUs, because those diagonals have different lengths. To avoid this imbalance, we propose a static partition scheduling

scheme which depends only on the size of the time series and the exclusion zone.

The way we tackle this problem is by assigning a set of pairs of diagonals to each NATSA PU such that the sum of their elements is equal to the number of cells of the main diagonal of the distance matrix minus the number of cells of the exclusion zone, $(n - m + 1) - m/4$.

Fig. 6 illustrates an example with two PUs, *PU0* and *PU1*, a distance matrix for a time series of $n = 13$ cells, a window size of $m = 4$, and an exclusion zone of 1 diagonal (crossed out rectangles). In this case, the number of elements that each pair of diagonals assigned to a PU should have is $(n - m + 1) - m/4 = 10 - 1 = 9$. Comparing a subsequence with itself gives zero distance value. As a consequence, the algorithm treats the main diagonal as exclusion zone and avoids computing it. The first diagonal of non-zero values, which starts in column D_2 and is represented with crossed out rectangles, belongs to the exclusion zone (see Fig. 2), so NATSA PUs also skip it.

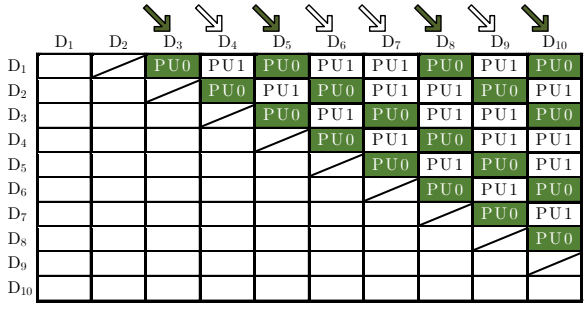


Fig. 6: Example of the diagonal scheduling scheme for two processing units, denoted as *PU0* (green) and *PU1* (white). Arrows show direction of computation.

Discarding the computation of the main diagonal and the diagonals in the exclusion zone, both PUs have to compute the diagonals from columns D_3 to D_{10}. To perform this efficiently and maintain the *anytime* property of SCRIMP, in the first step, *PU0* is assigned the first and last diagonal (9 elements in total), and *PU1* is assigned the second and the penultimate diagonal (totalling 9 elements as well). In the second step, *PU0* computes the third and the third-to-last diagonal, whereas *PU1* computes the fourth and fifth diagonals.

Our proposed scheduling scheme can be used in two ways: 1) *Randomly ordering* the indices of diagonals that each PU has to compute. Using this approach, we are able to preserve the *anytime* property of the algorithm, since if the execution is interrupted, the user obtains a partial exploration of the whole time series (i.e., events from any point of the time series can be detected). 2) *Sequentially ordering* the indices of diagonals that each PU has to compute. This approach violates the *anytime* property (i.e., only events up to the interruption point can be detected), but allows for further optimizations (e.g., exploiting data locality between consecutive diagonals).

Data mapping. Each PU has access to its corresponding portion of the time series and statistic vectors, and works with replicated profile and profile index vectors. This approach simplifies the overall architecture, enabling the use of many PUs without having to synchronize between them. NATSA assigns multiple diagonals to each PU with the specific scheduling scheme described in this section.

C. Programming Interface

In this section, we introduce the API to invoke NATSA from a host processor. While conventional loosely-coupled accelerators (e.g., GPUs or FPGAs) have their own memory, where data must be transferred to from the host's memory, NATSA is a tightly-integrated NDP accelerator, located between the host CPU and main memory. Thus, there is no need to transfer any data between the host memory and the accelerator memory, as loosely-coupled accelerators require. The user is responsible for 1) allocating the time series (T) and 2) providing the window length (m). NATSA will provide the user the profile vector (P) and profile index vector (I) in return. The size of the exclusion zone ($\frac{m}{4}$ by default) can be also passed as a parameter (exc).

Algorithm 2 outlines the `NATSA` API. First, `NATSA` function precalculates the statistics (μ, σ) (line 2) in the host CPU and allocates the private vectors (PP, II) to NATSA's PUs (line 3).

Algorithm 2 NATSA API

```
1: function P, I ← NATSA(T, m, exc, conf)
2:     μ, σ ← precalculateMeanDev(T, m)
3:     PP, II ← allocatePrivateProfiles(T, m, exc)
4:     idx ← diagonalScheduling(T, m, exc)
5:     START_ACCELERATOR(T, m, exc, conf, idx, PP, II)
6:     P, I ← reduction(PP, II)
```

Second, `NATSA` function implements the diagonal scheduling scheme presented in the previous section, setting the diagonals to be computed by each PU in idx (line 4). Third, it initiates the accelerator (line 5), which starts the computation, and the host CPU waits for all the processing units to finish. Once the computation finishes, the host CPU performs the final reduction of the private vectors (line 6) and the user can find the results in the P and I vectors. The `conf` argument (line 1), besides holding configuration parameters for the accelerator, allows for future extensions, such as using other distance metrics (e.g., Pearson correlation [13]).

V. Methodology

We describe the simulation environment and the workload we use to evaluate the performance of NATSA.

A. Simulation Environment

We simulate general-purpose cores using an in-house integration of *ZSim* [75], whose front-end is *Pin* [76], with *Ramulator* [61] [77]. ZSim is a simulator which can model 1) general purpose cores (both in-order and out-of-order cores), and 2) the conventional cache hierarchy. Ramulator is a cycle-level and extensible DRAM simulator that provides a wide variety of memory models, including DDR4 [25] and HBM [22]. We use *McPAT* [78] for power estimations.

For the NATSA accelerator, we use the *gem5* [79] and *Aladdin* [80] integration developed in [81]. Aladdin provides performance, area, and power estimations for a system-on-chip accelerator by requiring the equivalent C implementation of the accelerator design. Aladdin estimates the performance, power, and area of the accelerator within 0.9%, 4.9%, and 6.6% compared to that provided by RTL flows, but over 100× faster [80]. As Aladdin does not model the memory subsystem, we need to simulate it using gem5.

For a fair comparison, we evaluate our baseline platform (see the evaluated platforms below) in both ZSim and gem5 frameworks using the same workload (see Section V-B). We obtain up to 10% simulated time reduction using ZSim with respect to gem5 (i.e., the baseline system performs slightly better with ZSim). As a consequence, the performance benefits of NATSA with respect to the baseline simulated using gem5, would be even higher. However, we choose ZSim since simulations of manycore systems with ZSim are orders of magnitude faster than gem5 simulations [75], and this allows for the evaluation of general-purpose core platforms with large time series. For both general-purpose cores and accelerators, we obtain the power consumption of the memory system using the Micron Power Calculator [82], which we feed with the bandwidth usage from Ramulator and gem5, respectively.

Using these simulation environments, we define several representative hardware platforms for the evaluation:

- **DDR4-OoO (*Baseline*):** A conventional DDR4-based system with eight four-wide out-of-order cores at 3.75GHz. Each core has 32KB private L1 instruction/data caches and a private 256KB L2 cache. The cores share an 8MB L3 cache. The main memory is a dual channel 16GB DDR4-2400 with 38.4GB/s of memory bandwidth.
- **DDR4-inOrder:** A conventional architecture using 64 in-order cores at 2.5GHz. Each core has only a single level of private 32KB instruction/data caches. The main memory is the same DDR4 as in the baseline system. We use this simple core-cache configuration to compare with the following NDP general-purpose-core system.
- **HBM-OoO:** An NDP architecture with eight four-wide out-of-order cores at 3.75GHz. Each core has 32KB private L1 instruction/data caches and a private 256KB L2 cache. The main memory is a 4GB 3D-stacked HBM2 that provides a throughput of 256GB/s.
- **HBM-inOrder:** An NDP architecture with 64 in-order cores at 2.5GHz. Each core has a single level of private 32KB instruction/data caches. The main memory is a 4GB 3D-stacked HBM2 that provides a throughput of 256GB/s.
- **NATSA:** Our NDP accelerator with 48 PUs at 1GHz. Each PU has access to a private scratchpad memory of 1KB. The main memory is the same 4GB 3D-stacked HBM2 as in the *HBM-OoO* and *HBM-inOrder* platforms.

B. Workload

We use two real datasets and five synthetic datasets to evaluate the performance of NATSA against state-of-the-art architectures. The two real datasets are electrocardiogram (ECG) and seismology data obtained from [83] and [84]. We use these real datasets to 1) verify the correctness of the *matrix profile* computed by NATSA (the same approach used in [84]) and 2) evaluate the effect of using single-precision versus double-precision (see Section **??**). We generate the five synthetic datasets of different representative lengths [10] for performance evaluation using MATLAB, as shown in Table I.

Tabla I: Synthetic time series for performance evaluation.

Time Series	**rand_128K**	**rand_256K**	**rand_512K**	**rand_1M**	**rand_2M**
Length (n)	131072	262144	524288	1048576	2097152

VI. Evaluation

In this section, we first evaluate NATSA's performance, comparing it to the general-purpose platforms (DDR4-OoO, DDR4-inOrder, HBM-OoO, and HBM-inOrder). Second, we compare NATSA to both simulated and real architectures (e.g., manycore CPUs and GPUs [12]) in terms of power consumption and area. Third, we present a design space exploration of NATSA. Fourth, we analyze the performance of general-purpose cores and their bottlenecks. Finally, we evaluate SCRIMP in terms of precision and sensitivity to subsequence lengths (m).

A. Performance of NATSA

We evaluate the performance of two NATSA designs using single-precision (SP) and double-precision (DP), respectively. We present normalized performance of NATSA-DP with respect to the baseline platform (DDR4-OoO) in Fig. 7, using double-precision data. NATSA achieves significant performance improvements, up to 14.2× (9.9× on average) over the baseline system for large time series, and 6.3× over HBM-inOrder for all sizes. We observe that NATSA's speedup increases as the time series length becomes larger. This is because the arithmetic intensity decreases when the ratio of time series length (n) to window size (m) increases. Dot product update (Section II-B) causes the first dot product to take a significant part of the computation for shorter diagonals (lower n to m ratio). The cache hierarchy of the baseline system accelerates the first dot product. Conversely, a greater n to m ratio results in longer diagonals with the first dot product being less significant with respect to the total execution time, reducing the observed benefits of a cache hierarchy.

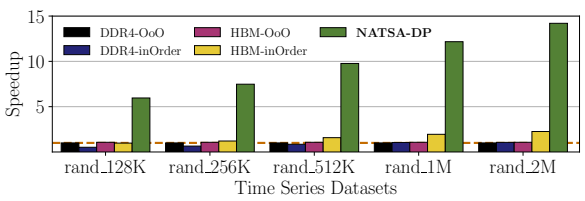


Fig. 7: Speedup with respect to the baseline platform (DDR4-OoO) using double precision data.

We evaluate the performance of the single-precision NATSA design. Table II presents the average execution time for the analyzed datasets. NATSA-SP, which provides higher performance with similar area cost to NATSA-DP, outperforms NATSA-DP by up to 1.75×, DDR4-OoO-DP by up to 24.9× and HBM-inOrder-DP by up to 11.1× for large time series.

Tabla II: Execution time (in seconds) for single-precision and double-precision data.

Config \ Dataset	rand_128K	rand_256K	rand_512K	rand_1M	rand_2M
DDR4-OoO-DP	14.72	77.55	414.55	2089.05	9810.30
DDR4-OoO-SP	6.46	44.47	207.85	1106.36	5206.75
HBM-inOrder-DP	14.95	64.20	262.33	1071.03	4347.38
HBM-inOrder-SP	8.16	35.68	130.23	625.27	2466.69
NATSA-DP	2.47	10.37	42.45	171.72	690.65
NATSA-SP	**1.41**	**5.91**	**24.19**	**97.84**	**393.45**

We conclude that NATSA provides the highest performance compared to modern general-purpose platforms.

B. Power, Energy and Area Consumption

Power and Energy Consumption. We compare the power and energy consumption of NATSA versus other existing hardware platforms in Figures 8 and 9. We use McPAT and Micron Power Calculators to evaluate energy consumption for the general-purpose platforms, getting the number of stalls and bandwidth usage from ZSim-Ramulator. For NATSA, we add Aladdin's energy estimations to the values obtained from the Micron Power Calculator. We also obtain energy measurements from real executions on GPUs using NVVP [85] and on CPUs using PCM [86], to compare NATSA with real platforms.

Fig. 8 shows the dynamic power consumption of each simulated or real hardware platform. We observe that NATSA has the lowest power consumption, and most of its power is consumed by memory.

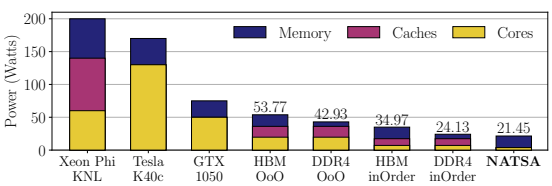


Fig. 8: Dynamic power consumption for simulated and real hardware platforms.

Fig. 9 shows the energy consumption of each simulated or real platform, for the computation of a time series of 524,288 elements (rand_512K) using double-precision. To calculate the energy consumption, we compute the power-delay product with the measured instantaneous power consumption and the execution time. NATSA reduces energy consumption by 27.2× (19.4× on average) over the baseline platform (DDR4-OoO), and by 10.2× over an NDP architecture with general-purpose cores (HBM-inOrder). NATSA consumes 1.7×, 4.1×, and 11.0× less energy than an NVIDIA Tesla K40c GPU, NVIDIA GTX 1050 GPU, and Intel Xeon Phi KNL, respectively. We conclude that NATSA is the most energy-efficient evaluated platform for *matrix profile*.

Area. We provide a scaled area comparison in Fig. 10. We observe that NATSA requires 9.6×, 7.9×, 3×, and 1.8× less area than an Intel Xeon Phi KNL (14nm), NVIDIA Tesla K40c (28nm), Intel Core i7 (32nm), and NVIDIA GTX 1050 (14nm).

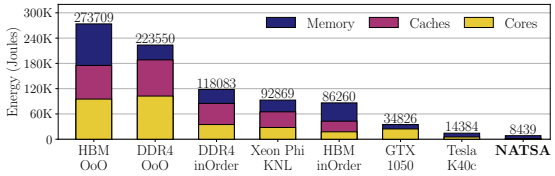


Fig. 9: Energy consumption for simulated and real hardware platforms.

We conclude that NATSA (at 45nm technology node) is the platform that requires the least area, while using the largest technology node (i.e., 45nm) compared to other evaluated architectures. Using a more recent and smaller technology node (e.g., 15nm instead of 45nm) could additionally reduce NATSA's energy consumption by 4× and area by 3× [87].

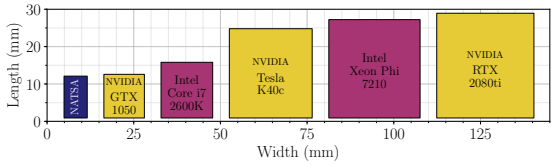


Fig. 10: Area comparison of different hardware platforms.

C. Performance of General-Purpose Cores

We evaluate the speedup over the baseline (DDR4-OoO) and memory bandwidth usage of SCRIMP, calculated using the ZSim-Ramulator framework for the DDR4-OoO, DDR4-inOrder, HBM-OoO and HBM-inOrder platforms using double-precision time series of different lengths (n), in Fig. 11.

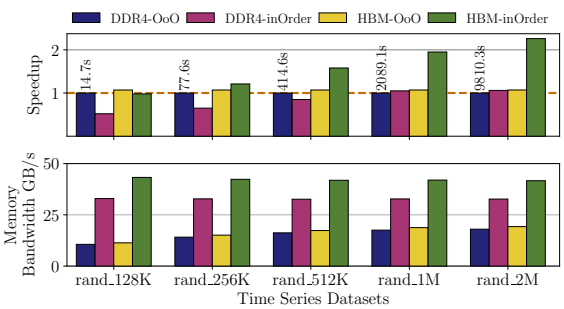


Fig. 11: Speedup over the baseline DDR4-OoO and memory bandwidth usage for general-purpose platforms.

We report execution time of the baseline (DDR4-OoO) on top of the respective performance bars in Fig. 11. Based on these results, we make three key observations. First, the DDR4-OoO platform does not use the peak available bandwidth of DDR4 (i.e., 38.4GB/s). We reinforce this observation with our HBM-OoO evaluation which replaces DDR4 with higher bandwidth HBM. HBM-OoO platform improves performance by only 7%, which means that providing more bandwidth does not significantly affect performance. This is because both platforms are compute-bound when executing SCRIMP. Second, the 64 lightweight cores of DDR4-inOrder slightly outperform the 8 complex cores of DDR4-OoO when $n \geq 1048576$ elements (i.e., rand_1M dataset). This

is because shorter time series can fit in the L3 cache. For long time series, the higher parallelism provided by the in-order platform enables higher memory-level parallelism and higher memory bandwidth demand, where DDR4 bandwidth becomes a bottleneck, resulting in a memory-bound system. Third, the HBM-inOrder platform provides up to 2.25× speedup over the baseline (DDR4-OoO), and consumes only 17% of the HBM's peak bandwidth with the largest dataset evaluated. In this case, even though performance is improved, the application is still compute-bound and simple NDP general-purpose cores cannot fully exploit the bandwidth provided by HBM (256GB/s) for the largest dataset we evaluate, which means that large datasets can be comfortably accommodated. We conclude that general-purpose platforms provide less performance than NATSA's balanced design because they do not effectively exploit the memory bandwidth of HBM.

VII. Related Work

To our knowledge, this is the first work that proposes a near-data processing accelerator for time series analysis. In this section, we briefly discuss prior work related to time series motif discovery and application-specific NDP accelerators.

Multiple techniques exist for time series motif and discord discovery [2,88]. A survey on time series motif discovery algorithms can be found in [5]. These implementations are approximate or exact in finding motifs and discords, which affects the time complexity of the algorithm. Exact motif and discord discovery processing of exceptionally large time series can be very time-consuming [13]. Consequently, *anytime* algorithms [9] are proposed to return a valid solution even if they are interrupted, and are expected to find better solutions the longer they run. *Matrix profile* [9] is the state-of-the-art exact *anytime* algorithm for time series motif and discord discovery. There are several implementations of *matrix profile*, including STAMP [9], STOMP [12], SCRIMP [10] and SCAMP [13]. SCRIMP is the state-of-the-art CPU-based implementation. Prior acceleration approaches to time series analysis [10, 12] mainly focus on accelerating STOMP and PreSCRIMP [10] on GPUs. Recently, SCAMP [13] framework combines a host (either a local machine or a server in a compute cluster) and workers that follow the directions from the host (either other CPUs in the cluster or accelerators such as GPUs). A SCRIMP version tuned for a many-core CPU (Intel Xeon Phi KNL) using vectorization can be found in [24].

Recent works explore Near Data Processing [14] for various applications using accelerators or general-purpose cores. In [39], ARM cores are used as NDP compute units to improve data analytics operators (e.g., group, join, sort). IMPICA [16] is an NDP pointer chasing accelerator. Tesseract [40] is a scalable NDP accelerator for parallel graph processing. TETRIS [41] is an NDP neural network accelerator. Lee et al. [42] propose an NDP accelerator for similarity search. GRIM-Filter [46] is an NDP accelerator for pre-alignment filtering in genome analysis [65]. Boroumand et al. [20] analyze the energy and performance impact of data movement for several widely-used Google consumer workloads, providing NDP accelerators for them. CoNDA [43] provides efficient cache coherence support for NDP accelerators. Finally, an NDP architecture [89] has been proposed for MapReduce-style applications.

VIII. Conclusion

We introduce NATSA, the first Near-Data-Processing (NDP) accelerator for time series analysis. NATSA 1) exploits the memory bandwidth of high-bandwidth memory (HBM) to analyze time series data at scale for a wide range of applications, 2) improves energy efficiency and execution time by using specialized low-power arithmetic units close to HBM memory, and 3) provides a novel workload scheduling scheme to prevent load imbalance and preserve the *anytime* property. NATSA outperforms the hardware platforms we evaluate in terms of performance, energy consumption and area requirements. We conclude that NATSA is an efficient NDP accelerator for time series, and hope that this work inspires future research directions in NDP for time series analysis.

Acknowledgments

We thank the anonymous reviewers of ICCD 2020 for feedback. This work has been supported by TIN2016-80920-R and UMA18-FEDERJA-197 Spanish projects, and Eurolab4HPC and HiPEAC collaboration grants. We also acknowledge support from the SAFARI Group's industrial partners, especially ASML, Facebook, Google, Huawei, Intel, Microsoft, and VMware, as well as support from Semiconductor Research Corporation.

References

[1] Robert H Shumway and David S Stoffer, "Time Series Analysis and Its Applications: With R Examples," 2017.
[2] Bill Chiu, Eamonn Keogh, and Stefano Lonardi, "Probabilistic Discovery of Time Series Motifs," in *SIGKDD*, 2003.
[3] Eamonn Keogh, Jessica Lin, Sang-Hee Lee, and Helga Van Herle, "Finding the Most Unusual Time Series Subsequence: Algorithms and Applications," *Knowledge and Information Systems*, 2006.
[4] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Zachary Zimmerman, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh, "Time Series Joins, Motifs, Discords and Shapelets: A Unifying View That Exploits The Matrix Profile," *JDMKD*, 2018.
[5] Sahar Torkamani and Volker Lohweg, "Survey on Time Series Motif Discovery," *WIREs: Data Mining and Knowledge Discovery*, 2017.
[6] Abdullah Mueen, "Time Series Motif Discovery: Dimensions and Applications," *WIREs: Data Mining and Knowledge Discovery*, 2014.
[7] Amardeep Sathyanarayana, Pinar Boyraz, Zelam Purohit, and John H. L. Hansen, "CAN-Bus Signal Analysis Using Stochastic Methods and Pattern Recognition in Time Series for Active Safety," *Springer-Verlag*, 2012.
[8] Abdullah Mueen, Eamonn J. Keogh, Qiang Zhu, Sydney Cash, and M. Brandon Westover, "Exact Discovery of Time Series Motifs," in *SDM*, 2009.
[9] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh, "Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets," in *ICDM*, 2016.
[10] Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman, Kaveh Kamgar, and Eamonn Keogh, "Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds," in *ICDM*, 2018.
[11] Avinash Sodani, "Knights Landing (KNL): 2nd Generation Intel® Xeon Phi Processor," in *HCS*, 2015.
[12] Yan Hu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah

Mueen, Philip Brisk, and Eamonn Keogh, "Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins," in *ICDM*, 2016.

[13] Zachary Zimmerman, Kaveh Kamgar, Nader Shakibay Senobari, Brian Crites, Gareth Funning, Philip Brisk, and Eamonn Keogh, "Matrix Profile XIV: Scaling Time Series Motif Discovery with GPUs to Break a Quintillion Pairwise Comparisons a Day and Beyond," in *SoCC*, 2019.

[14] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun, "Processing Data Where it Makes Sense: Enabling In-Memory Computation," *Microprocessors and Microsystems*, 2019.

[15] Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, and Onur Mutlu, "Enabling the Adoption of Processing-In-Memory: Challenges, Mechanisms, Future Research Directions," *arXiv*, 2018.

[16] Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu, "Accelerating Pointer Chasing in 3D-stacked Memory: Challenges, Mechanisms, Evaluation," in *ICCD*, 2016.

[17] Ximing Qiao, Xiong Cao, Huanrui Yang, Linghao Song, and Hai Li, "Atomlayer: a Universal Reram-based CNN Accelerator with Atomic Layer Computation," in *DAC*, 2018.

[18] Kevin K Chang, Prashant J Nair, Donghyuk Lee, Saugata Ghose, Moinuddin K Qureshi, and Onur Mutlu, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.

[19] Jeremie S Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu, "The DRAM latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern Commodity DRAM Devices," in *HPCA*, 2018.

[20] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," *ASPLOS*, 2018.

[21] Ka Hou Christien Li, Francesca Anne White, Timothy Tipoe, Tong Liu, Martin CS Wong, Aaron Jesuthasan, Adrian Baranchuk, Gary Tse, and Bryan P Yan, "The Current State of Mobile Phone Apps for Monitoring Heart Rate, Heart Rate Variability, and Atrial Fibrillation: Narrative Review," *JMIR Mhealth Uhealth*, 2019.

[22] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and S. Hong, "25.2 A 1.2V 8GB 8-channel 128GB/s High-Bandwidth Memory (HBM) Stacked DRAM with Effective Microbump I/O Test Methods using 29nm Process and TSV," in *ISSCC*, 2014.

[23] Donghyuk Lee, Saugata Ghose, Gennady Pekhimenko, Samira Manabi Khan, and Onur Mutlu, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," *TACO*, 2016.

[24] Ivan Fernandez, Alejandro Villegas, Eladio Gutierrez, and Oscar Plata, "Accelerating Time Series Motif Discovery in the Intel Xeon Phi KNL Processor," *The Journal of Supercomputing*, 2019.

[25] JEDEC JESD79-4C, "DDR4 SDRAM standard," `www.jedec.org/standards-documents/docs/jesd79-4a`, Accessed 23 September 2020.

[26] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh, "Searching and Mining Trillions of Time Series Subsequences Under Dynamic Time Warping," in *KDD*, 2012.

[27] Saugata Ghose, Amirali Boroumand, Jeremie S Kim, Juan Gómez-Luna, and Onur Mutlu, "Processing-In-Memory: A Workload-Driven Perspective," *IBM Journal of Research and Development*, 2019.

[28] H-S Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T Chen, and Ming-Jinn Tsai, "Metal–oxide RRAM," *Proceedings of the IEEE*, 2012.

[29] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das, "Compute caches," in *HPCA*, 2017.

[30] Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim, "Chameleon: Versatile and Practical Near-DRAM Acceleration Arch. for Large Mem. Sys.," in *MICRO*, 2016.

[31] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie, "PRIME: A Novel Processing-In-Memory Arch. for Neural Network Computation in ReRAM-Based Main Memory," in *ISCA*, 2016.

[32] Milad Hashemi, Onur Mutlu, and Yale N Patt, "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads," in *MICRO*, 2016.

[33] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-density 3D Memory," in *ISCA*, 2016.

[34] Gabriel H Loh, Nuwan Jayasena, M Oskin, Mark Nutter, David Roberts, Mitesh Meswani, Dong Ping Zhang, and Mike Ignatowski, "A Processing in Memory Taxonomy and a Case for Studying Fixed-Function PIM," in *WoNDP*, 2013.

[35] Vivek Seshadri and Onur Mutlu, "In-DRAM Bulk Bitwise Execution Engine," *arXiv*, 2019.

[36] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-In-Memory Architecture," in *ISCA*, 2015.

[37] Vivek Seshadri and Onur Mutlu, "Simple Operations in Memory to Reduce Data Movement," in *Advances in Computers*. Elsevier, 2017.

[38] Mingyu Gao, Grant Ayers, and Christos Kozyrakis, "Practical Near-Data Processing for In-Memory Analytics Frameworks," in *PACT*, 2015.

[39] Mario Paulo Drumond, Alexandros Daglis, Nooshin Mirzadeh, Dmitrii Ustiugov, Javier Picorel Obando, Babak Falsafi, Boris Grot, and Dionisios Pnevmatikatos, "The Mondrian Data Engine," in *ISCA*, 2017.

[40] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi, "A Scalable Processing-In-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.

[41] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *ASPLOS*, 2017.

[42] V T Lee, A Mazumdar, C C del Mundo, A Alaghi, L Ceze, and M Oskin, "Application Codesign of NDP for Similarity Search," in *IPDPS*, 2018.

[43] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T Malladi, Hongzhong Zheng, et al., "CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators," in *ISCA*, 2019.

[44] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Nastaran Hajinazar, Kevin Hsieh, Krishna T Malladi, Hongzhong Zheng, and Onur Mutlu, "LazyPIM: Efficient Support for Cache Coherence in Processing-In-Memory Architectures," *arXiv*, 2017.

[45] Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W. Keckler, " TOM: Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.

[46] Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu, "GRIM-Filter: Fast seed Location Filter. in DNA Read Mapping Using PIM Technologies," *BMC Genomics*, 2018.

[47] Damla Senol Cali, Gurpreet S Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, et al., "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis," in *MICRO*, 2020.

[48] Milad Hashemi, Eiman Ebrahimi, Onur Mutlu, Yale N Patt, et al., "Accelerating Dependent Cache Misses with an Enhanced Memory Controller," in *ISCA*, 2016.

[49] Vivek Seshadri, Kevin Hsieh, Amirali Boroum, Donghyuk Lee, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry, "Fast Bulk Bitwise AND and OR in DRAM," *CAL*, 2015.

[50] Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie, "Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories," in *DAC*, 2016.

[51] Gagandeep Singh, Juan Gómez-Luna, Giovanni Mariani, Geraldo F Oliveira, Stefano Corda, Sander Stuijk, Onur Mutlu, and Henk Corporaal, "NAPEL: Near-memory Computing Application Performance Prediction Via Ensemble Learning," in *DAC*, 2019.

[52] Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K Mishra, Mahmut T Kandemir, Onur Mutlu, and Chita R Das, "Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities," in *PACT*, 2016.

[53] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A Kozuch, Onur Mutlu, Phillip B Gibbons, and Todd C Mowry, "Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.

[54] Gagandeep Singh, Dionysios Diamantopoulos, Christoph Hagleitner, Juan Gomez-Luna, Sander Stuijk, Onur Mutlu, and Henk Corporaal, "NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling," in *FPL*, 2020.

[55] Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, et al., "RowClone: Fast and Energy-Efficient in-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.

[56] Jeremie S Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu, "D-RaNGe: Using Com. DRAM Devices to Generate True Random Numb. with Low Lat. and High Throughput," in *HPCA*, 2019.

[57] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L Greathouse, Lifan Xu, and Michael Ignatowski, "TOP-PIM: Throughput-Oriented Programmable Processing in Memory," in *HPDC*, 2014.

[58] Onur Mutlu, Hyesoon Kim, and Yale N Patt, "Efficient Runahead Execution: Power-Efficient Memory Latency Tolerance," *IEEE Micro*, 2006.

[59] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim, "HBM DRAM Technology and Architecture," in *IMW*, 2017.

[60] Ramyad Hadidi, Bahar Asgari, Burhan Ahmad Mudassar, Saibal Mukhopadhyay, Sudhakar Yalamanchili, and Hyesoon Kim, "Demystifying the Characteristics of 3D-stacked Memories: A case Study for Hybrid Memory Cube," in *IISWC*, 2017.

[61] Yoongu Kim, Weikun Yang, and Onur Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," *CAL*, 2015.

[62] Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu, "Demystifying Complex Workload-DRAM Interactions: An Experimental Study," in *SIGMETRICS*, 2019.

[63] Ziv Bar-Joseph, "Analyzing Time Series Gene Expression Data," *Bioinformatics*, 2004.

[64] Mohammed Alser, Hasan Hassan, Hongyi Xin, Oğuz Ergin, Onur Mutlu, and Can Alkan, "GateKeeper: A New Hardware Arch. for Accelerating Pre-alignment in DNA Short Read Mapping," *Bioinformatics*, 2017.

[65] Mohammed Alser, Zülal Bingöl, Damla Senol Cali, Jeremie Kim, Saugata Ghose, Can Alkan, and Onur Mutlu, "Accelerating Genome Analysis: A Primer on an Ongoing Journey," *IEEE Micro*, 2020.

[66] Peter Garrard, Vanda Nemes, Dragana Nikolic, and Anna Barney, "Motif Discovery in Speech: Application to Monitoring Alzheimer's Disease," *Current Alzheimer Research*, 2017.

[67] G Radhakrishnan, Deepa Gupta, S Sindhuula, Shrey Khokhawat, and TSB Sudarshan, "Experimentation and Analysis of Time Series Data from Multi-Path Robotic Environment," in *CONECCT*, 2015.

[68] Amy McGovern, Derek H Rosendahl, Rodger A Brown, and Kelvin K Droegemeier, "Identifying Predictive Multi-Dimensional Time Series Motifs: An Application to Severe Weather Prediction," *Data Mining and Knowledge Discovery*, 2011.

[69] Balázs Szigeti, Ajinkya Deogade, and Barbara Webb, "Searching for Motifs in the Behaviour of Larval Drosophila Melanogaster and Caenorhabditis Elegans Reveals Continuity Between Behavioural States," *Journal of The Royal Society*, 2015.

[70] Carmelo Cassisi, Marco Aliotta, Andrea Cannata, Placido Montalto, Domenico Patanè, Alfredo Pulvirenti, and Letizia Spampinato, "Motif Discovery on Seismic Amplitude T. Series: The Case Study of Mt Etna 2011 Eruptive Activity," *Pure Appl. Geophy.*, 2013.

[71] Eoin Cartwright, Martin Crane, and Heather J. Ruskin, "Financial Time Series: Motif Discovery and Analysis Using VALMOD," in *ICCS*, 2019.

[72] Anukool Lakhina, Mark Crovella, and Christiphe Diot, "Characterization of Network-Wide Anomalies in Traffic Flows," in *IMC*, 2004.

[73] Lal Hussain, Wajid Aziz, Jalal S. Alowibdi, Nazneen Habib, Muhammad Rafique, Sharjil Saeed, and Syed Zaki Hassan Kazmi, "Symbolic Time Series Analysis of (EEG) Epileptic Seizure and Brain Dynamics with Eye-Open and Eye-Closed Subjects During Resting States," *Journal of Physiological Anthropology*, 2017.

[74] Sameh Galal and Mark Horowitz, "Energy-Efficient Floating-Point Unit Design," *IEEE Transactions on Computers*, 2010.

[75] Daniel Sanchez and Christos Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems," in *ISCA*, 2013.

[76] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *PLDI*, 2005.

[77] SAFARI Research Group, "Ramulator Source Code," `https://github.com/CMU-SAFARI/ramulator`, Accessed 23 September 2020.

[78] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *MICRO*, 2009.

[79] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al., "The gem5 Simulator," *Comp. Arch. News*, 2011.

[80] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks, "Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures," in *ISCA*, 2014.

[81] Shao Yakun Sophia, Xi Sam Likun, Srinivasan Vijayalakshmi, Wei Gu-Yeon, and Brooks David, "Co-Designing Accelerators and SoC Interfaces Using gem5-Aladdin," in *MICRO*, 2016.

[82] "Micron Power Calculator," `www.micron.com/support/tools-and-utilities/power-calc`, Accessed 23 September 2020.

[83] A. Taddei, G. Distante, M. Emdin, P. Pisani, G. B. Moody, C. Zeelenberg, and C. Marchesi, "The European ST-T Database: Standard for Evaluating Systems for the Analysis of ST-T Changes in Ambulatory Electrocardiography," *European Heart Journal*, 1992.

[84] C M Yeh, H V Herle, and E Keogh, "Matrix Profile III: The Matrix Profile Allows Visualization of Salient Subsequences in Massive Time Series," in *ICDM*, 2016.

[85] "NVIDIA Visual Profiler," `https://developer.nvidia.com/nvidia-visual-profiler`, Accessed 23 September 2020.

[86] "Intel Processor Counter Monitor," `https://github.com/opcm/pcm`, Accessed 23 September 2020.

[87] Soheil Salehi and Ronald F DeMara, "Energy and Area Analysis of a Floating-Point Unit in 15nm CMOS Process Technology," in *SoutheastCon*, 2015.

[88] Pranav Patel, Eamonn Keogh, Jessica Lin, and Stefano Lonardi, "Mining Motifs in Massive Time Series Databases," in *ICDM*, 2002.

[89] Seth H Pugsley, Jeffrey Jestes, Huihui Zhang, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyuktosunoglu, Al Davis, and Feifei Li, "NDC: Analyzing the Impact of 3D-stacked Memory+Logic Devices on MapReduce Workloads," in *ISPASS*, 2014.

Análisis de la eficiencia energética y el rendimiento de SoCs ARM y RISC-V

Daniel Suárez[1], Alberto Cabrera[1], Francisco Almeida[1] y Vicente Blanco[1]

Resumen— **En los últimos años, ARM ha estado liderando el campo de los sistemas embebidos y los System-on-Chips (SoCs). Sin embargo, con la aparición de las máquinas basadas en RISC-V, se ha vuelto inevitable realizar una comparación entre estas dos Arquitecturas de Conjunto de Instrucciones (ISAs).**

En este documento, presentamos un análisis comparativo de la eficiencia energética y el rendimiento de las ISAs ARM y RISC-V en tres SoCs diferentes. Nuestro estudio incluye pruebas de referencia que miden el consumo de energía y el rendimiento global. Los resultados obtenidos son de gran valor para los desarrolladores e investigadores que buscan seleccionar la mejor ISA para aplicaciones informáticas de alta eficiencia energética.

Nuestros hallazgos sugieren que, si bien el consumo promedio de energía de la arquitectura RISC-V es menor que el de ARM, no necesariamente se traduce en un mejor rendimiento por vatio. En última instancia, nuestro objetivo es proporcionar un análisis integral de ambas ISAs y señalar las áreas en las que cada una puede mejorar para aumentar aún más la eficiencia energética.

Palabras clave— **Eficiencia Energética, SOCs, RISC-V**

I. Introducción

A medida que aumenta la demanda de computación eficiente en términos energéticos, se ha despertado un creciente interés en las arquitecturas alternativas de conjuntos de instrucciones (ISA) que ofrecen un mejor rendimiento por vatio en comparación con las opciones tradicionales. Dos de las ISAs más prometedoras en este sentido son RISC-V y ARM. Ambas ISAs presentan varias ventajas sobre las arquitecturas tradicionales como x86, incluyendo un diseño más simple, escalabilidad y un conjunto de instrucciones reducido, destacando especialmente que RISC-V es una especificación abierta.

El objetivo principal de este documento es realizar una comparativa exhaustiva de la eficiencia energética y el rendimiento de las implementaciones de las ISAs RISC-V y ARM. Para lograr este objetivo, llevaremos a cabo y analizaremos diversas pruebas de referencia que miden tanto el rendimiento como la eficiencia energética de ambas arquitecturas. Los hallazgos de este estudio serán de gran utilidad para los desarrolladores e investigadores interesados en seleccionar una arquitectura para aplicaciones de computación eficiente en términos energéticos. Además, estos resultados nos permitirán identificar áreas en las que cada ISA pueda mejorar para aumentar aún más su eficiencia energética. En última instancia, pretendemos ofrecer un análisis completo y riguroso de las ISAs RISC-V y ARM, y determinar cuál de las dos arquitecturas es más adecuada para la computación eficiente en términos energéticos.

La estructura de este trabajo se divide de la siguiente manera: En la sección II, realizaremos un breve repaso del estado del arte de los estudios de eficiencia energética en arquitecturas RISC-V. En la sección III, describiremos detalladamente la metodología empleada y los benchmarks realizados en nuestro análisis comparativo. Posteriormente, en la sección IV, presentaremos los resultados experimentales obtenidos al comparar todos los Sistemas en un Chip (SOCs) evaluados. Por último, en la sección V, expondremos las conclusiones obtenidas y discutiremos posibles líneas de trabajo futuro para continuar mejorando la eficiencia energética de estas arquitecturas.

Con este enfoque, buscamos contribuir al avance y desarrollo de la computación eficiente en términos energéticos, proporcionando información valiosa que permita a los profesionales de la industria y la investigación tomar decisiones fundamentadas sobre qué ISA utilizar en sus proyectos, maximizando así el rendimiento y minimizando el consumo de energía.

II. Estado del arte

En los últimos años, se ha observado un creciente interés en la investigación y análisis de la eficiencia energética y el rendimiento de los núcleos basados en RISC-V. Estudios como el realizado por Florian Zaruba en 2019 [1] han desempeñado un papel importante en este campo. En su investigación, Zaruba llevó a cabo un análisis exhaustivo del rendimiento y la energía de un núcleo RISC-V diseñado específicamente para sistemas Linux. Utilizando Ariane, una implementación de código abierto de la variante de 64 bits de RISC-V, los resultados revelaron una eficiencia energética excepcional, alcanzando hasta 40 Gop/sW en comparación con otros núcleos similares mencionados en la literatura científica. Un aspecto destacado de este estudio fue la importancia destacada de las extensiones de instrucciones para mejorar el rendimiento de cálculo, en lugar de centrarse únicamente en la operación a alta frecuencia.

En un estudio más reciente realizado por Elsadek y Tawfik en 2020 [2], se llevó a cabo una encuesta exhaustiva de los núcleos RISC-V de código abierto disponibles, clasificándolos en categorías de alto rendimiento y limitados en recursos. Posteriormente, se seleccionaron los núcleos más optimizados para dispositivos con recursos limitados, y se realizaron comparaciones basadas en la utilización de recursos y el consumo de energía. Los resultados de este estudio demostraron que el núcleo PicoRV32 destacó como el

[1]Dpto. de Ingeniería Informática y de Sistemas, Universidad de La Laguna, e-mail: dsuarezl@ull.es, falmeida@ull.es, vblanco@ull.es.

más eficiente en términos energéticos para dispositivos con recursos limitados. Estos hallazgos resaltan el potencial de RISC-V como una arquitectura de conjunto de instrucciones de procesador de código abierto, que permite el diseño de núcleos altamente eficientes en términos de consumo de energía.

La combinación de estos estudios y otros esfuerzos de investigación ha generado un mayor entendimiento y apreciación de las ventajas de los núcleos basados en RISC-V en términos de eficiencia energética. Esto ha impulsado aún más el interés en explorar y aprovechar todo el potencial de esta ISA abierta y escalable.

III. Metodología

A. Especificaciones de las máquinas

Realizamos una comparación de rendimiento y eficiencia energética entre las implementaciones de las ISAs RISC-V y ARM. Para ello, ejecutamos y analizamos pruebas de referencia en tres máquinas diferentes cuyas especificaciones se muestran en la tabla I.

Tabla I: Especificaciones de las máquinas

	Odroid XU4	**Rock 960**	**Nezha D1**
CPU	Exynos5422 Cortex-A15 + Cortex-A7 Octa core CPUs	Cortex-A72 Dual-core up + Cortex A53 Quad-core	Allwinner D1
Frequency	2GHz and 1.4GHz	1.8GHz and 1.4GHz	1.0GHz
Architecture	armv7l	aarch64	riscv64
RAM	2GB LPDDR3	4GB LPDDR3	1GB DDR3
Input power	20W	24W	10W
Operative System	Ubuntu 20.04.5 LTS	Ubuntu 20.04.4 LTS	Debian GNU/Linux 11 or Yocto Project

B. Descripción de las pruebas

Utilizamos dos pruebas de referencia, el Benchmark NAS [3] (versión 3.2, clase A, SER) y el Benchmark TFLite [4] (modelos mobilenet v1 a v3).

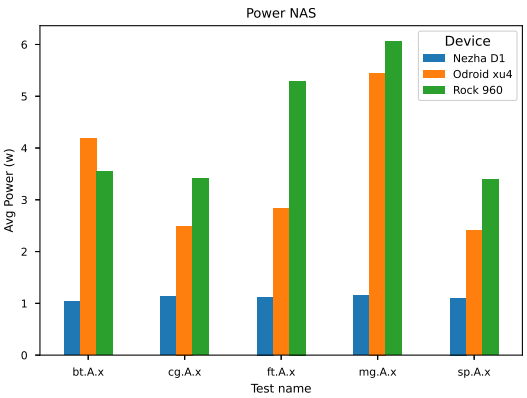
El Benchmark NAS es un conjunto de programas paralelos que miden el rendimiento de computadoras paralelas. Incluye un conjunto de núcleos computacionales que se utilizan para evaluar el rendimiento de varios algoritmos paralelos. Por otro lado, el Benchmark TFLite es una herramienta de referencia para evaluar el rendimiento de modelos de aprendizaje automático en dispositivos móviles y embebidos. Está diseñado para medir el tiempo de inferencia de modelos de aprendizaje profundo que se han convertido al formato TensorFlow Lite.

Estos benchmarks son ampliamente utilizados en el campo de la informática para evaluar el rendimiento de diversos sistemas informáticos, incluyendo procesadores, GPUs y sistemas distribuidos.

Para garantizar la equidad en las pruebas, implementamos scripts en Python que se encargaron de ejecutar de manera simultánea los tests en todas las máquinas. Estos scripts no solo coordinaron la ejecución de las pruebas, sino que también controlaron activamente la temperatura de las máquinas durante todo el proceso.

Antes de iniciar cada lote de pruebas, los scripts monitorearon constantemente la temperatura de las máquinas y esperaron hasta que todas alcanzaran una temperatura base establecida. Esta temperatura base se definió como la temperatura de las máquinas en reposo, lo cual aseguró un punto de referencia uniforme para todas las pruebas. Solo cuando se alcanzó esta temperatura base, los scripts procedieron a ejecutar el siguiente conjunto de pruebas.

Fig. 1: Resultados de potencia de NAS Benchmark



Los benchmarks se ejecutaron múltiples veces en cada máquina y se obtuvo el promedio de los resultados para obtener los valores finales. Para comparar la eficiencia energética y el rendimiento de las diferentes máquinas, se trazaron los resultados y se realizaron análisis estadísticos.

Para obtener los resultados energéticos de los benchmarks, se utilizó el dispositivo AccelPowerCape [5] para medir el consumo de energía actual. Esto implicó conectar un cable desde el transformador de los dispositivos hasta el AccelPowerCape. El dispositivo AccelPowerCape está diseñado específicamente para medir el consumo de energía de los dispositivos en tiempo real, lo que lo convirtió en una herramienta ideal para nuestros fines de evaluación energética. Al utilizar el dispositivo, pudimos medir con precisión el consumo de energía de los dispositivos y obtener resultados energéticos confiables para nuestros benchmarks.

IV. Resultados experimentales

En esta sección, se presentarán los resultados de las pruebas NAS Benchmark y TFLite Benchmark en los SoCs mencionados anteriormente. Se analizará el rendimiento y la eficiencia energética de estos SoCs, proporcionando una comparativa clara de sus capacidades. Para los resultados de los benchmark de TFLite nos encontramos con limitaciones que nos impidieron ejecutar la prueba en el dispositivo Odroid XU4. Además, fue necesario crear una imagen personalizada utilizando el Proyecto Yocto [6], específicamente para el Nezha D1, para llevar a cabo exitosamente la prueba.

A. Potencia

En los resultados experimentales de potencia econtrados en la figuras 1 y 2, se ha medido la potencia promedio en vatios consumida por cada una de las máquinas.

Se observó que la Nezha D1 fue la máquina que consumió la menor potencia tanto en el TFlite

Fig. 2: Resultados de potencia de TFlite Benchmark

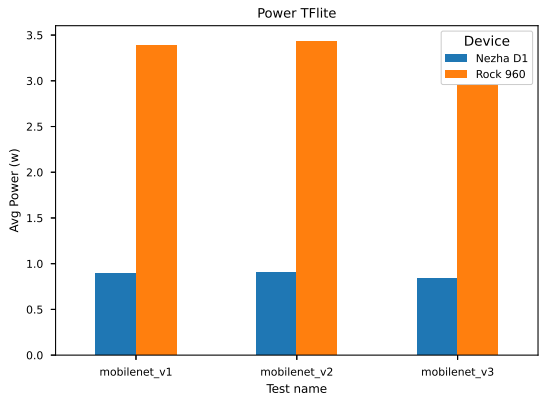


Fig. 3: Resultados de rendimiento de NAS Benchmark

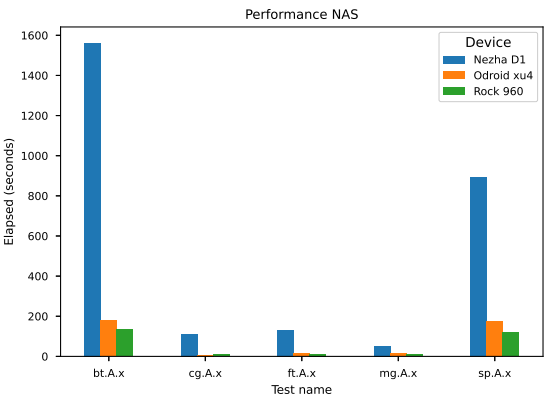
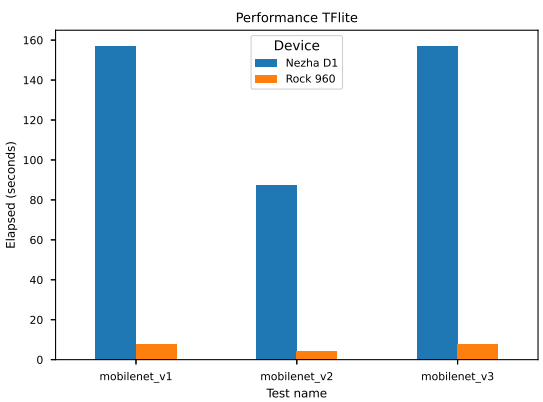


Fig. 4: Resultados de rendimiento de TFlite Benchmark



Benchmark como en el NAS Benchmark. La Odroid Xu4 ocupó el segundo lugar en términos de consumo de potencia, mientras que la Rock960 fue la que generalmente consumió más potencia. Cabe destacar que la potencia consumida por la Nezha D1 se mantuvo constante en todos los casos.

B. Rendimiento

En esta sección, se presentan los resultados de rendimiento obtenidos al medir el tiempo que cada máquina tardó en completar los tests, los cuales se encuentran detallados en las figuras 4 y 3.

Al analizar los tiempos de ejecución, se observó que la Nezha D1 registró el mayor tiempo, mostrando una diferencia significativa en comparación con las

Fig. 5: Resultados de energía de NAS Benchmark

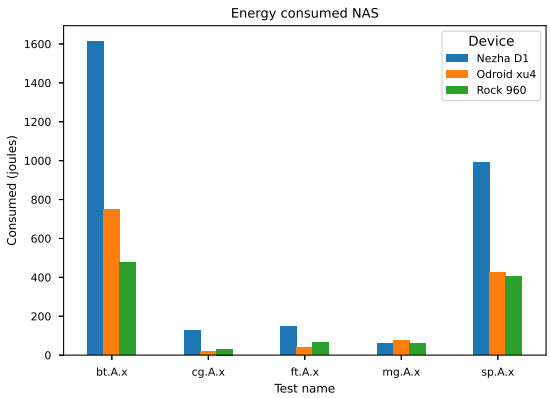
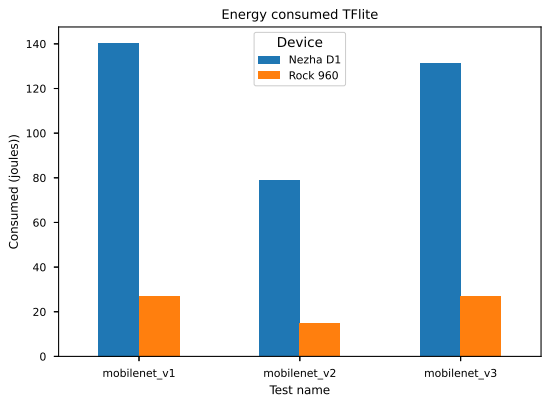


Fig. 6: Resultados de energía de TFlite Benchmark



demás máquinas. Esta disparidad en el rendimiento puede tener implicaciones en su idoneidad para ciertos escenarios. Por su parte, la Odroid XU4 se ubicó como la segunda más lenta en los tests en los que fue posible su ejecución. Por otro lado, la Rock960 demostró ser la máquina más rápida en términos de rendimiento, superando a las demás máquinas evaluadas.

C. Energía total

En esta sección, se presentarán los resultados enfocándonos en el consumo total de energía en julios de cada una de las máquinas evaluadas. Los resultados detallados se pueden encontrar en las figuras 5 y 6.

Durante el análisis, se observó que la máquina Nezha D1 registró el mayor consumo total de energía, posiblemente debido al mayor tiempo necesario para completar la ejecución de los benchmarks en comparación con las otras máquinas. Por otro lado, en los casos en los que se pudieron ejecutar los benchmarks en la Odroid XU4 y la Rock960, se encontró que ambas máquinas presentaban consumos de energía similares, aunque en algunas ocasiones una máquina consumía más energía que la otra y viceversa.

D. Operaciones por segundo

En el caso del NAS Benchmark, las operaciones por segundo representan las operaciones de punto flotante realizadas por la máquina en un intervalo

Fig. 7: Resultados de FLOPS de NAS Benchmark

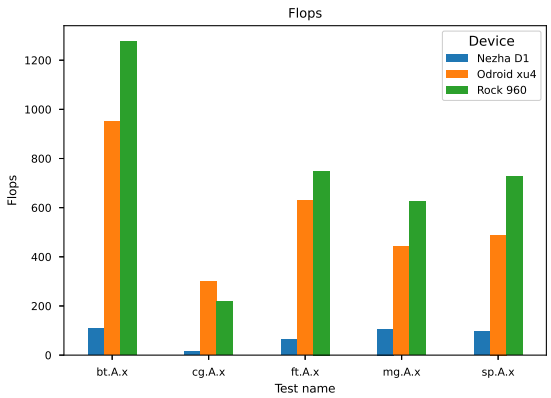


Fig. 8: Inferencia/segundo de TFlite Benchmark

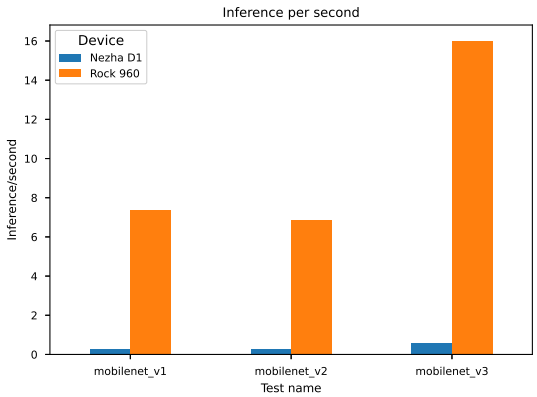


Fig. 9: Resultados de FLOPS por vatio de NAS Benchmark

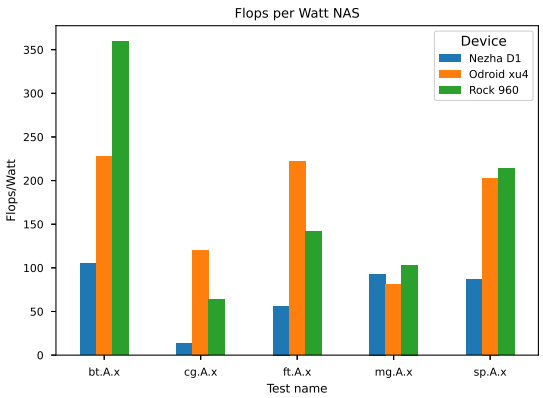
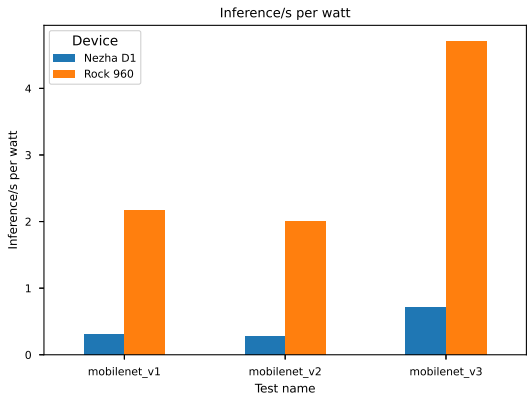


Fig. 10: Inferencia/segundo por vatio de TFlite Benchmark



de tiempo determinado. Por otro lado, en el caso del TFLite Benchmark, se refiere al número de inferencias realizadas por la máquina por segundo.

Al analizar los resultados del test NAS presentes en la figura 7, se observa que la máquina con el mejor rendimiento por vatio es claramente la Rock960, superando a la Odroid XU4 en la mayoría de los casos. La Nezha D1, de acuerdo con sus especificaciones, mostró un rendimiento inferior en comparación con sus competidores.

En cuanto a los resultados de los tests de TFLite mostrados en la figura 8, se encontró que la Rock960 obtuvo un rendimiento considerablemente superior al de la Nezha D1. Estos resultados indican que la Rock960 es capaz de realizar un mayor número de inferencias por segundo, lo que la convierte en una opción más potente para aplicaciones que requieren un procesamiento rápido.

E. Eficiencia energética

En los benchmarks NAS y TFlite, es importante diferenciar el concepto de “eficiencia energética”. En el NAS Benchmark, este término se refiere al número de operaciones de punto flotante realizadas por segundo por vatio, mientras que en el TFlite Benchmark se refiere al número de inferencias realizadas por segundo por vatio.

Si analizamos los resultados del test NAS, que se presentan en la figura 9, podemos observar que el rendimiento por vatio varía entre las máquinas evaluadas. Tanto el Odroid XU4 como el Rock960 obtuvieron buenos resultados en diferentes pruebas, demostrando un rendimiento competitivo. Sin embargo, es importante señalar que la Nezha D1 mostró un rendimiento pobre en comparación con las otras máquinas.

Es relevante destacar que ciertos problemas limitados por la memoria, como el caso de ”MG”, redujeron significativamente la diferencia de rendimiento entre la Nezha D1 y las otras máquinas. En este tipo de escenarios, la importancia de la CPU se ve reducida, lo que explica la disminución de la disparidad de rendimiento observada.

Finalmente, al analizar los resultados del test TFlite en la figura 10, podemos notar que la Rock960 mostró un rendimiento energético notablemente superior en comparación con la Nezha D1. Estos resultados indican que la Rock960 logró realizar un mayor número de inferencias por segundo por vatio, lo que demuestra su mayor eficiencia energética en este contexto específico.

V. Conclusiones

En conclusión, los resultados de nuestros benchmarks están en línea con las características esperadas de cada máquina. Tanto el Odroid XU4 como el Rock960 mostraron un mejor rendimiento en los tests NAS y TFlite en comparación con la Nezha D1. Sin embargo, es importante destacar que también consumieron más potencia promedio que la Nezha D1.

En cuanto a la arquitectura, no parece haber sido un factor especialmente relevante en los resultados. Aunque las implementaciones de la arquitectura RISC-V son relativamente nuevas en comparación con sus competidores, las diferencias en rendimiento y consumo de potencia se deben más a las características específicas de cada dispositivo que a la arquitectura en sí.

Es crucial considerar el caso de uso y el entorno específico al elegir un dispositivo. Si el consumo de potencia es una preocupación, el Nezha D1 puede ser una opción más adecuada, especialmente en entornos con suministro de potencia limitado. Por otro lado, si se prioriza el rendimiento o la eficiencia energética, y la potencia no es un factor limitante, los dispositivos Odroid XU4 y Rock960 podrían ser opciones más apropiadas.

Es relevante destacar que las implementaciones de la arquitectura RISC-V aún se encuentran en una etapa relativamente nueva en comparación con sus competidores más establecidos. Conforme esta arquitectura continúa desarrollándose y optimizándose, es probable que veamos mejoras adicionales en su rendimiento y eficiencia energética en el futuro

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia e Innovación de España con los proyectos PID2019–107228RB–I00, TED2021–131019B–I00 y PDC2022–134013–I00; y por el Gobierno de Canarias con el proyecto ProID2021010012.

Referencias

[1] Florian Zaruba and Luca Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.

[2] Islam Elsadek and Eslam Yahya Tawfik, "Risc-v resource-constrained cores: A survey and energy comparison," in *2021 19th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2021, pp. 1–5.

[3] "Nas parallel benchmark," `https://www.nas.nasa.gov/software/npb.html`.

[4] "Tflite benchmark," `https://www.tensorflow.org/lite/performance/measurement`.

[5] "Accelpowercape," `https://artecs.dacya.ucm.es/tools/accelpowercape/`.

[6] "Yocto project," `https://www.yoctoproject.org/`.

[7] J. M. Smith and A. B. Jones, *Book Title*, Publisher, 7th edition, 2023.

[8] A. B. Jones and J. M. Smith, "Article Title," *Journal title*, vol. 13, no. 52, pp. 123–456, 3 2024.

[9] "The risc-v instruction set manual. volume 1: User-level isa, version 2.0," 2014.

[10] "MS Windows NT kernel description," `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm`, Accessed: 2010-09-30.

Análisis del comportamiento de programas multihilo en sistemas NUMA

Javier Beiro[1], Ruben Laso[1], Oscar G. Lorenzo[1,2], José C. Cabaleiro[1,2], Tomás F. Pena[1,2], Francisco F. Rivera[1,2] y Juan A. Lorenzo[3]

Resumen— **Los sistemas NUMA son una opción escalable en sistemas de memoria compartida, comprender su comportamiento en relación con la localidad de los datos y la afinidad de los hilos es clave para optimizar su eficiencia. Este trabajo examina los sistemas NUMA multihilo, centrándose en cómo la disposición de dos hilos en nodos impacta en la ejecución de programas con predominio de operaciones de acceso a memoria. Estudiamos la localidad evaluando un solo hilo haciendo operaciones de memoria locales y remotas, luego introducimos un segundo hilo para evaluar los efectos de la compartición de recursos, y finalmente analizamos la afinidad observando efectos del acceso paralelo a un único vector de datos.**

Tras realizarse pruebas sobre un sistema NUMA de 4 nodos e interconexiones homogéneas se observó que; primero, el acceso local a los datos ofrece un rendimiento hasta 3 veces superior a los accesos remotos; segundo, la compartición de recursos puede penalizar rendimiento en programas con acceso intensivo a datos; y tercero, los tiempos de acceso a datos compartidos pueden verse afectados por los protocolos de coherencia caché, sincronizándose los hilos a aquel de acceso más lento. Este último efecto puede mitigarse retrasando la ejecución de un hilo, limitando conflictos de acceso a memoria, permitiendo que el hilo adelantado se aproxime al rendimiento aislado y el hilo con el retardo mejore sus tiempos de ejecución gracias a la precarga de datos.

Así, este estudio identifica pérdidas de rendimiento y sugiere posibles soluciones mediante planificación de accesos y migración de hilos/datos.

Palabras clave— **Afinidad, localidad, NUMA, QPI.**

I. Introducción

El constante aumento del número de núcleos y de la densidad de transistores en los microprocesadores actuales ha propiciado la aparición de distintas problemáticas como el incremento de la contención en los buses de comunicación y retardos en la propagación de las señales dentro de los propios bancos de memoria caché [1]. Ante esta situación, las arquitecturas NUMA (*Non-Uniform Memory Access*) constituyen una posible solución, permitiendo un acceso más eficiente y escalable a la memoria en sistemas multinúcleo [2].

Los sistemas NUMA, compuestos por nodos interconectados a través de una red dedicada, posibilitan que cada procesador o nodo tenga acceso directo a un segmento de la memoria física, mientras que el acceso a las ubicaciones restantes se realiza a través del controlador de memoria de otros procesadores [3]. Esta característica clave, junto con su capacidad para mitigar la contención del bus y distribuir de manera eficiente la memoria principal y la caché, configura el enfoque de nuestro estudio sobre la localidad y afinidad en estos sistemas.

En los sistemas NUMA el concepto de afinidad se encuentra íntimamente ligado a la asignación óptima de tareas a nodos para lograr la máxima eficiencia. El concepto de afinidad está relacionado con la velocidad y el *overhead* de los recursos del nodo requeridos por la tarea [4]. En entornos NUMA y de computación paralela, la afinidad se podría caracterizar por tres aspectos clave que tienen una influencia directa sobre el rendimiento: los recursos limitados de cada nodo, el estado de la jerarquía de memoria del sistema y la correlación entre distintas tareas en ejecución. Siguiendo este mismo enfoque, en este estudio, la afinidad se definirá según la disposición de los hilos entre los nodos, la configuración relativa de los hilos que operan en paralelo entre sí y su organización con respecto a los datos a los que acceden.

En este contexto, resulta de gran interés caracterizar los incrementos o las reducciones de rendimiento que conllevan cada una de estas configuraciones. De esta manera, podría ser posible determinar aquellas que proporcionan una mayor afinidad, permitiendo así obtener ejecuciones más eficientes sin necesidad de variar la algoritmia o lógica de los programas.

Trabajos previos muestran el efecto que tiene la ubicación de hilos y datos en arquitecturas con interconexiones asimétricas de los nodos, donde existen diferencias entre el ancho de banda de las distintas conexiones [5]. Otros estudios muestran los efectos del ancho de banda en función de la distribución de hilos y datos en el sistema [6] o se centran en los efectos de la congestión de los buses de interconexión y controladores de memoria en programas multihilo, con un gran volumen de datos desplazándose por el sistema [7].

En contraposición, este trabajo pretende aportar una base sistemática que modele comportamientos básicos con dos hilos e ilustre la variación en el rendimiento de los sistemas ante la compartición de recursos, la ejecución de hilos independientes concurrentes y/o el solapamiento de hilos paralelos con datos compartidos. Específicamente, las pruebas se enfocarán en establecer qué distribución es más "afín" en tres escenarios diferentes: el primero, al utilizar un solo hilo de ejecución que accede a datos privados; el segundo, al tener dos hilos trabajando en paralelo, cada uno con acceso a su propio conjunto de datos privados; y el tercero, donde dos hilos acceden a un conjunto compartido de datos, variando, en este últi-

[1]Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Galicia, España.

[2]Departamento de Electrónica y Computación, Universidade de Santiago de Compostela, Galicia, España.

[3]ETIS, UMR 8051, CY Cergy Paris Université, ENSEA, CNRS, Francia.

mo caso, el nivel de "solapamiento" en los accesos.

La organización de este documento es la siguiente: en la sección II se detallan las especificaciones de las ejecuciones experimentales, incluyendo las características del computador NUMA donde se llevarán a cabo, así como la estructura general de las pruebas, subrayando las decisiones clave de su diseño. La sección III proporciona una caracterización completa de las pruebas y presenta y discute los resultados obtenidos. Por último, la sección IV recapitula los hallazgos y presenta las conclusiones finales.

II. Entorno de ejecución

Antes de realizar los experimentos, se caracterizará por completo la arquitectura donde se ejecutará, así como la estructura general de las pruebas.

A. Arquitectura del computador empleado

Para la realización de las pruebas se ha utilizado un servidor NUMA conformado por 4 nodos distribuidos en forma de anillo, de manera que existen dos conexiones físicas por cada uno que los conecta a otros dos de la red. La topología del sistema se muestra en la figura 1, así como la distancia entre nodos reportada por `numactl` [8]. Cada uno de los 4 nodos NUMA posee un procesador Intel Xeon E5-4620 v4 con las características mostradas en la Tabla I. La frecuencia de los *cores* se ha fijado a 2,1 GHz para obtener resultados consistentes y estables.

Tabla I: Características técnicas del procesador.

Arquitectura:	x86_64t
CPU MHz:	2100
CPU max MHz:	2100
CPU min MHz:	1200
L1d cache:	32 KiB
L2 cache:	256 KiB
L3 cache:	25 600 KiB
Thread(s) per core:	1
Core(s) per socket:	10

Los nodos están interconectados con Intel QPI [9]. Cada uno de los puertos QPI soporta dos conexiones unidireccionales para permitir comunicación bidireccional entre dos componentes (nodos), de manera que se soporta el tráfico en ambas direcciones simultáneamente. Asimismo, la conexión QPI se define empleando un modelo en 5 capas. En este trabajo, destaca la relevancia de la *capa de protocolo*, que modela el mecanismo de coherencia caché y será especialmente relevante a la hora de comprender los comportamientos observados [9].

Para realizar las tareas de coherencia caché, QPI emplea dos tipos distintos de agentes: *caching agents* y *home agents*.

- *Caching agent*: representa una entidad que puede iniciar una transacción de coherencia de memoria y mantener copias de su propia estructura de caché. Está definido por los mensajes que transmite y su uso depende del comportamiento del protocolo de coherencia caché. Este agente también puede proveer copias de su memoria caché coherente a otros agentes (*forward*).
- *Home agent*: representa una entidad que ofrece transacciones coherentes, incluyendo el *handshaking*, en caso de ser necesario, con los *caching agents*. Supervisa la porción de memoria correspondiente a su nodo, tratando los conflictos que puedan ocasionarse entre los distintos *caching agents*.

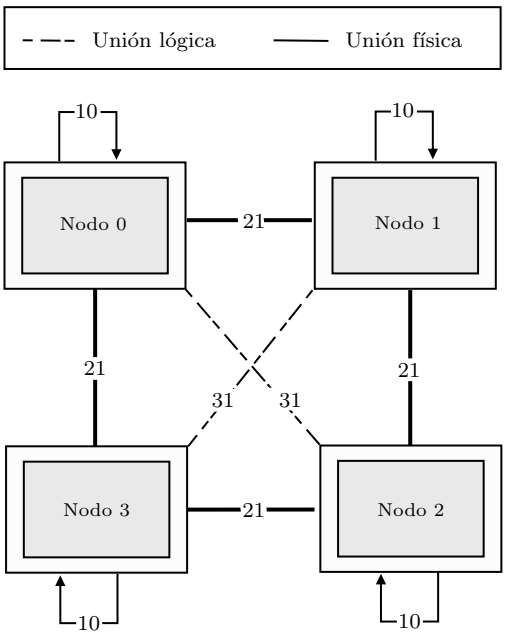


Fig. 1: Topología del servidor donde se realizarán las pruebas.

QPI permite configurarse de forma que se pueda variar el comportamiento de estos agentes ante las distintas solicitudes de datos de los nodos. En concreto, permite trabajar en los modos *home snoop* y *source snoop*.

- *Home snoop*: está optimizado para permitir gran escalabilidad y se utiliza para sistemas grandes donde existe mayor tráfico de *snoop*.
- *Source snoop*: está optimizado para ofrecer menor latencia y se utiliza principalmente en sistemas más pequeños donde el número de agentes crea una cantidad de trabajo de *snoop* relativamente baja.

Dado que versiones posteriores a QPI, como UPI, ya no incorporan el modo de *source snooping* o soluciones similares [10], el estudio se centrará en el modo *home snooping*.

QPI implementa el protocolo de coherencia MESIF, donde las líneas caché están en los siguientes estados: *Modified*, *Exclusive*, *Shared*, *Invalid* o *Forward*. Este último permite el intercambio y copia de líneas limpias entre nodos, de forma que un agente tiene la línea caché en estado *Forward* y el resto en estado *Shared*.

B. Estructura de las pruebas

Los experimentos realizados en este trabajo, se enfocarán en caracterizar de manera exhaustiva los

comportamientos más elementales de acceso a memoria, lecturas (R), escrituras (W) y lecturas-escrituras (R-W), como se muestra en los códigos 1, 2 y 3. Estas operaciones podrán ser ejecutadas por uno o dos hilos, buscando caracterizar su localidad y afinidad.

```
add = vector[i];
```

Código 1: Lecturas.

```
add = 'X';
vector[i] = add;
```

Código 2: Escrituras.

```
add = vector[i];
vector[i] = add + 1;
```

Código 3: Lecturas y escrituras.

Dado este tipo de accesos, se realizarán tres grupos de pruebas, que caracterizarán el coste temporal de:

1. El acceso de un hilo a datos privados posicionando el hilo y los datos en todas las combinaciones de nodos posibles.
2. El acceso de dos hilos paralelos a sus datos privados en las distribuciones de datos e hilos consideradas más representativas.
3. El acceso de dos hilos paralelos a un único grupo de datos compartido en las distribuciones consideradas más representativas.

Todos estos accesos comparten la estructura mostrada en el Código 4, donde:

- Se previenen optimizaciones del compilador de forma activa usando el *qualifier* `volatile` evitando cualquier tipo de reordenación u optimización por parte el compilador que imposibilite el estudio [11]. Además, el vector se recorre con dos bucles `while` al existir menos opciones de optimización de estos bucles que de los bucles `for`.
- El uso de los dos bucles anidados permite que el índice `i` del bucle más interno recorra el *array* con un *stride* de `dist` posiciones entre accesos. Para evitar los efectos del *prefetching*, se ha fijado `dist` como tres veces el tamaño de una línea.

Para la distribución de los datos y los hilos en los nodos se ha utilizado la librería `libnuma` [12] y la función `sched_setaffinity` [13], de manera que, en caso de querer realizar estudios distribuyendo los hilos en *cores* concretos en lugar de nodos, las modificaciones resulten inmediatas.

Todas las pruebas se han realizado recorriendo un *array* de `char` de forma que el número de elementos coincida con el número de *bytes* que ocupa. En concreto, el vector tendrá un tamaño de 209 715 200 B, que corresponde con 8 veces la capacidad máxima de la caché L3, de forma que el vector completo no pueda residir completamente en caché.

```
int accesos(const vectorSize size){
  volatile int add = 0;
  volatile vectorSize j = 0, i = 0;

  while(j < dist){
    i = j;
    while(i < size){
      //Operacion de acceso a memoria
      i+=dist;
    }
    j++;
  }
  return add;
}
```

Código 4: Función base para recorrer el vector.

Todas las pruebas se realizaron empleando: `gcc` versión 12.1.1 y el sistema operativo AlmaLinux 8.6 (*Sky Tiger*), con el kernel 4.18.0.

III. Resultados experimentales

Una vez caracterizado el entorno y ejecutadas las pruebas, podemos exponer los resultados para los tres grupos de pruebas estudiados.

A. Un hilo y datos privados

Antes de realizar pruebas en las que participen varios hilos de acceso simultáneo, es importante llevar a cabo un estudio del coste de acceso de un hilo a los diferentes nodos. De esta manera, se pueden caracterizar los retardos típicos de transmisión de datos, excluyendo los efectos de contención de bus, problemas de coherencia caché que pudieran surgir por el acceso concurrente de varios hilos, así como las interacciones entre hilos que comparten niveles de caché dentro de un mismo nodo.

En las tablas II, III y IV se muestran todas las permutaciones de un hilo y un vector de datos sobre los cuatro nodos, indicando en cada una el coste de acceso en segundos para realizar operaciones de lectura, escritura y lectura-escritura sobre 209 715 200 elementos de tipo `char`.

Los resultados obtenidos demuestran una simetría notable, resultando indiferente, por ejemplo, situar los datos en el nodo 0 y el hilo en el nodo 2 o viceversa. Además, cabe destacar que los resultados están en consonancia con la topología estudiada en la figura 1 dado que, en todos los escenarios, el coste más bajo corresponde al acceso del nodo a sus propios datos locales, seguido por aquellos nodos inmediatamente adyacentes con los que mantiene un enlace directo. Finalmente, los accesos que requieren pasar a través de un nodo intermedio presentan el mayor coste. De esta manera, las disposiciones más afines para un hilo que accede a sus datos privados sin la presencia de otros hilos concurrentes, son aquellas en las que el hilo y los datos se encuentran en el mismo nodo.

Cabe destacar el comportamiento observado en las escrituras, donde el tiempo de ejecución es mucho mayor si estas se realizan en ausencia de lecturas. Revisando distinta documentación [14] [15] y ejecutando distintas pruebas, se observó que el comportamiento podría estar asociado a un *bypass* de la caché.

Tabla II: Costes de lecturas (segundos).

		Hilo			
		N0	**N1**	**N2**	**N3**
Datos	**N0**	2,0307	5,8903	6,0408	5,8923
	N1	5,9153	2,0355	5,9121	6,1039
	N2	6,0733	5,9569	2,0281	5,9095
	N3	5,9467	6,0511	6,0511	2,0253

Tabla III: Costes de escrituras (segundos).

		Hilo			
		N0	**N1**	**N2**	**N3**
Datos	**N0**	19,1069	31,0287	40,3092	30,6787
	N1	30,7297	19,0377	30,6631	40,3776
	N2	40,3638	30,6935	18,6312	30,5306
	N3	30,7069	40,0940	30,8610	18,9247

Tabla IV: Costes de lectura-escritura (segundos).

		Hilo			
		N0	**N1**	**N2**	**N3**
Datos	**N0**	3,3372	5,0028	6,3672	4,9907
	N1	4,9637	3,3016	4,9850	6,3658
	N2	6,3785	4,9816	3,2891	4,9684
	N3	5,0374	6,3541	5,0072	3,3304

Al escribir un dato que no se reutiliza, las escrituras se estarían realizando directamente en memoria para prevenir la sustitución de una línea caché que sí pueda ser de utilidad en el futuro, resultando en un incremento notable del tiempo de ejecución.

B. Dos hilos y datos privados

En este experimento, dos hilos accederán, cada uno, a un *array* de datos distinto. Utilizaremos un conjunto de datos privados para cada hilo para caracterizar exhaustivamente los efectos de congestión del bus y la compartición (o no) del procesador durante la ejecución, evitando la aparición de problemas de coherencia. La contención de bus [16] se manifiesta cuando múltiples dispositivos o procesos intentan emplear el mismo canal de comunicación, o bus, de manera simultánea. Como resultado, se producen retrasos debido a la competencia por el acceso a este recurso compartido.

De esta manera buscaremos caracterizar la distribución más afín como aquella en la que se produce una menor contención. Dado que el mayor coste está asociado a los accesos a nodos no adyacentes, diseñaremos las pruebas seleccionando dos de estos nodos. Esto nos permitirá obtener una diferenciación más significativa entre los mejores y peores resultados. Hemos elegido específicamente los nodos 0 y 2 y, teniendo en cuenta las combinaciones posibles a la hora de distribuir hilos y datos entre los nodos, nos enfocaremos en aquellas que consideramos más representativas. Estas englobarán todas las combinaciones con repetición de los 3 recursos (bus QPI, bus y controlador de memoria y procesador), con excepción de las configuraciones en las que se comparte el bus de memoria y el bus QPI, o el bus QPI y el procesador únicamente, al no ser posibles en el problema a analizar.

Como resultando obtenemos un total de 6 configuraciones distintas a examinar para las cuales, tras realizadas las pruebas, se observó que las distribuciones más afines son aquellas donde cada pareja hilo-dato se encuentra en el mismo nodo. Así, expondremos las dos configuraciones más afines, emulando el comportamiento de OpenMP [17] en sus configuraciones *compact* y *scatter*.

B.1 *Compact*

En este caso, ubicamos tanto los hilos como los conjuntos de datos en un único nodo, ver figura 2.

Los resultados obtenidos se muestran en la tabla V, se aprecia un ligero incremento en el tiempo de ejecución de todas las operaciones, fluctuando entre una y varias décimas de segundo con respecto a los valores de las diagonales de las tablas II, III y IV.

Este comportamiento podría explicarse, en parte, porque ambos hilos están compartiendo y compitiendo por los recursos del procesador asociado al nodo. Sin embargo, dado que las operaciones realizadas son intensivas en accesos a memoria y no en cálculo, una contribución más significativa a este incremento puede atribuirse al hecho de que ambos hilos están accediendo a la memoria a través del mismo controlador y bus de datos.

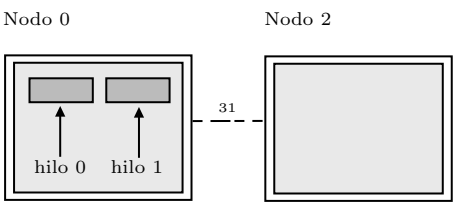


Fig. 2: Prueba *compact*.

Tabla V: Resultados prueba *compact* (segundos).

	R	**W**	**R-W**
Hilo 0	2,1509	19,2046	3,4267
Hilo 1	2,1705	19,6006	3,4447

B.2 *Scatter*

En este caso distribuimos los datos y los hilos entre los nodos de forma uniforme, colocando cada grupo hilo-dato en un nodo distinto, ver figura 3.

Los resultados obtenidos se muestran en la tabla VI, para todos los casos se observan unos valores muy similares a los de las tablas II, III y IV, dado que en esta ocasión no debería existir ningún tipo de interferencia entre hilos al estar empleando cada uno su controlador de memoria y su bus específico para el acceso a sus datos privados, así como un procesador distinto.

B.3 Recapitulación

De esta forma, la distribución más afín es aquella en la que los datos y el hilo que accede a ellos se

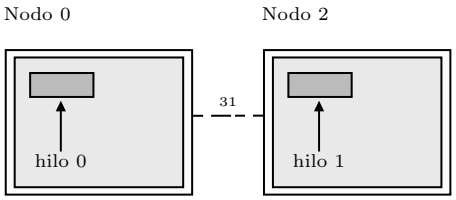


Fig. 3: Prueba *scatter*.

Tabla VI: Resultados prueba *scatter* (segundos).

	R	W	R-W
Hilo 0	2,0448	19,1014	3,3380
Hilo 1	2,0344	19,0986	3,3175

encuentran situados en el mismo nodo, con cada par hilo-dato asignado a su propio nodo independiente.

C. Dos hilos y accesos completamente solapados

En este caso, dos hilos accederán de forma simultánea a un mismo *array* compartido de manera que podamos observar el impacto sobre el tiempo de ejecución de elementos como el protocolo de coherencia caché.

Al distribuir dos hilos y un *array* de datos entre dos nodos, obtenemos 6 combinaciones posibles. Sin embargo, teniendo en cuenta que para cada combinación en un nodo existe una configuración simétrica en el otro nodo, únicamente realizaremos pruebas sobre una de ellas. De esta forma nos restringimos a 3 configuraciones distintas en las que:

1. Ambos hilos acceden a los datos de forma local a través del bus y el controlador de memoria de su nodo (caso Local, Local). Ver figura 4.
2. Ambos hilos acceden a los datos de forma remota a través del bus QPI (caso Remoto, Remoto), como se muestra en la figura 5.
3. Uno de los hilos tiene acceso al conjunto de datos localmente, mientras que el otro debe acceder a ellos de forma remota a través del canal QPI (caso Local, Remoto). Ver figura 6.

Dado que el estudio se centra en el comportamiento cuando existe compartición de datos entre hilos, es importante prestar atención a los potenciales efectos causados por la superposición en los accesos a los datos. Además del efecto de la distribución de los hilos y los datos, aspectos como la caché y los protocolos de coherencia pueden generar resultados relevantes en el análisis.

C.1 Caso LL (Local, Local)

En esta prueba observamos que, si ambos hilos acceden a datos locales y compartidos de forma simultánea, en comparación al caso *compact*, el coste de las operaciones de lectura y lectura-escritura sufre un leve incremento, mientras que en las operaciones de escritura se reduce en casi 1 segundo (ver tabla VII).

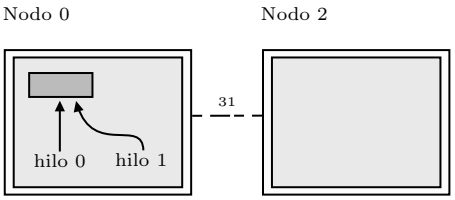


Fig. 4: Prueba LL.

Tabla VII: Resultados caso LL (segundos).

	R	W	R-W
Hilo 0	2,1874	18,2247	3,4637
Hilo 1	2,1862	18,2331	3,4588

C.2 Caso RR (Remoto, Remoto)

Los resultados de la tabla VIII muestran de nuevo, en comparación a los accesos remotos de un solo hilo, un incremento del coste en las lecturas y lecturas-escrituras, y una reducción en las escrituras.

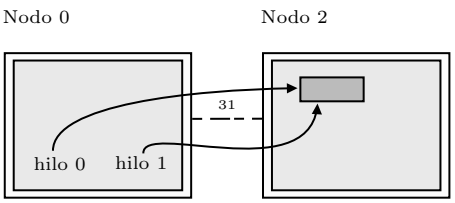


Fig. 5: Prueba RR.

Tabla VIII: Resultados caso RR (segundos).

	R	W	R-W
Hilo 0	6,1957	39,6130	6,9010
Hilo 1	6,1902	39,5395	6,8923

C.3 Caso LR (Local, Remoto)

En este caso, pese a que uno de los hilos se encuentre accediendo a los datos de forma local y otro acceda de forma remota, ambos poseen el mismo coste de acceso, llegando a sincronizarse a la milésima de segundo, como se muestra en la tabla IX.

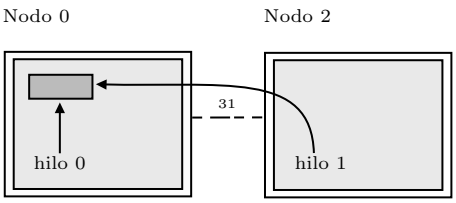


Fig. 6: Prueba LR.

Tabla IX: Resultados caso LR (segundos).

	R	W	R-W
Hilo 0	6,4975	40,4655	6,4060
Hilo 1	6,4977	40,4684	6,4061

C.4 Recapitulación

De esta forma, observamos cómo la distribución más afín es la asociada con el *caso LL*, donde ambos hilos acceden de forma local a sus datos.

Una vez determinada la configuración más afín, es necesario mencionar la sincronización que se está produciendo en todos los casos entre los dos hilos. En concreto, resalta el comportamiento del *caso LR*, donde el hilo 0, que accede a los datos de forma local, obtiene unos resultados 3 veces peores a los obtenidos con un solo hilo, sincronizándose con el hilo 1, que accede a los datos de forma remota.

Este fenómeno puede ser explicado a través de efectos producidos por el protocolo de coherencia caché. Según se detalló en la caracterización de la arquitectura QPI, esta permite un intercambio directo de líneas de caché entre nodos. Teniendo esto en cuenta, cuando existe una superposición en las lecturas, de acuerdo con las especificaciones del protocolo [9], podríamos esperar cuatro flujos de datos potenciales, determinados por qué hilo realiza el primer acceso a la memoria. Conociendo de antemano que el hilo 0 está ubicado en el nodo 0 y el hilo 1 en el nodo 2, estos comportamientos posibles son:

- **Casuística 1:** si el hilo 0 es el primero en acceder a los datos, estos residirán en su memoria local. Así, este hilo solicitará a su controlador la línea de datos correspondiente. A continuación, el controlador de memoria procederá a devolver la línea de datos solicitada.
- **Casuística 2:** en el caso de que el hilo 1 acceda a los datos en segundo lugar, estos se ubicarán en la caché del nodo 0. Bajo este escenario, el hilo 1 requerirá los datos del nodo 0. Seguidamente, el nodo 0 retornará los datos directamente desde su memoria caché a través del bus QPI.
- **Casuística 3:** si el hilo 1 es el primero en acceder a los datos, tendrá que solicitarlos al nodo 0. El nodo 0, al constatar que los datos no se encuentran en su caché, deberá hacer una petición al controlador para obtenerlos de su memoria local. Después de obtener los datos de la memoria, el nodo 0 finalmente enviará los datos al nodo 2 a través del bus QPI.
- **Casuística 4:** en el escenario donde el hilo 0 es el segundo en acceder a los datos, solicitará la línea correspondiente al nodo 2. A continuación, el nodo 2 remitirá la línea de datos desde su caché, utilizando el bus QPI para su transmisión.

Así, al analizar los comportamientos vinculados a las **casuísticas 1** y **3** vemos cómo, al requerir accesos a memoria, conllevan un mayor coste temporal que las **casuísticas 2** y **4**, donde se accede directamente a los datos desde caché a través de la red NUMA de QPI. Esto resulta en dos secuencias repetitivas de comportamientos en el acceso a los datos:

- Cuando el hilo 0 accede primero a los datos: en este escenario, la **casuística 1** ocurre en primer lugar, seguida por la **casuística 2**. Dado que la primera es sustancialmente más lenta que la segunda, llegará un punto en el que el orden de los accesos se invertirá, y será el hilo 1 el que acceda primero a los datos.
- Cuando el hilo 1 accede primero a los datos: en esta circunstancia, la **casuística 3** se da primero, seguida por la **casuística 4**. Nuevamente, debido a que la primera es notablemente más lenta que la segunda, habrá un momento en que el hilo 0 sobrepasará al hilo 1 y será el primero en acceder a los datos.

Estos dos comportamientos se entrelazan y alternan en la realización de los accesos, provocando que los hilos 0 y 1 se vayan adelantando uno al otro hasta que el *array* se haya recorrido por completo.

Este comportamiento explicaría la sincronización que se presenta en los accesos, especialmente notable en el *caso LR* donde produce efectos muy negativos para el hilo local. Bajo esta suposición, diseñamos nuevas pruebas con el objetivo de validar este comportamiento y proporcionar nuevos resultados con un grado de solapamiento diferente.

D. Dos hilos y accesos semi-solapados

En este caso, experimentaremos los efectos de reducir el acople entre los hilos sin afectar al número de operaciones realizadas. En este sentido, añadiremos un retardo de 0,5 s en el acceso del hilo 1 al vector con el objetivo de evitar la sincronización.

Añadiendo esta modificación, se repitieron las pruebas anteriores obteniendo los resultados que se muestran en las tablas X, XI y XII.

Tabla X: Resultados caso LL semi-solapado (segundos).

	R	**W**	**R-W**
Hilo 0	2,1678	18,9674	3,4663
Hilo 1	2,6643	19,5482	3,9626

Tabla XI: Resultados caso RR semi-solapado (segundos).

	R	**W**	**R-W**
Hilo 0	6,7216	40,3357	6,4034
Hilo 1	7,2230	40,6179	6,8949

Tabla XII: Resultados caso LR semi-solapado (segundos).

	R	**W**	**R-W**
Hilo 0	4,1735	39,6521	5,8246
Hilo 1	5,9891	40,8517	6,9050

En términos del tiempo de ejecución, observamos una vez más que la distribución más afín es aquella en la que el par hilo-dato se localiza en el mismo nodo. No obstante, de nuevo resulta de interés estudiar las variaciones que sufrieron el resto de pruebas.

Con el fin de facilitar la comprensión de los datos, estudiaremos los resultados en términos del *speedup* producido para cada uno de los casos (LL, RR y LR) entre las pruebas con accesos totalmente solapados

(ver tablas VII, VIII y IX) y las pruebas con accesos parcialmente solapados (ver tablas X, XI y XII). El *speedup* empírico de estos se indicarán con la letra (E) al lado de cada hilo. Además, se añadirá una fila con el *speedup* hipotético (H) que debería haber sufrido el hilo 1, al añadir el retraso de 0,5 s.

D.1 Caso LR

Los resultados se muestran en la tabla XIII. Se evidencia un incremento del rendimiento del hilo 0, justificable por la disminución de accesos solapados, lo cual aproxima este hilo al rendimiento que tendría en ausencia de intervención del hilo 1. Este incremento se observa en las operaciones de lectura-escritura, donde se obtiene una mejora de aproximadamente el 10 %, pero sobre todo destaca en las operaciones de lectura, donde obtenemos una mejora del 50 %. Si analizamos el comportamiento del hilo 1, vemos que para escrituras y lecturas-escrituras se obtiene un *speedup* muy similar al hipotético. Sin embargo, en las lecturas, en lugar de producirse un empeoramiento del rendimiento, se produce una mejora del 8 %. Este comportamiento puede ser explicado por una reducción del número de accesos a memoria, debido a que el hilo 0 se encuentra precargando datos en caché. De esta manera, dado que el hilo 0 comienza a iterar sobre el vector antes, cuando el hilo 1 alcanza las primeras posiciones, estas se encuentran en la caché del nodo 0. Así, pese al retardo introducido, los aciertos caché propician una ejecución más rápida.

Tabla XIII: *Speedup* caso LR.

	R	**W**	**R-W**
Hilo 0 (E)	1,5568	1,0205	1,0998
Hilo 1 (E)	1,0849	0,9906	0,9277
Hilo 1 (H)	0,9285	0,9877	0,9276

D.2 Caso RR

En este caso, en la tabla XIV podemos observar cómo en las operaciones de lectura se produce un comportamiento opuesto al observado en el *caso LR*. Ahora ambos hilos reducen su rendimiento, destacando sobre todo el hilo 1, que cae por debajo de la reducción hipotética. Este comportamiento puede ser explicado porque, al contrario que en el *caso LR*, ambos hilos se encuentran accediendo a datos remotos desde el mismo nodo origen. Por lo tanto, el factor dominante en el tiempo de ejecución es la latencia de acceso a memoria remota y no la posible sincronización debida al protocolo de coherencia caché. De esta manera, cuando se introduce el retraso en el hilo 1, si el hilo 0 se encuentra "demasiado" adelantado, el primero no podrá aprovecharse de los datos precargados por el segundo, ya que se habrán borrado de la caché. Así:

- El hilo 1: añadirá al sobrecoste del retardo, accesos extra a memoria principal.
- El hilo 0: deberá realizar todos los accesos a memoria principal y asumir posibles efectos asociados a la coherencia caché.

En términos de las escrituras, introducir retardo no provoca un cambio significativo en el comportamiento. En el caso de la operaciones de lectura-escritura, volvemos a obtener un patrón similar al de las lecturas del *caso LR*, donde romper la sincronización llega a ser beneficioso. El retardo introducido no resulta "demasiado" elevado, permitiendo que el hilo 1 se aproveche en cierta medida de la precarga y, el hilo 0 evite problemas de sincronización.

Tabla XIV: *Speedup* caso RR.

	R	**W**	**R-W**
Hilo 0 (E)	0,9217	0,9820	1,0777
Hilo 1 (E)	0,8570	0,9734	0,9996
Hilo 1 (H)	0,9252	0,9875	0,9323

D.3 Caso LL

Por último, en este caso, tal como se muestra en la tabla XV, podemos ver como en las lecturas y las lectura-escritura, no se producen prácticamente variaciones a los resultados teóricos esperables, mientras que en las escrituras se produce una reducción del rendimiento para ambos hilos.

Tabla XV: *Speedup* caso LL.

	R	**W**	**R-W**
Hilo 0 (E)	1,0090	0,9608	0,9992
Hilo 1 (E)	0,8205	0,9327	0,8728
Hilo 1 (H)	0,8138	0,9733	0,8736

D.4 Recapitulación

De esta manera vemos como, en algunos casos, esta reducción en el solapamiento se traduce en una mejora del rendimiento, por un lado acercando al hilo adelantado a una ejecución aislada sin intervención de otros hilos y por otro, reduciendo el coste del hilo atrasado gracias a los datos precargados por el anterior. Sin embargo, es importante determinar el grado de desincronización a utilizar: en caso de que primer hilo se adelante demasiado, el hilo atrasado no se verá beneficiado por la precarga y al coste de accesos a memoria se deberá añadir el sobrecoste del retardo introducido, así como posibles sobrecargas en la gestión de la coherencia de los datos.

IV. Conclusiones

En este trabajo, hemos analizado la influencia de varios factores en el rendimiento de un sistema NUMA conformado por cuatro procesadores Intel Xeon E5-4620 v4 interconectados en anillo mediante el bus Intel QPI. El objetivo principal ha sido identificar la configuración más eficiente en términos de distribución de hilos y datos sobre los nodos, así como de compartición de *socket* y solapamiento en el acceso a datos compartidos.

Utilizando programas de pruebas intensivos en accesos a memoria, hemos analizado distintas configuraciones y cuantificado su eficiencia en términos del tiempo de ejecución. Siguiendo este enfoque, las pruebas se diseñaron con el objetivo de obtener conclusiones claras y aplicables, que no estén oscurecidas por efectos de la precarga, la localidad espacial o diferentes optimizaciones del compilador, facilitando la replicación y ampliación de los experimentos.

Hemos constatado que, para una única pareja hilo-nodo, la distribución más eficiente es aquella en la que el hilo y los datos se ubican en el mismo nodo, garantizando accesos a memoria locales. En situaciones donde dos hilos acceden a su propio conjunto de datos privados, se ha encontrado que la distribución más eficiente es aquella en la que cada pareja hilo-datos posee un nodo exclusivo, evitando la compartición de recursos y la contención de bus.

Por otro lado, hemos comprobado que, cuando dos hilos en ejecución acceden al mismo conjunto de datos, los resultados se ven notablemente influenciados por el solapamiento de los accesos. En escenarios de accesos simultáneos los hilos tienden a sincronizarse, limitando su rendimiento al del hilo más lento. Sin embargo, si se introduce un retraso en uno de los hilos, se observa una mejora en el rendimiento del hilo adelantado, mientras que el hilo atrasado puede beneficiarse del trabajo previo del primero, siempre y cuando los datos en la caché no se reemplacen.

De cara a aumentar la completitud del estudio, sería de interés; primero, explorar los efectos de las distintas configuraciones *core*/hilo; segundo, aumentar el conjunto de pruebas empleando instrucciones vectoriales (SIMD); tercero, ampliar los tipos de datos empleados en las pruebas más allá de los `char` (`int`, `float`, `double`...); y cuarto, realizar grupos de pruebas con *benchmarks* de aplicaciones, modelando así comportamientos en situaciones cercanas a ejecuciones reales.

Es importante destacar que estas observaciones son específicas para la arquitectura y el protocolo de coherencia de caché utilizados en este estudio, y podrían variar con otras configuraciones. A pesar de ello, los hallazgos aquí presentados proporcionan un valioso punto de partida para futuras investigaciones sobre la optimización del rendimiento en sistemas NUMA.

Agradecimientos

Este trabajo ha recibido el apoyo económico del Consellería de Cultura, Educación y Ordenación Universitaria de la Xunta de Galicia (acreditaciones ED431C 2022/16 y ED431G 2019/04) y el Fondo Europeo de Desarrollo Regional (FEDER), que reconoce al CiTIUS de la Universidad de Santiago de Compostela como Centro de Investigación del Sistema Universitario de Galicia. Este trabajo también fue apoyado por el Ministerio de Economía y Competitividad del Gobierno de España (Proyecto PID2019-104834GB-I00).

Referencias

[1] Doug Matzke, "Will physical scalability sabotage performance gains?," *Computer*, vol. 30, no. 9, pp. 37–39, 1997.

[2] Sergey Blagodurov, Sergey Zhuravlev, Mohammad Dashti, and Alexandra Fedorova, "A Case for NUMA-aware Contention Management on Multicore Systems," in *USENIX SEDMS*, 2011.

[3] Nakul Manchanda and Karan Anand, "Non-uniform memory access (NUMA)," New York University, 2004.

[4] David Padua, *Encyclopedia of Parallel Computing*, Springer New York Dordrecht Heidelberg London Prentice Hall, 2011.

[5] Baptiste Lepers, Alexandra Fedorova, and Vivien Quema, "Thread and memory placement on numa systems: Asymmetry matters," Usenix, `https://www.usenix.org/system/files/conference/atc15/atc15-paper-lepers.pdf`, Consultado el 20 de mayo de 2023.

[6] Daniel Goodman, Roni Haecki, and Tim Harris, "Modeling memory bandwidth patterns on NUMA machines with performance counters," arXiv, `https://arxiv.org/pdf/2106.08026.pdf`, Consultado el 21 de mayo de 2023.

[7] Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quema, and Mark Roth, "Traffic management: A holistic approach to memory placement on NUMA systems," *SIGPLAN Not.*, vol. 48, no. 4, pp. 381–394, mar 2013.

[8] A. Kleen, "A NUMA API for Linux," *Novel Inc*, 2005.

[9] "An introduction to the Intel, QuickPath Interconnect," `https://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html`, 2009.

[10] "Xeon processor scalable family technical overview," Intel, `https://www.intel.com/content/www/us/en/developer/articles/technical/xeon-processor-scalable-family-technical-overview.html`, Consultado el 14 de mayo de 2023.

[11] "cv (const and volatile) type qualifiers," Documentación de cppreference, `https://en.cppreference.com/w/cpp/language/cv`, 2023.

[12] Linux man-pages project, "NUMA(3) - Linux programmer's manual," `https://www.man7.org/linux/man-pages/man3/numa.3.html`, Consultado el 10 de mayo de 2023.

[13] Linux man-pages project, "SCHED_SETAFFINITY(2) - Linux programmer's manual," `https://man7.org/linux/man-pages/man2/sched_setaffinity.2.html`, Consultado el 17 de mayo de 2023.

[14] Ulrich Drepper, "What every programmer should know about memory," `https://akkadia.org/drepper/cpumemory.pdf`, Consultado el 16 de mayo de 2023.

[15] "Intel Xeon Processor E5 v4 Product Family Datasheet Volume 2 - Registers," `https://en.wikichip.org/w/images/2/24/Intel_Xeon_Processor_E5_v4_Product_Family_Datasheet_Volume_2-_Registers.pdf`, Consultado el 16 de mayo de 2023.

[16] T Konstantakopoulos, Jonathan Eastep, James Psota, and Anant Agarwal, "Energy scalability of on-chip interconnection networks in multicore architectures," MIT CSAIL, `http://groups.csail.mit.edu/cag/raw/documents/Konstantakopoulos-Energy-2007.pdf`, Consultado el 22 de mayo de 2023.

[17] Leonardo Dagum and Ramesh Menon, "OpenMP: an industry standard API for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[18] "NAS Parallel Benchmarks," NASA Advanced Supercomputing (NAS) Division, `https://www.nas.nasa.gov/software/npb.html`, 2023.

[19] "SPEC OMP 2012," Standard Performance Evaluation Corporation, `http://www.spec.org/omp2012/`, 2023.

[20] "STREAM," Department of Computer Science, `https://www.cs.virginia.edu/stream/ref.html`, 2023.

Generación de imágenes de intensidad a partir de imágenes RGB con instrucciones de vectorización

Roberto Díaz-Cano[1], Enrique S. Quintana-Ortí[2] y Pedro Alonso-Jordá[3]

Resumen— **El *software* de visión/expulsión MVS de la empresa Multiscan Technologies controla el funcionamiento de las máquinas que produce esta compañía. Este programa ha sido desarrollado y mejorado con el paso de los años por el personal de la empresa. Sin embargo, los requerimientos computacionales de los procesos de selección de producto en las máquinas actuales han creado la necesidad de mejorar el software para incrementar su eficiencia. El *software* MVS organiza el análisis de las imágenes captadas por las cámaras como un flujo de proceso (o *pipeline*), explotando el paralelismo entre las tareas en las que se organiza este flujo. Las tareas son numerosas y comprenden multitud de funciones, pero el presente trabajo se centra en la aceleración con instrucciones de vectorización del algoritmo de procesamiento de imágenes.**

Palabras clave— **Aceleración de funciones, vectorización, computación de alto rendimiento, procesamiento de imágenes, visión artificial.**

I. Introducción

El presente trabajo se centra en explicar las tareas desarrolladas en el seno de una empresa para identificar y mejorar el tiempo de respuesta de una aplicación de clasificación de objetos contenidos en una secuencia de imágenes. Para ser más precisos, se detalla la implementación de un algoritmo de conversión mediante el uso de instrucciones de vectorización.

A. Contexto del trabajo

El trabajo se ha realizado mediante una colaboración con la empresa Multiscan Technologies situada en Cocentaina. Esta empresa se encarga de desarrollar y fabricar equipos de visión artificial para la selección e inspección de productos alimentarios. Por ello, es necesario que sus equipos tengan un tiempo de respuesta pequeño para tener una alta capacidad de procesado.

Para conseguir que la aplicación tenga unos tiempos de respuesta aceptables la empresa decidió realizar una colaboración con el Departamento de Informática de Sistemas y Computadores (DISCA) para asesorar e identificar todos los puntos que provocan el bajo rendimiento y encontrar una solución que permita reducir el tiempo de respuesta de su aplicación de clasificado. De esta forma, se decidió realizar este trabajo, que se encarga de identificar los distintos problemas de rendimiento e intenta mejorarlos aplicando técnicas de optimización para el cálculo sobre la Unidad Central de Procesamiento (CPU).

B. Motivación

En los últimos años, ha habido un aumento considerable de la demanda de soluciones que sean capaces de actuar de acuerdo con los estímulos obtenidos a través de diferentes dispositivos, como imágenes, archivos de sonido, etc. Este aumento de la demanda ha estado motivado, en parte, por los grandes avances que se han llevado a cabo a lo largo de estos años en el campo de la Visión Artificial.

La Visión Artificial [1] persigue el objetivo de identificar el entorno para actuar de acorde a él. Para conseguirlo, normalmente las máquinas capturan el entorno mediante imágenes, archivos de sonido, etc. Después, procesan estos datos y actúan en consecuencia. Alguna de sus aplicaciones sería la identificación de objetos que hay dentro de una imagen (ver Figura 1). Empresas como Multiscan Technologies aplica esta tecnología para identificar objetos y realizar una acción en consecuencia.

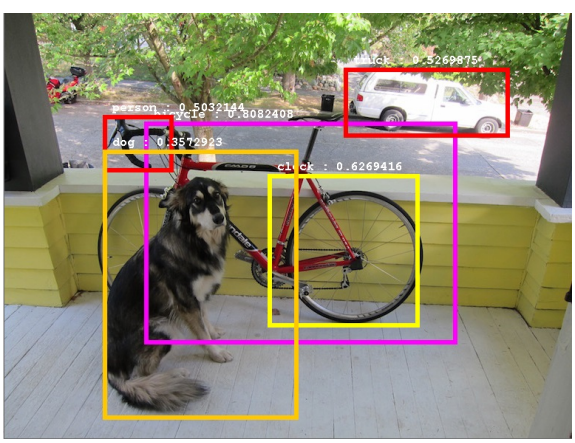


Fig. 1: Identificación de los distintos objetos que se muestran en una imagen. Como se puede observar en este ejemplo a partir de una imagen, el algoritmo intenta identificar y etiquetar todos los objetos presentes en la imagen (obtenida en el siguiente enlace https://www.inspiritai.com).

Multiscan Technologies[1] es una empresa dedicada al diseño y fabricación de maquinaria de inspección y clasificación para la industria alimentaria. Sus soluciones permiten la clasificación y la identificación de defectos en productos agroalimentarios como naranjas, tomates, aceitunas, etc. En funcionamiento,

[1]Dpto. I+D, Multiscan Technologies, e-mail: rdiaz@multiscan.eu
[2]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: quintana@disca.upv.es
[3]Dpto. de Sistemas Informáticos y Computación, Universitat Politècnica de València, e-mail: palonso@upv.es

[1]https://multiscan.eu/

sus máquinas obtienen imágenes de los frutos u objetos que pasan por la cinta transportadora para, después, identificar los frutos de la imagen y sus defectos mediante su aplicación Multiscan Visual System (MVS). En función de este análisis, la aplicación genera una serie de órdenes que permiten clasificar el fruto de acuerdo a unos criterios.

Para realizar la identificación y clasificación de los objetos de cada imagen, es necesario un tiempo de cómputo que, a veces, supera el tiempo de respuesta máximo que tiene la máquina. Por este motivo, es esencial reducir al máximo posible el coste computacional del análisis de las imágenes con objeto de obtener unas prestaciones óptimas en las máquinas de clasificación.

C. Objetivos

El objetivo general del presente trabajo consiste en solucionar uno de los cuellos de botella detectado en el flujo de procesos de MVS.

El cuello de botella abordado en este trabajo se encuentra en algoritmo encargado de la obtención de la intensidad del espacio de color HSI de una imagen a partir de una imagen en el espacio de color RGB. Se plantea el uso de la vectorización para reducir su coste temporal y, por tanto, mejorar el rendimiento de toda la aplicación.

II. Entorno de trabajo

Multiscan Technologies puso a disposición un entorno trabajo para la realización de las tareas pertinentes. En este apartado se detallan las características de cada uno de los componentes de este entorno.

A. Característica del sistema proporcionado

La empresa puso a disposición un equipo para que se realizaran los análisis y modificaciones, con el objetivo de acelerar el flujo de procesos de su aplicación.

El dispositivo prestado se caracteriza por ser un ordenador con unas prestaciones similares a los utilizados en las máquinas de la empresa. Este ordenador consta de un procesador Intel Core I7 de octava generación. Las características más apreciables son:

- Seis núcleos físicos, donde cada uno de ellos puede ejecutar dos hilos.
- Frecuencia máxima del procesador de 3,20 GHz.
- Memoria caché de tres niveles con un tamaño del nivel L3 de 12 MB, 1,5 MB para el nivel L2, 384 KB para el nivel L1 de datos y otra L1 de instrucciones con una capacidad de almacenamiento similar.
- El dispositivo es compatible con las instrucciones *Streaming SIMD Extensions* (SSE) versión 4.1 y 4.2; *Advanced Vector Extensions* (AVX) y *Advanced Vector Extensions* 2 (AVX2).

Por otro lado, también cabe destacar que el dispositivo cuenta con dos memorias RAM DDR4 de 8 GB cada una, sumando un total de 16 GB.

Por último, el equipo tiene instalado el sistema operativo Windows 10 Enterprise y para el entorno de programación se utiliza el Visual Studio 2017, el lenguaje C++ versión 17 y el compilador Microsoft Visual C++ (MSVC). Además, los *flags* de optimización empleados para la compilación de MVS fueron:

- `/O2`: Compilador intenta reducir el tamaño del código y del tiempo de ejecución. Igualmente, realiza casi todas las optimizaciones soportadas que no implican un compromiso espacio-velocidad.
- `/fp:precise`: Especifica cómo el compilador debe tratar las expresiones en coma flotante, las optimizaciones y las excepciones. En este caso, los resultados obtenidos en números de coma flotante serán precisos.
- `/arch:AVX2`: Especificación de la arquitectura para permitir el uso de instrucciones Intel Advanced Vector Extensions 2.
- `/Oi`: Ayuda a una ejecución más rápida, sustituyendo algunas llamadas a funciones por formas intrínsecas o especiales.

B. Multiscan Visual System

Multiscan Visual System (MVS) es una aplicación que se caracteriza por analizar las imágenes de entrada que recibe y clasificar los objetos que hay contenidos en estas. Principalmente, la aplicación está diseñada para la detección de frutas, sus elementos y sus defectos.

Para el análisis de las imágenes y la detección de los objetos se sigue una ejecución secuencial de grupos de procesos (o etapas) que permite, paso a paso, obtener la clasificación e identificación del objeto. A esta ejecución se le llama flujo de procesos.

El flujo de procesos recibe como entrada una secuencia de imágenes (ver Figura 2) correspondientes a los frutos u objetos que están pasando por la cinta.

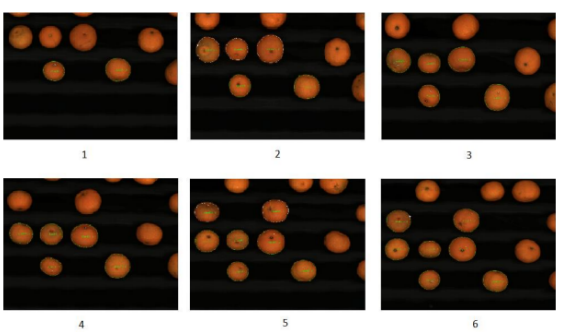


Fig. 2: Secuencia de imágenes de naranjas pasando por la cinta transportadora de la máquina de clasificación.

Además, cabe recalcar que la aplicación tiene diversas utilidades implementadas. Entre ellas hay que destacar:

- La visualización de la salida del flujo de procesos, independientemente de las etapas que contenga.
- La obtención de los costes temporales de cada etapa del flujo de procesos. Mediante métricas como: el tiempo de ejecución de cada uno, el tiempo máximo obtenido, la media, etc.

III. Preproceso de imágenes

En MVS el preproceso de las imágenes consiste en transformar imágenes de formato crudo o *RAW*. Este formato de archivo digital contiene la totalidad de los datos de la imagen. A partir de esta imagen, el algoritmo obtiene la imagen RGB (*Red*, *Green*, *Blue*). Este proceso se hace con un filtro de Bayer [2] que se encarga de aplicar una matriz de filtro de color para organizar los píxeles de la imagen y crear un patrón (ver Figura 3).

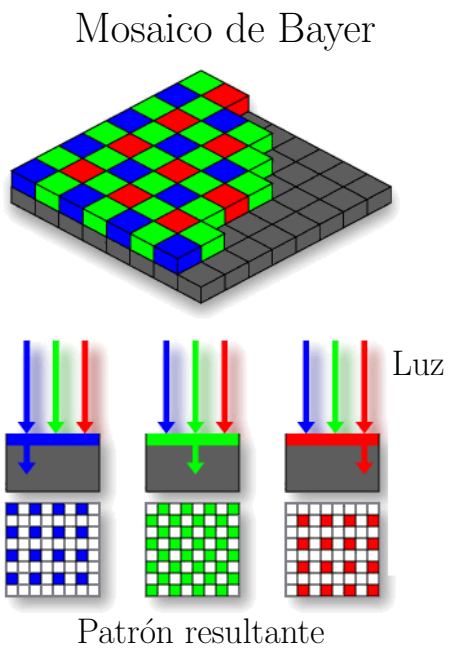


Fig. 3: A partir de una entrada se aplica un filtro que permite organizar los píxeles y crear un patrón o mosaico, es decir, la imagen de entrada a color.

Una vez se obtiene la imagen en RGB, tiene lugar la ejecución de otra técnica que genera la componente I (la intensidad, del espacio de color HSI) a partir de la imagen RGB, que será utilizada por el resto de procesos de la aplicación MVS.

Las dos transformaciones descritas forman parte de la etapa de preproceso y presentan el coste más elevado de todo el flujo de procesos. También hay que destacar que su correcto funcionamiento es crítico porque los demás procesos del flujo dependen de su correcta salida.

De los dos algoritmos se describe en este trabajo el segundo, es decir, la conversión de RGB a la intensidad del espacio de color HSI. Esta decisión viene dada porque el primer algoritmo se ha acelerado mediante una función de conversión de OpenCV, pero esto mismo no ha sido posible en el caso de la conversión a escala de grises dadas sus características. Para ser más precisos, no se ha efectuado con OpenCV porque la función que lleva a cabo este procedimiento realiza una operación de cálculo distinta a la implementada en MVS.

A. Algoritmo

El objetivo principal del algoritmo es la conversión de una imagen en el espacio de color RGB a la intensidad del espacio de color HSI. Para realizar esta conversión, hay distintas formas con las que se puede obtener el mismo resultado, pero en este caso concreto la operación que se efectúa es:

$$\text{Imagen[i]} = 2 * G[i] + R[i] + B[i],$$

donde *Imagen* hace referencia a la imagen resultante, i a la posición o píxel de la imagen dentro del recorrido, R al píxel que corresponde al color *Red*, G al color *Green* y B al color *Blue*. Por tanto, esta transformación consiste en un recorrido de toda la imagen aplicando la operación a cada posición o píxel.

El algoritmo tiene la particularidad de realizar la misma operación en cada píxel de la imagen sin ninguna comprobación de los resultados u operación condicional. Por ende, este algoritmo presenta unas condiciones favorables para el uso de instrucciones de vectorización como técnica de aceleración.

B. Formato y tipo de datos

El argumento de entrada del algoritmo original era una imagen, donde la información almacenada era de tipo `char`, pero a la hora de operar con sus píxeles, estos se transformaban "automáticamente" y se almacenaban como datos de tipo `unisgned long`. Por otro lado, los valores de R, G y B se distribuían en una estructura de memoria de cuatro elementos para cada posición. Es decir, la imagen tiene una estructura de memoria de una matriz de dos dimensiones, donde, en cada posición, hay un vector de cuatro elementos y, en cada uno de ellos, se almacena los valores de B, R, G y alpha en ese orden (ver Figura 4). Además, se vio que el tamaño de cada elemento de este vector era de 8 *bits*.

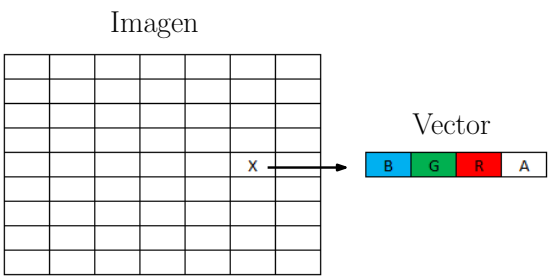


Fig. 4: La imagen que se representa es una estructura de matriz de dos dimensiones. En cada posición de la matriz se almacena un vector de cuatro elementos con la distribución de los valores B, R, G y alpha.

C. Vectorización

Las instrucciones vectorizadas permiten convertir una implementación escalar, que procesa un único par de operandos a la vez, en una implementación vectorial, que procesa una operación sobre múltiples operandos a la vez. Por ejemplo, ejecutar una multiplicación de dos vectores de tipo `float` con 8 elementos cada uno implica realizar un recorrido de los vectores y efectuar la operación con cada uno de los elementos. En cambio, con el uso de las instrucciones vectoriales solo haría falta una instrucción para ejecutar todas las operaciones.

Para la realización del algoritmo original con instrucciones de vectorización [3], [4] se dividió el código en diversas secciones y se comprobó que los resultados obtenidos de dichas instrucciones eran los correctos. También se analizó que la estructura de datos era la idónea para obtener los resultados en el formato correcto y mantener la compatibilidad con el resto de algoritmos del *software* MVS.

Este apartado se divide en diversos subapartados explicando en cada uno de ellos las tareas realizadas y el porqué de las mismas. Por otra parte, cabe destacar que, para comprobar que los resultados que se obtienen en cada paso eran los correctos, se implementaron unas instrucciones auxiliares para identificar si la salida obtenida se correspondía con la original.

C.1 Carga de valores y desenlazado

Los valores con los que se opera son de 8 *bits* y están almacenados secuencialmente. Para obtener los valores de un píxel o posición hay que efectuar una carga de 4 valores [5]. Por ello, para realizar las cargas se decidió ejecutar cuatro cargas de 128 *bits*, con lo que se almacenan 16 elementos que corresponden a 4 píxeles por vector. En total, se cargan 16 píxeles por cada iteración del recorrido de la imagen.

Los datos de los vectores tienen los colores entrelazados, por lo que es necesario efectuar alguna operación para desenlazar estos elementos y conseguir cuatro vectores donde en cada uno de ellos solo haya los valores de un único color. En otras palabras, se persigue disponer de un vector con los elementos de B, otro con los de R, un tercero con los de G y un último con los de Alpha. Para lograr este objetivo, se utilizaron las instrucciones de AVX2 `_mm_unpacklo_epi8()` y `_mm_unpackhi_epi8()` [6], que permiten entrelazar los datos de la parte baja o alta de los dos vectores de uno en uno. Con estas instrucciones se distribuyeron los valores entre distintas estructuras de memoria hasta conseguir tener en cada vector los valores correspondientes a un único color (ver Figura 5). Para conseguir tener los colores separados es necesario ejecutar estas instrucciones cuatro veces, ya que los vectores son de 16 elementos.

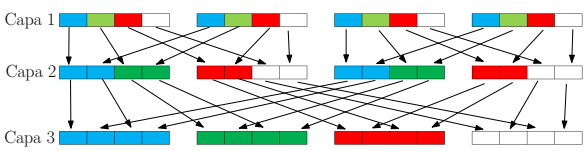


Fig. 5: Ejemplo básico de la aplicación de las instrucciones `_mm_unpacklo_epi8()` y `_mm_unpacklo_epi8()`. En la Capa 1 se visualizan cuatro vectores de cuatro elementos, es decir, azul (B), verde (G), rojo (R) y blanco (Alpha). En la Capa 2 se aplican las instrucciones para desenlazar los valores y se obtiene una distribución intermedia. Con la última aplicación de esta instrucción se consigue reunir cada color en un único vector.

Por último, cabe destacar que al realizar cargas de 16 elementos a la vez, el recorrido de las filas de la matriz no se lleva a cabo de 1 en 1, sino de 16 en 16. Por ello, se ha calculado cuántos elementos

```
int iter = (finalFila - inicioFila - 1) / 16;

for (int y = principioP; y < finalP; y++) {
    [...]
    for(int i = 0; i < iter; i++){
        [...]
            //Sección de carga
        __m128i _rgb1 = _mm_loadu_si128((__m128i
            *) pRGB);
        __m128i _rgb2 = _mm_loadu_si128((__m128i
            *) (pRGB + 4));
        __m128i _rgb3 = _mm_loadu_si128((__m128i
            *) (pRGB + 8));
        __m128i _rgb4 = _mm_loadu_si128((__m128i
            *) (pRGB + 12));
            //Sección de desenlazado
            //Primera capa
        __m128i layer11 = _mm_unpacklo_epi8(_rgb
            1, _rgb2);
        __m128i layer12 = _mm_unpackhi_epi8(_rgb
            1, _rgb2);
        __m128i layer13 = _mm_unpacklo_epi8(_rgb
            3, _rgb4);
        __m128i layer14 = _mm_unpackhi_epi8(_rgb
            3, _rgb4);
            //Segunda capa
        [...]
            //Tercera capa
        [...]
            //Cuarta capa
        __m128i b = _mm_unpacklo_epi8(layer31,
            layer33); //B
        __m128i g = _mm_unpackhi_epi8(layer31,
            layer33); //G
        __m128i r = _mm_unpacklo_epi8(layer32,
            layer34); //R
        [...]
    }
    //Recorrido original
    [...]
}
```

Fig. 6: Carga y desenlazado de los elementos del plano.

se han de recorrer en cada fila para que, en caso de este no sea divisible por 16, se ejecute el resto con la implementación inicial. La Figura 6 ejemplifica lo descrito anteriormente.

C.2 Conversión y separación de valores

Con los valores de cada color separados cada uno en un vector distinto, era necesario convertir los valores a `unsigned long` para operar correctamente. Para lograr la conversión se utilizó la instrucción `_mm256_cvtepu8_epi16()`. Esta convierte los datos de 8 *bits* en datos de 16 *bits*, lo que implica que el tamaño de los vectores cambia de 128 a 256 *bits*.

Con los valores en un formato que permitía operar con ellos se observó que los vectores tenían entrelazada la parte alta y baja. En otras palabras, las posiciones pares del vector pertenecían a la parte alta de la solución (los ocho primeros elementos), mientras que las posiciones impares correspondían a la parte baja (los ocho últimos elementos) (ver Figura 7). Para solucionar este inconveniente se decidió separar cada vector que contenía los valores de un color en dos, para almacenar la parte alta y la parte baja por separado.

Para conseguir separar los elementos que estaban entrelazados, se decidió crear un vector con las posiciones pares a cero y otro con las posiciones impares a cero. De esta forma, se tiene un vector con la parte alta y otro con la baja. Para conseguirlo se aplica una operación de `and` lógico bit a bit entre el vector a color y otro con las posiciones que se desean a cero.

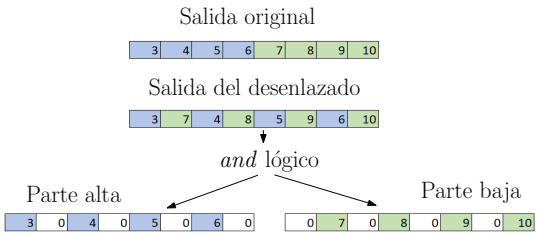


Fig. 7: Ejemplo de la salida que se debería obtener respecto a la salida obtenida del desenlazado de los valores. Con la aplicación de la instrucción `and` se consigue separar las dos partes del vector obteniendo como resultado dos vectores, cada uno de ellos con una parte del vector original.

```
int iter = (finalFila - inicioFila - 1) / 16;

const __m256i K16_1010 = _mm256_set_epi16(255,
     0, 255, 0, [...], 0, 255, 0);
const __m256i K16_0101 = _mm256_set_epi16(0,
     255, 0, 255, [...], 255, 0, 255);

for (int y = principioP; y < finalP; y++) {
    [...]
    for(int i = 0; i < iter; i++){
        [...]
            //Sección de carga
        [...]
            //Sección de desenlazado
        [...]
            //Sección de conversion
        __m256i b16 = _mm256_cvtepu8_epi16(b);
        __m256i g16 = _mm256_cvtepu8_epi16(g);
        __m256i r16 = _mm256_cvtepu8_epi16(r);
            //Sección de separación
        __m256i b16_lo = _mm256_and_si256(b16, K
             16_1010);
        __m256i b16_hi = _mm256_and_si256(b16, K
             16_0101);
        __m256i g16_lo = _mm256_and_si256(g16, K
             16_1010);
        __m256i g16_hi = _mm256_and_si256(g16, K
             16_0101);
        __m256i r16_lo = _mm256_and_si256(r16, K
             16_1010);
        __m256i r16_hi = _mm256_and_si256(r16, K
             16_0101);
        [...]
    }
}
```

Fig. 8: Conversión y desenlazado de los valores de cada vector.

De este modo, al aplicar la operación `and` se obtienen los vectores deseados. En la Figura 8 se muestra una ampliación de lo anterior con los vectores creados y las instrucciones de conversión y `and`.

C.3 Operaciones

Una vez se disponía de los valores compatibles con las operaciones y la separación o desenlazado de los vectores de color, se procedió a la implementación de las instrucciones que permiten realizar las operaciones de suma y multiplicación. El algoritmo realiza el siguiente cálculo:

$$\text{Imagen[i]} = 2 * G[i] + R[i] + B[i]$$

Para efectuar la multiplicación se utiliza una instrucción de desplazamiento determinado de *bits* a la izquierda: `_mm256_slli_epi16()`. En este caso, como la multiplicación es por dos, se realiza un desplazamiento de *bits* a la izquierda de una posición.

Después, para sumar los valores y almacenar la salida en un vector resultado se utiliza la instrucción

```
int iter = (finalFila - inicioFila - 1) / 16;

const __m256i K16_1010 = _mm256_set_epi16(255,
     0, 255, 0, [...], 0);
const __m256i K16_0101 = _mm256_set_epi16(0,
     255, 0, 255, [...], 255);

for (int y = principioP; y < finalP; y++) {
    [...]
    for(int i = 0; i < iter; i++){
        [...]
            //Sección de carga
        [...]
            //Sección de desenlazado
        [...]
            //Sección de conversion
        [...]
            //Sección de separacion
        [...]
            //Operacion: 2 * G
        __m256i g2_lo = _mm256_slli_epi16(g16_lo
             , 1);
        __m256i g2_hi = _mm256_slli_epi16(g16_hi
             , 1);
            //Operacion: 2 * G + B + R
        __m256i suma_lo = _mm256_add_epi16(b16_
             lo, _mm256_add_epi16(g2_lo, r16_lo)
             );
        __m256i suma_hi = _mm256_add_epi16(b16_
             hi, _mm256_add_epi16(g2_hi, r16_hi)
             );
        [...]
    }
}
```

Fig. 9: Operación de desplazamiento y suma de los vectores.

`_mm256_add_epi16()`. En la Figura 9 se presenta una actualización de las figuras anteriores donde se añade la instrucción para sumar y efectuar el desplazamiento a la izquierda.

Cabe destacar que se han utilizado las instrucciones que trabajan sobre 16 *bits*, ya que la conversión efectuada anteriormente aumentaba el tamaño de 8 a 16 y es necesario operar en este rango para conseguir los resultados deseados.

C.4 Permutación y almacenamiento de datos

El siguiente paso es unir la parte alta y la baja en un solo vector para obtener la correcta disposición de los datos. Para volver a juntar las dos partes en una sola, se utiliza la instrucción `_mm256_madd_epi16()`, la cual desplaza los datos una posición a la izquierda, es decir, en vez de tener en la primera posición del vector un 0, tener el primer elemento calculado. Por ello, se utiliza esta instrucción para mover los datos, ya que se suma el vector solución con un vector auxiliar rellenado de unos, obteniendo como resultado el mismo vector, pero en la posición anterior a la que estaban inicialmente. Por otro lado, también se añade la instrucción `_mm256_packs_epi32()` para unir la parte baja y la alta en un único vector y eliminar todos los ceros de ambos vectores.

Posteriormente, se añadió una instrucción para almacenar los valores resultantes en la estructura de datos correspondiente. Con la instrucción `_mm256_storeu_si256()` se carga el vector entero en la estructura de memoria a la que apunta el puntero.

Por último, se comprobó la salida resultante de todo el algoritmo implementado y se detectó que los 8 elementos centrales del vector resultado tenían un orden distinto respecto al original. En otras palabras,

si se divide los 16 elementos en bloques de 4 elementos, los dos bloques centrales no correspondían con los elementos originales, pero si se cambiaba el orden de estos dos, sí que era idéntico (ver Figura 10). Por tanto, era necesario aplicar una permutación para cambiar de posiciones estos bloques. Para ello, se utiliza la instrucción `_mm256_permute4x64_epi64()`.

Para realizar la permutación deseada hay que añadir a la instrucción un valor en formato hexadecimal. En este caso, como se quiere cambiar los dos bloques centrales, se añade el valor `0xD8`, que en binario es `11 01 10 00` y en decimal es `3 1 2 0`. Con esto se indica que los valores que estén en la posición 1 deben estar antes que los valores que están en la posición 2.

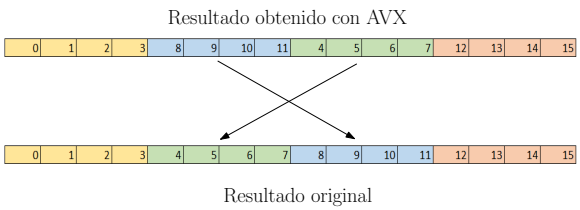


Fig. 10: Las dos estructuras simulan un vector de dieciséis posiciones. El primero representa la salida obtenida con el algoritmo vectorizado y el segundo representa la salida del algoritmo original. Como se aprecia, los dos bloques centrales están en las posiciones intercambiadas respecto a la salida original. Por ello, es necesario permutar estos dos bloques para conseguir la salida idéntica.

En la Figura 11 se muestra la actualización de las figuras anteriores con las instrucciones descritas en este apartado.

Finalmente, con esta última implementación se completaron todas las modificaciones para acelerar la conversión de RGB a escala de grises.

C.5 Salida obtenida

Una vez finalizado el algoritmo vectorizado y comprobado su salida en todas las etapas descritas, se utilizó la aplicación MVS para identificar que la salida era correcta e idéntica a la original y que no afectaba a ninguno de los procesos que utilizaban dicha salida (ver Figura 12). Además, se tomó una imagen de la secuencia y se efectuó la conversión con el algoritmo original y el vectorizado. Después, se comparó píxel a píxel el resultado de ambos algoritmos, con el objetivo de saber, de forma precisa, que los resultados del algoritmo acelerado eran idénticos al original.

IV. Resultados obtenidos

Para la obtención del coste temporal del algoritmo, antes y después de las modificaciones efectuadas, se ha empleado la aplicación con una configuración específica. La configuración se caracteriza por tener dos cámaras y los algoritmos descritos procesan las imágenes de ambas cámaras de forma paralela, es decir, se procesan las imágenes de cada cámara a la vez. En MVS hay implementada una ventana que indica, en tiempo real, el coste temporal de las distintas etapas que forman el análisis de las imágenes.

```
int iter = (finalFila - inicioFila - 1) / 16;

const __m256i K16_1010 = _mm256_set_epi16(255,
    0, 255, 0, [...], 0, 255, 0);
const __m256i K16_0101 = _mm256_set_epi16(0,
    255, 0, 255, [...], 255, 0, 255);
const __m256i K16_1111 = _mm256_set_epi16(1, 1,
    [...], 1);

for (int y = principioP; y < finalP; y++) {
    [...]
    for(int i = 0; i < iter; i++){
        [...]
            //Sección de carga
        [...]
            //Sección de desenlazado
        [...]
            //Sección de conversion
        [...]
            //Sección de separacion
        [...]
            //Operacion: 2 * G
        [...]
            //Operacion: 2 * G + B + R
        [...]
            //Desplazamiento de una posicion a
                la izquierda
        __m256i lo = _mm256_madd_epi16(suma_lo,
            K16_1111);
            //Union y permutacion de los datos
        __m256i gray = _mm256_permute4x64_epi
            64(_mm256_packs_epi32(suma2_hi, lo)
            , 0xD8);
            //Almacenamiento de los datos en el
                plano resultante
        _mm256_storeu_si256((__m256i*)pR, gray);
            //Avance de posiciones
        pR += 16;
        pRGB += 16;
    }
}
```

Fig. 11: Unión, permutación y almacenamiento de los datos en la imagen resultante.

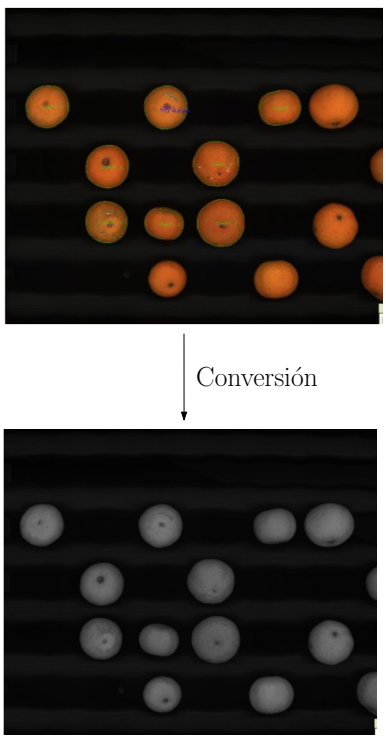


Fig. 12: Plano resultante de la implementación del algoritmo de conversión utilizando la vectorización. En esta imagen se representa la fruta de la cinta transportadora.

El procedimiento para obtener los costes deseados y compararlos consiste en utilizar una secuencia de imágenes para que la aplicación las analice. Una vez

finalizado el procesamiento de la secuencia se obtienen las métricas temporales de cada versión. En este caso, en la Tabla I se presenta el coste temporal de la etapa de preproceso original y la misma etapa con el algoritmo vectorizado.

Tabla I: Tiempos obtenidos con la implementación de la vectorización de la conversión de una imagen RGB a una de escala de grises con las imágenes de una la Cámara 1. Se muestra el tiempo total, la media y el SpeedUp respecto a la versión original.

Versión	T. total (ms)	T. medio (ms)	SpeedUp
Original	253,49	1,53	0,00
Vectorizada	152,95	0,90	1,66

Como se puede apreciar en la Tabla I, la versión original del algoritmo tenía un coste de 253,49 ms, mientras que la versión vectorizada tiene un coste de 152,95 ms, lo que se traduce en una aceleración del 66 %.

Por otro lado, los datos representados en la Tabla II reflejan que originalmente se obtenía un coste de 252,54 y la vectorización un coste de 153,00, esto se traduce en un SpeedUp de 1,65 respecto a la versión original.

Tabla II: Tiempos obtenidos con la implementación de la vectorización con las imágenes de la Cámara 2. Se muestra el tiempo total, la media y el SpeedUp respecto a la versión original.

Versión	T. total (ms)	T. medio (ms)	SpeedUp
Original	252,54	1,50	0,00
Vectorizada	153,00	0,89	1,65

En resumen, la implementación del algoritmo de conversión con instrucciones de vectorización AVX2 ha permitido obtener una aceleración del 66 % en el algoritmo que ejecuta la Cámara 1 y del 65 % en el de la Cámara 2.

V. Conclusiones

El objetivo principal del trabajo era identificar y mejorar los costes temporales de la etapa de preproceso de la aplicación Multiscan Visual System. Para ello, se ha estudiado su funcionamiento y aplicado la tecnología que mejor se adaptaba al problema, en este caso las instrucciones de vectorización.

El preproceso es de las primeras etapas de la aplicación para tratar las imágenes tomadas por las cámaras y de las más críticas, ya que su funcionamiento impacta directamente en la calidad del análisis. Este algoritmo formaba parte de una de las etapas con más coste de todo el flujo de procesos. Para reducir su coste temporal se ha añadido la vectorización a sus operaciones para obtener una aceleración del 65 %. Además, se ha obtenido la misma salida que el algoritmo inicial sin perder precisión en su calidad.

Agradecimientos

El presente trabajo ha sido financiado en su totalidad por la empresa Multiscan Technologies.

Referencias

[1] Diego Inácio Patrício and Rafael Rieder, "Computer vision and artificial intelligence in precision agriculture for grain crops: A systematic review," *Computers and Electronics in Agriculture*, vol. 153, pp. 69–81, 2018.

[2] Matthias Breier, Constantin Haas, Wei Li, and Dorit Merhof, "Color filter arrays revisited — evaluation of bayer pattern interpolation for industrial applications," in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, 2016, pp. 52–57.

[3] Chris Lomont, "Introduction to Intel advanced vector extensions," *Intel white paper*, vol. 23, 2011.

[4] Intel, "Guía de instrucciones AVX," `https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html#`, [Consultado el 6 de junio del 2023].

[5] Thomas Jakobs and Gudula Rünger, "On the energy consumption of load/store AVX instructions," in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2018, pp. 319–327.

[6] Alex Peleg, Sam Wilkie, and Uri Weiser, "Intel MMX for Multimedia PCs," *Commun. ACM*, vol. 40, no. 1, pp. 24–38, jan 1997.

PANES(PArejas afiNES): Estrategia de mejora de prestaciones en procesadores SMT de ARM

Marta Navarro [1], Josué Feliu[1], Salvador Petit[1], María E. Gómez[1] y Julio Sahuquillo[1]

Resumen—**Los procesadores *simultaneous multithreading* mejoran las prestaciones respecto a los procesadores monohilo gracias a que comparten los recursos internos del núcleo entre las instrucciones de distintos hilos. Sin embargo, este hecho introduce interferencias entre los hilos que se ejecutan en el mismo núcleo, lo que repercute negativamente en las prestaciones de las aplicaciones individuales y puede aumentar significativamente el tiempo de ejecución de las cargas de trabajo compuestas por múltiples aplicaciones. El impacto de la interferencia en las prestaciones de una aplicación depende de la demanda de recursos de la aplicación con la que comparte el núcleo. Por tanto, puede mitigarse aplicando una estrategia de asignación de hilos a núcleos que mueva los hilos seleccionando inteligentemente las aplicaciones que se ejecutan en el mismo núcleo para minimizar sus interferencias.**

Este artículo presenta PANES, una estrategia sencilla que asigna dinámicamente hilos a núcleos de un procesador SMT en función de su comportamiento dinámico en tiempo de ejecución. El enfoque se basa en un modelo de regresión para seleccionar parejas afines de aplicaciones de forma que se minimice la interferencia intra-núcleo. La principal novedad de esta estrategia es que sólo utiliza tres variables calculadas con los contadores de prestaciones de los procesadores ARM actuales. Los resultados experimentales muestran que PANES puede superar las prestaciones del planificador de Linux en un 37 % de media en cargas de trabajo de 8 aplicaciones que combinan aplicaciones principalmente limitadas por el *backend* del procesador con otras limitadas por el *frontend*.

Palabras clave—**SMT, aplicaciones afines, interferencia intra-núcleo, procesadores multinúcleo, ARM**

I. Introducción

LOS procesadores *simultaneous multithreading* (SMT) [1] es el paradigma multihilo comúnmente implementado recientemente en procesadores destinados a servidores de altas prestaciones. Este paradigma permite que las instrucciones de varios subprocesos/hilos se ejecuten en el mismo ciclo, lo que ayuda a aumentar las prestaciones del procesador. Para que el diseño sea eficiente, los principales recursos del núcleo, como las cachés de primer nivel y los operadores aritméticos, se comparten entre los hilos que se ejecutan juntos, lo que aumenta su utilización. De este modo, se mejoran las prestaciones con una sobrecarga mínima de área. Por ejemplo, según Marvell [2], el área apenas incrementa en un 5 % al añadir SMT en el procesador ThunderX3, mientras que la mejora en las prestaciones, a pesar de depender de las aplicaciones, es muy superior a ese valor.

El uso de recursos compartidos entre aplicaciones que se ejecutan en un núcleo SMT provoca interferencias entre las aplicaciones que afectan a las prestaciones de cada una de ellas. Dado que los recursos compartidos del núcleo son clave para las prestaciones, compartir los recursos entre núcleos puede hacer que el tiempo de ejecución de las aplicaciones aumente significativamente debido a la interferencia entre aplicaciones. Sin embargo, la degradación de prestaciones depende del consumo de recursos de las aplicaciones que se ejecutan juntas en el mismo núcleo SMT. El consumo de recursos por parte de las aplicaciones es diverso. Por ejemplo, las aplicaciones que necesitan más recursos de memoria presentan una alta tasa de acceso a la jerarquía de memoria, mientras que las aplicaciones de cálculo intensivo estresan las unidades aritméticas de enteros o de coma flotante. Esto significa que la interferencia entre aplicaciones en los recursos compartidos dependerá en gran medida de los recursos que necesite cada una de ellas en ejecución.

Como las prestaciones de una aplicación dependen de los *co-runners*, un enfoque viable para aumentar las prestaciones del procesador sería utilizar una política de asignación de hilos a núcleos (planificador) que tenga en cuenta la interferencia intra-núcleo. De manera más precisa, el objetivo del planificador, o más exactamente, la estrategia de asignación de hilos a núcleos, es seleccionar qué aplicaciones deben ejecutarse en cada núcleo para minimizar la interferencia intra-núcleo entre las aplicaciones. Esto puede lograrse, seleccionando aplicaciones con comportamientos complementarios; por ej., una aplicación que requiera mucha memoria con una aplicación de cálculo intensivo. Definimos el término *aplicaciones afines* para hacer referencia a aquellas aplicaciones que presentan comportamientos complementarios.

El diseño de este tipo de políticas enfocadas a procesadores reales debe afrontar principalmente dos retos. El primero consiste en estudiar si es posible caracterizar la *afinidad de las aplicaciones* desde el punto de vista de las prestaciones. Esto supone un reto porque, hoy en día, los procesadores implementan cientos de contadores de prestaciones que deben tenerse en cuenta para analizar si es factible estimar la afinidad de la aplicación en el núcleo que va a utilizar. El estudio abarca dos decisiones principales: en qué etapa del procesador debe analizarse las prestaciones y qué contadores de prestaciones de los disponibles en el procesador podrían ser útiles para el objetivo. Por ello, el segundo reto es la forma de estimar las parejas de aplicaciones más afines.

Se han propuesto algunas estrategias para algunos procesadores Intel [3] e IBM [4], [5], pero no se pueden aplicar a los procesadores ARM porque la arquitectura de contadores de prestaciones difiere amplia-

[1]Dpto. de Sistemas Informáticos y Computación, Universitat Politècnica de València, e-mail: marnaed@disca.upv.es

mente entre estos procesadores. A pesar de que los procesadores ARM siguen ganando popularidad en el segmento de los centros de datos debido a su alta eficiencia energética [6], según nuestros conocimientos, no existe ninguna estrategia que pueda aplicarse para aumentar sus prestaciones.

En este trabajo proponemos PANES, una política de asignación de hilos a núcleos que selecciona aplicaciones afines y las asigna a los núcleos SMT de un procesador ARM. PANES aborda el primer reto mencionado, cuantificando las prestaciones de una aplicación con un sencillo modelo de tres variables (tres categorías). El segundo reto, se aborda con un modelo de regresión lineal que estima las prestaciones de una aplicación cuando se ejecuta con otra en un núcleo SMT, basándose en el modelo de tres variables que caracteriza las prestaciones de las aplicaciones. Es decir, utilizamos una ecuación de regresión lineal por categoría para estimar la interferencia entre aplicaciones.

PANES es más sencillo que los enfoques anteriores propuestos para Intel e IBM. PANES sólo necesita tres ecuaciones y cuatro contadores de prestaciones para estimar cómo de afín es la ejecución SMT de una aplicación con un co-runner concreto. Por el contrario, [4], [5] proponen un modelo con cinco ecuaciones y seis contadores de prestaciones. Dado que el número de combinaciones posibles aumenta rápidamente con el número de núcleos y aplicaciones, la reducción en el número de ecuaciones se traduce en un 40 % menos de tiempo (*overhead*) necesario para estimar las prestaciones de todas las posibles parejas de aplicaciones. Del mismo modo, [3] requiere nueve métricas que implican 15 contadores de prestaciones para caracterizar las prestaciones de una aplicación.

Los resultados del estudio realizado muestran que PANES supera al planificador por defecto de Linux en términos de tiempo de ejecución, mejorando la utilización del procesador en torno a un 37 % de media en cargas de trabajo de 8 aplicaciones que combinan aplicaciones limitadas por el *frontend* con limitadas por *backend*, llegando a más del 60 % en algunas cargas de trabajo o mezclas. PANES también mejora el planificador de Linux en término de IPC en un 5,6 % en dichas mezclas.

Las principales aportaciones de este trabajo son las siguientes:

- Presentamos un modelo sencillo para estimar las prestaciones de aplicaciones en ejecución SMT basado en sólo tres variables.
- El modelo ideado presenta una gran simplicidad (sólo cuatro contadores), lejos de los trabajos existentes desarrollados para procesadores de otros vendedores [4]. Esto se traduce en una reducción del 40 % en la sobrecarga para estimar las prestaciones las posibles combinaciones.
- Hasta donde sabemos, esta es la primera estrategia *thread-to-core* para un procesador ARM con núcleos SMT.

II. Trabajos Relacionados

Uno de los puntos débiles conocidos de SMT es que las prestaciones de un hilo pueden verse muy afectadas en función de la afinidad de las aplicaciones ejecutadas simultáneamente.

Algunos trabajos de investigación han desarrollado heurísticas para abordar estas debilidades. Algunas estrategias [7], [8] se centran en procesadores genéricos evaluados utilizando simuladores. Snavely y Tullsen [7] intentan abordar este problema ejecutando periódicamente un subconjunto de las combinaciones posibles para muestrear sus prestaciones. Cazorla et al. [8] muestran que las prestaciones del sistema dependen del planificador, independientemente de la política de búsqueda de instrucciones aplicada a las instrucciones. Otros enfoques [9], [10], [11], [12], [13] se concentran en procesadores reales. [9] utiliza datos *off-line* (estáticos) para mejorar ligeramente las prestaciones del procesador en un Intel Pentium-4 Xeon. Feliu et al. [10] gestionan la ubicación de las aplicaciones con el objetivo de equilibrar la utilización del ancho de banda de L1 entre los núcleos en un Intel Xeon E5645. Asimismo, utilizando contadores de prestaciones en tiempo de ejecución pero centrándose en cargas de trabajo multihilo, Vega et al. [11] presentan una heurística para determinar cuándo deben consolidarse los hilos en los núcleos SMT del procesador IBM POWER7. Radojkovic et al. [12] proponen un método de inferencia estadística para mejorar la asignación de tareas en un procesador UltraSPARC T2. Navarro et al. [13] proponen una heurística, basada en el método [3], para identificar parejas afines.

Otros trabajos establecen modelos de regresión lineal que utilizan contadores de prestaciones para estimar las prestaciones de cada combinación de aplicaciones. Dos de los primeros intentos [14], [15] se centraron en un procesador genérico simulado y simplificado. Moseley et al. [14] emplean el modelado lineal y la partición recursiva para estimar la mejora al ejecutar dos aplicaciones simultáneamente. Sin embargo, no se menciona ningún contador de prestaciones en específico. Eyerman y Eeckhout [15] proponen un modelo analítico que predice la degradación que sufre cada aplicación cuando se ejecuta en el mismo núcleo físico con otras aplicaciones. El modelo propuesto se basa en pilas CPI personalizadas en la etapa de *dispatch* [16], [17]; lamentablemente, este enfoque no es práctico, ya que los contadores de prestaciones necesarios no están disponibles en los procesadores reales actuales.

Por último, otros trabajos más cercanos al nuestro también utilizan modelos de regresión lineal pero están limitados por el *hardware* disponible en procesadores reales. En SMiTE [18], los autores proponen un modelo de regresión que combina *sensibilidad* y *contención* para estimar la interferencia en las prestaciones en un procesador Intel Sandy Bridge. Este trabajo utiliza una única ecuación por núcleo y 24 contadores de prestaciones (12 por aplicación) relacionados con eventos de memoria como TLB y cachés. Lamentablemente, el modelo de predicción

Cavium ThunderX2 CN9975 processor	
# Núcleos	28
# Hilos	112
Microarquitectura del núcleo	
Ancho de *dispatch*	4
Tamaño del ROB	128 entradas
Tamaño de IQ	60 entradas
Load/Store buffer	64/36 entradas
# Puertos *Issue*	6
Subsistema de memoria	
L1I/L1D	32 KB / 32 KB
L2	256 KB
LLC compartida	28 MB
Memoria principal	64 GB

Tabla I: Principales características del procesador experimental y del subsistema de memoria.

no funciona correctamente con núcleos SMT, ya que la unidad de monitorización de prestaciones (Performance Monitoring Unit, PMU) está diseñada para contar los eventos por núcleo y, por tanto, no hay contadores disponibles para medir los eventos correspondientes a nivel de contexto SMT. Feliu et al. [4] aprovechan los mecanismos que contabilizan el CPI del IBM POWER8 y adaptan el modelo de interferencias para planificar la combinación óptima de aplicaciones en los núcleos SMT. Este trabajo utiliza cinco ecuaciones y seis contadores de prestaciones.

En resumen, pocos trabajos se han centrado en modelos de regresión que aborden las limitaciones impuestas por los procesadores reales, en los que identificar los contadores de prestaciones de entre los varios centenares disponibles en cada plataforma específica para afrontar el problema, si es posible, supone un verdadero reto. Además, a pesar de que los procesadores ARM han aparecido recientemente en el mercado de servidores, hasta donde sabemos, éste es el primer trabajo que aborda la ubicación de los hilos en núcleos SMT centrado en estos procesadores.

III. Entorno experimental y Metodología

Esta sección describe el entorno experimental, que consiste en el procesador utilizado y la infraestructura desarrollada para llevar a cabo los experimentos, el diseño de las cargas de trabajo (mezclas) y la metodología de evaluación.

A. Plataforma: Sistema e Infraestructura

El sistema utilizado para desarrollar esta investigación incluye un procesador Cavium ThunderX2 CN9975 [19] y una memoria principal DRAM de 64GB. Este procesador está basado en la microarquitectura Vulcan [20] e implementa el conjunto de instrucciones ARMv8.1A [21]. Está compuesto por 56 núcleos SMT2 y tiene una LLC compartida de 28 MB.

La Tabla I resume las principales características del sistema, incluidos los parámetros específicos de la microarquitectura del núcleo. El sistema ejecuta una distribución CentOS Linux 7 (AltArch) con kernel 4.18. Para llevar a cabo los experimentos, desarrollamos PANES como un gestor de hilos a nivel de usuario que recopila los datos de los contadores de prestaciones necesarios, los utiliza para estimar la afinidad de las posibles parejas de aplicaciones, selecciona la combinación óptima y realiza los cambios de ubicación de las aplicaciones en los núcleos asignados para cada una. La estrategia desarrollada utiliza la herramienta *perf* para configurar y leer los contadores de prestaciones y controla la ejecución de los hilos y su asignación a los distintos núcleos mediante la llamada al sistema *sched_setaffinity*. Para evaluar de forma justa las prestaciones de la política de programación de Linux utilizamos la misma plataforma software de ejecución pero permitimos que Linux seleccione qué hilo se ejecuta en cada núcleo (es decir, la asignación de hilo a núcleo).

B. Mezclas y Metodología

Para evaluar PANES, hemos diseñado un amplio conjunto de cargas de trabajo (mezclas) HPC de 8 aplicaciones de las suites SPEC CPU2006 [22], y CPU2017 [23]. Como ya hemos comentado, la interferencia en los recursos compartidos y, por tanto, las prestaciones de un procesador SMT varían fuertemente en función de las aplicaciones ejecutadas concurrentemente en cada núcleo. Con el fin de diseñar cargas de trabajo interesantes para evaluar las ganancias de prestaciones que *PANES* es capaz de lograr en diferentes escenarios, primero caracterizamos las 28 aplicaciones estudiadas en este trabajo. En cuanto a la longitud del intervalo (quantum), algunas aplicaciones pueden presentar un comportamiento inestable con un intervalo demasiado corto, que en ocasiones puede derivar a fuertes variaciones (subidas o bajadas) en el IPC. Por el contrario, un intervalo demasiado largo no puede detectar un cambio de fase al principio del intervalo hasta que éste expira. En nuestros experimentos, observamos que un intervalo de 100 ms presenta una longitud adecuada que tiene en cuenta las anteriores consideraciones.

La Figura 1 muestra la caracterización estática de cada aplicación teniendo en cuenta las tres categorías principales definidas: Full_Dispatch, Stall_Frontend y Stall_Backend. Esta caracterización se obtiene ejecutando cada aplicación en solitario.

Teniendo en cuenta esta caracterización, clasificamos las aplicaciones estudiadas en tres grupos principales en función de la fracción de backend y frontend que tienen. El primer grupo, denominado *Backend bound*, incluye las aplicaciones que se ven limitadas en más del 65 % del tiempo de su ejecución debido al *backend*, entorpeciendo el flujo de ejecución. El segundo grupo, *Frontend bound*, incluye las aplicaciones cuyas limitaciones debidas al frontend representan más del 35 %. Por último, el tercer grupo, (*Otros*), está formado por el resto de aplicaciones e incluye aplicaciones con diferentes comportamientos cuyos Backend_Stalls oscilan entre el 20 % (*hmmer*) y el 61,4 % (*nab_r*). La Tabla II muestra la clasificación de cada benchmark en estos tres grupos.

Como nuestro objetivo es evaluar un abanico representativo de escenarios, hemos creado cargas de

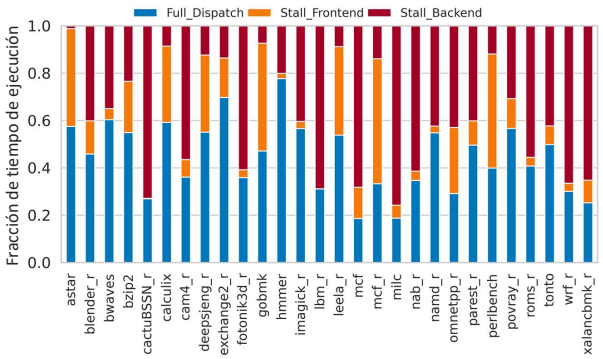


Fig. 1: Caracterización de las aplicaciones en ejecución en solitario.

Grupo	Categoría dominante	Aplicaciones
Backend bound	Backend stalls > 65 %	cactuBSSN_r, lbm_r, mcf, milc, xalancbmk_r,wrf_r
Frontend bound	Frontend stalls > 35 %	astar, gobmk, leela_r, mcf_r, perlbench
Otros	resto de aplicaciones	blender_r, bwaves, bzip2, calculix, cam4_r, deepsjeng_r, exchange2_r, fotonik3d_r, hmmer, imagick_r, nab_r, namd_r, omnetpp_r, parest_r, povray_r, roms_r, tonto

Tabla II: Aplicaciones agrupadas como backend bound y frontend bound teniendo en cuenta su fracción de backend y frontend stalls, respectivamente.

trabajo con uso intensivo de backend, cargas de trabajo con uso intensivo de frontend y cargas de trabajo mixtas para evaluar los respectivos escenarios. Cada carga de trabajo consta de 8 aplicaciones. Las cargas de trabajo con uso intensivo de backend incluyen 5 o 6 aplicaciones seleccionadas aleatoriamente del grupo *Backend Bound* y el resto de aplicaciones seleccionadas aleatoriamente del grupo *Otros*. Las cargas de trabajo con uso intensivo del frontend se construyen de forma análoga, pero toman la mayoría de las aplicaciones del grupo *Frontend bound*. Por último, las cargas de trabajo mixtas se crean seleccionando aleatoriamente la mitad de las aplicaciones del grupo *Backend bound* y la otra mitad del grupo *Frontend bound*. Evaluamos un total de 20 cargas de trabajo: 5 cargas de trabajo backend intensivas (be0-be4), 5 cargas de trabajo frontend intensivas (fe0-fe4) y 10 cargas de trabajo mixtas (fb0-fb9).

Las aplicaciones de las cargas de trabajo tardan distintos tiempos en ejecutarse. Por ello se utilizó la siguiente metodología de medición para ejecutar las aplicaciones. En primer lugar, ejecutamos cada aplicación en solitario durante 60 segundos y registramos su número de instrucciones retiradas, es decir, ejecutadas que han acabado con éxito. Este número será el número objetivo de instrucciones que se ejecutan cuando la aplicación se ejecuta dentro de una mezcla. Cuando se ejecutan mezclas con más de una aplicación, registramos las prestaciones de cada aplicación individualmente en el momento en que alcanza el número objetivo de instrucciones. A continuación, se relanza la aplicación para mantener la mezcla con el número de aplicaciones constante durante todo el experimento. La ejecución de la mezcla finaliza cuando la aplicación más lenta alcanza su número objetivo de instrucciones. Por último, los resultados experimentales presentados para cada carga de trabajo en la siguiente sección se obtuvieron como el valor medio de tres ejecuciones de la carga de trabajo para reducir posibles desviaciones.

C. Caracterización de las prestaciones

Investigamos los contadores de prestaciones disponibles de los procesadores ARM [21] para idear un nuevo enfoque, ya que ninguno de los trabajos relacionados se puede aplicar en este tipo de procesadores y arquitectura. Descubrimos que, a diferencia de otras arquitecturas existentes, los contadores hardware de ARM pueden utilizarse para evaluar las prestaciones en la etapa de *dispatch*. Más concretamente, descubrimos que en este punto podemos discernir entre los ciclos de parada del frontend (*frontend stalls*) y del backend (*backend stalls*) con los contadores disponibles.

Para evaluar las prestaciones y el comportamientos de las aplicaciones necesitamos contadores de prestaciones que, combinándolos, den el recuento total de ciclos en un punto determinado del flujo de ejecución.

Los ciclos de parada o *stalls* se refieren a ciclos de ejecución en los que el flujo de la ejecución se detiene. En este caso, los *stalls* pueden atribuirse al frontend o al backend del procesador. Cuando la cola de *dispatch* está vacía, no se puede enviar ninguna instrucción, el bloqueo debe atribuirse al frontend. Nos referimos a estos ciclos de parada como *frontend stalls*. Por ejemplo, esto puede ocurrir debido a *stalls* causados por un fallo de la caché de instrucciones. Por el contrario, cuando la cola de envío no está vacía pero las instrucciones no se pueden enviar debido a la falta de un recurso necesario (por ejemplo, una entrada del ROB), el bloqueo se asigna al backend (denominado *backend stalls*). Estos bloqueos se deben principalmente a instrucciones de larga latencia, como una carga que accede a la memoria principal, lo que provoca el bloqueo del ROB.

Para obtener los valores de las tres categorías principales, monitorizamos los contadores de prestaciones *CPU_CYCLES*, *STALL_FRONTEND*, y *STALL_BACKEND*. En la Tabla III se describen estos eventos hardware junto con el evento *INST_SPEC*, cuyo uso se describirá a continuación.

El procesador puede enviar tantas instrucciones como el ancho de envío de instrucciones en un ciclo dado. Esto significa que si el ancho de envío de instrucciones es N y se envía una única instrucción en un ciclo, ese ciclo no se contará como un problema en el envío a pesar de que se desperdicien $N-1$ ranuras del ancho de banda en esa etapa. En tal caso, para un ciclo determinado, las paradas reales pueden estimarse con precisión como $\frac{N-1}{N}$ %. Para tener en cuenta estos casos en la ejecución, calculamos los ciclos necesarios para enviar las instrucciones suponiendo que se consume todo el ancho de banda de envío (4 en nuestro procesador). Nos referimos a estos ciclos como ciclos de envío de instrucciones completos equivalentes (*Full-dispatch cycles*) y los calculamos como el número de instrucciones ejecutadas especulativamente (*INST_SPEC*) dividido por el ancho de banda de envío de nuestro procesador.

Por último, a diferencia de otros enfoques [16], [17],

Nombre del contador	Explicación
CPU_CYCLES	Ciclos
INST_SPEC	Operaciones (especulativamente) ejecutadas
STALL_FRONTEND	Ciclos donde ninguna operación se ha podido enviar porque no hay operaciones en la cola
STALL_BACKEND	Ciclos donde ninguna operación se ha podido enviar porque los recursos del backend no están disponibles

Tabla III: Contadores de prestaciones del procesador ARM.

no hacemos ninguna distinción entre instrucciones que acaban con éxito (retiradas) y aquellas mal especuladas. Nuestra clasificación de prestaciones busca estimar la interferencia entre aplicaciones en los *slots* de *dispatch*. Por lo tanto, desde la perspectiva del consumo de recursos, no hay distinción si el *slot* está siendo consumido por una instrucción que acaba con éxito o por una instrucción que es cancelada. El modelo de regresión implementado utilizará posteriormente la clasificación de ciclos y paradas para estimar la afinidad entre parejas de aplicaciones.

IV. Búsqueda de parejas afines

PANES selecciona las parejas de aplicaciones que se ejecutarán en cada núcleo del procesador con el fin de minimizar las interferencias entre aplicaciones en todos los núcleos del procesador. Para ello, se utiliza un modelo de regresión lineal con sólo tres variables. En esta sección se analiza el modelo de regresión desarrollado.

A. Estimación de prestaciones en ejecución SMT

El modelo calcula la degradación de las prestaciones de un hilo cuando se ejecuta simultáneamente con otro hilo en el mismo núcleo SMT en comparación con la ejecución en solitario. Por ejemplo, se espera que las *stalls* de backend de una aplicación, causadas por la falta de recursos de backend (por ejemplo, entradas del ROB), aumenten al ejecutarse junto a otra aplicación. En consecuencia, la suma de las tres categorías recogidas en la ejecución SMT normalizada a la ejecución en solitario superará los 100 % ciclos, lo que representa la ralentización que sufre la aplicación debido a la ejecución SMT.

Estimar con precisión en qué medida varía cada categoría es una tarea difícil, ya que depende no sólo de las características de la propia aplicación, sino también de las de las aplicaciones que se ejecutan con ella, es decir, de su interferencia dentro del núcleo. Para predecir la variación de las categorías en la ejecución SMT, ideamos un modelo de regresión lineal. El modelo se basa en la Ecuación 1, donde $C_{i,j}^{smt}$ representa el valor de la categoría C para la aplicación i cuando se ejecuta junto con la aplicación j en el mismo núcleo, C_i^{st} y C_j^{st} representan el valor de la categoría C para la aplicación i y j respectivamente, cuando se ejecutan en solitario, es decir, en modo ST (*single-threaded*), y α_C, β_C, γC, y ρ_C son los coeficientes del modelo.

$$C_{i,j}^{smt} = \alpha_C + \beta_C \cdot C_i^{st} + \gamma_C \cdot C_j^{st} + \rho_C \cdot C_i^{st} \cdot C_j^{st} \quad (1)$$

En otras palabras, la Ecuación 1 estima el valor para la categoría C de la aplicación i cuando se ejecuta con la aplicación j en el mismo núcleo como una suma ponderada de los tres componentes siguientes:

- El valor de la categoría C en la aplicación i, cuando i se ejecuta en solitario (C_i^{st}).
- El valor de la categoría C en j, cuando j se ejecuta en solitario (C_j^{st}).
- El producto de los dos valores anteriores.

Cada término se multiplica por un coeficiente (β_C, γC, y ρ_C) que pondera la influencia de cada término para modelar cada categoría. El coeficiente α_C, conocido como término independiente, ayuda a reducir las posibles imprecisiones del modelo y mejora su precisión. α_C, β_C, γC, y ρ_C se obtienen, para cada categoría en su conjunto, mediante regresión lineal, por lo que no dependen de cada aplicación en particular. En general, el modelo estima el tiempo para una categoría en SMT ($C_{i,j}^{smt}$) como la suma de esa categoría para la aplicación objetivo en solitario (C_i^{st}) más algunas interferencias introducidas por los co-runners.

La Tabla IV presenta los valores de los coeficientes obtenidos con la regresión lineal para modelar el comportamiento de cada categoría. Como era de esperar, la categoría *backend stalls* es la más afectada, ya que los hilos comparten (y compiten por) recursos críticos del backend del procesador, como la caché de datos, lo que puede aumentar significativamente la ejecución de SMT. En cuanto a la categoría *frontend stalls*, crece en ejecución SMT proporcionalmente a la ejecución en solitario, pero se añade un coeficiente α_C relativamente alto. Los resultados mostrarán que estos *stalls* son más difíciles de estimar, especialmente cuando la mezcla consta de muchas aplicaciones limitadas por el frontend. Hasta cierto punto, este modelo refleja el hecho de que las etapas del frontend están limitadas debido a las políticas de búsqueda de instrucciones, que permiten que un único hilo acceda a la caché de instrucciones en un ciclo. Este hecho puede afectar significativamente al tiempo que corresponda al frontend en modo SMT. Hay que tener en cuenta que el crecimiento en esta categoría puede introducir cambios menores independientemente de la aplicación en ejecución.

Respecto al número de ciclos *full-dispatch* en SMT, se observa que el valor de β_C es menor que en solitario. La razón principal es que en la ejecución SMT, una aplicación avanza más lentamente que en el modo ST. El número de *stalls* crece y el porcentaje de ciclos *full-dispatch* se reduce. Para esta categoría, el coeficiente ρ_C tampoco es despreciable, lo que indica que la tasa de envío de instrucciones en modo SMT se ve afectada por la tasa de envío de ambos hilos.

Para evaluar la precisión del modelo, obtuvimos el error cuadrático medio (MSE) para las categorías de *backend stalls*, *frontend stalls* y ciclos *full-dispatch*, que son de $0,1583$, $0,0703$ y $0,0021$, respectivamente. El error en la categoría de *backend stalls* es el más alto porque esta categoría es la más sensible a las variaciones de las posibles interferencias, ya que comprende varios recursos críticos del backend cu-

Category	α	β	γ	ρ
Full-dispatch cycles	0.0072	0.9060	0.0044	0.0314
Frontend stalls	0.2376	1.4111	0	0
Backend stalls	0.2069	0.3431	1.4391	0

Tabla IV: Coeficientes del modelo para las tres categorías

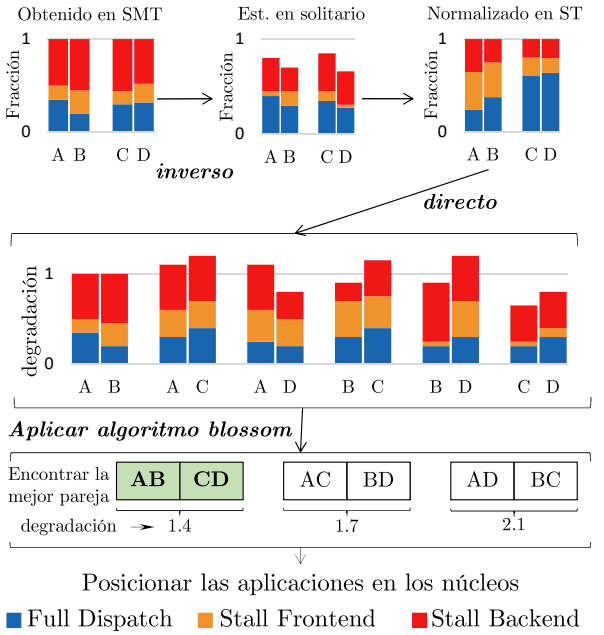


Fig. 2: Esquema general de los pasos seguidos por la estrategia PANES.

ya contención depende en gran medida del comportamiento del co-runner (como corrobora el valor de γ_C).

B. *Estrategia de asignación de hilos a núcleos parejas afines (PANES)*

En primer lugar, el procesador ejecuta una pareja diferente de aplicaciones en cada núcleo SMT. Los contadores de prestaciones están configurados para monitorizar los eventos mostrados en la Tabla III, obteniendo datos de cada aplicación. Los eventos se monitorizan en cada quantum de 100ms (intervalo) para obtener sus valores durante la ejecución. Una vez que el quantum expira, la estrategia actúa para determinar cuáles son las parejas de aplicaciones más afines y los asigna a núcleos para maximizar sus prestaciones. A continuación, describimos los diferentes pasos que realiza PANES, que se representan en la Figura 2. Obsérvese que estos pasos se repiten en cada quantum hasta completar la ejecución de la mezcla (carga de trabajo).

Estimar el valor de las categorías en ejecución en solitario (ST)

Como ya se ha explicado anteriormente, el modelo de regresión predice los valores de la categoría en SMT a partir de los valores de la ejecución en solitario (modo ST, single-thread). Sin embargo, durante la ejecución SMT, estos valores ST no están disponibles. Afortunadamente, Feliu et al. [4] ya abordaron este problema y demostraron que el modelo de interferencia propuesto podía invertirse para estimar, con una precisión relativamente buena, los valores C_i^{st} y C_j^{st} de las categorías de dos aplicaciones (i y j) que se ejecutan en el mismo núcleo SMT. Estos valores se calculan utilizando los valores de $C_{i,j}^{smt}$ y $C_{j,i}^{smt}$, que están disponibles en el modo SMT. Por lo tanto, aprovechamos esta inversión del modelo para obtener C_i^{st} y C_j^{st} en tiempo de ejecución. Para ello, después de que termine cada quantum, PANES recopila los datos de eventos para cada aplicación y calcula las categorías para la ejecución SMT (es decir, $C_{i,j}^{smt}$ y $C_{j,i}^{smt}$) durante el último intervalo. Estas categorías se normalizan con respecto a los ciclos de ejecución para obtener la probabilidad de ocurrencia de cada categoría ($C'^{smt}_{i,j}$ y $C'^{smt}_{j,i}$). A continuación, PANES aplica el *modelo inverso* para estimar el valor ST de cada categoría (C_i^{st} y C_j^{st}).

Estimar las prestaciones en SMT de las parejas de aplicaciones. Con el valor de las categorías de cada aplicación cuando se ejecutan en solitario, podemos aplicar el modelo de regresión para predecir las prestaciones en modo SMT. El modelo estima la degradación de las prestaciones (es decir, la ralentización) de una aplicación determinada cuando se ejecuta junto con otra aplicación en el mismo núcleo SMT. Aplicar la ecuación del modelo dos veces (una para la aplicación A cuando se ejecuta conjuntamente con la aplicación B y otra para la aplicación B cuando se ejecuta conjuntamente con la aplicación A) permite estimar el impacto de A en las prestaciones de B y viceversa. En otras palabras, estimar el grado de afinidad de una pareja de aplicaciones.

Seleccionar las parejas afines. Con la afinidad de cada pareja de aplicaciones, podemos seleccionar las mejores combinaciones, es decir, la combinación con la menor degradación global. Sin embargo, el número de combinaciones posibles crece rápidamente con el número de núcleos y aplicaciones, lo que aumenta la posible sobrecarga de seleccionar la combinación óptima. Para encontrar la combinación óptima con la mínima sobrecarga modelamos el problema de selección como un problema de grafos y lo resolvemos con el algoritmo Blossom [24]. De este modo, se garantiza una asignación de hilos a núcleos casi óptima con una sobrecarga reducida. Por último, las aplicaciones se asignan a su núcleo correspondiente en función de la combinación más afín.

C. *Modelo de entrenamiento*

Antes de poder utilizar el modelo, debemos obtener el conjunto de coeficientes α_C, β_C, γC, y ρ_C para el modelo de regresión. Como ya se ha comentado, diseñamos un modelo por categoría, no por aplicación. Esto hace que el modelo propuesto sea flexible y pueda funcionar con cualquier aplicación siempre que el conjunto de entrenamiento sea lo suficientemente diverso.

Para entrenar el modelo, ejecutamos el 80 % de las aplicaciones (22 de 28) de manera individual y creamos un perfil con el valor de las distintas categorías y el número de instrucciones ejecutadas con éxito para cada quantum (intervalo). A continuación, ejecutamos en modo SMT todos las parejas posibles con las aplicaciones de entrenamiento y recogemos los mismos datos. A medida que la ejecución avanza

en modo SMT, el número de instrucciones completadas con éxito nos permite estimar los valores de cada categoría en cada aplicación si se estuviera ejecutando en solitario. Con estos valores y los valores correspondientes cuando la aplicación se ejecuta en modo SMT con otra aplicación (*co-runner*), se pueden obtener los coeficientes para cada categoría.

Por tanto, se seleccionó un subconjunto aleatorio de los intervalos de ejecución de las anteriores ejecuciones para construir el modelo. El modelo de cada categoría se entrenó utilizando los datos de estos intervalos, y los coeficientes se seleccionaron minimizando al máximo el error del modelo. Cabe destacar que el modelo se construyó para cargas de trabajo científicas de cálculo intensivo, como las del conjunto de CPU SPEC, por lo que es válido para cargas de trabajo que muestren un comportamiento similar. Para cubrir aplicaciones que muestran un comportamiento distinto (por ejemplo, cargas de grafos), es necesario ampliar el entrenamiento del modelo considerando estas aplicaciones.

Siempre que el conjunto de aplicaciones utilizadas para el entrenamiento sea lo suficientemente diverso, el modelo sólo tendrá que entrenarse una vez. Después, los coeficientes del modelo pueden utilizarse para seleccionar las parejas de aplicaciones más afines del sistema. Por tanto, el coste del entrenamiento del modelo puede compensarse rápidamente gracias a las ventajas de prestaciones que proporciona.

V. Evaluación

Esta sección evalúa las prestaciones del planificador SYNPA en términos de tiempo de ejecución (Turnaround Time, TT) e instrucciones por ciclo (IPC). Los resultados se comparan con el algoritmo *Completely Fair Scheduler* de CentOS Linux 7 (AltArch). Cabe recordar que los resultados no pueden compararse con los enfoques existentes para procesadores Intel e IBM, ya que los eventos hardware (contadores de prestaciones) que utilizan no están disponibles en el procesador ARM.

A. Análisis de tiempo de ejecución (TT)

El tiempo de ejecución (TT) de una carga de trabajo se define por el tiempo que la mezcla de aplicaciones tarda en ejecutarse. Más concretamente, por el tiempo que tarda la aplicación (*benchmark*) más lenta de la mezcla. La Figura 3 presenta la mejora del tiempo de ejecución de PANES sobre Linux para las cargas de trabajo estudiadas y la mejora media para cada tipo de carga de trabajo. Como se puede observar, PANES consigue mejoras significativas sobre Linux. Centrándonos primero en las cargas de trabajo que tiene más aplicaciones de backend (intensivas en backend) (`be0-be4`), se aprecia que PANES obtiene mejores resultados que Linux, alcanzando una mejora media del 18 %. La mejora obtenida disminuye significativamente en las cargas de trabajo principalmente de frontend (`fe0-fe4`), a pesar de que PANES sigue superando a Linux en un 6 %. Esto significa que hay poco margen para que la estrategia

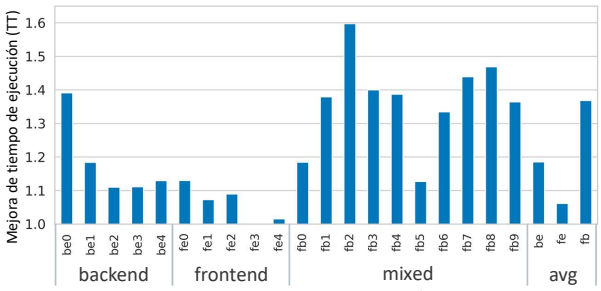


Fig. 3: Mejora en tiempo de ejecución con PANES con respecto a Linux.

mejore las prestaciones en las mezclas dominadas por las limitaciones provocadas por el frontend. Este razonamiento tiene sentido, ya que las prestaciones de estas aplicaciones se ven afectadas por los fallos de la caché de instrucciones cuando se ejecutan en solitario. Por lo tanto, es probable que este problema se agrave aún más en la ejecución SMT.

Una observación interesante es que las mejoras de prestaciones de PANES aumentan significativamente con cargas de trabajo mixtas (`fb0-fb9`), en las que se mezclan aplicaciones principalmente limitadas por el backend con aplicaciones que se ven limitadas por el frontend. En estas mezclas, PANES es capaz de mejorar el tiempo de ejecución (TT) hasta 1,60 en `fb2`, es decir, un 60 % en comparación con Linux, con una mejora media del 37 %. Esto significa que PANES es capaz de encontrar, en tiempo de ejecución, parejas afines de aplicaciones cuyo comportamiento se complementa y las asigna en los mismos núcleos SMT para maximizar las prestaciones globales.

B. Análisis de IPC

La ejecución de la parejas de aplicaciones más afines en los núcleos SMT reduce las interferencias, lo que no sólo acorta los tiempos de ejecución, sino que también aumenta las prestaciones del sistema. La Figura 4 presenta los resultados de las mejoras IPC de PANES sobre Linux. Calculamos el IPC de las cargas de trabajo utilizando la media geométrica de los IPCs de la aplicaciones de las mezclas. Como se observa, los valores de mejora en el IPC son inferiores a los de TT (Figura 3). Sin embargo, nuevamente las cargas de trabajo mixtas (`fb0-fb9`) son las que logran las mejoras más altas, que son, en promedio, alrededor del 5,6 %, y hasta alrededor del 12 % en dos cargas de trabajo (`fb2` y `fb6`). Como era de esperar, las cargas de trabajo donde predominan las aplicaciones limitadas por frontend (*Frontend bound*) presentan resultados similares a Linux, con una mejora media de alrededor del 0,3 %.

VI. Conclusiones

En este trabajo se ha presentado una estrategia simple de asignación de hilos, que utiliza un modelo de regresión lineal basado en tres categorías principales calculadas: *stalls frontend, stalls backend y full dispatch*. Estas categorías se definieron tras una búsqueda aproximada a través de todo el conjunto de contadores de prestaciones disponibles en el procesador. El modelo de regresión estima el tiempo de

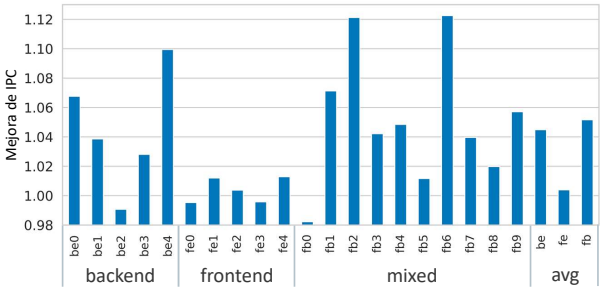


Fig. 4: Mejora del IPC (Media geométrica) con respecto a Linux.

ejecución de una aplicación en ejecución SMT con otros co-runners. De este modo, PANES puede seleccionar las mejores parejas de aplicaciones afines para su ejecución en cada núcleo del procesador.

Los resultados experimentales demuestran que PANES mejora el TT en torno a un 37 %, en promedio, para mezclas mixtas y por encima del 60 % en algunas de estas mezclas, gracias a las parejas afines correctamente elegidas. Además, el IPC mejora en este tipo de mezclas en torno a un 5,6 % de media.

Es importante destacar que PANES podría integrarse como parte del *scheduler* del sistema operativo de los procesadores ARM actuales, ya que todos los contadores de prestaciones utilizados en este trabajo forman parte de la PMU estándar de ARM v8.1. Por supuesto, el modelo de regresión debería entrenarse para las cargas de trabajo que se ejecutarán en el sistema.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación junto al FEDER europeo a través de los proyectos PID2021-123627OB-C51, TED2021-130233B-C3 y AICO/2021/266. Marta Navarro tiene la beca Subvenciones para la contratación de personal investigador predoctoral de la Generalitat Valenciana (ACIF2022). Josué Feliu cuenta con el apoyo del contrato RYC2021-030862-I financiado por MCIN/AEI /10.13039/501100011033 y por la Unión Europea NextGenerationEU/PRTR.

Referencias

[1] Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995, ISCA '95, p. 392–403.

[2] Rabin Sugumar, Mehul Shah, and Ricardo Ramirez, "Marvell thunderx3: Next-generation arm-based server processor," *IEEE Micro*, vol. 41, no. 2, pp. 15–21, 2021.

[3] Ahmad Yasin, "A top-down method for performance analysis and counters architecture," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 35–44.

[4] Josue Feliu, Stijn Eyerman, Julio Sahuquillo, and Salvador Petit, "Symbiotic job scheduling on the ibm power8," in *2016 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2016, pp. 669–680.

[5] Josué Feliu, Stijn Eyerman, Julio Sahuquillo, Salvador Petit, and Lieven Eeckhout, "Improving ibm power8 performance through symbiotic job scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 2838–2851, 2017.

[6] Tadviser, *Processors (Global Market)*, 2023.

[7] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for simultaneous multithreading processor," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000, pp. 234–244.

[8] C. Acosta, F. J. Cazorla, A. Ramirez, and M. Valero, "Thread to core assignment in SMT on-chip multiprocessors," in *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2009, pp. 67–74.

[9] Alex Settle, Joshua Kihm, Andrew Janiszewski, and Dan Connors, "Architectural support for enhanced SMT job scheduling," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2004, pp. 63–73.

[10] Josué Feliu, Julio Sahuquillo, Salvador Petit, and José Duato, "L1-bandwidth aware thread allocation in multicore smt processors," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, 2013, pp. 123–132.

[11] Augusto Vega, Alper Buyuktosunoglu, and Pradip Bose, "Smt-centric power-aware thread placement in chip multiprocessors," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2013, pp. 167–176.

[12] Petar Radojković, Vladimir Čakarević, Miquel Moretó, Javier Verdú, Alex Pajuelo, Francisco J. Cazorla, Mario Nemirovsky, and Mateo Valero, "Optimal task assignment in multithreaded processors: A statistical approach," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012, pp. 235–248.

[13] Marta Navarro, Lucia Pons, and Julio Sahuquillo, "Hysched: A simple hyperthreading-aware thread to core allocation strategy," *IEEE Computer Architecture Letters*, vol. 20, no. 1, pp. 26–29, 2021.

[14] T. Moseley, J.L. Kihm, D.A. Connors, and D. Grunwald, "Methods for modeling resource contention on simultaneous multithreading processors," in *International Conference on Computer Design: VLSI in Computers and Processors*, 2005, pp. 373–380.

[15] Stijn Eyerman and Lieven Eeckhout, "Probabilistic Job Symbiosis Modeling for SMT Processor Scheduling," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010, pp. 91–102.

[16] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A performance counter architecture for computing accurate CPI components," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2006, pp. 175–184.

[17] S. Eyerman and L. Eeckhout, "Per-thread cycle accounting in SMT processors," in *The International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2009, pp. 133–144.

[18] Yunqi Zhang, Michael A. Laurenzano, Jason Mars, and Lingjia Tang, "SMiTe: Precise QoS prediction on real-system SMT processors to improve utilization in warehouse scale computers," in *International Symposium on Microarchitecture (MICRO)*, 2014, pp. 406–418.

[19] "ThunderX2 CN9975 - Cavium," 2019, Accessed: 2023-02-04.

[20] "Vulcan - Microarchitectures - Cavium," `https://en.wikichip.org/wiki/cavium/microarchitectures/vulcan?utm_content=cmp-true`, 2019, Accessed: 2023-03-28.

[21] ARM, "Armv8.1-m performance monitoring user guide," 2020, Version 1.186.

[22] John L Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[23] Ankur Limaye and Tosiron Adegbija, "A workload characterization of the spec cpu2017 benchmark suite," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018, pp. 149–158.

[24] Jack Edmonds, "Maximum matching and a polyhedron with 0, 1-vertices," *Journal of research of the National Bureau of Standards B*, vol. 69, no. 125-130, pp. 55–56, 1965.

Paralelismo anidado utilizando BLAS maleable en procesadores multinúcleo

Sandra Catalán[1], Rafael Rodríguez-Sánchez[2], Adrián Castelló, Enrique S. Quintana-Ortí[3] y Francisco D. Igual[4]

Resumen— **La maleabilidad mejora el uso de los núcleos en procesadores multinúcleo al permitir variar el paralelismo en tiempo de ejecución. Es útil para aplicaciones con cargas de trabajo irregulares y trayectorias de ejecución divergentes. Sin embargo, la integración de la maleabilidad en las rutinas BLAS (Basic Linear Algebra Subprograms) de alto rendimiento es actualmente inexistente. En este trabajo, mostramos cómo se pueden obtener beneficios de rendimiento al explotar la maleabilidad en un marco diseñado para implementar operaciones BLAS portables y de alto rendimiento. Integramos la maleabilidad en la biblioteca BLIS y evaluamos experimentalmente los resultados en dos casos de uso distintos.**

Palabras clave— **Maleabilidad, BLAS, altas prestaciones, procesadores multinúcleo.**

I. Introducción

La *maleabilidad* a nivel de tarea es una propiedad muy apreciada en la computación en la nube, donde se considera como la capacidad de ampliar/reducir el número de recursos asignados a un determinado trabajo durante su ejecución. La maleabilidad a nivel de proceso también puede ofrecer importantes ventajas de rendimiento para las aplicaciones científicas distribuidas, aunque este contexto es considerablemente más complejo debido a la necesaria redistribución de la carga de trabajo y los datos; véase [1], [2] y las referencias que aparecen en ellas.

En cambio, el concepto de explotar *la maleabilidad a nivel de hilo* dentro de una aplicación puede parecer sencillo, tanto si se utilizan hilos POSIX (pthreads) como una interfaz de programación de aplicaciones paralelas (API) de alto nivel como OpenMP [3], pero sigue habiendo desafíos que deben abordarse para demostrar sus ventajas. En particular, muchas aplicaciones dependen de bibliotecas de hilos (TL) de álgebra lineal, como el nivel 3 de BLAS (Basic Linear Algebra Subprograms) o BLAS-3 [4], que están paralelizadas internamente para optimizar el rendimiento en procesadores multinúcleo [5], [6], [7].

Las TL pueden presentar tres grados diferentes de flexibilidad para determinar el número de hilos que ejecuta una rutina específica:

- *TLs estáticas*, en las que el número de hilos se decide a nivel de programa, por lo que no puede variar a lo largo de la ejecución del mismo. Por lo tanto, el número de hilos es constante en todos las rutinas dentro de la ejecución.
- *TLs Moldeables*, en las que el número de hilos que ejecuta una rutina puede variar dinámicamente a través de diferentes rutinas, pero en todos los casos *antes* de que comience la ejecución de cada rutina individual.
- *TLs Maleables*, en las que el número de hilos que ejecutan una rutina puede variar dinámicamente y bajo demanda *mientras* la rutina está en ejecución. Actualmente, el soporte para la maleabilidad en TLs es escaso o inexistente tanto en las TLs comerciales como en las de código abierto.

Además, las aplicaciones en las que cada tarea requiere la ejecución de una rutina de una biblioteca de álgebra lineal pueden explotar el paralelismo bien 1) sólo desde dentro de la aplicación 2) sólo desde dentro de las rutinas de la biblioteca, o 3) una combinación de ambas [8]. En este artículo, revisamos el tercer enfoque, explorando los beneficios de rendimiento al combinar una aplicación paralela basada en tareas (TA) con un BLAS *maleable* a nivel de hilo (MLB). Concretamente, nuestro enfoque TA+MLB divide los hilos existentes en equipos t_{App}, manejados por la aplicación, donde cada equipo (compuesto por uno o más hilos) se encarga de ejecutar una única tarea/rutina de álgebra lineal. Curiosamente, *en nuestro enfoque TA+MLB, el número de subprocesos de cada equipo no sólo puede variar dinámicamente durante la duración de la aplicación, sino también durante la ejecución de cada rutina individual.*

Este enfoque dinámico a nivel de rutina se construye sobre una instancia MLB desarrollada utilizando el *Basic Linear Algebra Instantiation Software* (BLIS) [7]. En [9], [10] esta solución ofrecía considerables ventajas para la ejecución de la factorización LU (densa) en procesadores multinúcleo actuales en comparación con las soluciones clásicas que explotan el paralelismo sólo dentro de BLAS, así como implementaciones más modernas, que explotan el paralelismo sólo a nivel de tarea/aplicación [11], [12], [13].

En este trabajo avanzamos significativamente hacia la demostración de las ventajas del enfoque TA+MLB mediante la elaboración y análisis de dos casos más allá de la simple descomposición matricial LU. En concreto, desarrollamos implementaciones TA+MLB de los siguientes casos de estudio, investigando diferentes estrategias de paralelización,

[1]Dpto. de Ingeniería y Ciencia de los Computadores, Universitat Jaume I, e-mail: catalans@uji.es.

[2]Dpto. de Sistemas Informáticos, Universidad de Castilla-La Mancha, e-mail: rafael.rodriguez@uclm.es.

[3]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: adcastel@disca.upv.es,quintana@disca.upv.es.

[4]Dpto. de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, e-mail: figual@ucm.es.

obteniendo información sobre la aplicabilidad/ventajas, y reportando el rendimiento de este enfoque:

- Factorización QR para la solución de problemas de mínimos cuadrados lineales (de rango completo) [14].
- Inferencia con redes neuronales profundas [15].

El resto del documento se estructura como sigue. Tras una visión general de los mecanismos para explotar el paralelismo en aplicaciones basadas en tareas, evaluamos los beneficios obtenidos de la introducción del mecanismo de maleabilidad en aplicaciones de álgebra densa y aprendizaje profundo. Cerramos el artículo con algunos comentarios finales y la propuesta de otras aplicaciones de maleabilidad.

II. Aplicaciones Paralelas Basadas en Tareas

Consideremos una aplicación compuesta por una colección de tareas, entrelazadas mediante dependencias, y donde cada tarea requiere la ejecución de una rutina de una biblioteca de álgebra lineal. El panel de la izquierda de la Figura 1 muestra un ejemplo sencillo para la siguiente discusión, que consta de cuatro tareas, numeradas como T0–T3, con dependencias de datos T0→T1→T3 y T0→T2→T3. La altura de cada casilla representada es proporcional al coste de la tarea (indicado por el valor dentro del paréntesis). En este caso sencillo, cada tarea implica la ejecución de una única rutina de álgebra lineal.

Este tipo de aplicación se puede paralelizar mediante uno de los tres esquemas siguientes [8]:

- **Aplicación secuencial invocando a una biblioteca de hilos (SA+TL)**. Esta opción extrae el paralelismo del interior de las rutinas de álgebra lineal ejecutando las tareas en orden secuencial. Es decir, en cualquier momento, sólo hay una tarea en ejecución, pero la rutina correspondiente puede ser procesada utilizando múltiples hilos a través de la llamada a una TL de álgebra lineal. Esto se ilustra en el panel etiquetado como SA+TL en la Figura 1. Suponemos un escenario ideal en el que las rutinas son altamente paralelas. Así, la ejecución en paralelo de una tarea con un coste de c unidades de tiempo, utilizando t hilos, reduce su tiempo de ejecución a c/t unidades de tiempo. Sin embargo, supone un problema para aquellas aplicaciones que requieren un número no despreciable de rutinas secuenciales, que no pueden beneficiarse de una ejecución multihilo. Por ejemplo, cuando las rutinas secuenciales presentan una alta complejidad computacional, su coste no puede ocultarse solapando su ejecución con la de otras rutinas (independientes). En el ejemplo de la Figura 1 esto ocurriría, por ejemplo, si la tarea T2 fuera mayoritariamente secuencial, de modo que su ejecución utilizando cuatro hilos no produjera ninguna reducción en el tiempo de ejecución respecto a la ejecución secuencial.
- **TA invocando una biblioteca secuencial (TA+SL)**. Ésta suele ser la solución preferida para las TA, en las que el paralelismo se extrae únicamente a nivel de aplicación, normalmente mediante un *runtime* que orquesta una ejecución del grafo de tareas que tiene en cuenta las dependencias, y en la que cada tarea se ejecuta invocando a una rutina de una biblioteca secuencial. Este caso se representa en el panel izquierdo de la Figura 1 donde cada tarea es ejecutada por un único hilo, respetando las dependencias de las tareas. Las tareas T1 y T2 no presentan dependencias y, por lo tanto, pueden proceder simultáneamente. Algo a tener en cuenta de este esquema es que requiere descomponer la aplicación en un número "suficiente" de tareas de granularidad "apropiada", exponiendo un delicado equilibrio:
  - Por un lado, si las tareas son demasiado pesadas, puede haber muy pocas de ellas, lo que limita la escalabilidad paralela (a nivel de aplicación). Además, dividir las tareas en subtareas de grano más fino puede ser inviable para algunas aplicaciones.
  - Por otro lado, crear demasiadas tareas aumenta la sobrecarga introducida por el *runtime* encargado de controlar las dependencias de las tareas. Además, puede dar lugar a una ejecución subóptima de las rutinas secuenciales de álgebra lineal, que pueden ser demasiado pequeñas para explotar adecuadamente la jerarquía de memoria del sistema [16].
  - Por último, para las TA que comprenden rutinas de álgebra lineal de cálculo intensivo con pocas dependencias (como las del BLAS-3), el esquema TA+SL ofrece en general un rendimiento inferior al de una alternativa que simplemente combina un SA con una versión multihilo de BLAS [9], [10].
- **TA invocando una biblioteca multihilo (TA+TL)**. Este esquema se ilustra en el panel TA+TL (1) en la Figura 1 donde, como en el caso anterior, el paralelismo se extrae a nivel de aplicación mediante un *runtime*. Sin embargo, cada tarea se ejecuta ahora utilizando (un número fijo de) dos hilos. Este caso es un ejemplo de la utilización de una TL *estática*, donde el número de hilos por tarea se define por aplicación, y no varía ni a través de la invocación de las rutinas ni dentro de las mismas. Esta opción debe ser adoptada con cuidado, ya que es propensa a incurrir en *oversubscription*, (la creación de más hilos que núcleos físicos), lo cual, para aplicaciones de tareas paralelas que implican rutinas de álgebra lineal, a menudo resulta en un bajo rendimiento en procesadores multinúcleo modernos.

En este trabajo revisamos el esquema TA+TL, mejorándolo con maleabilidad para producir una solución más eficiente. En concreto, consideremos la ejecución representada gráficamente en el panel TA+TL (2) en la Figura 1. Esta aproximación produce un

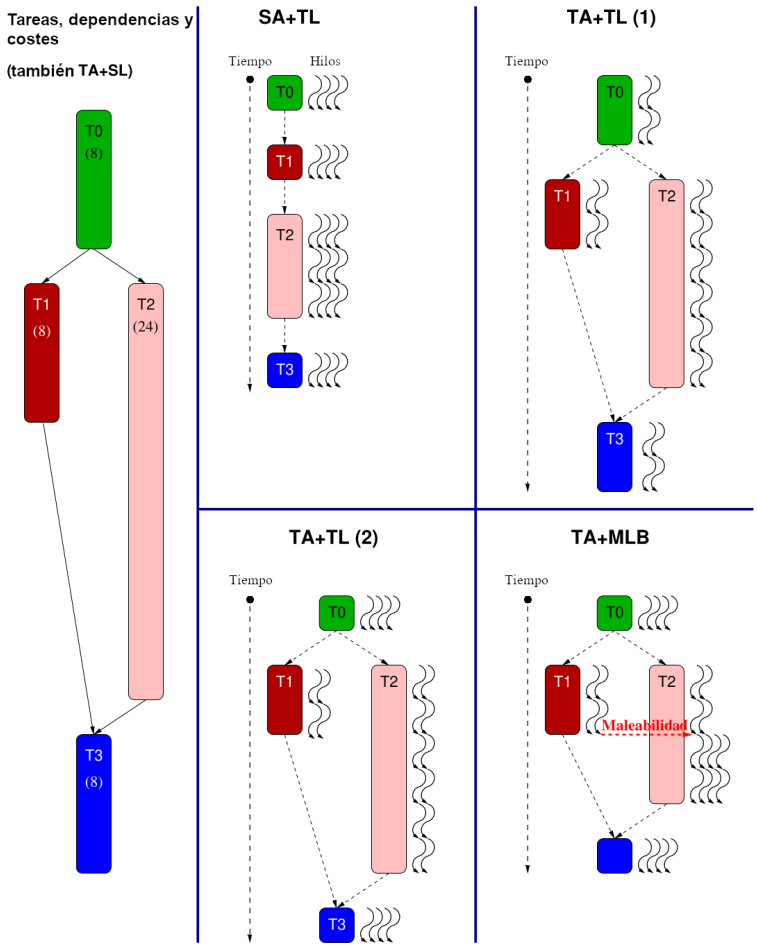


Fig. 1: Aplicación paralela basada en tareas y esquemas de paralelización. SA+TL y TA+TL(1) usan TLs *estáticas*; TA+TL(2) es un ejemplo de aplicación de TLs *moldeables*; TA+MLB requiere una TL *maleable*.

esquema más eficiente que su homólogo TA+TL (1), ya que el número de hilos asignados a la ejecución de una tarea no es fijo, sino que puede aprovechar todos los recursos disponibles al comienzo de la ejecución de una tarea. Este es un ejemplo de una TL *moldeable*, en la que el número de hilos se determina en función de cada rutina. Lamentablemente, esta opción sigue presentando una penalización de rendimiento ya que, con las instancias actuales de BLAS, al finalizar los dos subprocesos que participaron en la ejecución de la tarea T1 deben permanecer inactivos hasta que finalice la ejecución de la tarea T2.

El enfoque propuesto en este trabajo avanza un paso más en la dirección de la maleabilidad *para permitir que el número de hilos que participan en la ejecución de una tarea varíe durante la ejecución de la correspondiente rutina de álgebra lineal*. Esto se ilustra en el panel TA+MLB en la Figura 1, donde los dos hilos encargados de ejecutar la tarea T1 comienzan a colaborar en la ejecución de la tarea T2 en cuanto terminan con la primera. Este es un ejemplo de una TL *maleable*, en la que el número de hilos asignados a la ejecución de una determinada rutina puede ser modificado durante su ejecución.

Nuestro esquema dinámico y maleable a nivel de rutina aborda los dos problemas siguientes relacionados con la programación. Consideremos una aplicación paralela basada en tareas que debe ser ejecutada por un *runtime* utilizando un número de "equipos de hilos" $T\_A, T\_B, T\_C \ldots$, siendo cada equipo el encargado de ejecutar una única tarea en un momento dado. (Por ejemplo, en el esquema TA+TL (1) de la Figura 1 hay dos equipos, cada uno formado por dos hilos). Supongamos que el equipo T_A está a punto de comenzar la ejecución de la tarea/rutina T:

- El equipo T_A podría beneficiarse de un mayor número de hilos de los que tiene asignados actualmente. Sin embargo, los equipos $T_B, T_C \ldots$ están actualmente ejecutando otras rutinas de álgebra lineal y, con las bibliotecas no maleables existentes, no liberarán sus recursos hasta que completen su trabajo. ¿Debería el equipo T_A esperar hasta que esto ocurra o comenzar la ejecución de T de inmediato? En el segundo caso, la llamada a una biblioteca no maleable implica que T_A no puede beneficiarse de los recursos adicionales ociosos (es decir, hilos), liberados por cualquier equipo una vez iniciada su ejecución de la rutina T.
- Supongamos que, para esta rutina T, el equipo T_A realmente no ganará mucho en términos de rendimiento utilizando todos los hilos que tiene asignados. Sin embargo, si este equipo T_A libera los recursos innecesarios, estos hilos permanecerán inactivos hasta que cualquier otro equipo complete la ejecución de su rutina correspon-

diente y pueda hacer uso de ellos en la ejecución de una tarea posterior. Idealmente, preferiríamos que T_A libere los hilos innecesarios y que luego se muevan para ayudar a otros equipos en la ejecución de sus tareas.

Para evitar estos problemas, adoptamos una solución dinámica, en la que los hilos migran entre equipos durante la ejecución de las rutinas, utilizando una biblioteca MLB construida sobre BLIS [7]. En las siguientes secciones, ilustramos los beneficios de este enfoque utilizando dos casos relevantes de álgebra lineal densa y aprendizaje profundo, ejecutados en procesadores multinúcleo.

A. Integración de la maleabilidad en BLIS

El mecanismo de maleabilidad que hemos integrado en BLIS v0.5.1 se describe en detalle en [17] junto con un análisis de la sobrecarga introducida por este mecanismo. Por brevedad, a continuación sólo resumimos los aspectos más relevantes de esta integración.

Para introducir la maleabilidad la clave está en aprovechar (y modificar) la API de BLIS para distinguir en la biblioteca entre 1) el número máximo de hilos y 2) el número activo de hilos. El primer parámetro especifica la cantidad de hilos que están inicialmente activos al invocar una rutina, y suele coincidir con el número de núcleos de la máquina. El segundo parámetro especifica el número de subprocesos que participan en un cálculo determinado en un momento concreto, y puede ser modificado de forma asíncrona en cualquier momento por cualquier subproceso de la aplicación.

Utilizando estos dos parámetros, la carga de trabajo para cada uno de los bucles anidados de la rutina BLIS se distribuye entre los hilos. Como resultado, los hilos que no tienen asignada ninguna carga de trabajo en un bucle determinado pasan inmediatamente al final del mismo, donde permanecen bloqueados hasta que todos los hilos activos que participan en la ejecución del bucle completen su parte del trabajo.

III. Factorización QR con look-ahead

Consideremos la factorización QR [14] de una matriz A no singular $m \times n$, dada por $A = QR$, donde la matriz Q es ortogonal y el factor R es triangular superior. Para simplificar, en adelante supondremos que n es un múltiplo entero del tamaño de bloque algorítmico b. Además, consideramos una partición de las columnas de A en $s = n/b$ bloques, de b columnas cada uno. En nuestra notación $A(:, c_1 : c_2)$ se refiere a la submatriz de A que abarca los bloques $c_1, c_1 + 1, \ldots, c_2$-ésimos paneles (o bloques de columnas) de A, que comprende las columnas $c_1 \cdot b, c_1 \cdot b + 1, \ldots, c_2 \cdot b - 1$. (Obsérvese que, en nuestra notación, los índices de bloques y elementos empiezan en 0).

El Código 1 y la figura que lo acompaña (Figura 2) muestran una (versión simplificada) de un algoritmo por bloques que calcula la factorización QR de una matriz cuadrada A $n \times n$ expresada con un alto nivel de abstracción. Esta formulación corresponde a un algoritmo por bloques que ofrece un alto rendimiento siempre que b sea moderadamente grande, reduciendo la relación entre flops y accesos a memoria. El algoritmo realiza s iteraciones, calculando primero la factorización del panel "actual" (es decir, el bloque de la columna k-ésima); y luego actualizando los paneles a su derecha con las correspondientes transformaciones ortogonales (actualización de la submatriz). Para simplificar, estas dos operaciones se encapsulan dentro de las rutinas `PF` y `TU`, respectivamente. Entre ellas son bastante diferentes: `PF` es principalmente una operación secuencial, mientras que `TU` puede realizarse mediante BLAS-3 altamente paralelo.

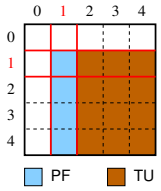


Fig. 2: Partición de una matriz formada por $s \times s$ bloques, con $s = 5$, y las operaciones realizadas en la iteración $k = 1$ del algoritmo para la factorización QR en el Listado 1.

Listing 1: Rutina simplificada para la factorización QR.

```
1 void QR( matrix A, int s )
2 {
3   for ( k = 0; k < s; k++ ) {
4     // Factorizar panel k
5     PF( A(k:s-1,k) );
6     // Actualizar paneles k+1:s-1 resp. panel k
7     TU( A(k:s-1,k), A(k:s-1,k+1:s-1) );
8   }
9 }
```

El enfoque convencional para paralelizar esta factorización matricial consiste en explotar el paralelismo de bucles sólo desde BLAS (Nivel 3), lo que corresponde al esquema SA+TL. En la última década, ha habido un notable interés en exponer y aprovechar el paralelismo de tareas a un nivel superior, desde las propias rutinas LAPACK, apoyándose en una versión secuencial del BLAS, como corresponde al esquema TA+SL [11], [12], [13]. Para la factorización QR, esto se realiza a menudo 1) dividiendo tanto `PF` como `TU` en tareas de grano más fino; 2) aprovechando un *runtime* para organizar una ejecución paralela de estas tareas que tenga en cuenta las dependencias; y 3) empleando una versión secuencial de BLAS para ejecutar cada tarea individual.

En nuestro trabajo, seguimos un esquema alternativo de tareas paralelas anidadas que explota el paralelismo de tareas a nivel de factorización (es decir, la aplicación) y el paralelismo de bucles dentro de las rutinas invocadas de BLAS (es decir, la "biblioteca"). En el caso de las TA que comprenden rutinas de álgebra lineal de BLAS de gran tamaño y alta carga computacional, hemos comprobado que este enfoque ofrece un rendimiento competitivo en comparación con los esquemas SA+TL y TA+SL [9], [10].

Con el fin de exponer el paralelismo anidado para

el esquema TA+ TL, primero tenemos que reformular el algoritmo básico del Listado 1, para evitar la estricta dependencia de tareas entre las dos operaciones (tareas) que aparecen en el bucle. Para ello, consideramos una variante de este algoritmo por bloques mejorado con *look-ahead* [18], donde una iteración del cuerpo del bucle comprende la actualización de la submatriz con respecto al panel actual y la factorización del "siguiente" panel ($k+1$-ésimo bloque de columnas). En concreto, el cuerpo del bucle reformulado para la factorización QR se muestra en Listado 2.

Listing 2: Factorización QR reformulada.

```
// Actualizar panel k+1 respecto panel k
TU( A(k:s-1,k), A(k:s-1,k+1) );
// Factorizar panel k+1 (si k+1<s)
PF( A(k+1:s-1,k+1) );
// Actualizar paneles k+2:s-1 respecto panel k
TU( A(k:s-1,k), A(k:s-1,k+2:s-1) );
```

Para simplificar, agreguemos la actualización del último $(k+1)$-ésimo panel y la posterior factorización del mismo en una sola operación, de ahora en adelante llamada actualización del panel (y encapsulada dentro de PU). Esto resulta en el algoritmo simplificado para la factorización QR con look-ahead del Listado 3.

Listing 3: Rutina simplificada para la factorización QR con look-ahead.

```
void QR_LA( matrix A, int s )
{
  // Factorizar primer panel
  PF( A(:,0) );
  for ( k = 0; k < s; k++ ) {
    // Actualizar panel k+1 respecto a panel k
    // y factorizar panel k+1 (si k+1<s)
    PU( A(k:s-1,k), A(k:s-1,k+1) );
    // Actualizar paneles k+2:s-1 respecto a
    // panel k
    TU( A(k:s-1,k), A(k:s-1,k+2:s-1) );
  }
}
```

Comparado con el algoritmo convencional para la factorización QR (es decir, la realización sin look-ahead del Listado 1), la variante con look-ahead cuenta con un cuerpo de bucle que consta también de dos tareas: PU y TU, donde la primera sigue siendo mayoritariamente secuencial mientras que la segunda puede realizarse mediante BLAS-3 altamente paralelo. Sin embargo, a diferencia del algoritmo estándar, las dos tareas del cuerpo del bucle son independientes y, por tanto, pueden ejecutarse en paralelo. La principal ventaja de la reformulación look-ahead del algoritmo es que, debido a la independencia entre las dos operaciones en el cuerpo del bucle, se puede solapar la ejecución de la actualización pequeña y secuencial del panel con la de la actualización grande y paralela de la submatriz. Esto es importante ya que, a medida que aumenta el número de núcleos, la actualización del panel no puede aprovechar el creciente volumen de recursos hardware, y acaba convirtiéndose en un cuello de botella para el rendimiento.

Para explotar la independencia de tareas en la variante look-ahead mediante paralelismo anidado (TA+TL) procedemos como sigue: dividimos los hilos en dos equipos de forma que en cada iteración del bucle, el equipo T_P se encarga de la ejecución de PU y el equipo T_T de TU. Además, para manejar los diferentes grados de paralelismo de las dos tareas, asignamos muchos más hilos al equipo T_T que al equipo T_P.

Durante la factorización, la relación entre las operaciones en coma flotante (flops) realizadas dentro del PU y el TU varía, introduciendo dos fuentes de desequilibrio en la carga de trabajo:

- *W1* En caso de que PU (que muestra poco paralelismo,) tarde más en ejecutarse que TU, el equipo T_T terminará su trabajo antes que el equipo T_P, y la cantidad (mucho mayor) de subprocesos asignados al equipo permanecerá inactivo hasta el comienzo de la siguiente iteración, perdiendo valiosos recursos. Para resolver este problema, podemos aplicar un mecanismo de terminación anticipada (desarrollado en [9] para la factorización LU) que finaliza la ejecución de PU (y retrasa la parte pendiente de la factorización del panel a la iteración siguiente) en cuanto el equipo T_T notifica que ha terminado su trabajo. Esto equivale a un ajuste dinámico y adaptativo del tamaño de bloque algorítmico b.
- *W2* En el otro extremo, el coste de PU puede ser considerablemente menor que el de TU, dando lugar a que el equipo T_P complete su ejecución antes que T_T, de nuevo desperdiciando recursos. En este caso, aprovechamos *MLB* para hacer cumplir que, tan pronto como los hilos en el equipo T_P hayan terminado con PU, se unan a los del equipo T_T para la ejecución colaborativa de TU.

En [10], analizamos cómo integrar el paralelismo de tareas con MLB, desde el punto de vista de la programación paralela. En dicho trabajo se evaluaba esta solución para tres factorizaciones matriciales: LU (con pivoteo parcial), QR, y reducción a matriz banda. Sin embargo, este estudio tenía algunas simplificaciones relevantes: 1) el equipo encargado de la factorización del panel estaba formado por un único hilo; 2) la solución no incluía la terminación anticipada; y 3) el mecanismo de maleabilidad no estaba integrado en BLIS. En este trabajo superamos estas simplificaciones para ofrecer una evaluación experimental más completa de las ventajas del esquema *TA+TL* anidado.

A. Evaluación del rendimiento

Los siguientes experimentos se han llevado a cabo en una plataforma equipada con un procesador Intel Xeon Gold 6138 de 20 núcleos (microarquitectura Skylake). Los códigos de referencia se han compilado con `icc` versión 18 y enlazado con OpenMP 4.5 y BLIS v0.5.1 o MKL 2018.1.163, el mecanismo de maleabilidad se ha integrado en BLIS v0.5.1 [17]. Todos los experimentos se han realizado en doble precisión.

Además, para evitar las distorsiones de rendimiento causadas por la utilización agresiva de los modos de potencia (y frecuencias asociadas) que ofrece el controlador Linux para el procesador Intel Gold 6138 [19], la frecuencia de funcionamiento para todos los núcleos se ha fijado a 1,7 GHz.

La Figura 3 muestra el rendimiento de los siguientes códigos para la factorización QR de matrices cuadradas ($m = n$):

- **MKL.** Implementación multihilo de esta operación en Intel MKL. Como esta biblioteca ofrece una solución "black-box", no es posible inferir cómo se explota el paralelismo desde la implementación de Intel.
- **SQR+MKL/BLIS.** Implementación en C de la rutina LAPACK para calcular la factorización QR (rutina `dgeqrf`) mejorada con algunas optimizaciones algorítmicas (en particular, en el esquema para acumular las transformadas ortogonales [20]) y enlazada con instancias multihilo (TL) de Intel MKL o BLIS para ejecutar las rutinas invocadas desde dentro de LAPACK. La implementación se basa exclusivamente en rutinas BLAS-3, es decir, no se instancian rutinas BLAS-1 ni BLAS-2. Nótese que esto corresponde a un esquema SA+TL ya que el algoritmo para la factorización QR es "secuencial" pero invoca rutinas BLAS de una biblioteca de álgebra lineal multihilo.
- **TQR+LA.** Variante de la implementación SQR modificada para codificar el mecanismo de look-ahead, con paralelismo de tareas usando OpenMP. Esta es la versión base para obtener un esquema TA+TL.
- **TQR+MLB.** La misma variante que TQR+LA, pero enlazada con la versión maleable de BLIS y mejorada con el mecanismo de terminación anticipada.

TQR+LA y TQR+MLB son ambas implemetaciones de esquemasTA+TL. Para estas dos opciones evaluamos tres configuraciones, que dividen los hilos en dos equipos, $T_P + T_T$, con el equipo encargado de la actualización del panel compuesto por $p = 1$, 2 o 3 hilos, y los restantes hilos $t - p$ dedicados a la actualización de de la submatriz. Esto se indica en los gráficos para $t = 12, 20$ núcleos con la adición de un sufijo de la forma "p+(t-p)" a la leyenda de la línea. Por ejemplo, para el gráfico con 12 núcleos, la línea etiquetada como "TQR+MLB 2+10" indica la ejecución de la variante TQR+MLB con dos hilos asignados inicialmente a la actualización del panel y 10 a la actualización de la submatriz. Obsérvese que, al tratarse de una variante maleable, los dos hilos encargados del panel pasan a colaborar en la actualización en cuanto se completa esta tarea. En todos los casos (excepto **MKL**, para el que no tenemos control) el tamaño de bloque algorítmico se ha ajustado manualmente para optimizar el rendimiento y se ha establecido en 384.

La Figura 3 muestra los resultados de la ejecución de las rutinas para la factorización QR utilizando 12 y 20 núcleos (es decir, el socket completo en este último caso). Centrándonos en las curvas SQR, los gráficos revelan que la versión enlazada con Intel MKL supera a la de BLIS por un amplio margen. La razón de ello es que MKL es capaz de extraer un mayor rendimiento paralelo que BLIS para ciertas formas del producto general de matrices (GEMM) (en particular, las que implican paneles muy estrechos) que aparecen principalmente en PU. Afortunadamente, la introducción del mecanismo de look-ahead (líneas TQR+LA) oculta parcialmente esta degradación del rendimiento para dimensiones de problema medias y grandes. Además, para el esquema TQR+LA la mejor configuración de hilos depende en gran medida de la dimensión del problema. Para problemas pequeños, PU puede convertirse en un cuello de botella de rendimiento y, por tanto, es necesario dedicar más hilos a su ejecución que para los casos de problemas grandes. En varios experimentos independientes (no mostrados por brevedad) comprobamos que el uso de más de tres hilos para ejecutar PU no aumenta el rendimiento global de la factorización QR. Para problemas grandes, donde PU no es un cuello de botella y su coste de ejecución está completamente oculto con el de TU, sólo necesitamos emplear un hilo para PU. En cuanto a los resultados de TQR+MLB, los gráficos revelan que la integración de maleabilidad alcanza tasas de rendimiento siempre superiores a las de sus homólogos que sólo aprovechan el look-ahead. Al añadir esta técnica no aparecen hilos ociosos durante la ejecución de toda la factorización QR y, en consecuencia, los recursos se utilizan de forma más eficiente. Por último, el enfoque propuesto en este trabajo supera a la rutina de MKL para la factorización QR de problemas medianos y grandes cuando se utilizan 12 núcleos. Sin embargo, este comportamiento no se observa para 20 núcleos porque con esta configuración, la ejecución de PU se vuelve crítica, y las rutinas de BLIS no son tan eficientes como los de MKL con las formas de operandos que aparecen en esta operación de álgebra lineal en particular.

IV. Algoritmos de Inferencia

Las redes neuronales profundas (DNN) consisten en una gran colección de capas de neuronas interconectadas, donde cada capa realiza una operación en sus entradas para producir las activaciones de salida que se pasan a la capa siguiente. En el proceso de inferencia de las redes neuronales convolucionales (CNN), estos cálculos son en gran medida equivalentes a una GEMM o una convolución [21] seguida de la aplicación elemental de una función no lineal. Además, bajo ciertas condiciones, las convoluciones pueden ser eficientemente moldeadas en términos de una GEMM ampliada a través de la transformación *im2col* [22].

El algoritmo de inferencia se ilustra con un alto nivel de abstracción en el Listado 4. Este consiste en un bucle "infinito" que comienza con una llamada de bloqueo para extraer una muestra (o colección

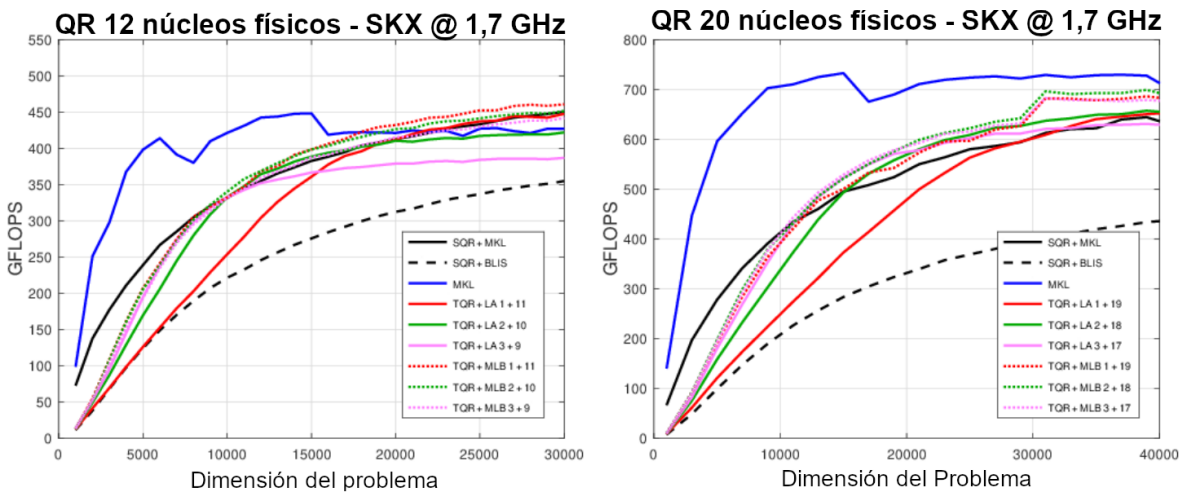


Fig. 3: Rendimiento de la factorización QR.

de muestras) del buffer de entrada I y pasarlo al búfer de activación A_1 (Capa 1 o capa de entrada). A continuación, estos datos son procesados en serie por la secuencia de capas que definen el modelo: para $l = 2, 3, \ldots, L$, la capa l recibe las activaciones de entrada A_{l-1} y aplica las transformaciones definidas por el modelo en esa capa, dadas por W_l, para producir las salidas, A_l, que se convertirán en las activaciones de entrada de la capa siguiente. El resultado final se pasa al buffer de salida (capa) O.

Listing 4: Rutina simplificada para inferencia con una DNN consistente en L capas.

```
void Infer( Input_buffer I, DNN_model W,
            Output_buffer O )
{
  while ( true ) {
    // Extraer muestra(s) de búfer I (Capa 1)
    Extract_sample( I, A_1 );
    // Aplicar transf. a capas 2, 3,...,L
    Layer2( A_1, W_2, A_2 );
    Layer3( A_2, W_3, A_3 );
    Layer4( A_3, W_4, A_4 );
    ...
    LayerL( A_{L-1}, W_L, A_L );
    // Producir resultado(s)
    Output_result( A_L, O );
  }
}
```

El escenario al que nos dirigimos en este caso de estudio refleja un servidor de inferencia de alto rendimiento que opera en el *edge*, que permanece inactivo hasta que recibe una muestra, por ejemplo, de uno de entre varios dispositivos conectados, a través del búfer de entrada. A continuación, el servidor debe aplicar un modelo DNN específico al *batch* de muestras para producir un resultado en un tiempo determinado. Mientras esto ocurre, pueden llegar nuevas muestras, de la misma u otra fuente, que se ponen en cola en el búfer de entrada hasta que el servidor esté listo para procesarlas.

Ahora bien, como se argumentó al principio de esta sección, cuando se trata de CNNs la mayoría de las capas implican una convolución, y este tipo de rutina domina el coste global de la inferencia. Por ejemplo, una capa completamente conectada (FC) con n_{l-1} entradas y n_l salidas, que recibe un *batch* de b muestras, se reduce a una GEMM de dimensiones definidas por estos tres parámetros: n_{l-1}, n_l y b. Por otro lado consideremos una capa convolucional que comprende un operador de convolución que consta de k_n filtros de dimensión $k_h \times k_w \times c_i$ cada uno. Supongamos que la capa recibe b muestras de dimensión $h_i \times w_i \times c_i$ cada una; y produce como salida b tensores de dimensión $h_o \times w_o \times k_n$ cada uno. Utilizando la transformada *im2col* la convolución se puede aplicar en términos de una GEMM con las dimensiones definidas por k_n, $(h_o \cdot w_o \cdot b)$ y $(k_h \cdot k_w \cdot c_i)$. Otros tipos de capas DNN presentan una contribución menor al coste aritmético y al tiempo de ejecución de una DNN.

Los retos para el proceso de inferencia en un sistema multinúcleo surgen de las restricciones de los tiempos de respuesta de muchas aplicaciones prácticas (por ejemplo, el reconocimiento de objetos en tiempo real), la llegada impredecible de los *batches* (y a veces incluso del número de muestras de cada *batch*), y los costes computacionales de las distintas capas. En estas condiciones un esquema SA+TL prioriza la ejecución del *batch* "actual" pero puede violar un acuerdo de nivel de servicio sobre el tiempo de respuesta de los *batches* pendientes en el búfer de entrada; además, puede resultar en un desperdicio de hilos durante la ejecución de capas que implican GEMMs pequeñas. Por otro lado, un esquema TA+SL puede producir una ejecución subóptima debido a conflictos durante el acceso a recursos compartidos como la memoria principal y (ciertos niveles de) caché; y también, si el número de *batches* en el búfer de entrada es pequeño, el grado de paralelismo de tareas es bajo, y por tanto varios hilos permanecerán inactivos. La tercera opción es seguir el esquema TA+TL mejorado con maleabilidad.

A. *Evaluación del rendimiento*

Los experimentos para el caso de uso de aprendizaje profundo (DL) en esta sección se han llevado a cabo en una plataforma equipada con un procesador ARMv8 de 48 núcleos (Huawei Kunpeng 920). Como en el caso anterior, se hace uso de la maleabi-

lidad integrada en BLIS v0.5.1 [17]. En esta plataforma se ha utilizado `gcc` 7.5.0 como compilador y los códigos se han enlazado con OpenMP 4.5. Todos los experimentos se han realizado utilizando 24 núcleos y aritmética de coma flotante IEEE de 32 bits. La frecuencia de funcionamiento de todos los núcleos es de 2,6 GHz.

El modelo DNN que se emplea para la siguiente evaluación corresponde a ResNet-50 [23] (pero se han obtenido resultados similares para AlexNet así como variantes de modelos VGG). Analizamos dos variantes de un escenario que emula, por ejemplo, un coche autónomo equipado con varias cámaras que capturan y envían imágenes a un servidor de inferencia centralizado (en el vehículo). En ambos casos, el servidor procesa las imágenes (muestras) por *batches*, cada *batch* incluye las muestras recibidas en los últimos 0,1 s. En el primer caso, cada *batch* contiene el mismo número de muestras: 1, 2, 4 u 8. En el segundo caso, cada *batch* puede constar ahora de un número distinto de muestras: entre 1 y 8. Estas dos variantes reflejan casos en los que todas las cámaras capturan imágenes al mismo ritmo o a ritmos diferentes.

Consideramos 7 esquemas de paralelización para el caso de estudio DL: la configuración base (etiquetada como SDL–BLIS) aprovecha todos los recursos *(desplegando 24 hilos y, por tanto, utilizando 24 núcleos)* para ejecutar la inferencia de un único *batch* a la vez. Alternativamente, existen tres esquemas que dividen estos recursos en dos, tres o cuatro equipos (con 12, 8 o 6 hilos/núcleos por equipo, respectivamente), encargándose estos equipos de la ejecución simultánea de dos, tres o cuatro *batches* concurrentes (etiquetados como TDL 12+12, TDL 8+8+8 y TDL 6+6+6+6). Estos esquemas de paralelización multiequipo tienen sus homólogos maleables, enlazados con una versión maleable de BLIS, que dividen asimétricamente los hilos en 8+16 (dos equipos), 7+7+10 (tres equipos) o 5+5+5+9 (cuatro equipos). En las soluciones maleables, un equipo está formado por un único subproceso OpenMP que genera tantos subprocesos BLIS como sean necesarios. Por ejemplo, cuando hay dos equipos, se crean dos subprocesos OpenMP al principio y cada uno genera 11 subprocesos BLIS adicionales para los cálculos.

En DL, todos los equipos ejecutan exactamente el mismo código, pero aún así puede producirse un desequilibrio en la carga porque los equipos pueden estar ejecutando capas DNN distintas en un momento dado o tienen que tratar con *batches* de distinto tamaño. En este caso de estudio, aprovechamos la maleabilidad para aumentar el número de recursos asignados a un equipo que esté "ejecutando actualmente" una capa costosa. En ResNet-50, esto ocurre sobre todo durante la ejecución de la primera capa (debido a las dimensiones de la convolución que se ejecuta). Así, el objetivo es que un equipo que comience la ejecución de esa capa, tome prestados hilos de los demás equipos. Por ejemplo, en el caso de tres equipos (con 7, 7 y 10 hilos por equipo) cuando uno de ellos llega a la ejecución de la primera capa, establece el número de hilos de los otros equipos a 7 y su propio número de hilos a 10 (a menos que también estén ejecutando la misma capa inicial).

Analizamos el rendimiento de los esquemas descritos anteriormente utilizando tres métricas clave: 1) el tiempo total de ejecución necesario para procesar un determinado número de *batches*; 2) el tiempo de ejecución por *batch*; y 3) el tiempo de espera por *batch* (es decir, el tiempo que transcurre desde que el *batch* llega a la cola de entrada hasta que se procesa). El tiempo total de ejecución pone de manifiesto las ventajas de una ejecución concurrente de varios *batches*. Por un lado, el tiempo de ejecución por *batch* ilustra el impacto en el rendimiento debido al aumento del número de equipos (que se traduce en una disminución del número de hilos por equipo). Por otra parte, el tiempo de espera por *batch* muestra las ventajas reales de la ejecución multiequipo desde el punto de vista de la satisfacción de una determinada restricción en el tiempo de respuesta.

Los dos gráficos superiores de la Figura 4 demuestran que al aumentar el número de equipos concurrentes se reduce el tiempo total de ejecución para la inferencia con *batches* tanto de tamaño fijo como variable. La razón es que las capas de RestNet-50 no son lo suficientemente complejas (desde el punto de vista computacional) como para aprovechar plenamente los 24 núcleos. Como resultado, la ejecución de varios procesos de inferencia en paralelo reduce el tiempo total de ejecución en un factor de 1,85× cuando se utilizan dos equipos, 2,4× con tres equipos, y un factor de 2,7× con cuatro equipos, tanto para escenarios de tamaño de *batch* fijo como variable. La maleabilidad contribuye a un aumento del rendimiento de hasta el 5 % (dos equipos), el 10 % (tres equipos) y el 13 % (cuatro equipos) en el caso de tamaño de *batch* fijo, y de hasta el 4 % (dos equipos), 3 % (tres equipos) y 4 % (cuatro equipos) en el caso de tamaño de *batch* variable.

Los gráficos del centro de la Figura 4 representan el tiempo medio de ejecución. Los resultados muestran que añadir más equipos (lo que implica reducir los recursos por equipo) aumenta el tiempo de ejecución por *batch*. Este crecimiento no es lineal con el tamaño del *batch*, ya que algunas capas implican cálculos ligados a la memoria, con una escalabilidad paralela reducida. Además, observamos que dividir los recursos entre dos equipos no aumenta el tiempo de ejecución por *batch*. La razón es la baja complejidad de las capas individuales, que no permite que el esquema de dos equipos se beneficie de toda la potencia de cálculo del procesador. Sin embargo, al añadir el tercer equipo se produce un incremento del tiempo de ejecución por *batch*. El efecto de la maleabilidad es visible cuando se compara con su correspondiente contrapartida no maleable, con una reducción del tiempo para el esquema de dos equipos de hasta un 11 % con un tamaño de *batch* de 4 y de hasta un 7 % con un tamaño de *batch* aleatorio. Para el esquema de tres equipos, el aumento del rendimiento es de hasta un 4 % en el primer escenario y de hasta un

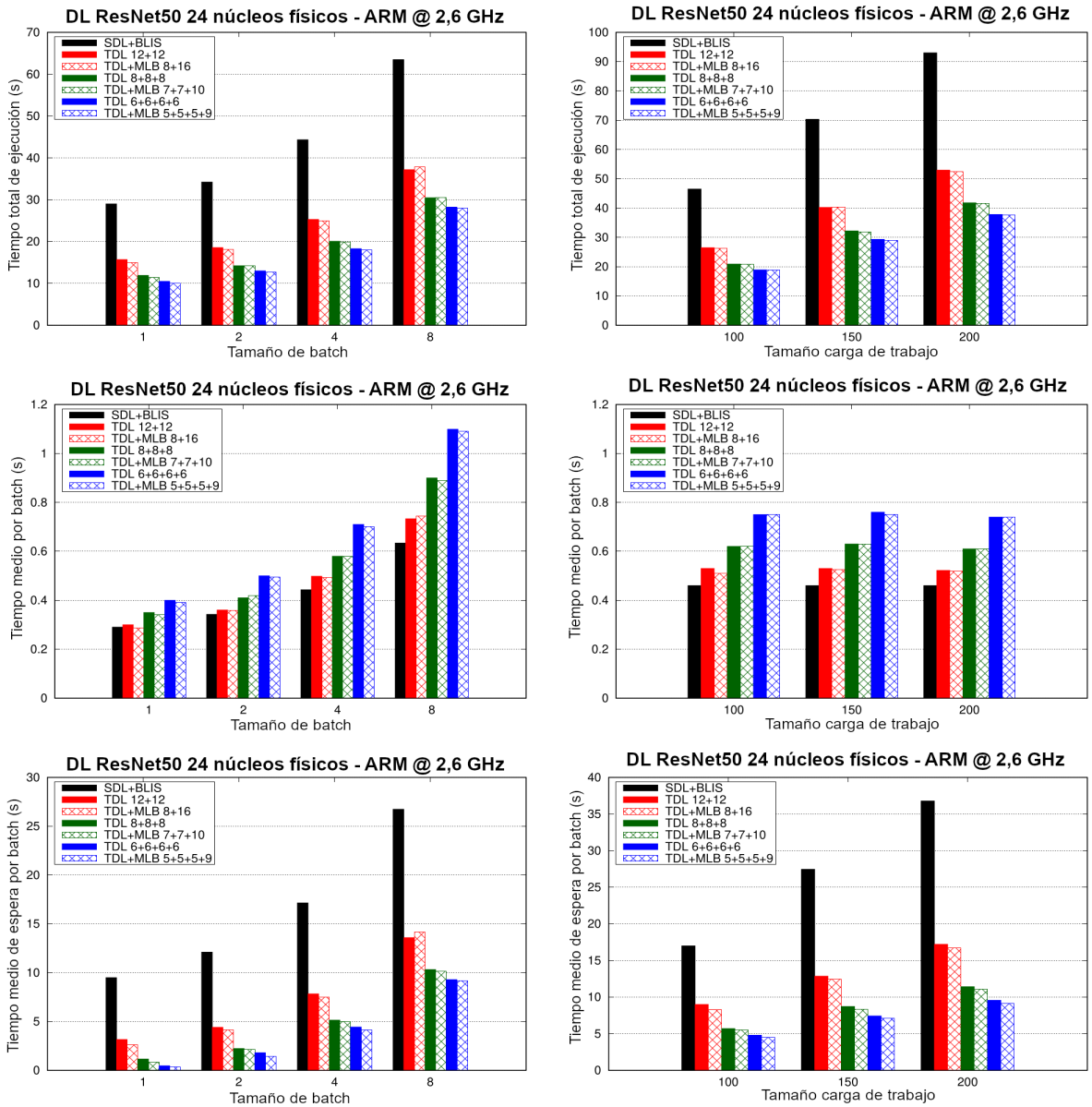


Fig. 4: Tiempo total de ejecución, tiempo medio de ejecución por *batch* y tiempo medio de espera por *batch* (arriba, centro y abajo, respectivamente) para realizar la inferencia con ResNet-50 con 50 *batches* de varios tamaños (izquierda) y tres *batches* de tamaños aleatorios (derecha).

3 % con un tamaño de *batch* aleatorio. Para el esquema de cuatro equipos, la maleabilidad añade un 4 % de mejora para el tamaño de *batch* fijo y hasta un 5 % para el escenario aleatorio. También hay una pérdida de rendimiento en el esquema de dos equipos cuando se fija el tamaño del *batch*. Esto puede deberse al efecto negativo de que un equipo robe un hilo a otro equipo que también puede estar ejecutando una capa crítica. Esta penalización del rendimiento se compensa no sólo en el tiempo total de ejecución, sino también en el tiempo medio de espera por *batch*.

Por último, los dos gráficos inferiores de la figura ilustran las ventajas de aumentar el número de equipos de ejecución simultánea en el tiempo de espera por *batch*: utilizar dos, tres y cuatro equipos disminuye el tiempo medio de espera en factores de hasta 3×, 8× y 26× cuando el tamaño del *batch* es fijo; y de hasta 2,13×, 3,22× y 3,8× cuando el tamaño del *batch* es variable; respectivamente. Al igual que en las pruebas anteriores, la maleabilidad proporciona un aumento adicional del rendimiento de hasta el 12 %, el 14 % y el 23 % para dos, tres y cuatro equipos, respectivamente.

V. Conclusiones

En este artículo, hemos evaluado las ventajas de integrar la maleabilidad en BLIS para evitar la rigidez de las instancias actuales de BLAS, para las que el número de hilos utilizados en la ejecución de una rutina es fijo de principio a fin. En la práctica, esta nueva funcionalidad se expone al programador a través de una modificación mínima de la API experta de BLIS. El programador sólo tiene que identificar dónde debe modificarse la distribución de los hilos y el cambio se realiza automáticamente.

Los resultados de rendimiento demuestran que las ventajas de este enfoque son amplias en escenarios en los que el paralelismo se extrae tanto a nivel de

aplicación como de biblioteca. Más concretamente, cuando la carga de trabajo no está igualmente equilibrada, la maleabilidad permite modificar, en tiempo de ejecución, el paralelismo a nivel de biblioteca con el fin de mejorar la ocupación global de los núcleos.

Como trabajo futuro, creemos que la maleabilidad también puede ofrecer ventajas significativas en la programación de tareas en tiempo de ejecución, por ejemplo, SuperMatrix de libflame [24] o modelos de programación basados en tareas como StarPU [25] y OmpSs [26]. En ellos, el escaso paralelismo a nivel de tarea en algunas partes de la aplicación [8] puede ser compensado de forma dinámica con el paralelismo dentro de las tareas. Por lo tanto, una biblioteca totalmente maleable es indispensable.

Agradecimientos

Este trabajo cuenta con el apoyo de las subvenciones PID2020-113656RB-C22, PID2021-123627OB-C52, RTI2018-093684-B-I00 y PID2021-126576NB-I00 financiado por MCIN/AEI/10.13039/50-1100011033 y por "ERDF Una forma de hacer Europa", subvención S2018/TCS-4423 del *Comunidad Autónoma de Madrid*, por el Convenio Plurianual con la UCM en la línea Programa de Estímulo a la Investigación para Jóvenes Doctores en el contexto del V PRICIT con cargo al proyecto PR65/19-22445, y el proyecto Prometeo/2019/109 de la *Generalitat Valenciana*. A. Castelló es becario FJC2019-039222-I con el apoyo de MCIN/AEI/10.13039/501100011033. Sandra Catalán está contratada con la ayuda RYC2021-033973-I, financiada por MCIN/AEI/10.13039/501100011033 y Unión Europea NextGenerationEU/PRTR.

Referencias

[1] Sergio Iserte, Rafael Mayo, Enrique S. Quintana-Ortí, Vicenç Beltran, and Antonio J. Peña, "Efficient scalable computing through flexible applications and adaptive workloads," in *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*, 2017, pp. 180–189.

[2] Sergio Iserte, Rafael Mayo, Enrique S. Quintana-Ortí, Vicenç Beltran, and Antonio J. Peña, "DMR API: Improving cluster productivity by turning applications into malleable," *Parallel Computing*, vol. 78, pp. 54 – 66, 2018.

[3] OpenMP Architecture Review Board, "OpenMP application programming interface. Version 5.0," November 2018.

[4] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff, "A set of level 3 basic linear algebra subprograms," *ACM Trans. Math. Softw.*, vol. 16, no. 1, pp. 1–17, Mar. 1990.

[5] Intel, "Math Kernel Library," `https://software.intel.com/en-us/intel-mkl`, 2020.

[6] "OpenBLAS," `http://www.openblas.net`, 2015.

[7] Field G. Van Zee and Robert A. van de Geijn, "BLIS: A framework for rapidly instantiating BLAS functionality," *ACM Trans. Math. Softw.*, vol. 41, no. 3, pp. 14:1–14:33, 2015.

[8] Manuel F. Dolz, Francisco D. Igual, Thomas Ludwig, Luis Piñuel, and Enrique S. Quintana-Ortí, "Balancing task- and data-level parallelism to improve performance and energy consumption of matrix computations on the intel xeon phi," *Computers & Electrical Engineering*, vol. 46, pp. 95 – 111, 2015.

[9] Sandra Catalán, Jose R. Herrero, Enrique S. Quintana-Ortí, Rafael Rodríguez-Sánchez, and Robert Van De Geijn, "A case for malleable thread-level linear algebra libraries: The LU factorization with partial pivoting," *IEEE Access*, vol. 7, pp. 17617–17633, 2019.

[10] Sandra Catalán, Adrián Castelló, Francisco D. Igual, Rafael Rodríguez-Sánchez, and Enrique S. Quintana-Ortí, "Programming parallel dense matrix factorizations with look-ahead and OpenMP," *Cluster Computing*, vol. 23, pp. 359–375, 2020.

[11] Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra, "A class of parallel tiled linear algebra algorithms for multicore architectures," *Parallel Computing*, vol. 35, no. 1, pp. 38 – 53, 2009.

[12] Gregorio Quintana-Ortí, Enrique S. Quintana-Ortí, Robert A. van de Geijn, Field G. Van Zee, and Ernie Chan, "Programming matrix algorithms-by-blocks for thread-level parallelism," *ACM Trans. Math. Softw.*, vol. 36, no. 3, pp. 14:1–14:26, 2009.

[13] Rosa M. Badia, José R. Herrero, Jesús Labarta, Josep M. Pérez, Enrique S. Quintana-Ortí, and Gregorio Quintana-Ortí, "Parallelizing dense and banded linear algebra libraries using SMPSs," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 18, pp. 2438–2456, 2009.

[14] Gene H. Golub and Charles F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

[15] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec 2017.

[16] Tze Meng Low, Francisco D. Igual, Tyler M. Smith, and Enrique S. Quintana-Ortí, "Analytical modeling is enough for high performance BLIS," *ACM Trans. Math. Soft.*, 2014, In review. Available at `http://www.cs.utexas.edu/users/flame`.

[17] Rafael Rodríguez-Sánchez, Francisco D. Igual, and Enrique S. Quintana-Ortí, "Integration and exploitation of intra-routine malleability in BLIS," *J. Supercomput.*, vol. 76, no. 4, pp. 2860–2875, 2020.

[18] Peter Strazdins, "A comparison of lookahead and algorithmic blocking techniques for parallel matrix factorization," Tech. Rep. TR-CS-98-07, Department of Computer Science, The Australian National University, Canberra 0200 ACT, Australia, 1998.

[19] Intel, "Intel® Xeon® Processor Scalable Family. Specification udpate," Tech. Rep., November 2019, `https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/xeon-scalable-spec-update.pdf`.

[20] Thierry Joffrain, Tze Meng Low, Enrique S. Quintana-Ortí, Robert van de Geijn, and Field G. Van Zee, "Accumulating Householder transformations, revisited," *ACM Trans. Math. Softw.*, vol. 32, no. 2, pp. 169–179, June 2006.

[21] Catherine F. Higham and Desmond J. Higham, "Deep learning: An introduction for applied mathematicians," 2018, arXiv:1801.05894.

[22] Kumar Chellapilla, Sidd Puri, and Patrice Simard, "High performance convolutional neural networks for document processing," in *International Workshop on Frontiers in Handwriting Recognition*, 2006, Available as INRIA report inria-00112631 from `https://hal.inria.fr/inria-001126`.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[24] Ernie Chan, Field G. Van Zee, Paolo Bientinesi, Enrique S. Quintana-Ortí, Gregorio Quintana-Ortí, and Robert van de Geijn, "Supermatrix: A multithreaded runtime scheduling system for algorithms-by-blocks," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, New York, NY, USA, 2008, pp. 123–132, ACM.

[25] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier, "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures," *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009*, vol. 23, pp. 187–198, Feb. 2011.

[26] Alejandro Duran, Eduard Ayguadé, Rosa M. Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas, "OmpSs: a proposal for programming heterogeneous multi-core architectures," *Parallel Processing Letters*, vol. 21, no. 2, pp. 173–193, 2011.

Diseño y Exploración de Aceleradores Hardware en la Plataforma SELENE

Pablo Andreu[1], Tomas Picornell[1], Vatistas Kostalampros [2], Pedro Lopez[1], Jose Flich[1], Miquel Moreto[2] y Carles Hernandez[1]

Resumen— **El diseño de aceleradores hardware esta adquiriendo una gran importancia debido al aumento de aplicaciones que usan redes neuronales y la aproximación al límite de la ley de Moore. En este contexto, la exploración de las opciones de diseño de estos aceleradores y su impacto en el rendimiento necesita de plataformas donde integrar dichos aceleradores de manera ágil. Con esa finalidad este trabajo presenta la plataforma de SELENE, una plataforma abierta donde es sencillo integrar aceleradores diseñados en Xilinx HLS o con interfaz estándar AXI para su testeo en FPGA.**

Palabras clave— **Hardware abierto, RISC-V, Aceleradores hardware**

I. Introducción

LA ralentización de la ley de Moore [1] en los últimos años ha favorecido el desarrollo de procesadores heterogéneos que incluyen aceleradores hardware para aplicaciones especificas. Este tipo de diseños permite obtener un gran rendimiento en dichas tareas con un consumo de potencia reducido. No obstante los procesadores convencionales cuentan a su vez con numerosas opciones de optimización lo cual dificulta determinar para que tareas la utilización de aceleradores hardware resulta conveniente. En este contexto, disponer de una una plataforma donde integrar aceleradores que permita evaluar el rendimiento de los mismos en comparación con la ejecución monolítica resulta de especial utilidad.

La integración de aceleradores junto a procesadores de propósito general, permite a los arquitectos de computadores analizar el rendimiento que estos proporcionan en tareas reales teniendo en cuenta el impacto de la comunicación entre procesadores, memoria y aceleradores y verificar su correcto funcionamiento. Sin este proceso de análisis de la arquitectura integrada resulta difícil evaluar la conveniencia o no de incorporar aceleradores hardware.

En este artículo presentamos la plataforma de código abierto SELENE como un vehículo para la exploración de arquitecturas heterogéneas. La capa hardware de SELENE implementa un sistema multiprocesador con procesadores RISC-V de 64 bits, una cache de nivel 2 compartida y una red de interconexión para la integración de aceleradores, permitiendo el acceso a memoria compartido entre estos y los procesadores RISC-V. En la parte software, SELENE tiene soporte para Linux e hipervisores basados en RISC-V como Jailhouse o XtratumNG.

Para integrar aceleradores en un System-on-chip (SoC) a un coste reducido, nuestra plataforma de evaluación requiere de las siguientes las características :

1. Interoperabilidad: La plataforma debe contar con soporte para buses estándar, como aquellos propuestos por AMBA (Advanced Microcontroller Bus Architecture). Dichos buses permiten la integración sencilla de aceleradores que integren dichos protocolos.
2. Flexibilidad: La plataforma debe soportar la máxima cantidad de configuraciones de los buses que implemente posibles, de esta manera se pueden integrar aceleradores de todo tipo.
3. Soporte software: La plataforma debe proveer soporte para interactuar con los aceleradores integrados mediante una librería software desde un sistema operativo.

Con el fin de facilitar el proceso de exploración de arquitecturas heterogéneas en SELENE se ha implementado soporte específico para la integración de aceleradores. Este soporte permite la rápida integración de aceleradores hardware que pueden ser fácilmente descritos en C y utilizar herramientas de síntesis de circuitos para lenguajes de alto nivel (HLS [2]). EL módulo hardware resultante de la síntesis con la herramienta Xilinx HLS se integra de manera automática en nuestro SoC a través de las interfaces AXI4 que proporciona nuestro sistema.

Este artículo esta organizado de la siguiente manera. En la sección 2 se introduce el contexto necesario para entender la necesidad de los aceleradores de tareas y la utilidad de HLS para desarrollar dichos aceleradores. En la sección 3 se presenta la plataforma SELENE, sus redes de interconexión y el software que hace posible instanciar aceleradores. Mostrando el proceso de instanciar aceleradores en dicha plataforma, tanto de manera manual como automática mediante la herramienta presentada. En la sección 4 se presentan dos casos de uso de dos aceleradores instanciados en la plataforma SELENE. Mostrando la capacidad de la plataforma para orquestar dichos aceleradores y generar comparativas de rendimiento y área. En la sección 5 se presentan otras plataformas similares a SELENE y las diferencias de estas plataformas respecto a la presentada. Finalmente en la sección 6 se presentan las conclusiones del trabajo.

II. Contexto

El desarrollo de aceleradores para aplicaciones especificas, como los del proceso de inferencia de redes neuronales o criptografía post-quantica que presen-

[1]Dpto. de Informática y Computadores (DISCA), Universitat Politècnica de València, e-mail: pabance@upv.es
[2]Barcelona Supercomputing Center,

tamos en este articulo, requiere de mucho esfuerzo de diseño y validación. Además, una vez estos aceleradores son diseñados e implementados con lenguajes de descripción de hardware, deben ser integrados en un SoC lo cual añade esfuerzo adicional al proceso de validación y verificación de los mismos.

Para reducir el alto coste de implementación de aceleradores específicos se pueden emplear lenguajes de diseño con un mayor poder de abstracción, como C/C++ y utilizar directivas para la optimización de la síntesis en hardware. Esto se puede hacer con herramientas de síntesis de alto nivel como Xilinx HLS. El uso de esta aproximación permite reducir el tiempo de implementación del acelerador a cambio de reducir ligeramente su rendimiento y consumo en algunos casos.

Por otro lado la integración de dichos aceleradores en una plataforma es esencial para comparar rendimiento de un sistema con diferentes CPUs contra el rendimiento del acelerador, el área de este y su consumo. Para ello se puede emplear técnicas de simulación de lenguajes de descripción hardware o emplear FPGAs (Field Programable Gate Arrays). Las FPGAs permiten sintetizar el diseño completo y comparar métricas de área, consumo y rendimiento con la CPU de manera rápida. Por otro lado simular el rendimiento del diseño, especialmente para proyectos grandes y detallados como los que se necesitan para validar un SoC, es un proceso extremadamente lento en comparación a la implementación del diseño en una FPGA. Esto dificulta la obtención de métricas realistas en el contexto de tareas con altos requerimientos computacionales como la inferencia en redes neuronales.

Nuestra plataforma explota esta reciente tendencia al desarrollo de aceleradores en High Level Synthesis (HLS) integrando una herramienta que permite integrar dichos aceleradores en nuestra plataforma. Nuestra herramienta consigue integrar dichos aceleradores debido a la previa traducción del código HLS a lenguajes de descripción de hardware como VHDL y Verilog realizada por la herramienta de síntesis de HLS.

Las herramientas de HLS como las de Xilinx implementan interfaces de interconexión AXI para el *backend* y AXI-lite para el *frontend*. Eso nos permite integrar dichos diseños implementados en HLS en nuestro sistema de manera sencilla y rápida, ya que nuestro sistema usa una red de interconexión AXI para el *backend* y tiene una red de interconexión AXI-lite, que permite conectar el *frontend* de los aceleradores.

III. La plataforma SELENE

En esta sección se describe la plataforma SELENE y las adaptaciones realizadas para facilitar el proceso de integración y validación de aceleradores hardware.

A. SELENE SoC

Una simplificación de la configuración del hardware de la plataforma SELENE se presenta en la Figura 1. Dicha plataforma es Open-Source y se puede encontrar en el siguiente enlace[1].

Respecto del hardware la plataforma de SELENE implementa un SoC descrito en RTL que incluye seis procesadores de ejecución en orden superescalares de dos vías que poseen caches de datos e instrucciones de 16KB y 4 vías cada uno (Configurables en síntesis). Dichos procesadores NOEL-V poseen una arquitectura RISC-V de 64 bits con las extensiones IMAFD (Integer, Multiplicación de enteros, Atómicos, (F) Coma flotante de precisión simple, (D) Coma flotante de doble precisión). Dicho SoC (System on Chip) posee una cache L2 configurable en vías y política de reemplazo, con un tamaño estándar de 512 KB y 4 vías con la política de remplazo por defecto siendo aleatorio entre vías, pese a que el reemplazo pseudo LRU esta disponible en dicha plataforma.

Estos procesadores NOEL-V son la evolución de los conocidos procesadores de la familia LEON de Cobham Gaisler, siendo esta familia de procesadores de especial relevancia debido a su uso por la Agencia Espacial Europea. Los procesadores NOEL-V tienen soporte para memoria virtual gracias a la MMU (Memory Management Unit) disponible en dicha plataforma, con lo que permiten ejecutar Linux sobre dichas CPUs. La distribución soportada de Linux en dicha plataforma es Debian. La imagen de dicha distribución se puede generar utilizando la herramienta ISAR, siendo esta open source y disponible en el siguiente enlace[2]. Dicha imagen de ISAR Linux contiene el SELENE Accelerator Framework [3], siendo este un modulo del kernel de Linux que añade soporte para los aceleradores que se quieran integrar dentro de la plataforma SELENE.

Respecto a las redes de interconexión internas de la plataforma SELENE, los procesadores están interconectados entre si por la red AHB (Advanced High-performance Bus). Dicha red es parte del estándar AMBA de ARM. Los periféricos a dichos procesadores están conectado usando la interfaz APB (Advanced Peripheral Bus), siendo dicha red muy parecida a la AHB, pero comúnmente usada para periféricos. Respecto a la interconexión de los aceleradores que están actualmente integrados en la plataforma, dichos aceleradores se conectan mediante la interfaz AXI lite. Siendo esta una interfaz muy conocida y usada para la interconexión de periféricos punto a punto. Todos los componentes AHB del sistema forman parte de la librería estándar de componentes de Cobham Gaisler, el fabricante del procesador. Todos los componentes de las redes de AXI/AXI lite están integrados desde los componentes del interconnect de la plataforma PULP [4].

B. Herramienta de integración de aceleradores

Para facilitar la integración de aceleradores en nuestro sistema hemos desarrollado una herramienta capaz de integrar dichos aceleradores en el SoC de SELENE a partir de un fichero de descripción de las

[1]https://gitlab.com/selene-riscv-platform/selene-hardware
[2]https://gitlab.com/selene-riscv-platform/isar-selene

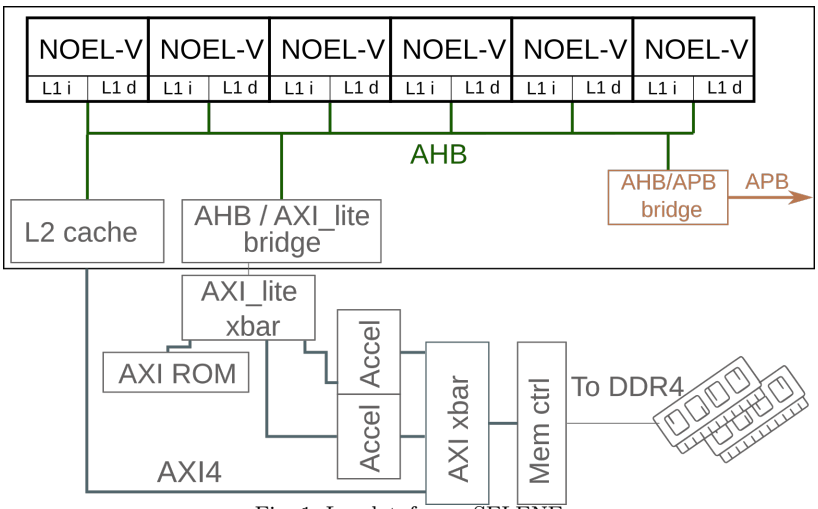


Fig. 1: La plataforma SELENE

interfaces del acelerador.

Además, gracias a la integración de los componentes AXI de la plataforma PULP en nuestro sistema, la instanciación de aceleradores es flexible en numero de puertos de entrada y salida y ancho de las interfaces de AXI. Por ejemplo, para la integración de el acelerador HLSinf [5] en nuestro diseño, nuestra herramienta automática de instanciación de aceleradores en SELENE produce la instancia mostrada en la Figura 2

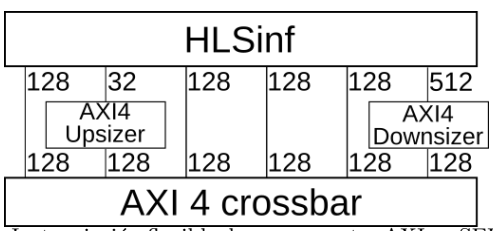


Fig. 2: Instanciación flexible de componentes AXI en SELENE

En dicha instancia como vemos, la herramienta automática de instanciación de aceleradores HLS ha descubierto que el acelerador de redes neuronales HLSinf presenta seis puertos, de los cuales dos no tienen el tamaño del bus AXI conectado al crossbar. Dicha herramienta decide generar up/downsizers que ajustan el tamaño de todos los puertos del acelerador al tamaño de la red, permitiendo una integración simple de los aceleradores en nuestro sistema.

Además de instanciar dichos up-downsizers para ajustar el tamaño de los puertos AXI del acelerador al tamaño de los puertos de nuestra plataforma, dicha herramienta automática ajusta el numero de puertos de entrada del crossbar de memoria para que acepte las nuevas entradas dadas por el acelerador. Así como convierte las interfaces AXI expuestas por el acelerador en los tipos de datos vhdl y verilog que envuelven el estándar AXI en la librería de GRLIB y PULP, reduciendo drásticamente el tiempo de implementación de un acelerador en la plataforma gracias a la generación automática de wrappers.

Esta herramienta de integración puede ser encontrada en el siguiente enlace[3]. Con esta herramienta la integración de módulos de aceleración descritos en HLS es sencilla y rápida. Con esta herramienta se han integrado en la plataforma SELENE el acelerador de redes neuronales HLSinf [5] y el acelerador de criptografía postcuantica de Classic McEliece presentado en [6]

C. Software: Selene Accelerator Framework

El objetivo del SELENE Acceleration Framework (SAF) es ejecutar y acelerar aplicaciones de usuarios en aceleradores hardware instanciados en el SoC de SELENE permitiendo el uso de esto en baremetal o usando Linux. La capa mas baja del SAF es el SELENE Acceleration Runtime (SAR). El objetivo del SAR es poder ejecutar kernels generados por Vitis-HLS como el HLSinf o cualquier acelerador diseñado en HLS que se quiera integrar en SELENE. Para ello proveyendo una API (Interfaz de programación) simple y eficiente a los niveles superiores del software, como seria la librería de inferencia en el caso del acelerador de redes neuronales.

El SAR puede ejecutar múltiples kernels y esta diseñado para poder ser configurado con el JSON resultante de implementar un kernel en HLS. Dichos kernels HLS generan un JSON describiendo la interfaz de memoria y las localizaciones de los registros de configuración de los aceleradores descritos. Si el acelerador que se busca integrar no esta generado por Vivado HLS, se debe generar este JSON con la configuración de dicho acelerador para que el SAR pueda usarlo.

```
1 "Kernels" : [
2 {
3   "name" : "conv",
4   "base_address" : "0xfffc0000",
5   "ports" : {
6     "control" :{
7       "offset" : "0x00",
8       "bits" : {
9         "ap_start" : "0x01",
```

[3]https://gitlab.com/selene-riscv-platform/selene-hardware/-/tree/master/accelerators/hlsIntegrationTool

```
10          "ap_done" : "0x02",
11          "ap_idle" : "0x03",
12          "ap_ready" : "0x04",
13          "ap_continue" : "0x05",
14          "ap_restart" : "0x07"
15          }
16        },
17        "signals" :  [
18          {
19            "name" : "ptr_data",
20            "type" : "buffer",
21            "offset" : "0x10",
22            "data_type": "float",
23            "size" : 64,
24            "pos" : 0
25          }]}}
26  ]
```

Listing 1: JSON de configuración del SAR

Los aceleradores de Xilinx HLS usan memoria física contigua para la entrada y salida de sus datos. Sin embargo, cuando usamos Linux, no podemos escribir directamente en memoria principal, debido a que Linux implementa memoria virtual. Para poder escribir directamente en memoria principal usamos un driver del kernel escrito para SELENE y que esta integrado en la imagen ISAR de Linux.

D. *Herramienta de medición de contención entre actores*

Finalmente la plataforma SELENE también dispone de una herramienta capaz de detectar el incremento de tiempo de ejecución que los aceleradores integrados están generando al código ejecutado por las CPUs en tiempo de ejecución y viceversa. Cabe aclarar que dicho incremento de tiempo de ejecución no es el de la tarea estudiada, sinó el de las distintas tareas del sistema. Dicha herramienta es configurable para obtener estimaciones del tiempo de ejecución de sus aceleradores en la plataforma sin la interferencia de las CPUs y del tiempo de ejecución de las CPUs sin los aceleradores.

Esta herramienta también permite observar la contención que otras CPUs están generando a determinada CPU, con lo que el usuario puede asignar la tarea de control del acelerador a una determinada CPU y calcular el tiempo de ejecución de la tarea estudiada sin interferencia de otras CPUs o aceleradores del sistema.

En resumen, esta herramienta permite obtener medidas del tiempo de ejecución de las diferentes tareas del sistema multiprocesador heterogeneo como si dichas tareas estuvieran funcionando en un sistema monoprocesador, debido a que es capaz de detectar y asignar a la fuente la contención que las diferentes tareas generan en las redes de interconexión.

IV. Integración de aceleradores

Para ilustrar el funcionamiento de la plataforma SELENE para la exploración de arquitecturas heterogéneas se han integrado dos aceleradores hardware. Dichos aceleradores son un acelerador de criptografía post-cuántica llamado PQC y un acelerador de redes neuronales llamado HLSinf.

La integración y testeo de los aceleradores se ha realizado en la PFGA Xilinx VCU118, con el SoC de SELENE sintetizado a 100MHz.

A. *HLSinf*

El HLSinf es un acelerador de redes neuronales escrito en HLS y basado en el paradigma de diseño *dataflow* [7], en el cual los datos fluyen a través del acelerador para realizar una inferencia.

HLSinf es un acelerador de inferencia en redes convolucionales de código libre, cuyo código puede consultarse en [4], dicho acelerador es configurable en dos dimensiones. Primeramente, se pueden añadir módulos nuevos al diseño del pipeline del acelerador, proveyendo mas funcionalidad y capas soportadas.

Finalmente dicho acelerador también puede crecer en potencia de computo, ya que dicho acelerador esta diseñado alrededor del concepto de *channel slicing*, donde diversos canales son leídos en paralelo y diversos canales son producidos en paralelo. Por lo tanto, múltiples instancias del mismo acelerador pueden ser obtenidas cambiando ambos parámetros, ajustándose a la capacidad de computo requerida para la plataforma y el área disponible.

Gracias a las herramientas incluidas en el repositorio abierto previamente mencionado y descritas en la sección anterior, la integración del acelerador HLSinf resulta en una instanciación automática de conversores de anchos de red como se puede observar en la Figura 2.

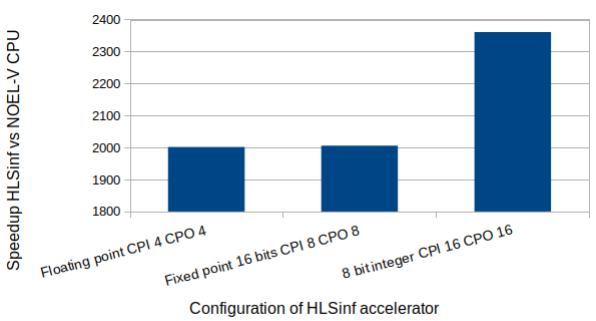


Fig. 3: Speedup de la convolucion de redes neuronales conseguido con el HLSinf en SELENE

Finalmente, para obtener el rendimiento del acelerador integrado en la plataforma, hemos obtenido el tiempo medio de inferencia para una imagen sobre el modelo tiny-yolo v4 en dicha plataforma usando el acelerador y la CPU, comparando ambas. Como se puede observar en la Figura 3, el *speedup* obtenido con el acelerador es de mas de 2000. Dicho speedup es muy grande debido a la baja frecuencia de funcionamiento de la CPU de SELENE y a su diseño en orden. El HLSinf implementado a 250 MHz en comparación con una CPU Intel i7-7800x obtiene un speedup de 2.4/7.1/11 para inferencias con precisión FP32/FP16/INT8 respectivamente.

[4]https://github.com/PEAK-UPV/HLSinf

	LUT	BRAM	DSP
SELENE	524470	360	1358
HLSINF	131199	54	1247
% Area	25 %	15 %	92 %

Tabla I: Utilización de recursos en la FPGA con el HLSinf

	LUT	BRAM	DSP
SELENE	490677	708	111
PQC	97406	348	6
% Area	20 %	49 %	5 %

Tabla II: Utilización de recursos en la FPGA con el PQC

La tabla I presenta el uso de los recursos de la FPGA por el acelerador integrado. Como se puede observar, dicho acelerador hace un uso exhaustivo de los DSP (Digital Signal Processors) de la FPGA debido a su programación en Xilinx HLS, que es muy propensa a implementar DSPs. Sin embargo, para tener un *overhead* de área de menos del 25 % en LUTs y BRAMs el *speedup* en tareas de inferencia de convoluciones de mas de 2000 se considera adecuado. Dicho speedup es posible devido al bajo rendimiento de las CPUs de la plataforma y al enorme rendimiento del acelerador, que si es comparado con un procesador moderno como

A.1 Herramienta de control de contención

Respecto a la herramienta de control de contención previamente mencionada, su funcionamiento se puede observar en los resultados presentados en la Figura 4.

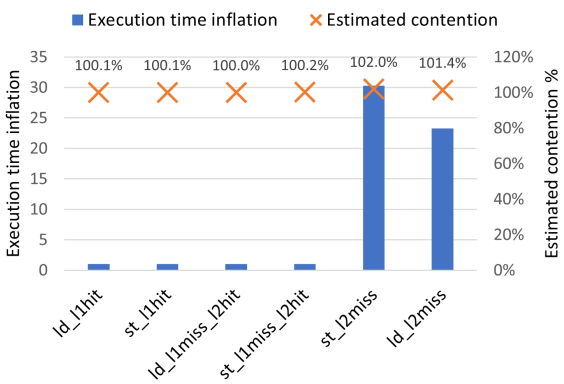


Fig. 4: Inflación en el tiempo de ejecución y contención medida por nuestro mecanismo para diferentes tareas de la CPU con el HLSinf como agresor

En dicha figura se presenta la inflación del tiempo de ejecución que experimentan seis diferentes tareas en una CPU cuando el HLSinf esta ejecutando convoluciones en diferentes imágenes. Como se puede observar en dicha figura, las únicas tareas que ven modificado su tiempo de ejecución son aquellas que acceden a memoria principal haciendo miss en la L2. Esto es debido a que el acelerador y las CPUs comparten el crossbar de memoria principal y el controlador de memoria.

Sin embargo, pese a que la tarea presenta inflaciones de tiempo de ejecución muy altas (de mas de 20 veces mas lento), nuestro mecanismo es capaz de medir dicha inflación de tiempo de ejecución con menos de un 2 % de error. Siendo por lo tanto capaces de aislar el tiempo de ejecución de la tarea ejecutándose en la CPU y la tarea ejecutándose en el acelerador instanciado.

B. PQC

Para la integración del acelerador de Post Quantum Criptography se realiza un proceso similar a el seguido con el acelerador HLSinf. Al pasar dicho acelerador por nuestra herramienta se redimensionan los puertos de AXI4 con up/downsizers y se le asigna una dirección de memoria para su configuración desde el software.

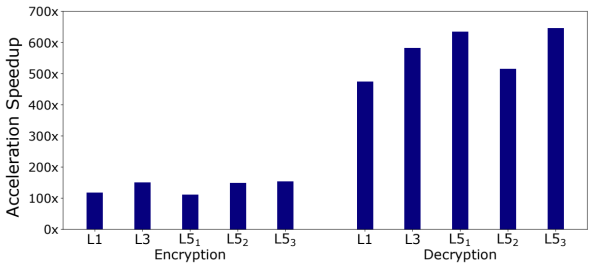


Fig. 5: Speedup de la encriptación y desencriptación de diferentes tareas con el acelerador PQC en SELENE

La figura 5 presenta el *speedup* que presentan los diferentes tipos de tareas de encriptación y decriptación post-quantica cuando funcionan en SELENE a través del acelerador en comparación a funcionando sobre la CPU NOEL-V nativa de dicha plataforma. Como podemos observar, en el proceso de encriptación vemos aceleraciones de 150 veces, mientras que en el proceso de desencriptación, vemos aceleraciones de entre 474 y 645 para los diferentes niveles de seguridad. Dicho aumento en la aceleración de las tareas de desencriptación es debido a su gran intensidad de computo, beneficiándose mucho del incremento de capacidad de computo sin verse muy perjudicado por los dimensionamientos en sus interfaces.

La TablaII presenta el incremento de recursos producido por el acelerador PQC en la plataforma SELENE. Como podemos observar, hay un incremento notable en BRAMS, mientras que los incrementos en LUTs y DSPs son menores. Sin embargo, pese al incremento en las BRAMs utilizadas, dicho acelerador PQC cabe en la FPGA de prototipado que teníamos como objetivo, por lo que se considera una integración adecuada.

V. Otras plataformas similares

En la actualidad, una de las principales plataformas de hardware libre es chipyard [8], siendo esta una plataforma modular basada en chisel [9], un lenguaje de programación basado en escala que busca simplificar el proceso de diseño hardware. Sin embargo, dicha plataforma requiere conocimiento del lenguaje chisel para el integrador del modulo hardware, ya que requiere de escribir un *wrapper* chisel para cada modulo integrado. Dicha plataforma es adecua-

da para integradores de aceleradores hardware que conozcan chisel y hayan desarrollado su acelerador hardware con la interfaz tilelink [10].

Sin embargo, los aceleradores hardware suelen estar programados en lenguajes de descripción de hardware como vhdl y verilog, y dichos programadores no suelen estar familiarizados con lenguajes de programación como chisel.

Otra plataforma libre que permite la integración de aceleradores AXI es la plataforma PULP [11] [12]. Dicha plataforma también hace uso de la arquitectura RISC-V y de componentes Open-source, como su red de interconexión que integramos en nuestra plataforma. Sin embargo, dicha plataforma tiene procesadores de muy baja potencia y consumo centrados en procesamiento en el edge. Con caches únicamente de instrucciones y diseños muy centrados en la eficiencia energética.

VI. Conclusiones

Este trabajo describe las posibilidades de la plataforma de SELENE para la exploración del diseño de arquitecturas heterógeneas. SELENE es una plataforma open-source con procesadores industriales de tiempo real y con un alto soporte software capaz de sintetizarse en FPGA. Para facilitar la integración de aceleradores hardware como coprocesadores, la plataforma provee de soporte tanto hardware como software que permite la programación de los mismos tanto en baremetal como en Linux. A su vez el proceso de integración de los módulos en el SoC se puede realizar través de herramientas que automatizan la integración de dichos aceleradores en la plataforma.

La facilidad de integración de aceleradores de dicha plataforma se ejemplifica utilizando dos casos de uso de aceleración para tareas de seguridad e inteligencia artificial. Para cada caso de uso mostramos resultados de integración de *speedup* y *overhead* hardware.

Agradecimientos

El presente trabajo ha sido parcialmente financiado mediante el proyecto IPMAN (Técnicas Innovadoras para Infraestructuras, Aplicaciones y Servicios en Centros de Datos y Sistemas Altamente Distribuidos) con referencia PID2021-123627OB-C51 y por la Generalitat Valenciana mediante el proyecto Subvenciones para la contratación de personal investigador predoctoral con código CIACIF/2021/412.

Referencias

[1] John Shalf, "The future of computing beyond moore's law," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, pp. 20190061, 2020.

[2] Jason Cong, Jason Lau, Gai Liu, Stephen Neuendorffer, Peichen Pan, Kees Vissers, and Zhiru Zhang, "Fpga hls today: Successes, challenges, and opportunities," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 4, aug 2022.

[3] Laura Medina, Salva Carrión, Pablo Andreu, Tomas Picornell, Jose Flich, Carles Hernàndez, Michael Sandoval, Markel Sainz, Charles-Alexis Lefebvre, Martin Rönnbäck, Martin Matschnig, Matthias Wess, and Herbert Taucher, "The selene deep learning acceleration framework for safety-related applications," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 636–639.

[4] Andreas Kurth, Wolfgang Rönninger, Thomas Benz, Matheus Cavalcante, Fabian Schuiki, Florian Zaruba, and Luca Benini, "An open-source platform for high-performance non-coherent on-chip communication," *IEEE Transactions on Computers*, vol. 71, no. 8, pp. 1794–1809, 2022.

[5] José Flich, Laura Medina, Izan Catalán, Carles Hernández, Andrea Bragagnolo, Fabrice Auzanneau, and David Briand, "Efficient inference of image-based neural network models in reconfigurable systems with pruning and quantization," in *2022 IEEE International Conference on Image Processing (ICIP)*, 2022, pp. 2491–2495.

[6] Vatistas Kostalabros, Jordi Ribes-González, Oriol Farràs, Miquel Moretó, and Carles Hernandez, "Hls-based hw/sw co-design of the post-quantum classic mceliece cryptosystem," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 52–59.

[7] Lana Josipović, Andrea Guerrieri, and Paolo Ienne, "From c/c++ code to high-performance dataflow circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 7, pp. 2142–2155, 2022.

[8] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste Asanović, and Borivoje Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[9] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović, "Chisel: Constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*, 2012, pp. 1212–1221.

[10] Henry Cook SiFive, "Diplomatic design patterns : A tilelink case study," 2017.

[11] Antonio Pullini, Davide Rossi, Igor Loi, Giuseppe Tagliavini, and Luca Benini, "Mr.wolf: An energy-precision scalable parallel ultra low power soc for iot edge processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019.

[12] Michael Gautschi, Pasquale Davide Schiavone, Andreas Traber, Igor Loi, Antonio Pullini, Davide Rossi, Eric Flamand, Frank K. Gürkaynak, and Luca Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

Mejorando la tolerancia a fallos en circuitos cuánticos comparadores

L.M Donaire[1], Gloria Ortega [1], Ester M. Garzón[1] y Francisco Orts[2]

Resumen— **Los comparadores son uno de los circuitos con más interés para la comunidad científica, ya que forman parte fundamental de numerosos algoritmos. En este trabajo, proponemos tres versiones de circuitos comparadores presentes en la literatura que utilizan puertas T, unas puertas realmente interesantes en el diseño de circuitos para poder implementar códigos de detección de errores pero con un coste demasiado elevado, es por esto, que se han optimizado las puertas que utilizan estos circuitos ,y por lo tanto los circuitos, para reducir el número de puertas T que utilizan y de esta forma mejorar la tolerancia a fallos de los mismos.**

Palabras clave— **Comparadores, comparadores cuánticos, computación cuántica, tolerancia, T-count.**

I. Introducción

En los últimos años, la computación cuántica ha sido considerada como una de las tecnologías más prometedoras para superar las limitaciones físicas de los computadores actuales, en lo que se conoce como la era Post-Moore. Aunque aún no se ha determinado definitivamente qué tipos de problemas pueden ser resueltos de manera más eficiente mediante la computación cuántica, cada vez se presentan más ejemplos que demuestran las ventajas de la computación cuántica sobre la clásica[1], [2].

La computación cuántica busca aprovechar los principios de la mecánica cuántica para obtener beneficios en el ámbito computacional. Mediante el uso de la superposición, el entrelazamiento y la interferencia cuántica, este enfoque computacional tiene el potencial de resolver problemas complejos en un tiempo significativamente más rápido que los computadores tradicionales. Por ejemplo, un problema que llevaría billones de años en ser resuelto por un ordenador clásico podría ser resuelto en cuestión de horas utilizando la computación cuántica. Esto abre la puerta a avances significativos en diversos campos como los sistemas ambientales, la salud, la energía y la seguridad[3].

Sin embargo, el paradigma de programación de los computadores cuánticos es diferente al enfoque tradicional y exige la modificación de los algoritmos convencionales para adaptarse a las contraintuitivas reglas de la mecánica cuántica. El modelo principal utilizado en la computación cuántica es el de los circuitos cuánticos, donde las puertas lógicas tradicionales son reemplazadas por puertas cuánticas. Estas puertas cuánticas deben seguir las leyes de la física cuántica y, por lo tanto, son siempre reversibles.

[1]Dpto. de Informática, ceiA3, Universidad de Almería
[2]Instituto de Ciencia de Datos y Tecnologías Digitales, Universidad de Vilnius

En el diseño de circuitos y algoritmos cuánticos, uno de los principales desafíos actuales es la escasez de recursos, especialmente en términos del número de qubits disponibles en las plataformas cuánticas. Los qubits son la unidad fundamental de información en la computación cuántica[4], y mientras que los bits clásicos solo pueden tener valores de 0 o 1, los qubits pueden estar en un estado $|0\rangle$ o $|1\rangle$, conocidos como ket cero y ket uno, respectivamente, o en un estado de superposición que abarca todos los posibles estados. Sin embargo, estos estados de superposición se colapsan en un único valor cuando son observados o medidos. Es entonces crucial optimizar el uso de los recursos disponibles y reducir la cantidad de puertas cuánticas necesarias.

El ruido es otro desafío importante en el diseño de circuitos cuánticos[5]. El ruido introduce errores en las operaciones y reducir la cantidad de operaciones puede disminuir la exposición al ruido y, por lo tanto, reducir los errores. Para abordar el ruido, es necesario diseñar circuitos de manera compacta y utilizar códigos de detección y corrección de errores. Actualmente, nos encontramos en la era de la Computación Cuántica en Escala Intermedia con Ruido (NISQ, por sus siglas en inglés), donde los ordenadores cuánticos tienen un número moderado de qubits pero aún no son lo suficientemente grandes como para resolver problemas relevantes sin ser afectados por el ruido. Después de esta era NISQ, se espera que tengamos acceso a un mayor número de recursos cuánticos, lo que pondrá el énfasis en lograr circuitos rápidos. Por lo tanto, es fundamental construir circuitos altamente optimizados.

Los circuitos utilizados para realizar operaciones aritméticas desempeñan un papel fundamental en muchos algoritmos cuánticos que logran una aceleración significativa en comparación con los métodos clásicos más conocidos[6], [7], [8]. Sin embargo, el diseño de la parte aritmética requiere un enfoque perspicaz para minimizar el número de puertas utilizadas y reducir el error operativo, especialmente dado que las operaciones deben ser reversibles, lo que hace que las implementaciones reales sean complicadas[8].

En la actualidad, los circuitos pequeños para la implementación de operaciones aritméticas son de gran interés, ya que pueden ser invocados desde circuitos mucho más complejos, teniendo en cuenta la limitación de recursos en las plataformas cuánticas. Es importante destacar que, aunque estos circuitos no ofrezcan una ventaja en términos de velocidad debido al uso de la computación cuántica, siguen siendo valiosos para la comunidad científica[9].

Retomando el tema del ruido, este representa un

desafío físico que puede abordarse en parte mediante el diseño cuidadoso de circuitos. Como se mencionó anteriormente, se busca reducir tanto la duración del circuito como el número de operaciones. Además, se pueden emplear códigos de detección y corrección de errores para mitigar los efectos causados por el ruido. Por lo tanto, es importante diseñar circuitos de la manera más compacta posible al tiempo que se garantiza la tolerancia a fallos.

En cuanto a la tolerancia a fallos, el conjunto de puertas conocido como grupo Clifford+T permite que los circuitos compuestos exclusivamente por estas puertas implementen códigos de corrección de errores. Dentro de este grupo se encuentra la puerta T, también conocida como la puerta de fase pi/8. Esta puerta es ampliamente utilizada en computación cuántica debido a su capacidad para implementar rutinas de corrección de errores. Sin embargo, la puerta T tiene un costo computacional más elevado en comparación con otras puertas cuánticas básicas, como las puertas de Hadamard o las puertas de Pauli [10], [11], [12]. Esto se debe a que la puerta T requiere operaciones más complejas y un mayor número de operaciones elementales para ser implementada. Por lo tanto, su utilización en los circuitos cuánticos puede tener un impacto significativo en el costo total del circuito.

Dado el impacto del costo de la puerta T, es crucial optimizar su uso en los circuitos comparadores cuánticos. El objetivo principal es minimizar la cantidad de puertas T necesarias para reducir el ruido y mejorar la eficiencia del circuito sin aumentar significativamente el costo total.

Con respecto a la operación aritmética que se va a estudiar, la comunidad científica ha mostrado un gran interés en los comparadores cuánticos debido a su importancia en numerosos algoritmos y aplicaciones. Por ejemplo, en el procesamiento cuántico de imágenes, los comparadores juegan un papel fundamental en la extracción de características y la clasificación de imágenes cuánticas[13]. En el aprendizaje automático cuántico, los comparadores son utilizados para realizar operaciones de comparación y toma de decisiones en algoritmos de clasificación y reconocimiento de patrones[14].

Por lo tanto, en este trabajo nos hemos basado en la implementación de tres circuitos comparadores presentes en la literatura para proponer nuevos circuitos comparadores tolerantes a fallos. Esta optimización se ha realizado mediante el desarrollo de técnicas innovadoras destinadas a reducir la cantidad de puertas T necesarias. Como resultado de estos esfuerzos, los nuevos comparadores logran reducir la cantidad de puertas T en más de la mitad, lo que representa una mejora significativa en términos de eficiencia y costo computacional.

II. Puertas Cuánticas y Métricas

Las puertas cuánticas son diseñadas específicamente para su uso en computadoras cuánticas [15]. Algunas de estas puertas cuánticas realizan las mismas operaciones que las puertas clásicas, pero también existen puertas cuánticas que no tienen un equivalente en los circuitos clásicos. Estas puertas se pueden entender como matrices reversibles que operan en uno o varios qubits, transformando su valor inicial en otros valores. Una característica peculiar de estas puertas cuánticas es que son reversibles, lo que significa que nunca pierden información. Anteriormente se ha comentado que el grupo Clifford+T permite que los circuitos que están formados completamente por las puertas que forma este grupo implementen códigos de detección de errores. Sin embargo, hay que controlar el uso de la puerta T, de forma que, en este trabajo se trata de optimizar varios circuitos para reducir su cantidad de puertas T.

A continuación, se van a describir las puertas que se verán en este trabajo.

- La puerta Hadamard o puerta H se utiliza para poner el qubit en superposición. [16]
- La puerta CNOT realiza una operación con dos qubits, referidos como qubit de control y qubit objetivo. Si el qubit de control toma el 0 el qubit objetivo no se altera [17]
- La puerta Pauli-X equivale a la puerta NOT clásica, permutando el estado base de un qubit.
- La puerta T induce una fase pi/4.
- La puerta S equivale a dos puertas T consecutivas.
- La puerta Pauli-Z mantiene el estado inicial $|0\rangle$ sin cambios y asigna $|1\rangle$ a -$|1\rangle$.
- La puerta Peres dados tres qubits A, B y C, devuelve $A, B \oplus A$ y $C \oplus AB$, la implementación propuesta cuenta con 1 puerta CNOT, 1 puertas Controlled-V y 2 puertas Controlled-V$^+$ [18].
- La puerta TR presentada en [19] dados tres qubits A, B y C devuelve $A, A \oplus B$ y $A\bar{B} \oplus C$. La versión optimizada se propuso en [20] y se puede observar en la Figura 1.

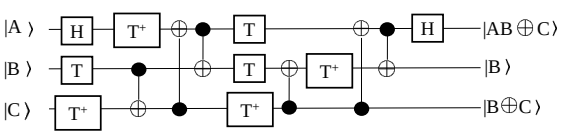


Fig. 1: Versión de la puerta TR.

- La puerta Temporary logical-AND se trata de una alternativa a la puerta Toffoli presentada en [21], centrada en reducir el coste de puertas T. Realiza la operación AND de dos qubits y lo almacenada en un qubit auxiliar que necesita ser inicializado a un valor específico. En la Figura 2 se puede observar su implementación.

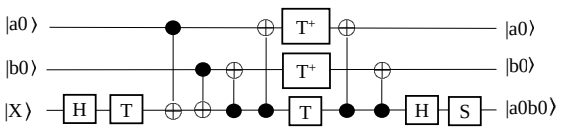


Fig. 2: Puerta Temporary logical-AND.

- Las puertas Controlled-V y Contolled-V$^+$ [15] mantienen las mismas propiedades que las puer-

tas V y V$^+$, pero pueden ser activadas o desactivadas.

Para medir el rendimiento de los circuitos, se van a seguir las siguientes métricas consolidadas:

- Coste cuántico: definido por el número de puertas que componen un circuito o una puerta.
- Retardo: definido por el número de puertas que deben calcularse secuencialmente. Si hay dos puertas o más puertas que pueden ser calculadas en paralelo, el retardo estará determinado por el retardo de la puerta más lenta.
- Entradas auxiliares: entradas que tienen establecido un valor constante, usadas para realizar operaciones auxiliares.
- T-count: número total de puertas T usadas en un circuito cuántico.
- T-depth: número de capas de la puerta T en el circuito, donde una capa consiste en las operaciones cuánticas que pueden realizarse simultáneamente.

Las métricas de las puertas mencionadas anteriormente se encuentran en la Tabla I

Tabla I: Métricas de las puertas descritas.

Puerta	Coste	Retardo	T-count	T-depth
Pauli-X	1	1	0	0
Pauli-Z	1	1	0	0
CNOT	1	1	0	0
T	1	1	1	1
H	1	1	0	0
C-V y C-V$^+$	1	1	3	2
S	1	1	0	0
TR	15	10	7	4
Peres	4	4	9	6
Temporary	11	9	4	2

III. Propuestas

Basándonos en tres circuitos comparadores presentes en la literatura se han propuesto tres nuevos circuitos comparadores que tienen el objetivo de reducir el número de puertas T que utilizan y, por consiguiente, mejorar la tolerancia a fallos del circuito.

El primer circuito en el que nos hemos basado es el propuesto en [22]. Este medio comparador de 2-bit esta formado por 2 R-Bcomp, dos puertas CNOT y dos puertas TR. Un R-Bcomp está formado por dos puertas TR y una puerta CNOT, por lo que poseería un coste cuántico de 31, un retardo de 21, un T-count de 14, un T-depth de 8 y dos entradas auxiliares (ver Figura 3)).

Sin embargo, en [22] siguen una implementación de la puerta TR diferente a la vista en la Sección II, y es esta la que se va optimizar. Esta versión esta formada por 2 puertas Controlled-V, una puerta Controlled-V+ y una puerta CNOT, de forma que el coste en términos de puertas T es de 9 y la profundidad de T es de 6, mientras que el coste cuántico y el retardo es de 4. Siguiendo esta implementación y las métricas para las puertas restantes presentes en la Tabla I, las métricas para el módulo R-Bcomp serían las presentas en la Tabla II.

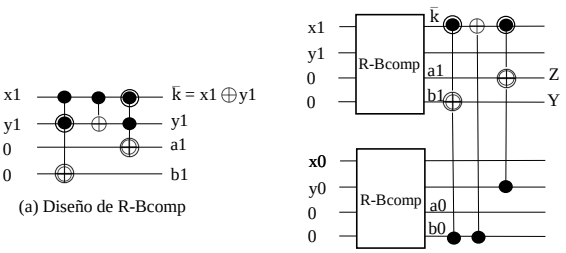


Fig. 3: Medio comparador de 2-bit, etiquetado como *Circuito 1*, a la izquierda se puede observar el diseño de cada módulo R-Bcomp y a la derecha el circuito reversible de 2-bit.

Tabla II: Métricas módulo R-Bcomp

Coste	Retardo	T-count	T-depth	Aux
9	9	18	12	2

Ahora bien, teniendo en cuenta las métricas de cada módulo R-Bcomp, las métricas para este circuito se pueden observar en la Tabla III.

Tabla III: Métricas Circuito 1

Coste	Retardo	T-count	T-depth	Aux
27	18	54	36	4

Observando el circuito se puede ver que el alto número de puertas T proviene de la puerta TR, por lo que, se propone una nueva implementación de esta puerta que reduce el número de puertas T. Esta nueva implementación se basa en el uso de una puerta temporary logical-AND, dos puertas Pauli-X, y dos puertas CNOT (ver Figura 4).

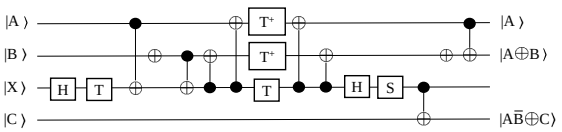


Fig. 4: Nueva implementación del la puerta TR

En esta versión se puede observar que:

- La primer puerta Pauli-X niega el qubit B antes de que la puerta temporary logical-AND realice la multiplicación de los qubits.
- A continuación, la puerta temporary logical-AND realiza la operación de multiplicación A$\overline{\text{B}}$ y guarda el resultado en el qubit auxiliar.
- Una puerta CNOT actuará sobre la información del qubit auxiliar, es decir, sobre A$\overline{\text{B}}$ realizando una operación $\oplus$ junto con el qubit C obteniendo A$\overline{\text{B}}\oplus$C
- La segunda puerta Pauli-X devuelve el qubit B a su estado no negado para que la puerta CNOT pueda realizar la operación A$\oplus$B.

Las métricas de esta nueva implementación de la puerta TR se pueden observar en la Tabla IV.

A continuación, basándonos en el comparador propuesto por Thapliyal en su artículo[22] se propone un nuevo circuito comparador que se puede observar en la Figura 5) donde se pueden apreciar los cambios

Tabla IV: Métricas de la versión propuesta de la puerta TR, donde A, B y C serán los qubits sobre los que se realiza la operación y X es un qubit auxiliar.

Coste	Retardo	T-count	T-depth
17	13	4	2

realizados en las puertas TR del módulo R-Bcomp y del circuito comparador, que han sido reemplazadas por la nueva versión propuesta.

Para calcular las métricas de este circuito, se ha considerado que dos puertas iguales consecutivas se anulan entre sí (se puede observar en la Figura 5). Por lo tanto, al reemplazar las puertas TR con la implementación propuesta, varias puertas Pauli-X y CNOT se han anulado. Después de reducir el circuito, se encuentra:

- 6 puertas TR, en las al sustituirlas en el circuito original se han eliminado las puertas secuenciales duplicadas.
- 4 puertas H que inicializan los qubits que forman los números a comparar.
- 1 puerta CNOT

Observando la Figura 5), se determina que el circuito está compuesto por un total de 101 puertas, cada una con un coste cuántico de 1 (consultar la Tabla I). Por lo tanto, el coste cuántico total es de 101. Al observar las puertas que se realizan en paralelo y considerar la mayor demora entre ellas, se obtiene un retardo de 44.

En cuanto al T-count y T-depth, el circuito contiene 6 puertas TR, lo que significa que el número total de puertas T es de $6 * 4 = 24$. Por último, para calcular el T-depth, se observa el número de puertas T que se realizan en paralelo. Se observa que 2 puertas TR se ejecutan en paralelo 2 veces, lo que resulta en un T-depth de $4(puertasTRsecuenciales) * 2 = 8$.

Las métricas se pueden observar en la Tabla V. Al comparar estos resultados con la Tabla III, se puede observar que tanto el T-count como el T-depth se reducen aproximadamente a la mitad.

Tabla V: Métricas del Circuito 1p

Coste	Retardo	T-count	T-depth	Aux
101	44	24	8	10

El siguiente comparador que se ha estudiado fue propuesto por Maity et al. en su artículo [23]. Este comparador completo de dos qubits utiliza numerosas puertas Peres. La implementación de la puerta Peres que utilizan consta de 3 puertas Controlled-V, una puerta Controlled-V^+ y una puerta CNOT. En consecuencia, el T-count y el T-depth para cada puerta Peres es de 9 y 6, respectivamente. Puede observarse este comparador en la Figura 6.

De forma que las métricas para este comparador, se pueden observar en la Tabla VI.

Con el fin de optimizar este circuito se propone una nueva versión de la puerta Peres, que hace uso de la puerta temporary logical-AND y una puerta CNOT (ver Figura 7).

Tabla VI: Métricas del Circuito 2

Coste	Retardo	T-count	T-depth	Aux
25	14	36	16	4

En esta versión se puede observar que:

- La puerta temporary logical-AND realiza la operación de multiplicación AB y guarda el resultado en el qubit auxiliar.
- Una puerta CNOT actuará sobre la información del qubit auxiliar, es decir, sobre AB realizando una operación $\oplus$ junto con el qubit C obteniendo AB$\oplus$C
- La segunda puerta CNOT realiza la operación A$\oplus$B.

Teniendo en cuenta las puertas por la que esta formada esta nueva implementación, las métricas de esta puerta se pueden observar en la Tabla VII.

Tabla VII: Métricas de la versión propuesta de la puerta Peres

Coste	Retardo	T-count	T-depth
15	12	4	2

Siguiendo esta versión, el circuito propuesto optimizado se puede observar en la Figura 8.

Se puede observar que el circuito está compuesto por un total de 69 puertas con un coste cuántico de 1, por lo que el coste total es de 69. Con respecto al retardo se puede observar que 2 puertas Peres se realizan de forma paralela, además, la inicialización necesaria para el qubit auxiliar solamente se realiza al inicio, por lo que la segunda y tercera puerta tienen reducido el retardo a 10 y 11, respectivamente. El aumento del retardo en la última puerta Peres se debe a que una de las puertas CNOT que se realizaban de forma paralela, se realiza de forma secuencial. Dado esto, el retardo final es de $12 + 10 + 11 + 1 = 34$. Con respecto al T-count y al T-depth, se tiene que cada puerta Peres tiene un T-count de 4, por lo que, el número de puertas T en este circuito es de $4 * 4 = 16$, mientras que el T-depth es de $4 * 2 = 8$. Las métricas del comparador propuesto se pueden observar de forma resumida en la Tabla VIII.

Tabla VIII: Métricas de la versión propuesta del comparador

Coste	Retardo	T-count	T-depth	Aux
69	35	16	8	8

Comparando estos resultados con la Tabla VI se puede observar, de nuevo, el T-count y T-depth se reducen más de la mitad.

El último comparador en el que nos basamos es el propuesto por Kalita et al. en el artículo [24] (ver Figura 9), específicamente en su versión de 1 bit. En este comparador completo, utilizan una puerta propia denominada GN que está formada por 2 puertas Controlled-V, 1 puerta Controlled-V^+ y una puerta CNOT. Las métricas de esta puerta pueden observarse en la Tabla IX.

Por lo tanto, las métricas de este comparador se pueden observar en la Tabla X.

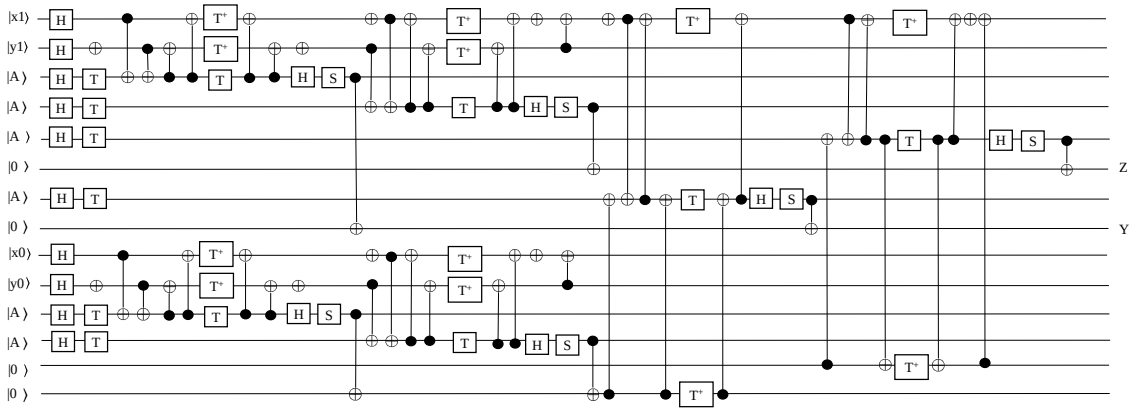


Fig. 5: Circuito propuesto optimizado, etiquetado como *Circuito 1p* donde x1 y x0 representan un número de dos bits, al igual que y1 e y0, mientras que los qubits marcados como A representa qubits auxiliares que tienen un estado inicial específico para la puerta temporary logical-AND.

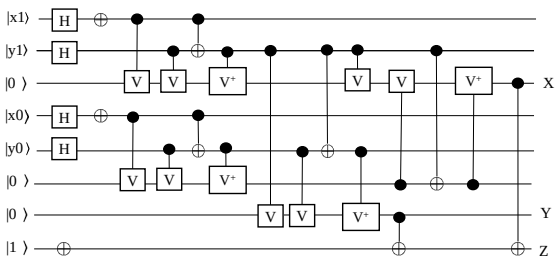


Fig. 6: Comparador completo de 2-bit, etiquetado como*Circuito 2*

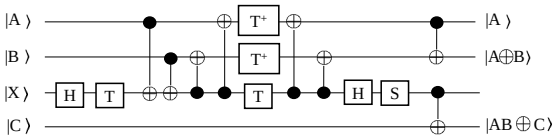


Fig. 7: Versión propuesta de la puerta Peres

Tabla IX: Métricas puerta GN.

Coste	Retardo	T-count	T-depth
4	4	9	6

Tabla X: Métricas Circuito 3

Coste	Retardo	T-count	T-depth	Aux
15	11	4	2	2

Se propone una versión mejorada de su puerta GN que reduce el T-count y el T-depth. Esta versión utilizada una puerta Temporary, dos puertas CNOT y dos puertas Pauli-X. Esta propuesta puede observarse en la Figura 10.

En esta versión se puede observar que:

- La primer puerta Pauli-X niega el qubit A antes de que la puerta temporary logical-AND realice la multiplicación de los qubits.
- A continuación, la puerta temporary logical-AND realiza la operación de multiplicación $\overline{A}C$ y guarda el resultado en el qubit auxiliar.
- Una puerta CNOT actuará sobre la información del qubit auxiliar, es decir, sobre $\overline{A}C$ realizando una operación $\oplus$ junto con el qubit B obteniendo $\overline{A}C \oplus B$.
- La segunda puerta Pauli-X devuelve el qubit A a su estado no negado para que la puerta CNOT pueda realizar la operación $A \oplus C$.

Las métricas de esta versión de la puerta GN se pueden ver en la Tabla XI.

Tabla XI: Métricas de la versión propuesta de la puerta GN

Coste	Retardo	T-count	T-depth
15	11	4	2

Siguiendo esta implementación, se propone el tercer y último circuito que se puede observar en la Figura 11.

Con respecto al cálculo de las métricas de este circuito, se puede observar que está compuesto por un total de 22 puertas con un coste cuántico de 1, por lo que el coste total es de 22. Con respecto al retardo se puede observar que la versión propuesta de la puerta GN se realiza de forma secuencial con dos puertas CNOT y una puerta Pauli-X. La puerta Pauli-X se realiza de forma paralela que una puerta CNOT, por lo que el retardo es de 1. Por lo tanto, el retardo final es de 14. Con respecto al T-count y al T-depth, la única puerta que presenta puertas T es la puerta GN propuesto, de forma que, el T-count total es de 4, mientras que el T-depth es de 2. que cada puerta Peres tiene un T-count de 4, por lo que, el número de puertas T en este circuito es de $4 * 4 = 16$, mientras que el T-depth es de $4 * 2 = 8$. Las métricas de esta versión del comparador se pueden ver en la Tabla XII. Al comparar estos resultados con la Tabla X, se puede observar que tanto el T-count como el T-depth se reducen a más de la mitad.

Tabla XII: Métricas del comparador 3p

Coste	Retardo	T-count	T-depth	Aux
22	15	4	2	3

IV. Resultados

La Tabla XIII muestra las métricas correspondientes a los diferentes circuitos estudiados. Estas métricas incluyen el coste cuántico, el retardo, el T-count,

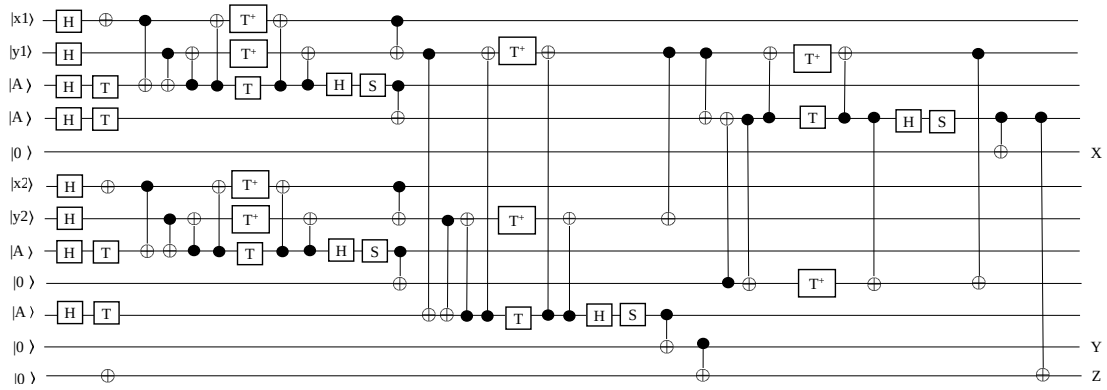


Fig. 8: Circuito propuesto optimizado, etiquetado como *Circuito 2p* donde x1 e x2 representan un número de dos bits, al igual que y1 e y2, mientras que los qubits marcados como A representa qubits auxiliares que tienen un estado inicial específico para la puerta temporary logical-AND.

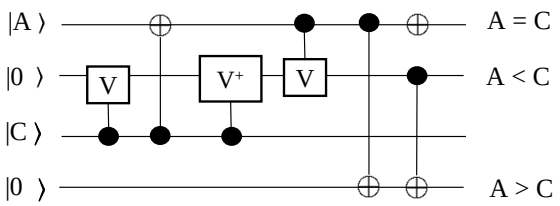


Fig. 9: Comparador completo de 1-bit, etiquetado como *Circuito 3* donde A y C son los qubits que se van a comparar.

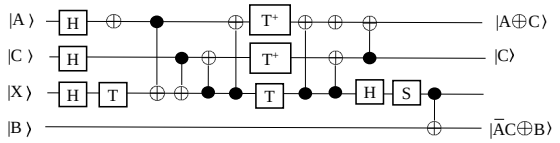


Fig. 10: Versión propuesta de la puerta GN donde A, C y B son los qubits sobre los que se realizan las operaciones y X es el qubit auxiliar inicializado con un estado especial para la puerta temporary logical-AND.

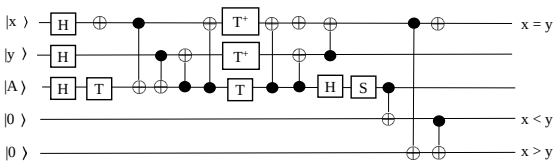


Fig. 11: Circuito optimizado, etiquetado como *Circuito 3p*, donde x e y son los valores que se van a comparar y A representa un qubit auxiliar inicializado con un estado especial necesario para la puerta temporary logical-AND

el T-depth y el número de qubits auxiliares utilizados en cada circuito.

Tabla XIII: Métricas de los circuitos estudiados y sus optimizaciones

Circuito	Coste	Retardo	T-count	T-depth	Aux
Circuito 1	27	18	54	36	4
Circuito 1p	101	44	24	8	10
Circuito 2	25	14	36	16	4
Circuito 2p	69	35	16	6	8
Circuito 3	9	7	9	6	2
Circuito 3p	22	15	4	2	3

Se puede observar que los circuitos optimizados (etiquetados con "p.ª final) presentan una reducción significativa en el T-count y el T-depth en comparación con sus versiones originales. Esto demuestra que las optimizaciones propuestas para las puertas TR, Peres y GN han tenido un impacto positivo en el rendimiento de los circuitos, mejorando su tolerancia a fallos. Aunque el aumento en el coste cuántico, el retardo y el número de qubits auxiliares puede parecer una desventaja, la reducción del T-count y el T-depth tiene un impacto más significativo en la capacidad de un circuito para realizar cálculos cuánticos precisos y fiables. Por lo tanto, el beneficio de mejorar la tolerancia a fallos y reducir el impacto del ruido supera los costos adicionales asociados con estas optimizaciones.

V. Conclusiones

En este estudio, nos hemos basado en tres comparadores cuánticos presentes en la literatura para presentar tres nuevos comparadores cuánticos tolerantes a fallos. Para lograr este cometido se han propuesto nuevas versiones para las puertas TR, Peres y GN, puertas con un alto coste en términos de T-count y T-depth, utilizando puertas presentes en el grupo Clifford+T.

Al utilizar estas nuevas implementaciones de puertas, se logra mejorar la tolerancia a fallos del circuito, lo cual es un paso importante hacia la construcción de circuitos más robustos y confiables.

En resumen, la optimización de las puertas TR, Peres y GN mediante la utilización de puertas del grupo Clifford+T ha permitido mejorar la tolerancia a fallos de los circuitos estudiados. Estos resultados son prometedores y abren el camino para futuras investigaciones en la mejora de la calidad y fiabilidad de los sistemas cuánticos.

Agradecimientos

Esta publicación es parte del la proyecto de I+D+i PID2021-123278OB-I00, TED2021-132020B-I00 financiado por MCIN/ AEI/10.13039/501100011033/ y por "FEDER Una manera de hacer Europa".

Referencias

[1] M. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[2] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al.,

"Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.
[3] Microsoft Azure, "What is quantum computing?," https://azure.microsoft.com/en-us/overview/what-is-quantum-computing/.
[4] Microsoft Azure, "What is a qubit?," https://azure.microsoft.com/en-us/overview/what-is-a-qubit/.
[5] K.-W. Cheng and C.-C. Tseng, "Quantum plain and carry look-ahead adders," *arXiv:quant-ph/0206028*, 2002.
[6] F. Orts, G. Ortega, E. Fernández-Combarro and E.M. Garzón, "A review on reversible quantum adders," *Journal of Network and Computer Applications*, 2020.
[7] H. Thapliyal, *Mapping of Subtractor and Adder-Subtractor Circuits on Reversible Quantum Gates*, vol. 9570, Springer, 2016.
[8] Michael Kirkedal Thomsen, Robert Glück, and Holger Bock Axelsen, "Reversible arithmetic logic unit for quantum arithmetic," *Journal of Physics A: Mathematical and Theoretical*, vol. 43, no. 38, pp. 382002, 2010.
[9] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster and J.I. Latorre, "Data re-uploading for a universal quantum classifier.," *Quantum-Journal*, vol. 4, pp. 226, 2020.
[10] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, , no. 6, pp. 818–830, 2013.
[11] Miller, D. Michael and Soeken, Mathias and Drechsler, Rolf, "Mapping ncv circuits to optimized clifford+t circuits," *International Conference on Reversible Computation*, pp. 163–175, 2014.
[12] Matthew Amy and Dmitri Maslov and Michele Mosca, "Polynomial-time t-depth optimization of clifford+t circuits via matroid partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, pp. 1476–1489, 2014.
[13] Jian Wang, Nan Jiang, and Luo Wang, "Quantum image translation," *Quantum Information Processing*, vol. 14, no. 5, pp. 1589–1604, 2015.
[14] Zufar Kayumov, Dmitrii Tumakov, and Sergey Mosin, "An effect of binarization on handwritten digits recognition by hierarchical neural networks," in *International Conference on Image Processing and Capsule Networks*. Springer, 2021, pp. 94–106.
[15] F. Orts, G. Ortega and E. Garzón, "A faster half subtractor circuit using reversible quantum gates," *Baltic J. Modern Computing*, vol. 7 (1), pp. 99–111, 2019.
[16] Robert S. Sutor, *Dancing with Qubits: How quantum computing works and how it can change the world*, Packt Publishing Ltd, 2019.
[17] D. Deutsch and P. Hayden, "Information flow in entangled quantum systems," *Royal Society*, vol. 456, pp. 1759–1774, 2000.
[18] F. Orts, G. Ortega, E.F. Combarro and E.M. Garzón, "Implementation of reversible peres gate using electro-optic effect inside lithium-niobate based mach-zehnder interferometers," *Optics and Laser Technology*, vol. 117, pp. 28–37, 2019.
[19] H. Thapliyal and N. Ranganathan, "Design of efficient reversible logic-based binary and bcd adder circuits," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 9 (3), pp. 1–31, 2013.
[20] Hai-Sheng Li, Ping Fan, Haiying Xia, Huiling Peng, and Gui Long, "Efficient quantum arithmetic operation circuits for quantum image processing," *Sci. China Phys. Mech. Astron.*, vol. 63, pp. 1–13, 2020.
[21] Craig Gidney, "Halving the cost of quantum addition," *Quantum*, vol. 2, pp. 74, 2018.
[22] H. Thapliyal, N. Ranganathan and R. Ferreira, "Design of a comparator tree based on reversible logic," *2010 10th IEEE Conference on Nanotechnology*, pp. 1113–1116, 2010.
[23] H. Maity, "Design and implementation of a two-qubit quantum comparator circuit (q-cc)," *Journal of Computational Electronics*, pp. 1572–8137, 2022.
[24] G. Kalita and N. Saikia, "Reversible comparator circuit using a new reversible gate," *Proceedings of the Sixth International Conference on Computer and Communication Technology 2015*, 2015.
[25] M. Mohammadi and M. Eshghi, "On figures of merit in reversible and quantum logic designs," *Quantum Information Processing*, vol. 8, pp. 297–318, 2009.

Arquitecturas heterogéneas y modelos de programación para GPU, FPGA y aceleradores de IA

Estudio del rendimiento en entrenamientos distribuidos para simulaciones CFD data-driven

A. González[1], P. Barreda[1], K. Rojek[3], and S. Iserte[1,2]

[1]Universitat Jaume I, Castelló, España.
[2]Barcelona Supercomputing Center, Barcelona, España.
[3]Czestochowa University of Technology, Poland.

Resumen— **Los métodos *data-driven* para simulaciones informáticas están creciendo en muchas áreas científicas. El enfoque tradicional para simular comportamientos físicos se basa en resolver ecuaciones diferenciales parciales (EDP). Dado que el cálculo de estas ecuaciones iterativas es muy exigente desde el punto de vista computacional y temporal, los métodos *data-driven* aprovechan las técnicas de Inteligencia Artificial (IA) para aliviar esa carga de trabajo. Estos tienen que ser entrenados de antemano para proporcionar las rápidas predicciones. Sin embargo, el coste de la fase de entrenamiento no debe ser despreciable.**

Este artículo presenta un modelo predictivo para inferir estados futuros de una simulación de fluidos específica que sirve como caso de uso para evaluar diferentes alternativas de entrenamiento. En concreto, compara el rendimiento de enfoques basados únicamente en CPU, multi GPU, y enfoques distribuidos para el entrenamiento de un modelo de *Deep Learning* (DL) de predicción de series temporales. Con algunas ligeras adaptaciones del código, los resultados muestran y comparan, en diferentes implementaciones, las ventajas del entrenamiento distribuido basado en la GPU para la predicción de series temporales.

Palabras clave— ***Deep Learning*, Simulaciones CFD, Computación en CPU/GPU, Entrenamiento distribuido**

I. Introducción

Los métodos de Inteligencia Artificial (IA) se han generalizado en los últimos años debido a los numerosos avances algorítmicos y a la accesibilidad de la potencia computacional Archibald et al. (2020) [1]. En *Computational Fluid Dynamics* (CFD), la IA se ha utilizado para ayudar, acelerar, mejorar o incluso incluso reemplazar los solucionadores físicos existentes basados en ecuaciones Brunton et al. (2020) [2]. Especialmente, durante los últimos años, los métodos de aprendizaje profundo se han utilizado ampliamente en el campo de la CFD.

En este trabajo, proponemos un modelo de IA específico del dominio que puede integrarse fácilmente con simulaciones CFD. El modelo de entrenamiento se evalúa exhaustivamente con técnicas distribuidas en dispositivos CPU y GPU. El estudio presentado a lo largo de este trabajo se basa en un tanque de homogeneización. Los tanques de homogeneización son esenciales en diversos campos industriales, especialmente en aquellos procesos productivos en los que se manejan grandes volúmenes de líquidos o mezclas acuosas.

Este tipo de tanques suelen ubicarse en las etapas previas a la mezcla industrial o a los tratamientos de deposición/fermentación de diferentes tipos de sustancias. Por ello, como la mayoría de industrias tienen procesos de operación continua trabajando 24 horas al día, así como, largos tiempos de espera entre cada fase, estas instalaciones permiten un control más detallado sobre el flujo de entrada a la siguiente fase del proceso, manteniendo un volumen de entrada regular para asegurar que el proceso posterior disponga del tiempo de retención hidráulica necesario para llevar a cabo su función.

De ahí que nuestra investigación se lleve a cabo aprovechando la geometría de un tanque de homogeneización a escala real utilizado en una gran variedad de instalaciones industriales. En la figura 1 se representa la geometría del tanque y la ubicación de sus áreas de interés, que no son simples paredes y pueden configurarse. Como se muestra, el tanque está compuesto por una cuba de $15m$ de longitud, $6m$ de anchura y $5m$ de profundidad. El flujo entra en el tanque a través de dos por dos puntos diferentes: *inflow #1* y la *inflow #2*, y sale del reactor por la *outflow* a través del flujo de salida. Dentro del tanque, también hay un agitador que es responsable de mejorar la mezcla dentro del flujo para que la mezcla líquida o acuosa no se deposite en su fondo y continúe su camino hacia la salida del tanque. En función de una serie de valores diferentes, como la velocidad del impulsor o los caudales de entrada al depósito, se pueden obtener velocidades inferiores o superiores en el interior de la instalación, lo que puede afectar al rendimiento de las operaciones siguientes. Por lo tanto, analizar una amplia variedad de casos de funcionamiento es útil para optimizar el rendimiento de la instalación.

Nótese que este caso de estudio se limita a evaluar un modelo predictivo, su rendimiento al entrenarlo y hacer inferencias con diferentes configuraciones de recursos. Por lo tanto, los detalles específicos sobre el tanque quedan fuera del alcance de este documento. No obstante, tanques similares han sido ampliamente estudiados en otros trabajos en los que se puede encontrar más información aplicable como en Iserte et al. (2021) [3].

Este trabajo investiga el rendimiento y las diferentes estrategias de aprendizaje del modelo de IA

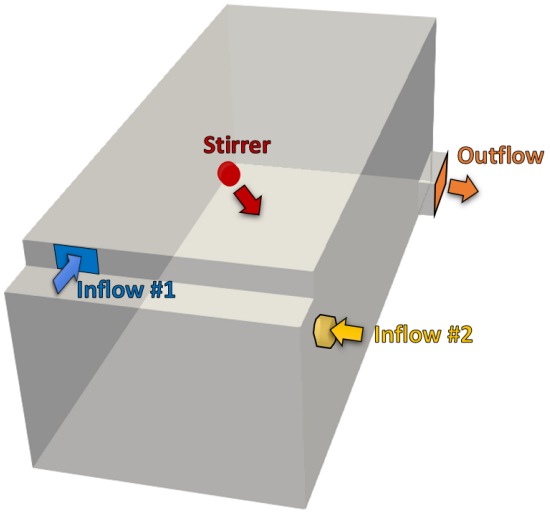


Fig. 1: Geometría del reactor bajo estudio. Las flechas representan las dirección del flujo en las diferentes áreas de interés.

centrado en un enfoque basado en datos. A continuación, proporcionamos una interacción entre la IA y el solver CFD para un análisis mucho más rápido y reducir el coste de los experimentos de prueba y error. El ámbito de esta investigación incluye simulaciones que utilizan un esquema iterativo para llegar a la convergencia. Los modelos de estado casi estacionario realizan un balance de masa y energía de un proceso en un estado de equilibrio independiente del tiempo Bhatt et al. (2010) [4]. En otras palabras se supone que un solucionador calcula un conjunto de pasos temporales para alcanzar el estado de convergencia del fenómeno simulado. De ahí que el método ideado se encargue de predecir el estado posterior con el modelo IA basado en los anteriores generados por el solver CFD.

Las aportaciones más relevantes de este trabajo son:

- un método basado en IA que puede integrarse con un solver CFD y permite predecir la evolución de simulaciones a partir de tiempos iniciales generados por el solver;
- una red neuronal recursiva (RNN) que predice futuros en una serie temporal de flujo;
- un análisis del rendimiento de diferentes métodos de métodos de entrenamiento distribuido considerando Horovod y Tensorflow `MultiworkerStrategy`; y
- una comparación de rendimiento entre clusters de CPU y GPU para entrenar y evaluar el modelo predictivo modelo predictivo ideado.

II. Trabajos relacionados

La adaptación de los códigos de CFD a las arquitecturas de hardware modernas y la aceleración de las simulaciones de CFD con IA son temas de investigación importantes debido a las crecientes necesidades de potencia informática. Según Vinuesa y Brunton (2021) [5], la IA en las simulaciones de CFD puede reducir el tiempo de solución y proporcionar una precisión aceptable en comparación con la optimización manual tradicional. Sin embargo, la escalabilidad del entrenamiento en diferentes configuraciones de hardware es un aspecto que a menudo se pasa por alto en los avances tecnológicos de las simulaciones de CFD *data-driven*.

Diversos estudios han abordado la aceleración de las simulaciones de CFD mediante el uso de modelos de IA. Por ejemplo, Rojek et al. (2021) [6] propusieron un método para acelerar la simulación de mezcla química utilizando un modelo de IA, obteniendo resultados precisos y una aceleración significativa. Sin embargo, estos resultados se lograron en un solo nodo de CPU/GPU. Otros estudios, como el de Xiao et al. (2019) [7], han desarrollado modelos de orden reducido no intrusivos (NIROM) que pueden predecir la evolución de flujos turbulentos en estados cuasiestacionarios. Estos modelos se entrenaron en máquinas con un número limitado de núcleos y datos reducidos.

Además, se han propuesto enfoques que combinan simulaciones de CFD y modelos de IA, como el trabajo presentado por Kim et al. (2019) [8], que utiliza un modelo generativo para sintetizar simulaciones de CFD a partir de parámetros reducidos. En otros casos, se han desarrollado modelos sustitutos en los que se predicen algunas etapas de los pasos de tiempo y otras se calculan. Estos enfoques muestran resultados con errores similares a las simulaciones tradicionales de CFD, pero en un tiempo mucho más reducido.

En términos de escalabilidad del entrenamiento, se han realizado investigaciones para evaluar el rendimiento en diferentes configuraciones de hardware. Por ejemplo, Ramirez-Gargallo et al. (2019) [9] utilizaron TensorFlow, la Math Kernel Library (MKL) y Horovod para acelerar el entrenamiento de redes neuronales en arquitecturas distribuidas. Awan et al. (2019) [10] evaluaron el entrenamiento distribuido en múltiples GPUs y mostraron un aumento de velocidad significativo. Estos estudios demuestran la importancia de considerar la escalabilidad del entrenamiento en el desarrollo de modelos de CFD basados en datos.

En resumen, la adaptación de los códigos de CFD a las arquitecturas de hardware modernas y la aceleración de las simulaciones de CFD con IA son áreas de investigación activas. Varios estudios han demostrado resultados prometedores en términos de precisión y aceleración, pero es necesario abordar la escalabilidad del entrenamiento en diferentes configuraciones de hardware para lograr avances significativos en este campo.

Este es el primer trabajo que evalúa ampliamente la escalabilidad del entrenamiento de un modelo predictivo de series temporales de CFD. Este ha evaluado un modelo de CFD *data-driven* que, a diferencia de los modelos de clasificación, se basa en unas pocas capas y grandes cantidades de datos, lo cual aumenta el tráfico de comunicación.

III. Materiales y métodos

La siguiente sección describe el modelo CFD que resuelve la hidrodinámica dentro del tanque, así co-

mo también, la generación y procesado de los datos. Las técnicas de distribución también son presentadas durante la sección.

A. Simulaciones CFD

Este punto describe la metodología utilizada en las simulaciones de dinámica de fluidos computacional (CFD) en este trabajo. Se utilizó el software OpenFOAM para generar una malla del tanque y resolver las ecuaciones de Navier-Stokes no estacionarias y promediadas de Reynolds (URANS) para simular la hidrodinámica en el tanque. Se utilizaron las herramientas de OpenFOAM `blockMesh` y `snappyHexMesh` para generar la malla y el solucionador de OpenFOAM `pimpleFoam` para la resolución numérica hidrodinámica dentro del dominio. En resumen, se ejecutaron simulaciones hasta el segundo $4,201$ del tiempo simulado, almacenando 420 pasos de tiempo por cada simulación ejecutada.

B. Modelo predictivo

La siguiente subsección describe la generación del conjunto de datos, el diseño del modelo predictivo, la evaluación y validación del mismo, y los métodos de distribución del entrenamiento que a su vez son evaluados.

B.1 Dataset

Se generó un conjunto de datos para un modelo *data-driven* mediante múltiples simulaciones CFD, limitando las variaciones a dos variables, *inflow #1* y *inflow #2*. El conjunto de datos tiene 131 casos de 420 pasos de tiempo, cada uno con información de 125,565 celdas y se centra en la métrica tridimensional de la velocidad. El tamaño total del conjunto de datos es de 38,6 GB. Antes de alimentar el modelo, se normaliza cada dimensión de la velocidad y se divide el conjunto de datos en subconjuntos de entrenamiento, prueba y validación cruzada.

B.2 Red neuronal

El objetivo de este estudio es desarrollar un modelo de red neuronal capaz de predecir series de tiempo a partir de una secuencia corta de intervalos de tiempo. Para ello, se ha diseñado una red neuronal recurrente (RNN) capaz de retener conocimiento temporal. Las RNN son una variación de las redes neuronales de alimentación directa, cuyas neuronas pueden conectarse en ciclos, lo que permite que las salidas de las neuronas se utilicen en su propia entrada, creando así un efecto de memoria.

La arquitectura de la red neuronal diseñada se puede apreciar en la figura 2, la cual consta de dos capas de memoria a corto plazo (LSTM), cada una de 200 unidades, activadas por una función no lineal ReLU. El modelo se compila con el optimizador Adam y una tasa de aprendizaje de 0,00025. La función de pérdida elegida es el MAE. Debido a las limitaciones de hardware, el tamaño de lote se establece en 14. El modelo se entrena utilizando un generador basado en la clase de secuencia de Keras que no solo organiza los lotes pequeños, sino que también reformatea los datos y el objetivo de acuerdo con las características de la red.

B.3 Evaluación

En el estudio se utilizaron varias métricas estadísticas para evaluar la precisión del modelo predictivo basado en inteligencia artificial en comparación con un solucionador de dinámica de fluidos computacional (CFD) tradicional.

Los resultados mostraron un alto grado de correlación entre los valores predichos por el modelo de inteligencia artificial y los valores calculados por el CFD, con un coeficiente de correlación de Pearson que varía de 0.98 a 1.0 y un coeficiente de correlación de Spearman que varía de 0.96 a 1.0. Además, el error cuadrático medio (RMSE) fue en promedio de 0.023 para todos los timesteps validados.

Además, la comparación de histogramas utilizando el coeficiente de determinación mostró una precisión promedio del 98 %, lo que indica una buena concordancia entre los valores calculados y los valores predichos.

En general, estas métricas estadísticas sugieren que el modelo de inteligencia artificial es altamente preciso y bien ajustado para predecir la magnitud de la velocidad en la región de interés.

Se puede encontrar el estudio más detallado de la evaluación del modelo predictivo en la publicación Iserte et al. [11]

IV. Aprendizaje distribuido

Otra contribución de este trabajo ha sido evaluar el desempeño del entrenamiento del modelo utilizando múltiples aceleradores distribuidos en varios nodos. Para este propósito, el estudio utiliza dos estrategias diferentes que brindan soporte para el entrenamiento distribuido. Mientras que la primera estrategia presentada es implícitamente soportada por Tensorflow, la segunda es un framework para varias bibliotecas de aprendizaje profundo, entre ellas, Tensorflow.

A. Tensorflow

Tensorflow, Abadi et al. (2016) [12], es una biblioteca de código abierto para inteligencia artificial. Está diseñada para el entrenamiento y la inferencia de redes neuronales profundas.

Tensorflow proporciona mecanismos nativos de entrenamiento síncrono en múltiples réplicas en una o más máquinas, especialmente:

- *MirroredStrategy*[1]*:* una variable creada con esta estrategia es una *MirroredVariable* replicada en todas las GPU involucradas o en las CPU si no se encuentran GPU. Todas las CPU de una máquina se tratan como un único dispositivo (como una GPU) y se genera un hilo por núcleo para la paralelización.

[1] `https://www.tensorflow.org/api_docs/python/tf/distribute/MirroredStrategy`

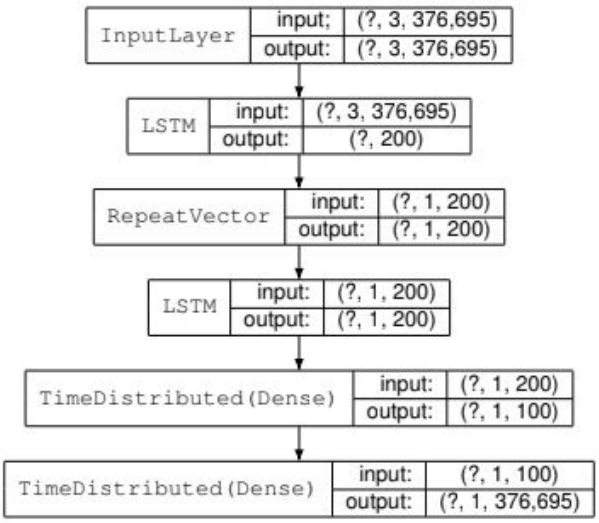


Fig. 2: Arquitectura RNN. Al lado del tipo de capa, se indica el shape del input y el output.

- *Multiworker*[2]*:* esta estrategia complementa *MirroredStrategy* con la capacidad de comunicar varios nodos (trabajadores), cada uno con sus propias GPU. Para este propósito, se replican todas las variables y cálculos en cada dispositivo local. Varios trabajadores pueden trabajar juntos utilizando una operación de all-reduce.

B. Horovod

Horovod, Sergeev and Balso (2018) [13], es un framework de aprendizaje distribuido implementado utilizando el modelo de Interfaz de Paso de Mensajes (MPI) (aunque también admite otras bibliotecas de comunicación como Gloo) en un esfuerzo por reducir la complejidad de configuración introducida por *MultiworkerStrategy* de Tensorflow para operar en clusters computacionales. Horovod brinda soporte para bibliotecas de IA como PyTorch, MXNet y TensorFlow. Una vez que se ha escrito el script de entrenamiento para escalar, puede ejecutarse en cualquier número de GPU y/o nodos sin necesidad de realizar cambios en el código.

Horovod se basa en la Biblioteca de Comunicación Colectiva NVIDIA[3] (NCCL) para implementar comunicaciones optimizadas de múltiples GPU y nodos. El núcleo de Horovod se basa en primitivas MPI como *rank* o *all-reduce*. En este sentido, la documentación de Horovod informa que al utilizar NCCL, el rendimiento será similar al de la biblioteca original. Sin embargo, gracias al alto grado de especialización de MPI en entornos de computación de alto rendimiento (HPC), especialmente en comunicaciones, los usuarios pueden esperar que sus ejecuciones de entrenamiento solo en CPU sean más rápidas.

V. Análisis de rendimiento

En esta sección se realiza una evaluación exhaustiva del rendimiento del entrenamiento y la validación del modelo en diferentes escenarios. Los resultados presentados en esta sección se han obtenido con el siguiente hardware:

- Las simulaciones CFD se ejecutan en el superordenador Tirant III. Los servidores de Tirant III están compuestos por dos sockets Intel Xeon SandyBridge E5-2670 @ 2.6GHz con un total de 16 hilos y 32 GB de memoria principal.
- El entrenamiento e inferencia del modelo predictivo se llevan a cabo en el clúster CTE-Power. Los nodos de CTE-Power están equipados con dos procesadores IBM Power9 8335-GTH @ 2.4GHz con un total de 160 hilos, 512GB de memoria principal y cuatro GPU NVIDIA V100 con 16GB de HBM2. Las GPU dentro de un nodo están conectadas a través de NVLink 2.0 @ 25Gb/s. Concretamente, las GPU 0-1 en cada host están conectadas atravesando un conjunto unido de 3 enlaces NVLink y puentes PCIe dentro del nodo NUMA de CPUs 0-79. Correspondientemente, las GPU 2-3 están conectadas atravesando sus 3 enlaces NVLink y los puentes PCIe dentro del nodo NUMA de CPUs 80-159. Cada nodo NUMA está asociado con una tarjeta de interfaz de red (NIC) diferente. Los nodos están interconectados a través de un solo puerto Mellanox EDR (25Gb/s).

Cabe destacar que con CTE-Power no solo se podría haber llevado a cabo el entrenamiento de la red neuronal, sino también la simulación CFD. Sin embargo, optamos por utilizar Tirant III para las simulaciones CFD, ya que los solucionadores no están implementados para arquitecturas de GPU. Esto se habría traducido en mucho tiempo de GPU desperdiciado durante sus ejecuciones, y muchos usuarios podrían haber visto obstaculizada su investigación.

En cuanto al conjunto de software, las simulaciones CFD se han realizado en Tirant III con OpenFOAM v2006 e Intel MPI 2018 Update 3. Mientras tanto, el modelo predictivo se ha desarrollado en

[2] `https://www.tensorflow.org/api_docs/python/tf/distribute/MultiWorkerMirroredStrategy`
[3] `https://developer.nvidia.com/nccl`

# Nodos	1	2	3	4
Tiempo (s)	75,346 s.	66,573 s.	58,678 s.	52,908 s.
Speedup	1.02x	1.15x	1.31x	1.45x

Tabla I: Tiempo de ejecución de Horovod en CPU y *Speedup* comparado con el entrenamiento de un solo nodo de Tensorflow.

# Nodos	1	2	3	4
Tiempo (s)	10,803 s.	43,288 s.	38,420 s.	32,928 s.
Speedup	-	0.25x	0.28x	0.33x

Tabla II: Timepo de ejecución y *Speedup* con GPU de *MultiworkerStrategy*.

	1 GPU	2 GPUs	3 GPUs	4 GPUs
TF `mirr.`	10,803 s.	6,703 s.	5,259 s.	4,490 s.
Hvd 1 nodo	10,931 s.	6,764 s.	5,205 s.	4,492 s.
Hvd 2 nodos	6,825 s.	4,793 s.	4,140 s.	3,585 s.
Hvd 3 nodos	5,453 s.	3,975 s.	3,649 s.	3,652 s.
Hvd 4 nodos	4,917 s.	3,803 s.	3,131 s.	3,261 s.

Tabla III: Tiempo de ejecución con GPU.

CTE-Power con Python 3.7.4, CUDA 10.2.89, Keras 2.4, Tensorflow-gpu 2.3, Horovod 0.20.3 y OpenMPI 4.0.1.

Las simulaciones CFD tardan un promedio de 9,000 segundos en ejecutarse en un nodo de 16 núcleos de Tirant III. Además, para cada uno de los 131 casos simulados, los resultados deben ser postprocesados para extraer y adaptar los datos a un formato legible por el modelo predictivo. Este postprocesamiento tarda alrededor de 2,700 segundos en un solo núcleo.

Los tiempos presentados a continuación corresponden exclusivamente al tiempo de entrenamiento, no a la inicialización del programa ni a la carga de datos. El tiempo de cada configuración se ha medido utilizando el promedio de tres ejecuciones.

Con el fin de evaluar la reproducibilidad de los resultados, la semilla aleatoria para inicializar las variables de entrenamiento se ha fijado en un valor constante.

A. Entrenamiento solo con CPU

Como prueba inicial, se realizó un estudio sobre el entrenamiento solo con CPUs en el CTE-Power con hasta cuatro nodos. Por defecto, Tensorflow y Horovod utilizan todos los núcleos con sus hilos, por lo que el número de procesos iniciados se establece en uno por nodo.

Para empezar, se ajustó el modelo con Tensorflow. Utilizando el enfoque tradicional en un solo nodo, el tiempo de entrenamiento es de 76,674 segundos. Al habilitar el entrenamiento distribuido con *MultiworkerStrategy*, surgieron una serie de volcados de memoria debido a que la versión de Tensorflow-GPU se ejecuta sin GPUs, y esa estrategia no lo admite.

Por el contrario, los tiempos de ejecución del ajuste del modelo basado en Horovod se recopilan en la Tabla I. La tabla también muestra la aceleración de Horovod en comparación con el tiempo de entrenamiento base de Tensorflow presentado antes.

Teniendo en cuenta que la ejecución de un solo nodo funciona mejor en Horovod ya que está compilado explícitamente para CPU, a pesar de que Tensorflow-GPU está instalado en la máquina CTE-Power que no puede aprovechar las optimizaciones de CPU.

Aunque la aceleración aumenta con el número de nodos, está lejos de ser lineal y demasiado alta en comparación con la configuración habilitada para GPU que se muestra en la siguiente sección.

B. Entrenamiento con GPU

En esta subsección, se estudia el entrenamiento distribuido de RNN en GPU. Como se describió en la sección Aprendizaje Distribuido, el entrenamiento distribuido del modelo en TensorFlow se puede realizar tanto con sus herramientas nativas (*MultiworkerStrategy*) como con el marco Horovod. Por esta razón, el entrenamiento se implementó en ambas alternativas.

En primer lugar, se evaluó el entrenamiento distribuido en TensorFlow. La Tabla II contiene los resultados de entrenamiento de *MultiworkerStrategy* para diferentes números de nodos involucrados. En esta evaluación, solo se ha utilizado una GPU por nodo.

Sin embargo, distribuir la carga de trabajo en varios nodos se traduce en una degradación del rendimiento. Este es un problema conocido en CTE-Power para comunicaciones de redes basadas en TCP/sockets que se informa tener la mitad del ancho de banda. A pesar de estos problemas, el entrenamiento del modelo sigue escalando junto con el número de nodos. El estudio con *MultiworkerStrategy* no se continúa con más GPUs por nodo debido a los altos tiempos obtenidos en esta etapa. En este sentido, de ahora en adelante, los resultados de las configuraciones distribuidas corresponden a Horovod con MPI, compilado para usar Infiniband sobre los puertos Mellanox EDR instalados en los nodos.

La Tabla III contiene las aceleraciones de los diferentes diseños de nodos y GPUs en comparación con TensorFlow en una sola GPU (segunda fila y columna). Los resultados de MirroredStrategy para varias GPUs en un solo nodo se colocan en la segunda fila de la tabla. El resto de las filas contiene los resultados utilizando Horovod con un número diferente de nodos y GPUs.

En las tablas anteriores, también podemos apreciar que para el mismo número de procesos, dependiendo de cómo se distribuyan entre el hardware subyacente, los tiempos varían. Las tablas revelan que la operación de entrenamiento escala con la cantidad de recursos aprovechados. A partir de esos resultados, podemos calcular el efecto negativo de la red de comunicación en diferentes diseños de procesos. Tenga en cuenta que la cantidad de procesos generados corresponde a la expresión Procesos = Nodos × GPUs.

La Tabla IV recopila la sobrecarga generada por la comunicación para los diferentes diseños para el mismo número de procesos. Cada celda contiene la variación en el porcentaje de tiempo en comparación con la celda simétrica al otro lado de la diagonal.

	1 GPU	2 GPUs	3 GPUs	4 GPUs
1 nodo	-	0.89 %	4.55 %	8.64 %
2 nodos	-0.90 %	-	2.22 %	5.73 %
3 nodos	-4.76 %	-2.27 %	-	-5.76 %
4 nodos	-9.46 %	-6.08 %	5.45 %	-

Tabla IV: Efecto de la comunicación de la red en Horovod en los tiempos de ejecuón comparado con el mimsmo número de procesos distribuidos en diferentes configuraciones.

Por ejemplo, dos procesos en un nodo que utilizan dos GPUs se ejecutan un 0,89 % más rápido que la configuración de dos nodos que utilizan una GPU en cada uno, lo que necesita un 0,9 % más de tiempo para completarse.

Teóricamente, las configuraciones en la parte superior del triángulo deberían ejecutarse más rápido (porcentajes positivos) que en la parte inferior del triángulo (porcentajes negativos), ya que menos nodos implican menos comunicación. Sin embargo, el caso de 12 GPU revela que la ejecución en cuatro nodos (tres GPU cada uno) es más rápida que en tres nodos (cuatro GPU cada uno).

Con más detalle, la figura 3 representa el tiempo de ejecución de configuraciones para diferentes números de procesos. En el eje y, el número que multiplica a la izquierda es el número de nodos y el número de GPUs a la derecha.

La tendencia teórica se basa en reducir el tiempo de ejecución con un número menor de nodos para la misma cantidad de procesos se sigue, excepto para el caso previamente mencionado de doce procesos.

Para estudiar la escalabilidad del rendimiento, la figura 4 representa el tiempo de entrenamiento para la configuración de menor tiempo de cada configuración de proceso. Además, los tiempos están asociados con el *Speedup* incremental (el tiempo de una configuración dividido por el tiempo de la inmediatamente anterior) y el *Speedup* paralelo (el tiempo de una configuración dividido por la configuración de 1 proceso).

En el gráfico, se puede identificar fácilmente la mejor configuración de rendimiento con el menor tiempo y la mayor aceleración paralela. Corresponde a la configuración de 12 procesos. En este sentido, parece que debido a la naturaleza del problema, se ajusta mejor a esta configuración al usar cuatro nodos.

Además, dependiendo de nuestras necesidades, podemos determinar una configuración "perfecta" que logre un equilibrio entre el tiempo de ejecución y los recursos utilizados. Para este estudio esta puede establecerse en la configuración de dos procesos, donde el incremento de la velocidad es superior a 1,5x. Y probablemente estaría en entornos compartidos, sin embargo, dado que todos los experimentos se han realizado utilizando nodos exclusivos, el mejor equilibrio entre tiempo y recursos se puede definir como cuatro procesos ejecutándose en el mismo nodo. Esta configuración muestra una aceleración incremental de 1,16x y una aceleración paralela de 2,43x.

VI. Conclusiones

En resumen, el estudio investigó el entrenamiento distribuido del modelo de red neuronal recurrente (RNN) para la simulación de flujo de fluidos computacional (CFD) impulsada por datos, aplicando y evaluando dos métodos de distribución, Horovod y las estrategias *MirroredStrategy* y *MultiworkerStrategy* de Tensorflow. El modelo propuesto se validó con un solucionador OpenFOAM y los resultados mostraron una precisión del 99 % en la predicción de pasos de tiempo individuales. También se comparó el rendimiento de las etapas de entrenamiento e inferencia entre diferentes configuraciones de CPU y GPU, y se observó que el tiempo de entrenamiento con una GPU reducía el tiempo de entrenamiento en un 7.1x y el tiempo de CPU con una configuración de rendimiento máximo de cuatro nodos y tres GPU por nodo se aceleraba en un 24.49x.

Además, el estudio reveló que la baja aceleración obtenida al usar múltiples GPU se debió a la sobrecarga de comunicación y coordinación de los cálculos entre las GPU, lo que superó los beneficios de rendimiento de usar múltiples GPU. También se comparó la experiencia del usuario al usar Horovod y *MultiworkerStrategy*, y se observó que Horovod habilitado para MPI aprovecha la configuración de red del clúster implementado para MPI, lo que permite a los usuarios explotar todo el potencial de la red sin necesidad de ajustar nada.

En general, el modelo y el análisis de rendimiento presentados en el estudio pueden ser útiles en estrategias híbridas AI-CFD de simulación basadas en modelos de sustitución o co-simulación, y el modelo predictivo desarrollado en el estudio se transferirá a la arquitectura de Unidades de Procesamiento de Inteligencia (IPU) para evaluar el rendimiento del entrenamiento en otras plataformas. Todos los códigos desarrollados para el estudio están disponibles en https://github.com/AlejandroGB13/CFD_AI y los conjuntos de datos están disponibles bajo demanda.

Agradecimientos

Los autores agradecen el acceso a los clústers Tirant III del Servei d'Informàtica de la Universidad de Valencia (UV) y CTE-Power del Barcelona Supercomputing Center (BSC).

Referencias

[1] Rick Archibald, Edmond Chow, Eduardo D'Azevedo, Jack Dongarra, Markus Eisenbach, Rocco Febbo, Florent Lopez, Daniel Nichols, Stanimire Tomov, Kwai Wong, and Junqi Yin, "Integrating Deep Learning in Domain Sciences at Exascale," in *Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI*, Jeffrey Nichols, Becky Verastegui, Arthur 'Barney' Maccabe, Oscar Hernandez, Suzanne Parete-Koon, and Theresa Ahearn, Eds., Cham, 2020, pp. 35–50, Springer International Publishing.

[2] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos, "Machine Learning for Fluid Mechanics," *Annual Review of Fluid Mechanics*, vol. 52, no. 1, may 2020.

[3] Sergio Iserte, Pablo Carratala, Rosario Arnau, Paloma Barreda, Luís Basiero, Raúl Matínez-Cuenca, Javier Climent, and Sergio Chiva, "Modeling of Wastewater Treat-

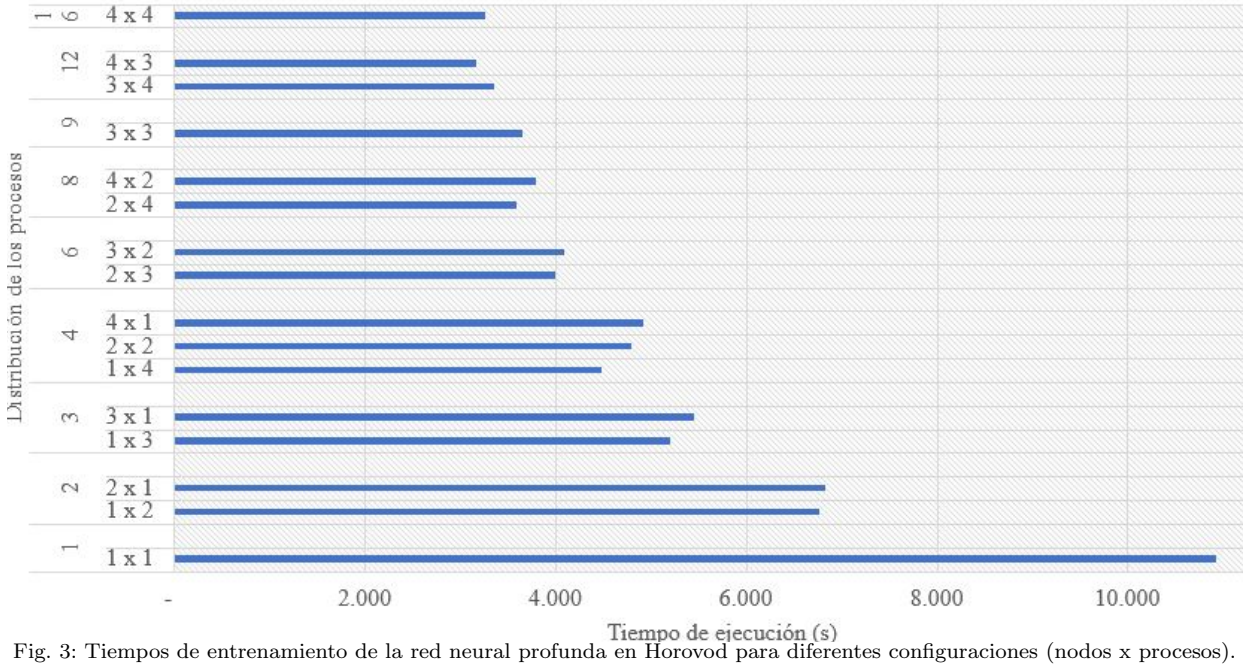


Fig. 3: Tiempos de entrenamiento de la red neural profunda en Horovod para diferentes configuraciones (nodos x procesos).

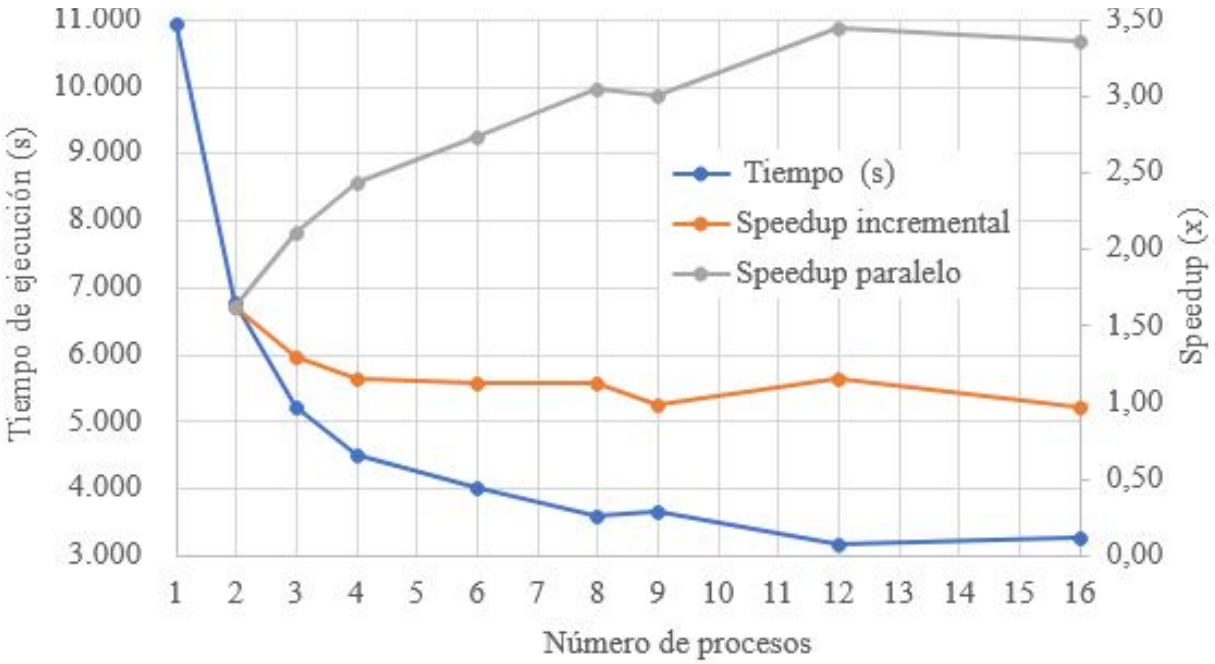


Fig. 4: Tiempo de entrenamiento y *Speedup* con las diferentes configuraciones en Horovod.

ment Processes with HydroSludge," *Water Environment Research*, pp. 1–38, 2021.

[4] D. Bhatt, B.W. Zhang, and D.M. Zuckerman, "Steady-state Simulations Using Weighted Ensemble Path Sampling," *The Journal of Chemical Physics*, vol. 133, no. 1, 2010.

[5] Ricardo Vinuesa and Steven L Brunton, "The Potential of Machine Learning to Enhance Computational Fluid Dynamics," *arXiv preprint arXiv:2110.02085*, 2021.

[6] Krzysztof Rojek, Roman Wyrzykowski, and Pawel Gepner, "AI-Accelerated CFD Simulation Based on OpenFOAM and CPU/GPU Computing," in *Computational Science – ICCS 2021*. 2021, pp. 373–385, Springer International Publishing.

[7] D. Xiao, C. E. Heaney, L. Mottet, F. Fang, W. Lin, I. M. Navon, Y. Guo, O. K. Matar, A. G. Robins, and C. C. Pain, "A Reduced Order Model for Turbulent Flows in the Urban Environment Using Machine Learning," *Building and Environment*, vol. 148, pp. 323–337, jan 2019.

[8] Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler, "Deep Fluids: A Generative Network for Parameterized Fluid Simulations," *Computer Graphics Forum*, vol. 38, no. 2, pp. 59–70, 2019.

[9] Guillem Ramirez-Gargallo, Marta Garcia-Gasulla, and Filippo Mantovani, "Tensorflow on state-of-the-art hpc clusters: A machine learning use case," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2019, pp. 526–533.

[10] Ammar Ahmad Awan, Jereon Bédorf, Ching-Hsiang Chu, Hari Subramoni, and Dhabaleswar K. Panda, "Scalable distributed dnn training using tensorflow and cuda-aware mpi: Characterization, designs, and performance evaluation," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2019, pp. 498–507.

[11] Sergio Iserte, Alejandro González-Barberá, Paloma Barreda, and Krzysztof Rojek, "A study on the performance of distributed training of data-driven cfd simulations," *The International Journal of High Performance Compu-*

ting Applications, p. 109434202311605, May 2023.

[12] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "TensorFlow: A System for Large-Scale Machine Learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, USA, 2016, OSDI'16, p. 265–283, USENIX Association.

[13] Alexander Sergeev and Mike Del Balso, "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow," *CoRR*, vol. abs/1802.05799, 2018.

Usando Múltiples GPU para Acelerar la Detección de Regiones Diferencialmente Metiladas

Carlos Reaño[1], Ricardo Olanda[1], Elvira Baydal[2], Mariano Pérez[1] y Juan M. Orduña[1]

Resumen— **El análisis de la metilación del ADN se ha convertido en un tema importante en el estudio de la salud humana. En trabajos anteriores, hemos desarrollado un conjunto completo de herramientas que permiten llevar a cabo el proceso completo de análisis de metilación del ADN y la detección automática de Regiones Diferencialmente Metiladas (DMR) entre diferentes muestras, incluyendo una versión basada en web. El back-end de esta herramienta web debe admitir un número indefinido de solicitudes simultáneas de detección de las DMR en diferentes conjuntos de datos. En la actualidad la GPU del back-end procesa las peticiones simultáneas de forma secuencial, lo que reduce enormemente su rendimiento. En este trabajo proponemos una arquitectura paralela para el servidor back-end de esta herramienta basada en web. La arquitectura propuesta introduce el uso de un planificador y varios demonios que permiten atender las solicitudes simultáneamente. Los resultados de evaluación de las prestaciones muestran que la arquitectura paralela propuesta reduce de forma importante los tiempos de ejecución en comparación con la arquitectura anterior. Además, la velocidad aumenta linealmente con el número de GPU disponibles (hasta 1,96x en nuestra configuración experimental con 2 GPU). Estos resultados demuestran que la arquitectura propuesta es capaz de explotar al máximo el paralelismo subyacente disponible en la plataforma hardware del servidor back-end.**

Palabras clave— **Análisis de metilación del ADN, software como servicio, computación GPU.**

I. Introducción

El análisis de la metilación del ADN es, en estos momentos, un tema importante en el estudio de la salud humana y en otros campos de la biotecnología como la fitología [1] o la química orgánica [2]. La razón de esta tendencia es que la metilación del ADN desempeña un papel clave en el control local de la expresión génica [3], el establecimiento y mantenimiento de la identidad celular [4], la regulación del desarrollo embrionario de los mamíferos [5] y otros procesos biológicos [6]. Por ejemplo, el proceso de metilación puede impedir que un gen causante de tumores "se active"(se transcriba al ARN), previniendo el cáncer, y también parece desempeñar un papel decisivo en otras enfermedades complejas como la obesidad, la hipertensión y el desarrollo de la diabetes mellitus tipo 2 (DM2) [7].

El genoma humano de referencia está compuesto por 3 x 10^9 nucleótidos. Para analizar la metilación del ADN de una persona a sus muestras de ADN se les añade bisulfito, lo que provoca que las citosinas que no están metiladas se conviertan en timinas, mientras que las citosinas metiladas permanezcan en las muestras como citosinas. Los secuenciadores de ADN producen de esta forma millones de segmentos cortos de ADN (llamados *lecturas* o *reads*), cuyo tamaño no suele superar los cientos de nucleótidos, en un archivo fastq (archivo de texto). Cada lectura en el archivo fastq debe compararse con todas las posibles ubicaciones (nucleótidos) en el genoma de referencia para encontrar la ubicación correcta de la lectura en el genoma de referencia (operación de alineación), así como el estado de metilación de las citosinas en la lectura. Cada archivo fastq procedente de un secuenciador de nueva generación puede contener fácilmente decenas o cientos de millones de lecturas. Además, los casos típicos de estudio requieren el análisis y la comparación de diferentes muestras biológicas procedentes de diferentes tejidos o diferentes individuos, con el fin de detectar diferentes niveles de metilación en diferentes regiones del ADN, que se denominan regiones diferencialmente metiladas (DMR). Normalmente, los estudios sobre los efectos de la metilación en el cáncer [8] utilizan muestras procedentes de un número de personas normales (control) y otras muestras procedentes del mismo número de personas que padecen cáncer (casos), e intentan encontrar DMR entre las muestras de control y las de los casos. Por lo tanto, el enorme tamaño de los datos implicados en el análisis de metilación del ADN requiere el uso de arquitecturas de alto rendimiento para procesar las muestras de forma conveniente.

En trabajos anteriores, desarrollamos un conjunto completo de herramientas (disponibles en GitHub [9]) que permiten llevar a cabo el proceso completo de alineamiento de ADN y el análisis de metilación (HPG-Methyl ([10], [11])), la creación de mapas de metilación de todo el genoma (HPG-HMapper [12]), y la detección y visualización gráfica de DMR entre diferentes muestras (HPG-DHunter [13], [14]). HPG-DHunter permite identificar y visualizar las DMR de diferentes muestras a diferentes niveles. Sin embargo, HPG-DHunter se desarrolló como una herramienta independiente para ser instalada en un servidor de altas prestaciones con dispositivos GPU de nueva generación. Por lo tanto, su uso requiere conocimientos sobre la instalación, configuración y mantenimiento de las GPU. Este requisito limita su uso por parte de investigadores biomédicos con escasos o nulos conocimientos

[1]Departament d'Informàtica, Universitat de València, e-mails: {carlos.reano, ricardo.olanda, mariano.perez, juan.orduna}@uv.es.

[2]Departament d'Informàtica de Sistemes i Computadors, Universitat Politècnica de València, e-mail: mebaydal@disca.upv.es.

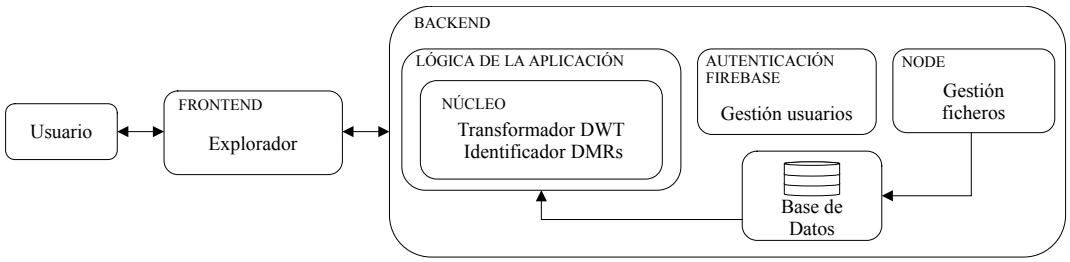


Fig. 1: Arquitectura previa de la aplicación basada en web.

sobre este hardware y los paquetes de software relacionados. Por ello, se desarrolló una versión web de la herramienta [15], que permite a los investigadores biomédicos aprovechar su potencial para el análisis de metilación, incluso aunque no estén familiarizados con el manejo de las GPU y su software relacionado.

No obstante, el back-end de la herramienta web debe admitir un número indefinido de peticiones simultáneas para detectar las DMR en distintos conjuntos de datos. El algoritmo de detección de las DMR se basa en procedimientos llevados a cabo en la GPU, por lo que las peticiones simultáneas pueden tener que esperar mucho tiempo en una cola FIFO de peticiones entrantes, hasta que la GPU se queda inactiva. Esto reduce enormemente el rendimiento del back-end. En este trabajo, proponemos una arquitectura paralela para el servidor back-end de la herramienta basada en web. Los resultados de la evaluación del rendimiento muestran que, independientemente del número de GPU físicas disponibles en la plataforma de cálculo que actúa como servidor back-end, una arquitectura paralela con varios demonios puede explotar plenamente el paralelismo de las GPU. Cada demonio se encarga de atender una de las solicitudes simultáneas, por lo que la arquitectura propuesta reduce significativamente los tiempos de ejecución con respecto a la arquitectura anterior, en la que un único demonio procesaba secuencialmente todas las peticiones simultáneas. Para obtener esta mejora la arquitectura propuesta debe utilizarse en plataformas con varias GPU físicas.

El resto del trabajo se organiza de la forma siguiente. La Sección II describe los antecedentes y la arquitectura de la versión web actual de HPG-DHunter. La Sección III describe la arquitectura propuesta para el back-end del servicio HPG-DHunter. A continuación, la Sección IV muestra la evaluación de las prestaciones de la arquitectura propuesta. Finalmente, la Sección V muestra las conclusiones de este trabajo.

II. Antecedentes

La versión web [15] de la herramienta HPG-DHunter [13] se puede instalar en cualquier ordenador, y su uso no requiere que los investigadores biomédicos dispongan de los conocimientos necesarios para instalar y gestionar esta herramienta ni del software necesario para el cálculo en la GPU.

La infraestructura de esta aplicación web se basa en una arquitectura cliente-servidor. Se ha diseñado para separar el front-end (interfaz de usuario y lógica de control) del back- end (infraestructura del lado del servidor que aloja los servicios para procesar las peticiones del front-end), como puede verse en la Figura 1. El front-end se implementa con el framework Angular. Se ejecuta en el navegador del usuario utilizando HTML5 y CSS para dar formato al contenido en la interfaz del navegador, y JavaScript para la captura de eventos y la comunicación con el back-end. Para minimizar la latencia al transferir grandes cantidades de datos, se utiliza el protocolo Web-Socket sobre una conexión TCP permanente, con JSON como formato de intercambio de datos. El back-end se implementa con el framework Qt, siendo el software HPG-DHunter el núcleo del sistema, con un módulo separado para el control de acceso de usuarios y la gestión de archivos.

La aplicación web se ha diseñado para ofrecer un alto grado de seguridad, ya que las muestras de metilación proporcionadas por los usuarios son datos sensibles. El acceso individual se controla con un nombre de usuario y una contraseña, y los datos enviados al servidor por un usuario determinado se almacenan en carpetas independientes y aisladas, que quedan ocultas para los demás usuarios. Además, las conexiones se cifran mediante Web-Sockets sobre TLS (Transport Layer Security).

La interfaz front-end pretende ofrecer un nivel de usabilidad igual o superior al de la versión autónoma (HPG-DHunter). La identificación de la DMR se lleva a cabo después de que el usuario configure los parámetros de análisis, como el nivel de transformación, la cobertura mínima, un umbral de validación y el porcentaje mínimo de posiciones con cobertura mínima. El sistema puede mostrar cualquier DMR encontrada en cualquier nivel de metilación de cada muestra cargada por el usuario.

El back-end, como se muestra en la Figura 1, incluye el servicio de autenticación, el gestor de archivos y el servicio principal de transformación wavelet e identificación de la DMR. En el lado del servidor, el módulo de base de datos se encarga de la gestión de archivos y proporciona al módulo lógico de la aplicación los datos necesarios según las peticiones del usuario.

La aplicación web puede gestionar múltiples tareas solicitadas simultáneamente por distintos usuarios. El sistema back-end gestiona la disponibilidad de memoria RAM para cargar datos de archivos y el estado de la GPU para tareas de cálculo. En la

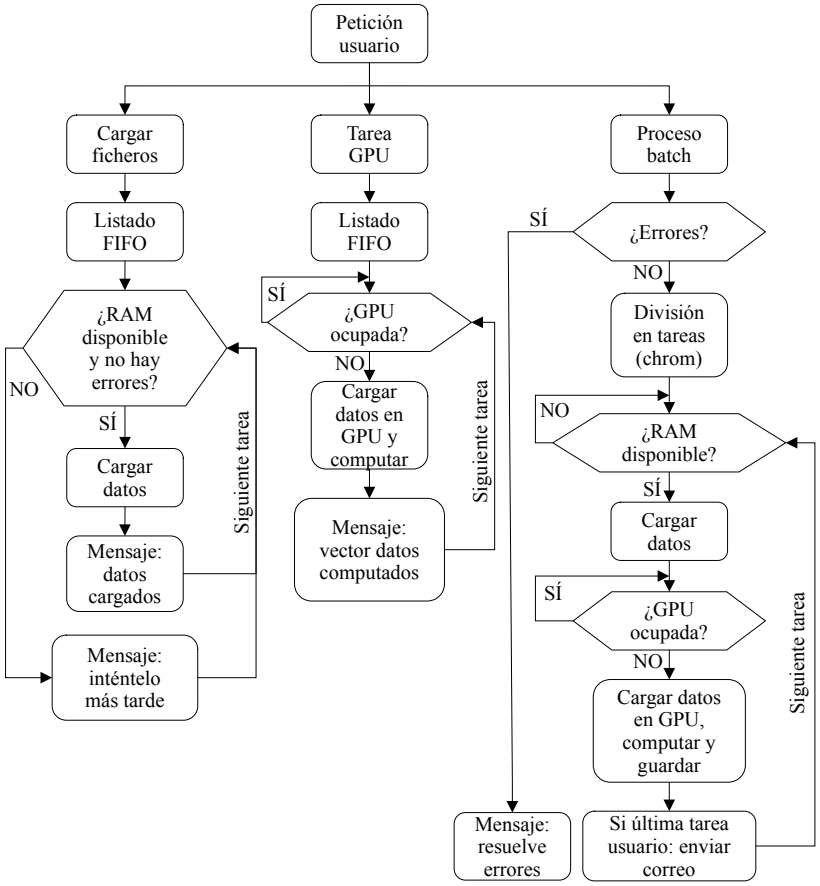


Fig. 2: Diagrama de flujo para la gestión de peticiones concurrentes en la arquitectura anterior de la herramienta basada en web.

Figura 2 se muestra un diagrama de flujo que ilustra la gestión de las peticiones concurrentes procedentes de distintos clientes. Esta figura muestra tres tipos diferentes de peticiones, teniendo cada una su propia lista FIFO de tareas. Las tareas se lanzan independientemente de estas listas. El proceso de carga de archivos no requiere disponibilidad de GPU, pero sí de memoria RAM, por lo que debe comprobarse la disponibilidad de memoria RAM antes de iniciar este proceso. Las tareas necesarias para la visualización requieren el uso de la GPU, por lo que debe comprobarse su estado antes de cargar la matriz de datos y calcular la wavelet de la señal de metilación. Por último, otro proceso que puede ser solicitado por el usuario es un proceso por lotes, que es un procedimiento largo y no interactivo. En este caso, el sistema envía un correo electrónico al usuario cuando el proceso se ha completado, y el procedimiento se divide en tareas más pequeñas. Todas estas estrategias tienen como objetivo mejorar el uso de la GPU y la memoria RAM cuando se producen accesos concurrentes al servidor. En el caso particular de las tareas de la GPU, y dado que se supone que se dispone de una única GPU, las peticiones se ponen en cola y se gestionan utilizando una política FIFO (First-In, First-Out). Se trata de una forma sencilla y eficaz de garantizar que las solicitudes se gestionan de forma puntual y justa, y garantiza que ninguna solicitud tiene prioridad sobre otra.

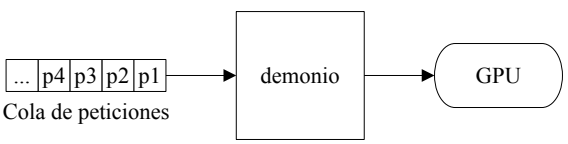


Fig. 3: Arquitectura actual.

III. Una Nueva Arquitectura Diseñada para Utilizar Varias GPU

En esta sección, describimos en primer lugar la arquitectura actual del sistema back-end a la hora de gestionar las peticiones de los usuarios que requieren el uso de la GPU (es decir, una tarea de GPU o un proceso por lotes o batch, como se muestra en la Figura 1. En el resto del documento, nos referiremos a estas peticiones de usuario como peticiones de GPU). A continuación, describimos la arquitectura propuesta, diseñada para utilizar múltiples GPU si están disponibles en la plataforma de computación del servidor.

La Figura 3 muestra la arquitectura actual del servidor del sistema para atender las peticiones de GPU de los usuarios. Independientemente del número de GPU de la plataforma de computación que actúe como servidor back-end, esta arquitectura sólo es capaz de utilizar una única tarjeta GPU, gestionada por un único demonio. Por tanto, todas las peticiones de GPU se encolan en una cola FIFO y son procesadas

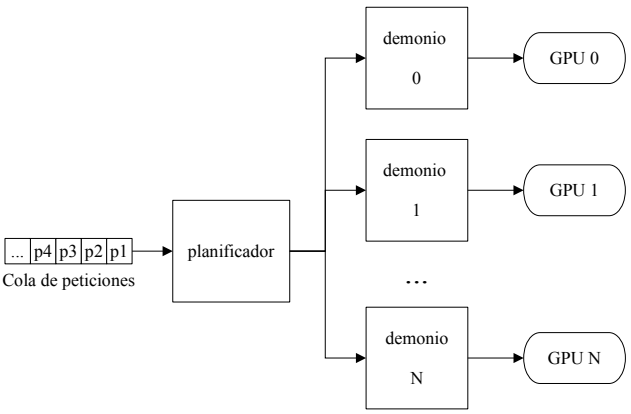


Fig. 4: Arquitectura propuesta.

secuencialmente por el único demonio.

Aunque esta arquitectura proporciona procesamiento paralelo dentro de la GPU, el procesamiento de las distintas peticiones se realiza de forma secuencial y, por tanto, puede dar lugar a tiempos de respuesta muy largos para algunas peticiones de usuario si varios usuarios acceden simultáneamente al servidor. Así pues, para proporcionar una herramienta escalable con el número de peticiones de los usuarios, se necesita una nueva arquitectura que pueda aprovechar las ventajas de disponer de varias tarjetas GPU en el servidor back-end.

La Figura 4 muestra la arquitectura propuesta del servidor del sistema para atender las peticiones de GPU de los usuarios, junto con los posibles escenarios que permite. Esta nueva arquitectura incluye los siguientes elementos:

- Una cola de solicitudes que almacena las solicitudes recibidas de los usuarios.
- Un planificador, responsable de la gestión de la cola y del reenvío de las peticiones a los demonios.
- Demonios para atender las peticiones de los usuarios. Estos demonios deben utilizar todas las GPU disponibles en el sistema.

En cuanto a los posibles escenarios que permite la arquitectura propuesta, en este artículo nos centraremos en el mostrado en la Figura 4. En este escenario, el sistema tiene varias GPU y hay un único demonio por GPU. Cada demonio utiliza una GPU diferente para atender una petición determinada. Como puede observarse, la principal diferencia entre la arquitectura actual (Figura 3) y la arquitectura propuesta (Figura 4) es el planificador, que permite la utilización de todas las GPU disponibles en la plataforma de computación.

IV. Resultados experimentales

En esta sección evaluamos el rendimiento de la arquitectura propuesta. En primer lugar, se detalla la configuración experimental. A continuación, se compara el rendimiento de la arquitectura anterior con el de la propuesta al ejecutar una única petición en una plataforma informática con una única GPU disponible. Después, se analiza el rendimiento al recibir múltiples peticiones simultáneas en la misma plataforma. Por último, evaluamos el rendimiento cuando se utilizan varias GPU. Cada valor mostrado en las tablas y figuras incluidas en esta sección se ha calculado como el valor medio de 10 ejecuciones de la prueba correspondiente.

Tabla I: Comparación de rendimiento entre la arquitectura anterior y la propuesta para una única petición y GPU.

Valores Medios	Versión Anterior	Versión Propuesta
Tiempo Ejecución (s)	9.49	9.52
Utilización CPU (%)	1.46	1.46
Memoria CPU (%)	1.01	1.01
Utilización GPU (%)	1.25	1.25
Memoria GPU (%)	1.02	1.02

A. Configuración experimental

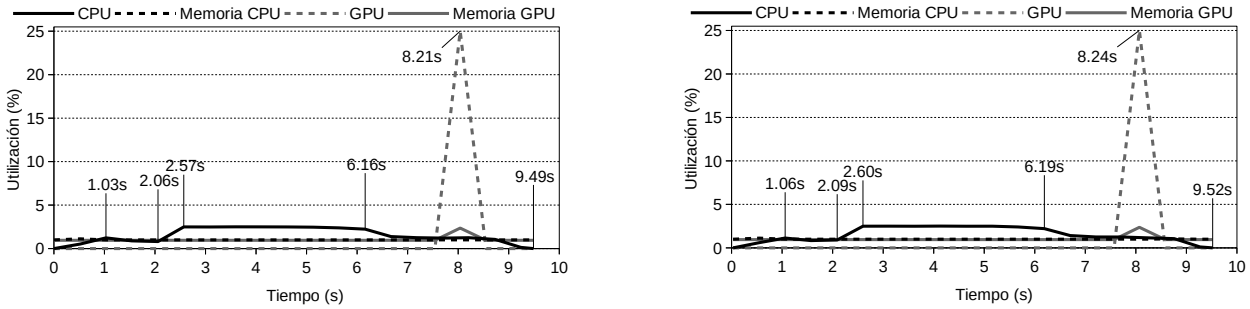
El sistema utilizado en los experimentos es un servidor BullSequana X450-E5 de 2 sockets, con dos CPU Intel(R) Xeon(R) Gold 6230 de 20 núcleos a 2,10 GHz y 191 GB de RAM, un RAID 1 con dos discos duros de 10,9 TB y dos GPU Tesla V100 PCIe con 32 GB de RAM cada una. El sistema operativo instalado en esta plataforma es Rocky Linux 8.7 (Green Obsidian), incluyendo NVIDIA CUDA 12 con el controlador NVIDIA versión 525.60.13.

La prueba utilizada para los experimentos es muy corta y requiere muy poco tiempo de cálculo en la GPU. La razón de seleccionar este tipo de prueba es que en este trabajo nos centramos en el planificador y no en el cálculo en la GPU, que ya se ha tratado en trabajos previos [13]. Como se ha comentado en secciones anteriores, la principal diferencia entre la arquitectura anterior y la arquitectura propuesta en este trabajo es el planificador. Por tanto, una prueba corta es el peor caso para probar nuestro enfoque, ya que la sobrecarga introducida por el planificador no se ocultará tras un largo tiempo de cálculo en la GPU.

B. Evaluación del rendimiento de una sola solicitud

La Tabla I presenta una comparación de la arquitectura anterior con la arquitectura propuesta descrita en la sección anterior. Se comparan distintos valores, a saber, tiempo de ejecución, utilización de la CPU, memoria de la CPU, utilización de la GPU y memoria de la GPU. La desviación estándar relativa (DER) máxima observada en estos experimentos fue de 0,290 para el tiempo de ejecución de la versión anterior.

La Tabla I muestra que la arquitectura propuesta aumenta ligeramente el tiempo de ejecución (30 ms) de media. El resto de las métricas obtienen diferencias insignificantes (no se aprecian con sólo dos dígitos decimales). Para explicar mejor la razón de este aumento en el tiempo de ejecución, la Figura 5 presenta un gráfico más detallado, donde la Figura 5a



(a) Versión original.

(b) Nueva versión.

Fig. 5: Comparación detallada de la arquitectura anterior y la propuesta para una única petición y una única GPU.

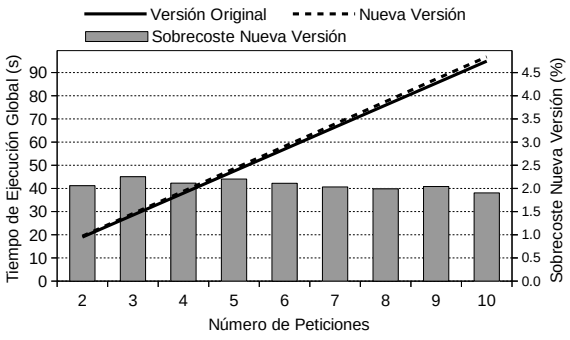


Fig. 6: Resultados de la evaluación para varias peticiones simultáneas y una única GPU.

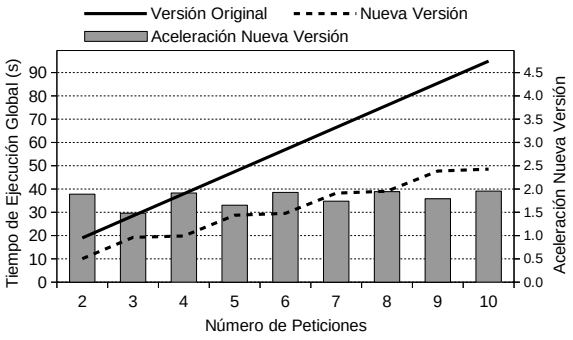


Fig. 7: Resultados de la evaluación cuando se envían múltiples peticiones de forma concurrente al sistema utilizando dos GPU.

muestra los valores de rendimiento de la versión anterior y la Figura 5b los de la versión propuesta. En esta figura se aprecia que el incremento en el tiempo de ejecución de la versión propuesta es el tiempo requerido por el planificador para reenviar la petición desde la cola de peticiones hasta el demonio que sirve la petición. Como se puede observar, una vez que esto ha ocurrido, la nueva versión simplemente imita a la versión original con un retardo de 30 ms. En la figura se han etiquetado varios puntos para facilitar la comparación.

Hay que tener en cuenta que en este experimento sólo se envió una petición al sistema y sólo un demonio estaba sirviendo peticiones. Por ello, el planificador necesitó muy poco tiempo para reenviar la petición al demonio disponible. En la siguiente sección evaluamos cómo se incrementa este tiempo cuando se ejecutan múltiples peticiones.

C. *Evaluación del rendimiento de varias solicitudes simultáneas*

En esta sección evaluamos el rendimiento cuando se envían dos o más peticiones simultáneas al sistema, y se dispone de una única GPU en la plataforma informática del back-end.

Para los resultados de la nueva versión, solo un demonio utiliza la única GPU, es decir, es el escenario mostrado en la Figura 4 con una sola GPU. La Figura 6 muestra los resultados para un número de solicitudes concurrentes que oscila entre 2 y 10 peticiones.

Como era de esperar, en este caso la sobrecarga introducida por el planificador es mayor (1,85 % de media) que en el caso de una única solicitud. También puede observarse que esta sobrecarga tiende a disminuir ligeramente a medida que aumenta el número de peticiones. La DER máxima observada en estos experimentos fue de 2,050, correspondiente al tiempo de ejecución de 3 peticiones concurrentes con la nueva versión. En cuanto a los tiempos de ejecución, no existen diferencias significativas entre la arquitectura anterior y la propuesta. Estos resultados demuestran que la arquitectura propuesta no añade una sobrecarga significativa para un ordenador con una única GPU.

D. *Evaluación del rendimiento utilizando varias GPU*

En esta sección, evaluamos el rendimiento cuando se envía más de una solicitud simultáneamente al sistema y se utilizan varias GPU. Es decir, queremos medir las ventajas de la nueva arquitectura propuesta cuando hay más de una GPU presente en la plataforma informática que actúa como servidor back-end. Para estos experimentos, suponemos que hay un demonio por GPU, como se muestra en la Figura 4. Los resultados de la evaluación se observan en la Figura 7 y en la Figura 8.

La Figura 7 muestra los resultados para un número de peticiones concurrentes que oscila entre 2 y 10 cuando se utilizan 2 GPU. La DER máxima observada en estos experimentos fue de 1,775, para el tiempo de ejecución de 3 peticiones concurrentes con la nue-

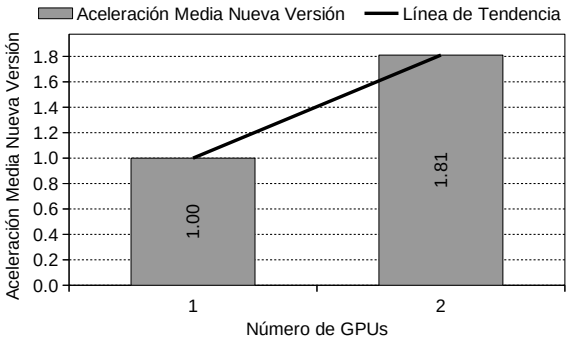


Fig. 8: Aceleración media obtenida cuando se envían múltiples peticiones de forma concurrente al sistema utilizando dos GPU.

va versión. Este gráfico muestra que, como cabía esperar, la nueva versión de la arquitectura aprovecha claramente el hardware subyacente en la plataforma de cálculo, consiguiendo un aumento de velocidad de hasta 1,96x, bastante próximo al máximo teórico.

Por último, la Figura 8 muestra la aceleración media de estos experimentos cuando hay dos GPU disponibles en la plataforma de cálculo que actúa como servidor back-end. Como podemos observar, la arquitectura propuesta escala adecuadamente con el número de GPU disponibles, presentando una tendencia lineal.

V. Conclusiones

En este artículo, hemos propuesto una nueva arquitectura paralela para mejorar el rendimiento de un servidor back-end que procesa las peticiones de los usuarios enviadas a una herramienta web para la detección automática de DMR. En esta arquitectura, existen múltiples demonios que atienden simultáneamente las peticiones. Un planificador se encarga de gestionar las peticiones y reenviarlas a los demonios para que las atiendan. Los resultados muestran que la arquitectura paralela es capaz de explotar plenamente el paralelismo del servidor back-end. La arquitectura propuesta reduce significativamente los tiempos de ejecución en comparación con la arquitectura anterior, en la que un único demonio procesaba secuencialmente todas las peticiones. Por último, en nuestra configuración experimental con dos GPU, hemos observado que la velocidad aumenta linealmente hasta 1,96x.

Agradecimientos

Este trabajo está financiado por la Generalitat Valenciana mediante la subvención CIGE/2021/132. Los autores también desean dar las gracias a Enrique Bonet Esteban y Manolo Pérez Aixendri por su apoyo inestimable en la instalación y configuración del clúster informático utilizado en los experimentos.

Referencias

[1] Javier Gallego-Bartolomé, "Dna methylation in plants: mechanisms and tools for targeted manipulation," *New Phytologist*, vol. 227, no. 1, pp. 38–44, 2020.

[2] Yantao Chen, "Recent advances in methylation: A guide for selecting methylation reagents," *Chemistry – A European Journal*, vol. 25, no. 14, pp. 3405–3439, 2019.

[3] D Schubeler, "Function and information content of dna methylation," *Nature*, vol. 517, pp. 321–326, 2015.

[4] Shizhao Li, Min Chen, Yuanyuan Li, and Trygve O. Tollefsbol, "Prenatal epigenetics diets play protective roles against environmental pollution," *Clinical Epigenetics*, vol. 11, pp. 82, 2019.

[5] Helena Fulka, Milan Mrazek, Olga Tepla, and Josef Fulka, "Dna methylation pattern in human zygotes and developing embryos," *Reproduction*, vol. 128, no. 6, pp. 703 – 708, 2004.

[6] K Robertson, "Dna methylation and human disease," *Nature Reviews Genetics*, vol. 6, pp. 597–610, 2005.

[7] Alexander Raciti, Cecilia Nigro, Michele Longo, Luca Parrillo, Claudia Miele, Pietro Formisano, and Francesco Béguino, "Personalized medicine and type 2 diabetes: lesson from epigenetics.," *Epigenomics*, vol. 6, no. 2, pp. 229–238, 2014.

[8] N Shenoy, TD Bhagat, J Cheville, C Lohse, S Bhattacharyya, A Tischer, V Machha, l S Gordon-Mitchel, G Choudhary, Wong. L.F., L Gross, E Ressigue, B Leibovich, SA Boorjian, U Steidl, X Wu, K Pradhan, l B Gartrel, B Agarwal, L Pagliaro, M Suzuki, JM Greally, D Rakheja, RH Thompson, K Susztak, T Witzig, Y Zou, and A Verma, "Ascorbic acid-induced tet activation mitigates adverse hydroxymethylcytosine loss in renal cell carcinoma," *The Journal of Clinical Investigation*, vol. 129, no. 4, pp. 1612–1625, 2019.

[9] Networks and Virtual Environments Group (GREV), Universitat de València, "HPG-Msuite, the methylation analysis ultimate tools suite," `https://grev-uv.github.io/`, 2020.

[10] Joaquín Tárraga, Mariano Pérez, Juan M. Orduña, José Duato, Ignacio Medina, and Joaquín Dopazo, "A parallel and sensitive software tool for methylation analysis on multicore platforms," *Bioinformatics*, vol. 31, no. 19, pp. 3130, 2015.

[11] Ricardo Olanda, Mariano Pérez, Juan M. Orduña, Joaquín Tárraga, and Joaquín Dopazo, "A new parallel pipeline for DNA methylation analysis of long reads datasets," *BMC Bioinformatics*, vol. 18, no. 1, pp. 161, 2017.

[12] César González, Mariano Pérez, and Juan M. Orduña, "Hpg-hmapper: A dna hydroxymethylation analysis tool," *International Journal of High Performance Computing Applications*, 2019.

[13] Lisardo Fernández, Mariano Pérez, Ricardo Olanda, Juan M. Orduña, and Joan Marquez-Molins, "HPG-DHunter: an ultrafast, friendly tool for DMR detection and visualization," *BMC Bioinformatics*, vol. 21, no. 1, pp. 287, 2020, ID: Fernández2020.

[14] Networks and Virtual Environments Group (GREV), Universitat de València, "HPG-DHunter, a tool for detecting Differentially Methylated Regions (DMRs)," `https://github.com/grev-uv/hpg-dhunter-batch`, 2020.

[15] Lisardo Fernández, Ricardo Olanda, Mariano Pérez, and Juan M. Orduña, "A web-based tool for automatic detection and visualization of dna differentially methylated regions," *Electronics*, vol. 10, no. 9, 2021.

mRMR+KNN Energéticamente Eficiente para Clasificación de EEGs: Caso de Estudio en Plataformas Heterogéneas

Juan José Escobar[1], Francisco Rodríguez[2], Beatriz Prieto[2], Andrés Ortiz[3], Alberto Prieto[2] y Miguel Damas[2]

Resumen— **El creciente consumo de energía provocado por las TIC está obligando a los desarrolladores a considerar la eficiencia energética como parámetro fundamental de diseño. Este parámetro adquiere gran relevancia en sistemas HPC cuando se ejecutan redes neuronales artificiales y aplicaciones de Machine Learning. Es por ello que en este artículo se muestra un ejemplo de cómo estimar y considerar el consumo de energía en un caso real de clasificación de Electroencefalogramas (EEGs). Se propone una implementación eficiente y distribuida del algoritmo KNN que utiliza mRMR como técnica de selección de características para reducir la dimensionalidad del conjunto de datos. También se analiza el rendimiento de tres distribuciones de carga de trabajo diferentes para identificar cuál es la más adecuada según las condiciones experimentales. El enfoque propuesto supera los resultados de clasificación obtenidos por trabajos anteriores, alcanzando una tasa de precisión del 88,8 % y una ganancia de velocidad de 74,53 cuando se ejecuta en un clúster heterogéneo multinodo, consumiendo sólo el 13,38 % de la energía de la versión secuencial.**

Palabras clave— **Programación paralela y distribuida, Clústeres heterogéneos, Computación Consciente de la Energía, Clasificación de EEGs, KNN, mRMR.**

I. Introducción

LAS emisiones de gases de efecto invernadero causadas por el consumo de energía asociado a la proliferación de equipos, aplicaciones y programas informáticos crece de forma preocupante. Hay estimaciones que determinan que las TIC podrían aportar hasta un 23 % de las emisiones globales de gases de efecto invernadero en 2030 [1]. Para reducir este consumo, es necesario abordar el problema simultáneamente desde diferentes enfoques. Uno de ellos es analizar el consumo de energía de los programas e intentar ejecutarlos en la configuración más eficiente energéticamente. Otro enfoque es explotar las cualidades de las plataformas paralelas distribuidas y heterogéneas. Actualmente, a la hora de diseñar aplicaciones, no basta con considerar sólo la precisión de los resultados y el tiempo de ejecución, sino que también se debe considerar la eficiencia energética como uno de los parámetros fundamentales. De entre las aplicaciones más complejas en cuanto a tiempo de ejecución y consumo energético, se encuentran las relacionadas con Machine Learning. Estas aplicaciones a menudo tienen que procesar grandes cantidades de datos (Big Data) y se caracterizan por su alta complejidad algorítmica. Por tanto, este trabajo muestra los resultados de una investigación que analiza la eficiencia energética, tiempo de ejecución y precisión de resultados de una aplicación de bioingeniería basada en el algoritmo KNN (K Vecinos más Cercanos), que es capaz de explotar las cualidades de una plataforma paralela distribuida y heterogénea para obtener el máximo rendimiento.

Tras esta introducción, el resto del artículo se estructura como sigue: la Sección II hace referencia a diferentes trabajos de la literatura relacionados con el tema abordado. La Sección III detalla la implementación del KNN propuesto para clasificación de EEGs. Luego, la Sección IV analiza los resultados experimentales y debate sobre la importancia de la conciencia energética en sistemas HPC. Finalmente, la Sección V proporciona las conclusiones.

II. Estado del arte

El uso de redes neuronales artificiales y técnicas de Machine Learning en bioinformática ha experimentado un crecimiento exponencial en los últimos años. Entre otras causas, se puede destacar el gran incremento de los conjuntos de datos biológicos, como en el caso de la Electroencefalografía. La bioinformática, entre otros temas, se ocupa de las señales EEG, las cuales representan la actividad eléctrica de diferentes partes del cerebro. Estas señales se utilizan para ayudar en el diagnóstico de trastornos como la esquizofrenia [2], la epilepsia [3], la dislexia [4], la depresión [5], el autismo [6] o en problemas del sueño [7]. También se utilizan para clasificar las funciones motoras del sistema nervioso, como el movimiento de las extremidades o los ojos [8], y para la clasificación de las emociones humanas [9]. Sin embargo, el principal problema de trabajar con señales EEG es su alta dimensionalidad, lo que dificulta su correcta clasificación ya que la mayoría de ellas no contienen información relevante. Por ello, es importante aplicar técnicas de selección de características para obtener las más relevantes. Uno de los objetivos fundamentales de las redes neuronales artificiales y del Machine Learning es reconocer patrones en estas señales para su posterior clasificación. De hecho, el problema de la clasificación de EEGs suele tratarse a través de distintos métodos de aprendizaje automático, como los algoritmos de regresión o agrupamiento (*clustering*).

[1]Dpto. de Lenguajes y Sistemas Informáticos, CITIC, Universidad de Granada (España). E-mail: jjescobar@ugr.es

[2]Dpto. de Ingeniería de Computadores, Automática y Robótica, CITIC, Universidad de Granada (España). E-mails: cazz@correo.ugr.es, {beap, aprieto, mdamas}@ugr.es

[3]Dpto. de Ingeniería de Comunicaciones, Universidad de Málaga (España). E-mail: aortiz@ic.uma.es

Algoritmo 1: Pseudocódigo utilizado por los nodos trabajadores para evaluar todos los posibles valores de K de un subconjunto de características.

```
1  Función evaluarSubset(Tr, Te, Idx)
       Entrada: Dataset de entrenamiento, Tr
       Entrada: Dataset de test, Te
       Entrada: Índice de la última columna del
                subconjunto de características a
                evaluar, Idx
       Salida : Precisión del mejor valor de K, acc
2      N_I ← getNumeroInstancias(Te)
       // Predicciones correctas para cada K
3      C_P ← {0}
       // Crear tantas hebras como núcleos lógicos
4      #pragma omp parallel for
5      para i ← 1 a N_I filas de test hacer
           /* Distancia entre la instancia te[i] y
              todas las de entrenamiento          */
6          D ← calcularDistancias(Te[i], Tr, Idx)
7          para k ← 1 a N_I hacer
8              P_C ← clasePredominante(k, D)
9              si la predicción P_C es correcta entonces
10                 C_P[k]++
11             fin
12         fin
13     fin
14     C_P ← ordenar(C_P, "Descendiente")
15     acc ← C_P[1] / N_I
16     devolver acc
17 Fin
```

Algoritmo 2: Pseudocódigo maestro-trabajador implementado en el enfoque propuesto.

```
1  Función main(Tr, Te, C_S, N_Wk)
       Entrada: Dataset de entrenamiento, Tr
       Entrada: Dataset de test, Te
       Entrada: Máximo número de características a
                enviar a los nodos trabajadores
                (tamaño del chunk), C_S
       Entrada: Número de nodos trabajadores, N_Wk
       Salida : Precisión del mejor subconjunto de
                características, bestAcc
2      bestAcc ← −1
3      si Maestro entonces
4          N_F ← getNumeroCaracteristicas(Tr)
5          indexList ← {1, ..., N_F}
6          stops ← 0
7          mientras stops ≠ N_Wk hacer
8              MPI::Recv(acc, tag)
9              si tag is RESULT y acc > bestAcc entonces
10                 bestAcc ← acc
11             fin
               /* Próximo chunk a enviar, e.g.
                  índices 10,11,12 cuando C_S = 3   */
12             chunk ← getProximoChunk(indexList, C_S)
13             si getTamano(chunk) > 0 entonces
14                 MPI::Send(chunk, JOB_DATA)
15             en otro caso
16                 MPI::Send(NULL, STOP)
17                 stops ← stops + 1
18             fin
19         fin
20     en otro caso
           /* mRMR. Reordenar los datasets para
              tener acceso coalescente a memoria  */
21         rankedFeatures ← mRMR(Tr)
22         Tr ← ordenarDataset(Tr, rankedFeatures)
23         Te ← ordenarDataset(Te, rankedFeatures)
           // Pedir carga de trabajo al maestro
24         MPI::Send(NULL, FIRST_JOB)
25         MPI::Recv(chunk, tag)
26         mientras tag sea JOB_DATA hacer
27             para i ← 1 a getTamano(chunk) hacer
28                 acc ← evaluarSubset(Tr, Te, chunk[i])
29                 si acc > bestAcc entonces
30                     bestAcc ← acc
31                 fin
32             fin
33             MPI::Send(bestAcc, RESULT)
34             MPI::Recv(chunk, tag)
35         fin
36     fin
37     devolver bestAcc
38 Fin
```

En este trabajo se ha considerado el algoritmo KNN para clasificación supervisada debido a su buen desempeño en este tipo de aplicaciones. Este algoritmo generalmente intenta clasificar las instancias (patrones) asignándolas a la clase predominante de entre sus K vecinos más cercanos. Los pasos necesarios para clasificar cada nueva instancia son:

1. Calcular la distancia entre la instancia a clasificar y todas las de entrenamiento. En este trabajo se ha utilizado la distancia euclídea para calcular las distancias.
2. Ordenar las distancias en orden creciente.
3. Identificar la clase predominante de entre las K distancias más cercanas (vecinos).
4. Asignar la nueva instancia a la clase predominante.

III. El enfoque propuesto

La clasificación de EEGs se ha abordado utilizando el algoritmo KNN y la técnica de mínima Redundancia Máxima Relevancia (mRMR) [10], la cual ordena las N_F características del conjunto de datos de menor a mayor relevancia. Este enfoque ayuda a tratar con el problema de la maldición de la dimensionalidad [11] y a reducir el tiempo de cómputo al evitar la evaluación de todos los 2^{N_F} posibles subconjuntos de características. En cambio, el algoritmo propuesto sólo evalúa N_F de ellos, donde en cada uno se agrega la siguiente característica de la lista proporcionada por mRMR. Además, para cada subconjunto también se evalúan todos los valores posibles del parámetro K para poder obtener la mejor precisión de clasificación. El mRMR implementado en este trabajo está basado en el indicado en [12] y toma el criterio del cociente de correlación de la prueba F (FCQ) para seleccionar la siguiente característica.

El algoritmo KNN se ha paralelizado distribuyendo subconjuntos de características entre los nodos trabajadores con MPI y distribuyendo las instancias de test a clasificar entre las hebras CPU mediante OpenMP. Esto último se puede ver en la directiva `#pragma omp parallel for` del Algoritmo 1 (Línea 4). La evaluación de todos los valores de K para una instancia de test ha sido optimizada almacenando su distancia con respecto a todas las instancias de entrenamiento (Línea 6). De esta forma, el vector D se reutiliza en el bucle de la Línea 7 para obtener la clase predominante según el valor de

Tabla I: Etiquetas MPI usadas durante las comunicaciones entre el maestro y los nodos trabajadores.

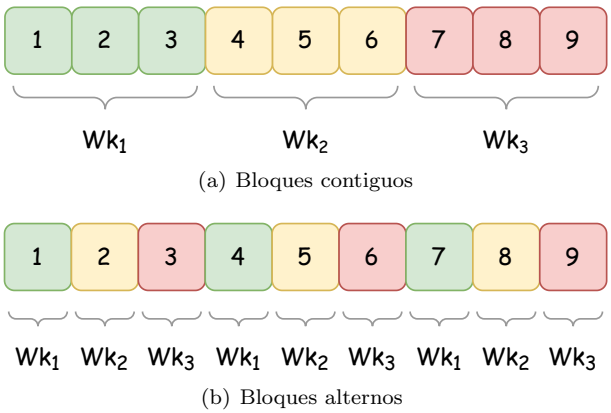
Etiqueta MPI	Descripción	Emisor	Receptor
`FIRST_JOB`	Pedir el primer trabajo	Trabajador	Maestro
`JOB_DATA`	Hay trabajo por hacer	Maestro	Trabajador
`RESULT`	Devolver el resultado del trabajo	Trabajador	Maestro
`STOP`	No hay más trabajo por hacer	Maestro	Trabajador

K (Línea 8). Si la predicción es correcta, el valor de la posición k-ésima del vector de predicción, C_P, se incrementa en uno. Una vez que se han clasificado todas las instancias, se ordena el vector de predicción. La primera posición corresponderá a la precisión de clasificación del mejor K (Líneas 14 y 15).

A. *Esquema maestro-trabajador distribuido con paralelismo a nivel de nodo*

La implementación propuesta, cuyo pseudocódigo puede verse en el Algoritmo 2, sigue un esquema maestro-trabajador donde el nodo maestro le dice a cada nodo trabajador qué subconjunto de características debe usar para evaluar un KNN. La ejecución finaliza cuando no hay más trabajo para procesar y por tanto la función devuelve la mejor precisión encontrada (Línea 37). El funcionamiento es el siguiente: el maestro espera en la Línea 8 a que algún trabajador solicite su primer trabajo, o le devuelva el resultado de uno de ellos, lo cual implica también la asignación de nuevo trabajo (*chunk*). El tipo de mensaje se identifica mediante las etiquetas MPI (*tags*) indicadas en la Tabla I. Cuando el maestro recibe un resultado, verifica si la precisión de ese trabajo (subconjunto de características) es mejor que la actual. Si es así, actualiza su valor (Líneas 9 a 11) y envía un nuevo *chunk* con la etiqueta `JOB_DATA` (Línea 14). Sin embargo, antes de enviar trabajo a un trabajador, el maestro verifica los *chunks* sin procesar. Si no hay disponibles, el trabajador recibirá la etiqueta `STOP` y detendrá su ejecución ya que no hay más trabajo por hacer (Línea 16). Como todos los trabajadores tienen que recibir la etiqueta para finalizar, el maestro debe registrar cuántos trabajadores la han recibido (Línea 17).

Con respecto a los nodos trabajadores (Líneas 20 a 36), en primer lugar aplican el algoritmo mRMR al conjunto de datos de entrenamiento para obtener la lista de características según su relevancia. Un trabajador solicita trabajos enviando al maestro un mensaje con la etiqueta `FIRST_JOB` (Línea 24). Para cada *chunk* de características recibidas (Líneas 27 a 32), el trabajador obtiene la precisión del subconjunto de características correspondiente llamando en la Línea 28 del Algoritmo 1 a la función `evaluarSubset`. Si la precisión del subconjunto de características procesado es mayor que la existente, se actualizará. Una vez que se han procesado todos los subconjuntos posibles del *chunk* recibido, el trabajador devuelve la mejor precisión obtenida al maestro mediante el envío de un mensaje con la etiqueta `RESULT`, y espera la asignación de nuevo trabajo (Líneas 33 y 34). Este proceso se repite hasta que el trabajador reciba la etiqueta `STOP`, indicando que ya puede finalizar su ejecución.



(a) Bloques contiguos

(b) Bloques alternos

Fig. 1: Las dos distribuciones estáticas de carga de trabajo utilizadas por el nodo maestro.

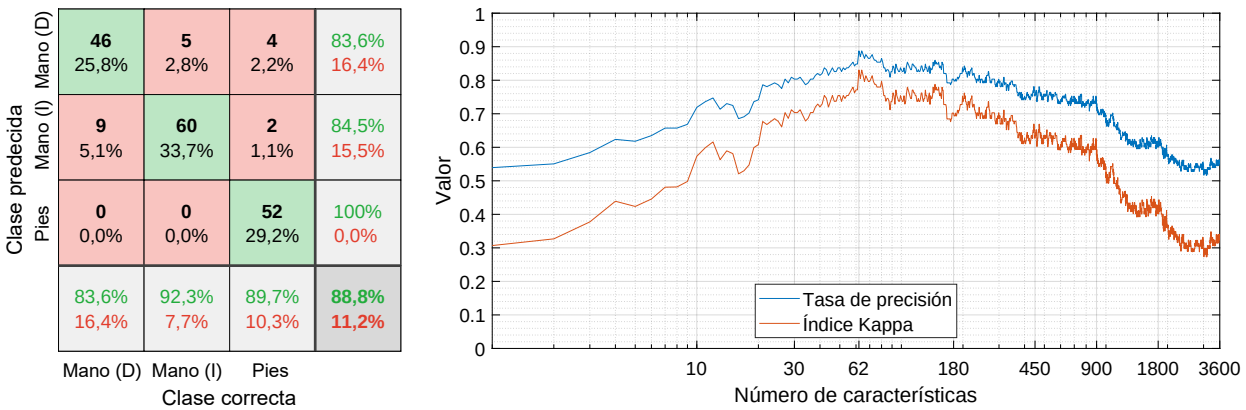
B. *Distribuciones de carga de trabajo*

Tal y como se ha visto anteriormente en la Sección III-A, los *chunks* de características se envían a los nodos trabajadores mediante paso de mensajes utilizando la librería MPI. Esto permite a la aplicación distribuir la carga de trabajo entre los diferentes nodos del clúster. Sin embargo, en el algoritmo propuesto aquí, la carga de trabajo de cada subconjunto de características es asimétrica ya que el número de características en cada uno de ellos es variable. Por ejemplo, supongamos un conjunto de datos con 10 características, dos nodos trabajadores y un tamaño de *chunk* de 2. En este escenario, el primer *chunk* que el maestro enviará contendrá los índices 1 y 2, correspondientes a los subconjuntos $\{1\}$ y $\{1, 2\}$. El segundo trabajador recibirá los índices 3 y 4 para calcular los subconjuntos $\{1, 2, 3\}$ y $\{1, 2, 3, 4\}$. En otras palabras, un índice más alto implica que el KNN debe computar más características y, en consecuencia, el tiempo de ejecución será mayor. Para lidiar con el desbalanceo de la carga de trabajo, de forma predeterminada, el procedimiento distribuye *chunks* de forma dinámica según sus tamaños. Aunque esto tiene la desventaja de aumentar las comunicaciones, es fundamental en sistemas heterogéneos para evitar caídas de rendimiento. Si el usuario lo desea, el maestro también puede dar a cada trabajador un *chunk* de características estático al comienzo del algoritmo dividiendo la cantidad total de *chunks* entre el número de trabajadores. Esto se puede hacer de dos maneras: repartiendo *chunks* con características contiguas

Tabla II: Características del clúster utilizado para la experimentación.

Nodo	CPU				RAM	
	Modelo	Total núcleos/hebras	TDP[a] (W)	Frecuencia (MHz)	Frecuencia (MHz)	Tamaño (GB)
Maestro	2x Intel Xeon E5-2620 v2	12/24	160	2,100	1,600	32
1	1x Intel Xeon E5-2620 v4	8/16	85		2,133	
2	2x Intel Xeon E5-2620 v4	16/32	170			
3 a 7	2x Intel Xeon Silver 4214	24/48	170	2,200	2,933	64

[a]El parámetro de Potencia de Diseño Térmico (TDP) indica la cantidad de calor que un chip genera en condiciones normales de operación, medido en Vatios (W), el cual se utiliza con frecuencia para estimar el consumo energético del chip.



(a) Matriz de confusion del mejor caso (62 características)

(b) Tasa de precisión e índice Kappa al añadir características

Fig. 2: Resultados de clasificación del método propuesto al utilizar mRMR and $K = 18$.

o alternas (ver la Figura 1). La asignación de características alternas podría reducir el desequilibrio de la carga de trabajo [13] presente en la distribución de características contiguas, ya que cada nodo computaría subconjuntos similares. El impacto que tiene en el rendimiento las diferentes distribuciones de carga de trabajo y el tamaño del *chunk* se analizará en la Sección IV.

IV. Resultados experimentales y discusión

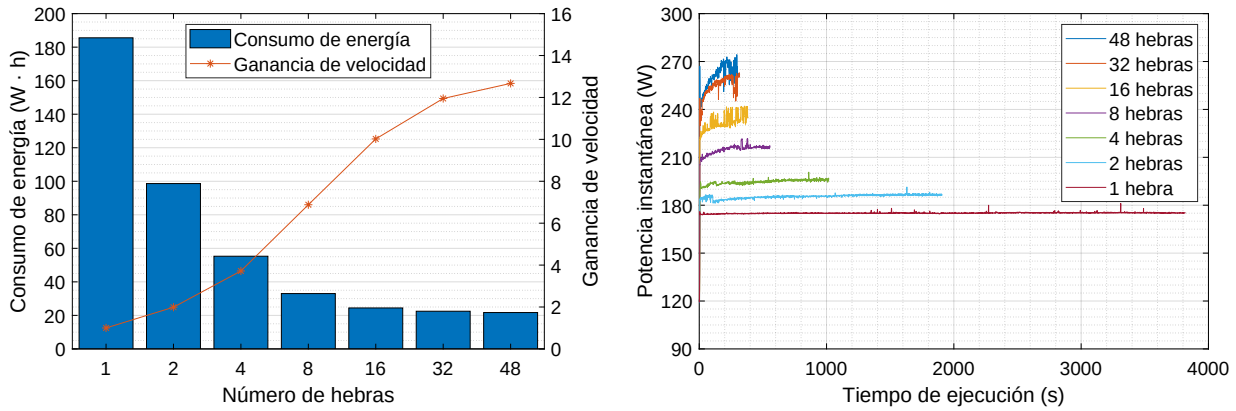
A. Setup experimental

Todos los experimentos se han repetido diez veces para obtener medidas más fiables del comportamiento de la aplicación, la cual ha sido ejecutada en un clúster HPC de 8 nodos NUMA heterogéneos y cuyos dispositivos CPU se detallan en la Tabla II. El clúster ejecuta la distribución *Rocky Linux* (v8.5) y planifica la ejecución de los trabajos mediante el administrador de tareas SLURM (v20.11.7) [14]. El código fuente *C++* ha sido compilado con el compilador GNU (GCC v8.5.0), la librería OpenMPI (v4.0.5) y las opciones de optimización `-O2 -funroll-loops`. Las medidas de energía de cada nodo se han obtenido con un vatímetro personalizado, denominado *Vampire*, capaz de capturar en tiempo real información de potencia instantánea (W) y energía acumulada consumida (W · h) para cada nodo de cómputo.

El conjunto de datos de EEGs pertenece al laboratorio BCI de la Universidad de Essex y corresponde a un sujeto humano codificado como #104 [15]. El conjunto de datos incluye 178 señales EEG para entrenamiento y otras 178 para test, cada una con 3.600 características. Como las señales pueden pertenecer a tres movimientos de imaginación motora diferentes (mover la mano izquierda, la mano derecha y los pies), el algoritmo KNN trata con un problema de clasificación de 3 clases.

B. Análisis del proceso de clasificación

El algoritmo propuesto logra un índice Kappa de 0,83 usando las primeras 62 características proporcionadas por mRMR y con $K = 18$. Esta solución supera ampliamente una ejecución sin usar mRMR (0,34) y otros enfoques en la literatura que utilizan el mismo conjunto de datos: [15] (0,790), [16] (0,754) y [17] (0,750). El resultado se ha validado al obtener los mismos resultados tras ejecutar el KNN con los



(a) Ganancia de velocidad y consumo de energía

(b) Potencia instantánea

Fig. 3: Rendimiento del enfoque propuesto al utilizar un único nodo de cómputo variando el número de hebras OpenMP.

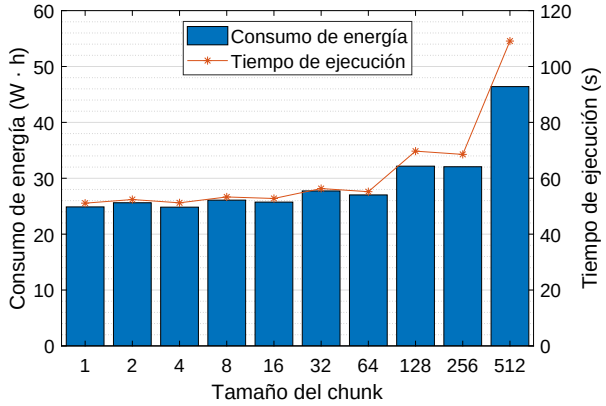
lenguajes Python y Matlab, y los mismos parámetros de entrada. La Figura 2(a) muestra la matriz de confusión correspondiente, la cual revela una tasa de precisión global del 88,8 %. La evolución de la tasa de precisión y el índice Kappa en función del número de características seleccionadas se puede observar en la Figura 2(b). La tendencia general es que ambas métricas aumentan a medida que se agregan nuevas características hasta alcanzar el pico máximo (62 características), y luego descienden progresivamente. Parece ser que la convergencia del algoritmo se ve penalizada al seleccionar muchas características, las cuales serían irrelevantes. También se observa que los valores de precisión y Kappa se distancian para valores extremos de la gráfica.

La validación estadística del resultado se ha llevado a cabo realizando un test de permutaciones con 10.000 iteraciones. Comprobando la hipótesis nula se puede asegurar que los resultados no han sido obtenidos por casualidad. En nuestro caso, la hipótesis nula consiste en obtener mayores valores del índice Kappa en el conjunto de datos permutado que en el no permutado. En la prueba se ha obtenido un p-valor $< 10^{-7}$, demostrando que se puede rechazar la hipótesis nula ya que su valor es menor a 0,01.

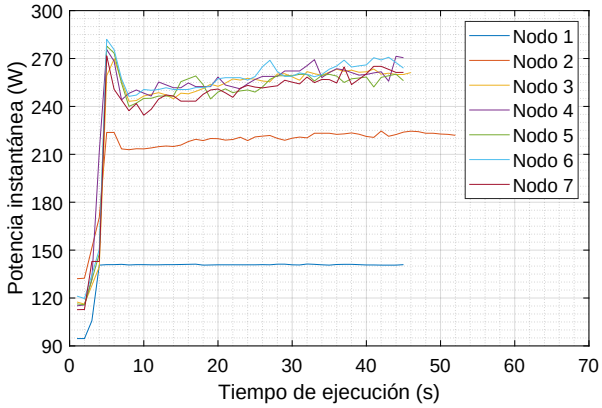
C. *Rendimiento energía-tiempo*

La Figura 3 muestra el rendimiento de la aplicación tras ser ejecutada sólo en el Nodo 3. El objetivo es mostrar la escalabilidad de la ganancia de velocidad del primer nivel de paralelismo, el cual ocurre dentro de cada nodo de cómputo al aumentar la cantidad de núcleos CPU lógicos. De la Figura 3(a), se puede ver que la ganancia máxima de 12,67 se obtiene usando las 48 hebras disponibles en el nodo. Su comportamiento es aproximadamente lineal, hasta cuatro hilos, y logarítmico para valores superiores. La razón principal es que la placa base puede acceder a memoria principal a través de cuatro canales de transmisión de datos. Aumentar el número de hebras por encima de cuatro provoca competencia por los accesos a la memoria, ya que no todos pueden hacerlo simultáneamente. También se debe, aunque en menor medida, a que la carga de trabajo de cada hebra disminuye y el coste de gestionarlas cobra mayor importancia. Esto significa que la ganancia de velocidad podría aumentar con conjuntos de datos más grandes que permitan que a las hebras computar durante periodos más largos. También se puede ver que distribuir las instancias a clasificar entre las hebras de forma estática proporciona el mejor rendimiento (Línea 4 del Algoritmo 1). Esto es de esperar porque la carga de trabajo es la misma para cada hebra, por lo que una distribución dinámica aquí no tendría cabida. En cuanto al consumo de energía, también para el caso de usar 48 hebras se obtiene el menor valor. Esto puede parecer contradictorio ya que el uso de más recursos está asociado con una mayor potencia instantánea (ver la Figura 3(b)). Sin embargo, el consumo de energía también depende linealmente del tiempo de ejecución y, dado que la ganancia aumenta a un ritmo mayor que la energía, el consumo total es menor.

El rendimiento del enfoque híbrido MPI-OpenMP correspondiente al segundo nivel de paralelismo se muestra en las Figuras 4 y 5. Por un lado, la Figura 4 expone el comportamiento de la aplicación cuando se utilizan todos los nodos y la distribución de la carga de trabajo es dinámica. La Figura 4(a) revela que un tamaño de fragmento muy grande conlleva peor tiempo de ejecución y consumo de energía, principalmente debido al desbalanceo de la carga de trabajo. La potencia instantánea de cada nodo para un tamaño de *chunk* de 4 se puede ver en la Figura 4(b). A pesar de que el tamaño óptimo va de 1 a 64, se ha fijado como definitivo el valor 4 ya que funciona bien con pocos nodos y debería hacerlo con más de 7. Lo que se observa en la figura es que la mayoría de nodos terminan simultáneamente, que es lo esperado en las distribuciones de carga de trabajo dinámicas. Por otro lado, la Figura 5 compara las diferentes distribuciones de carga de trabajo. El número de nodos de cómputo indicado en la Figura 5(a) no corresponde al orden mostrado en la Tabla II, sino que en la gráfica los nodos corresponden a los homogéneos y heterogéneos, en ese orden. Es decir, primero los No-

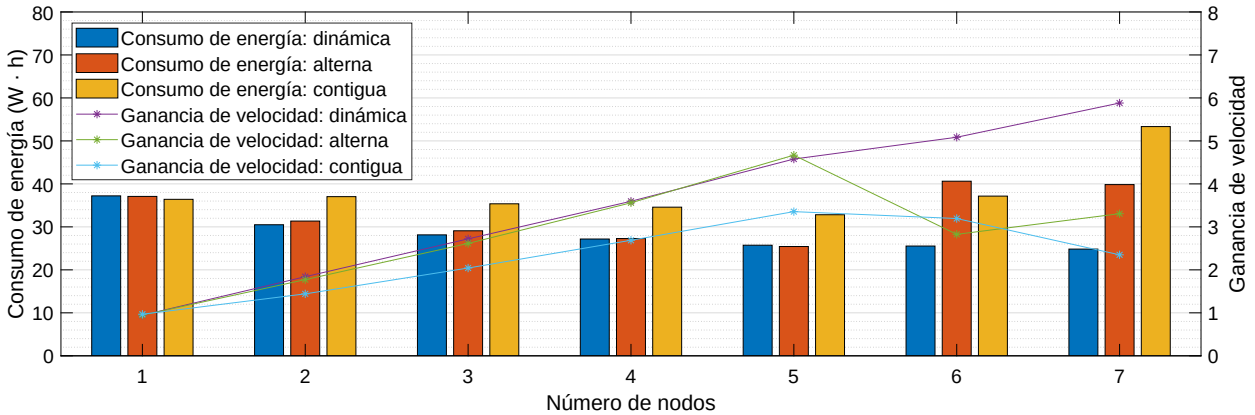


(a) Tiempo de ejecución y consumo de energía al incrementar el tamaño del *chunk*

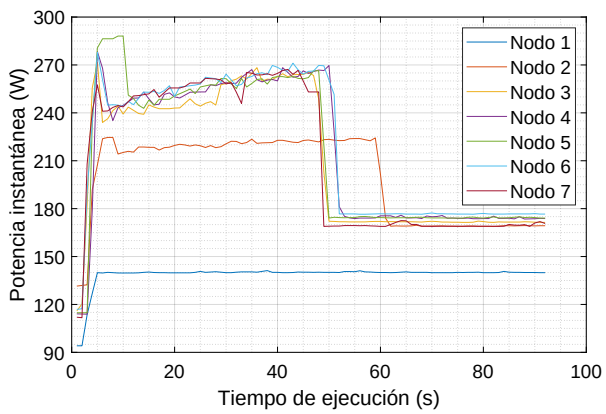


(b) Potencia instantánea del mejor tamaño de *chunk* (4)

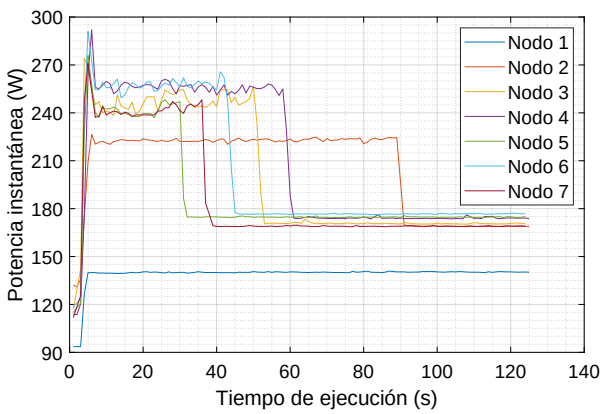
Fig. 4: Rendimiento de la distribución de carga de trabajo dinámica al usar todos los nodos.



(a) Ganancia de velocidad y consumo de energía al incrementar el número de nodos de cómputo



(b) Potencia instantánea de la distribución estática alterna cuando se utilizan todos los nodos



(c) Potencia instantánea de la distribución estática contigua cuando se utilizan todos los nodos

Fig. 5: Comparativa de rendimiento entre las diferentes distribuciones de carga de trabajo.

dos 3 a 7, y posteriormente los Nodos 1 y 2. De esta forma, se puede analizar la escalabilidad del programa según el tipo de nodo agregado. Como era de esperar, para todas las distribuciones, la ganancia de velocidad observada crece linealmente a medida que se utilizan más nodos, pero hasta 5 y en diferentes magnitudes. A partir de ese punto, sólo la distribución dinámica sigue escalando su rendimiento, aunque en menor medida ya que se empiezan a utilizar nodos heterogéneos. De hecho, se puede observar que el aumento de la ganancia va en consonancia con el nodo heterogéneo añadido: al añadir el Nodo 2 la velocidad aumenta más que al añadir el Nodo 1 (el más lento), hasta llegar a una ganancia máxima de 5,88. Con respecto a una ejecución secuencial (1 hebra), la aplicación consigue una ganancia de 74,53 consumiendo tan sólo un 13,38 % de energía. El uso de nodos heterogéneos también afecta negativamente a las diferentes distribuciones pero de diferentes maneras. En el caso de la distribución alterna, la ganancia

cae en picado con 6 nodos y mejora ligeramente después de agregar el último. No así para la distribución contigua, que empeora su rendimiento por cada nodo añadido. La potencia instantánea en la Figura 5(c) revela que el desbalanceo de la carga de trabajo es el responsable. Aquí, los nodos terminan de computar de forma escalonada, con un largo intervalo entre el primero ($t = 30$) y el último ($t = 125$). En el caso de la distribución alterna (Figura 5(b)), sólo los nodos homogéneos terminan al mismo tiempo, pero antes que los nodos heterogéneos, provocando un cuello de botella. En base a los resultados, se puede afirmar que la distribución dinámica proporciona los mejores resultados de ganancia, consumo de energía y escalabilidad ya que la ganancia de velocidad es muy cercana a la cantidad de nodos utilizados para computar.

V. Conclusiones

En este trabajo se ha propuesto investigar la eficiencia energética de una aplicación de bioingeniería capaz de explotar las cualidades de plataformas paralelas distribuidas y heterogéneas. El uso de mRMR para la selección de características ha permitido mejorar el rendimiento de los enfoques existentes en la literatura que utilizan el mismo conjunto de datos. Otra de las aportaciones de este artículo ha sido considerar la eficiencia energética como un parámetro fundamental, a diferencia de otros trabajos que se centran únicamente en la precisión de los resultados y en el tiempo de ejecución. Además, se han analizado diferentes distribuciones de carga de trabajo para el procedimiento propuesto. Los resultados han demostrado que una distribución dinámica es la opción más adecuada para distribuir trabajos asimétricos en sistemas heterogéneos. También han mostrado la importancia de conocer la arquitectura a la hora de escribir código paralelo para aprovechar todos los recursos disponibles, alcanzando una ganancia de velocidad de hasta 74,53 consumiendo tan sólo un 13,38 % de energía que una ejecución secuencial. Aun así, el siguiente paso es mejorar este resultado usando aceleradores como GPUs y aumentando el paralelismo de datos a través de técnicas de vectorización [18]. Otra forma de reducir el consumo de energía podría ser mediante el uso de una política de energía que detenga o reanude la ejecución del programa de acuerdo al coste por Megavatio. En consecuencia, esta política permitiría a los centros de datos ahorrar energía o dinero, según las preferencias del usuario, aunque en este último caso sería a costa de alargar el tiempo de ejecución y la energía consumida.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades y por fondos FEDER a través de los proyectos PGC2018-098813-B-C31 y PID2022-137461NB-C32.

Referencias

[1] Charlotte Freitag, Mike Berners-Lee, Kelly Widdicks, Bran Knowles, Gordon Blair, and Adrian Friday, "The climate impact of ict: A review of estimates, trends and regulations," *arXiv*, 2021.

[2] Hesam Akbari, Sedigheh Ghofrani, Pejman Zakalvand, and Muhammad Tariq Sadiq, "Schizophrenia recognition based on the phase space dynamic of EEG signals and graphical features," *Biomedical Signal Processing and Control*, vol. 69, 2021.

[3] Hemant Choubey and Alpana Pandey, "A combination of statistical parameters for the detection of epilepsy and EEG classification using ANN and KNN classifier," *Signal, Image and Video Processing*, vol. 15, no. 3, pp. 475–483, 2021.

[4] Ahmad Zuber Ahmad Zainuddin, Wahidah Mansor, Lee Yoot Khuan, and Zulkifli Mahmoodin, "Classification of EEG signal from capable dyslexic and normal children using KNN," *Advanced Science Letters*, vol. 24, no. 2, pp. 1402–1405, 2018.

[5] Maryam Saeedi, Abdolkarim Saeedi, and Arash Maghsoudi, "Major depressive disorder assessment via enhanced K-nearest neighbor method and EEG signals," *Physical and Engineering Sciences in Medicine*, vol. 43, no. 3, pp. 1007–1018, 2020.

[6] Sutrisno Ibrahim, Ridha Djemal, and Abdullah Alsuwailem, "Electroencephalography (EEG) signal processing for epilepsy and autism spectrum disorder diagnosis," *Biocybernetics and Biomedical Engineering*, vol. 38, no. 1, pp. 16–26, 2018.

[7] Hemant Sharma and K.K. Sharma, "An algorithm for sleep apnea detection from single-lead ECG using hermite basis functions," *Computers in Biology and Medicine*, vol. 77, pp. 116–124, 2016.

[8] Kadir Sabancı and Murat Koklu, "The classification of eye state by using kNN and MLP classification models according to the EEG signals," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 3, no. 4, pp. 127–130, 2015.

[9] Mi Li, Hongpei Xu, Xingwang Liu, and Shengfu Lu, "Emotion recognition from multichannel EEG signals using K-nearest neighbor classification," *Technology and Health Care*, vol. 26, no. S1, pp. 509–519, 2018.

[10] Insik Jo, Sangbum Lee, and Sejong Oh, "Improved measures of redundancy and relevance for mRMR feature selection," *Computers*, vol. 8, no. 2, pp. 42, 2019.

[11] Sarunas J. Raudys and Anil K. Jain, "Small sample size effects in statistical pattern recognition: Recommendations for practitioners," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 3, pp. 252–264, 1991.

[12] Chris Ding and Hanchuan Peng, "Minimum redundancy feature selection from microarray gene expression data," in *Computer Society Bioinformatics Conference*, Stanford, CA, USA, Aug. 2003, CSB'2003, pp. 523–528, IEEE.

[13] Fan Ding, Sandra Wienke, and Ruisheng Zhang, "Dynamic MPI parallel task scheduling based on a master-worker pattern in cloud computing," *International Journal of Autonomous and Adaptive Communications Systems*, vol. 8, no. 4, pp. 424–438, 2015.

[14] Nataliia Gvozdetska, Larysa Globa, and Volodymyr Prokopets, "Energy-efficient backfill-based scheduling approach for SLURM resource manager," in *15th International Conference on the Experience of Designing and Application of CAD Systems*, Polyana, Ukraine, Feb. 2019, CADSM'2019, pp. 1–5, IEEE.

[15] J. Asensio-Cubero, J. Q. Gan, and R. Palaniappan, "Multiresolution analysis over simple graphs for brain computer interfaces," *Journal of Neural Engineering*, vol. 10, no. 4, pp. 21–26, 2013.

[16] Javier León, Juan José Escobar, Andrés Ortiz, Julio Ortega, Jesús González, Pedro Martín-Smith, John Q. Gan, and Miguel Damas, "Deep learning for eeg-based motor imagery classification: Accuracy-cost trade-off," *PLoS ONE*, vol. 15, no. 6, 2020.

[17] J. Ortega, A. Ortiz Pedro Martín-Smith, J. Q. Gan, and J. González, "Deep belief networks and multiobjective feature selection for BCI with multiresolution analysis," in *14th International Work-Conference on Artificial Neural Networks*, Cádiz, Spain, June 2017, IWANN'2017, pp. 28–39, Springer.

[18] Mahmoud Hassaballah, Saleh Omran, and Youssef B. Mahdy, "A review of SIMD multimedia extensions and their usage in scientific and engineering applications," *The Computer Journal*, vol. 51, no. 6, pp. 630–649, 2008.

Estudio sobre la redundancia a nivel de bit y el impacto de Bit Flips en las Redes Neuronales Convolucionales

Izan Catalán Gallach[1], José Flich Cardo[1] y Carles Hernández Luz[1]

Resumen— **Las redes neuronales convolucionales se están utilizando masivamente en entornos críticos como la salud, los vehículos autónomos o la videovigilancia. Por tanto, es esencial validar su correcto funcionamiento para proporcionar seguridad a los sistemas que las utilizan. En este trabajo analizamos el comportamiento de varios modelos de Redes Neuronales Convolucionales (CNNs) en la presencia de fallos. Para ello, estudiamos y analizamos la sensibilidad de estos modelos e identificamos el impacto de los cambios de bit en su precisión así como el origen de los mismos.**

Palabras clave— **Redes Neuronales, Convoluciones, Invariantes**

I. Introducción

EN los últimos años, las redes neuronales convolucionales (CNN) [1] han sido ampliamente utilizadas para el tratamiento de imágenes [2], la conducción autónoma [3] u otros campos debido a su gran rendimiento a la hora de procesar imágenes [4].

En estos sistemas se utilizan diferentes estrategias para reducir el tiempo de ejecución (inferencia), como técnicas de cuantización, *tailoring*[5], o *sparsity*[6]. La arquitectura de una CNN incluye principalmente capas de convolución, *pooling* y *fully connected*.

La capa de convolución aplica una operación de convolución a la imagen de entrada. Para ello, normalmente se utilizan conjuntos de filtros $KH \times KW$. La capa de *pooling* realiza un muestreo descendente de las salidas producidas por la capa de convolución, también llamadas *feature maps*, y la capa *fully connected* realiza la clasificación asignando las *feature maps* a las clases de salida.

Las CNN se han propuesto como una alternativa en muchos sistemas críticos en los que los procesos de detección, segmentación y clasificación de imágenes son vitales. Por tanto, resulta necesario proteger estos sistemas en presencia de fallos.

Una vez que un modelo CNN se ha entrenado y optimizado satisfactoriamente, se utiliza en producción. En esta fase, su rendimiento (precisión) puede variar debido al impacto de los fallos. Los fallos pueden estar relacionados con situaciones ambientales, como la corrupción de la memoria inducida por radiaciones externas, o por fallos internos debido al proceso de fabricación o la degradación por el uso. Los fallos pueden afectar a los parámetros de las CNN (fallos en los pesos) o a los estados intermedios (fallos en las neuronas).

Por tanto, la calidad de un modelo CNN puede degradarse significativamente debido a la presencia de fallos [7] [8] que provoquen predicciones erróneas (por ejemplo, en las GPU [9]). Esto podría tener consecuencias catastróficas, como confundir un animal con un coche [10][11][12]. Para evitar estas situaciones, las normas de seguridad funcional (por ejemplo, ISO26262[13]) definen tasas de fallos específicos (por ejemplo, 10 FIT). Estas situaciones demuestran lo importante que es proteger dichas redes neuronales para evitar estos eventos de fallo.

En las redes neuronales convolucionales hay presencia también de redundancia a nivel de bit. Esto quiere decir que existen valores dentro de la operación de convolución con los mismos bits en las mismas posiciones. Esto es algo lógico dado que por ejemplo, los pesos de la operación de convolución están organizados dentro del umbral [-1,1] en modelos de coma flotante de 32 bits (FP32), por lo que comparten cierta parte decimal y entera. Es por tanto un objetivo añadido de este estudio verificar si existe o no esta redundancia a nivel de bit entre los pesos, con el fin de desarrollar mecanismos de tolerancia a fallos que se aprovechen de dicha redundancia.

Por otro lado, se analiza la robustez de las CNN en presencia de fallos. Para ello, se realizan inyecciones de fallos de bit flip en varios modelos de CNN ampliamente utilizados. Para estos modelos, también se analiza el impacto de los fallos cuando se utiliza precisión reducida, como INT8 (enteros de 8 bits). Los resultados del análisis muestran que la precisión Top1 de los modelos CNN FP32 disminuye, de media, del 1,3 % a más del 3 % cuando se inyecta un solo bit flip.

El resto del documento se organiza como sigue. En la sección II se revisan los trabajos e investigaciones sobre mecanismos de protección de las CNN. La sección III describe un análisis bit-a-bit sobre los modelos CNN en estudio. La sección IV incluye los experimentos realizados y los resultados que hemos obtenido y la sección V muestra las conclusiones del trabajo.

II. Estudios de Protección sobre las CNN

En los últimos años se han propuesto una amplia gama de soluciones para mejorar la robustez de las redes neuronales en presencia de fallos. Algunos trabajos se centran en soluciones para la detección de errores, mientras que otros se centran en soluciones

[1]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: izcagal@inf.upv.es, jflich@disca.upv.es y carherlu@upv.es

de corrección de errores o ambas a la vez.

La solución más inmediata para proteger las CNN es utilizar redundancia [14] (modular doble o triple). Sin embargo, replicar el modelo y/o el hardware subyacente es costoso, por lo que en su lugar se pueden proteger componentes de hardware específicos [15]. Proteger las memorias también es una opción interesante, aplicándose para ello varios códigos de corrección de errores (ECC) con diferente coste computacional y de área [16]. Otros investigadores también proponen o analizan la resiliencia a errores de los aceleradores de redes neuronales a nivel de transistor [17], con una memoria 3D *die-stacked* basada en fallos en CNNs [18] o con la adición de circuitos redundantes para reforzar los *latches* basados en el análisis de la resiliencia de los datos [19].

Alternativamente, los modelos de CNN pueden protegerse a nivel de software y en tiempo de ejecución. En [20], se muestra cómo añadir ciertas restricciones al proceso de entrenamiento puede aumentar la resiliencia de un modelo CNN. Por otro lado, en [21], los autores proponen reconstruir los pesos de un modelo localmente mientras se realiza la inferencia. En estos métodos, aunque resulta más difícil predecir erróneamente una imagen con un *bit flip* cuando se añaden estas restricciones, la precisión de la red neuronal se reduce, y puede que no se detecte el fallo.

Otros trabajos incluyen mecanismos de aprendizaje automático para detectar fallos [22]. En estos casos se entrena una red neuronal 'B' más pequeña que la original 'A' para verificar la salida de la original. Por tanto, si la salida de A no es igual a la de B, el resultado se invalida. Como desventaja, este mecanismo requiere entrenar y crear una segunda red neuronal y, en segundo lugar, asumir que se pierde precisión, ya que si B es más pequeña que A, su precisión será menor.

Otros autores [23] también han investigado la protección de un conjunto de pesos utilizando *checksums*. La eficacia de este mecanismo varía en función del grupo y del tamaño de los pesos a proteger. Cuanto mayor es el grupo de pesos, mayor es el nivel de detección y prevención. Sin embargo, la tasa de falsos negativos aumenta en grupos con muchos valores [23]. Del mismo modo, se han aplicado funciones *hash* a determinadas capas[24]. Con este mecanismo, se generan códigos *hash* para capas específicas con el fin de comparar las salidas reales con su referencia libre de fallos para detectar errores. Sin embargo, los fallos no se corrigen.

Un enfoque basado en umbrales es el presentado en [25]. Los autores implementan un mecanismo de protección utilizando los rangos de los pesos de un modelo CNN. Estas protecciones se determinan limitando el rango de valores y comprobando que ninguno supera un determinado umbral. Esto limita el efecto de un cambio de bit sin causar una sobrecarga excesiva en términos de tiempo y coste de implementación.

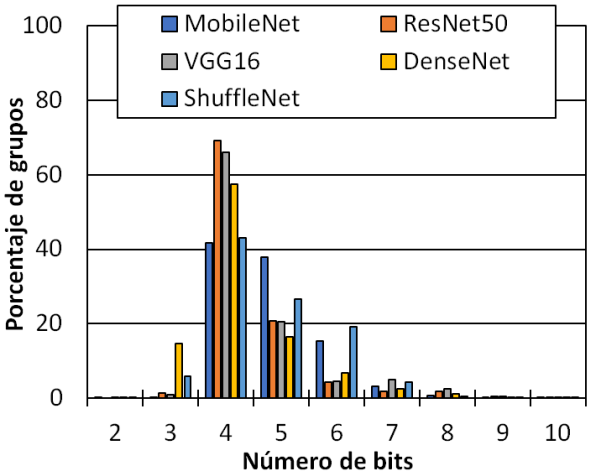


Fig. 1: Número de bits invariantes por grupos (FP32)

III. Análisis bit-a-bit de modelos de Redes Neuronales Convolucionales

En esta sección se analizan a nivel de bit los modelos que posteriormente se van a usar para las inyecciones de fallos, concretamente los pesos de los mismos y la existencia de algún patrón entre ellos. La idea de base es que al ser todos números decimales, una parte de los bits, concretamente en el exponente (la parte más crítica de un modelo en FP32), pueden repetirse.

En este estudio analizamos los modelos MobileNet, ResNet50, VGG16, DenseNet y ShuffleNet tanto utilizando pesos FP32 como en versiones cuantizadas utilizando INT8 (véase la tabla I para las especificaciones). Todos estos modelos se han descargado del repositorio ONNX Model Zoo [26]. Para todos los modelos, se agrupan todos los pesos de las capas de convolución en conjuntos de 9 pesos. Para capas con filtros de 3×3 , todos los pesos de un filtro pertenecen al mismo grupo. Por otro lado, para capas con filtros de 1×1 se agrupan en grupos de nueve filtros.

Para cada grupo se analizan los bits que tienen el mismo valor en todos los elementos en la misma posición. Se analiza todo el rango de bits para los pesos (32 bits para FP32 y 8 bits para INT8). Obsérvese que el bit de signo, el exponente y los bits de la mantisa se incluyen en el estudio.

A continuación, se muestran los resultados promediados obtenidos. Lo primero que se puede observar es que en los modelos FP32 (Figura 1), la mayor parte de la redundancia de bits se encuentra en grupos con 4-6 bits redundantes. Por ejemplo, el modelo ResNet50 tiene hasta un 70 % de sus grupos con 4 bits redundantes. Otro aspecto interesante es la posición en la que los bits redundantes son más frecuentes. En la Figura 2 se puede ver que casi toda la redundancia se encuentra entre los bits 27-30, que pertenecen a los bits más significativos del exponente. Esto significa que la mitad de los bits del exponente, de media, tienen el mismo valor en la mayoría de los grupos. Obsérvese también, que esos bits son los que más contribuyen al valor final del peso.

Tabla I: Especificaciones de los Modelos de Redes Neuronales Convolucionales.

Model	Data Type	Top 1 Acc %	Top 5 Acc %	Data Type	Top 1 Acc %	Top 5 Acc %
MobileNet v2-1.0	FP32	69.45	89.23	INT8	62.43	84.54
ResNet-50 v2-1.0	FP32	75.01	92.38	INT8	69.00	88.63
VGG-16	FP32	72.38	91.02	INT8	69.30	88.83
DenseNet-121-12	FP32	60.94	84.49	INT8	59.92	83.64
ShuffleNet-v2	FP32	66.38	86.58	INT8	58.60	80.40

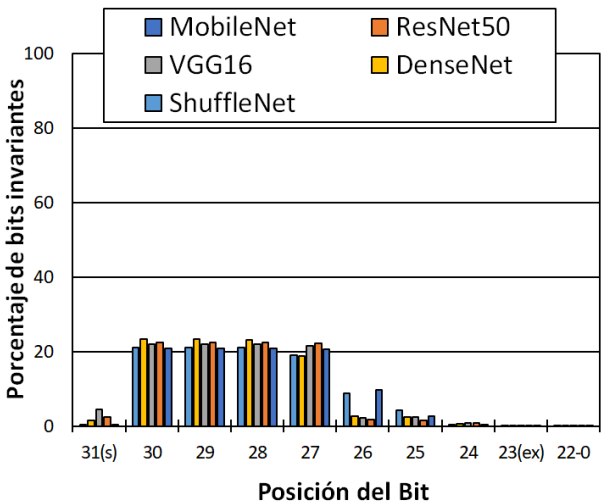


Fig. 2: Bits invariantes por posición (FP32)

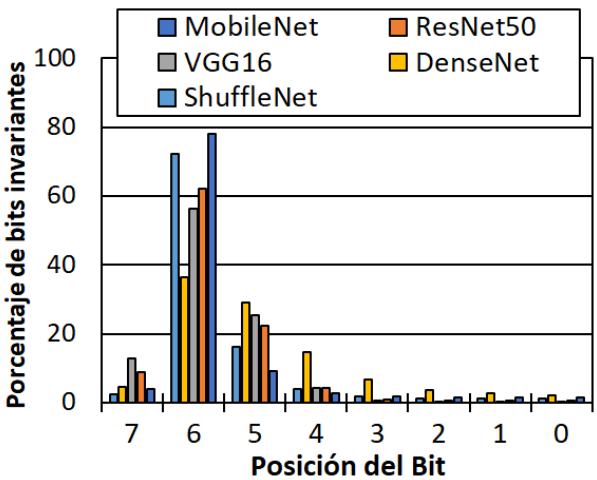


Fig. 4: Bits invariantes por posición (INT8)

Para los modelos INT8, sin embargo, la redundancia encontrada es menor. En la Figura 3 se puede ver que, dependiendo del modelo, la redundancia de bits varía. Por ejemplo, MobileNet tiene la mayoría de sus grupos (55 %) sin ninguna redundancia de bits. Sin embargo, el resto de modelos presentan una redundancia de bits moderada, con 1, 2 y 3 bits redundantes. En la Figura 4 se muestra la posición del bit más redundante. Los bits 4-6 deberían ser protegidos, ya que tienen hasta un 80 % de redundancia en algunos modelos y al igual que con FP32, contribuyen con más importancia a si el valor del peso es mayor o menor.

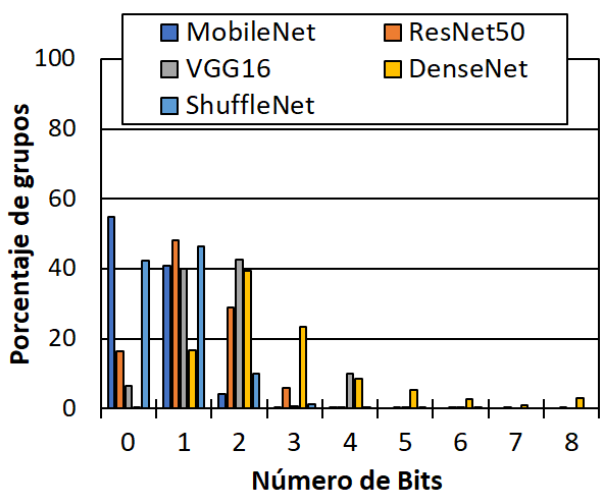


Fig. 3: Número de bits invariantes por grupos (INT8)

IV. Resultados experimentales

En esta sección se analizamos el impacto de los *bit flips* en el rendimiento de los modelos. Para ello, se inyecta un *bit flip* aleatorio en el conjunto de filtros convolucionales y, a continuación, se analiza su impacto en el modelo en términos de precisión (utilizando el conjunto de imágenes de pruebas completo de ImageNet[27]). Una vez completado, restauramos el bit e inyectamos un nuevo *bit flip* aleatorio obteniendo de nuevo la precisión del modelo sobre el conjunto de prueba. De este modo, los cambios de bit no se acumulan. Se realizan los experimentos hasta alcanzar un intervalo de confianza del 99 % con un margen de error del 5 %.

Los experimentos se han llevado a cabo utilizando el motor de inferencia Onnx Runtime-GPU (versión 1.12.0) [28], aplicando la librería de python MxNet [29] para obtneder los resultados de precisión de los modelos. Hemos utilizado un equipo con un procesador AMD EPYC 7282 16-Core y una tarjeta gráfica NVIDIA A100-40GB.

Para inyectar un bit, se selecciona al azar una capa de convolución del modelo. A continuación, se elige aleatoriamente un elemento del tensor de pesos de la capa y se inyecta un cambio de bit aleatorio. Se inyectan fallos sólo en las capas convolucionales, ya que son el objeto del estudio, y de forma similar a la metodología seguida en [25], donde se utiliza un *framework* para inyectar bits sólo en las capas convolucionales. Además, como se muestra en [30], las neuronas de una CNN son 50 veces más resistentes que sus pesos comprobado con el *framework* Ares. [31].

Analizamos tres métricas para evaluar el rendimiento de los modelos antes y después de inyectar

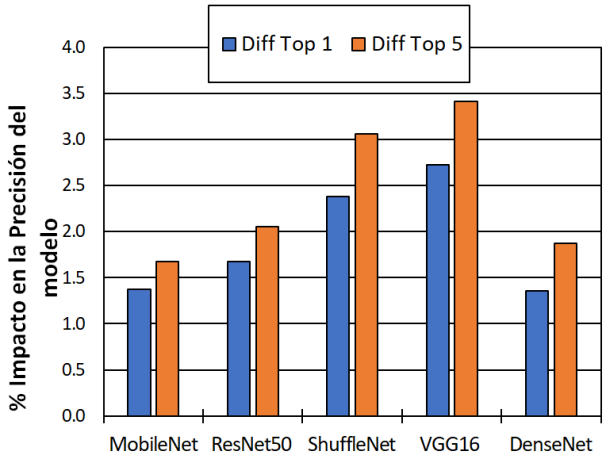


Fig. 5: Diferencia entre la precisión del modelo sin y con fallo con tipo de datos FP32 (mayor diferencia peor, dado que se pierde precisión)

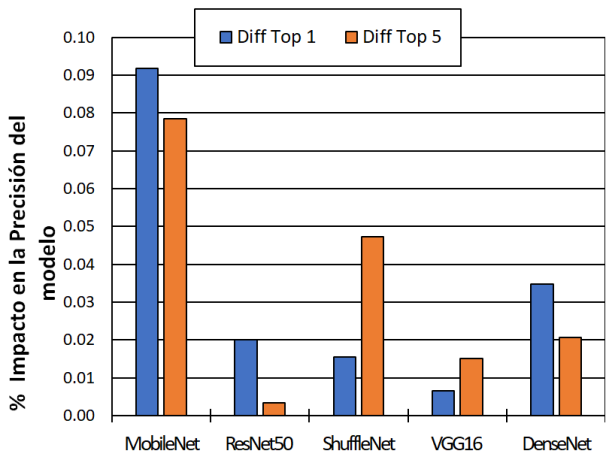


Fig. 6: Diferencia entre la precisión del modelo sin y con fallo con tipo de datos INT8 (mayor diferencia peor, dado que se pierde precisión)

un *bit flip*. En primer lugar, se muestra el impacto del cambio de bits en las métricas de precisión Top 1 y Top 5 al utilizar el conjunto de pruebas ImageNet al completo. El Top 1 es el porcentaje de imágenes correctamente detectadas, mientras que el Top 5 es el porcentaje de imágenes correctamente detectadas entre las cinco predicciones más probables. En segundo lugar, se muestra la sensibilidad del modelo a los cambios de bits. En concreto, se analiza los porcentajes de imágenes en las que el modelo cambia su predicción debido a un cambio de bit, tanto correcta como erróneamente. En la tercera métrica se analiza el porcentaje de fallos que producen un error en la salida (SDC) encontrado en los modelos. Es decir, el SDC representa el número de imágenes cuya predicción es solamente la errónea debido al cambio de bit.

Las Figuras 5 y 6 muestran el impacto de los cambios de bits en la precisión de los modelos Top 1 y Top 5. Este impacto es el porcentaje de la precisión original del modelo para Top 1 y 5 que disminuye una vez se inyecta el fallo. Se puede ver primero en la Figura 5, como los modelos FP32 sufren, de media, una degradación del 1,35 % (DenseNet) al 2,72 % (VGG-16) de su precisión Top 1 y una degradación del 1,67 % (Mobilenet) al 3,05 % (VGG-16) de su precisión Top 5. El impacto en los modelos INT8 es significativamente menor, alcanzando un impacto máximo del 0,092 % para Top 1 (Mobilenet) y del 0,078 % (también Mobilenet) para Top 5, como muestra la Figura 6.

Obsérvese que estos resultados mostrados en las Figuras 5 y 6 son la media de todos los cambios de bits inyectados. Se han visto casos en los que la precisión del Top 1, con un bit flip en el exponente, cambia completamente su predicción, aumentando el valor original del peso afectado por el fallo en 10^{37} y la precisión final cayendo a 0,01 % en FP32. Por ejemplo, para VGG16 (el modelo más afectado por SDC Ratio), el impacto máximo da una precisión total Top 1 de 0,00088 % y una precisión total Top5 de 0,00494 % (una disminución de 72,37 % y 91,01 % respectivamente) para FP32. Para INT8, el impacto máximo en VGG16 da una precisión total Top1 de 69,001 % y una precisión total Top5 de 88,66 % (una disminución de 0,3 % y 0,17 %, respectivamente). En MobileNet y ShuffleNet hay un mayor margen de impacto del *bit flip* en el rendimiento del modelo.

Con los modelos INT8 (véase la Figura 6), la conclusión es que las inyecciones de fallos no tienen un impacto tan grande como en FP32. Aunque se pueden proteger los bits más significativos, un *bit flip* no se traduce en valores enteros mucho mayores que podrían producir algún error de predicción (el rango va de -128 a 128). Los resultados muestran cómo, estadísticamente, no se observa ningún patrón.

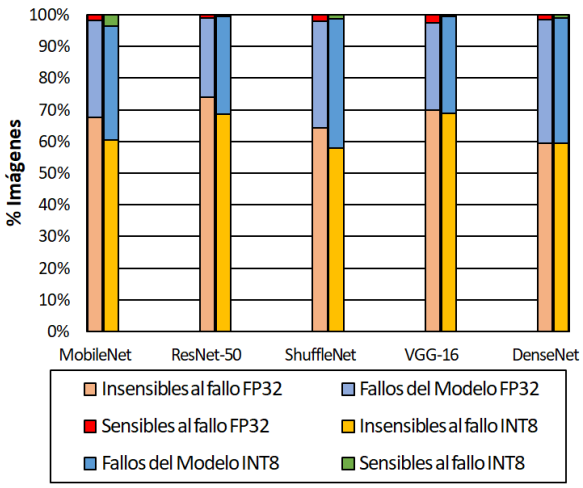


Fig. 7: Predicciones Imagen por Imagen para todo el *dataset* Imagenet

La segunda métrica consiste en medir como varían las predicciones sensibles a fallos (tanto cuando un fallo provoca una predicción errónea como correcta). Para ello, tenemos que fijarnos en la Figura 7 y centrarnos en el color rojo. Aquí comprobamos las predicciones correctas o la precisión general del modelo (predicciones insensibles a fallos), las predicciones erróneas siempre (fallos del modelo) y las predicciones sensibles a fallos (las que varían una vez se inyecta el fallo). Cuanto mayor sea el porcentaje de predicciones sensibles a fallos, menor será la precisión

obtenida en los modelos.

En los modelos FP32, se observa que todos los modelos tienen un mayor porcentaje de predicciones sensibles y menos predicciones correctas. Por otro lado, los modelos INT8, como hemos visto antes, mantienen un porcentaje similar de predicciones sensibles a los fallos.

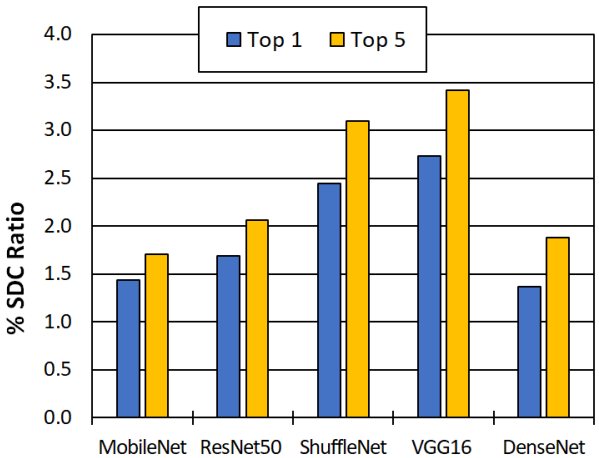


Fig. 8: Ratio de SDCs por modelo con tipo de dato FP32

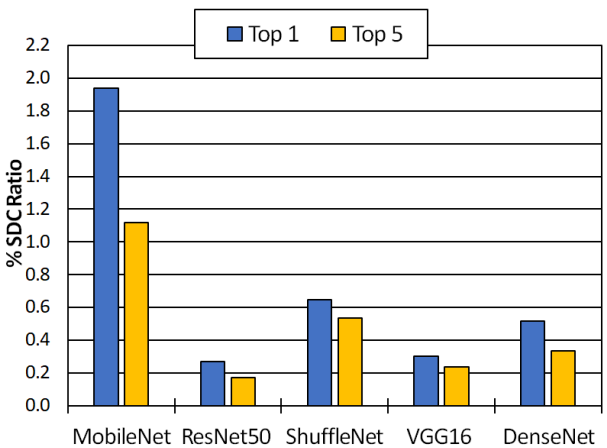


Fig. 9: Ratio de SDCs por modelo con tipo de Dato INT8

Por último, analizamos con más detalle las predicciones sensibles a fallos. Centrándonos únicamente es las que se predicen erróneamente (el Ratio de SDCs), la tercera métrica. Se trata de la proporción de predicciones de imágenes (de las 50.000 imágenes del conjunto de datos Imagenet) que se clasifican erróneamente cuando se produce un fallo y que antes se clasificaban correctamente. Observamos que el Ratio SDC es mayor en los modelos FP32 no protegidos (Figura 8 en escala logarítmica). Esto se debe a que un fallo provoca una pérdida de precisión. Todos los modelos tienen más de un 1 % de las imágenes del conjunto de datos que cambian la predicción y fallan, lo que hace que el modelo pierda precisión.

En los modelos INT8 (Figura 9), se puede ver las mismas tendencias que en los gráficos anteriores. La relación SDC es similar, un bit flip es mucho menos crítico en los enteros de 8 bits que en los modelos de coma flotante. Dado que los enteros de 8 bits no tienen un rango tan amplio de números ([-128,128] frente al rango de 32 bits de coma flotante [-3,4E+38 a +3,4E+ 38]), es mucho más probable y plausible que un *bit flip* en el exponente de un número FP32 provoque un cambio a un número drásticamente superior o inferior y por consiguiente, cambie la predicción a errónea.

V. Conclusiones

En este artículo se muestra un amplio estudio dividido en dos partes. Primeramente, se ha llevado a cabo una búsqueda de bits invariantes dentro de los pesos en las operaciones de convolución según nuestra hipótesis inicial. Se ha demostrado por tanto que efectivamente, el hecho e que los pesos de la convolución estén en un mismo rango ya sea [-1,1] con coma flotante de 32 bits FP32 o [-128,128] en enteros hace que los bits de mayor peso coincidan y por tanto sean un objetivo a proteger por nuestra parte en futuros trabajos.

En segundo lugar se ha realizando un análisis del impacto de bit flips en una variedad de modelos de redes neuronales convolucionales con números de coma flotante de 32 bits y enteros de 8 bits. Nuestros resultados muestran que, como era de esperar, los modelos de coma flotante son mucho más sensibles que los de enteros de 8 bits y que su punto crítico es el exponente. Proteger este rango de bits, como hemos demostrado, reduciría significativamente el impacto de los cambios de bit y sería razonable, dado que se podría aprovechar su mayor invariabilidad.

Agradecimientos

Este artículo ha sido realizado gracias a la dotación de la BECA FPU "PROGRAMA PROPIO DE LA UNIVERSITAT POLITÈCNICA DE VALÈNCIA – SUBPROGRAMA 1 (PAID-01-20)"No 20210037

Referencias

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6, DOI: 10.1109/ICEngTechnol.2017.8308186.

[2] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[3] Akhil Agnihotri, Prathamesh Saraf, and Kriti Rajesh Bapnad, "A convolutional neural network approach towards self-driving cars," *CoRR*, vol. abs/1909.03854, 2019.

[4] A. Vasuki and S. Govindaraju, *Deep neural networks for image classification*, pp. 27–49, 12 2017.

[5] Maria Bauza Nurullah Giray Kuru Tomás Lozano-Pérez Ferran Alet, Kenji Kawaguchi and Leslie Pack Kaelbling, "Tailoring: encoding inductive biases by optimizing unsupervised objectives at prediction time," 2018, `https://meta-learn.github.io/2020/papers/60_paper.pdf`.

[6] "How sparsity adds umph to ai inference," `https://blogs.nvidia.com/blog/2020/05/14/sparsity-ai-inference/`, 2021.

[7] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan, "Bit-flip attack: Crushing neural network withprogressive bit search," *CoRR*, vol. abs/1903.12269, 2019.

[8] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," *CoRR*, vol. abs/1906.01017, 2019.

[9] Daniel Alfonso Gonçalves Gonçalves de Oliveira, Laercio Lima Pilla, Thiago Santini, and Paolo Rech, "Evaluation and mitigation of radiation-induced soft errors in graphics processing units," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 791–804, 2016.

[10] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," New York, NY, USA, 2017, Association for Computing Machinery.

[11] Le Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," *CoRR*, vol. abs/1912.00941, 2019.

[12] Zitao Chen, Guanpeng Li, and Karthik Pattabiraman, "Ranger: Boosting error resilience of deep neural networks through range restriction," *CoRR*, vol. abs/2003.13874, 2020.

[13] Rick Salay, Rodrigo Queiroz, and Krzysztof Czarnecki, "An analysis of ISO 26262: Using machine learning safely in automotive software," *CoRR*, vol. abs/1709.02435, 2017.

[14] Wenshuo Li, Guangjun Ge, Kaiyuan Guo, Xiaoming Chen, Qi Wei, Zhen Gao, Yu Wang, and Huazhong Yang, "Soft error mitigation for deep convolution neural network on fpga accelerators," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020, pp. 1–5.

[15] Muhammad Hanif and Muhammad Shafique, "Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, pp. 20190164, 02 2020.

[16] Adam Neale and Manoj Sachdev, "Neutron radiation induced soft error rates for an adjacent-ecc protected sram in 28 nm cmos," *IEEE Transactions on Nuclear Science*, vol. 63, no. 3, pp. 1912–1917, 2016.

[17] Olivier Temam, "A defect-tolerant accelerator for emerging high-performance applications," vol. 40, no. 3, 2012.

[18] Jae-San Kim and Joon-Sung Yang, "Dris-3: Deep neural network reliability improvement scheme in 3d die-stacked memory based on fault analysis," 06 2019, pp. 1–6.

[19] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," New York, NY, USA, 2017, Association for Computing Machinery.

[20] Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan, "Defending and harnessing the bit-flip based adversarial weight attack," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 14083–14091.

[21] Jingtao Li, Adnan Siraj Rakin, Yan Xiong, Liangliang Chang, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti, "Defending bit-flip attack through dnn weight reconstruction," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[22] Yu Li, Min Li, Bo Luo, Ye Tian, and Qiang Xu, "Deepdyve: Dynamic verification for deep neural networks," *CoRR*, vol. abs/2009.09663, 2020.

[23] Jingtao Li, Adnan Siraj Rakin, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti, "RADAR: run-time adversarial weight attack detection and accuracy recovery," *CoRR*, vol. abs/2101.08254, 2021.

[24] Mojan Javaheripi and Farinaz Koushanfar, "HASHTAG: hash signatures for online detection of fault-injection attacks on deep neural networks," *CoRR*, vol. abs/2111.01932, 2021.

[25] Florian Geissler, Syed Qutub, Sayanta Roychowdhury, Ali Asgari, Yang Peng, Akash Dhamasia, Ralf Graefe, Karthik Pattabiraman, and Michael Paulitsch, "Towards a safety case for hardware fault tolerance in convolutional neural networks using activation range supervision," *CoRR*, vol. abs/2108.07019, 2021.

[26] Onnx, "Model zoo github repository," 2023.

[27] ImageNet, "Data set," 2023.

[28] Onnx Runtime, "Cross-platform inference and training machine-learning accelerator," 2023.

[29] Apache MxNet, "Library for deep learning," 2023.

[30] Syed Qutub, Florian Geissler, Yang Peng, Ralf Gräfe, Michael Paulitsch, Gereon Hinz, and Alois Knoll, "Hardware faults that matter: Understanding and estimating the safety impact of hardware faults on object detection dnns," Berlin, Heidelberg, 2022, Springer-Verlag.

[31] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei, "Ares: A framework for quantifying the resilience of deep neural networks," New York, NY, USA, 2018, Association for Computing Machinery.

Simulación de circuitos cuánticos mediante diagramas de decisión tensoriales

Vicente López[1], Alfred M. Pastor[2] y José M. Badía[1]

Resumen— **En una época en la que es de prever que durante los próximos años dispongamos de unos pocos ordenadores cuánticos poco fiables y de pequeño tamaño, es fundamental el desarrollo de simuladores eficientes de algoritmos cuánticos sobre ordenadores clásicos. La simulación puede implementarse mediante la contracción de la red de tensores asociada al circuito cuántico. El orden en que se realizan las contracciones puede modificar el coste de la simulación en varios ordenes de magnitud. Por otro lado, el uso de diagramas de decisión para la representación y manipulación de los tensores, permite aprovechar las redundancias de datos y operaciones para reducir el coste de las contracciones. En este trabajo presentamos un nuevo método de ordenación de las contracciones basado en el emparejamiento de las mismas siguiendo su orden en el circuito. La evaluación del nuevo método con distintos tipos de circuitos cuánticos demuestra que aprovecha muy eficientemente los diagramas de decisión tensoriales para igualar o mejorar heurísticas ampliamente utilizadas como las basadas en la descomposición en árbol de la red de tensores.**

Palabras clave— **Simulación de circuitos cuánticos, redes de tensores, diagramas de decisión.**

I. Introducción

La computación cuántica es un nuevo paradigma que permite resolver algunos problemas inabordables con los ordenadores clásicos actuales. El impacto de los futuros ordenadores cuánticos puede ser muy grande en múltiples áreas con un enorme potencial económico y social, como pueden ser los campos de la química, la farmacología o el aprendizaje automático, entre otros [1]. Un claro ejemplo es cómo los ordenadores cuánticos serán capaces de resolver en tiempo polinómico la factorización de grandes números enteros, operación que se encuentra en la base de gran parte de los sistemas criptográficos actuales [2, Cap. 4].

El principal problema actual de la computación cuántica es la dificultad de construir ordenadores cuánticos fiables y escalables. Por un lado, es extremadamente complicado evitar el problema de la decoherencia cuántica que hace que los cúbits, unidad básica de información cuántica, pierdan su estado en tiempos muy reducidos debido a su interacción con el entorno. Por otro lado, las puertas cuánticas son poco fiables al tener niveles de error muy elevados. Como consecuencia, solo es posible ejecutar algoritmos con pocos cúbits y puertas antes de acumular grandes errores que hacen que los resultados sean muy poco fiables. Todo ello hace que nos encontremos en la denominada era *Noisy Intermediate-Scale Quantum* (NISQ) con ordenadores poco fiables con unos pocos cientos de cúbits. Además de la mejora en la tecnología utilizada para implementar y controlar los cúbits, se está avanzando rápidamente en el diseño y aplicación de nuevos métodos de mitigación y corrección de errores. En los próximos años solo dispondremos de pocos ordenadores cuánticos de pequeña dimensión. Por ello es imprescindible el desarrollo de simuladores eficientes sobre ordenadores clásicos que tendrán una enorme utilidad para el diseño, validación y mejora de algoritmos cuánticos y también para el diseño y testeo de nuevos ordenadores cuánticos.

Actualmente existen más de 100 simuladores cuánticos disponibles escritos en distintos lenguajes de programación [3]. La forma más utilizada de simular un circuito es reducirlo a un conjunto de multiplicaciones de vectores (que describen los estados cuánticos) y de matrices (que describen los operadores). Dado que tanto los vectores como las matrices crecen exponencialmente en tamaño con número de cúbits, el coste temporal de las operaciones crece muy rápidamente con el tamaño de los circuitos. No obstante, la principal limitación para la simulación de algoritmos cuánticos de tamaño mediano es el espacio disponible, dada la enorme cantidad de memoria necesaria para almacenar el estado y los operadores a aplicar [4], [5].

Para abordar este problema, se han desarrollado diferentes técnicas para simular circuitos. Las principales son la evolución completa de la función de onda [6], los caminos de Feynman [7] y la contracción de redes de tensores [8]. Los simuladores basados en redes de tensores han demostrado ser excepcionalmente buenos simulando los circuitos cuánticos aleatorios (RQC) [9]. No obstante, la eficiencia de este tipo de simuladores depende mucho del orden en el que se contraen los tensores. Determinar el orden óptimo para contraer los tensores es un problema NP-dificil [10], por lo que es importante desarrollar buenos métodos y heurísticas que permitan aproximar dicho orden de forma eficiente.

Además del desarrollo de nuevas técnicas para simular circuitos cuánticos, también se han desarrollado nuevas formas de representar, guardar y manipular la información. Como alternativa a la representación en forma de vectores y matrices, existen los Diagramas de Decisión (DD) [11], que están cobrando especial relevancia actualmente para la representación de la información cuántica debido a sus dos principales ventajas. Por un lado, son capaces de aprovechar ciertas propiedades matemáticas de los circuitos y operadores cuánticos para reducir el espa-

[1]Dpto. de Ingeniería y Ciencia de Computadores, Universitat Jaume I, e-mail: {voliva,badia}@uji.es.

[2]Dpto. Informática, Universitat de València, e-mail: alfred.pastor@uv.es.

cio necesario para almacenar los datos. Por otro lado, guardan las operaciones realizadas y aprovechan la estructura de los operadores cuánticos para no repetirlas [12]. Un buen ejemplo de simuladores basados en DD es la herramienta DDSIM, que para simular circuitos cuánticos utiliza QuIDDs (Quantum Information Decision Diagram) [13], una variante de los DD que aprovecha propiedades de la computación cuántica.

En este trabajo evaluamos un nuevo método de ordenación de las contracciones basado en el emparejamiento de los tensores asociados a las puertas cuánticas en el orden en que se ejecutan en el circuito. Esta ordenación se basa en la idea utilizada en [14] para la construcción de la matriz que representa la funcionalidad completa del circuito. Hemos implementado este método de ordenación, que denominamos emparejamiento iterativo, utilizado matrices y un tipo de diagramas de decisión asociados con redes de tensores denominado *Tensor Decision Diagrams* (TDD) [15]. Hemos comparado el nuevo método con otros basados en la descomposición en árbol de la red de tensores, que es una heurística ampliamente utilizada en la literatura. Para hacer la comparación hemos utilizado diversos tipos de circuitos asociados a algoritmos cuánticos bien conocidos como la Transformada Cuántica de Fourier (QFT) o el algoritmo de búsqueda de Grover, entre otros.

Los resultados obtenidos demuestran que el nuevo método de ordenación saca muy buen provecho de la representación de los tensores utilizando TDD. La implementación del emparejamiento iterativo con matrices ofrece resultados mucho peores que los algoritmos basados en la descomposición en árbol. Sin embargo, cuando usamos TDD, obtenemos rendimientos similares o incluso superiores para algunos tipos de circuitos. Los experimentos nos han permitido también confirmar que el factor limitante para incrementar el tamaño de los circuitos que pueden simularse es principalmente la memoria disponible y no tanto el tiempo necesario para completar la simulación.

El resto del artículo está estructurado de la siguiente forma. En el apartado II se hace un repaso de trabajos relacionados, tanto a nivel de simulación de circuitos como de las diferentes estrategias y estructuras de datos que se pueden utilizar. En el apartado III proporcionamos el marco teórico de las redes de tensores, así como de los métodos de ordenación y de los diagramas de decisión. En el apartado IV se explica el método de emparejamiento iterativo y cómo se ha implementado. En el apartado V se explican y se discuten los experimentos que se han realizado para comparar el rendimiento del emparejamiento iterativo frente a otros tipos de métodos de ordenación utilizando diferentes estructuras de datos. Por último, en el apartado VI se ofrecen las conclusiones y se proponen posibles líneas de investigación para trabajo futuro.

II. Trabajos relacionados

Existen una gran cantidad de simuladores cuánticos que trabajan tanto en secuencial como en paralelo [3]. Algunos de los más conocidos son QuEST [16] y qHiPSTER [17]. La mayoría de estos simuladores cuánticos se basan en evolucionar el estado cuántico completo haciendo uso de matrices. El crecimiento exponencial del coste temporal y espacial con el número de cúbits hace que el supercomputador más potente no pueda simular un circuito general de más de 50 cúbits con este tipo de simuladores.

Con el objetivo de tratar de reducir los costes, Markov y Shi [18] propusieron utilizar redes de tensores para simular circuitos cuánticos. De esta forma, mostraron en su artículo que, bajo una serie de condiciones, los costes podrían llegar a ser polinómicos respecto al número de puertas del circuito. A raíz de este artículo, se han desarrollado una serie de simuladores basados en redes de tensores tales como Jet [19] o el presentado por Huang y colaboradores [20] (versión mejorada del presentado por Gray y colaboradores [5]). El problema de las redes de tensores es que el coste de realizar este tipo de simulación es muy dependiente del orden elegido para realizar las contracciones. Encontrar el orden óptimo es un problema NP-dificil, por lo que se han invertido muchos esfuerzos en desarrollar métodos y heurísticas que lo aproximen.

Entre los diferentes tipos de métodos de ordenación existentes, en primer lugar encontramos los inspirados directamente por el artículo de Markov y Shi [18] que se basan en la descomposición en árbol de la red de tensores. En esta familia encontramos algunos de los más conocidos como QuickBB [21] y Flowcutter [22]. A esta familia también pertenece el método `min-fil` [23], que es uno de los métodos de ordenación que utilizaremos como referencia en el trabajo. Otra de las familias es la basada en hipergrafos, cuya herramienta más conocida es cotengra [24] que en su artículo inicial [5], los autores presentan como un método para construir árboles de contracción utilizando métodos de partición de hipervínculos como KaHyPar [25].

Todos los ejemplos anteriores tienen algo en común, y es que trabajan con representaciones matriciales para realizar las operaciones. No obstante, esta no es la única forma que se tiene para trabajar con los circuitos. En [26] se propone por primera vez el uso de diagramas de decisión para representar los datos y las operaciones, reduciendo de esta forma tanto el coste temporal como el espacial. La librería más conocida que implementa esta idea es la llamada DD [12], que es utilizada tanto para simular como para verificar circuitos cuánticos. Esta herramienta también trabaja evolucionando el estado del circuito cuántico. La única herramienta que combina las redes tensoriales con los diagramas de decisión es TDD [27]. No obstante, a pesar de su combinación única, la herramienta suele presentar menores prestaciones que las anteriores, ya que está escrita íntegramente en Python. Esta será la herramienta que

se utilizará en el presente trabajo para explorar el potencial de esta combinación para simular circuitos cuánticos.

III. Marco teórico

En este apartado comentaremos brevemente los fundamentos teóricos básicos sobre redes de tensores y diagramas de decisión, que son en los que se apoya el desarrollo de este trabajo, en particular el método de ordenación y contracción propuesto. Un estudio más profundo sobre el funcionamiento de la computación cuántica y su fundamento matemático se puede encontrar por ejemplo en [28] y [29].

A. Redes de tensores

Los tensores han sido ampliamente utilizados tanto por físicos como por matemáticos desde hace muchos años. Popularizados en el año 1900 por Tullio Levi-Civita y Gregorio Ricci-Curbastro [30], éstos han tenido aplicaciones muy diversas, ya que proporcionan un marco matemático conciso con el que se pueden formular y resolver problemas en áreas tan diversas como la mecánica, electrodinámica o relatividad general, entre otras. También han sido utilizados en el campo de la inteligencia artificial, siendo por ejemplo la base de la famosa biblioteca desarrollada por Google, Tensorflow [31].

En el campo de la computación cuántica, Markov y Shi sentaron las bases para su uso en la simulación de circuitos. En su trabajo de 2008 demostraron que cualquier circuito cuántico de tamaño (número de puertas) polinómico con un *treewidth* logarítmico se puede simular de forma determinista en tiempo polinómico [18]. A raíz de este trabajo, se han desarrollado diferentes herramientas basadas en redes de tensores con el objetivo de simular o verificar circuitos.

Los tensores son la generalización natural de los vectores y matrices. Un tensor de rango r se define como un elemento $d_1 \times \cdots \times d_r$ existente en el espacio $\mathbb{C}^{d_1,\cdots,d_r}$. De este modo, un vector con d elementos complejos (dimensión d) se puede considerar un objeto del espacio $\mathbb{C}^d$ y por tanto es un tensor de rango 1, mientras que una matriz de dimensión $n \times m$ se puede considerar un objeto del espacio $\mathbb{C}^{n\times m}$ y por tanto es un tensor de rango 2. Uno de los motivos por el que los tensores han alcanzado su popularidad es debido a su notación, que es sencilla y transparente [32], ya que permite una representación gráfica que facilita la visualización de las redes de tensores. En dicha notación, un tensor se representa como una forma geométrica (normalmente un círculo) del que salen unas aristas abiertas etiquetadas con un índice. Podemos ver un ejemplo de este tipo de representación en la Figura 1.

Los tensores tienen un grupo de operaciones definido, tales como el producto tensorial, la traza, la contracción y el particionado [32]. La contracción de dos tensores es la operación más utilizada, y corresponde a un producto tensorial seguido de una traza entre los índices de dos tensores. Podemos ver un ejemplo

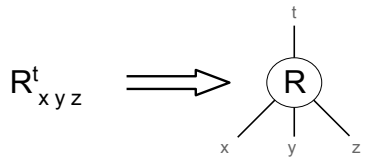


Fig. 1: Representación de un tensor R con cuatro índices x, y, z, t.

representado gráficamente en la figura 2, donde se muestra la contracción entre dos tensores de rango 3 que comparten 2 índices.

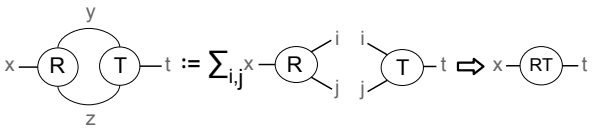


Fig. 2: Ejemplo de contracción entre dos tensores $R_x^{y,z}$ y $T_{y,z}^t$.

La figura 2 es también un ejemplo de red de tensores, que no es más que un conjunto de tensores (los vértices) que se conectan entre ellos mediante los índices (las aristas) para formar un grafo. La operación de contracción permite reducir el número de tensores que forma una red. En particular, aplicando la contracción un número suficiente de veces entre pares de tensores, es posible reducir una red de tensores a un solo tensor. El coste temporal de reducir (contraer) la red a un único tensor es muy dependiente del orden en el que se decidan contraer los tensores. Dada una red de tensores, determinar el orden óptimo de contracción es un problema NP-dificil [10]. Es por ello que se han definido diferentes heurísticas para determinar el orden de contracción de los tensores de forma que se aproxime al orden óptimo.

B. Circuitos cuánticos como redes de tensores

Un estado cuántico $|\psi\rangle$ se puede representar como un vector de la forma

$$|\psi\rangle = [\alpha_0, \alpha_1]^T, \text{ donde } \alpha_0, \alpha_1 \in \mathbb{C} \qquad (1)$$

Al ser $|\psi\rangle$ un vector, se puede representar como un tensor T_q de rango 1. Por otro lado, si pensamos en las puertas de un cúbit como una matriz de tamaño 2×2, éstas se pueden representar como tensores T_{q_1,q_2} de rango 2. Notemos que en general, en la notación de tensores no se diferencian entre los índices de entrada o de salida. Por extensión, una puerta de n cúbits se puede representar como un tensor de rango $2n$. Cuando se representan estados cuánticos, es conveniente utilizar la dirección de las aristas para diferenciar si los índices del vector viven en el espacio de Hilbert ("kets") o en su dual ("bra"). En la figura 1 se puede observar la representación usando este convenio para marcar la diferencia del espacio donde viven los índices x, y, z del índice t. Esta diferenciación también se puede realizar en la propia notación, denotando el tensor de la figura 1 como $R_{x,y,z}^t$ (en lugar de $R_{x,y,z,t}$). Ambas notaciones se pueden utilizar indistintamente, ya que son equivalentes.

Un circuito cuántico no es más que una sucesión de puertas que se aplican sobre unos determinados cúbits. Teniendo en cuenta las equivalencias anteriores, podemos ver un circuito cuántico como una red de tensores. En esta red, cada vértice (tensor) representa una puerta y cada arista representa un índice que tienen en común ambos tensores. Para un circuito de n cúbits, la matriz unitaria que representa su funcionalidad será de dimensión $2^n \times 2^n$, por lo que se puede representar como un tensor de rango $2n$. Este tensor se puede obtener al contraer el circuito cuando está representado como una red de tensores. Dado que podemos simular el comportamiento de un circuito cuántico solo con la matriz que representa su circuito, también lo podemos hacer con su forma tensorial. Este es el principio básico que utilizan los simuladores y verificadores basados en redes de tensores.

C. Métodos de ordenación de las contracciones

El objetivo de los métodos de ordenación es aproximar el orden óptimo de contracción que reduzca su coste temporal y espacial. Ambos costes dependen exponencialmente del rango de los tensores implicados en las distintas contracciones de pares de tensores llevadas a cabo. En concreto, el rango del mayor tensor obtenido en el proceso de contracción es un buen indicador del coste de contraer toda la red [18].

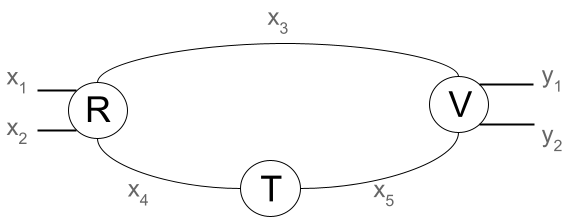


Fig. 3: Ejemplo de red tensorial con tres tensores, 3 índices cerrados(x_3, x_4, x_5) y 4 abiertos.

Veamos un ejemplo de la importancia de elegir un buen orden de contracción con la figura 3. En ella tenemos una red con tres tensores llamados R, T, V. Para esta red particular existen 3 posibles órdenes de contracción. Si elegimos contraer primero R con T, obtendremos un tensor de 4 índices (x_1, x_2, x_3, x_5), que al contraerlo con V dará lugar a un nuevo tensor con 4 índices (x_1, x_2, y_1, y_2). Un caso muy parecido es el de contraer primero los tensores T y V y contraer el resultado con R. Sin embargo, el resultado cambia si elegimos contraer primero los tensores R y V. En ese caso, el tensor resultante tiene un total de 6 índices, $(x_1, x_2, x_4, x_5, y_1, y_2)$, y al contraerlo con T se vuelve a reducir a uno de rango 4. No obstante, al haber elegido este orden, la primera contracción ha tenido un coste considerablemente mayor debido al rango del mayor tensor intermedio. Elegir un buen orden consiste en reducir al máximo este tipo de casos para evitar que los tensores intermedios crezcan más de lo necesario y reducir así el coste de la contracción de la red.

Existen diferentes familias de métodos para realizar las contracciones. La familia de métodos más importante, inspirada directamente de los resultados de Markov y Shi [18], es la basada en la búsqueda de una descomposición en árbol del *grafo lineal* asociado a la red de tensores. Uno de los primeros algoritmos de esta familia que se introdujeron para obtener el orden fue el método *tree decomposition min fil* [23]. Este será uno de los métodos que se utilizará como referencia para este trabajo, ya que es el método disponible en la herramienta TDD que utiliza la versión implementada en la librería `networkX` de Python [33].

La familia más simple de métodos de ordenación es la conocida como métodos voraces. En cada paso se usa una heurística basada en el estado actual de la red para elegir el siguiente par de tensores a contraer. Por ejemplo, los dos tensores que más índices compartan o los dos tensores de menor rango. Es precisamente en esta familia de métodos donde se enmarca la propuesta de un nuevo método de ordenación: el método de emparejamiento iterativo.

D. Diagramas de decisión

Los diagramas de decisión nacieron en computación clásica, teniendo diferentes usos y aplicaciones [34], [11]. El primer esfuerzo por adaptar esta estructura de datos para la computación cuántica se dio en [26], donde se utiliza para simular un circuito cuántico en un ordenador clásico. Esta adaptación es conocida como *Reduced Ordered Binary Decision Diagram* (ROBDD). A partir de entonces, se han desarrollado diferentes herramientas que utilizan este tipo de diagramas, ya que permiten aprovechar la estructura de las matrices asociadas a los circuitos cuánticos para reducir el coste espacial y temporal de las operaciones. En este trabajo nos vamos a centrar en una versión particular de los ROBDD conocida como *Tensor Decision Diagram* (TDD). Este tipo de diagramas de decisión fue propuesto en [27] y tiene la particularidad de que combina la idea de utilizar redes de tensores para representar un circuito con la representación basada en diagramas de decisión.

Los TDD son un tipo de diagrama de decisión que utilizan nodos para representar los índices. Éstos están unidos a sus sucesores mediante aristas, que además contienen un peso que indica el valor por el que se debe multiplicar el TDD representado por el sucesor. La ausencia de peso en la arista en su representación gráfica indica que su peso es de 1. Los nodos tienen dos sucesores, uno que representa el diagrama cuando el valor del índice es 0 (el de la izquierda, representado normalmente por una línea discontinua) y otro que lo representa cuando el índice toma el valor 1 (el de la derecha, representado normalmente por una línea continua).

Vamos a ver un ejemplo de cómo representar una circuito como un TDD. Supongamos que queremos representar el circuito mostrado en la parte izquierda la figura 4. En la parte derecha de la figura podemos

ver la matriz que define la funcionalidad del circuito. A partir de matriz podemos construir el TDD mostrado en la figura 5. Si todos los elementos de la matriz fuesen distintos y sin factores comunes, el diagrama obtenido sería un árbol con 4 niveles de nodos, uno por índice, y un último nivel con los 16 valores de la matriz. Los diagramas de decisión aprovechan la estructura y repetición de elementos o bloques enteros en las matrices para reducir, en ocasiones de modo muy notable, el número de nodos y aristas y, por tanto, el espacio necesario para almacenar y operar con redes de tensores. En el ejemplo podemos ver que no solo se repiten mucho los elementos, sino que los dos cuadrantes antidiagonales son iguales y los dos diagonales también lo son, aunque con el signo invertido. De este modo, utilizando técnicas de normalización y reducción es posible representar el circuito con un diagrama de tipo TDD como el de la figura 5 conteniendo tan solo 7 nodos y 5 pesos distintos de 1 en sus aristas [27]. En la matriz de la figura 4 hemos representado las 4 combinaciones de valores posibles de los índices de entrada asociados a las filas (x_1, x_2) y de los índices de salida asociados a las columnas (y_1, y_2). Por ejemplo, en rojo podemos ver que el valor asociado a los índices de entrada `10` y de salida `10` es `0`. Siguiendo el orden en que están situados dichos índices en los 4 niveles del TDD, podemos recorrer en rojo el camino $x_1 = 1, y_1 = 1, x_2 = 0, y_2 = 0$ hasta alcanzar el nodo hoja. Si multiplicamos los pesos asociados a las aristas recorridas, podemos ver que el valor del elemento obtenido es $i/\sqrt{2} \cdot (-1) \cdot 1 \cdot 0 = 0$, lo que coincide con el valor correspondiente en la matriz.

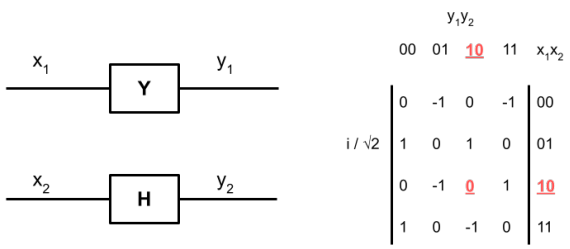


Fig. 4: Ejemplo de un circuito cuántico y matriz unitaria asociada al tensor representando su funcionalidad. En rojo, elemento representado en el TDD de la siguiente figura siguiendo las aristas del mismo color.

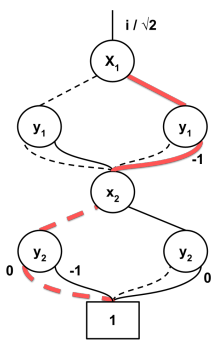


Fig. 5: TDD que representa el circuito de la figura 4.

Es muy habitual utilizar dos estructuras de datos para almacenar la información y las operaciones asociadas a la construcción y manipulación de los diagramas de decisión. La tabla de unicidad (`unique table`) almacena la información sobre los nodos del árbol de modo que los nodos con información repetida no se repitan y nos permitan reducir el coste espacial. Por otro lado, se utiliza una tabla de cómputo (`compute table`) para almacenar las operaciones y no repetirlas si tienen que volver a realizarse. Las operaciones sobre diagramas de decisión de tipo árbol suelen implementarse de modo recursivo en base a los subárboles de cada nodo. En muchas ocasiones, dichos subárboles se repiten, tanto en la construcción de un diagrama como en la contracción de dos de ellos. Cuando una operación entre subárboles ya se ha realizado previamente, ésta se recupera de la tabla de cómputo para evitar repetirla. Los subárboles afectados pueden ser de gran tamaño, lo que evitaría un porcentaje elevado de los cálculos con diagramas de decisión.

IV. Método de emparejamiento iterativo

Una estrategia sencilla para ordenar las contracciones en una red de tensores para obtener el diagrama que representa la funcionalidad del circuito completo es contraer sucesivamente los diagramas de decisión asociados a las puertas del circuito acumulando el resultado. El problema de esa estrategia es que el tamaño de los diagramas crece rápidamente al eliminarse las redundancias existentes, reducirse la dispersión e incrementarse el grado de entrelazamiento. Así, contracciones consecutivas aumentan muy rápidamente el tamaño de los diagramas y el coste espacial y temporal de las operaciones. En [14] se propone una posible solución a un problema equivalente. En su caso se trata de obtener la matriz que representa la funcionalidad completa del circuito llevando a cabo multiplicaciones de matrices asociadas a las puertas. Las matrices se representan usando otro tipo de diagrama de decisión distinto a los TDD obtenido a partir de los cuadrantes de las matrices. Los autores proponen aprovechar la propiedad asociativa del producto de matrices y multiplicar las matrices asociadas a pares de puertas disjuntos, combinando en sucesivos niveles los pares de matrices obtenidos. De esta forma los diagramas asociados a las matrices que se manejan para obtener el mismo diagrama final aprovechan mejor las redundancias y tienen tamaños medios que pueden ser muy inferiores a los obtenidos con las multiplicaciones sucesivas, tal y como se demuestra en dicho artículo. En concreto, el emparejamiento de las multiplicaciones propuesto en dicho artículo es el siguiente:

$$U = (U_1 \cdot U_2) \cdot (U_3 \cdot U_4) \cdot \ldots \cdot (U_{n-1} \cdot U_n) \qquad (2)$$

siendo U la matriz que representa la funcionalidad del circuito y siendo U_i la matriz que representa la funcionalidad de la puerta i para cualquiera de las n puertas del circuito. De esta forma obtendremos lo siguiente:

$$U = U_{1,2} \cdot U_{3,4} \cdot \ldots \cdot U_{n-3,n-2} \cdot U_{n-1,n} \qquad (3)$$

sobre lo que se puede volver a aplicar la misma propiedad asociativa para emparejar las matrices entre ellas. Esta idea se puede aplicar de forma iterativa hasta obtener una única matriz. El esquema propuesto define un orden en forma de árbol en el cual, para cada nivel $l \in \{0, 1, \cdots, k\}$ intervienen un máximo de 2^l operaciones.

El método de emparejamiento iterativo para redes de tensores es una adaptación de esta idea pero aplicada a la contracción de redes de tensores. El principio básico es contraer la red de tensores siguiendo el mismo orden en el que se multiplicarían los pares de matrices, tratando así de generar contracciones repetidas que puedan ser ahorradas por los TDD. Este método funciona de la forma descrita por el pseudocódigo de la figura 6. El método recibe los tensores como una lista. El funcionamiento básico es el siguiente: el método va a ir reduciendo el número de tensores presentes en la lista hasta que únicamente quede uno de ellos, que será el tensor que representa la funcionalidad del circuito. Mientras la lista de tensores contenga más de un tensor, el método creará una nueva lista para guardar el resultado de las contracciones que va a realizar, cuyo tamaño será de la mitad redondeado a la baja del número de tensores contenidos en la lista actual. Los tensores de la lista se contraen por parejas guardándolos en una nueva lista. Si el número de elementos de la lista es impar, el último tensor se contraerá con el resultado de contraer los dos tensores anteriores.

```
def emparejamiento_iterativo(tensores):
    ta = tensores
    while tamaño(ta) > 1:
        nt = crear_lista()
        i = 0
        while ( i < tamaño(ta) - 1 ):
            nt[i//2] = contraer(ta[i], ta[i+1])
            i = i + 2

        if (i == tamaño(ta) - 1 ):
            nt[i//2] = contraer(ta[i], ta[i//2])

        ta = nt

    return nt[0] # Contendrá un único tensor final
```

Fig. 6: Pseudocódigo para la implementación del método de emparejamiento iterativo.

V. Simulación de circuitos y análisis de resultados

A. Entorno experimental y banco de pruebas

El análisis experimental se ha llevado a cabo utilizando dos servidores multinúcleo:

- `nasp` es un servidor con 4 processadores Intel Xeon Gold 6330H a 2 GHz con 24 núcleos cada uno, un total con 394 GiB de memoria RAM DDR4 y 33 MiB de memoria caché con el sistema operativo Linux 5.4.0-72-generic 80-Ubuntu.
- `seb` es un servidor con 2 procesadores Intel Xeon E5-2695 v4 a 2.1 GHz con 36 núcleos cada uno, un total de 128 GiB de memoria RAM y un sistema operativo Linux Centos v7.

Para realizar las pruebas hemos utilizado dos entornos de simulación, cada uno con una estructura de datos diferente. Uno de ellos está basado en el uso de diagramas de tipo TDD para implementar los tensores y el otro está basado en el uso de matrices.

En `nasp` hemos utilizado el entorno de simulación basado en TDD [1] descrito en [15]. La herramienta está implementada en Python 3 y permite utilizar los distintos métodos de ordenación de las contracciones descritos en [15]. Además, los autores de dicho artículo han añadido una versión del método de ordenación `min fil` apoyándose en el paquete `networkX` de Python. Hemos modificado la herramienta para añadir el nuevo método de ordenación por parejas iterativo presentado en este artículo.

En `seb` hemos utilizado el entorno de simulación QXTools [2] descrito en [35]. Esta herramienta forma parte del proyecto europeo QuantEx pensado para la simulación de circuitos cuánticos sobre clusters de ordenadores y aceleradores. QXTools forma parte de un conjunto de paquetes escritos en el lenguaje Julia que busca obtener altas prestaciones mediante representación y manipulación eficiente de tensores basados en matrices.

Dependiendo del entorno hemos comparado 4 métodos de ordenación de las contracciones distintos:

- `iter`: nuevo método de ordenación mediante emparejamiento iterativo descrito en el apartado IV.
- `sec`: método de emparejamiento secuencial. Comenzamos con el TDD asociado a la primera puerta del circuito y lo contraemos con el asociado a la segunda, a continuación contraemos el TDD resultante con el asociado a la tercera puerta y así sucesivamente. Este es el método de contracción por defecto incluido en el entorno descrito en [15].
- `mf`: método `min fil` basado en la descomposición en árbol y descrito brevemente en el subapartado III-C.
- `fc`: método `Flowcutter`, basado también en la descomposición en árbol, comentado en el apartado II y cuyas excelentes prestaciones se muestran en [5].

Las pruebas se han llevado a cabo con un amplio y variado conjunto de circuitos cuánticos obtenidos en [36]. Para mostrar los resultados obtenidos, hemos seleccionado 4 de dichos circuitos que implementan algoritmos cuánticos ampliamente conocidos:

- `grover`: Algoritmo para la búsqueda de datos en secuencias no ordenadas.
- `qft`: Transformada Cuántica de Fourier (QFT por sus siglas en inglés). Es utilizada con frecuencia en la implementación de diversos algo-

[1] `https://github.com/VeriQC/TDD`
[2] `https://github.com/JuliaQX/QXTools.jl`

ritmos cuánticos, como el algoritmo de Shor.
- `qpe`: Estimador de fase cuántica (QPE por sus siglas en ingles). Utilizado para estimar el valor propio de un operador unitario y componente básica del algorithm de Shor o el algoritmo cuántico para la resolución de sistemas lineales de ecuaciones.
- `qwalk`: Versión cuántica de los caminos aleatorios. Es una formalización matemática de la trayectoria que resulta de hacer sucesivos pasos aleatorios y que, en su versión cuántica, aprovecha la aleatoriedad de la superposición cuántica y la medida para obtener la aleatoriedad.

Cada tipo de circuito se probó variando el número de cúbits y, por tanto, modificando también el número de puertas y la profundidad.

B. Resultados experimentales

El coste de la contracción de un circuito depende de su tamaño en términos de número de cúbits y número de puertas, pero también de su topología y su grado de entrelazamiento. Estas características, y el orden en que se realizan las contracciones, determinan el tamaño de los tensores a contraer y, por tanto, el coste espacial y temporal del proceso de contracción. En la figura 7 podemos ver, en escala logarítmica, el rendimiento del método de emparejamiento iterativo para los 4 tipos diferentes de circuitos probados. Como se puede observar, no todos los tipos de circuitos llegan hasta la misma cantidad de cúbits. Esto es debido a que tanto el tiempo como el espacio necesario para realizar las contracciones varían dependiendo del tipo de circuito. En el caso de los circuitos `qft` y `qwalk`, el espacio fue el factor limitante, ya que el proceso agotó los 394 GiB de memoria disponible al llegar a 18 y 43 cúbits, respectivamente. En el caso de los circuitos `grover` y `qpe` el factor limitante fue el tiempo, ya que, transcurridas las 24 horas dadas para todas las pruebas de cada tipo de circuito, se llegó a 19 y 18 cúbits sin que se agotará la memoria disponible. La figura nos permite también diferenciar claramente el coste de contracción de los distintos tipos de circuitos. Este coste crece exponencialmente con el número de cúbits en todos los casos. Sin embargo, podemos constatar que el coste de contracción de los circuitos de tipo `qwalk` es muy inferior, lo que permite más que doblar el número de cúbits de los circuitos contraídos con respecto a los otros tres tipos de circuitos. Aunque el coste de los tres restantes sigue una pendiente más similar, podemos observar que los circuitos de tipo `qft` son los más costosos de contraer, tanto en tiempo como en espacio.

Para analizar la eficiencia del método de emparejamiento iterativo, lo comparamos con otros dos métodos de ordenación de las contracciones: `sec` y `mf`. La figura 8 muestra, en escala logarítmica, el coste temporal de la contracción de dos de los tipos de circuitos cuando se aplican los tres métodos de ordenación de las contracciones comparados. Lo primero que podemos constatar es la confirmación cuantitativa de

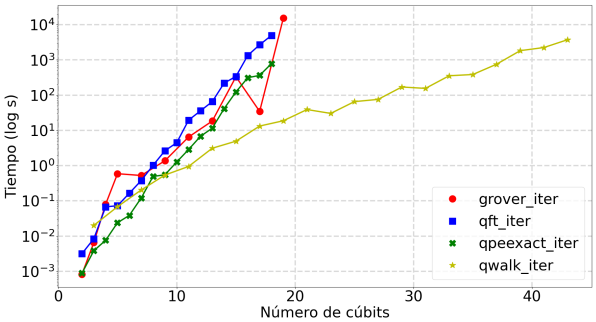


Fig. 7: Rendimiento del método de emparejamiento iterativo para 4 tipos de circuitos cuánticos utilizando TDDs.

la ventaja del método de emparejamiento iterativo sobre el método secuencial comentada en el apartado IV. El emparejamiento iterativo nos permite contraer con tensores de menor tamaño de media, lo que además saca un mayor partido de las ventajas de utilizar TDD como estructura de representación de dichos tensores. La figura muestra que el método de emparejamiento secuencial tiene un coste claramente superior para ambos tipos de circuitos y es previsible que lo tenga para otros tipos, independientemente de sus características.

Por otro lado, la figura nos permite constatar que el método de emparejamiento iterativo siempre tiene un coste temporal similar o inferior que el método `mf`. A pesar de tratarse de un método de ordenación de las contracciones en apariencia más simple que el basado en la heurística más utilizada para la ordenación, obtiene resultados muy positivos en todos los casos. En concreto, en el caso de los circuitos de tipo `qwalk` supera siempre el método `mf` independientemente del número de cúbits del circuito. Este comportamiento es debido al buen aprovechamiento del TDD que logra el emparejamiento iterativo, confirmando los resultados obtenidos en [14] al utilizar otro tipo de diagramas de decisión para construir la representación funcional de un gran número y variedad de circuitos.

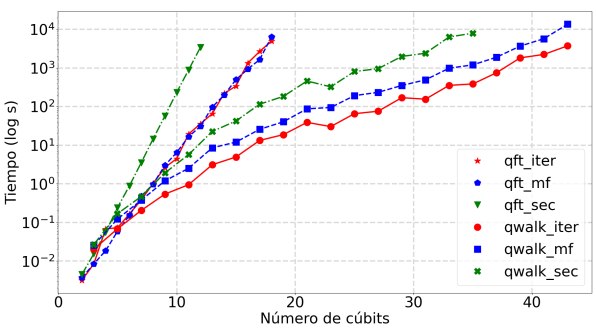


Fig. 8: Comparación del tiempo de contracción entre los métodos de ordenación textttmin fil, el emparejamiento secuencial y emparejamiento iterativo para 2 tipos de circuitos utilizando TDDs en `nasp`.

En la figura 9 podemos ver el porcentaje de tiempo ahorrado por el uso de la tabla de cómputo. Estos porcentajes se han calculado ejecutando dos versiones del método de emparejamiento iterativo. En la primera de ellas se utiliza esta estructura de datos para evitar la repetición de operaciones de contracción.

En la segunda, se ha modificado el código por defecto para que el algoritmo nunca encontrase operaciones repetidas en dicha estructura. La figura muestra un crecimiento casi lineal del ahorro temporal para ambos métodos de ordenación de las contracciones con los circuitos de tipo `qft`. El comportamiento es similar para otros tipos de circuitos. Podemos ver que el ahorro llega a casi el 100 % cuando alcanzamos los 35 cúbits. Es evidente que el objetivo de la tabla de cómputo es ahorrar operaciones y coste temporal a los algoritmos que usan TDDs, sin embargo resulta llamativo un ahorro tan elevado. Además, podemos ver que el ahorro del método más lento (`mf`) es mayor que el del más rápido (`iter`). Este comportamiento merece un análisis más detallado del uso de la estructura de datos que realizan estos métodos de ordenación con distintos tipos de circuitos. En concreto, es importante analizar no solo el tiempo ahorrado al usar la tabla de cómputo, sino también el espacio ocupado por los datos para distintos tamaños de circuitos y la eficiencia con que se usa la memoria, e incluso si se está utilizando la memoria virtual en algunos casos.

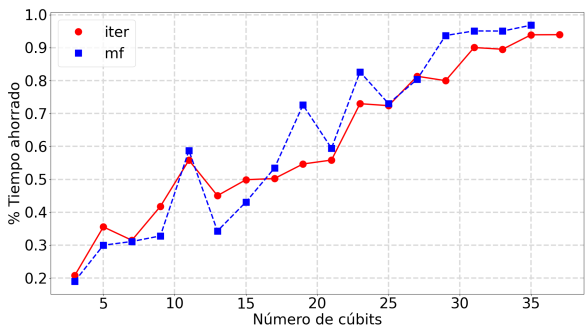


Fig. 9: Comparación del tiempo ahorrado gracias al efecto de la tabla de cómputo entre los métodos de ordenación `min fil` (línea punteada) y emparejamiento iterativo (línea continua) para los qwalk utilizando TDDs en `nasp`.

Para analizar las posibles ventajas de utilizar estructuras de datos de tipo diagrama de decisión como los TDD hemos comparado el rendimiento de los algoritmos `iter` y `mf` implementados con ellas y con matrices. En concreto, la versión de los métodos con matrices se ha implementado y ejecutado con el entorno `QXTools` descrito en el subapartado V-A. Los experimentos se han realizado en `seb` para circuitos de tipo `qft`. Es importante tener en cuenta que los métodos basados en TDD están implementados en Python, mientras los métodos basados en matrices están implementados en un Julia, un lenguaje especialmente diseñado para obtener altas prestaciones.

La figura 10 muestra el tiempo de contracción de circuitos de tipo `qft` utilizando la ordenación proporcionada por tres métodos distintos: `iter`, `mf` y `fc`. Lo primero que podemos destacar de los resultados mostrados es que el método `iter` implementado con matrices tiene un rendimiento muy inferior a los dos métodos basados en descomposición en árbol. Este comportamiento contrasta claramente con el observado al comparar estos tipos de métodos implementados con TDD. En ese caso (ver Fig. 8) los

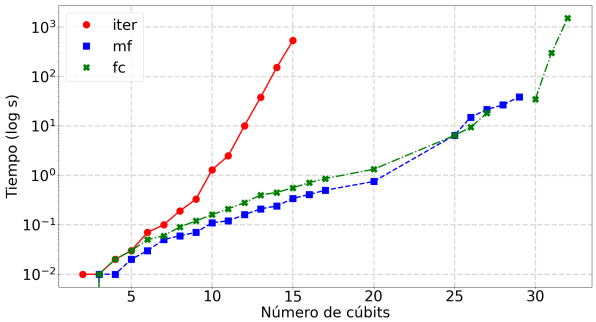


Fig. 10: Comparación del tiempo de contracción entre 3 métodos de ordenación para los circuitos de tipo `qft` utilizando matrices (QXTools) en `seb`.

métodos `iter` y `mf` obtienen rendimientos similares. Esta diferencia en el comportamiento podría ser un reflejo directo de la ventaja que supone evitar la repetición de operaciones utilizando TDD, frente a la realización de todas ellas, aunque sea de modo muy eficiente, cuando utilizamos matrices.

Por otro lado, observamos que cada método llega hasta un límite de cúbits diferente. En este caso, el factor limitante en todas las ocasiones fue el coste espacial. El método `iter` tiene también un coste espacial muy superior a los otros dos cuando utilizamos matrices para almacenar los tensores a contraer. En cuanto a los dos métodos basados en descomposiciones en árbol, `mf` tiene un coste temporal inferior a `fc` hasta circuitos con 25 cúbits. Sin embargo, el coste de ambos se iguala hasta que `mf` agota la memoria con 29 cúbits. El método `fc` puede contraer circuitos hasta 32 cúbits, aunque su coste temporal se incrementa muy notablemente. De nuevo, todos estos resultados merecen un análisis más detallado utilizando otros tipos de circuitos. Claramente es de prever que el método `iter` será el más costoso utilizando matrices. Sin embargo, el comportamiento relativo de los otros dos métodos y el máximo número de cúbits alcanzables puede variar considerablemente. Es importante entender con mayor detalle la causa de dicho comportamiento, tanto en relación con las características de los circuitos, como en relación al uso de la CPU y la memoria que hace cada uno de los métodos.

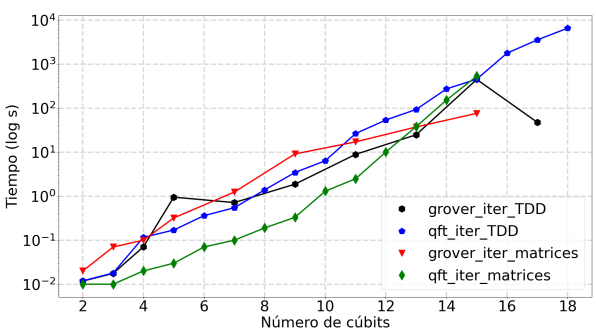


Fig. 11: Comparación del tiempo de contracción del método binario para los circuitos `grover` y `qft` usando la implementación basada en TDDs y la implementación basada en matrices (QXTools) en `seb`.

Por último, vamos a comparar los tiempos de contracción usando el nuevo método propuesto cuando implementamos las redes de tensores usando TDDs

y matrices. Hay que tener en cuenta que en el primer caso se usa Python y en el segundo un lenguaje enfocado a las altas prestaciones como Julia. Este hecho supondrá siempre una clara ventaja para la implementación con matrices. En la figura 11 se muestran, en escala logarítmica, los tiempos de contracción en `seb` para los circuitos de tipo `grover` y `qft`. Se puede observar que el comportamiento es diferente dependiendo del tipo de circuito, ya que para los circuitos de tipo `grover`, los TDDs obtienen un tiempo menor que las matrices a pesar de usar Python. No obstante, en el caso de las `qft` son las matrices las que obtienen tiempos menores. Sin embargo, usando matrices solo es posible contraer circuitos con hasta 15 cúbits debido a las limitaciones de memoria, mientras con TDDs podemos incrementar algo ese tamaño. Este factor podría también explicar por qué la ventaja de usar matrices disminuye conforme incrementamos el número de cúbits.

También es llamativo el hecho de que los TDDs tienen dos picos superiores en tiempo de contracción para los circuitos `grover` en 5 y 15 cúbits, que no se observan para el mismo tipo de circuitos con la implementación de matrices. Esta anomalía podría ser explicada por la forma que tienen estos dos circuitos particulares, que están haciendo que se aprovechen menos las ventajas de los TDDs, haciendo así que el tiempo se dispare respecto a circuitos parecidos donde sí se están aprovechando. No obstante, esto merece un análisis más profundo.

VI. Conclusiones

En este trabajo evaluamos un nuevo método de ordenación de las contracciones para la simulación de algoritmos cuánticos sobre ordenadores cuánticos utilizando redes de tensores. El nuevo método se basa en el emparejamiento de las contracciones de los tensores en el orden en que aparecen en el circuito. Su implementación utilizando una representación matricial de los tensores ofrece resultados peores que otras heurísticas basadas en la descomposición en árbol de la red. Sin embargo, el nuevo método aprovecha de modo muy eficiente las estructuras de datos utilizadas para representar las redes de tensores mediante diagramas de decisión. En concreto, utilizando diagramas de decisión tensoriales (TDD), esta ordenación de las contracciones mejora claramente el orden de contracción del emparejamiento secuencial de los tensores del circuito. Además, su rendimiento es similar o superior al de las heurísticas basadas en la descomposición en árbol, dependiendo de las características del circuito simulado. El coste de la simulación aumenta rápidamente con el número de cúbits de los circuitos, aunque depende en buena medida de las características de entrelazamiento de los mismos. Los experimentos demuestran que en muchos casos el factor que limita el tamaño del circuito simulado es más el coste espacial de almacenar los tensores generados que el coste temporal de llevar a cabo su contracción.

Como trabajo futuro existen dos vías principales que se pueden abordar. La primera es analizar con mayor detalle y una mayor variedad de circuitos cuánticos la relación entre las características de los circuitos y el ahorro de tiempo y memoria que supone el uso de las estructuras de datos con que se implementan los TDD. La otra vía es tratar de utilizar técnicas para paralelizar la librería para permitir la contracción en paralelo de los circuitos. También puede resultar interesante realizar una implementación eficiente de los TDD con un lenguaje como C++ o Julia para comparar su eficiencia en igualdad de condiciones con las versiones basadas en matrices.

Agradecimientos

Este trabajo ha sido financiado con el proyecto UJI-B2021-26 de la Universitat Jaume I y por la ayuda PID2020-113656RB-C21 financiada por MCIN/AEI/10.13039/50110011033 y el MICINN.

Referencias

[1] World Economic Forum, "State of Quantum Computing: Building a Quantum Economy," `https://www3.weforum.org/docs/WEF_State_of_Quantum_Computing_2022.pdf`, 2022.

[2] National Academies of Sciences, Engineering, and Medicine, *Quantum Computing: Progress and Prospects*, The National Academies Press, Washington, DC, 2019.

[3] Quantiki, "List of QC simulators," `https://quantiki.org/wiki/list-qc-simulators`, 2023.

[4] Cupjin Huang, Fang Zhang, Michael Newman, Xiaotong Ni, Dawei Ding, Junjie Cai, Xun Gao, Tenghui Wang, Feng Wu, Gengyan Zhang, et al., "Efficient parallelization of tensor network contraction for simulating quantum computation," *Nature Computational Science*, vol. 1, no. 9, pp. 578–587, 2021.

[5] Johnnie Gray and Stefanos Kourtis, "Hyper-optimized tensor network contraction," *Quantum*, vol. 5, pp. 410, 2021.

[6] Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T Chong, "Full-state quantum circuit simulation by using data compression," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–24.

[7] Ethan Bernstein and Umesh Vazirani, "Quantum complexity theory," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 11–20.

[8] Trevor Vincent, Lee J O'Riordan, Mikhail Andrenkov, Jack Brown, Nathan Killoran, Haoyu Qi, and Ish Dhand, "Jet: Fast quantum circuit simulations with parallel task-based tensor-network contraction," *Quantum*, vol. 6, pp. 709, 2022.

[9] Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J Bremner, John M Martinis, and Hartmut Neven, "Characterizing quantum supremacy in near-term devices," *Nature Physics*, vol. 14, no. 6, pp. 595–600, 2018.

[10] Lam Chi-Chung, P Sadayappan, and Rephael Wenger, "On optimizing a class of multi-dimensional loops with reduction for parallel execution," *Parallel Processing Letters*, vol. 7, no. 02, pp. 157–168, 1997.

[11] Sheldon B. Akers, "Binary decision diagrams," *IEEE Transactions on computers*, vol. 27, no. 06, pp. 509–516, 1978.

[12] Alwin Zulehner and Robert Wille, "Advanced simulation of quantum computations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 848–859, 2018.

[13] George F Viamontes, Igor L Markov, and John P Hayes, "High-performance QuIDD-based simulation of quantum circuits," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*. IEEE, 2004, vol. 2, pp. 1354–1355.

[14] Lukas Burgholzer, Rudy Raymond, Indranil Sengupta, and Robert Wille, "Efficient construction of functional

representations for quantum algorithms," in *Reversible Computation: 13th International Conference, RC 2021, Virtual Event, July 7–8, 2021, Proceedings*. Springer, 2021, pp. 227–241.

[15] Xin Hong, Xiangzhen Zhou, Sanjiang Li, Yuan Feng, and Mingsheng Ying, "A tensor network based decision diagram for representation of quantum circuits," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 6, pp. 1–30, 2022.

[16] Tyson Jones, Anna Brown, Ian Bush, and Simon C Benjamin, "QuEST and high performance simulation of quantum computers," *Scientific reports*, vol. 9, no. 1, pp. 1–11, 2019.

[17] Gian Giacomo Guerreschi, Justin Hogaboam, Fabio Baruffa, and Nicolas PD Sawaya, "Intel quantum simulator: A cloud-ready high-performance simulator of quantum circuits," *Quantum Science and Technology*, vol. 5, no. 3, pp. 034007, 2020.

[18] Igor L Markov and Yaoyun Shi, "Simulating quantum computation by contracting tensor networks," *SIAM Journal on Computing*, vol. 38, no. 3, pp. 963–981, 2008.

[19] Trevor Vincent, Lee J O'Riordan, Mikhail Andrenkov, Jack Brown, Nathan Killoran, Haoyu Qi, and Ish Dhand, "Jet: Fast quantum circuit simulations with parallel task-based tensor-network contraction," *Quantum*, vol. 6, pp. 709, 2022.

[20] Cupjin Huang, Fang Zhang, Michael Newman, Junjie Cai, Xun Gao, Zhengxiong Tian, Junyin Wu, Haihong Xu, Huanjun Yu, Bo Yuan, et al., "Classical simulation of quantum supremacy circuits," *arXiv preprint arXiv:2005.06787*, 2020.

[21] Vibhav Gogate and Rina Dechter, "A complete anytime algorithm for treewidth," *arXiv preprint arXiv:1207.4109*, 2012.

[22] Ben Strasser, "Computing tree decompositions with flowcutter: PACE 2017 submission," *arXiv preprint arXiv:1709.08949*, 2017.

[23] Hans L Bodlaender and Arie MCA Koster, "Treewidth computations i. upper bounds," *Information and Computation*, vol. 208, no. 3, pp. 259–275, 2010.

[24] Johnnie Gray, "Cotengra," `https://github.com/jcmgray/cotengra`, 2023.

[25] Sebastian Schlag, Tobias Heuer, Lars Gottesbüren, Yaroslav Akhremtsev, Christian Schulz, and Peter Sanders, "High-quality hypergraph partitioning," *ACM Journal of Experimental Algorithmics*, vol. 27, pp. 1–39, 2023.

[26] Randal E Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[27] Xin Hong, Xiangzhen Zhou, Sanjiang Li, Yuan Feng, and Mingsheng Ying, "A tensor network based decision diagram for representation of quantum circuits," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 6, pp. 1–30, 2022.

[28] Robert S Sutor, *Dancing with Qubits: How quantum computing works and how it can change the world*, Packt Publishing Ltd, 2019.

[29] Vicente López Oliva, "Fundamentos de la computación cuántica," *TEMat*, vol. 6, pp. 31–47, 2022.

[30] Gregorio Ricci-Curbastro, Robert Hermann, MMG Ricci, and Tullio Levi-Civita, *Ricci and Levi-Civita's Tensor Analysis Paper: Translation, Comments, and Additional Material*, Number 2. Math Science Press, 1975.

[31] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, and Co, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Software available from tensorflow.org.

[32] Jacob C Bridgeman and Christopher T Chubb, "Hand-waving and interpretive dance: an introductory course on tensor networks," *Journal of physics A: Mathematical and theoretical*, vol. 50, no. 22, pp. 223001, 2017.

[33] Aric Hagberg and Drew Conway, "Networkx: Network analysis with python," *URL: https://networkx. github. io*, 2020.

[34] David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John Hooker, *Decision diagrams for optimization*, vol. 1, Springer, 2016.

[35] John Brennan, Momme Allalen, David Brayford, Kenneth Hanley, Luigi Iapichino, Lee J O'Riordan, Myles Doyle, and Niall Moran, "Tensor network circuit simulation at exascale," in *2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS)*. IEEE, 2021, pp. 20–26.

[36] Nils Quetschlich, Lukas Burgholzer, and Robert Wille, "MQT Bench: Benchmarking software and design automation tools for quantum computing," 2022, MQT Bench is available at `https://www.cda.cit.tum.de/mqtbench/`.

[37] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[38] Hajo J Broersma, Elias Dahlhaus, and Ton Kloks, "A linear time algorithm for minimum fill-in and treewidth for distance hereditary graphs," *Discrete Applied Mathematics*, vol. 99, no. 1-3, pp. 367–400, 2000.

[39] Albert Frisch, "IBM Q—Introduction into quantum computing with live demo," in *2017 30th IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2017, pp. 1–2.

[40] Philipp Gerbert and Frank Rueß, "The next decade in quantum computing and how to play," *Boston Consulting Group*, 2018.

[41] Norbert M Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A Landsman, Kenneth Wright, and Christopher Monroe, "Experimental comparison of two quantum computing architectures," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3305–3310, 2017.

[42] Esteban A Martinez, Christine A Muschik, Philipp Schindler, Daniel Nigg, Alexander Erhard, Markus Heyl, Philipp Hauke, Marcello Dalmonte, Thomas Monz, Peter Zoller, et al., "Real-time dynamics of lattice gauge theories with a few-qubit quantum computer," *Nature*, vol. 534, no. 7608, pp. 516–519, 2016.

[43] Xiao Yuan, "A quantum-computing advantage for chemistry," *Science*, vol. 369, no. 6507, pp. 1054–1055, 2020.

[44] Alwin Zulehner, Stefan Hillmich, and Robert Wille, "How to efficiently handle complex values? implementing decision diagrams for quantum computing," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–7.

Metodología de adquisición y procesamiento de datos para la detección de crisis epilépticas utilizando dispositivos IoT y GPUs

F. Romero[1] , L.F. Romero[2], M. Lupión[1], J.F. Sanjuan[1] y P.M. Ortigosa[1]

Resumen— **En este estudio, se describe una metodología para la toma de datos de pacientes con epilepsia, utilizando dispositivos portátiles y tecnologías inalámbricas. Se utilizan cuatro fuentes de datos principales, de distintos equipos médicos y electrónica de consumo con el objetivo de identificar las señales relevantes durante un ataque epiléptico, y diseñar una pulsera final que integre los sensores implicados.**

Las peculiaridades de cada fuente de datos se explica en detalle, incluyendo los protocolos de comunicación utilizados y el almacenamiento de los datos en una base de datos InfluxDB. Además, se menciona la necesidad de optimización de los datos debido a su volumen y se proponen soluciones como el análisis en el dominio de frecuencias, así como el uso de GPUs para el procesamiento eficiente. La metodología propuesta optimiza el almacenamiento, transmisión y análisis de las señales, impulsando el entrenamiento de redes neuronales para la predicción y detección en tiempo real de las crisis epilépticas.

Este enfoque tiene un impacto significativo en la atención médica y la calidad de vida de los pacientes, al permitir una detección precisa de las crisis epilépticas, la evaluación de la carga de la enfermedad, así como la prevención de accidentes o complicaciones graves durante el ataque.

Palabras clave— **IoT, Epilepsia, GPU, Preprocesamiento de datos.**

I. Introducción

La epilepsia es una enfermedad caracterizada por la hiperexcitabilidad e hipersincronización de la actividad neuronal. Según la Organización Mundial de la Salud, alrededor de 50 millones de personas sufren de epilepsia [1], y aproximadamente el 30 % de los pacientes son refractarios a los tratamientos convencionales [2]. La detección precisa de las crisis epilépticas es crucial para evaluar la carga de la enfermedad, prevenir la muerte súbita inesperada en la epilepsia y mejorar la calidad de vida de los pacientes.

Actualmente, la monitorización de las convulsiones se basa en las declaraciones de los pacientes y sus familias, pero esto puede ser poco confiable debido a la falta de informes, las convulsiones que se pasan por alto y las dificultades para recordar las convulsiones [3]. Si bien la video-electroencefalografía a largo plazo es el estándar para diagnosticar y evaluar la epilepsia, es costosa y requiere mucho tiempo [4], lo que lleva a la necesidad de dispositivos portátiles para la monitorización de las crisis.

[1]Dpto. de Informática, Universidad de Almería, ceiA3, e-mail: fr@uma.es, {marcoslupion, jsanjuan,ortigosa}@ual.es

[2]Dpto. de Arquitectura de computadores, Universidad de Málaga, e-mail: felipe@uma.es

Existen diferentes tipos de crisis epilépticas con distintas características y gravedad. Los sensores como el acelerómetro, la actividad electrodérmica, la electromiografía, el electrocardiograma, el electroencefalograma y la grabación de vídeo se utilizan para detectar y evaluar estas crisis [5]. Sin embargo, muchos dispositivos actuales tienen limitaciones en términos de precio (superando los 1000 euros), autonomía (24 horas como máximo) y disponibilidad en el mercado.

En el desarrollo de dispositivos de detección de epilepsia, se han establecido cinco fases de evaluación y validación clínica [6]. Estas fases clasifican los dispositivos según el número de usuarios probados, el porcentaje de acierto en la detección de ataques y otros criterios. Actualmente, solo unos pocos dispositivos se encuentran en las fases avanzadas de desarrollo y validación.

El Internet de las Cosas (IoT) juega un papel fundamental en este escenario, ya que permite la conexión y comunicación entre dispositivos médicos, recopilando datos valiosos en tiempo real [7]. Sin embargo, esta interconexión masiva de dispositivos también genera una enorme cantidad de datos que deben ser gestionados de manera eficiente. Con el fin de optimizar el almacenamiento y procesamiento de datos, es necesario implementar técnicas de simplificación y compresión de datos [8], lo que permite reducir el tamaño de la información sin comprometer su calidad y utilidad.

Además, otro desafío importante en este contexto es la alta tasa de muestreo de los equipos médicos. Estos dispositivos registran una gran cantidad de datos en períodos de tiempo muy cortos, lo que aumenta exponencialmente la cantidad de información generada. La capacidad de capturar datos a alta frecuencia es esencial para obtener una visión precisa y detallada de la salud del paciente, pero también implica la necesidad de contar con sistemas robustos capaces de manejar y procesar rápidamente estos datos.

II. Fases para el desarrollo de la EPILSERA

El primer paso para desarrollar un dispositivo de predicción o detección de crisis epilépticas implica la comparación y selección de dispositivos comerciales para recolectar señales físicas de alta calidad en pacientes. A priori, no se conocen cuáles son las variables fisiológicas en la muñeca del usuario que

pueden ser transcendentales para la detección de crisis epilépticas. Los dispositivos seleccionados deben proporcionar APIs y SDKs con los cuales se obtengan los datos en crudo. Estos se configuran para la recolección contínua de datos.

En algunos hospitales, se disponen de unidades de Video-EEG. Están formadas por una cámara, un Electrocardiograma (ECG) y Electroencefalograma (EEG). Su principal objetivo es diagnosticar posible epilepsia en pacientes. Cada paciente, durante un periodo de 5 días, realiza vida normal en dicha unidad. En todo momento, el equipo médico monitoriza el estado del paciente y las señales de EEG/ECG , realizando una anotación manual de las crisis que sufren. Esta fase tiene una duración de 3 meses aproximadamente, llegando a obtener datos de 20 pacientes. Una vez que finaliza esta fase, dado que la señales de los dispositivos están sincronizadas, se realiza una comparación entre el EEG, ECG y el resto de señales fisiológicas, con el objetivo de medir la importancia de cada una de ellas para la predicción y detección de crisis epilépticas.

Una vez que se han detectado qué variables fisiológicas cambian antes y durante las diferentes crisis epilépticas, se diseña un dispositivo wearable que integra los sensores capaces de medir dichas señales. Una vez se producen varios dispositivos prototipo, se vuelve a llevar a cabo otra fase de monitorización en la unidad Video-EEG del hospital durante otros 3 mses. En este caso, incorporando también los prototipos.

Al finalizar el segundo período de monitorización en la unidad Video-EEG, se desarrollan modelos de aprendizaje automático utilizando la información de los sensores de los prototipos y la anotación de las crisis con estándar de referencia EEG. Los modelos deben cumplir las restricciones hardware de los prototipos, permitiendo su ejecución contínua y una autonomía superior a 24 horas.

Finalizada la fase de desarrollo y despliegue de modelos de aprendizaje automático, se lleva a cabo otra fase en el hospital, en la cual se comparan las predicciones y detecciones de crisis del prototipo, y las anotaciones del equipo médico. Al final de esta fase se obtiene la especifidad y sensibilidad del dispositivo.

Una vez se ha concluido la fase clínica, se realiza el despliegue en residencias con el objetivo de evaluar el prototipo en situaciones naturales. Durante esta fase, además de predicir o detectar crisis epilépticas, se almacenan los datos de los sensores y se pide a los cuidadores que anoten las crisis epilépticas que se detectan en los pacientes. El principal objetivo es evaluar el prototipo, pero además, actualizar los modelos de aprendizaje automático para que estos estén mejor adaptados a entornos no controlados.

III. Metodología

En esta sección, se describe la metodología utilizada para la monitorización de pacientes en el hospital. Se explican los componentes principales del sistema, las fuentes de datos utilizadas, la resolución de cada fuente, los protocolos de comunicación iniciales y la integración de los datos en una base de datos.

El sistema de monitorización consta de cuatro fuentes de datos principales, que capturan más de 150 variables diferentes relacionadas con el estado fisiológico del paciente y las condiciones ambientales de luminosidad, humedad y temperatura. Estas fuentes de datos se muestran en la Figura 1.

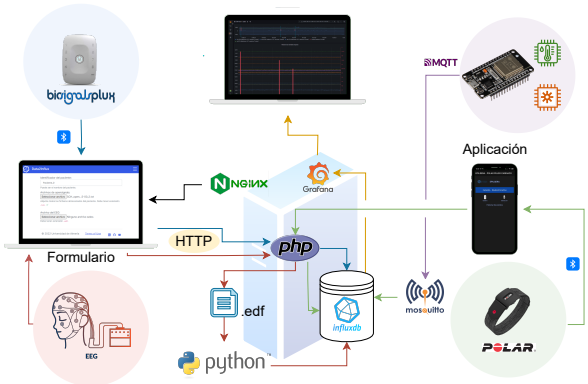


Fig. 1: Esquema general de la arquitectura IoT para la toma de datos.

La integración de todos estos datos es fundamental para identificar las señales relevantes y diseñar la pulsera final. Debido a que se trata de series temporales, donde cada dato está asociado a un *timestamp*, se ha optado por almacenar los datos en una base de datos InfluxDB. Cada medición se etiqueta con el identificador del paciente y la habitación correspondiente. Sin embargo, cada fuente de datos utiliza diferentes tecnologías o protocolos de comunicación, lo que requiere un procesamiento específico para cada una.

A. Electroencefalograma (EEG)

El equipo médico homologado utilizado en el Hospital Universitario Carlos Haya recoge 148 señales cerebrales a una frecuencia de 512 Hz durante dos o tres días de observación del paciente. Estas señales se registran en un archivo `.edf` que se transfiere al ordenador del laboratorio. Posteriormente, el equipo médico analiza las señales y realiza anotaciones en el archivo `.edf` para identificar momentos de crisis.

Debido al volumen de datos generado por el EEG (149 datos por paciente, incluyendo señales y *timestamps*), cada archivo resultante tiene un tamaño mínimo de 98 GB. Para facilitar la transferencia de estos archivos pesados, se ha implementado un formulario que divide el archivo en partes más pequeñas y las une nuevamente en el destino. Para almacenar los datos en InfluxDB, se utiliza código en Python o C++ con las librerías adecuadas para leer archivos `.edf`. En este punto ya se aprecia la necesidad de optimización de esta fuente de datos.

Las anotaciones del EEG son fundamentales para el entrenamiento de la red neuronal encargada de seleccionar las variables fundamentales para detectar una crisis.

B. Biosignals

Se utiliza un kit de investigación comercializado por la empresa PLUX está enfocado a la exploración y medición de señales relacionadas con el estrés. Se ha demostrado la relación entre las crisis y los niveles de ansiedad en los pacientes diagnosticados de epilepsia [9], por lo que se consideró adecuado este kit para la investigación. Cuenta con cinco sensores distintos: un electrocardiograma (ECG); un electromiograma (EMG), que mide la actividad muscular; un sensor de actividad electrodérmica (EDA), un sensor de temperatura corporal y un sensor del nivel de saturación de oxígeno en sangre (SpO_2).

El kit permite cambiar la frecuencia de muestreo de los datos entre 10 Hz y 4000 Hz, pero se establece en 500 Hz para que coincida con la frecuencia del EEG. Los datos se almacenan en un archivo de texto y, para transferirlos al ordenador, se utiliza una conexión Bluetooth. A diferencia del EEG, estos archivos son más manejables en tamaño, pero aún así se dividen en partes más pequeñas para su posterior procesamiento. En este caso, al tratarse de archivos de texto, se pueden procesar directamente en código Php y almacenar en InfluxDB mediante un cliente adecuado.

Los sensores de esta fuente se pueden colocar en la muñeca del paciente, lo que los hace más accesibles y cómodos de usar en comparación con los del EEG.

C. Polar Verity Sense

Se utiliza una pulsera comercializada por la empresa Polar, que cuenta con un sensor de pulso cardíaco, sensor fotopletismográfico, acelerómetro, giróscopo y magnemómetro. Estas variables son relevantes para la detección de epilepsia fuera de los ensayos clínicos [10]. Además, la pulsera proporciona información sobre la actividad motriz del paciente.

Los datos del dispositivo Polar se envía de forma local a un dispositivo Android a través de Bluetooth. Para realizar esto, ha sido necesario desarrollar una aplicación móvil con la API del SDK que proporciona Polar. El dispositivo Android es el encargado de descargar los datos, guardarlos en formato .csv, y enviarlos al servidor centralizado haciendo uso de HTTPS. El dispositivo Polar genera un total de 6 tipos de archivos (pulso cardíaco, aceleración, rotación, valor de campo magnético, fotopletismografía de pulso e intervalo pulso-pulso). Los archivos, en formato `.csv`, se procesan y los datos se almacenan en InfluxDB utilizando un cliente de Php. La frecuencia de muestreo de los datos varía según cada variable, pero generalmente es de aproximadamente 50 Hz.

D. ESP32

Se utiliza un dispositivo ESP32 conectado a la red WiFi para medir la luminosidad, la temperatura ambiente y la humedad. Para medir la luminosidad se utiliza un fotorresistor LDR y para la humedad y temperatura, un sensor de tipo DHT11. Cada medición se publica en tiempo real mediante el protocolo MQTT, y el servidor, que actúa como *broker* y cliente, se suscribe a los datos y los almacena en InfluxDB utilizando un agente de Telegraf. La frecuencia de muestreo puede ajustarse mediante mensajes MQTT. En este caso, la frecuencia de muestreo se establece a 5 Hz.

IV. Resultados y alternativas

En esta sección, se indicará cómo ha ido el desempeño de la arquitectura del diagrama anterior para comprender los problemas y las soluciones que se han llevado o se pretenden llevar a cabo.

A. Integración

La integración de las medidas recogidas por cada fuente se realiza con éxito y se almacenan en la base de datos. Como ejemplo, en la Fig. 2 se ilustra cómo se pueden extraer los datos de un intervalo de tiempo específico de dos fuentes de datos distintas. Específicamente, se muestra la humedad de la habitación obtenida a través de MQTT del ESP32 y la señal del electromiograma (EMG) obtenida del Biosignal.

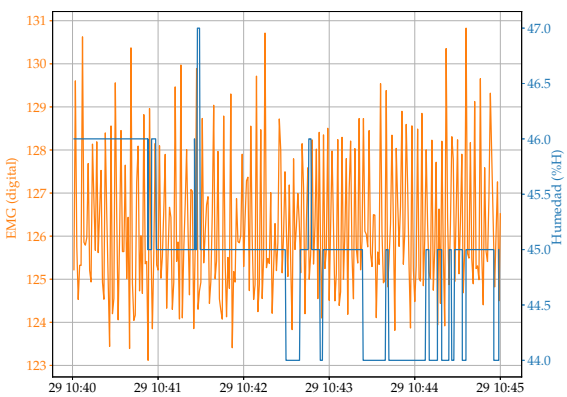
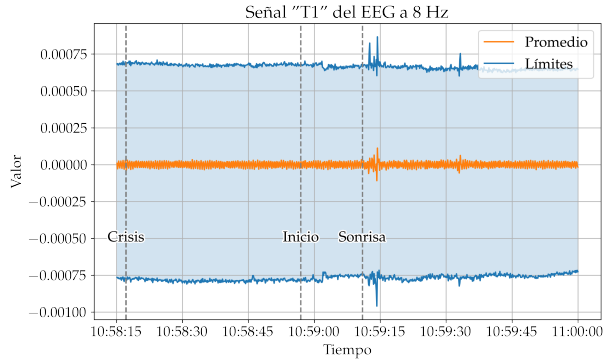


Fig. 2: Integración de señales de fuentes distintas: humedad de la habitación (ESP32) y EMG (Biosignals).

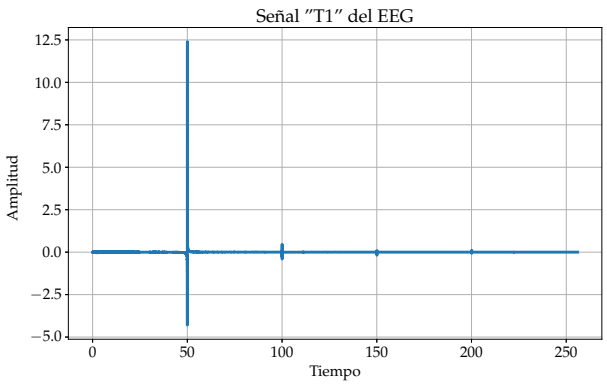
Sin embargo, debido a la gran cantidad de datos de cada fuente, con frecuencias de muestreo de hasta 512 Hz, se requiere simplificar la información para representarla de manera gráfica e intuitiva. En la Fig. 3a, se muestra un ejemplo de cómo se simplifica la señal marcada como "T1" del EEG (referente al lóbulo temporal) mediante un promedio con una frecuencia de 8 Hz, marcando los límites máximo y mínimo de cada tramo. Además, considerando que muchas de las señales son periódicas, la información en el dominio de la frecuencia se vuelve más relevante (Fig. 3b). Por lo tanto, resulta más interesante almacenar los valores de las frecuencias más representativas en lugar de las series temporales completas.

B. Preprocesamiento de las series temporales en GPU

Uno de los grandes problemas a los que se enfrenta la infraestructura de recogida de datos de Epilsera, y que, posteriormente, se trasladará de alguna forma a la propia implementación de la pulsera, es la



(a)



(b)

Fig. 3: Señal T1 del EEG simplificada a 5Hz con las anotaciones del equipo médico (a) y análisis espectral de la señal (b).

enorme cantidad de datos que se recogen por paciente y unidad de tiempo. Como ejemplo, las señales de un ECG o un EEG suelen estar muestreadas a frecuencias superiores a 500Hz, y los propios dispositivos la almacenan en formatos poco compactos (por ejemplo, en modo texto, en base decimal). Así, es bastante normal que la señal de un ECG de un día ocupe 1GB. Sin embargo, en muchas situaciones, es innecesario que la señal esté muestreada a tan alta frecuencia. Por ejemplo, las ondas cerebrales normalmente se clasifican en ondas Delta: con una frecuencia de 0.2-4 Hz, ondas Theta: 4-8 Hz, ondas Alfa: 8-12 Hz, ondas Beta: 12-30 Hz, y ondas Gamma: 30-90 Hz, pero las dos últimas, de mayor frecuencia, no suelen aparecer en periodos de actividad normal. En el caso del ECG, las frecuencias de interés para los cardiólogos son muy bajas, pero se desconoce a priori cuáles serían de interés desde la perspectiva del neurólogo, por lo que se intuye que no hay que descartar la que ofrecen, por defecto, los dispositivos de captura. Otras series temporales, como son, por ejemplo las que proceden de los sensores ambientales no sufren de este problema, e incluso es posible que se aplique un sobremuestreo mediante interpolación.

En cualquier caso, parece importante realizar un análisis en el dominio de la frecuencia a algunas de las señales, usando la Transformada Rápida de Fourier (FFT) para analizar la presencia o ausencia de frecuencias altas en las series temporales y tomar las decisiones de remuestreo, basadas en el resultado. Por ejemplo, si durante las fases de sueño no REM no aparecen ondas beta o gamma, se podrá reducir la frecuencia. Tanto para realizar el submuestreo de la serie como para el cálculo de la FFT, así como la clasificación de las muestras, utilizaremos GPUs. Aún está por determinar el tamaño de la muestra (o quantum), aunque parece razonable, para las arquitecturas actuales de GPUs, que se trabaje con tramas de 1 segundo. Así por ejemplo, si se reduce la frecuencia de muestreo de 512 a 32Hz, cada nuevo dato se calculará mediante la agregación de 8 datos antiguos, y posteriormente ejecutaremos una cuantificación de la serie temporal. La cuantificación es un proceso en el cual se reduce ligeramente la precisión de los datos originales al reemplazarlos por valores discretos o índices enteros dentro de un rango específico. Se cuantificará la serie temporal utilizando el valor medio y la escala. Primero, calculamos el valor medio de un quantum de la serie temporal, que representa una medida de tendencia central. Luego, determinamos la escala, que es la diferencia entre el valor máximo y el valor mínimo de la serie temporal. Después de obtener el valor medio y la escala, se sustituye cada valor de la serie temporal por un índice entero de 8 o 16 bits dentro del rango definido por el valor máximo y mínimo. Esta cuantificación reduce la precisión original de los datos al representarlos con valores enteros, aunque está en el mismo rango de error en el que suelen operar los aparatos de medida.

La aplicación de la Transformada Rápida de Fourier (FFT) también resulta relevante en el contexto del análisis de las señales del Electroencefalograma (EEG), ya que permite examinar los acoplamientos entre estas señales. Los profesionales médicos afirman que cuando el cerebro actúa de manera caótica, el funcionamiento es considerado adecuado. Sin embargo, si las señales del EEG se armonizan, es un indicio de que existe una alta probabilidad de que se produzca una crisis. En la Fig. 4 se puede comprobar un momento de constancia de las frecuencias más dominantes (excluyendo las 3 primeras para simplificar la representación) justo después de la anotación de crisis por parte del equipo médico.

Por lo tanto, la FFT se convierte en una herramienta valiosa para identificar patrones y correlaciones entre las diferentes frecuencias cerebrales, proporcionando información relevante para el monitoreo y la detección temprana de eventos adversos. Por tanto, esta optimización de datos en GPU adquiere un valor trascendental para la investigación.

C. Conectividad

Uno de los principales desafíos en términos de conectividad se relaciona con el nivel de seguridad informática en un entorno hospitalario. En el Hospital Regional Universitario de Málaga, se han implementado tres niveles de seguridad que requieren un proceso de autorización a través de la Unidad de Seguridad de Tecnologías de la Información y Comunicaciones del SAS (USTIC) para solicitar una conexión al servidor (ver Fig. 1). Esta configuración implica ciertas limitaciones, como la imposibilidad de subir

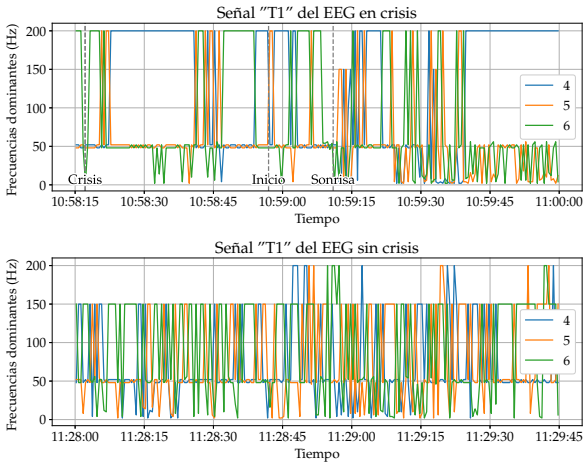


Fig. 4: Comportamiento de la señal "T1" del EEG a lo largo del tiempo en momentos de crisis (arriba) y estabilidad (abajo).

archivos a través del formulario en la intranet del hospital y la conexión WiFi entre el ESP32 y la aplicación Polar en una tablet o smartphone.

Aunque se ha iniciado el procedimiento para solicitar acceso al servidor e incluir los dispositivos en la intranet del hospital, este proceso puede retrasar el desarrollo del proyecto. Como alternativa, se propone que la tablet utilizada para la sincronización con Polar cuente con una tarjeta SIM que proporcione una conexión a Internet con datos ilimitados, permitiendo que la tablet funcione como un punto de acceso para el ESP32. Además, los archivos generados por el EEG y el Biosignals pueden ser transferidos a otro ordenador fuera de la intranet, ya que estos datos pueden ser almacenados de manera offline.

D. Transferencia de datos

Una de las mayores limitaciones de esta arquitectura es la transferencia de archivos por tecnología Bluetooth dado el tamaño de estos y la velocidad de transferencia de esta tecnología. Son precisamente los dispositivos comerciales los que únicamente permiten esta opción por defecto.

Por ejemplo, un archivo generado por el Biosignals en formato `txt` a una frecuencia de 500 Hz y una precisión digital de 16 bits durante 24 horas ocupa 1,82 Gb, y se necesitaron alrededor de 4 horas para transferir toda esa información al ordenador usando Bluetooth 5.0. Por otra parte, el dispositivo Polar genera señales con una frecuencia máxima de 52 Hz. A pesar de que el tamaño de los archivos generados en el dispositivo Polar no es tan grande como en el caso del Biosignals, la comunicación Bluetooth a través de la API del SDK de Polar no está tan optimizada. Al descargar grabaciones largas (de más de 6 horas, sobre 60 MB en el caso de la fotopletismografía), pueden producirse cortes en la comunicación Bluetooth, y en consecuencia, fallar la descarga.

Para solucionar estos problemas se proponen como alternativas:

- Reducir la precisión de los datos de 16 bits a 8 bits, la frecuencia de muestreo o el tiempo de medición de 24 a 12 horas (Biosignals).
- Adquirir el adaptador USB de PLUX par al transferencia de datos a mayor velocidad (Biosignals).
- Realizar una conexión automática del Polar con la tablet cada 2 horas, con el objetivo de descargar grabaciones más cortas, evitando fallos de conexión.

V. Conclusiones

En este estudio, se ha propuesto una arquitectura para la recopilación de datos de pacientes utilizando dispositivos portátiles y tecnologías inalámbricas. La integración de distintas fuentes de datos ha demostrado ser factible, permitiendo obtener una visión holística del estado de salud del paciente a lo largo del tiempo y que servirán para entrenar las redes neuronales con el fin de predecir y detectar las crisis de epilepsia en tiempo real.

El estudio resalta la problemática de la gran cantidad de datos recopilados por paciente y unidad de tiempo en la infraestructura de recogida de datos de Epilsera. Esta sobrecarga de datos plantea desafíos en términos de almacenamiento, transmisión y procesamiento eficiente. Se ha identificado que, en muchos casos, no es necesario muestrear las señales a una frecuencia tan alta. Por ejemplo, en el caso de las ondas cerebrales y las frecuencias de interés para los neurólogos. Para abordar este problema, se propone realizar un análisis en el dominio de la frecuencia utilizando la Transformada Rápida de Fourier (FFT), identificando las frecuencias relevantes en las series temporales y tomando decisiones de remuestreo en consecuencia.

La utilización de GPUs se plantea como una solución para el procesamiento eficiente de las muestras. Se sugiere trabajar con tramas de 1 segundo, aprovechando las capacidades actuales de las arquitecturas de GPU. Esto permite reducir la frecuencia de muestreo y realizar el cálculo de nuevos datos mediante la agregación de datos antiguos. Posteriormente, se lleva a cabo una cuantificación de la serie temporal, en la cual se reduce ligeramente la precisión de los datos originales mediante la sustitución de valores por índices enteros dentro de un rango definido por el valor máximo y mínimo.

Estos avances demuestran el potencial de las tecnologías inalámbricas y los dispositivos portátiles para la monitorización remota de pacientes, superando obstáculos técnicos y mejorando la eficiencia en la recopilación y transferencia de datos. El enfoque propuesto, combinando el análisis en el dominio de la frecuencia y el uso de GPUs, optimiza el almacenamiento, transmisión y análisis de las señales, lo que impulsa el entrenamiento de redes neuronales para la predicción de la epilepsia y su ejecución en tiempo real. Estos avances tienen un impacto significativo en la atención médica y la calidad de vida de los pacientes.

Agradecimientos

Este trabajo ha sido financiado por los proyectos R+D+i PID2021-123278OB-I00 y PDC2022-133370-I00, del MCIN/AEI/10.13039/501100011033/; proyecto del Ministerio de Ciencia y Tecnología de España con el PID2019-105396RB-I00 y FEDER "Una forma de hacer Europa". M. Lupión es beneficiario del programa FPU del Ministerio de Educación (FPU19/02756).

Agradecemos al Hospital Regional Universitario de Málaga por aceptar nuestro proyecto de investigación y por su colaboración invaluable. También queremos agradecer al equipo médico de Neurología por su apoyo en la monitorización de los pacientes y el acceso a los dispositivos médicos necesarios.

Referencias

[1] World Health Organization, "Epilepsy," `https://www.who.int/news-room/fact-sheets/detail/epilepsy`, Accessed: 2023-05-29.

[2] Ettore Beghi, "The Epidemiology of Epilepsy," *Neuroepidemiology*, vol. 54, no. 2, pp. 185–191, 2020.

[3] Christian E. Elger and Christian Hoppe, "Diagnostic challenges in epilepsy: seizure under-reporting and seizure detection," *The Lancet Neurology*, vol. 17, no. 3, pp. 279–288, 2018.

[4] Jerry J. Shih, Nathan B. Fountain, Susan T. Herman, Anto Bagic, Fred Lado, Susan Arnold, Mary L. Zupanc, Ellen Riker, and David M. Labiner, "Indications and methodology for video-electroencephalographic studies in the epilepsy monitoring unit," *Epilepsia*, vol. 59, no. 1, pp. 27–36, Jan. 2018.

[5] A. Ulate-Campos, F. Coughlin, M. Gaínza-Lein, I. Sánchez Fernández, P. L. Pearl, and T. Loddenkemper, "Automated seizure detection systems and their effectiveness for each type of seizure," *Seizure*, vol. 40, pp. 88–101, 2016.

[6] Sándor Beniczky and Philippe Ryvlin, "Standards for testing and clinical validation of seizure detection devices," *Epilepsia*, vol. 59, no. S1, pp. 9–13, 2018.

[7] Mohammad S. Al-kahtani, Faheem Khan, and Whangbo Taekeun, "Application of internet of things and sensors in healthcare," *Sensors*, vol. 22, no. 15, 2022.

[8] C. J. Deepu, C. H. Heng, and Y. Lian, "A hybrid data compression scheme for power reduction in wireless sensors for iot," *IEEE transactions on biomedical circuits and systems*, vol. 11, no. 2, pp. 245–254, 2017.

[9] Lisa-Dounia Soncin, Sylvane Faure, Aileen McGonigal, Tatiana Horowitz, Sara Belquaid, Fabrice Bartolomei, and Eric Guedj, "Correlation between fluorodeoxyglucose positron emission tomography brain hypometabolism and posttraumatic stress disorder symptoms in temporal lobe epilepsy," *Epilepsia*, vol. 63, no. 7, pp. e74–e79, Jul 2022.

[10] Benjamin H. Brinkmann, Philipp J. Karoly, Ewan S. Nurse, Sonya B. Dumanis, Maryam Nasseri, Pedro F. Viana, Andreas Schulze-Bonhage, Dean R. Freestone, Gregory Worrell, Mark P. Richardson, and Mark J. Cook, "Seizure diaries and forecasting with wearables: Epilepsy monitoring outside the clinic," *Frontiers in Neurology*, vol. 12, pp. 690404, Jul 2021.

[11] M. R. Ramya and A. Umamakeswari, "A survey on data compression techniques for internet of things in healthcare applications," *Wireless Personal Communications*, vol. 118, no. 3, pp. 1637–1660, 2021.

FancyJCL: A High Level Approach for Parallel Development in Mobile Devices

Sergio Afonso[1], Óscar Gómez–Cárdenes[2], Paula Expósito, Vicente Blanco y Francisco Almeida

Resumen— **Mobile devices and handheld systems such as smartphones and tablets are universally extended. State-of-the-art heterogeneous architectures composed of multi-core processors and some kind of accelerator, as GPUs or DSPs, usually constitute their basic hardware configuration. Specific code adapted to the architecture is mandatory if high-performance computation is required and low-level libraries and parallelism are mandatory, constituting an important barrier for the usual developer in such devices. In this context, we propose the FancyJCL framework, which provides a high abstraction layer that hides implementation details and allows to develop parallel programs for mobile devices. The target platform for FancyJCL is mainly Android, and Java developers, due to high market penetration. A very simple, apparently sequential encoding produces parallel efficient OpenCL code. FancyJCL is in turn based on the Fancier framework, which enables optimal memory management across memory spaces on unified memory systems. Benchmarks for FancyJCL developed code over a high range of image processing algorithms show good performances to a low development effort.**

Palabras clave— **Heterogeneous systems, Image processing, Mobile computing, Parallel programming, Performance analysis.**

I. Introduction

Since public adoption of mobile platforms is so pervasive and their unused performance potential is so high, it is important to consider compatibility with these platforms in the design of new parallel programming models and tools. The vast majority of modern mobile devices are running the Android or iOS Operating Systems (OS), the former being much more widespread. The development of native Android applications is mainly done in the Java and Kotlin programming languages, both compiled into Java bytecode and seamlessly interoperating and running on top of the same runtime. Using this type of managed high-level programming languages simplifies the development of interactive applications, at the cost of adding overhead that hinders execution performance. The potential of providing parallel and accelerated execution capabilities to Java-based applications has been explored for several years, evidenced by all the work around Java Grande [1], or using Java for large scale parallel applications. However, we believe that, even though the programmability problem of creating these types of Java applications has not been completely solved, the demands of the mobile sector make this issue more relevant than ever. Typically, managed programming languages, such as Java, tend to run slower than native ones, like C/C++, due to the overhead of their runtime systems and their higher level of abstraction from the hardware and OS. However, some of these shortcomings have improved over time due to improvements in Virtual Machine optimizations and Just-in-Time (JIT) compilation techniques [2].

Nevertheless, Java execution on accelerators still requires the use of specialized Java Virtual Machines (JVMs) or libraries, or code translation tools integrating Java bytecode execution and native libraries with low-level access to accelerators. As a consequence, the programmability effort is still very high. Alternatives such as language bindings like JCUDA [3] or JOCL [4] are targeted towards experts and do not reduce the programming effort significantly. Multiple approaches exist to tackle the issue of accelerating Java applications trying to reduce the programming complexity. Projects Sumatra [5] and TornadoVM [6] propose implementation features or extensions to JVMs and standard libraries that require reduced effort from developers. Although working at the JVM level has the benefit of being able to decide at runtime, through a JIT compilation process, the processor to target for each task or a dynamic task migration, not always is possible to rely on the use of custom JMVs such as in Android devices.

Tools like Paralldroid [7], Aparapi [8], Rootbeer [9], or ParallelME [10], on the other hand, can work on standard JVMs, and they can translate Java code or compiled bytecode into native or accelerated implementations. Each of these alternatives define their own parallel programming model on top of Java, adding certain restrictions over what Java code is supported for parallel or accelerated execution. This makes it difficult to port parallel code written for one of these environments to another, and most of them have to solve a similar set of challenges, mainly relating to memory management, on top of making their particular parallel programming model work efficiently. Despite these efforts, none of the existing approaches has been widely adopted by the developer or scientific community, so there does not exist a standard system implementing hardware acceleration of Java code yet.

In the Fancier framework [11], we proposed a common Java API that simplifies the automatic production of efficient native code for acceleration, which can be used as a platform to build independent automatic acceleration tools. This API, built on top

[1]Dpto. de Ingeniería Informática y de Sistemas, Universidad de La Laguna, e-mail: safonsof@ull.edu.es

[2]Dpto. de Ingeniería Industrial, Universidad de La Laguna, e-mail: ogomezca@ull.es

of the OpenCL 1.1 standard libraries, includes fixed-size vector data types, a math library, and multiple containers for primitive data types and images, and it is designed to emulate value semantics used in native languages, allowing a simpler mapping of Java code to C/C++ or OpenCL for accelerators. Fancier provides containers that feature transparent zero-copy memory support on unified memory systems, while giving direct access to the memory from the Java, native, and OpenCL contexts. This is especially relevant on mobile SoC where memory copy overhead can easily become a performance bottleneck. The approach does not demand any modification to the Java compiler or the JVM, and it only depends on OpenCL.

In this paper we present the FancyJCL framework[1], we propose a high abstraction layer that hides implementation details when developing parallel Java programs for mobile devices. Genericity, flexibility and efficiency are basic issues of the design strategy. The parallelism is provided in a transparent manner through a sequential interface, so that, sequential users may access to the parallel system. Our proposal is close to Aparapi [8], however, they operate at the bytecode layer what may generate dependences on de VMs and could penalize the efficiency. FancyJCL is based on the Fancier framework and provides a set of "sequential" classes, methods and data structures that encapsulate the access to the parallel context, providing easy development and fast prototyping. No previous knowledge is needed about JNI and OpenCL since all the glue code for JNI and OpenCL is wrapped into FancyJCL. A very simple, apparently sequential encoding, produces parallel efficient OpenCL code. This ease of programming increases the development productivity, improves the delivery of new parallel applications due to the rapid development time and contributes to the efficient exploitation of new emerging architectures. FancyJCL also benefits from some of the Fancier advantages, since OpenCL has been chosen as the hardware acceleration backend, support for general-purpose computations and widespread adoption among all types of architectures, from low-power SoC to state-of-the-art high-performance computing accelerated distributed systems is provided. The efficient memory management avoids unnecessary copies among the different memory spaces and increases the performance. The decoupled design of the FancyJCL classes is introduced with no loss of efficiency. Benchmarks of FancyJCL generated code show good speedups and ease of use on different mobile devices tested over a wide range of representative image processing kernels.

This paper is structured as follows: Section II introduces the process by which Android applications are compiled and executed, Sect. III presents a summary of the Fancier framework and its advantages, and Sect. IV describes in detail the building of FancyJCL on top of Fancier, Sect. V includes the performance analysis developed to validate our approach, and in Sect. VI we give our conclusions and future related lines of work.

[1] https://github.com/HPC-ULL/FancyJCL

II. Application Execution on Android

Managed Execution: Android native applications are mainly developed using the Java and Kotlin managed programming languages, providing simple development environments with a reasonable performance. Both are compiled into Java Bytecode, allowing the interoperability of those languages transparently. However, their higher level of abstraction comes with a performance cost over native programming languages. In order to mitigate that, since Android 5.0, Java Bytecode is transformed and optimized in various stages until it gets ahead-of-time (AOT) compiled into binaries that run natively with help from the Android Runtime (ART) [12], as shown in Figure 1. More recent Android releases use a hybrid just-in-time (JIT) and AOT profile-guided optimization process [13]. Its performance in many cases is still far from that of the Native code.

Native Execution: Java applications can interface with native code by using the Java Native Interface [14] (JNI), also supported by Android [15]. The native implementation of the me-thod requires some JNI API calls as a bridge between Java and C to the actual, adding some development complexity and execution overhead. As an advantage, it allows to link and use Android-supported native system libraries (OpenGL, Vulcan, ...) or any other native independent library, and native existing code can be integrated instead of re-implementing it in Java or Kotlin. Compute or memory intensive codes can be accelerated.

Accelerated Execution: The acceleration of regular Android applications is possible through the use of the Renderscript language [16]. Conversely, OpenCL is a multi-platform standard for general-purpose accelerated execution that has existed for longer than Renderscript, and it has enjoyed continued support by most of the main SoC vendors at the core of modern smartphones, such as Arm or Qualcomm. Even though this alternative has not seen official support from Android, its widespread adoption together with its higher performance potential and finer-grained control makes it an interesting alternative. Additionally, the officially supported framework for Android acceleration from version 12 is Vulkan instead of RenderScript, for which work is being done to allow running OpenCL kernels [17]. OpenCL is our best option as this work is focused on general-purpose cross-platform acceleration.

The main disadvantage of OpenCL or Vulkan compared to Renderscript is that they demand a higher development effort to integrate into a managed application written in Java or Kotlin and a steeper learning curve. This is why approaches like those presented in this paper are still relevant and necessary.

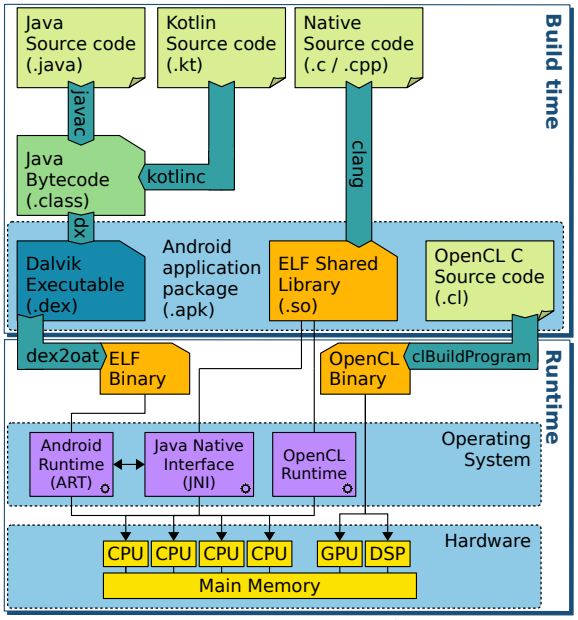


Fig. 1: Compilation and execution of an Android application.

III. The Fancier Framework

A. Overview

The Fancier framework [11] is a multi-platform, modular, and extensible programming interface to facilitate the acceleration of Java applications without requiring any special features from the JVM they are running in. Fancier uses OpenCL as the high-performance multi-platform backend for accelerated execution. It targets Linux and Android platforms and defines a Java and Native API modeled after the OpenCL C programming language for accelerators. Fancier provides several utilities that simplify the integration of C/C++ and OpenCL execution to a Java application efficiently. There are two main features from Fancier that have been decisive in building FancyJCL:

- Fancier defines a Java and C/C++ subset of features and functions where the mismatch between reference semantics of Java and the value semantics of C/C++ and OpenCL is addressed. Fancier forces value semantics by returning a new result object when operating. By preallocating objects for all intermediate results the pressure on the Java memory management is reduced. The Fancier unification of the Java, Native, and OpenCL languages greatly reduces development costs and has no performance penalty. Fancier facilitates the translation of a standardized Java subset to C/C++ and OpenCL for accelerated execution into a Java application, and their runtime integration by creating a unified interface and library.
- The information flow between Java, C/C++ and OpenCL is not easy and requires manual management. There exist multiple ways of performing this, but most incur performance penalties that may go unnoticed. Fancier implements optimal strategies for container data types that work transparently and significantly reduces development cost. Unified memory systems introduce the possibility of sharing memory buffers between a host processor and accelerators, which can provide great performance gains. However, the optimal management of this type of memory becomes difficult when buffers must be accessed from managed, native, and accelerated contexts. The Fancier memory management strategy features efficient and transparent zero-copy read and write access from all contexts.

Accelerated Java applications using Fancier for OpenCL execution are able to take full advantage of all the features of the OpenCL 1.1 standard, with the advantage that passing data between the Java, C/C++ and OpenCL layers is much simpler, and the control flow between Java and C/C++ is more seamless. As a result, Fancier enables the design and implementation of parallel accelerated programming models on top of Java, and high-performance Fancier Native or OpenCL C code can be easily generated by automated transformation tools.

B. Fancier: APIs and Memory Management

The Fancier framework provides three layers, the Fancier Java API, the Fancier Native API and the Fancier OpenCL API. The three layers implement all the functionality to allow an algorithm to be expressed in a similar way in Java, C++ or OpenCL so its translation from and to any of them is trivial. Arrays, vectors of sizes 2, 3, 4 and 8 and basic types `Byte`, `Short`, `Int`, `Long`, `Float`, and `Double` are supported in the three layers. All Fancier containers are allocated through OpenCL API calls. The provided functions for managing buffers are much simpler than manually writing JNI and OpenCL API calls. The Fancier Native API allows to obtain and compile OpenCL C kernels at runtime that add math functions supported by Fancier Java and Fancier Native which are not included in the OpenCL C standard library. In the case of `DirectBuffer` Java Objects, the native address is used by Fancier to provide zero-copy operations.

Data transfers between contexts have great performance implications. An OpenCL-accelerated Java application deals with three independent memory spaces: the Java-managed heap, the OS-managed native heap, and the OpenCL device memory. Fancier handles automatically the memory buffers efficiently taking advantage of the unified memory avoiding unnecessary copies. The Fancier functions also manage memory synchronizations. Moreover, Fancier includes methods to get the `ByteBuffers` and copies of their content when be assumed that the copies will perform more efficiently than the direct use of the `ByteBuffer`. In summary, Fancier enables a transparent use of the unified memory architecture, avoids unnecessary memory movements and allows read and write data from the three layers: Java, Native and OpenCL.

IV. The FancyJCL Framework

A. Overview

The FancyJCL framework is a high-level tool oriented to Java developers willing to accelerate their applications on mobile devices. Core ideas such as ease of use, genericity, flexibility and efficiency are basic goals of the design strategy in FancyJCL. A high-level sequential abstraction layer is provided so that users without any expertise in native or parallel programming can use it in a simple way. FancyJCL collects a package of *sequential* classes, methods and data structures that encapsulate the access to the native and parallel contexts allowing easy development and fast prototyping. No previous knowledge is needed about parallelism, JNI or OpenCL since all the glue code for JNI and OpenCL is wrapped into FancyJCL. A very simple, apparently sequential encoding, produces parallel efficient OpenCL code. Since FancyJCL is based on Fancier, is also multiplatform, extensible and does not require any special features from the JVM they are running in, so the generated parallel code can be executed into a mobile device but also on server or desktop computers.

Genericity appears naturally since OpenCL is the hardware acceleration backend, with support for general-purpose computations, fine-grained hardware control, and widespread adoption among all types of architectures. The object-oriented design makes of FancyJCL a flexible tool that can be easily extended to new scenarios. Optimizations techniques as vectorization or tiling could be introduced in a transparent manner. The Fancier parallel structures are decoupled from the FancyJCL classes so no loss of efficiency is introduced by the approach. The zero-copy memory support provided by Fancier avoids unnecessary copies among the different memory spaces increasing the performance. Improved application quality, increased use of parallel architectures by non-expert users, rapid inclusion of emerging technology benefits into their systems, dynamic and transparent optimization enhancements are some of the benefits that can be reached.

B. Design

Figure 2 shows a general overview of the FancyJCL architecture. FancyJCL builds a layer on top of Fancier that encapsulates the Fancier initialization, the glue code necessary for the JNI and OpenCL generation, the mapping to Fancier data structures and objects, the generation of the functional Kernel including inputs and outputs, and also the full OpenCL program and the synchronization among the different contexts. Some of these actions are supported by Fancier and the FancyJCL interface just wraps up and adapts the user calls but, in some others, more elaborated extra code is added internally in FancyJCL to generate new code or before a call to the Fancier framework be performed.

Figure 3 goes deeper into the architecture and shows a set of representative FancyJCL classes, objects and methods. The FancyJCL environment

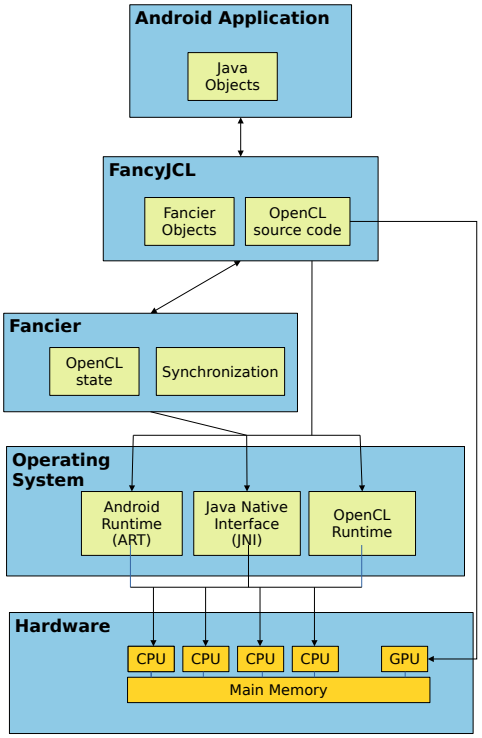


Fig. 2: Overview of the FancyJCL architecture.

is initialized through a call to the `initialize()` method of the `FancyJCLManager` static class. This initializes the FancyJCL and Fancier contexts at the same time, creates the JNI and the OpenCL context and the OpenCL queues. The list of namespaces and attributes that will be passed through the different contexts is created.

The FancyJCL `Stage` class is designed to create execution units that manage the information about an algorithm, its parameters and the configuration run, i.e., the range of the index variables and how many GPU threads to use among others. Objects declared of the `Stage` class are intended to run a parallel process over input data objects to produce output data objects. The `setInputs` and `setOutputs` methods of a `Stage` object will manage the corresponding java objects. FancyJCL converts them into Fancier objects, if the declared object belongs to the DirectBuffer class (or a derived from it) only pointer reassignments to the Fancier objects will be needed and there will not be data copies among the different contexts. This has an important impact in the overall performance. Otherwise, a copy is performed into a Fancier object and from that moment on, no additional copies among the different contexts are needed. Even though, only one data copy is performed. The parallel procedure is assumed as a simplified kernel, defining the operation to be applied over the input and output data, passed through the `setKernelSource` method. Predefined variables $d_0, ..., d_k, ..., d_{n-1}$ manage the index on the $k-th$ dimension, this eases and encapsulates the call to the `get_global_id(k)` OpenCL kernel function. Before an execution is launched the length

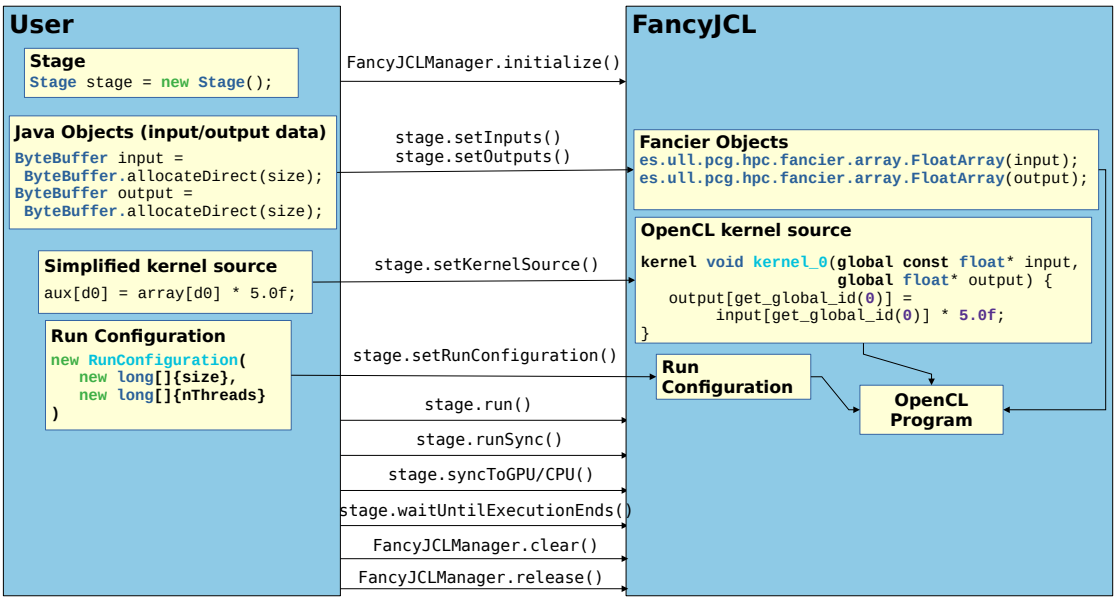


Fig. 3: FancyJCL use case diagram.

and number of threads to use per dimension must be fixed (`setRunConfiguration()`). This method calls to the OpenCL functions to create the OpenCL kernel. The `Stage` class provides methods to run the kernel, including synchronous blocking and asynchronous executions of several kernels, and the corresponding methods to ensure the synchronization among the different contexts. An attribute list created at `FancyJCLManager` keeps a list of references to the `Stage` which addresses the attributes. This allows an output to be the input on the same or different kernel.

C. FancyJCL Example

An example illustrating the use of FancyJCL appears in figures 3 and 4 where a simple code that computes the average of two buffers is shown. The flow control follows the steps previously described. No native code is included by the user into their application nor glue code. Every declaration, definition and method call adopts the natural coding style in Java. The user only needs to know two FancyJCL classes, `FancyJCLManager` and `Stage` and some of their methods. The only different concern is the notion of simplified kernel. The user must understand how to express the computation of an input data item to produce the output. This is straightforward if the user is familiar with the CUDA or OpenCL streaming programming model but, in any case, it involves a quite simple semantic and does not require too much effort from the user side. Some attention must also be paid by the user to the synchronization of the results. This piece of program (about 20 lines) encapsulates a code of about 56 lines.

Another important element that usually goes unnoticed is the effort involved in the cross-compilation for this case where java and native code are mixed. Since FancyJCL is delivered in a precompiled package, the extra compilation stage and the CMake management are removed from the development cy-

```
1  // Initialize FancyJCL
2  FancyJCLManager.initialize(
3                 getApplicationContext()
4                     .getCacheDir()
5                     .getAbsolutePath()
6                 );
7  // Memory allocation
8  ByteBuffer input0 = ByteBuffer
9                      .allocateDirect(size);
10 ByteBuffer input1 = ByteBuffer
11                     .allocateDirect(size);
12 ByteBuffer output = ByteBuffer
13                     .allocateDirect(size);
14 // or
15 // int[] input0 = new int[size];
16 // int[] input1 = new int[size];
17 // int[] output = new int[size];
18
19 // Constants declaration
20 float k0 = 0.3f;
21 float k1 = 0.7f;
22
23 // Create an execution Unit
24 Stage stage = new Stage();
25
26 // Set inputs and outputs
27 stage.setInputs(Map.of("input0",input0,
28                        "input1",input1,
29                        "k0", k0, "k1", k1));
30 stage.setOutputs(Map.of("output", output));
31
```

```
32 // Create a simplified Kernel
33 stage.setKernelSource("""
34   // Get pixels and multiply by constant
35    float p0 = input0[d0] * k0;
36    float p1 = input1[d0] * k1;
37    // Store pixel
38    output[d0] = (int) round(p0 + p1);
39   """);
40
41 // Fix size elements in the first dimension
42 // and numThreads threads to be launched
43 stage.setRunConfiguration(
44   new RunConfiguration(new long[]{size},
45   new long[]{numThreads}));
46
47 // Run the kernel and synchronize
48 // data among the contexts
49 stage.runSync();
50
51 // Clear the FancyJCL state
52 FancyJCLManager.clear();
53
```

Fig. 4: Computing in FancyJCL a weighted average of elements of two buffers and storing it in a third buffer.

	Xiaomi Mi Mix 2	Snapdragon 865 HDK	Vivo iQOO 7
OS	Android 9	Android 10	Android 11
SoC	Qualcomm Snapdragon SD845	Qualcomm Snapdragon SD865	Qualcomm Snapdragon SD888
CPU	4 Kryo 385@1.7GHz + 4 Kryo 385@2.8 GHz	4 Kryo 585@2.1GHz + 3 Kryo 585@2.42 GHz + 1 Kryo 585@2.84GHz	4 Kryo 680@1.8GHz + 3 Kryo 680@2.42 GHz + 1 Kryo 680@2.84GHz
GPU	Adreno 630@670-710 MHz	Adreno 650@250-587MHz	Adreno 660@700MHz
RAM	6GB LPDDR4x@1866MHz	6GB LPDDR5@2750MHz	8GB LPDDR5@3200MHz
Year	2017	2019	2021

Tabla I: Hardware platforms.

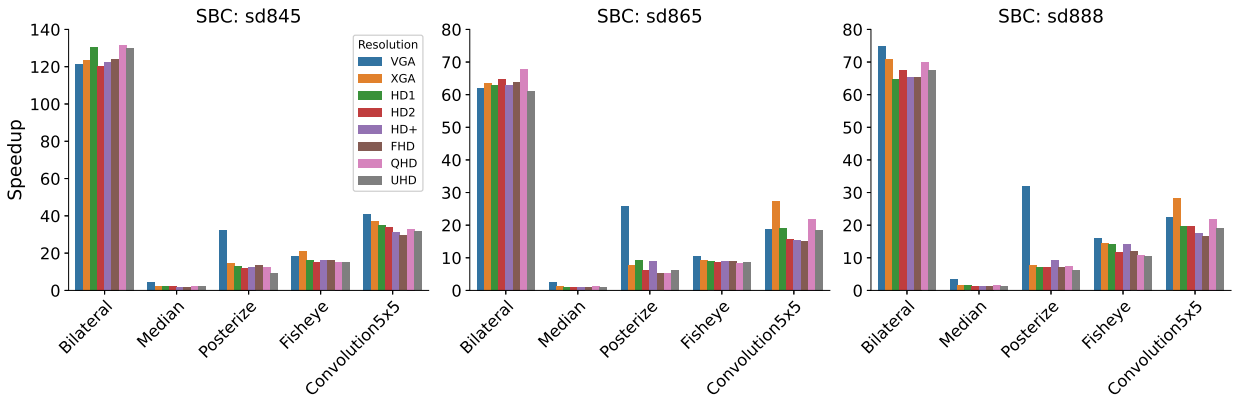


Fig. 5: Speedups of the Java averaged execution time by the FancyJCL equivalent implementation of 5 groups of kernels with 8 resolutions. One subplot per SoC.

cle. The compilation is faster and the required knowledge to do it is lower.

V. Evaluation

Three Android devices have been used in order to evaluate the performance improvements of the FancyJCL API over Java reference implementations (Table I). Their results are representative of a range of recent SoC present in modern Smartphones. Two of the devices are off-the-shelf products and the other is an HDK (mobile Hardware Developement Kit). FancyJCL takes advantage of two main key aspects: the existence of a GPU that is programmable using OpenCL and the unified memory model that allows for implementing zero-copy strategies when executing code in CPU and GPU contexts.

Several image filter kernels have been developed in order to evaluate the performance of the FancyJCL library. The kernels have been programmed naively, meaning that no parallel strategy has been applied to the FancyJCL implementation such as loop unrolling, vectorization or sharing memory in a work group (local memory). The only strategy followed is to set a high global workgroup size. The image processing kernels have been implemented using a 32-bit RGBA pixel format. The set of kernels implemented is the following:

- **Bilateral**: An edge-preserving smoothing filter. It is a stencil code, each neighbor is weighted according to its color and distance to the center pixel.
- **Median**: A median filter, it is a stencil code that evaluates the neighbors of each pixel within a given radius and applies the median intensity of these input pixels to the output. Our implementation only uses the red pixel channel, producing a gray scale output.
- **Posterize**: A filter that applies pre-selected colors to given ranges of input pixel intensity. It is a pixel-wise kernel that is called multiple times, one per range. We use five ranges in our testing.
- **Levels**: A pixel-wise kernel that applies a saturation and contrast levels change to an image.
- **Fisheye**: A distortion kernel which applies a fisheye lens effect to an image. The coordinates of each pixel are transformed using several math functions, and the resulting output pixel is calculated through a bilinear interpolation of these transformed coordinates.
- **Contrast**: A pixel-wise kernel that applies a parameterized contrast enhancement of an image.
- **Convolution 5x5; 3x3**: Convolution kernels using a 5x5 or 3x3 mask, implemented without loops.
- **GrayScale**: A pixel-wise kernel that converts a color image to gray scale.
- **GaussianBlur**: A smoothing filter based on the Gaussian function. It is implemented as two successive passes where one applies the filter considering the horizontal neighbors and the other considers only the vertical neighbors of each pixel. This reduces computation and provides the same result as a single-kernel variant due to it being separable.

Each of these kernels has been implemented in Java and FancyJCL, the Java reference implementation uses Java arrays (`byte []`) to fetch, process, and write pixel data and the FancyJCL is a parallel OpenCL implementation using `DirectBuffer` Java Buffers and taking advantage of the FancyJCL library.

Due to the highly dynamic performance charac-

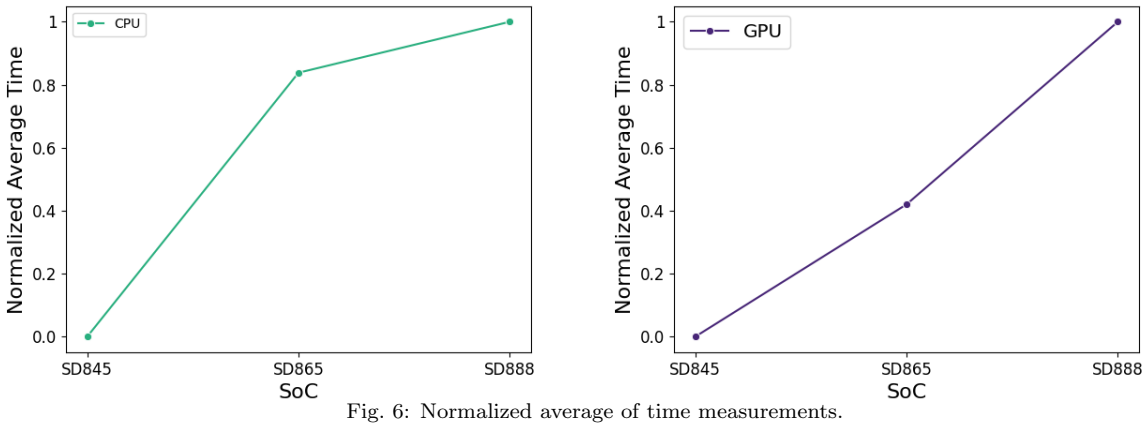


Fig. 6: Normalized average of time measurements.

teristics of these SoC and because to the power and thermal constraints they have, the testing algorithms must be repeated enough times so the standard deviation is low enough for the average to be considered a valid measurement. Also, in the multiple-context application such as the use-case of executing in the GPU, the benchmarking process must be aware of when to synchronize with the CPU. If `DirectBuffer` Objects are used in an architecture that implements unified memory, the synchronization of input and output memory buffers is not needed. In order to achieve fairness, the measurement process in GPU implementations is the following:

- Start measuring time
- Enqueue execution of the algorithm on GPU `N` times
- Wait until the OpenCL queue finishes
- End `time_measurement`
- Return `time_measurement / N`

The class `FancyJCL.Benchmark` allows for executing a benchmark a number of times and includes a synchronization prologue that will only be executed once, but included in the measured time. The benchmarking application was compiled in release mode, producing native binaries in arm64-v8a mode (64-bit). The implementation of the image-processing kernels was evaluated over the following set of image inputs of different sizes: **VGA** (640 × 480); **XGA** (1024 × 768); **HD1** (1280 × 720); **HD2** (1366 × 768); **HD+** (1600 × 900); **FHD** (1920 × 1080); **QHD** (2560 × 1440); **UHD** (3840 × 2160). The number of executions was variable for each experiment. A fixed execution time of at least 30 seconds was set as `experiment_time`. The number of executions `N` is therefore the amount needed to take the `experiment_time`. In all cases `N` was greater than 10 and the measured standard deviation was lower than `0.01`.

Table II shows the averaged execution times and speedups obtained from the computational experience developed. Figure 5 shows the speedups for a selected group of kernels on each SoC. For almost every experiment, the average execution time of the FancyJCL version was lower than its Java reference implementation. The average speedup is 23. This means that by using this library not only a significant increase in speed is gained, but also the complexity of the code is greatly reduced, by hiding all the OpenCL library calls and the Native C++ code. The worst case is the **Mean** filter, since it fetches a large number of pixels to be sorted, a local memory strategy should be used to achieve a significant speedup. The best case is the **Bilateral** filter, where most of the computational cost lies on floating point mathematical operations such as multiplication and power because their Adreno implementations are much faster.

The speedups achieved in the SoC SD845 are about twice of those of the SD865 or SD888. This is explained since the CPU and the GPU did not evolve at the same rate between SD845 and SD865 while they did between SD865 and SD888. The difference in performance between the CPU and GPU is greater for the SD845 than the SD865 or the SD888. This can be better appreciated using our data in the Figure 6 where the normalized execution times are compared for the three SoC and for CPU and GPU. The GPU evolution behaves almost linearly while the CPU evolution behaves exponentially.

VI. Conclusion

We have presented FancyJCL, a framework providing easy development and fast prototyping to generate parallel OpenCL codes in mobile devices. A set of Java classes is provided to the developer that must fill code gaps in the apparently sequential methods of these classes. The set of classes made available to the user is quite reduced so the learning curve is very low and the parallelism is offered in a transparent manner. Genericity and efficiency are guaranteed through the internal use of the Fancier Library. Fancier constitutes a promising backend for parallel programming models built on top. It is not constrained by any particular JVM and solves efficiently the problem of data movement among the various memory spaces involved in parallel computations in a mobile device. The ease of use and efficiency of FancyJCL is validated by benchmarking a wide range of representative image processing kernels over three different SoC architectures. The flexibil-

ity of the framework is inherent to the design of FancyJCL, for example, natural extensions to FancyJCL could provide support for Image2D, vectorization or autotuning just by extending the `RunConfiguration` method. More elaborated extensions would allow to generate OpenGL or Vulcan code, that are supported by a larger number of vendors, or generate code for a different language as JavaScript to cover a greater community of mobile developers.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia e Innovación de España con los proyectos PID2019–107228RB–I00, TED2021–131019B–I00 y PDC2022–134013–I00; y por el Gobierno de Canarias con el proyecto ProID2021010012.

Referencias

[1] L.A. Smith, J.M. Bull, and J. Obdrizalek, "A parallel java grande benchmark suite," in *SC '01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, 2001, pp. 6–6.

[2] Shiv, K. et al, "Impact of JIT/JVM optimizations on JAVA application performance," in *7th Workshop on Interaction Between Compilers and Computer Architectures (INTERACT-7), 2003.*, 2003, pp. 5–13, DOI: 10.1109/INTERA.2003.1192351.

[3] Yonghong Yan, Max Grossman, and Vivek Sarkar, "JCUDA: A programmer-friendly interface for accelerating Java programs with CUDA," in *Euro-Par 2009 Parallel Processing*, Henk Sips, Dick Epema, and Hai-Xiang Lin, Eds., Berlin, Heidelberg, 2009, pp. 887–899, Springer.

[4] Marco Hutter, "JOCL: java bindings for opencl," 2022.

[5] OpenJDK, "Project Sumatra," `https://openjdk.java.net/projects/sumatra/`.

[6] Fumero, J. et al, "Dynamic application reconfiguration on heterogeneous hardware," in *15th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environments*, NY, USA, 2019, p. 165–178, ACM, DOI: 10.1145/3313808.3313819.

[7] Alejandro Acosta, Sergio Afonso, and Francisco Almeida, "Extending Paralldroid with object oriented annotations," *Parallel Computing*, vol. 57, pp. 25 – 36, 2016, DOI: 10.1016/j.parco.2016.04.003.

[8] Aparapi, "API for data parallel Java," `http://aparapi.com/`.

[9] Philip C Pratt-Szeliga, James W Fawcett, and Roy D Welch, "Rootbeer: Seamlessly using GPUs from Java," in *9th HPCC-ICESS Conference*. IEEE, 2012, pp. 375–380.

[10] Andrade, G. et al, "ParallelME: A parallel mobile engine to explore heterogeneity in mobile computing architectures," in *Euro-Par 2016: Parallel Processing*, Pierre-François Dutot and Denis Trystram, Eds., Cham, 2016, pp. 447–459, Springer.

[11] Sergio Afonso and Francisco Almeida, "Fancier: A unified framework for java, c, and opencl integration," *IEEE Access*, vol. 9, pp. 164570–164588, 2021, DOI: 10.1109/ACCESS.2021.3134788.

[12] Android Open Source Project, "ART and Dalvik," 2020.

[13] Android Open Source Project, "Implementing ART just-in-time (JIT) compiler," 2020.

[14] Sheng Liang, *The Java native interface: programmer's guide and specification*, Addison-Wesley Professional, 1999.

[15] Android Open Source Project, "Get started with the NDK," 2020.

[16] Android Open Source Project, "Renderscript overview," 2021.

[17] Google, "clspv," 2021.

		sd845			sd865			sd888		
Filter	Res.	Java	FancyJCL	SpeedUp	Java	FancyJCL	SpeedUp	Java	FancyJCL	SpeedUp
	VGA	407.36	3.36	121.24	192.26	3.10	62.02	146.87	1.96	74.93
	XGA	1013.65	8.20	123.62	488.66	7.69	63.54	351.58	4.96	70.88
	HD1	1204.78	9.22	130.67	563.48	8.94	63.03	399.11	6.16	64.79
	HD2	1318.84	10.94	120.55	633.75	9.79	64.73	448.24	6.65	67.40
Bilateral	HD+	1792.37	14.61	122.68	864.35	13.74	62.91	621.09	9.49	65.45
	FHD	2580.64	20.81	124.01	1242.93	19.43	63.97	889.58	13.60	65.41
	QHD	4748.36	36.14	131.39	2287.54	33.67	67.94	1654.15	23.68	69.85
	UHD	10 388.87	79.79	130.20	5023.37	82.05	61.22	3595.64	53.16	67.64
	VGA	392.08	88.97	4.41	170.81	70.51	2.42	154.98	43.75	3.54
	XGA	559.73	231.53	2.42	214.89	179.54	1.20	180.36	111.21	1.62
	HD1	601.30	270.88	2.22	238.55	209.40	1.14	198.28	130.71	1.52
	HD2	648.80	307.32	2.11	225.85	236.03	0.96	193.20	147.46	1.31
Median	HD+	866.83	423.69	2.05	304.20	327.12	0.93	264.97	203.02	1.31
	FHD	1223.02	613.13	1.99	434.74	475.13	0.91	374.80	291.39	1.29
	QHD	2514.40	1087.86	2.31	1006.75	848.10	1.19	787.04	517.95	1.52
	UHD	5182.31	2421.49	2.14	2026.95	1895.44	1.07	1616.36	1160.93	1.39
	VGA	13.00	0.40	32.5	9.34	0.36	25.94	7.33	0.23	31.87
	XGA	14.03	0.96	14.61	6.41	0.82	7.82	4.14	0.53	7.81
	HD1	14.92	1.14	13.09	9.25	0.98	9.44	4.90	0.68	7.21
	HD2	15.93	1.31	12.16	9.81	1.56	6.29	5.50	0.77	7.14
Posterize	HD+	21.72	1.74	12.48	16.97	1.91	8.88	9.21	0.98	9.4
	FHD	32.59	2.43	13.41	18.11	3.49	5.19	10.31	1.45	7.11
	QHD	53.51	4.36	12.27	28.06	5.18	5.42	18.94	2.53	7.49
	UHD	117.55	12.54	9.37	61.97	9.99	6.20	35.42	5.68	6.24
	VGA	49.99	0.67	74.61	7.37	0.44	16.75	6.82	0.21	32.48
	XGA	23.79	0.89	26.73	12.72	0.83	15.33	6.05	0.49	12.35
	HD1	23.66	1.18	20.05	15.63	1.43	10.93	7.75	0.58	13.36
	HD2	26.75	1.20	22.29	17.27	1.92	8.99	7.98	0.65	12.28
Levels	HD+	35.04	1.73	20.25	24.77	2.48	9.99	12.28	1.07	11.48
	FHD	52.43	2.56	20.48	27.64	3.51	1.87	17.13	1.43	11.98
	QHD	86.30	4.40	19.61	56.21	4.84	11.61	29.12	2.89	10.08
	UHD	223.13	9.65	23.12	113.49	9.91	11.45	67.82	4.83	14.04
	VGA	76.47	4.19	18.25	40.50	3.85	10.52	38.55	2.40	16.06
	XGA	235.93	11.12	21.22	90.57	9.71	9.33	87.35	6.01	14.53
	HD1	209.92	13.01	16.14	100.75	11.34	8.88	99.32	7.04	14.11
	HD2	227.91	14.86	15.34	113.01	13.00	9.69	95.17	8.00	11.90
Fisheye	HD+	330.27	20.56	16.06	152.66	17.16	8.90	154.96	10.97	14.13
	FHD	465.10	28.95	16.07	219.47	24.46	8.97	188.64	15.83	11.92
	QHD	776.28	51.44	15.09	380.27	45.62	8.34	312.67	28.74	10.88
	UHD	1735.60	113.28	15.32	855.84	97.16	8.81	690.15	65.57	10.53
	VGA	16.37	0.52	31.48	11.10	0.46	24.13	8.54	0.34	25.12
	XGA	20.04	1.34	14.96	9.27	1.67	5.55	4.24	0.74	5.73
	HD1	24.26	1.40	17.33	14.06	1.88	7.48	9.03	0.88	10.26
	HD2	24.94	1.55	16.09	15.85	1.38	11.49	10.14	1.03	9.84
Contrast	HD+	35.27	2.17	16.25	18.01	2.80	6.43	13.12	1.53	8.58
	FHD	51.82	3.22	16.09	21.41	5.41	3.96	16.98	2.26	7.51
	QHD	92.52	6.09	15.19	37.69	6.24	6.04	25.19	4.00	6.30
	UHD	205.35	14.82	13.86	80.95	11.66	6.94	47.73	8.35	5.72
	VGA	89.88	2.19	41.04	36.72	1.95	18.83	26.58	1.19	22.34
	XGA	202.51	5.47	37.02	112.46	4.09	27.50	91.58	3.24	28.27
	HD1	231.60	6.58	35.20	108.34	5.71	18.97	76.72	3.87	19.82
	HD2	253.70	7.46	34.01	93.27	5.98	15.60	78.72	3.99	19.73
Convolution 5x5	HD+	339.92	10.89	31.21	129.88	8.44	15.39	100.93	5.77	17.49
	FHD	453.16	15.36	29.50	182.59	12.13	15.05	138.67	8.33	16.65
	QHD	890.39	27.11	32.84	456.13	20.90	21.82	334.08	15.25	21.91
	UHD	1903.40	60.22	31.61	913.72	49.04	18.63	647.45	33.71	19.21
	VGA	13.32	0.44	30.17	5.63	0.38	14.82	11.00	0.23	47.83
	XGA	9.33	0.92	10.14	5.23	0.83	6.30	3.19	0.54	5.91
	HD1	12.10	1.15	10.52	7.57	1.06	7.14	3.78	0.64	5.91
	HD2	13.44	1.27	10.58	9.11	1.92	4.74	5.25	0.75	7.00
Grayscale	HD+	15.66	1.76	8.90	10.88	2.12	5.13	6.42	1.09	5.89
	FHD	22.34	2.47	9.04	15.35	3.20	4.80	9.07	1.69	5.37
	QHD	46.79	4.38	10.68	24.33	5.35	4.55	13.53	3.03	4.47
	UHD	92.05	10.48	8.78	51.63	9.45	5.46	28.95	6.39	4.53
	VGA	44.63	1.55	28.79	16.30	0.95	17.16	12.72	0.63	20.19
	XGA	84.15	3.99	21.09	38.66	3.40	11.37	30.01	1.72	17.45
	HD1	92.65	4.48	20.68	40.13	3.80	10.56	26.72	2.18	12.26
	HD2	129.85	5.09	25.51	40.79	4.22	9.67	27.62	2.38	11.61
Convolution 3x3	HD+	164.50	7.04	23.37	55.35	5.52	10.03	41.43	3.37	12.29
	FHD	194.53	10.82	17.98	74.79	7.20	10.39	53.64	4.89	10.97
	QHD	354.32	19.13	18.52	170.46	11.54	14.77	116.32	8.71	13.35
	UHD	719.95	41.73	17.25	284.46	24.46	11.63	199.14	19.21	10.37
	VGA	239.09	3.54	67.54	119.20	2.56	46.56	73.42	1.83	40.12
	XGA	415.69	9.26	44.89	142.87	6.35	22.50	105.44	4.45	23.69
	HD1	491.50	11.02	44.60	167.29	7.20	23.23	127.52	5.23	24.38
	HD2	572.96	12.35	46.39	189.30	8.27	22.89	139.29	5.94	23.45
GaussianBlur	HD+	757.25	17.36	43.62	254.75	11.36	22.43	190.78	8.10	23.55
	FHD	1106.58	24.60	44.98	365.29	15.86	23.03	278.61	11.68	23.85
	QHD	1927.03	43.09	44.72	648.62	27.24	23.81	488.22	20.45	23.87
	UHD	4348.09	103.44	42.03	1445.02	63.08	22.91	1086.51	46.71	23.26

Tabla II: Mean of execution times in milliseconds.

Aprendizaje profundo para la detección de BAAR en microscopía de esputo

Lara Visuña, Javier Garcia-Blas y Jesús Carretero[1]

Resumen— **Hoy en día, la tuberculosis es una de las enfermedades más mortíferas. Sin embargo, se ha demostrado que un diagnóstico rápido tiene una gran influencia en el pronóstico y evolución de la enfermedad. El objetivo de este trabajo es la aceleración del tiempo de diagnóstico, así como la mejora de la sensibilidad de la microscopía de esputo como herramienta de diagnóstico de la tuberculosis. Este trabajo presenta una novedosa técnica de aprendizaje profundo para la detección automática de bacilos acidorresistentes (BAAR) en la microscopía de esputo con tinción de Ziehl Neelsen (ZN). En primer lugar, las imágenes de microscopía se preprocesan y fragmentan. A continuación, una única red convolucional indica que fragmentos de la imagen contienen bacilos. Los resultados demuestran la eficacia del sistema, obteniendo un *recall* de 92,86 % y una precisión de 99,49 %, además de una reducción del tiempo de análisis respecto a las técnicas convencionales. Finalmente, comparando el sistema con trabajos previos, demostramos obtener una mejora considerable en los resultados. El procedimiento presentado en este artículo es generalizable a otras técnicas de microscopía y a otros campos de estudio en los que el objeto a localizar tenga un tamaño y contraste reducido respecto a la imagen completa.**

Palabras clave— **CNN, aprendizaje profundo, visión artificial, BAAR, Tinción ZN.**

I. Introducción

HOY en día, la tuberculosis (TB) es una de las enfermedades más mortíferas. Es una enfermedad contagiosa causada por la bacteria Mycobacterium Tuberculosis, la cual, se propaga a través del aire. Se calcula que en 2021 se produjeron más de 1,6 millones de muertes por tuberculosis en todo el mundo, siendo la mayor causa de muerte por un único agente infeccioso, hasta la llegada del brote pandémico de coronavirus (COVID-19) [1]. A pesar de este gran número de muertes, alrededor del 85 % de las personas pueden curarse con un diagnóstico y tratamiento adecuados [1].

Pese al progreso de la tecnología y de los avances médicos, la Organización Mundial de la Salud (OMS) informó de reducciones en el número total de personas diagnosticadas de tuberculosis en 2020 y 2021, lo que, según ellos, podría reflejar un aumento del número de casos no diagnosticados. Es importante resaltar que un diagnóstico rápido y preciso permite romper la transmisión de la enfermedad, así como el inicio temprano del tratamiento. Existen múltiples técnicas de diagnóstico, como son el cultivo bacteriano o las pruebas moleculares. Uno de los métodos de diagnóstico más utilizados es la microscopía de esputo, una técnica muy conocida que permite la detección de la enfermedad, así como el seguimiento del tratamiento y la evolución de la infección. La microscopía de esputo es una técnica de diagnóstico barata y rápida, lo que la convierte en la primera herramienta de diagnóstico en los países en vías de desarrollo. La microscopía de esputo es un proceso manual y laborioso que requiere la observación a través de un microscopio para localizar los bacilos. Esta técnica es propensa a errores, su sensibilidad es de alrededor del 55-95 % dependiendo de la experiencia de los técnicos de laboratorio, pero también dependiendo de la carga infecciosa del paciente [2].

Una de las técnicas de microscopía de esputo más extendida es la tinción de Ziehl Neelsen (ZN), en la que los bacilos acidorresistentes (BAAR), como la Mycobacterium Tuberculosis, cambian de color, pasando a rojo o rosa fuerte. Los bacilos se pueden ver en los portaobjetos de microscopía en diferentes formas: bacilos individuales, en forma de V, cúmulos o fragmentos de bacilo. Cabe destacar que, la cantidad de bacilos reportada está directamente relacionada con el nivel de infección y cuán infeccioso es un paciente [3]. Además, incluso los fragmentos presentes en la muestra deben ser reportados, aunque debido a su reducido tamaño pueden pasar desapercibidos.

El diagnóstico de la tuberculosis requiere analizar múltiples imágenes de microscopía en busca de signos de la enfermedad. Si no se encuentran BAAR en 100 campos microscópicos, la prueba se notifica como negativa. Si se informan nueve o menos bacilos en 100 campos de microscopía, el resultado es escaso. Con una mayor cantidad de bacilos informados, la graduación debe ser de 1+ a 3+, siguiendo las indicaciones descritas en la tabla I. Es importante resaltar que la sensibilidad de la microscopía ZN disminuye si la concentración de bacilos es inferior a 1000/ML en el esputo, entonces, sólo el 10 % de los campos analizados en el portaobjetos mostrarán la presencia de BAAR [3].

Tabla I: Diagnóstico y clasificación de la tuberculosis según el recuento de BAAR en imágenes microscópicas [3], [4].

BAAR	Campos de microscopía	Clasificación
0	Reportados en 100 campos de microscopía	Negativo
1-9	Reportados en 100 campos de microscopía	Escaso
10-99	Reportados en 100 campos de microscopía	1+
1-10	Reportados por cada campo de microscopía	2+
>10	Reportados por cada campo de microscopía	3+

Dado la importancia de detectar todos los bacilos y con el objetivo de evitar la dependencia del factor humano y aumentar la sensibilidad, este trabajo diseña, implementa y evalúa un detector automático de bacilos en microscopía de esputo utilizando técnicas de aprendizaje profundo. La literatura existen-

[1]Departamento de Informática, Universidad Carlos III de Madrid (UC3M), e-mail: lvisuna@pa.uc3m.es, {fjblas,jcarrete}@inf.uc3m.es.

te ha reportado resultados notables para la detección automática de bacilos en microscopía de esputo con tinción ZN. Debido al color rosado que adquieren los BAAR diferentes trabajos han explotado su detección mediante procesado de imagen [5], [6]. En la literatura actual el procesamiento de imágenes se complementa en diferentes estudios con técnicas de *Deep learning* [7].

Para detectar bacilos individuales y lindantes, Panicker et al. [8] desarrollan un detector automático de bacilos. Esta solución se evaluó con 22 imágenes microscópicas de frotis, logrando un 97,13 % de *recall* y un 78,4 % de precisión. El método propuesto utilizó un enfoque de segmentación simple para clasificar el primer plano y el fondo de la imagen, después de eso, el primer plano es clasificado por una CNN entre bacilos y no bacilos. También en 2018, un algoritmo completo de aprendizaje profundo presentado por Kant et al. [9]. Se propuso extraer fragmentos de (20x20) píxeles de la imagen original, luego dos CNN en cascada clasifican los fragmentos entre bacilos y no bacilos. Otra técnica completa de aprendizaje profundo fue propuesta por El-Melegy et al. [10], donde se utilizó una arquitectura *Faster* R-CNN para la detección de bacilos. *Faster* R-CNN combina una red de propuesta de región (RPN), y una CNN (VGG16, en este caso), logrando un *F-score* del 89,7 %.

En 2021, V. Shwetha et al. [11] propusieron otro algoritmo de dos etapas. En primer lugar, tras una etapa de preprocesamiento, se utiliza el método *k-means* para la segmentación. A continuación, una CNN preentrenada clasifica la segmentación en bacilos u otra célula. SqueezeNet alcanza un *accuracy* del 97 %, diferenciando las células de los bacilos. Además también se han realizado esfuerzos para detectar otros parámetros relacionados con la enfermedad, como el nivel de infección [12] o la presencia o ausencia de tuberculosis [13].

La literatura existente presenta algoritmos y métodos disponibles para la detección de BAAR. Sin embargo, esta línea de investigación sigue siendo un reto. La práctica habitual para la detección de bacilos tuberculosos es la siguiente: tras el preprocesamiento, un primer algoritmo propone regiones candidatas o descarta el fondo, seguido de un algoritmo de clasificación. Con este esquema, algunos bacilos pueden ser contados erróneamente o etiquetados como fondo. Podemos observar la necesidad de un algoritmo de una etapa en la literatura. En este trabajo, proponemos un algoritmo de una etapa basado en *deep learning* para la detección de BAAR, evitando la fase de propuesta de candidatos. Este esquema evita falsos negativos y acelera la detección de BAAR así como el diagnóstico de tuberculosis. El detector se ha diseñado enfatizando la localización de bacilos individuales y fragmentos de estos, debido a su mayor dificultad a la hora de ser vistos y reportados en las imágenes microscópicas. Las principales aportaciones del trabajo son las siguientes:

- Se ha diseñado y desarrollado una técnica para detectar bacilos con un algoritmo de aprendizaje profundo de una etapa. En el sistema, las imágenes de microscopía son preprocesadas y fragmentadas. A continuación, se introducen en la Red Neuronal Convolucional (CNN), que decide que áreas de la imagen son reportadas como bacilos.
- Se han comparado múltiples métricas de evaluación con trabajos previos, mostrando una mejora significativa en la detección de bacilos acidorresistentes en imágenes de microscopía. El sistema mostró un *recall* del 92,86 % y una precisión del 99,49 %.
- Este enfoque acelera el proceso de recuento de BAAR, especialmente cuando se hace uso de unidades de procesamiento gráfico (GPUs), reduciendo el tiempo de diagnóstico. Además, mejora la sensibilidad frente a la búsqueda manual de bacilos.
- Se ha demostrado que el algoritmo alcanza altos niveles de abstracción, además su arquitectura de una etapa favorece que sea altamente generalizable. Puede ser entrenado para localizar otras bacterias y gérmenes en imágenes de microscopía. Además se puede aplicar a otras áreas, en las que sea necesario localizar elementos de tamaño reducido respecto al tamaño de la imagen completa.

A continuación se presentan los materiales y procedimientos implementados durante el desarrollo del sistema. Las secciones III y IV muestran los resultados experimentales y las discusiones sobre estos, respectivamente. Finalmente, en la sección V se presentan las principales conclusiones del proyecto.

II. Materiales y Métodos

El sistema propuesto para la detección automática de bacilos de tuberculosis utiliza imágenes microscópicas de tinción ZN, que pueden introducirse en el sistema de manera individual o en lotes. Las imágenes de microscopía se mejoran y fragmentan siguiendo el esquema presentado en las secciones II-A y II-B. Los fragmentos resultantes que, tienen unas dimensiones de 80 píxeles de alto y 80 píxeles de ancho, son introducidos en la arquitectura CNN, entrenada siguiendo los procedimientos descritos en la sección II-C.

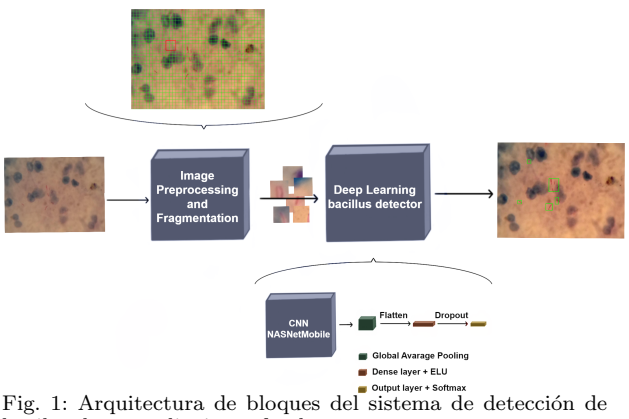


Fig. 1: Arquitectura de bloques del sistema de detección de bacilos de aprendizaje profundo.

El detector está formado por una CNN seguida de una red neuronal densa. Para la parte convolucional del sistema se implementa la arquitectura NasNetmobile, propuesta por Google Brain [14]. La familia de redes NASNet se diseñan automáticamente mediante una *Neural Architecture Search* (NAS) junto con un controlador *Recurrent Neural Network* (RNN). NASNetMobile se diseñó con el objetivo de lograr una alta precisión y una baja latencia de inferencia. Tras esta red, se implementa la red neuronal profunda compuesta de una capa oculta con 128 neuronas que utilizan ELU (mostrada en la ecuación 1) como función de activación. A continuación, la capa de salida incluye 2 neuronas diseñada con Softmax (mostrado en la ecuación 2) como función de activación. Las dos neuronas de salida diferencian entre los fragmentos de la imagen que contiene bacilos o únicamente de fondo. Entre las dos capas densas se incluye un *dropout* con el objetivo de controlar el sobreajuste. La arquitectura completa del sistema de detección se presenta en la figura 1.

$$f(x) = \begin{cases} x, x > 0 \\ \alpha(e^x - 1), x \leq 0 \end{cases} \quad (1)$$

$$S(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad (2)$$

Con el fin de hacer un cuadro delimitador ajustado y evitar la notificación de información imprecisa, cuando se detecta la presencia de bacilos en un fragmento de imagen, sólo el área central de este se señala como bacilo. Una vez que el sistema ha analizado todos los fragmentos, la imagen de microscopía se muestra con todos los bacilos identificados.

A. Adquisición y procesamiento de los datos

Para el desarrollo de este proyecto se emplearon 200 imágenes de microscopía ZN de esputo. Las imágenes de microscopía fueron recogidas por "AI Research and Automated Laboratory Diagnostics" y son de acceso abierto atreves de la plataforma Kaggle[1]. Todas las imágenes están en formato *jpg* con un tamaño de 1.124 x 1.636 píxeles. El conjunto de datos incluye cuadros delimitadores que enmarcan los bacilos, reportados en archivos *XML*.

Tabla II: División de la base de datos en imágenes para entrenamiento, validación y *test*.

	Entrenamiento	**Validación**	***Test***	**Total**
Imágenes	120	40	40	**200**
Bacilos reportados	436	235	188	**859**

La división de las imágenes de microscopía para entrenamiento, validación y *test* se llevó acabo de manera aleatoria, resultando como muestra la tabla II. Además, la tabla resume la cantidad de bacilos reportados en los archivos *XML*. El número de bacilos reportados por imagen microscópica se encuentra entre 0 y 18, siendo la media de 4 bacilos por imagen. Durante la fase de preparación de los datos, se corrigieron los cuadros delimitadores de las imágenes de entrenamiento para facilitar y mejorar el entrenamiento de la CNN.

Las imágenes de microscopía del conjunto de datos presentan diferentes intensidades de luz, estructuras biológicas y fondos. La figura 2 contiene una muestra de las imágenes de esputo. Las imágenes incluyen principalmente fragmentos de BAAR, bacilos individuales o pequeños grupos de bacilos, debido a que son más difíciles de localizar mediante el procedimiento manual y es el objetivo central de la investigación.

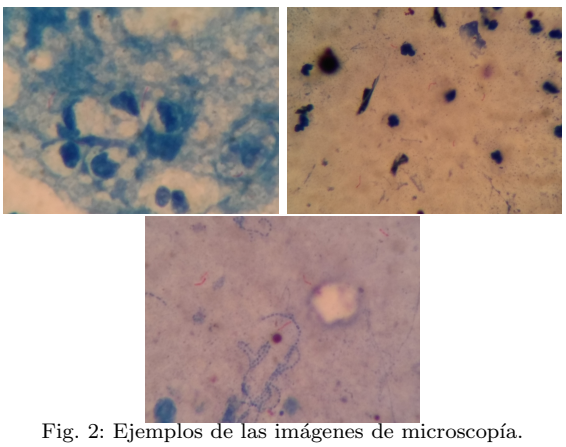


Fig. 2: Ejemplos de las imágenes de microscopía.

Se aplica un esquema de normalización a las imágenes de esputo para homogeneizarlas aplicando la ecuación 3.

$$Img_{norm}(x_i) = \frac{Img_{ori}(x_i) - Min_{pixel}}{Max_{pixel} - Min_{pixel}} \quad (3)$$

Como conjunto control se incluyen seis imágenes de microscopía pertenecientes a pacientes con una mayor carga bacteriana. Las imágenes de control presentan no solo mayor cantidad de bacterias sino también diferentes formaciones de cúmulos. Estas imágenes nos permiten evaluar como actuaría el sistema frente a situaciones para las que no ha sido entrenado.

B. Procedimientos de fragmentación

Las imágenes preprocesadas pasan a fragmentarse con el objetivo de localizar los bacilos. Como las imágenes de microscopía suelen tener una alta resolución su fragmentación facilita su manejo evitando limitaciones de memoria RAM o GPU [15]. Para seleccionar las dimensiones de las imágenes fragmentadas se analizaron los cuadros delimitadores de las imágenes de entrenamiento (ver figura 3). Buscando un tamaño que permita encuadrar la mayoría de bacilos de manera precisa. Se selecciona un tamaño de 80 píxeles de alto y 80 píxeles de ancho.

Los fragmentos se forman con una ventana deslizante, con pasos de 40 píxeles, hasta cubrir todo el área de la imagen de microscopía. Este método

[1]https://www.kaggle.com/datasets/saife245/tuberculosis-image-datasets

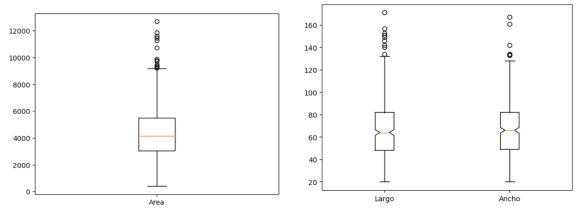


Fig. 3: Distribución del área, longitud y ancho de los cuadros delimitadores informados para las imágenes de entrenamiento.

permite que cada fragmento comparta píxeles con el fragmento anterior y posterior, así como, con el fragmento superior y el inferior, consiguiendo así que todos los bacilos sean captados por la ventana deslizante para su posterior clasificación.

Para entrenar la red neuronal es necesario indicar que fragmentos incluyen BAAR y cuales no. Para esto se utiliza la ecuación 4, solo si el resultado es mayor de 0,5 el fragmento se le presentará a la red como con bacilo. Dado que los bacilos tienen un tamaño reducido en referencia a la imagen y ocupan un área pequeña incluso en los fragmentos, en la ecuación 4 se considera como $Area_C$ el área central del fragmento, de dimensiones de (40x40) píxeles. Este $Area_C$ se evalúa respecto los cuadros delimitadores informados ($Area_R$).

$$IOA = \frac{Area_R \cap Area_C}{Area_C} \tag{4}$$

En la figura 4 se muestran los fragmentos resultantes de esta clasificación, mostrando la primera fila fragmentos informados como con bacilos y la segunda como no bacilos. Podemos observar que un porcentaje de bacilo elevado tiene que estar incluido en el fragmento para clasificarse como bacilo.

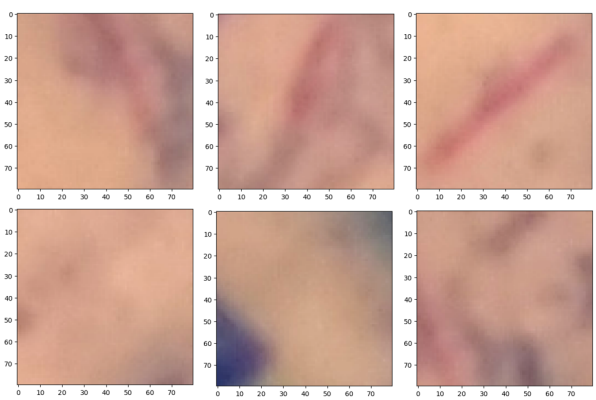


Fig. 4: Fragmentos resultantes.

C. Procedimientos de entrenamiento

El entrenamiento es la etapa más importante del aprendizaje de una red neuronal. Para entrenar la CNN, sólo se utilizó el conjunto de entrenamiento. Las 120 imágenes de microscopía se fragmentaron, de modo que tras el preprocesamiento y la fragmentación se obtuvieron 130180 fragmentos. Según el procedimiento descrito anteriormente, los fragmentos se etiquetaron como bacilos o fondo. Los datos de entrenamiento finales están muy desequilibrados, en los 130180 fragmentos, sólo 995 fueron etiquetados como bacilos. En base a esto, se seleccionó la función de pérdida de Poisson (mostrada en la ecuación 5).

$$L_{poisson} = \sum_{i=1}^{n} y_{pred_i} - Y_i * log(y_{pred_i}) \tag{5}$$

Para evitar el sobreajuste y aprovechar los entrenamientos previos, se aplica un esquema de aprendizaje por transferencia seguido de un entrenamiento suave. Explotamos NASNetmobile con el entrenamiento previo obtenido en ImageNet [16]. El sistema completo de aprendizaje profundo se entrenó en 50 *epoch*, se seleccionó un tamaño de *batch* de 32 imágenes y una tasa de aprendizaje de 1,00E-04 en el optimizador Adam. Los hiperparámetros se seleccionaron en función de los resultados de validación, con el fin de optimizar varias métricas como el *accuracy*, el *recall*, la precisión y el valor F1.

III. Resultados experimentales

En esta sección presentamos los resultados de entrenamiento y rendimiento de nuestro modelo. Los experimentos se realizaron con Jupyter Notebooks, y se implementaron utilizando Python 3.6.9, TensorFlow 2.5.0 y Keras 2.5.0. El entrenamiento se ejecutó con una NVidia RTX 3090. El sistema se entrenó durante 4,5 horas hasta alcanzar 50 *epoch*. Las curvas de pérdida y *accuracy* se presentan en la figura 5. Muestran una *accuracy* final del 99,98 % para las imágenes de entrenamiento y del 99,06 % para las imágenes de validación.

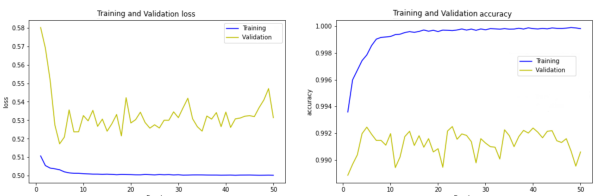


Fig. 5: Curvas de pérdidas y *accuracy* en las imágenes de entrenamiento y validación.

Utilizamos 40 imágenes de prueba para evaluar nuestro sistema, el resultado de las 40 imágenes se obtuvo en 50,76 segundos. El sistema de detección alcanzó un *accuracy* en la clasificación de fragmentos del 99,34 %. Sin embargo, los fragmentos no están equilibrados, ya que hay unas 100 veces más fragmentos de fondo que fragmentos que contienen bacilos. Por lo tanto, el resultado a nivel de bacilos se presenta en la tabla III, donde el *recall*, la precisión y el *F1-score* se calculan analizando los bacilos detectados en las imágenes de microscopía frente a los bacilos presentes en las imágenes de microscopía. El sistema obtuvo un 92,86 % de *recall* y un 99,49 % de precisión en la detección de bacilos en imágenes de microscopía con tinción ZN de esputo.

En la figura 6 se muestran resultados del sistema de detección de bacilos. Cada columna muestra una imagen de microscopía diferente, incluyendo imagen

Tabla III: Eficacia del detector a nivel de bacilos.

	Recall	**Precisión**	**F1**
Resultados de *Test*	92.86 %	99.49 %	96.06 %

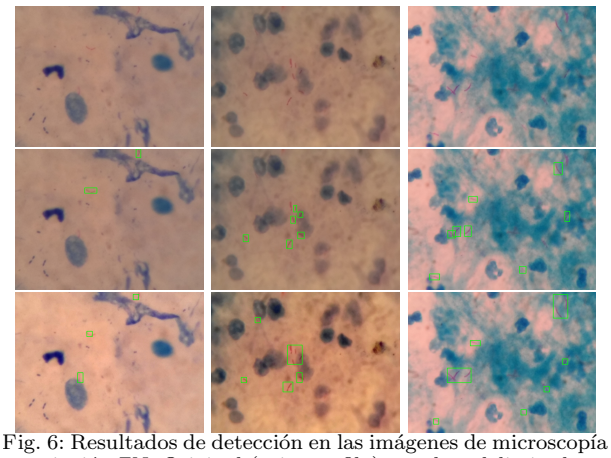


Fig. 6: Resultados de detección en las imágenes de microscopía con tinción ZN. Original (primera fila), cuadros delimitadores originales(segunda fila) y salida del sistema (última fila).

original, el cuadro delimitador original y el resultado del sistema.

Para evaluar la generalización del sistema se realizaron experimentos con imágenes de microscopía que pertenecen a pacientes con una mayor cantidad de infección (conjunto de control). Estas imágenes presentan un mayor número de bacilos así como la formación de grandes cúmulos. Alguno de los resultados obtenidos se muestran en la figura 7.

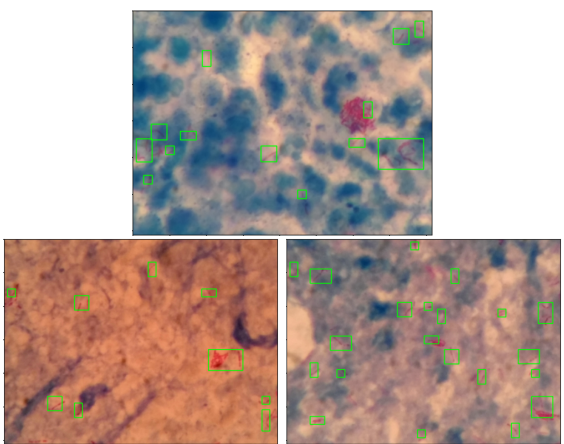


Fig. 7: Muestra de resultados del sistema sobre el conjunto de control.

IV. Discusión

Este artículo diseña e implementa un detector automático de bacilos basado en aprendizaje profundo. El detector trabaja con imágenes de microscopía con tinción ZN. El sistema está entrenado para la detección de BAAR en imágenes de microscopía con baja presencia de bacilos, debido a su mayor complejidad de detección en una búsqueda manual.

El sistema fue entrenado con 120 imágenes de microscopía, con los bacilos previamente señalados por cajas delimitadoras. Estas cajas delimitadoras fueron indicadas manualmente y sólo señalan bacilos individuales. A pesar de que el sistema propuesto fue entrenado con esta información, el análisis de los resultados de *test* demostró que el sistema de aprendizaje profundo logró abstraer este conocimiento para, no solo señalar bacilos individuales, también bacilos solapados. Este comportamiento se puede ver en la segunda y tercera columnas de la figura 6, donde grupos de dos bacilos son detectados por el sistema presentado.

El sistema también puede detectar fragmentos de bacilos o bacilos parcialmente visibles, como se ilustra en la primera columna de la figura 6. El recuadro original señala dos localizaciones de bacilos frente a las tres señaladas por el sistema. El cuadro delimitador original no contabiliza el tercer bacilo, ya que se encuentra bajo el pigmento azul.

Tras analizar los resultados de *test*, observamos que el sistema omite un número bajo de BAAR. Esto ocurre mayormente en imágenes borrosas del conjunto de datos y en bacilos poco visibles, que en muchos casos pasaron desapercibidas por el cuadro delimitador original. Esto sugiere que aumentando las imágenes de entrenamiento y mejorando los cuadros delimitadores originales, el sistema mejoraría y se evitaría este problema.

En la tabla **??** se comparó el sistema propuesto con modelos anteriores de detectores de bacilos basados en aprendizaje profundo. La tabla muestra diferentes tamaños de conjuntos de datos, técnicas de preprocesamiento y algoritmos de aprendizaje profundo.

Todos los estudios anteriores muestran una visión de los algoritmos en dos etapas, en la que la primera etapa selecciona los bacilos candidatos y, a continuación, un clasificador decide cuáles de ellos son realmente bacilos. En el enfoque presentado, la arquitectura del detector es un algoritmo de una etapa. Incluso prescindiendo de la primera etapa, el sistema muestra resultados fiables, logrando alcanzar una precisión superior a los trabajos previos. Con respecto al tiempo de inferencia, no hay reportes en la literatura consultada.

Los resultados presentados reportan una aceleración frente al proceso de recuento manual de BAAR, reduciendo así el tiempo de diagnóstico de tuberculosis. De acuerdo con los resultados, el detector también es capaz de aumentar la sensibilidad de la microscopía de tinción ZN como herramienta de diagnóstico evitando bacilos inadvertidos debido a la falta de experiencia de los técnicos de laboratorio, la gran carga de trabajo, o bacilos parcialmente ocultos.

El trabajo en entornos reales implicará evaluar imágenes pertenecientes a pacientes con mayor carga bacteriana, por lo que es importante evaluar la respuesta del sistema. La figura 7 muestra esta respuesta, encontramos un comportamiento similar al reportado en el conjunto de *test*, además de una gran capacidad de abstracción, consiguiendo detectar la presencia de formaciones de cúmulos de BAAR. El sistema no es capaz de detectar todas las formaciones, los grandes cúmulos o superposiciones en X pueden pasar desapercibidas por el sistema.

Tabla IV: Técnicas de *Deep learning* y rendimiento para la detección de bacilos reportados en trabajos anteriores.

	Recall	**Precisión**	**F1**	***Test-set***	**Técnica de detección**
[11]	97,00 %	98,00 %	97,50 %	298	Segmentación + CNN
[8]	97,13 %	78,40 %	86,77 %	22	Segmentación + CNN
[9]	83,78 %	67,55 %	74,79 %	40	Dos CNN en cascada
[10]	98,30 %	82,60 %	89,77 %	300	*Faster* R-CNN (*Data Augmentation*)
Este trabajo	**92,86 %**	**99,49 %**	**96,06 %**	**40**	**Una única CNN**

Es importante resaltar que además de no ser entrenada con este tipo de casuísticas, la técnica desarrollada está enfocada en la localización de estructuras o formaciones pequeñas en relación al tamaño global de la imagen.

La técnica de visión artificial presentada puede ser utilizada en otros campos y áreas. Su uso estaría recomendado sobretodo cuando la poca saturación, contraste o dimensión del objeto a localizar imposibilite su detección mediante los métodos de dos etapas.

V. Conclusiones

En este trabajo se propone un detector de bacilos de una etapa basado en aprendizaje profundo que localiza automáticamente los BAAR presentes en imágenes de microscopía de esputo con tinción ZN.

Las imágenes de esputo se preprocesan y fragmentan, la red de aprendizaje profundo utiliza estos fragmentos como entrada. Durante el entrenamiento, más del 95 % de los fragmentos se etiquetan como fondo. A pesar del desequilibrio del conjunto de datos, el método obtiene una precisión del 99,49 % y un *recall* del 92,86 % en la detección de BAAR en el conjunto de *test*. El sistema propuesto supera la detección manual original, detectando incluso bacilos superpuestos.

Nuestros resultados se compararon con investigaciones bibliográficas anteriores, demostrando la viabilidad del método propuesto. Los resultados reportados mejoran las soluciones anteriores evitando la etapa de propuesta de candidatos, y alcanzando la mayor precisión reportada en la literatura. El método presenta una detección precisa y rápida de BAAR en conjuntos de imágenes de microscopía, ayudando así al diagnóstico y mejorando el pronóstico de la tuberculosis.

Nuestros esfuerzos actuales se centran en la fase de entrenamiento de nuestro detector de bacilos basado en aprendizaje profundo. La inteligencia artificial en ámbitos biomédicos presenta la dificultad añadida para encontrar datos veraces con los que trabajar. Nuestro objetivo es demostrar que el método de una etapa puede entrenarse con diferentes técnicas de microscopía, manteniendo la precisión actual.

Agradecimientos

El presente trabajo ha sido financiado por "Innovative Medicines Innitiative 2 Joint Undertaking (JU)" bajo el identificador del acuerdo de subvención número 853989. JU recibe apoyo de "European Union's Horizon 2020 research and innovation programme and EFPIA" y la organización sin ánimo de lucro "Global Alliance for TB Drug Development, Bill & Melinda Gates Foundation and University of Dundee".

Esta es una versión ampliada y revisada de un trabajo previo publicado por *Springer Nature, Lecture Notes in Networks and Systems book series*, volumen 721, páginas 269-279[17]. Se han incluido nuevos resultados, y ampliado la descripción metodológica.

Referencias

[1] G WHO, "Global tuberculosis report 2022," *Glob Tuberc. Rep*, 2022.

[2] Prasanta Kumar Das, Somtirtha B Ganguly, Bodhisatya Mandal, et al., "Sputum smear microscopy in tuberculosis: It is still relevant in the era of molecular diagnosis when seen from the public health perspective," *Biomedical and Biotechnology Research Journal (BBRJ)*, vol. 3, no. 2, pp. 77–79, 2019.

[3] Bipul Chandra Deka, Devabrata Saikia, and Manash pratim Kashyap, "Diagnosis of tuberculosis," *European Journal of Molecular & Clinical Medicine*, vol. 9, no. 07, pp. 2022.

[4] Global Laboratory Initiative (GLI), *Laboratory Diagnosis of Tuberculosis by Sputum Microscopy*, SA Pathology, 2013.

[5] Rafikha Aliana A Raof, MY Mashor, R Badlishah Ahmad, and SSM Noor, "Image segmentation of ziehl-neelsen sputum slide images for tubercle bacilli detection," *Image Segmentation*, pp. 365–378, 2011.

[6] Zulfiqar Ahmad Khan, Waseem Ullah, Amin Ullah, Seungmin Rho, Mi Young Lee, and Sung Wook Baik, "An adaptive filtering technique for segmentation of tuberculosis in microscopic images," in *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*, 2020, pp. 184–187.

[7] Zhisong Li, Jiayao Ling, Jing Wu, Nian Luo, Min Tan, and Ping Zhong, "Research on preprocessing method for microscopic image of sputum smear and intelligent counting for tubercule bacillus," in *IOP Conference Series: Materials Science and Engineering*. IOP Publishing, 2018, vol. 466, p. 012112.

[8] Rani Oomman Panicker, Kaushik S Kalmady, Jeny Rajan, and MK Sabu, "Automatic detection of tuberculosis bacilli from microscopic sputum smear images using deep learning methods," *Biocybernetics and Biomedical Engineering*, vol. 38, no. 3, pp. 691–699, 2018.

[9] Sonaal Kant and Muktabh Mayank Srivastava, "Towards automated tuberculosis detection using deep learning," in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2018, pp. 1250–1253.

[10] Moumen El-Melegy, Doaa Mohamed, and Tarek ElMelegy, "Automatic detection of tuberculosis bacilli from microscopic sputum smear images using faster r-cnn, transfer learning and augmentation," in *Pattern Recognition and Image Analysis: 9th Iberian Conference, IbPRIA 2019, Madrid, Spain, July 1–4, 2019, Proceedings, Part I 9*. Springer, 2019, pp. 270–278.

[11] V Shwetha, Keerthana Prasad, Chiranjoy Mukhopadhyay, Barnini Banerjee, and Abir Chakrabarti, "Automatic detection of bacilli bacteria from ziehl-neelsen sputum smear images," in *2021 2nd International Conference on Communication, Computing and Industry 4.0 (C2I4)*. IEEE, 2021, pp. 1–5.

[12] KS Mithra and WR Sam Emmanuel, "Automated identification of mycobacterium bacillus from sputum images for tuberculosis diagnosis," *Signal, Image and Video Processing*, vol. 13, pp. 1585–1592, 2019.

[13] Rani Oomman Panicker and MK Sabu, "Automatic detection of tuberculosis bacilli from conventional sputum smear microscopic images using densely connected convolutional networks," *SN Computer Science*, vol. 3, no. 4, pp. 263, 2022.

[14] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[15] Fatemeh Hoorali, Hossein Khosravi, and Bagher Moradi, "Automatic bacillus anthracis bacteria detection and segmentation in microscopic images using unet++," *Journal of Microbiological Methods*, vol. 177, pp. 106056, 2020.

[16] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al., "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[17] Lara Visuña, Javier Garcia-Blas, and Jesus Carretero, "Novel deep learning-based technique for tuberculosis bacilli detection in sputum microscopy," in *International Conference on Interactive Collaborative Robotics*. Springer, 2023, pp. 269–279.

Instalación y configuración de nodos de cómputo heterogéneos para centros de datos

Antonio Joaquín Tárraga[1], Jesús Escudero[2] y Francisco José Quiles[3]

Resumen— **En este artículo, investigamos la configuración de nodos servidor heterogéneos en un clúster de Computación de Altas Prestaciones llamado CELLIA (Cluster for the Evaluation of Low-Latency Interconnection Architectures). Evaluamos diferentes configuraciones utilizando aplicaciones de entrenamiento de aprendizaje profundo basadas en el framework distribuido Horovod, aprovechando la potencia de cálculo proporcionada por las unidades de procesamiento gráfico. Nuestro objetivo fue determinar la configuración más adecuada para optimizar el rendimiento del clúster. Los resultados obtenidos proporcionan información valiosa para mejorar el rendimiento de clústeres similares en diversas disciplinas. Estos mostraron mejoras significativas en áreas como el acceso a memoria remota, el ancho de banda entre GPUs distribuidas (hasta 15 veces mayor con GPUDirect RDMA) y los accesos a memoria más rápidos con GPUDirect Storage.**

Palabras clave— **Infiniband, NVMeoF, GPUDirect Storage, GPUDirect RDMA, GDRCopy, CUDA, MPI, NCCL, Horovod, Clúster.**

I. Introducción

En los últimos años, ha surgido una creciente demanda de mayor potencia de cálculo y capacidad de procesamiento de datos en superordenadores y centros de datos. Estos sistemas son fundamentales para una amplia gama de aplicaciones, como simulaciones científicas, predicción meteorológica, análisis de datos científicos e ingeniería, inteligencia artificial, redes neuronales y aprendizaje automático. Con el fin de manejar eficientemente grandes volúmenes de datos a alta velocidad, los superordenadores y centros de datos requieren herramientas esenciales como la computación heterogénea.

La computación heterogénea está teniendo un impacto significativo en la arquitectura de los nodos de procesamiento debido a que las aplicaciones requieren el uso de diversos recursos de cálculo y almacenamiento, los cuales deben ser compartidos sin problemas, independientemente de su ubicación física. Además de la CPU, la RAM y los discos SSD, los nodos de servidor también incluyen dispositivos de propósito específico, como unidades de procesamiento gráfico (GPU), discos de almacenamiento de gran capacidad, matrices de puertas programables en campo (FPGA), aceleradores para algoritmos de computación cuántica e interfaces de red inteligentes (SmartNIC), entre otros.

La comunicación entre los dispositivos de cómputo y almacenamiento no se limita únicamente a la comunicación dentro de un mismo nodo, ya que algunas aplicaciones requieren comunicarse entre dispositivos ubicados en diferentes nodos del servidor. Por lo tanto, es crucial analizar con precisión la comunicación entre las unidades funcionales dentro de un mismo nodo, así como su impacto en la comunicación entre los nodos. Esto permitirá identificar posibles cuellos de botella.

El objetivo principal de este artículo es configurar un conjunto de nodos de servidor heterogéneos que estén compuestos por unidades de procesamiento gráfico (GPU), dispositivos de almacenamiento de alta velocidad (como memoria no volátil PCI-Express o discos duros NVMe) y dispositivos de interconexión de alto rendimiento. El propósito de esta configuración es permitir una comunicación rápida y eficiente entre los nodos de servidor dentro de un sistema de clúster. Se configurarán los nodos servidores de un clúster real llamado CELLIA (Cluster for the Evaluation of Low-Latency Interconnection Architectures), y se utilizará la red de interconexión de CELLIA para ejecutar aplicaciones paralelas en todo el clúster. Las aplicaciones harán uso distribuido de las GPU y los dispositivos NVMe, tratándolos como un conjunto de recursos de cálculo y almacenamiento. Para evaluar la configuración más adecuada de estos dispositivos, se ejecutarán en CELLIA aplicaciones de entrenamiento de aprendizaje profundo utilizando el framework distribuido Horovod, incluyendo modelos como VGG, ResNet e Inception.

El resto del artículo está organizado de la siguiente manera. En la Sección II, revisamos las tecnologías GPUDirect y NVMeoF. La Sección III describe las herramientas utilizadas para configurar los nodos de cómputo heterogéneo. También describe el clúster utilizado para evaluar el rendimiento de la topología así como las herramientas utilizadas para medir dicho rendimiento. En la Sección IV, se analizan los resultados de rendimiento de varios experimentos realizados en un clúster InfiniBand real. Por último, se extraen algunas conclusiones en la Sección V.

II. Antecedentes y estado del arte

A. GPUDirect

GPUDirect[1] es una tecnología desarrollada por NVIDIA que tiene como objetivo lograr transferencias directas de datos entre procesadores y GPUs, mejorando el rendimiento y eliminando los cuellos de botella en la comunicación entre CPU y GPU. GPUDirect permite la transferencia directa de datos entre las GPUs y los procesadores, evitando la intervención de la CPU en el proceso de transferencia.

[1]e-mail: antonioj.tarraga@uclm.es
[2]Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, e-mail: jesus.escudero@uclm.es
[3]Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, e-mail: francisco.quiles@uclm.es

Esto significa que cuando las operaciones de transferencia de datos son gestionadas por las GPUs, los procesos se completan más rápidamente, lo que aumenta el rendimiento y la escalabilidad del sistema.

GPUDirect se basa en el concepto de peer-to-peer (P2P), que permite la transferencia directa de datos entre GPUs sin tener que pasar por la CPU. Esto hace que la comunicación sea más rápida y eficiente, ya que no se utilizan recursos de la CPU para enviar o recibir datos. La tecnología también permite a los compiladores acceder directamente a la memoria compartida entre la CPU y la GPU, lo que proporciona una gestión de memoria mucho más eficiente.

Para aprovechar al máximo GPUDirect, es necesario configurar cada GPU y procesador con los controladores y bibliotecas de API adecuados. Una vez que se ha establecido el entorno, los desarrolladores pueden acceder fácilmente a las funciones de GPUDirect a través de la API, lo que permite transferencias de datos más rápidas y eficientes entre la CPU y la GPU.

A lo largo de los años, el proyecto ha introducido varias mejoras. Por ejemplo, en el pasado, las transferencias P2P solo funcionaban entre dos GPUs de la misma tarjeta. Con GPUDirect, ahora es posible realizar transferencias entre múltiples GPUs ubicadas en diferentes tarjetas. Esto proporciona una mayor flexibilidad y mejora el rendimiento.

A.1 GPUDirect Storage

GPUDirect Storage es una tecnología que permite a los dispositivos de almacenamiento acceder directamente a la memoria de la GPU, acelerando así la transferencia de datos. Esta tecnología elimina la necesidad de enviar los datos a través del procesador principal, lo que reduce la latencia del sistema, mejora el rendimiento y disminuye la carga en el sistema host. GPUDirect Storage es compatible con dispositivos de almacenamiento NVMe y puede transmitir grandes conjuntos de datos a través de PCIe, lo que mejora el rendimiento de aplicaciones que manejan grandes cantidades de datos. Permite a los desarrolladores crear aplicaciones que acceden directamente a la memoria virtual, procesan datos con baja latencia y aprovechan el ancho de banda de almacenamiento PCIe para mejorar los tiempos de carga y reducir el uso de memoria del sistema.

El almacenamiento GPUDirect se basa en un conjunto de tecnologías que utilizan la conexión PCIe sobre NVMe (PCIe-to-NVMe) y el acceso remoto directo a memoria de datos (RDMA) para establecer una conexión directa entre el dispositivo de almacenamiento y la GPU. La tecnología PCIe-to-NVMe proporciona una ruta de datos directa entre el dispositivo de almacenamiento y la GPU, lo que permite enviar grandes volúmenes de datos de forma simultánea entre la GPU y el dispositivo de almacenamiento sin tener que pasar por el procesador principal (como se ilustra en la Figura 1). Por su parte, la tecnología RDMA permite el acceso directo a la memoria de la GPU desde el dispositivo de almacenamiento, lo que acelera la transferencia de datos.

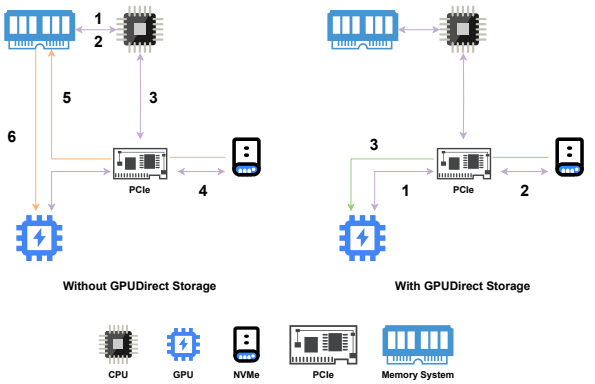


Fig. 1: GPUDirect Storage

A.2 GPUDirect RDMA

GPUDirect RDMA es una tecnología que habilita el acceso remoto a los datos almacenados en la memoria de la GPU de un nodo de computación host desde la memoria de la GPU de un nodo de computación remoto, sin necesidad de copias de datos ni búferes intermedios. Esto se logra al permitir que la memoria de la GPU sea mapeada directamente por la tarjeta de interfaz de red (NIC), lo que posibilita transferencias directas de datos y una latencia mejorada. GPUDirect RDMA fue desarrollado con el propósito de proporcionar transferencias de datos de alto rendimiento entre nodos en sistemas en clúster, y para mejorar el rendimiento de aplicaciones de cómputo de alto rendimiento (HPC) al reducir los costos de comunicación asociados con el flujo de trabajo tradicional de transferencia entre CPU, GPU y red. Esta tecnología resulta fundamental en aplicaciones de supercomputación y centros de datos que requieren una comunicación intensiva, así como para mejorar la eficiencia general al reducir los tiempos de transferencia de datos en la comunicación entre nodos.

GPUDirect RDMA se basa en la tecnología de peer-to-peer (P2P) de NVIDIA, la cual permite la transferencia directa de datos entre dispositivos GPU emparejados, ya sea entre GPU o entre GPU y tarjetas de red. Esta tecnología aprovecha el protocolo de acceso remoto directo a memoria (RDMA) para permitir transferencias de datos directamente desde la memoria de la GPU a la memoria de otra GPU o a una tarjeta de red, sin necesidad de realizar una copia de los datos en un búfer adicional.

B. NVMeoF

NVMeoF (Non-Volatile Memory Express over Fabrics) es una extensión del protocolo de almacenamiento NVMe que se integra con redes como Infiniband y Ethernet para proporcionar un acceso rápido y de baja latencia a recursos de almacenamiento distribuidos. NVMeoF permite el acceso directo a la memoria entre el recurso de almacenamiento y la carga de trabajo, eliminando la necesidad de controladores intermedios. Esto reduce el número de pasos y operaciones entre la solicitud y el proceso, lo que aumenta el rendimiento y reduce significativamente la

latencia. Al utilizar redes con capacidad RDMA, el almacenamiento se puede acceder de forma remota, pero el procesamiento puede realizarse como si fuera local en el dispositivo, lo que brinda un mayor control y flexibilidad. En consecuencia, NVMeoF permite un acceso más eficiente a los datos en comparación con los protocolos tradicionales, lo que lo hace ideal para cargas de trabajo que requieren altos niveles de rendimiento.

Las principales características de NVMeoF son las siguientes:

- **Acceso directo a memoria:** NVMeoF utiliza RDMA para permitir el acceso directo a la memoria entre el recurso de almacenamiento y la aplicación, eliminando intermediarios y controladores, lo cual reduce la latencia, mejora el rendimiento y optimiza el acceso a los datos.
- **Alto rendimiento:** Al utilizar RDMA y el protocolo NVMe, las operaciones se realizan de manera más rápida y eficiente, lo que se traduce en un mayor rendimiento en comparación con otros protocolos de almacenamiento.
- **Baja latencia:** NVMeoF reduce el número de pasos y operaciones entre la solicitud y el proceso, lo que resulta en una latencia reducida y un acceso más rápido a los recursos de almacenamiento.
- **Seguridad:** Las redes Infiniband, que son compatibles con NVMeoF, ofrecen niveles superiores de seguridad y confiabilidad en comparación con otras redes, lo que las hace ideales para entornos que requieren altos niveles de seguridad y disponibilidad.
- **Escalabilidad:** NVMeoF permite una mayor escalabilidad al permitir el acceso remoto al almacenamiento sin comprometer el rendimiento, lo que permite a las empresas expandir sus recursos de almacenamiento sin preocuparse por la latencia o el rendimiento.

III. Metodología y desarrollo

En esta sección, hablaremos de las herramientas y controladores que se han instalado en el clúster CELLIA, así como del proceso para configurar sus dispositivos de manera que las aplicaciones distribuidas aprovechen el hardware disponible.

A. Visión general

La estructura general de este proyecto se muestra en la Figura 2, donde se pueden observar las estructuras de hardware y software utilizadas, así como las aplicaciones utilizadas para evaluar el rendimiento de la configuración del sistema.

En este caso, nos centraremos en configurar los controladores de Mellanox para las HCAs. Además, configuraremos NVMe over Fabrics (NVMEoF) para permitir el acceso a los discos NVMe a través de la red Infiniband. Por último, configuraremos GPUDirect (Storage y RDMA), CUDA y GDRCopy para las GPUs.

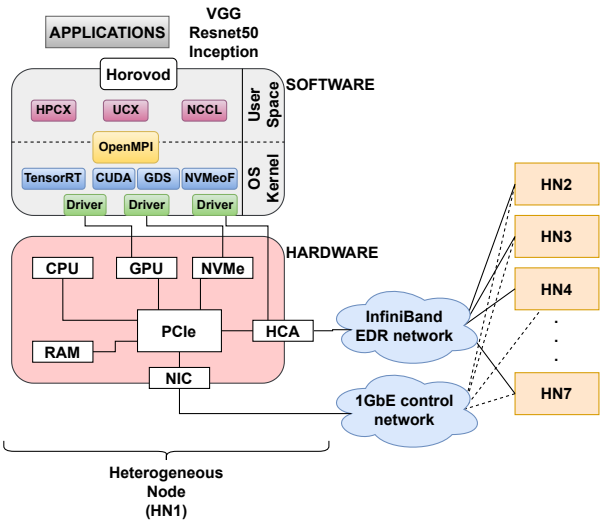


Fig. 2: Visión general.

B. Entorno de configuración

Durante el desarrollo del análisis y la configuración de los nodos de cómputo heterogéneos, se ha utilizado una nueva extensión del clúster CELLIA del grupo de investigación RAAP. En primer lugar, llevaremos a cabo un estudio detallado de las redes de interconexión intra-nodo e inter-nodo que se utilizan en el desarrollo de este artículo.

B.1 Red de interconexión intra-nodo

Dentro de la arquitectura intra-nodo, podemos observar los componentes específicos involucrados en el sistema. Esto se muestra en la Figura 3, donde cada servidor posee:

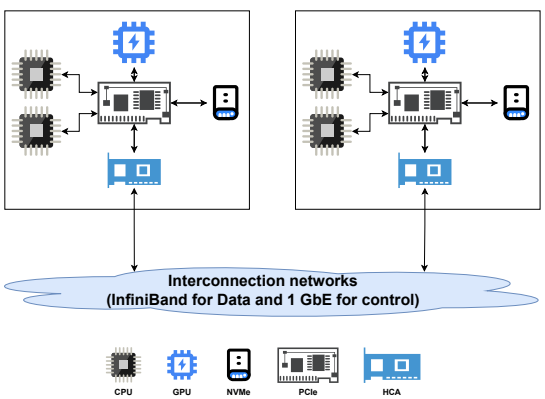


Fig. 3: Diagrama de la arquitectura de dos servidores conectados en el clúster CELLIA.

- Dos procesadores Intel Xeon Silver 4116 [2].
- Una unidad de procesamiento gráfico NVIDIA TU104GL (Tesla T4) sobre PCIe Gen3 x16 [3].
- Interconexión de Componentes Periféricos Express (PCIe) Gen3.
- Disco SSD NVMe Ultrastar serie SN200 sobre PCIe Gen3 x8 [4].
- Tarjeta adaptadora ConnectX-5 VPI, EDR IB (100Gb/s) y 100GbE QSFP28 de doble puerto sobre PCIe Gen3 x16 [5].

Esta arquitectura cuenta con una gran potencia

de cálculo y su estructura heterogénea plantea un desafío para las redes de interconexión, ya que la comunicación puede convertirse en un cuello de botella.

Es necesario gestionar de manera eficiente toda la comunicación entre los componentes y entre los nodos para aprovechar al máximo la potencia de cálculo.

B.2 Red de interconexión inter-nodo

Para obtener información sobre la red de interconexión inter-nodo involucrada, utilizaremos la herramienta *ibnetdiscover* [6]. Esta herramienta está incluida en el conjunto de herramientas de Infiniband y nos proporciona una descripción en formato de texto de la topología de la red Infiniband, incluyendo las HCAs y los switches.

En concreto, obtendremos la topología mostrada en la Figura 4, donde se pueden observar los 10 nodos involucrados en la topología y los 4 switches. En este caso, solo utilizaremos los nodos 51 a 57 debido a que son los únicos que cuentan con una GPU integrada.

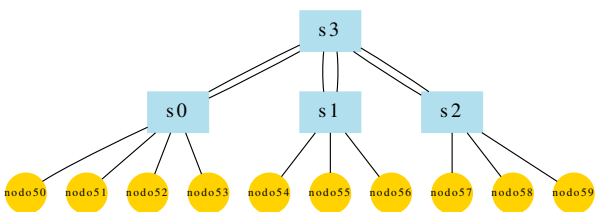


Fig. 4: Topología de red del clúster.

C. Configuración de la red

El primer paso es configurar la red Infiniband, que estará involucrada en el sistema, a través de los controladores específicos, en este caso Mellanox OFED 5.4. Para ello, es necesario habilitar ciertos requisitos importantes, entre los que se incluyen el módulo NVMe over Fabrics, la compatibilidad con el módulo NFS sobre RDMA (NFSoRDMA), la compatibilidad con la tecnología GPUDirect Storage (GDS) y el soporte de módulos dinámicos del núcleo.

D. Configuración de NVMeoF

La configuración de NVMeoF se basa en un modelo de cliente-servidor (ver Figura 5). En su nivel más básico, el modelo cliente-servidor define dos funciones principales que puede asumir NVMeoF: el servidor, que es el dispositivo que proporciona los servicios NVMeoF, y el cliente, que es el sistema anfitrión que envía solicitudes NVMeoF. De esta manera, los clientes pueden ver el dispositivo NVMe de destino en su sistema como si el NVMe estuviera instalado en él.

Una vez configurado el cliente-servidor de NVMeoF, podremos comunicar el disco NVMe del servidor directamente con el cliente, como se muestra en la Figura 6. En ella, se puede observar la comunicación entre un disco NVMe y una CPU sin NVMeoF configurado (ver Figura 6a) y con NVMeoF configurado (ver Figura 6b).

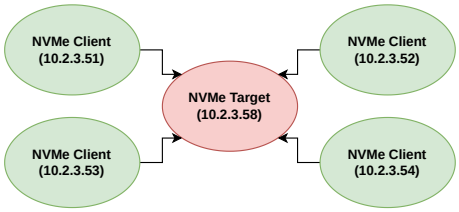
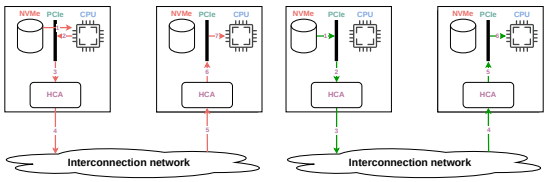


Fig. 5: Ejemplo de cliente-servidor NVMeoF.



(a) Sin NVMeoF (b) Con NVMeoF

Fig. 6: Transferencia de datos desde un dispositivo NVMe remoto a una CPU.

Para configurar el servidor NVMeoF, es necesario exponer su dirección IP a través de los módulos del kernel *nvmet* y *nvmet-rdma* en un subsistema específico. Por otro lado, el cliente necesita la dirección IP y el nombre del subsistema del servidor para establecer la comunicación utilizando el módulo del kernel *nvme-rdma*.

E. Configuración de GPUDirect

A continuación, procederemos a configurar GPUDirect, que permitirá al sistema utilizar la red Infiniband para establecer comunicación directa entre las GPUs y acceder a los datos sin pasar por la CPU. Esta comunicación directa entre GPUs reduce la latencia, ya que no es necesario involucrar a la CPU en el proceso.

E.1 GPUDirect Storage

GPUDirect Storage permite establecer una ruta de datos directa entre el almacenamiento local o remoto y la memoria de la GPU. Esto permite mover los datos hacia o desde la memoria de la GPU sin cargar la CPU. Para configurar GPUDirect Storage, es necesario cumplir con los siguientes requisitos:

- Deshabilitar la Unidad de Gestión de Memoria de Entrada-Salida (IOMMU) para obtener una mayor funcionalidad y rendimiento.
- Instalar el kit de herramientas CUDA 11.8 y GPUDirect Storage.
- [Opcional] Deshabilitar la configuración de ahorro de energía si se está realizando la configuración en un clúster de producción.
- Incluir los binarios del kit de herramientas CUDA en la variable de entorno *PATH*.
- Configurar POWER9 activando el inicio automático del demonio de persistencia de NVIDIA y desactivando la regla udev instalada por defecto.

Una vez instalado GPuDirect Storage y cumplidos todos los requisitos, es necesario verificar que la instalación se ha realizado correctamente. Esto se pue-

de hacer utilizando una herramienta proporcionada por el kit de herramientas CUDA llamada *gdscheck*. La salida de este comando indicará si un sistema de archivos o dispositivo instalado en el sistema es compatible con GDS.

E.2 GPUDirect RDMA

Una característica clave para lograr un sistema de computación totalmente heterogéneo es GPUDirect RDMA, que permite a las GPUs comunicarse entre sí utilizando el protocolo RDMA (Acceso Directo a Memoria Remota). Dado que esta función implica la comunicación a través de la red de interconexión, se deben cumplir ciertos requisitos del sistema.

Los requisitos de plataforma y servidor para GPUDirect RDMA incluyen el uso de HCAs (Adaptadores de Canal de Host) compatibles, GPUs compatibles y software o complementos específicos, como se detalla en la documentación oficial [7]. Si todos estos requisitos se han cumplido, se puede proceder a la instalación de GPUDirect RDMA utilizando el gestor de paquetes correspondiente y se debe iniciar el servicio *nv_peer_mem*.

Una vez instalado GPUDirect RDMA, se puede verificar su correcta instalación utilizando la herramienta *gdscheck*, que proporciona información sobre la configuración y compatibilidad de GPUDirect RDMA en el sistema.

E.3 Configuración de librerías, protocolos y aplicaciones

Una vez que los nodos del clúster CELLIA han sido configurados para utilizar dispositivos de computación heterogénea, es necesario configurar las librerías, protocolos y aplicaciones necesarias para permitir la comunicación entre los nodos y ejecutar aplicaciones distribuidas y paralelas. A continuación se detalla la configuración de las principales librerías y aplicaciones:

E.4 Configuración de GDRCopy

GDRCopy es una herramienta que permite crear mapeos del espacio de usuario de la memoria de la GPU. Es necesario configurar GDRCopy antes que UCX, ya que se necesita especificar la ruta de GDRCopy durante la instalación de UCX.

E.5 Configuración de OpenMPI

OpenMPI es una librería de comunicación de alto rendimiento que permite la comunicación entre los nodos del clúster. Es necesario configurar OpenMPI para permitir la ejecución de aplicaciones en paralelo utilizando las GPUs de forma distribuida.

E.6 Configuración de NCCL

NCCL es una librería que acelera las operaciones colectivas en aplicaciones y frameworks paralelos que utilizan GPUs. Es necesario configurar NCCL con la versión compatible con las demás librerías y aplicaciones. En el caso de utilizar Horovod, se debe asegurar que la versión de NCCL sea compatible con Horovod. Por ejemplo, Horovod puede ser compatible con la versión NCCL 2.11.4 como máximo.

Es importante tener en cuenta que la versión de NCCL debe ser compatible con la versión de CUDA utilizada en el clúster. Se debe elegir una versión de NCCL compatible con CUDA y Horovod, si se va a utilizar.

E.7 Configuración de UCX

UCX es una librería de comunicación que ayuda a reducir la complejidad y optimizar las respuestas de comunicación en aplicaciones paralelas y distribuidas. Es necesario clonar el repositorio oficial de UCX [8] y realizar una configuración personalizada, especificando la ruta de instalación de UCX, la ruta al conjunto de herramientas CUDA y la ruta de GDRCopy.

E.8 Configuración de TensorFlow y TensorRT

TensorFlow y TensorRT son librerías utilizadas en aplicaciones de aprendizaje profundo. Es necesario configurar TensorRT y luego TensorFlow. Se debe seleccionar la versión adecuada de TensorRT según los requisitos de la aplicación. Por ejemplo, si se utiliza Horovod, se debe asegurar que la versión de TensorRT sea compatible con Horovod.

E.9 Configuración de HPC-X

A este punto, nuestro sistema se encuentra completamente configurado, pero aún necesitamos realizar análisis de rendimiento utilizando benchmarks o aplicaciones específicas. Uno de los benchmarks utilizados es HPC-X, una herramienta que nos permite evaluar la conexión entre dos puntos del sistema y analizar parámetros como el ancho de banda y la latencia.

La configuración de HPC-X implica la ejecución del script init proporcionado en el repositorio oficial de HPC-X, así como la carga de HPC-X en el sistema. Además, es necesario exportar la variable *LD_LIBRARY_PATH* con la ruta correspondiente a la librería CUDA lib64 para poder ejecutar pruebas utilizando HPC-X tests.

E.10 Configuración del benchmark FIO

Otra herramienta importante para nuestro análisis de rendimiento es el benchmark FIO, utilizado específicamente para evaluar el rendimiento de NVMeoF. En este trabajo, hemos utilizado una herramienta llamada *fio-plot*, que nos permite generar gráficos utilizando los datos obtenidos a través del benchmark de memoria FIO.

Una vez instaladas estas herramientas, podemos utilizar *fio-plot* junto con el script de benchmark correspondiente, el cual puede ser clonado desde el repositorio asociado.

E.11 Configuración de Horovod

Finalmente, una vez que todas las librerías y herramientas de benchmark han sido configuradas, es necesario utilizar una aplicación real para llevar a

cabo el análisis de rendimiento del sistema. En este trabajo, hemos utilizado Horovod, un framework que nos permite explorar cómo las GPUs aprovechan las capacidades de las redes neuronales convolucionales.

La configuración de Horovod ha sido desarrollada en colaboración con un compañero del grupo de investigación RAAP, en el marco de su proyecto Horovod [9]. En este repositorio, se proporciona un script llamado install.sh, que permite realizar la configuración necesaria para utilizar Horovod en conjunto con NCCL y GPUs, siguiendo las directrices oficiales de Horovod.

IV. Experimentos y resultados

En este capítulo, se presentan los resultados de diversos experimentos realizados en CELLIA con el propósito de evaluar el rendimiento de la configuración de los servidores. En las secciones siguientes, se describen de manera detallada dichos experimentos y se analizan las métricas de rendimiento obtenidas.

Todos los experimentos han sido ejecutados sobre las redes disponibles en el clúster. Una red de control Ethernet 1GbE (1 Gb/s) y una red de datos InfiniBand EDR (100 Gb/s).

A. Rendimiento de la GPU

Con el objetivo de aprovechar la capacidad de cálculo paralelo de las unidades de procesamiento gráfico (GPU, por sus siglas en inglés), se ha utilizado Horovod para evaluar el rendimiento de una única GPU y una única CPU en problemas de aprendizaje profundo.

En la Figura 7 se muestra el rendimiento del modelo resnet50 con un *"batch size"* que varía entre 7 y 189. Se observa que una única GPU puede procesar más de cien imágenes por segundo, dependiendo del *"batch size"*. En contraste, una única CPU no supera las 10 imágenes por segundo, lo cual representa una capacidad inferior en comparación con la GPU. Además, se calcularon las medias y desviaciones estándar de cada situación. Para la CPU, se obtuvo una media de 9.6 ± 0.37 imágenes por segundo, mientras que para la GPU la media fue de 114.92 ± 7.47 imágenes por segundo. Esto indica una diferencia de rendimiento de 105.32 ± 7.84 imágenes por segundo entre la CPU y la GPU.

En resumen, la GPU ofrece un rendimiento significativamente superior al de la CPU, llegando a ser hasta 12 veces más rápido en operaciones de cálculo paralelo. Esto la convierte en una opción ideal para llevar a cabo tareas que requieren un alto grado de paralelismo y capacidad de procesamiento.

B. Experimentos de NVMeoF

NVMeoF (NVMe over Fabrics) proporciona una ruta de acceso a un dispositivo NVMe específico, permitiendo que todos los nodos de cálculo accedan al mismo conjunto de datos. En el experimento, se realizaron 10 ejecuciones para obtener un promedio, utilizando un tamaño de bloque fijo de 4KB.

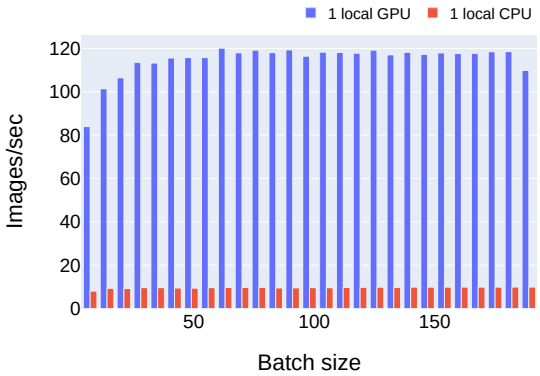


Fig. 7: Comparativa de GPU y CPU con Horovod.

En la Figura 8, se muestra el ancho de banda obtenido para el disco SATA local (alrededor de 100 MB/s) y los discos NVMe local y remoto (alrededor de 5GB/s). Se puede observar que el disco NVMe es más rápido que el disco SATA, y la diferencia entre el acceso a NVMe local y remoto no es relevante. Por lo tanto, acceder a un NVMe remoto a través de NVMeoF permite tener un rendimiento comparable al acceso local. Estos resultados demuestran que la posibilidad de acceder a este NVMe remoto mediante NVMeoF permite compartir el mismo conjunto de datos entre varios nodos sin afectar negativamente el ancho de banda.

C. Experimentos de GPUDirect Storage

GPUDirect Storage permite que la GPU acceda directamente a los datos que pasan a través de la red PCIe, lo cual puede tener un impacto en el ancho de banda y el tiempo de ejecución del sistema. En esta sección, utilizamos la herramienta *gdsio* para evaluar el rendimiento de GPUDirect Storage.

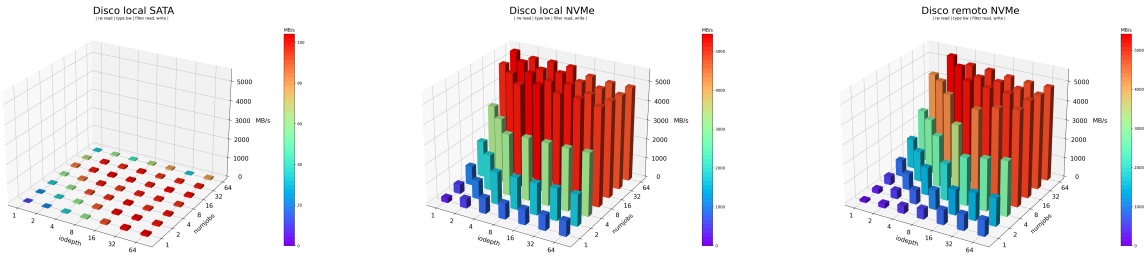
Con el objetivo de probar todas las variables que podrían afectar a nuestro sistema, dividimos estas pruebas en tres secciones principales: comparación del conjunto de datos, tamaño del archivo e hilos de ejecución. Todas las pruebas se realizaron en operaciones de lectura, ya que son las operaciones más frecuentes en modelos de aprendizaje profundo.

C.1 Comparación del conjunto de datos

En las Figuras 10 y 11, se puede observar que el rendimiento no se ve afectado por las variaciones en el tamaño del conjunto de datos. Esto indica que GPUDirect Storage no se ve afectado por el tamaño del conjunto de datos.

C.2 Comparación del tamaño del archivo

En las Figuras 9 y 10, se puede observar que el rendimiento del tamaño del archivo se ve afectado por el número de hilos involucrados en la comunicación. En la Figura 10, se puede observar que con un tamaño de archivo pequeño, GPUDirect Storage ofrece un rendimiento mayor que la transferencia tradicional entre la CPU y la GPU. Además, en la Figura 9, se puede observar que en los escenarios con muchos hilos y un



(a) Disco SATA local (b) Disco NVMe local (c) Disco NVMe remoto

Fig. 8: Gráficos de ancho de banda utilizando la herramienta fio-plot.

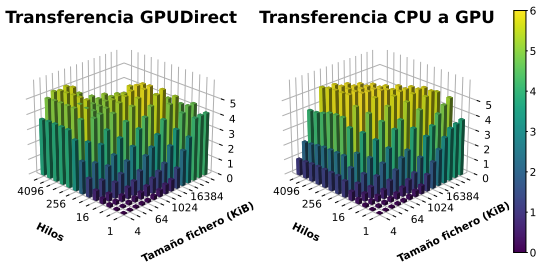


Fig. 9: Comparación de rendimiento (GiB/seg) de GPUDirect Storage con un conjunto de datos de 2 GB.

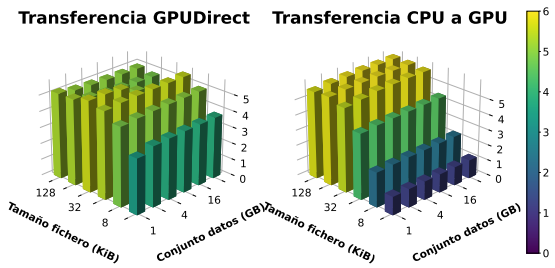


Fig. 10: Comparativa de rendimiento (GiB/seg) de GPUDirect Storage con 512 hilos.

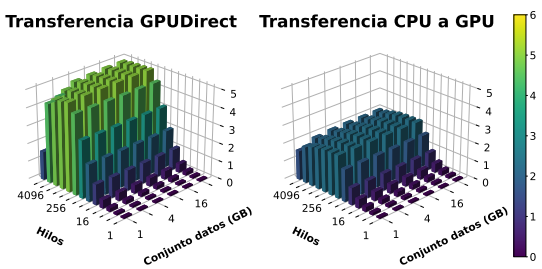


Fig. 11: Comparativa de rendimiento (GiB/seg) de GPUDirect Storage con ficheros de tamaño 8 KB.

tamaño de archivo pequeño, o viceversa, GPUDirect Storage sigue ofreciendo un mejor rendimiento que la transferencia tradicional. Sin embargo, en los casos intermedios en los que se involucran varios hilos con tamaños de archivo medianos, GPUDirect Storage no ofrece un mejor rendimiento que la transferencia tradicional.

Por lo tanto, podemos concluir que en el caso del tamaño del archivo, GPUDirect Storage nos beneficiará en situaciones con muchos hilos y tamaños de archivo pequeños. Esto se puede observar en la Figura 11.

C.3 Comparación de hilos de ejecución

En la comparación de hilos de ejecución, ocurre lo mismo que en el caso del tamaño del archivo: GPUDirect Storage ofrece un mejor rendimiento cuando se tienen muchos hilos y tamaños de archivo pequeños. Esto también se puede observar en la Figura 11.

C.4 Conclusiones

En resumen, podemos concluir que GPUDirect Storage ofrece mejores prestaciones que la transferencia de datos tradicional entre la CPU y la GPU. Esto significa que, con GPUDirect Storage habilitado, el sistema realiza accesos más rápidos y proporciona un mejor rendimiento al aprovechar la potencia de cálculo de la GPU. Esto es especialmente evidente en situaciones con muchos hilos y tamaños de archivo pequeños, donde GPUDirect Storage supera significativamente a la transferencia tradicional. Sin embargo, en casos con tamaños de archivo medianos y un número moderado de hilos, el rendimiento de GPUDirect Storage puede ser similar al de la transferencia tradicional.

En general, la implementación de GPUDirect Storage en el sistema puede mejorar significativamente la eficiencia y el rendimiento de las operaciones de transferencia de datos entre la CPU y la GPU.

D. Experimentos de GPUDirect RDMA

Un desafío importante consiste en lograr la comunicación directa entre GPUs. Esta es una mejora crucial, ya que la transferencia de datos entre la CPU y la GPU suele ser el cuello de botella del sistema. En la actualidad, sigue siendo un desafío en algunos clústeres que desean migrar de CPUs a GPUs debido a problemas de interconexión.

En esta sección, utilizamos la herramienta *HPC-X* para evaluar el rendimiento de GPUDirect RDMA. Realizamos experimentos enfocados en el ancho de banda y la latencia, variando la configuración de hardware del clúster para obtener el máximo rendimiento. El objetivo es investigar cómo GPUDirect RDMA puede mejorar la comunicación entre GPUs y reducir la dependencia de la transferencia de datos a través de la CPU.

D.1 Primer análisis

En primer lugar, al analizar las Figuras 12 y 13, se observa que GPUDirect RDMA permite una comunicación más rápida entre las GPUs. Esto facilita el uso más eficiente de CUDA y mejora el rendimiento del sistema. En términos de ancho de banda, se logra una mejora de rendimiento de 5,82 veces, lo cual es significativo. Sin embargo, esta cifra no es alta en comparación con la capacidad de comunicación de nuestra red, que puede llegar hasta 12.500 MB/s (100 Gbits/s), como se muestra en las Figuras 14 y 15. En este caso, GPUDirect RDMA es aproximadamente 5 veces más lento que la comunicación entre CPUs de diferentes nodos.

Aunque se ha logrado una mejora significativa, no es suficiente para obtener un rendimiento óptimo cuando la comunicación se realiza exclusivamente a través de las GPUs.

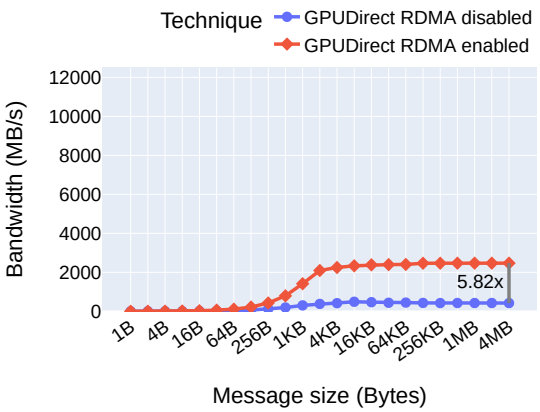


Fig. 12: Comparativa de ancho de banda de GPUDirect RDMA.

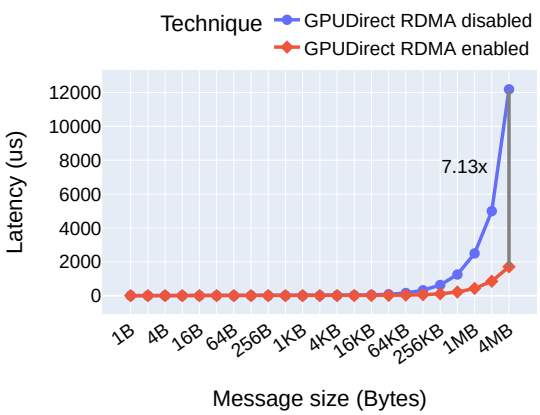


Fig. 13: Comparativa de latencia de GPUDirect RDMA.

D.2 Análisis en diferentes nodos NUMA

Para analizar el rendimiento obtenido, nos hemos enfocado en el servidor HPE ProLiant DL380 [10], que es el sistema utilizado en nuestro estudio.

En este sistema, hay dos nodos NUMA diferentes, uno para el Procesador 1 y otro para el Procesador 2, con diferentes ranuras PCIe, como se muestra en la

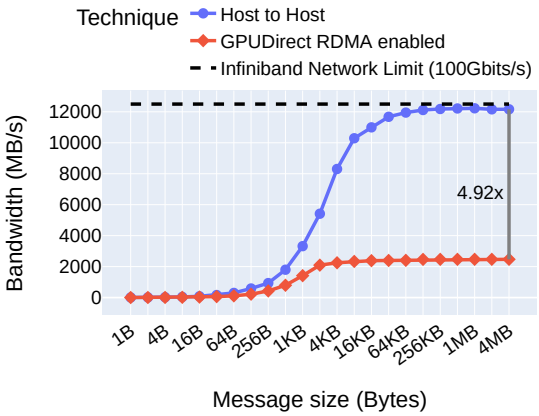


Fig. 14: Comparativa de ancho de banda de GPUDirect RDMA y comunicación entre nodos.

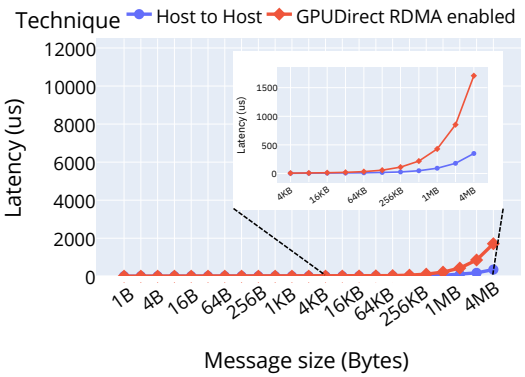


Fig. 15: Comparativa de latencia de GPUDirect RDMA y comunicación entre nodos.

Figura 16. Podemos observar que el riser PCIe primario está conectado a P1, mientras que los risers secundario y terciario están conectados a P2. Esto implica que la HCA (Host Channel Adapter) está conectada a uno de los dos procesadores y la GPU está conectada al otro. Esta configuración de hardware puede afectar el rendimiento del sistema.

Para evitar la comunicación entre nodos NUMA, es necesario conectar la HCA y la GPU en el mismo nodo NUMA. Siguiendo las recomendaciones, se conecta la HCA al riser PCIe secundario y la GPU al terciario, lo que evita la comunicación entre nodos NUMA.

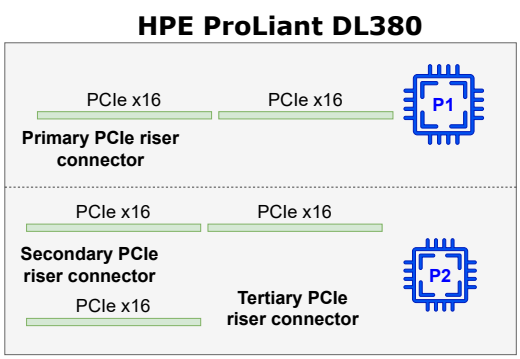


Fig. 16: Componentes de la placa del sistema.

D.3 Comparativa con diferentes configuraciones NUMA

Al analizar las Figuras 17 y 18, podemos observar que esta nueva configuración proporciona un aumento en el ancho de banda de 15,30 veces en comparación con la configuración inicial. Si la comparamos con la configuración inicial que proporciona un ancho de banda de 5,82 veces, esta nueva configuración nos permite comunicarnos casi 3 veces más rápido.

Sin embargo, al comparar esta nueva configuración con la comunicación entre hosts utilizando toda la capacidad de la red, como se muestra en las Figuras 19 y 20, podemos observar que el rendimiento obtenido con GPUDirect RDMA es aproximadamente 1,87 veces más lento. Esto significa que cuando GPUDirect RDMA está habilitado y se utiliza, la comunicación es casi 2 veces más lenta que la comunicación entre nodos. Sin embargo, dado que las GPU son capaces de realizar cálculos mucho más rápidos que las CPU, esta configuración puede ser beneficiosa. Si se utiliza una GPU con un rendimiento de cálculo 2 veces mayor que cualquier CPU, el sistema será más rápido que las CPUs incluso con la comunicación entre nodos.

En resumen, esta configuración final nos permite aprovechar la potencia de cálculo de la GPU hasta 15 veces más rápido que antes, aunque no podamos alcanzar el máximo potencial de rendimiento de la red. Esto supone un factor de mejora significativo y permite al sistema aprovechar el poder de cálculo de la GPU de manera más eficiente.

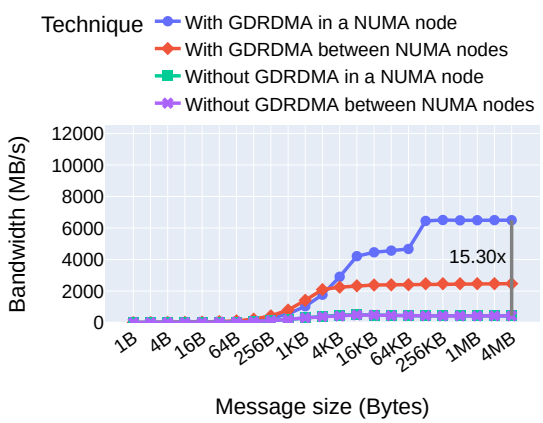


Fig. 17: Comparativa de ancho de banda de GPUDirect RDMA con diferentes configuraciones NUMA.

D.4 Conclusiones

GPUDirect RDMA ha demostrado ser una solución altamente efectiva para mejorar el rendimiento del sistema al permitir una comunicación más rápida y eficiente entre las GPUs. Los experimentos realizados han revelado que GPUDirect RDMA puede aumentar el ancho de banda en un factor de 15x en comparación con las comunicaciones tradicionales entre CPU y GPU.

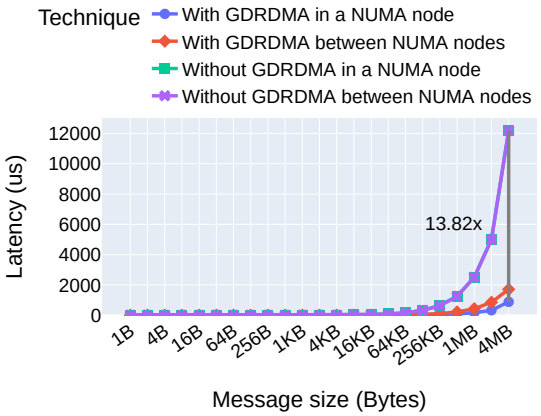


Fig. 18: Comparativa de latencia de GPUDirect RDMA con diferentes configuraciones NUMA.

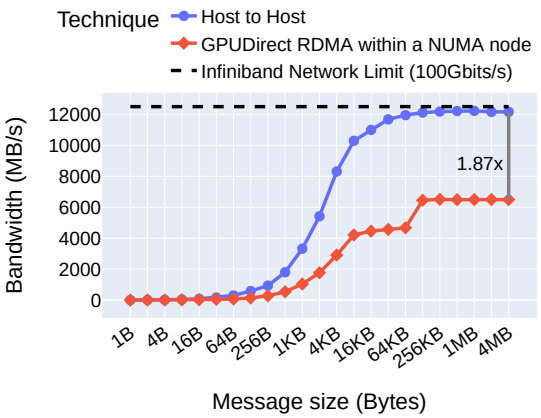


Fig. 19: Comparativa de ancho de banda de GPUDirect RDMA y comunicación entre nodos con diferentes configuraciones NUMA.

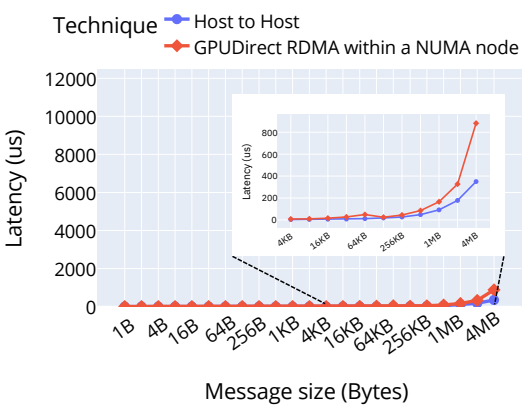


Fig. 20: Comparativa de latencia de GPUDirect RDMA y comunicación entre nodos con diferentes configuraciones NUMA.

E. Conclusiones finales

En este trabajo, se han explorado y evaluado diferentes mejoras y funcionalidades para optimizar el rendimiento de sistemas de computación heterogéneos. Estas mejoras incluyen el acceso a NVMe remotos, la comunicación directa entre dispositivos y la comunicación eficiente entre GPUs.

Los experimentos realizados han demostrado que estas funcionalidades suponen una mejora significa-

tiva en el rendimiento del sistema en comparación con los entornos iniciales. El acceso a NVMe remotos ha permitido a los nodos de cálculo acceder a un conjunto de datos compartido sin penalizar el ancho de banda, lo que facilita el procesamiento de grandes volúmenes de datos de manera eficiente.

Por otro lado, la comunicación directa entre dispositivos, como GPUDirect Storage, ha demostrado ser altamente beneficioso al permitir a la GPU acceder directamente a los datos, evitando cuellos de botella en la transferencia de datos entre la CPU y la GPU. Esto se traduce en un mejor rendimiento y tiempos de ejecución más rápidos para aplicaciones de aprendizaje profundo y cálculos intensivos.

Además, GPUDirect RDMA ha sido una mejora destacada al habilitar la comunicación rápida y eficiente entre GPUs, lo que permite un intercambio de datos más rápido y optimizado en sistemas heterogéneos. Esta funcionalidad ha logrado aumentar el ancho de banda en un factor de 15x, lo que potencia el rendimiento de las aplicaciones que requieren cálculos paralelos intensivos.

En conclusión, la combinación de estas mejoras y funcionalidades ha permitido que el sistema de computación heterogéneo se comunique y procese datos de manera más eficiente, sin penalizar el rendimiento independientemente de si se utiliza la GPU o la CPU. Estas mejoras abren nuevas posibilidades y oportunidades para el procesamiento de datos a gran escala y aplicaciones de aprendizaje automático en entornos de computación heterogéneos.

V. Conclusiones y trabajo futuro

A. Conclusiones

En la actualidad, el desarrollo del aprendizaje profundo, el análisis de big data en tiempo real y los problemas de inteligencia artificial demandan una mayor capacidad de cálculo paralelo. En este contexto, las Unidades de Procesamiento Gráfico (GPU) se han consolidado como una opción real y viable para ejecutar estos modelos de IA de manera eficiente. La computación de alto rendimiento se ha adentrado en esta nueva utilización del hardware, utilizando las GPU en lugar de las Unidades de Procesamiento Central (CPU) para realizar estas tareas. Asimismo, resulta fundamental considerar la transferencia de datos a través de este nuevo enfoque, enfocándose en la elección de dispositivos de almacenamiento de alto rendimiento, como NVMe, y en la rápida transferencia de datos hacia las GPU.

Esta publicación ha demostrado que la comunicación entre la GPU y la CPU constituye un cuello de botella significativo, afectando el rendimiento de las GPU al realizar tareas paralelas. En este trabajo, se ha analizado cómo reducir este cuello de botella en un factor de 15x, lo que ha permitido a las GPU comunicarse entre sí y llevar a cabo ciertas tareas específicas más rápidamente que antes.

En cuanto al acceso a los datos, se ha abordado utilizando NVMeoF, donde varios nodos intentan acceder al mismo conjunto de datos. Este enfoque ha permitido al sistema aprovechar el espacio de almacenamiento y tener los datos en un solo dispositivo, evitando la necesidad de duplicar los mismos datos en discos NVMe en cada nodo. Como se ha demostrado en este estudio, el acceso a NVMeoF no constituye un cuello de botella y permite a los nodos acceder a los datos con un buen rendimiento, como si el NVMe estuviera físicamente presente en ese nodo.

En resumen, podemos afirmar que los nodos de computación heterogéneos representan el futuro de la computación de alto rendimiento, y es crucial aprovechar al máximo sus características de hardware. Para ello, la red de interconexión juega un papel fundamental, donde la comunicación entre elementos debe ser lo más rápida posible, siendo esta una característica clave para lograr un rendimiento óptimo.

B. Trabajo futuro

Para lograr un máximo aprovechamiento del ancho de banda disponible en la red del sistema, se pueden explorar diferentes enfoques, como:

- Mejorar la arquitectura de la red.
- Optimizar los protocolos de comunicación.
- Investigar nuevas tecnologías de almacenamiento.

Agradecimientos

Este trabajo ha sido cofinanciado por el Ministerio de Ciencia español MCIN/AEI/10.13039/501100011033, y la Unión Europea (NextGenertionEU/PRTR) con el proyecto TED2021-130233B-C31, y por la Universidad de Castilla-La Mancha bajo el proyecto 2023-GRIN-34056.

Referencias

[1] "GPUDirect," June 2022, [Online; accessed 29. May 2023].
[2] "Intel® Xeon® Silver 4116 Processor (16.5M Cache, 2.10 GHz)," May 2023, [Online; accessed 23. May 2023].
[3] "NVIDIA T4 Tensor Core GPUs for Accelerating Inference," May 2023, [Online; accessed 23. May 2023].
[4] "NVM Express," Jan. 2023, [Online; accessed 23. May 2023].
[5] "NVIDIA ConnectX-5 InfiniBand/Ethernet Adapter Cards User Manual - ConnectX-5 InfiniBand/Ethernet - NVIDIA Networking Docs," May 2023, [Online; accessed 23. May 2023].
[6] "ibnetdiscover(8): discover InfiniBand topology - Linux man page," May 2023, [Online; accessed 23. May 2023].
[7] "Mellanox OFED GPUDirect RDMA," May 2023, [Online; accessed 23. May 2023].
[8] openucx, "ucx," May 2023, [Online; accessed 23. May 2023].
[9] "Files · GPU · Miguel Sánchez de la Rosa / Horovod. Instalación en CELLIA · GitLab," May 2023, [Online; accessed 23. May 2023].
[10] "HP HPE ProLiant DL380 Gen10," May 2023, [Online; accessed 25. May 2023].

Convolución basada en GEMM para Aprendizaje Profundo en Procesadores Multinúcleo RISC-V

Cristian Ramírez[1], Adrián Castelló[1] Héctor Martínez [2] y Enrique S. Quintana-Ortí[1]

Resumen— **En este trabajo abordamos la implementación eficiente del operador de convolución en la plataforma paralela de ultra-bajo consumo (GAP8 PULP), un diseño multicore equipado con un controlador (FC); un clúster de ocho núcleos de cálculo; y una jerarquía de memoria de cuatro niveles con bloques de memoria de tipo *scratchpad* en lugar de memorias caché asistidas por hardware. Nuestra solución para esta plataforma transforma la convolución en una multiplicación de matrices (gemm) a través del enfoque conocido como *lowering*, demostrando que es posible alcanzar un rendimiento razonable en el GAP8 adaptando cuidadosamente técnicas como el teselado de los operandos matriciales (*blocking*) y el paralelismo de bucles, que son corrientes en la realización multi-thread y consciente de la jerarquía de la caché para gemm.**

Palabras clave— **Multiplicación de Matrices, Alto Rendimiento, RISC-V GAP8**

I. Introducción

La implementación de algoritmos de aprendizaje profundo (*deep learning* o DL) en dispositivos de borde para aplicaciones de Internet de las Cosas (*Internet-of-Things* o IoT) es esencial para mejorar la privacidad y la seguridad. Además, trasladar la computación de la nube a los nodos IoT más cercanos a los sensores puede reducir significativamente la cantidad de datos enviados a través de la red, reduciendo así la latencia y el consumo de energía [1–3]. La gran variedad de aplicaciones IoT, muchas de las cuales dependen de tecnologías DL, ha dado lugar a una amplia gama de arquitecturas de procesadores de borde (*edge*), incluyendo núcleos con ISA RISC-V (arquitectura de conjunto de instrucciones) [4]. Esta diversidad, combinada con severas restricciones de energía, memoria y rendimiento computacional para dispositivos de borde, enfatiza la necesidad de una cuidadosa selección de algoritmos y la optimización del software que se ejecuta en ellos.

En este trabajo, nos centramos en la implementación de redes neuronales profundas (*deep neural networks* o DNNs) convolucionales en procesadores de borde. Con este objetivo, paralelizamos uno de los algoritmos más popular para la convolución basado en el enfoque de reducción (*lowering*), que descompone la operación en una transformación lineal de replicación de datos, conocida como IM2COL o IM2ROW, seguida de una multiplicación general de matrices (GEMM) [5]. Además, abordamos la arquitectura heterogénea compuesta por 1+8 núcleos RISC-V integrados en la plataforma de ultra-bajo consumo en paralelo GAP8 PULP para IoT. En más detalle, este documento hace las siguientes contribuciones:

- Desarrollamos una implementación de alto rendimiento y multithread de GEMM que opera con datos y aritmética de 8 bits enteros (INT8) sobre el producto dot (escalar), una instrucción que recibe un tratamiento especial en el ISA del GAP8. En nuestra solución, los 8 núcleos de cálculo del GAP8 se encargan de toda la aritmética, mientras que el núcleo restante, conocido como el controlador (*fabric controller* o FC), coordina los movimientos de datos.
- Orquestamos una cuidadosa secuencia de transferencias de datos a través de las áreas de memoria del GAP8 mediante transferencias DMA, insertando estos movimientos en las técnicas de teselado de un algoritmo bloqueado en paralelo para GEMM.
- Realizamos una evaluación experimental completa de la convolución para las dos transformaciones mencionadas anteriormente: IM2COL y IM2ROW.

El resto del documento se estructura de la siguiente manera. En la Sección II presentamos brevemente el operador de convolución y en la Sección III revisamos la implementación de alto rendimiento de GEMM en procesadores multicore con una memoria multinivel que incluye cachés. En la Sección IV detallamos las principales características de la plataforma GAP

II. Convolución a través de IM2COL+GEMM

En este apartado, como primera parte introducimos la operación de convolución [6] para luego presentar las transformaciones IM2COL y IM2ROW que, combinadas con GEMM, potencialmente proporcionan un enfoque de alto rendimiento para computar este operador, a costa de un espacio de trabajo aumentado y algunas copias de datos [5].

A. Convolución

El proceso de inferencia para una cap convolucional (Conv) requiere la aplicación de un operador de convolución. Esta operación recibe un tensor de activación de entrada I, de dimensión $b \times c_i \times h_i \times w_i$, donde b es el número de imágenes de entrada o muestras (también conocido como tamaño del lote), c_i especifica el número de canales de imagen de entrada, y $h_i \times w_i$ son la altura,×anchura de la imagen de entrada. Además, la convolución también recibe un

[1]Universitat Politécnica de Valencia, e-mail: crirabe@upv.es, {adcastel,quintana}@disca.upv.es
[2]Universidad de Córdoba, e-mail: {el2mapeh}@uco.es

```
1  for (i=0; i<b; i++)
2   for (j=0; j<ci; j++)
3    for (k=0; k<wo; k++)
4     for (l=0; l<ho; l++)
5      for (m=0; m<wf; m++)
6       for (n=0; n<hf; n++)
7        for (o=0; o<co; o++)
8          O[i][o][l][k]
9            += F[o][ic][n][m]
10            * I[i][j][s*l + n][k*s + m];
```

Fig. 1: Algoritmo directo para la aplicación del operador de convolución $O = \text{CONV}(F, , I)$.

```
1  for ( i=0; i<ci; i++ )
2    for ( j=0; j<hf; j++ )
3      for ( k=0; k<wf; k++ ) {
4        r = i*j*k + j*k + k;
5        for ( l=0; l<b; l++ )
6          for ( m=0; m<ho; m++ )
7            for ( n=0; n<wo; n++) {
8              c = l*m*n + m*n + n;
9              B[r][c] = I[l][i][m*s + j][n*s + k
       ];
10     } }
```

Fig. 2: Algoritmo para la transformación IM2COL.

tensor de filtro (o núcleo) de entrada F, de dimensión $c_o \times c_i \times h_f \times w_f$, donde c_o es el número de filtros y $h_f \times w_f$ denotan la altura×anchura del filtro. Siguiendo con la definición del operador:

$$O = \text{CONV}(F, , I), \tag{1}$$

devuelve el tensor de activación de salida O, de dimensión $b \times c_o \times h_o \times w_o$, donde c_o especifica el número de canales de salida y $h_o \times w_o$ son la altura×anchura de la imagen de salida.

El algoritmo en la Figura 1 proporciona una *realización directa* del operador de convolución. Allí, cada filtro individual combina un subconjunto de las entradas, con la misma dimensión que el filtro, para producir un único valor escalar (o entrada) en una de las c_o salidas. Al aplicar repetidamente el filtro a toda la entrada, con un cierto desplazamiento horizontal/vertical s, el operador de convolución obtiene las entradas de esta única salida [6]. Asumiendo factores de relleno vertical y horizontal dados por p_h y p_w, respectivamente, las dimensiones de altura×anchura de salida vienen dadas por $h_o \times w_o = \lfloor (h_i - h_f + 2p_h)/s + 1 \rfloor \times \lfloor (w_i - w_f + 2p_w)/s + 1 \rfloor$.

B. *Convolución indirecta a través de las transformaciones* IM2COL/IM2ROW

En las arquitecturas actuales, el rendimiento del algoritmo directo en la Figura 1 está fuertemente restringido por el ancho de banda de la memoria y, por lo tanto, en general este enfoque sólo proporciona una fracción del rendimiento máximo de coma flotante del procesador. En la práctica, este inconveniente se suele abordar adoptando un *enfoque indirecto o basado en* GEMM que representa este operador en términos de una multiplicación de matrices a través de las transformaciones IM2COL o IM2ROW [5]. Esta realización se conoce a menudo como el *algoritmo de reducción* o *lowering*.

En detalle, la Figura 2 muestra el algoritmo IM2COL[1] que "aplana" el tensor de entrada de 4 dimensiones (4D) I en una matriz aumentada (de 2 dimensiones, 2D) $\hat{B}$ de manera que la salida de la convolución puede obtenerse a partir de la GEMM

$$\hat{C} = \hat{A} \cdot \hat{B},$$

donde $\hat{C} \equiv O \rightarrow m \times n = c_o \times (h_o \cdot w_o \cdot b)$ es la salida de la convolución (vista como una matriz 2D, con $m = c_o$ y $n = h_o \cdot w_o \cdot b$); $\hat{A} \equiv F \rightarrow m \times k = c_o \times (h_f \cdot w_f \cdot c_i)$ contiene una organización 2D de los filtros; y $\hat{B} \rightarrow k \times n = (h_f \cdot w_f \cdot c_i) \times (h_o \cdot w_o \cdot b)$ es la matriz aumentada mencionada anteriormente.

Por simplicidad, el algoritmo mostrado en la Figura 2 no tiene en cuenta el acceso a la memoria cuando el desplazamiento de la convolución es mayor que uno. Además, la implementación real de esta transformación elimina algunos de los invariantes de bucle dentro de algunos de los bucles para reducir la sobrecarga de indexación.

III. ALGORITMOS POR BLOQUES PARA GEMM

A partir el operador de convolución aplanado en una multiplicación de matrices a través de la transformación IM2COL (o IM2ROW), este apartado revisa la estrategia convencional para obtener una realización de alto rendimiento de GEMM en arquitecturas de procesadores con jerarquías profundas de memoria caché y unidades vectoriales de múltiples instrucciones de datos simultáneos (SIMD).

A. *El algoritmo base para GEMM*

Las implementaciones de alto rendimiento actuales de GEMM, tanto en bibliotecas de álgebra lineal de código abierto como comerciales, siguen a GotoBLAS [7] para formular este núcleo computacional como una colección de cinco bucles anidados alrededor de dos rutinas de empaquetado y un *microkernel*; véase la Figura 3 (izquierda). En detalle, las instancias de GEMM en estas bibliotecas aplican un enfoque de bloque (o *tiling*) de la siguiente manera:

- Un bloque de $k_c \times n_c$ de la matriz B se empaqueta en un búffer B_c, destinado a residir en la memoria caché L3 (o en la memoria principal, en caso de que no haya caché L3); véase la línea 4 en el algoritmo.
- Un bloque de $m_c \times k_c$ de la matriz A se empaqueta en un búffer A_c, destinado para la memoria caché L2; línea 7.
- Durante la ejecución del microkernel (líneas 11–14), se espera que un bloque específico de $k_c \times n_r$ de B_c, denominado micropanel B_r, se encuentre en la memoria caché L1.
- El microkernel realiza la aritmética, en principio accediendo a los datos para A_c desde la caché L2, para B_r desde la caché L1, y para C directamente desde la memoria principal.

[1] El algoritmo para la transformación IM2ROW es básicamente una variante de IM2COL, con los roles de $\hat{A}$ y $\hat{B}$ y las correspondientes dimensiones intercambiadas.

```
1  for (jc=0; jc<n; jc+=nc)          // Loop L1
2    for (pc=0; pc<k; pc+=kc) {      // Loop L2
3      // Pack B
4      Bc := B(pc:pc+kc-1,jc:jc+nc-1);
5      for (ic=0; ic<m; ic+=mc) {    // Loop L3
6        // Pack A
7        Ac := A(ic:ic+mc-1,pc:pc+kc-1);
8        for (jr=0; jr<nc; jr+=nr)   // Loop L4
9          for (ir=0; ir<mc; ir+=mr) // Loop L5
10           // Micro-kernel
11           for (kr=0; kr<kc; kr++) // Loop L6
12             C(ic+ir:ic+ir+mr-1,jc+jr:jc+jr+nr-1)
13               += Ac(ir:ir+mr-1,kr)
14               *  Bc(kr,jr:jr+nr-1);
15
16
17   }}
```

```
1  for (jc=0; jc<n; jc+=nc)          // Loop L1
2   for (pc=0; pc<k; pc+=kc) {       // Loop L2
3    // Pack B
4    Bc := B(pc:pc+kc-1,jc:jc+nc-1);
5     for (ic=0; ic<m; ic+=mc) {    // Loop L3
6      // Pack C
7      Cc := C(ic:ic+mc-1,jc:jc+nc-1);
8      for (pr=0; pr<kc; pr+=kr)   // Loop L4
9        for (ir=0; ir<mc; ir+=mr) // Loop L5
10         // Micro-kernel
11           for (jr=0; jr<nc; jr++) // Loop L6
12             Cc(ir:ir+mr-1,jr)
13               += A(ic+ir:ic+ir+mr-1,
14                    pc+pr:pc+pr+kr-1)
15                * Bc(pr:pr+kr-1,jr);
16       // Unpack C
17       C(ic:ic+mc-1,jc:jc+nc-1) := Cc;
18   }}
```

Fig. 3: Algoritmos B3A2C0 (izquierda) y B3C2A0 (derecha) para GEMM.

Las transferencias de datos a través de la jerarquía de memoria están ilustradas en la Figura 4. Además, el empaquetado de A_c, B_c de la forma mostrada en la Figura 5 asegura que sus entradas sean recuperadas con una unidad de paso (o *stride*) desde el microkernel.

El algoritmo base para GEMM, también conocido como B3A2C0,[2] cuenta con un microkernel que incluye el sexto bucle, iterando sobre la dimensión k_c. Este componente del algoritmo es el único codificado directamente en ensamblador o en C con instrucciones vectoriales; véase la Figura 3 (izquierda). En cada iteración del bucle, el microkernel actualiza un microbloque de $m_r \times n_r$ de C, digamos C_r, realizando un producto exterior que involucra (parte de) una fila del búffer A_c y una columna del micropanel B_r.

B. *Algoritmos alternativos para GEMM*

Reorganizando los bucles de GEMM en el algoritmo base en la Figura 3 (izquierda) en un orden distinto, combinado con una selección apropiada de los pasos de bucle, podemos obtener diferentes variantes algorítmicas de GEMM, que favorecen que ciertos bloques de A, B, C residan en distintos niveles de la jerarquía de memoria [8–10]. Por ejemplo, la Figura 3 permite una comparación visual entre los códigos para las variantes B3A2C0 (base) y B3C2A0, exponiendo implícitamente las siguientes diferencias principales entre las dos:

- En B3C2A0, un bloque de $m_c \times n_c$ de C se empaqueta en un búffer C_c para la caché L2; véase la línea 7 en el algoritmo. Además, esta variante también requiere un paso de desempaquetado que mueve las entradas de C_c a C una vez que se ha ejecutado el microkernel; línea 16.
- Para asegurar que las entradas de C, B se accedan con una unidad de paso desde el microkernel para B3C2A0, tanto C_c como B_c se almacenan siguiendo el mismo patrón mostrado para A_c en la Figura 5, con las entradas de C_c organizadas

[2]La notación introducida en [8] se refiere al algoritmo base como B3A2C0, donde cada letra denota uno de los operandos de la matriz, y el número indica el nivel de caché donde reside ese operando (con 0 refiriéndose a los registros del procesador). El mismo operando de matriz reside tanto en las cachés L1 como L3.

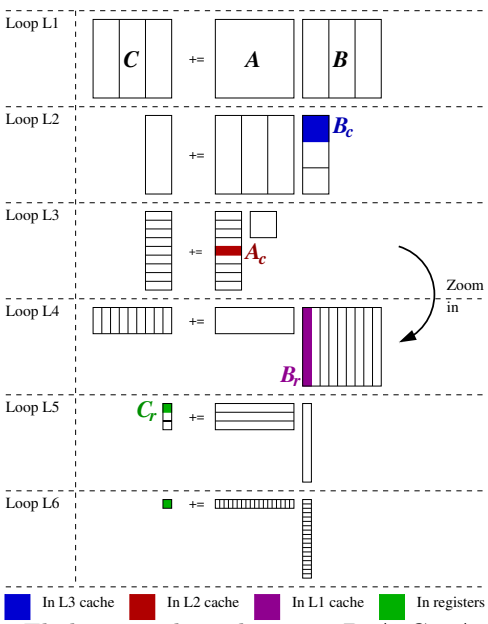


Fig. 4: El algoritmo base de GEMM B3A2C0. Aquí C_r es un artefacto de notación, introducido para facilitar la presentación del algoritmo mientras que A_c y B_c son búfferes reales que mantienen copias de ciertos bloques de A y B.

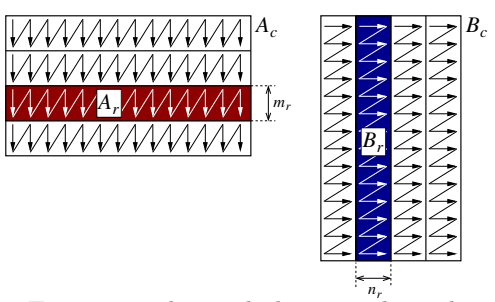


Fig. 5: Empaquetado en el algoritmo base de GEMM B3A2C0. Nótese cómo las entradas de A, B son reorganizadas en A_c, B_c en micropaneles de m_r filas, n_r columnas, respectivamente.

en micropaneles de m_r filas y las de B_c en micropaneles de k_r filas.
- El microkernel para B3C2A0 opera con un microbloque de $m_r \times k_r$ de A transmitido directamente desde la memoria a los registros, donde residirá durante la ejecución completa del microkernel. Este realiza un pequeño producto matriz-vector de $m_r \times k_r$ por iteración del bucle `L6` (para un total de n_c iteraciones), cada uno involucrando una única columna de un micropanel C_r y una única columna de un micropanel B_r; líneas 11–14.

Como discutiremos en el apartado V, la variante B3C2A0 presenta varias características que la hacen especialmente interesante para su implementación en la plataforma GAP8.

IV. La Plataforma GAP8

GAP8 es una plataforma comercial diseñada para aplicaciones de IoT, con un procesador basado en la arquitectura PULP [11]. Como se muestra en la Figura 6, el procesador GAP8 incorpora tres componentes de computación principales: 1) Una unidad de microcontrolador de bajo consumo (MCU), conocida como FC, que es responsable de gestionar las funciones de control, comunicaciones y seguridad; 2) un motor de cálculo (CE) que comprende un clúster de 8 núcleos de cálculo específicamente diseñados para la ejecución de algoritmos paralelos; y 3) un acelerador de hardware especializado (HWCE).

El FC integra una memoria de solo lectura (ROM) que almacena el código de arranque principal, además de un scratchpad privado de 16 KB `L1` (también conocido como área de memoria o MA). En el lado del CE, los núcleos de cálculo y el HWCE comparten un scratchpad `L1` de memoria de datos acoplada estrechamente (TCDM) de 64 KB de múltiples bancos (MA). Además, FC y CE comparten un scratchpad `L2` de 512 KB. El dispositivo también incluye una MA `L3` de 8 MB que actúa como la memoria principal de la plataforma y es accesible desde el FC. Para permitir transferencias rápidas de datos entre las MAs, la plataforma cuenta con dos unidades de acceso directo a memoria (DMA). Una de estas unidades asiste en la transferencia de datos entre el dominio FC y el dominio CE, mientras que la unidad micro-DMA transfiere datos hacia/desde los periféricos, incluyendo la MA `L3`.

Tanto el FC como los núcleos del clúster soportan la arquitectura de conjunto de instrucciones RISC-V *RV32IMCXpulpV2*, que cuenta con aritmética de enteros (I), instrucciones comprimidas (C), extensiones de multiplicación y división (M), y una parte del subconjunto ISA de supervisor. La extensión ISA *RV32ICMXpulpV2* incluye instrucciones especializadas para bucles de hardware sin sobrecarga, accesos a memoria post/pre-modificados por punteros, instrucciones que mezclan el flujo de control con el cálculo, multiplicar/restar y acumular, operaciones vectoriales, operaciones en punto fijo, manipulación de bits y el *producto escalar de dos vectores*.

V. Adaptando GEMM para GAP8

En este apartado, describimos nuestra adaptación de GEMM para ejecutarse en paralelo en los núcleos de cómputo integrados en el CE GAP8. Este esfuerzo de personalización está fuertemente determinado por las siguientes características de la plataforma GAP8:

- Las unidades aritméticas en los núcleos de cómputo tienen soporte de hardware especial para el producto escalar.
- La jerarquía de memoria en la plataforma se estructura en cuatro niveles: registros vectoriales, dos niveles intermedios de scratchpad (`L1`, `L2` MAs), y una memoria principal (también conocida como RAM o `L3` MA).
- El sistema integra scratchpads en lugar de memorias caché convencionales.
- Un solo FC controla las transferencias de memoria entre la memoria principal y los scratchpads `L1`, `L2`.
- El CE cuenta con 8 núcleos de cálculo.

A. Microkernel para B3C2A0

Un primer aspecto a tener en cuenta es que, dado que el FC y los núcleos de cálculo admiten la misma ISA orientada a RISC-V, incluyendo las instrucciones especializadas para el producto escalares y los registros vectoriales, adaptar el microkernel inicial del FC de [12] a los núcleos del cluster básicamente no requirió cambios. Para ilustrar esto, la Figura9 muestra una versión simplificada de un microkernel que opera con un microbloque de 4×4 A_r, implementando el bucle interno en el algoritmo B3C2A0 (ver líneas 11-14 en la Figura 3 a la derecha) de la siguiente manera:

- El microkernel recibe como parámetros de entrada 1) la dirección de inicio en la memoria principal del microbloque A_r (parámetro `Ar`), que se asume almacenado en orden de filas; 2) la dimensión principal de la matriz operando A (parámetro `ldA`); 3) la dirección de inicio del micropanel B_r (parámetro `Br`) en la MA `L1` del CE; y 4) la dirección de inicio del micropanel C_r incrustado en C_c (parámetro `Cc`) en la MA `L2` compartida con el FC.
- A continuación, el código para el microkernel incluye las declaraciones de variables correspondientes para datos escalares y vectoriales (líneas 4-6). El tipo de datos para estos últimos es `v4s`, que identifica un vector con capacidad para cuatro números INT8.
- Luego, el código carga las cuatro filas (con cuatro números INT8 cada una) del microbloque de 4×4 A_r en el mismo número de registros vectoriales: `A0`, `A1`, `A2`, `A3` (líneas 9-10).
- En cada iteración del bucle principal (línea 12), el microkernel carga una columna del micropanel C_r (cuatro números INT8) en el registro vectorial `cr` y una columna del micropanel B_r (cuatro números INT8) en el registro vectorial `br` (líneas 14-15).

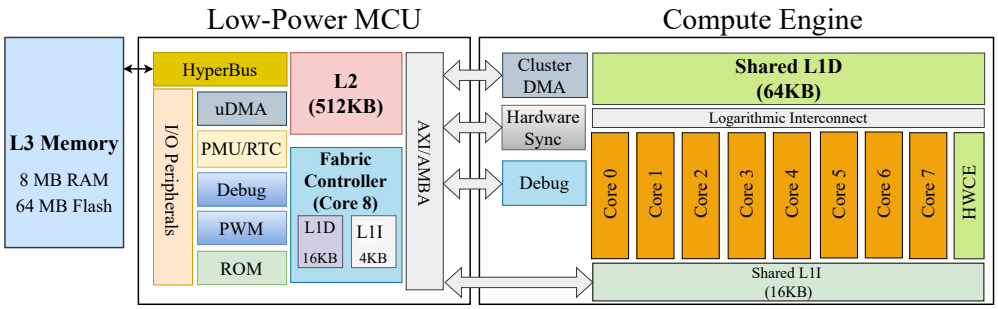


Fig. 6: Disposición del GAP8.

- Dentro del bucle, el microkernel procede a multiplicar los contenidos del microbloque A_r con la columna de B_r, actualizando la columna de C_r mediante cuatro productos escalares (líneas 18-19); y almacenando los cuatro elementos INT8 en la columna de C_r de nuevo en la MA L2 (líneas 22-23).

En este punto, notamos que hemos implementado y probado microkernels de diferentes "formas"(o dimensiones) $m_r \times k_r$. Además, mientras que el mismo microkernel puede ejecutarse básicamente en el FC y en un solo núcleo de cálculo, *los datos para los operandos de la matriz deben colocarse en las MAs apropiadas, que son diferentes dependiendo de qué componente (FC o núcleo de cálculo) tiene que ejecutar el microkernel.* Discutimos este punto en detalle a continuación.

```
1  gemm_ukernel_gap8(int nc, signed char *A, int
       ldA,
2                            signed char *Br,
3                            signed char *Cc) {
4  int jr, baseCB=0;
5  v4s A0, A1, A2, A3, // Rows of the micro-tile
       Ar
6      br, cr;          // Columns of Br, Cr
7
8  // Load the micro-tile Ar into vector
       registers
9  A0=*((v4s *) (&A[0]);      A1=*((v4s *) (&A[ldA
       ]);
10 A2=*((v4s *) (&A[2*ldA]); A3=*((v4s *) (&A[3*
       ldA]);
11
12 for ( jr = 0; jr < nc; jr++ ) { // Loop L6
13   // Load jr-th columns of Cr, Br into vector
       registers
14   cr=*((v4s *) (&Cr[baseCB]);
15   br=*((v4s *) (&Br[baseCB]);
16
17   // Update i-th entry of cr as cr[i]+=Ai*br
18   cr[0]+=gap8_dotp4(A0, br); cr[1]+=gap8_dotp4
       (A1, br);
19   cr[2]+=gap8_dotp4(A2, br); cr[3]+=gap8_dotp4
       (A3, br);
20
21   // Store the column of Cr in memory
22   Cr[baseCB+0]=cr[0]; Cr[baseCB+1]=cr[1];
23   Cr[baseCB+2]=cr[2]; Cr[baseCB+3]=cr[3];
24
25   baseCB+=4; // Prepare for next iteration
26 }}
```

Fig. 7: Realización simplificada de la implementación secuencial de un microkernel con un microbloque $m_r \times k_r = 4 \times 4$ de A residente en los registros del procesador (FC o núcleo de cálculo) para la variante B3C2A0 de GEMM.

B. *Transferencias de datos a través de la jerarquía de memoria*

La plataforma GAP8 integra las memorias intermedias L1 y L2, las cuales le dan al programador control total sobre las transferencias de datos a través de la jerarquía de memoria, pero también la responsabilidad de orquestarlas. Nuestra solución dirigida a los núcleos de cálculo en el CE aborda esta tarea incorporando los movimientos de datos de manera natural en el algoritmo B3C2A0 de la siguiente manera:

- El FC empaqueta B en el buffer B_c, con ambos operandos de datos residiendo en la memoria principal. Por lo tanto, este movimiento de datos solo involucra el "hardware" en la MCU (memorias intermedias y núcleo).
- El FC empaqueta los datos para C en C_c, en este caso transfiriéndolo de la memoria a la MA L2 en la MCU. La transferencia para el desempaquetado también es gobernada por el FC, pero obviamente se lleva a cabo en la dirección opuesta. Nuevamente, estas copias solo involucran la parte MCU de GAP8.
- El FC copia el micropanel B_r de B_c, de la MA L2 en la MCU a la MA L1 en el CE.
- El núcleo de cálculo que ejecuta un microkernel espera que los datos para B_r residan en la MA L1, para C_c en la MA L2, y para A en la memoria principal. Sin embargo, el CE no puede acceder directamente a los datos en la memoria principal, por lo que el FC copia los microkernels apropiados de A desde allí a la MA L1 en el CE. A continuación, dentro del microkernel, cada núcleo transmite los datos de su propio microbloque de la L1 a sus registros vectoriales.

Los movimientos de datos requeridos para la colaboración de FC y núcleos de cálculo se ilustran gráficamente en la Figura 8. Aunque el gráfico allí muestra la ejecución del algoritmo paralelo, a ser discutido a continuación, los movimientos de datos en el caso del algoritmo secuencial son básicamente los mismos, y se pueden derivar considerando los movimientos que involucran solo al Núcleo 1.

Para cerrar este subapartado, recordamos que los pasos de bucle para el algoritmo B3C2A0 se establecen en n_c, k_c, m_c, m_r, k_r (respectivamente para los bucles L1, L2, ..., L5; ver Figura 3 (derecha)). Las dos últimas variables, m_r, k_r, determinan la forma del microkernel y suelen ajustarse dependiendo del

número de registros vectoriales por núcleo. Las primeras tres variables son conocidas como los parámetros de configuración de caché, y deben establecerse de acuerdo con las dimensiones de los niveles de memoria L1, L2 y L3, así como la forma del microkernel. Para un algoritmo secuencial dirigido a un único núcleo de cálculo de la plataforma GAP8, tenemos que tener en cuenta que

$$\begin{aligned} k_r \times n_c + m_r \times k_r &\leq \mathsf{CL1}, \\ m_c \times n_c &\leq \mathsf{CL2}, \end{aligned}$$

donde CL1, CL2, denotan respectivamente la capacidad de las MAs L1, L2 a las que accede el núcleo de cálculo.

C. Parallelization

Siguiendo el enfoque convencional para la realización de múltiples hilos de GEMM, explotamos el paralelismo de bucle para el algoritmo B3C2A0. La primera pregunta es, por tanto, ¿a cuál de los seis bucles que aparecen en el algoritmo se debe apuntar? Para tomar esta decisión, hacemos las siguientes observaciones sobre el código en la Figura 3 (derecha):

- Paralelizar el bucle `L1` (indexado por `jc`) divide la operación en una colección de núcleos GEMM independientes. La consecuencia es que esto requiere espacios de trabajo separados por hilo para los búfferes C_c, B_c, B_r en cada nivel de memoria, dividiendo en la práctica la capacidad de estas memorias entre los hilos. Dado que los núcleos de cálculo en el CE comparten las MAs L1, L2 y, obviamente, la memoria principal, esto no parece el mejor enfoque para una colaboración eficiente.
- Un algoritmo paralelo que se dirige a los bucles `L2` (indexado por `pc`) o `L4` (indexado por `pr`) enfrenta condiciones de carrera porque varios hilos pueden actualizar las mismas partes de la matriz de salida al mismo tiempo. Es posible controlar este tipo de comportamiento utilizando varias técnicas de software (y, en algunos casos, mecanismos de hardware), pero en general introducen una sobrecarga no despreciable.
- Paralelizar el bucle `L3` (indexado por `ic`) requeriría un búffer separado por hilo para C_c, B_r. Por tanto, por las mismas razones que se expusieron para el bucle `L1`, esto no parece una buena opción desde una perspectiva de colaboración inter-núcleo.

Este análisis deja solo los bucles `L5` (indexado por `ir`) o `L6` (indexado por `jr`), con este último dentro del microkernel, como posibles candidatos. Para aumentar la granularidad de la distribución de la carga de trabajo y reducir la sincronización de hilos, elegimos la opción más externa: el bucle `L5`. El objetivo de memoria de los distintos operandos y búfferes y los movimientos de datos para el algoritmo paralelo se ilustran en la Figura 8. Hay que tener cuenta que, con el esquema de paralelización seleccionado, todos los núcleos de cálculo acceden al mismo búffer C_c de tamaño $m_c \times n_c$ en la MA L2, el mismo micropanel B_r de tamaño $k_r \times n_c$ en la MA L1, pero un microtile A_r de tamaño diferente $m_r \times k_r$ en la MA L1. Por esta razón, en el caso paralelo, la forma del microkernel y los parámetros de configuración de la memoria caché deben cumplir con las siguientes condiciones:

$$\begin{aligned} k_r \times n_c + c\,(m_r \times k_r) &\leq \mathsf{CL1}, \\ m_c \times n_c &\leq \mathsf{CL2}, \end{aligned}$$

donde c especifica el número de núcleos que participan en la ejecución paralela.

Para aprovechar al máximo las capacidades del procesador GAP8 y mejorar el rendimiento general, distribuimos el espacio de iteración del bucle `L5` de manera uniforme entre los 8 núcleos de cálculo RISC-V en el GAP8 CE.

La Figura 9 muestra el fragmento del código paralelo que comprende el bucle `L5` junto con la invocación al microkernel. Las principales diferencias con la versión secuencial son las siguientes:

- Todos los núcleos de cálculo del clúster iteran sobre el bucle `ir`, pero cada núcleo solo ejecuta sus propias iteraciones. La carga de trabajo se distribuye siguiendo una política simple *round-robin* con bloques de `mr` filas (ver líneas 10–11 en el algoritmo).
- El núcleo encargado de ejecutar un microkernel determinado instruye al FC para copiar el microtile A_r de tamaño $m_r \times k_r$ desde la memoria a la MA L1 (líneas 14–18).
- El núcleo luego ejecuta el microkernel invocando el código secuencial presentado anteriormente (línea 21) y prepara las variables y punteros de dirección para la siguiente iteración (línea 23). Recuérdese que, desde el interior del microkernel, el núcleo transfiere en streaming A_r desde L1 a sus registros vectoriales.
- Se incluyen puntos de sincronización en las líneas 17 y 25. El primero garantiza que los datos ya se hayan copiado en la MA L1, mientras que el segundo es para la sincronización general de los hilos.

Finalmente, volvemos al operador de convolución para notar que, en el caso de la transformación IM2COL, la matriz $\hat{A}$ contiene los filtros del operador y, para la inferencia, este tensor permanece constante. Lo mismo se aplica a $\hat{B}$ en el caso de la transformación IM2ROW. En el siguiente apartado describiremos cómo podemos aprovechar esto para preempaquetar el tensor de filtros y eliminar/acelerar algunas de las transferencias de datos para el algoritmo B3C2A0.

VI. Análisis de rendimiento

En este apartado evaluamos el rendimiento del operador de convolución basado en el enfoque de reducción, analizando las diferencias entre las variantes IM2COL y IM2ROW.

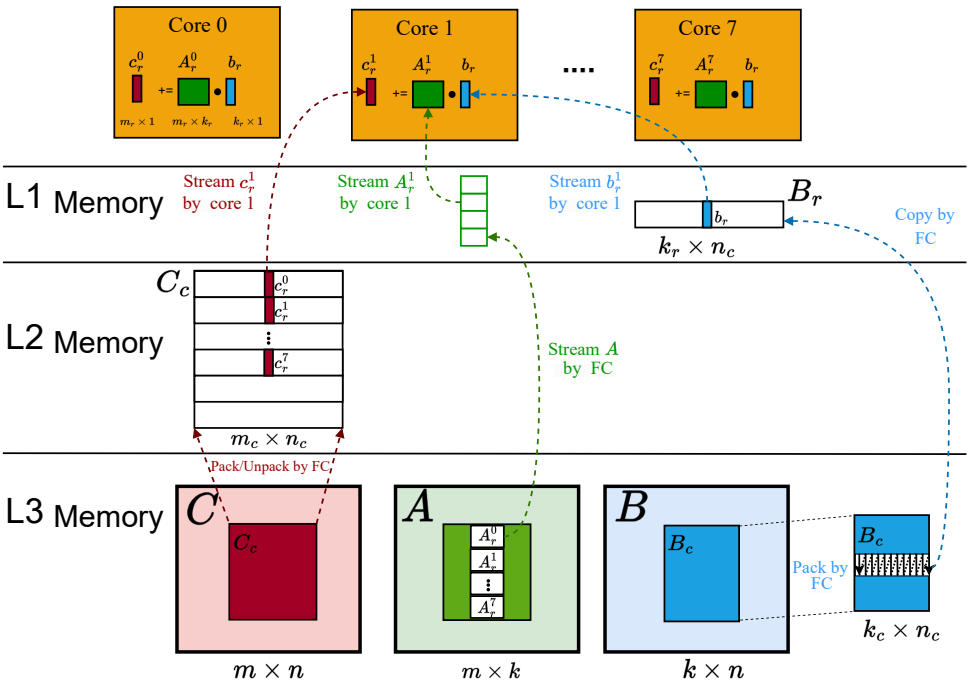


Fig. 8: Movimientos de datos en la versión paralela del algoritmo B3C2A0. Para simplificar, las transferencias de datos correspondientes a la transmisión de las columnas $c_r^0, c_r^1, \ldots, c_r^7$ desde los 8 micropaneles correspondientes de C_c a los registros del procesador se anotan con flechas solo para el Core 1. Lo mismo se aplica a la transmisión (replicación) de la columna b_r del micropanel B_r, y la transmisión de los 8 micro-mosaicos $A_r^0, A_r^1, \ldots, A_r^7$ desde A.

#Id. capa	Parámetros de la convolución						Dimensiones GEMM			Mejor microkernel
	c_o	w_o	h_o	h_f	w_f	c_i	m	n	k	
1	32	224	224	3	3	3	32	50176	27	4×24
2	32	112	112	3	3	32	32	12544	288	4×24
3	64	112	112	1	1	32	64	12544	32	4×20
4	64	56	56	3	3	64	64	3136	576	4×24
5	128	56	56	1	1	128	128	3136	128	4×24
6	128	56	56	3	3	128	128	3136	1152	4×24
7	128	56	56	1	1	128	128	3136	128	4×20
8	128	28	28	3	3	128	128	784	1152	4×20
9	256	28	28	1	1	128	256	784	128	4×20
10	256	28	28	3	3	256	256	784	2304	4×20
11	256	28	28	1	1	256	256	784	256	4×24
12	256	14	14	3	3	256	256	196	2304	4×24
13	512	14	14	1	1	256	512	196	256	4×24
14,16,18,20,22	512	14	14	3	3	512	512	196	4608	4×24
15,17,19,21,23	512	14	14	1	1	512	512	196	512	4×24
24	512	7	7	3	3	512	512	49	4608	4×24
25	1024	7	7	1	1	512	1024	49	512	4×24
26	1024	7	7	3	3	1024	1024	49	9216	4×24
27	1024	7	7	1	1	1024	1024	49	1024	4×24

Tabla I: Parámetros de las capas de convolución que surgen en MobileNet-v1 y dimensiones de la GEMM obtenida con la aplicación de la transformación IM2COL. Para IM2ROW, los valores de las columnas m y n se intercambian. La capa 28 se reduce a un producto de matriz-vector y, por lo tanto, se omite de los experimentos. Para todas las capas, el desplazamiento (*stride*) y los rellenos vertical/horizontal son iguales a 1.

A. Setup

Hemos evaluado el rendimiento de nuestra convolución paralela basada en GEMM en la plataforma GAP8 utilizando un modelo de aprendizaje profundo (DL) real y aritmética INT8. Específicamente, ejecutamos la fase de inferencia para las capas convolucionales que están presentes en la red neuronal MobileNet-v1, estableciendo el tamaño del lote de entrada b en 1 (es decir, un escenario de entrada única). Para este propósito, preprocesamos los ope-

```
1  gemm_loop_L5_gap8(int nc, signed char *A,
2                    int ldA,
3                              signed char *Br,
4                              signed char *Cc,
5                              signed char *Atmp) {
6  int ir;
7  unsigned int cl_core_id, // Core identifier
8               num_cores;  // Number of cores
9  pi_cl_ram_req_t req_a;
10
11 for (ir=0; ir<mc; ir+=mr) { // Loop L5
12   if (((ir/mr)%num_cores)!=cl_core_id)
       continue;
13
14   // Copy Ar from main memory to L1 in CE
15   for (i = 0; i < mr; i++) {
16     pi_cl_ram_read(ram, &A[i*ldA]),
17                    &A_local[i*kr],
18                    kr, &req_a);
19
20     pi_cl_ram_read_wait(&req_a);
21   }
22
23   // Micro-kernel (same as in sequential case)
24   gemm_ukernel_gap8(nc, A_local, kr, Br, Cc);
25
26   A=A+mr; Cr=(Cr+(mr*nc)); // Prepare for next
       iter.
27 }
28 pi_cl_team_barrier(0); // Thread
       synchronization
29 }
```

Fig. 9: Implementación simplificada de la versión paralela del bucle L5 para la variante B3C2A0 de GEMM.

radores de convolución utilizando la transformación IM2COL o IM2ROW, obteniendo operaciones de tipo GEMM de la forma $\hat{C} = \hat{A} \cdot \hat{B}$ que operan con matrices aumentadas de diferentes dimensiones; consulte el apartado II y la Tabla I.

Los resultados informados en este apartado corresponden a números promedio para un gran número de ejecuciones de cada experimento. Hemos implementado y probado microkernels de varias dimensiones $m_r \times k_r$. Para mayor brevedad, para cada operador de convolución solo informamos los resultados obtenidos con el microkernel de mejor rendimiento. Para las capas 1–2, 4–7 y 11–27 de MobileNet-v1, esto corresponde a un microkernel con $m_r \times k_r = 4 \times 24$. Para las capas 3 y 8–10, el mejor microkernel fue $m_r \times k_r = 4 \times 20$.

B. Análisis preliminar

Como punto de partida para nuestro análisis, la Figura 10 desglosa el tiempo empleado en la capa 10 de MobileNet-v1 en los diferentes componentes del algoritmo:

- Arithmetic (por núcleos de cómputo),
- Stream_Cc (desde L2 a registros por núcleos de cómputo),
- Stream_Br (desde L1 a registros por núcleos de cómputo),
- Stream_A (desde L3 a L1 por FC, y desde allí a registros por núcleos de cómputo),
- Copy_Br (desde L3 a L1 por núcleos de cómputo),
- Pack_Cc (desde L3 a L2 por FC),
- Unpack_Cc (desde L2 a L3 por FC), y
- Pack_Bc (desde L3 a L3 por FC);

ver el apartado V y Figura 8. La figura muestra dos gráficos, para las variantes de convolución basadas en IM2COL e IM2ROW, e informa del tiempo de ejecución (en segundos) usando 1, 4 y 8 núcleos de cálculo.

A partir de los gráficos en la Figura 10, es evidente que los dos primeros componentes, Arithmetic y Stream_Cc, se benefician significativamente de una ejecución paralela. Por el contrario, hay un comportamiento diferente para muchos otros componentes, con una disminución muy pequeña del tiempo de ejecución al utilizar 4 núcleos de cálculo y un beneficio insignificante para 8 núcleos de cálculo. La explicación en todos estos casos es común: Aquellos componentes que se ejecutan dentro del bucle que se paraleliza en nuestra solución (bucle L5), para los cuales además no hay participación de FC, realmente se ejecutan en paralelo y consecuentemente se aceleran en una ejecución multihilo. Este es el caso de Arithmetic, Stream_Cc, Stream_Br, and Copy_Br. En comparación, los componentes restantes requieren la participación de FC (básicamente para programar las transferencias DMA de/a L3) y, por lo tanto, son intrínsecamente secuenciales ya que solo hay un FC. Para la ejecución secuencial, el coste está dominado por Arithmetic seguido por Stream_Cc y por la variante IM2ROW variant, Stream_A. La falta de escalabilidad paralela del último componente para IM2ROW tiene un fuerte impacto en el coste de las ejecuciones paralelas para esa variante.

Se puede apreciar una diferencia significativa entre los costes de las variantes IM2COL y IM2ROW para Stream_A. La razón principal es que este coste es proporcional a la dimensión m del GEMM asociado con esta capa: 256 para IM2COL y 784 para IM2ROW (ver Tabla I).

Además, para IM2COL la matriz de filtro corresponde al operando de la matriz GEMM $\hat{A}$. Por lo tanto, hemos acelerado este tipo de transferencia, desde la memoria principal hasta el L1 MA, mediante 1) la pre-empaquetación del operando, de modo que los elementos $m_r \times k_r$ de cada microtile se encuentren en posiciones contiguas en la memoria principal; y 2) programando el DMA/FC para copiar los elementos $(m_r \cdot k_r)$ del microtile con una sola llamada. Esto permite reemplazar el bucle en las líneas 14–18 de la Figura 9 con una sola llamada a `pi_cl_ram_read+pi_cl_ram_read_wait`.

Finalmente, comparando las distribuciones de costes entre las variantes IM2COL y IM2ROW, observamos que no hay coste asociado con Pack_Bc para esta última. La razón es que, para IM2ROW, la matriz GEMM operando $\hat{B}$ corresponde a los filtros de convolución, que permanecen constantes durante la etapa de inferencia. En consecuencia, podemos pre-empaquetar esta matriz (y reutilizarla para cualquier número de inferencias posteriores) de modo que la reorganización de la matriz se vuelve innecesaria. En contraste, para IM2ROW el operando de la matriz $\hat{A}$ corresponde a la matriz aumentada que resulta de aplicar la transformación al tensor de activación de entrada. Como estos datos varían de una muestra

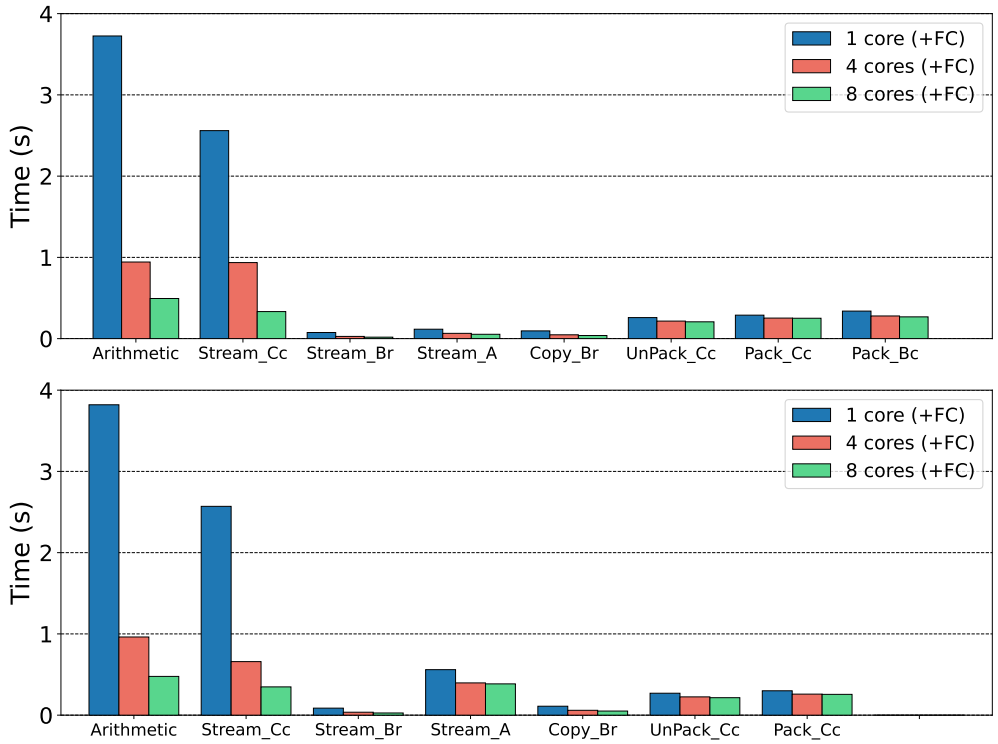


Fig. 10: Distribution of costs for layer #10 of MobileNet-v1 using IM2COL+GEMM (top) and IM2ROW+GEMM (bottom).

a la siguiente, no podemos preempaquetarla. Por lo tanto, para IM2ROW no podemos beneficiarnos de las transferencias más rápidas de los microtiles entre la memoria principal y L1 descritas en el párrafo anterior.

C. Global comparison

La Figura 11 muestra las tasas de rendimiento para todas las capas de MobileNet-v1, obtenidas con la versión "secuencial"[3] de las dos variantes de convolución (IM2COL y IM2ROW) así como el incremento de velocidad correspondiente observado con la contraparte paralela, utilizando 4 y 8 núcleos de cálculo.

En general, la versión secuencial que usa IM2COL supera ligeramente a la alternativa basada en IM2ROW para las capas iniciales (1–11), pero es mínimamente inferior para las capas finales. La razón de este comportamiento similar es que, en el caso secuencial, no hay diferencias para Arithmetic, ya que ambas variantes obviamente realizan el mismo número de operaciones aritméticas y este componente no incluye ningún coste de transferencia de datos; además, 2) las diferencias entre el coste de Stream_Cc para las dos variantes son insignificantes, ya que esto depende de m, n, y estos dos valores simplemente se intercambian entre las dos variantes. En el caso paralelo, el factor que puede marcar una diferencia entre IM2COL y IM2ROW es la falta de escalabilidad de Stream_A, que tiene una contribución significativa al tiempo de ejecución para este último. Sin embargo, esto se compensa con el componente Pack_Bc para IM2COL, que contribuye con un coste que la contraparte IM2ROW no tiene que pagar.

En relación con el algoritmo paralelo, en 4 núcleos de cálculo observamos una aceleración máxima de 3,19 para IM2COL y ligeramente mayor, 3,24, para IM2ROW. Con 8 núcleos de cálculo, la aceleración máxima es 5.00 para IM2COL y nuevamente marginalmente superior, 5,15, para IM2ROW. La aceleración promedio para IM2COL es 2,96 en 4 núcleos de cálculo y 4,39 en 8 núcleos de cálculo; para IM2ROW, es 2,71 en 4 núcleos de cálculo y 4,09 en 8 núcleos de cálculo.

D. IM2COL *vs* IM2ROW

La discusión anterior expone las pequeñas diferencias de rendimiento entre IM2COL y IM2ROW. En la práctica, el primer tipo de transformación está asociado con el llamado formato NHWC de los tensores de activación de entrada/salida, donde las cuatro letras (N,H,W,C) especifican respectivamente el orden en memoria de las dimensiones $(b, h_i/h_o, w_i/w_o, c_i/c_o)$ del operador de convolución. En comparación, la transformación IM2ROW está vinculada con el formato NCHW. Esto es relevante porque, para IM2ROW, es posible concatenar dos (o más) capas convolucionales consecutivas (con un número de capas elementales entre ellas) de modo que el tensor de activación de salida de una convolución se pase directamente como activación de entrada a la siguiente.

Para IM2COL, en contraste, la concatenación de capas convolucionales consecutivas requiere reorga-

[3]Nótese que, aunque el algoritmo secuencial utiliza un solo núcleo de cálculo para la aritmética, todavía requiere la participación del FC para orquestar las transferencias de datos. Lo mismo se aplica al algoritmo paralelo, en el cual el FC se encarga de mover los datos entre los MAs pero toda la aritmética es realizada por los núcleos de cálculo.

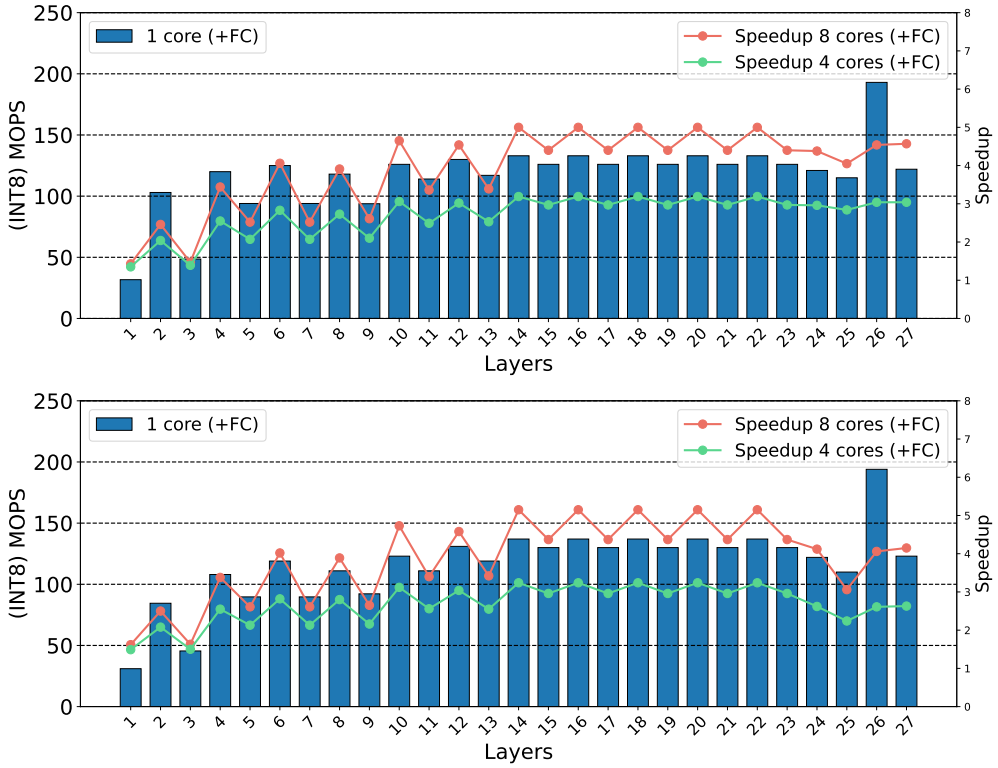


Fig. 11: Rendimiento alcanzado para las capas convolucionales en MobileNet-v1 usando IM2COL+GEMM (arriba) y IM2ROW+GEMM (abajo). Los resultados con un solo núcleo muestran la tasa aritmética (en millones de operaciones INT8 por segundo, o MOPS) observada con el algoritmo "secuencial", ejecutado utilizando un solo núcleo de cálculo y el FC.

nizar el tensor de activación de salida en memoria, antes de pasar sus datos a la siguiente capa. La clave aquí es que el coste de esta reorganización de datos no es, en general, despreciable en una plataforma como la GAP8.

Una estrategia potencial para acelerar la ejecución de IM2ROW es emplear una variante algorítmica basada en A3C2B0 para GEMM en lugar de B3C2A0. Para IM2ROW, la matriz de filtros corresponde al operando de matriz GEMM $\hat{B}$ y, para A3C2C0, este operando puede ser pre-empaquetado en microtile en la memoria principal. Por lo tanto, esta solución puede beneficiarse de las mismas transferencias rápidas reportadas anteriormente para $\hat{A}$ en la variante IM2COL combinada con el algoritmo B3C2A0.

VII. Comentarios finales

Hemos propuesto una implementación eficiente del operador de convolución, para el procesador multicore heterogéneo GAP8 PULP, que aprovecha el controlador FC para las transferencias de datos y los núcleos en el motor de cálculo para las operaciones aritméticas. GAP8 presenta una jerarquía de memoria de cuatro niveles, con scratchpads en lugar de caches convencionales, además de un controlador y 8 núcleos de cálculo. Para adaptarse a esta arquitectura, nuestra solución transforma la convolución en un GEMM, a través del enfoque de reducción; aplica teselado para dividir los operandos de matriz GEMM; y orquesta las transferencias de datos a través de la jerarquía de memoria como parte de las operaciones de empaquetado en GEMM. Además, el enfoque propuesto formula la operación GEMM como un algoritmo donde un pequeño bloque de A (el micromosaico) reside en los registros vectoriales de cada núcleo de cálculo (para efectuar el cálculo más interno en términos de un producto escalar); y explota el paralelismo de uno de los bucles más internos de GEMM para distribuir la carga de trabajo entre los ocho núcleos de cálculo.

Nuestros experimentos en la plataforma, utilizando el modelo MobileNet-v1 y un escenario de entrada única, muestran pequeñas diferencias en el tiempo de ejecución y la escalabilidad paralela entre las variantes IM2COL y IM2ROW, aunque esperamos que esta última sea más eficiente al considerar la concatenación de capas que aparecen en una DNN, especialmente si se integra en un algoritmo A3C2B0 para GEMM.

Como parte del trabajo futuro, planeamos explorar las posibilidades de solapamiento de las transferencias con el cálculo a través del doble buffering. Esperamos que esto reduzca el impacto de los tiempos de inactividad debido a la comunicación en el rendimiento global. Sin embargo, hay un delicado equilibrio a inspeccionar aquí ya que también reduce la reutilización de los datos almacenados en los buffers, un factor que es relevante debido a la pequeña capacidad de los scratchpads.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto PID2020-113656RB-C22 (MCIN/AEI/10.13039/501100011033), project TED2021-129334B-I00 (MCIN/AEI/10.13039/501100011033), and by the "European Union NextGeneration EU/PRTR". C. Ramírez is a "Santiago Grisolía" fellow supported by *Generalitat Valenciana*. Adrián Castelló is a FJC2019-039222-I fellow supported by MCIN/AEI/10.13039/501100011033. H. Martínez is a POSTDOC_21_00025 postdoctoral fellow supported by *Consejería de Transformación Económica, Industria, Conocimiento y Universidades de la Junta de Andalucía*.

Referencias

[1] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at Facebook: A datacenter infrastructure perspective," in *IEEE Int. Symp. HPC Architecture*, 2018, pp. 620–629.

[2] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. Khudia, J. Law, P. Malani, A. Malevich, S. Nadathur, J. Pino, M. Schatz, A. Sidorov, V. Sivakumar, A. Tulloch, X. Wang, Y. Wu, H. Yuen, U. Diril, D. Dzhulgakov, K. Hazelwood, B. Jia, Y. Jia, L. Qiao, V. Rao, N. Rotem, S. Yoo, and M. Smelyanskiy, "Deep learning inference in Facebook data centers: Characterization, performance optimizations and hardware implications," 2018, arXiv 1811.09886.

[3] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at Facebook: Understanding inference at the edge," in *IEEE Int. Symp. HPC Architecture*, 2019, pp. 331–344.

[4] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "Pulp-nn: A computing library for quantized neural network inference at the edge on risc-v based parallel ultra low power clusters," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 33–36.

[5] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *10th Int. Workshop Frontiers in Handwriting Recogn.*, 2006, Université de Rennes, France.

[6] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[7] K. Goto and R. A. van de Geijn, "Anatomy of a high-performance matrix multiplication," *ACM Trans. Math. Softw.*, vol. 34, no. 3, pp. 12:1–12:25, 2008.

[8] T. M. Smith and R. A. van de Geijn, "The MOMMS family of matrix multiplication algorithms," *CoRR*, vol. abs/1904.05717, 2019.

[9] J. A. Gunnels, F. G. Gustavson, G. M. Henry, and R. A. van de Geijn, "A family of high-performance matrix multiplication algorithms," in *Proc. 7th Int. Conf. on Applied Parallel Computing*, 2004, p. 256–265.

[10] A. Castelló, F. D. Igual, and E. S. Quintana-Ortí, "Anatomy of the BLIS family of algorithms for matrix multiplication," in *2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2022, pp. 92–99.

[11] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr. Wolf: An energy-precision scalable parallel ultra low power SoC for IoT edge processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019.

[12] C. Ramírez, A. Castelló, and E. S. Quintana-Ortí, "A BLIS-like matrix multiplication for machine learning in the RISC-V ISA-based GAP8 processor," *J. Supercomput.*, vol. 78, no. 16, p. 18051–18060, nov 2022. [Online]. Available: https://doi.org/10.1007/s11227-022-04581-6

Reducción del coste computacional de Redes Convolucionales mediante la detección de la redundancia en los datos de entrada

Laura Medina[1] y Jose Flich [1]

Resumen— **Las Redes Neuronales Convolucionales (CNN) se encuentran presentes en muchas aplicaciones cotidianas. Éstas se componen de cientos de millones de operaciones de producto escalar, lo que ha llevado a la aparición de nuevos algoritmos para reducir su elevada complejidad computacional. Este artículo introduce la repetición de entrada (*input repetition*): una nueva solución para reducir el número de multiplicaciones de los modelos CNN cuantificados sin degradar su precisión. La repetición de entrada se produce cuando se encuentran varios elementos idénticos en un grupo de entrada, permitiendo reducir el número de multiplicadores necesarias para realizar la operación de punto escalar. En este trabajo se analizan varias estrategias para hallar la repetición en la entrada de las convoluciones utilizando CNN cuantificadas: YoloV3, MobileNET y ResNet50. Los resultados muestran que explotando la repetición de la entrada se puede reducir las multiplicaciones en un factor de 2,4x sin ningún impacto en la precisión.**

Palabras clave— **CNN, Input Repetition, Convoluciones**

I. Introducción

LAS Redes Neuronales Convolucionales (CNN) se están convirtiendo en esenciales en muchas aplicaciones del mundo real. Más concretamente, las CNN son vitales en el campo de la detección de objetos, la segmentación de imágenes y el reconocimiento facial. Sin embargo, las aplicaciones deben hacer frente a su complejidad computacional para alcanzar el notable rendimiento de las CNN, ya que son computacionalmente costosas debido al gran número de parámetros que contienen y al tamaño de las imágenes que procesan. Además, los modelos CNN comprenden operaciones de convolución en las que se aplican pequeños filtros sobre toda la imagen de entrada. Este componente crítico de las CNN requiere un gran número de multiplicaciones y sumas.

Con el fin de reducir la complejidad computacional de los modelos CNN, se aplican varias técnicas para disminuir el tamaño de los modelos CNN y, por ejemplo, los cálculos necesarios para realizar las operaciones. La cuantificación puede aplicarse para decrementar el número de bits de los parámetros de la red neuronal (NN), reduciendo significativamente el tamaño del modelo y dando lugar a menores tiempos de inferencia y un menor consumo energético. Otro enfoque comúnmente utilizado para simplificar el tamaño de los modelos de NN es la poda (conocido como *prunning*). La poda es una técnica que elimina neuronas, conexiones o filtros con el fin de reducir el número de parámetros de una red, lo que puede conducir a una inferencia más rápida y eficiente [1]. Aunque las técnicas de poda y cuantificación reducen el tamaño de los modelos y los requisitos informáticos, estos modelos siguen siendo demasiado grandes para determinados escenarios. En sistemas embebidos con restricciones de tiempo real es de suma importancia reducir los requisitos computacionales de dichos modelos y vincular el tiempo de inferencia a un umbral determinado.

Los requisitos computacionales de los modelos CNN cuantificados hacen que los dispositivos FPGAs (*Field-Programmable Gate Arrays*) sean idóneos para estas aplicaciones, ya que ofrecen un bajo consumo energético, alta eficiencia y alta flexibilidad. Sin embargo, estos dispositivos disponen de recursos limitados. Dado que las operaciones convolucionales y los algoritmos NN comprenden millones de multiplicaciones y sumas, se necesitan más recursos a medida que se desea más paralelismo.

De forma similar a las técnicas de poda y cuantificación, proponemos abordar una fuente de redundancia alternativa. Nuestro objetivo es reducir el número de multiplicaciones necesarias para realizar la operación de convolución identificando la repetición entre los datos de entrada de las convoluciones y, a continuación, realizar sólo las multiplicaciones de los valores diferentes. Al reducir las multiplicaciones, pretendemos reducir los recursos necesarios o mantener los recursos pero acelerar el proceso.

Presentamos un análisis que mide la repetición de estos valores utilizando modelos enteros de 8 bits (INT8). Para este estudio, analizamos la reorganización de diferentes píxeles utilizando los modelos cuantificados más avanzados de ONNX Model Zoo[2]. Utilizamos los modelos Yolo-V3[3], ResNet-50[4] y MobileNet[5]. En este artículo se demuestra una repetición significativa entre las entradas de las capas convolucionales, lo que permite reducir un 58,3 % las multiplicaciones. También se verifica la posibilidad de desarrollar aceleradores específicos cuyo fin es explotar esta repetición, mediante la identificación de las entradas con el mismo valor y reduciendo el número de multiplicaciones y, en consecuencia, reducir también el número de recursos necesarios en las FPGAs y acelerando el proceso de inferencia de los modelos NN. En resumen, en este trabajo se pueden observar las siguientes aportaciones:

- Se analiza la repetición de los valores de las entradas de las capas de convolución en varios mo-

[1]Universitat Politècnica de València, Spain, e-mail: {laumecha,jflich}@disca.upv.es

delos cuantificados INT8. Para este estudio, se utilizan diferentes estrategias de agrupación para estudiar cómo afecta a la repetición.
- Se demuestra que reutilizando la redundancia de entrada, se consigue mejorar el rendimiento de un proceso de inferencia en factores de hasta 2,4 en los modelos utilizados.

El resto del documento se organiza como sigue. La sección II describe la bibliografía relacionada con nuestro tema. En la sección III se introduce el concepto de repetición en la entrada y se describen las diferentes estrategias de agrupación. A continuación, la Sección IV muestra los resultados obtenidos. Por último, la sección V muestra las conclusiones del trabajo.

II. Trabajos relacionados

Sparsity. El concepto de *sparsity*[6] se aplica a las Redes Neuronales para reducir el número de operaciones eliminando los cálculos de pesos o activaciones equivalentes a cero, encontrándose normalmente en modelos hiperparametrizados. Sin embargo, este enfoque no es de utilidad para los modelos cuantificados, ya que los valores de los datos se encuentran escalados para ajustarse a un rango de valores y, por consiguiente, las apariciones de cero no suelen ser explotables.

Repetición para poda. La repetición también puede ser explotada con otro fin distinto al presentado en este artículo. Estudios anteriores han propuesto técnicas para reducir el número de cálculos realizados mediante la eliminación de la repetición intrínseca en los modelos. En [7] se presenta un método de poda que elimina los filtros convolucionales tomando la repetición entre filtros, permitiendo reducir la complexidad aritmética.

Repetición para el procesamiento de fotogramas. Otros estudios previos proponen técnicas para explotar la repetición de múltiples imágenes. En [8], [9], proponen reutilizar los resultados intermedios de las activaciones si se encuentran imágenes idénticas.

Reutilización de los pesos. Más afín a este trabajo, se han propuesto varios aceleradores de repetición en los pesos donde, a diferencia de nuestro trabajo, se comparan los valores de los pesos de las convoluciones para eliminar el cómputo de aquellos valores redundantes. En [10], [11] se consigue una reducción de multiplicadores del 33 % en [10] y se presenta un ahorro del 26 % en [11].

Este trabajo se enfoca en los patrones de repetición de los datos de **entrada** en capas de convolución, que, hasta donde sabemos, no están cubiertos por ningún trabajo anterior. Este trabajo también es complementario a todos los anteriores.

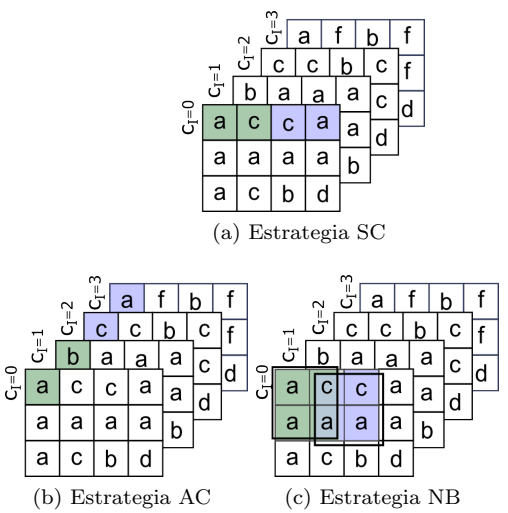
III. Repetición

La operación convolucional consiste en multiplicar cada valor de la matriz de datos de entrada (formada por $CI \times H \times W$ elementos) por filtros (formados por $CO \times KH \times KW$ pesos). La operación convolucional (sin el *bias*) viene dada por:

$$O[c_o, h, w] = \sum_{c_i=0}^{CI-1} \sum_{k_h=0}^{KH-1} \sum_{k_w=0}^{KW-1} I[c_i, h + k_h, w + k_w] \times \times W[c_o, c_i, k_h, k_w] \quad (1)$$

donde O, I y W son salidas, entradas y filtros, respectivamente. Numerosos modelos de última generación cuantifican estas dos matrices en forma INT8. Por lo tanto, los valores de entrada y de filtro sólo pueden adoptar $2^8 = 256$ valores posibles. Dado que $CI \times H \times W > 256$, se presentará repetición de valores en ambas matrices.

La hipótesis inicial es que existe un factor de repetición más significativo dentro de la matriz de entrada que dentro de la matriz de filtros. De hecho, los filtros están entrenados y deben especializarse para detectar características específicas de la imagen. Por lo tanto, tienden a tener valores diferentes entre ellos. Asimismo, los píxeles cercanos de una imagen tienen una gran probabilidad de tener valores iguales o similares si representan el mismo objeto o patrón como, por ejemplo, un cielo azul. Además, el vector de entrada de una convolución representa características detectadas por la convolución anterior. Por lo tanto, los datos de entrada consecutivos o cercanos pueden tener valores iguales o similares. Además, las entradas situadas en el mismo lugar en diferentes canales de entrada pueden tener el mismo valor, dado que derivan de las mismas entradas de la imagen original. Esta hipótesis se confirmará en la sección de evaluación, donde los resultados mostrarán que la repetición de entradas es mucho más frecuente que la repetición de pesos.



(a) Estrategia SC

(b) Estrategia AC (c) Estrategia NB

Fig. 1: Ejemplos de estrategias de agrupación entradas suponiendo un tamaño de grupo $G = 2$. En verde se ve el primer grupo, en azul el siguiente. Se utiliza un tensor de entrada de $4 \times 4 \times 3$.

Se define el factor de repetición (FR) de un grupo como el número de repeticiones dentro del grupo. Por ejemplo, en un grupo de 4 elementos con valores $\{a, b, b, d\}$ el factor de repetición se fijará en 1 y

para un grupo con los valores $\{a, a, a, a\}$ el factor de repetición se fijará en 3. Obsérvese que FR oscilará entre 0 y $G-1$, donde G es el tamaño del grupo.

Se presentan tres enfoques para el análisis y la optimización de la repetición de entrada en modelos cuantificados INT8. Todos estos enfoques agruparán conjuntos de datos de entrada siguiendo una estrategia de agrupación diferente. La figura 1 muestra los tres enfoques para la repetición de entrada: agrupación del mismo canal, del inglés *Same Channel* (SC) (1a), agrupación entre píxeles de diferentes canales (AC), del inglés *Across Channels* (1b) y agrupación entre píxeles cercanos, del inglés Nearby (NB) (1c). Se analizarán distintos tamaños de grupo. Obsérvese que no hay solapamiento de grupos en las estrategias SC y AC. Sin embargo, en la estrategia NB los grupos sí se solapan.

A. Repetición Same Channel

En la repetición *Same Channel* (SC), todos los datos de cada canal de entrada se leen secuencialmente en grupo. Los datos de entrada leídas de un canal de entrada deben multiplicarse por todos los pesos de los filtros asociados al canal de entrada. Por ejemplo, para una entrada $1 \times H \times W$ (un canal de tamaño $H \times W$), O canales de salida, y filtros 3×3, un dato de entrada p_x se multiplicará por todos los pesos de todos los filtros (suponiendo un relleno típico). Sin embargo, ninguna de esas multiplicaciones se sumará entre ellas, ya que cada multiplicación es la contribución a diferentes datos de salida y canales de salida. No obstante, si identificamos entradas (p_x y p_y) dentro del mismo grupo con el mismo valor ($p_x == p_y$), la multiplicación de p_x por w_a puede reutilizarse para p_y por w_a.

Obsérvese que las sumas realizadas en la operación de producto escalar pueden distribuirse en el tiempo mientras se realizan multiplicaciones de otras entradas. De hecho, se pueden refactorizar múltiples operaciones de producto escalar y separar las fases de multiplicación y suma. Suponiendo un dato de entrada $p_{x,y}$ con coordenadas x e y, cuando se multiplica por un peso $w_{a,b}$ de un filtro, la salida a la que contribuye la entrada es $o_{x-a,y-b}$ (suponiendo que el stride se establece en uno y sin considerar el *padding*). Por lo tanto, la operación de producto escalar puede ser refactorizada a:

$$O[c_o, x-a, y-b] += I[c_i, x, y] \times W[c_o, c_i, a, b] \quad (2)$$

La figura 2a muestra el enfoque estándar para implementar una operación de producto escalar para una trama de entrada de 2×2 y un filtro de convolución de 2×2. Se realizan cuatro productos en paralelo utilizando los cuatro datos de entrada y los cuatro pesos, y se realiza una operación de reducción utilizando tres sumadores. Obsérvese que, en principio, no podemos aprovechar directamente las oportunidades de repetición en la entrada, ya que los cuatro datos de entrada se multiplican por pesos diferentes.

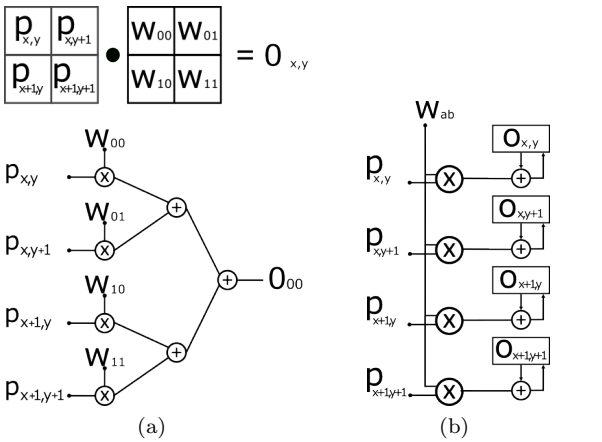


Fig. 2: Operación de producto escalar estándar (a) y operación de producto escalar distribuido (b) en una operación de convolución para una trama de entrada de 2×2 y un filtro de 2×2.

En cambio, en la Figura 2b, podemos ver cómo se puede utilizar un enfoque distribuido. En este caso, multiplicamos cuatro datos de entrada por el mismo peso. Se utilizan cuatro multiplicadores, y cuatro sumadores (uno añadido al caso anterior). Nótese que estamos produciendo cuatro datos de salida ya que ninguna de las multiplicaciones contribuye al mismo dato de salida. Por lo tanto, se necesitan cuatro registros para acumular las salidas de los productos. De hecho, la operación de convolución puede descomponerse en dos términos separados: la fase de multiplicación y la fase de reducción.

$$P[c_0, c_i, h, w, k_h, k_w] = I[c_i, h+k_h, w+k_w] \times \\ \times W[c_o, c_i, k_h, k_w] \quad (3)$$

$$O[c_o, h, w] = \sum_{c_i=0}^{CI-1} \sum_{k_h=0}^{KH-1} \sum_{k_w=0}^{HW-1} P[c_o, c_i, h, w, kh, kw] \quad (4)$$

Un acelerador o algoritmo orientado a detectar la repetición de los píxeles del mismo canal puede beneficiarse del enfoque distribuido. En este caso, se puede crear la lógica necesaria para leer un grupo de datos de entrada consecutivos y se multiplique por el mismo peso. Suponiendo que el tamaño del grupo es igual a 4 ($G = 4$), si se implementan dos operadores de multiplicación se puede utilizar un ciclo para realizar las cuatro multiplicaciones y sumas si todos los elementos tienen el mismo valor ($FR = 3$) o incluso si tienen dos valores distintos ($FR = 2$). Así, ganamos potencialmente dos ciclos al reducir el número de multiplicadores. Nótese que podemos reducir el número de multiplicadores manteniendo el rendimiento (tiempo de inferencia), o podemos mantener el número de multiplicadores y mejorar el rendimiento (reducir el tiempo de inferencia).

La figura 3 representa un ejemplo simplificado para SC en una operación de convolución sobre una matriz de entrada de $1 \times 2 \times 4$ con 1×1 de *padding* y filtros de 2×2. En este ejemplo, cada dato

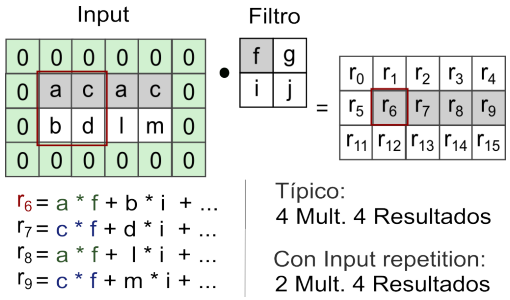


Fig. 3: Ejemplo SC. $G = 4$. El color verde representa el *padding*. El recuadro rojo representa los datos de entrada necesarias para obtener el dato de salida r6.

de entrada (excepto los datos que conforman el *padding*) se multiplicará por cade peso que conforma la matriz de filtros. Como podemos ver, explotamos la repetición de entrada agrupando los cuatro primeros elementos, ya que las multiplicaciones $a \times f$ y $c \times f$ pueden reutilizarse, necesitando dos multiplicaciones en lugar de cuatro.

B. Repetición Across Channels

En la repetición *Across Channels* (AC), los elementos se agrupan por el mismo índice H y W de diferentes canales consecutivos. En este caso, el tamaño del grupo corresponde al número de canales agrupados. En AC, cada dato de entrada debe multiplicarse por un peso diferente. Esto significa que, a diferencia de la estrategia SC, las multiplicaciones deben sumarse entre sí. Por tanto, para utilizar la repetición de AC, podemos utilizar la propiedad distributiva donde: $A \times B + A \times C = A \times (B + C)$. En este caso, si encontramos repetición entre canales, podemos distribuir la operación para reducir el número de multiplicaciones.

La figura 4 muestra un ejemplo de repetición AC de una operación de convolución para una entrada de $2 \times 2 \times 2$ y un canal de salida, por lo que tenemos dos filtros (uno para cada canal de entrada). En este caso, los resultados intermedios (representados como $M_{x,y}$) se sumarán entre ellos, dando como resultado O_a. Como describe la figura, podemos ahorrarnos una multiplicación aprovechando la repetición AC para calcular O_0.

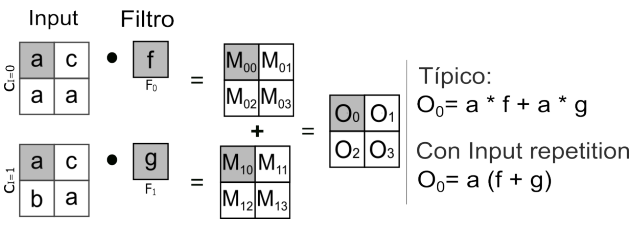


Fig. 4: Ejemplo AC con $G = 2$.

C. Repetición Nearby

La repetición *Nearby* (NB) explota la repetición de los datos de entrada próximos del mismo canal. En este caso, se seleccionan nueve entradas, formando un marco de 3×3 (Figura 1c), cuyo tamaño se

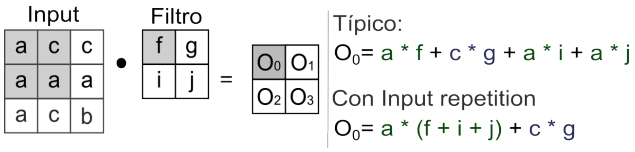


Fig. 5: Ejemplo NB con $G = 2$.

ajusta a los tamaños de filtro típicos (3×3) utilizados en las capas de convolución. Tras realizar la convolución sobre una región de la imagen de entada (típicamente conocido como *frame*) con el filtro correspondiente, se selecciona una nueva región, desplazando una columna a la izquierda o una fila hacia abajo. Las entradas se multiplican y se suman con los filtros, utilizando la operación estándar de producto escalar. Sin embargo, en este caso, con NB se identifica el patrón de repetición y se utilizan menos multiplicaciones.

La figura 5 muestra un ejemplo de repetición NB en su lugar para una entrada de $1 \times 3 \times 3$ y un filtro de 2×2. Se realizan dos multiplicaciones en lugar de cuatro.

D. Ahorro y coste de las estrategias SC, AC, y NB

En una operación de convolución (véase la ecuación 3), se realiza una multiplicación por cada combinación del canal de entrada (CI), el canal de salida (CO), el dato de entada de entrada (H y W) y el peso del filtro (KH y KW), donde el número total de multiplicaciones viene dado por:

$$Muls = CI \times CO \times H \times W \times KH \times KW \qquad (5)$$

Al explotar la repetición de entrada, se reduce el número de multiplicaciones realizadas. Definimos el factor de aceleración o SpeedUp (S) dentro de un grupo de tamaño G y un valor dado de FR como:

$$S = \frac{G}{G - FR} \qquad (6)$$

Como resultado, en la explotación de repetición de entrada, el número de multiplicaciones a realizar es:

$$Muls = \frac{CI \times CO \times H \times W \times KH \times KW}{S_{avg}} \qquad (7)$$

donde S_{avg} es la aceleración media alcanzada para todos los grupos de datos de entrada. Obsérvese que las diferentes estrategias pueden lograr diferentes rendimientos, ya que encontrarán una FR diferente. Asimismo, el tamaño del grupo (G) determinará el rendimiento.

La repetición de AC puede tener costes heredados basados en el orden en que se realizan las multiplicaciones. Sin embargo, esto depende de cómo se diseñe el acelerador o algoritmo para las operaciones de convolución. Las tres estrategias de agrupación pueden seleccionarse en función del enfoque objetivo. Por ejemplo, si el paralelismo se explota a nivel de

canal de entrada (varios canales leídos en paralelo), puede utilizarse la estrategia AC. Por el contrario, si el paralelismo se explota a nivel dato de entrada (múltiples entradas del mismo canal se leen en paralelo), entonces se puede utilizar la estrategia SC. Por último, si se realiza la operación producto escalar, entonces se puede utilizar NB.

No obstante, la estrategia AC no realiza lecturas múltiples de los datos de entrada, por lo que no tiene que hacer frente a costes energéticos adicionales por las lecturas de memoria. La sobrecarga evidente se produce a la salida de la operación, donde pueden calcularse simultáneamente varios datos de salida. Esto requiere registros adicionales para acumular los resultados. Sin embargo, el número de registros depende exclusivamente del tamaño del grupo (G). El coste del multiplexor, demultiplexor y del módulo responsable de detectar la repetición (al que llamaremos módulo RP) depende también del tamaño del grupo. Los tamaños de grupo pequeños, de 4 u 8, no aumentan significativamente el área ni el coste. Como veremos, los tamaños de grupo pequeños ofrecen un buen rendimiento.

Por otro lado, en las estrategias AC y NB, el módulo RP realiza la misma comparación de los datos de entrada que en la estrategia SC. La sobrecarga puede incurrirse en el precálculo de las sumas de pesos. Sin embargo, estas sumas pueden realizarse *offline* en tiempo de carga. Como ocurre en SC, los costes de las estrategias AC y NB están directamente relacionados con el tamaño del grupo (G).

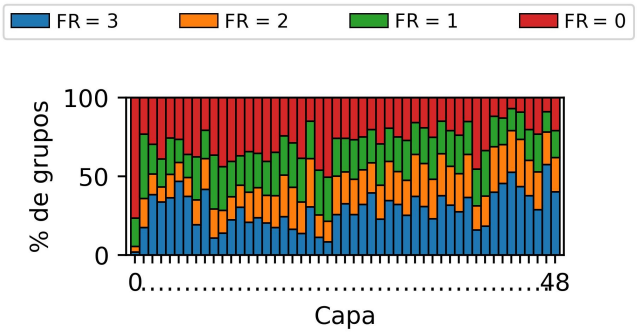
IV. Experimentos

Para la evaluación se han utilizado CNN del estado del arte, altamente usados en aplicaciones del mundo real. En primer lugar, se analizan los modelos ResNet-50 y MobileNet-V2, ambos orientados a realizar clasificación de imágenes y, además, el modelo Yolo-V3, el cual es utilizado para segmentación de detección de objetos en tiempo real. Para todos los modelos, se ha elegido la versión cuantificada en INT8, disponibles en ONNX Model Zoo, lo cual añade un error aceptable en comparación con las versiones de coma flotante. Para los conjuntos de datos, utilizamos Imagenet[12] para la inferencia de ResNet-50 y MobileNet-V2 y COCO[13] para la inferencia de Yolo-V3.

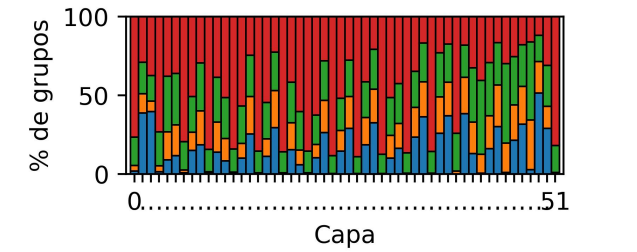
A. Análilis de la repetición de entrada

En primer lugar, se analiza el FR de los modelos utilizando los estudios descritos en la sección anterior. Para los resultados siguientes, se utilizan los modelos extrayendo los tensores de entrada de cada convolución. A continuación, se realiza el estudio con 50 imágenes diferentes y extraemos el FR medio para cada capa.

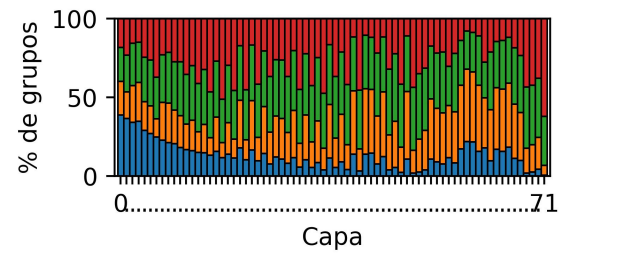
La figura 6 se observa el análisis de los tres modelos con la estrategia SC con un tamaño de grupo igual a 4 ($G = 4$). Los colores representan el porcentaje de FR calculado para procesar la convolución de entrada de cada capa del modelo. Para los modelos ResNet-50 y MobileNet se han utilizado el mismo conjunto de imágenes de entrada. En azul, se muestra el porcentaje de grupos con FR igual a tres, lo que implica el porcentaje de veces que los cuatro elementos dentro del grupo presentan el mismo valor. En amarillo, se observa el porcentaje de grupos con FR igual en 2. Del mismo modo, los colores verde y rojo muestran los porcentajes para FR fijada en uno y cero, respectivamente. Un FR igual a cero implica que todos los elementos del grupo difieren en valor.



(a) ResNet-50



(b) MobileNet



(c) YoloV3

Fig. 6: FR en SC en los diferentes modelos CNN. $G = 4$.

Lo primero que hay que señalar es que la distribución de FR no es constante entre capas y modelos. La repetición entre los datos de entrada de las capas de convolución viene más determinada por la topología de la red que por la imagen en sí, ya que la Figura 6a y la Figura 6b representan la inferencia de las mismas imágenes pero utilizando dos topologías diferentes. El modelo ResNet-50 presenta un porcentaje de repetición significativamente mayor. La primera capa en los modelos ResNet-50 y MobileNet muestra un bajo porcentaje de repetición. El patrón de repetición es más alto en capas sucesivas en esos modelos, aunque es más notable en el modelo ResNet-50. Por el contrario, en Yolo-V3, el patrón de repetición es más frecuente en las primeras capas del modelo. No

obstante, el porcentaje de FR establecido en tres y dos es superior al 50 % en ResNet-50 y Yolo-V3, ligeramente inferior en MobileNet.

La tabla I amplía el análisis a diferentes tamaños de grupo y estrategias de agrupación. Esta tabla muestra el FR medio de cada modelo para cada estrategia de agrupación y tamaño de grupo. Los resultados muestran una considerable repetición en los datos de entrada que puede ser explotada en la creación de futuros aceleradores. La tabla muestra que el factor de repetición mejora con un tamaño de grupo mayor. Sin embargo, aumentar el tamaño del grupo significaría aumentar la complejidad del posible acelerador.

B. SpeedUp

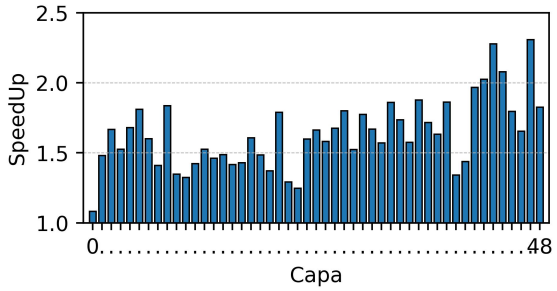
En la repetición de entrada, la reducción teórica máxima que puede lograrse es igual al tamaño del grupo. Los resultados de FR mostrados anteriormente confirman una oportunidad significativa de repetición de entrada en todos los modelos estudiados, lo que significa que podríamos beneficiarnos de ello implementando un acelerador o algoritmo especializado. En la Figura 7, podemos ver la aceleración (*SpeedUp*) de la multiplicación de cada capa resultante al explotar la repetición SC de un tamaño de grupo de 4 en cada modelo.

Como se observa, en todas las capas para todos los modelos obtenemos resultados de aceleración superiores a uno. Destacan los resultados obtenidos tanto en ResNet-50 como en Yolo-V3. También es remarcable el efecto en MobileNet con capas casi sin aceleración ($S = 1$) mezcladas con capas con una buena aceleración ($S = 1{,}5$). Nótese que cada capa tiene un número diferente de datos de entrada y filtros.

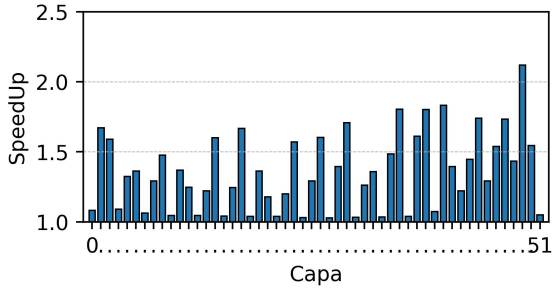
La figura 8 muestra la aceleración de cada estrategia en la cual sí se ha tenido en cuenta el tamaño de la capa para el cálculo. Esta figura también muestra la barra de error, representando la desviación típica de la repetición de entrada entre imágenes. Podemos ver que la estrategia AC explota mejor la repetibilidad de la entrada. Con el tamaño de grupo más pequeño ($G = 4$) tenemos una aceleración global de 1,5, 1,4 y $1,5$ en cada modelo, respectivamente. Como alternativa, podemos ahorrar el número de recursos para multiplicaciones en un 33,3 %, 28,6 % y 33,3 % para cada modelo, respectivamente. Con el mayor tamaño de grupo ($G = 16$), observamos con SC una aceleración global de $2,1$, $1,9$ y $2,4$ para cada modelo, lo que supone un 52 %, 47,4 % y 58,3 % de ahorro de multiplicaciones preservando la precisión del modelo. Las barras de error representadas en esta figura muestran que la repetición también se ve influida por la imagen estudiada. Por ejemplo, la estrategia NB es la que presenta una mayor variación en función de la imagen de entrada, seguida de la estrategia SC.

C. Repetición en los filtros

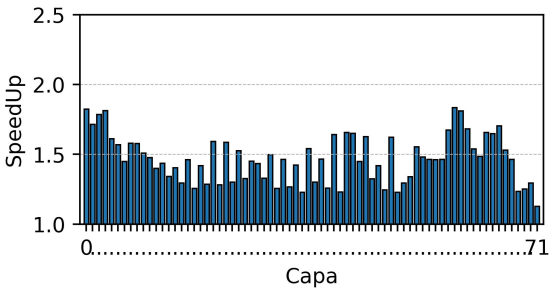
En este análisis final, se compara el FR encontrado en los filtros de los modelos, conocido como repetición de pesos (tal y como se explota en [10]). La



(a) ResNet-50



(b) MobileNet



(c) YoloV3

Fig. 7: Aceleración para SC por cada capa de los diferentes modelos. $G = 4$.

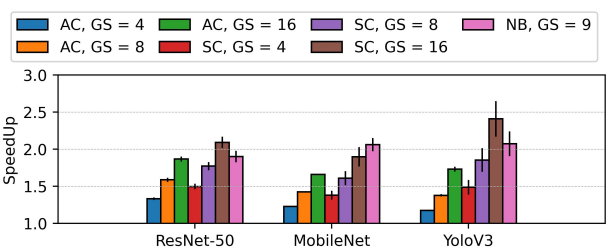


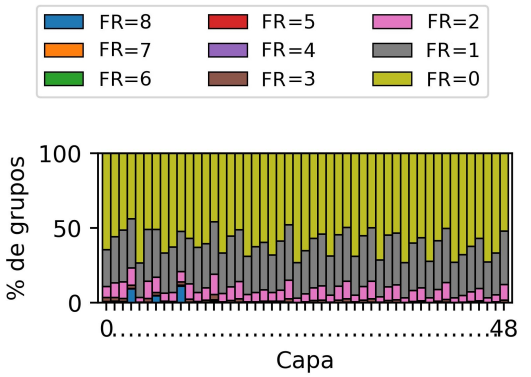
Fig. 8: Aceleración por cada estrategia con modelos y tamaño de grupo diferentes.

figura 9 muestra la repetición en los pesos de los tres modelos. El tamaño del grupo (G) se establece en 9. Como podemos observar, en la repetición de pesos predominan los FR ajustados a cero y a uno, que representan aproximadamente el 90 % en ResNet-50, el 97 % en MobileNet y el 75 % en Yolo-V3.

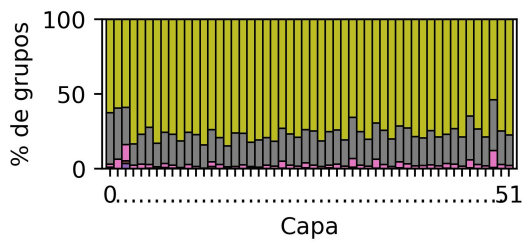
Estos resultados muestran que, explotando la repetición de entrada, el número de multiplicaciones ahorradas es considerablemente mayor. Por ejemplo, si comparamos los resultados de NB con los de la repetición de pesos, los FR medios en NB con $G = 9$

Tabla I: FR medio para cada modelo y estrategia de agrupación

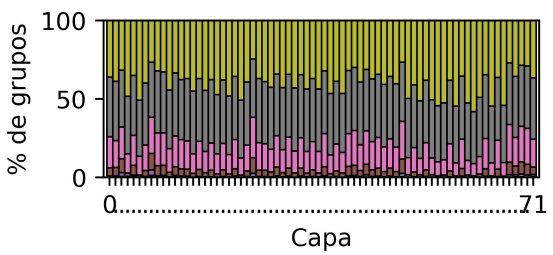
	SC					AC					NB
Group size:	4	8	16	32	64	4	8	16	32	64	9
ResNet-50	1.40	3.48	8.41	19.50	44.72	1.10	3.00	7.49	17.71	41.39	4.29
Yolo-V3	1.40	3.48	2.42	22.39	50.89	1.21	2.29	6.80	18.50	45.82	4.69
MobileNet	1.18	3.09	9.39	18.32	43.08	0.69	2.41	6.36	16.31	41,46	4.69



(a) ResNet-50



(b) MobileNet



(c) YoloV3

Fig. 9: Repetición en los pesos con $G = 9$.

son iguales a 4,29, 4,69 y 4,69 en los tres modelos respectivos. En cambio, en la repetición ponderada, los FR medios son iguales a 0,5, 0,4 y 0,9 para Resnet, Mobilenet y Yolo-V3, respectivamente.

V. Conclusiones

En este trabajo, se ha presentado un análisis de datos de repetición de la entrada de capas convolucionales de modelos CNN cuantificados en 8 bits. Se han utilizado los modelos Yolo-V3, ResNet-50 y MobileNet-v1, CNN del estado del arte. Este estudio demuestra que es factible la creación de un futuro acelerador *hardware* que reduzca el número de operadores de multiplicación explotando la repetición de entrada y, como resultado, reduciendo los recursos necesarios y el tiempo de ejecución, sin degradar la precisión. Se demuestra que existe una redundancia explotable en la entrada, y se calcula el factor de reducción teórico analizando tres estrategias de agrupación diferentes. Los resultados muestran que es posible reducir el 52 %, el 47,4 % y el 58,3 % de las multiplicaciones para ResNet-50, MobileNet-v1 y Yolo-V3, respectivamente, conservando toda su precisión.

VI. Trabajo futuro

Como trabajo futuro, se creará un acelerador especializado para detectar en tiempo de ejecución la redundancia de los datos de entrada de las capas convolucionales. Mas específicamente, se llevará a cabo la implementación de la detección SC, dado que en esta se han obtenido los mejores resultados en todos los modelos estudiados.

Agradecimientos

Este trabajo ha recibido financiación de el proyecto "Ifac: Implementing Fault-Tolerant Autonomous Computers (CISEJI/2022/30)" financiado por la Generalitat Valenciana.

Referencias

[1] Torsten Hoefler and et al., "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 10882–11005, 2021.

[2] "Onnx model zoo," `https://github.com/onnx/models`, Accessed: 2023-2-27.

[3] Joseph Redmon and Ali Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[4] Kaiming He, , and et al., "Deep residual learning for image recognition," in *Proceedings of the IEEE conf. on computer vision and pattern recognition*, 2016, pp. 770–778.

[5] Mark Sandler, , and et al., "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conf. on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[6] Sanghoon Kang, , and et al., "An overview of sparsity exploitation in cnns for on-device intelligence with software-hardware cross-layer optimizations," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 4, pp. 634–648, 2021.

[7] Lili Geng and Baoning Niu, "Pruning convolutional neural networks via filter similarity analysis," *Machine Learning*, vol. 111, no. 9, pp. 3161–3180, 2022.

[8] Marc Riera and et al., "Computation reuse in dnns by exploiting input similarity," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 57–68.

[9] Larissa Rozales Gonçalves and et al., "Using frame similarity for low energy software-only iot video recognition," in *Embedded Computer Systems: Architectures, Modeling, and Simulation: 19th International Conference, SAMOS 2019, Samos, Greece, July 7–11, 2019, Proceedings 19*. Springer, 2019, pp. 157–168.

[10] Kartik Hegde and et al., "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition,"

in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 674–687.

[11] Ali Yasoubi and et al., "Power-efficient accelerator design for neural networks using computation reuse," *IEEE computer architecture letters*, vol. 16, no. 1, pp. 72–75, 2016.

[12] Jia Deng, , and et al., "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conf. on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[13] Tsung-Yi Lin, , and et al., "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

Análisis de rendimiento y eficiencia de un sistema Deep Learning optimizado con TensorRT

Nicolás Hernández[1], Francisco Almeida[1] y Vicente Blanco[1]

Resumen— **En este documento se analizan el rendimiento y eficiencia energética de un sistema Deep Learning que emplea el aprendizaje mediante transferencia. Se ha usado Tensorflow para el desarrollo y entrenamiento de los modelos de aprendizaje junto a TensorRT para la optimización, haciendo uso de la cuantización post entrenamiento con el fin de reducir la carga computacional de los modelos y así ganar eficiencia sin perder precisión. La experiencia computacional ha sido desarrollada en un ordenador de propósito general con un procesador Intel i7-1260P y una tarjeta gráfica NVIDIA RTX 3080 usada como aceleradora, y un NVIDIA Jetson AGX Orin como ejemplo de dispositivo IoT, a fin de comparar la mejora obtenida al optimizar el modelo con la biblioteca TensorRT y al emplear el acelerador de inferencia Deep Learning Accelerator (DLA).**

En este trabajo se estudia la viabilidad del entrenamiento en un dispositivo IoT, el impacto de la optimización por cuantización en un sistema entrenado por transferencia de conocimiento y el análisis del rendimiento y eficiencia de la versión 2.0 del acelerador de inferencia de NVIDIA.

Los resultados obtenidos muestran tiempos de entrenamiento mayores en la Jetson aunque asumibles, mientras que el tiempo de inferencia puede llegar a ser similar en los distintos dispositivos gracias al proceso de optimización. Ambos procesos han mostrado una mayor eficiencia energética en la Jetson, no obstante, debido a la incompatibilidad con determinadas capas, el uso del DLA empeora los resultados tanto en rendimiento como eficiencia.

Palabras clave— **Deep Learning, TensorRT, NVIDIA Jetson, Cuantización, Aceleración de IA.**

I. Introducción

Entre los constantes avances tecnológicos que se han dado en el campo de la Inteligencia Artificial (IA), y más concretamente el Deep Learning (DL) con sus redes neuronales artificiales, es posible determinar los dos principales obstáculos tecnológicos que se deben afrontar para su desarrollo. En primer lugar, las redes de mayor complejidad necesitan de máquinas de una capacidad de cómputo sólo al alcance de los supercomputadores para obtener tiempos de entrenamiento asequibles, mientras que en las fases de inferencia es cada vez más común la necesidad de emplear aceleradores a fin de conseguir mejoras de rendimiento. En segundo lugar, y directamente relacionado con lo anterior, está el alto consumo energético que suponen estas operaciones, en particular la fase de entrenamiento.

Es posible apreciar la difícil integración de esta tecnología en el contexto de los dispositivos móviles y del Internet de las Cosas (IoT). En este tipo de dispositivos, de bajo consumo a costa de un menor rendimiento, es común encontrar casos de uso en los que se ha disminuido el rendimiento de forma notable con el fin de poder ejecutar los procesos de inferencia, dado que el entrenamiento no es viable dadas las bajas especificaciones de tales dispositivos.

El *Deep Learning* ha ganado en importancia e ciertas áreas frente a las técnicas clásicas de aprendizaje automático al demostrar una mejor capacidad de aprendizaje y un menor error de generalización ante problemas con grandes cantidades de datos. Este es el caso de los problemas de Visión por Computador [1] que nos ocupan en este trabajo pero también en otros problemas como reconocimiento de voz [2] y el procesamiento del lenguaje natural [3].

Asociados al *Deep Learning* surgen otros inconvenientes, el diseño de una red neuronal artificial desde cero es una tarea de gran dificultad, en especial si se pretende que lleve a cabo tareas de gran complejidad. Por un lado se debe diseñar una red para la tarea a desarrollar, con la dificultad que esto supone, y por otro, el modelo debe ser lo suficientemente complejo como para realizar su función con la suficiente precisión, sin que esto suponga una carga de trabajo inasumible para dispositivos IoT.

En este trabajo se optó por el reconocimiento facial como caso de uso. Se parte de redes neuronales pre-entrenadas para el reconocimiento facial, y se hace uso de la transferencia de conocimiento [4] con el fin de adaptar estas redes a la tarea en cuestión. De este modo, es posible mantener un alto nivel de precisión y rendimiento, empleando un conjunto de datos relativamente pequeño para su entrenamiento. Como etapa final se procede a optimizar el modelo, con el objetivo de reducir la carga computacional, lo que a su vez se traduce en un menor consumo, con un impacto mínimo en la precisión.

Se estudian modelos de IA mediante la transferencia de conocimiento de modelos pre-entrenados, con el objetivo de que dichos modelos sean capaces de identificar el rostro de una persona concreta frente a cualquier otro rostro, todo ello con un rendimiento y consumo energético razonables en un dispositivo IoT. Para este desarrollo se ha decidido emplear la librería TensorFlow para el desarrollo de la red, el kit de desarrollo NVIDIA Jetson AGX Orin como dispositivo especializado y la librería TensorRT para la optimización de los modelos.

Los resultados obtenidos muestran tiempos de entrenamiento en la Jetson asumibles, aunque de mayor duración que los obtenidos en el ordenador de

[1]Dpto. de Ingeniería Informática y de Sistemas, Universidad de La Laguna, e-mail: nhernang@ull.es, falmeida@ull.es, vblanco@ull.es.

escritorio (DC). Gracias al proceso de optimización llevado a cabo por TensorRT, el tiempo de inferencia en ambos dispositivos mejora hasta tal punto que la diferencia entre dispositivos es mínima. Las tareas de entrenamiento e inferencia han mostrado una mayor eficiencia energética en la Jetson, no obstante, debido a la incompatibilidad de los modelos pre-entrenados con el DLA, el uso de acelerador empeora los resultados tanto en rendimiento como eficiencia, en comparación a su uso únicamente en la GPU.

A continuación enumeramos las principales contribuciones de este trabajo:

- Se demuestra la viabilidad de ejecutar el entrenamiento directamente en el dispositivo IoT, un aspecto que comúnmente se descarta en trabajos similares.
- Se evidencia que el proceso de optimización y cuantización tiene un menor impacto en la precisión de los modelos entrenados mediante aprendizaje por transferencia, debido a la reducción del número de categorías a clasificar.
- El análisis de la versión más reciente hasta la fecha del acelerador de inferencia DLA de NVIDIA confirma la presencia de problemas de compatibilidad con los modelos pre-entrenados, lo que afecta el rendimiento tal y como ocurre en su primera versión. No obstante, se observa una mejora en la eficiencia energética en aquellos modelos con menos incompatibilidades, en comparación con el uso de la GPU.

En las secciones posteriores veremos el estado del arte y trabajos similares relacionados con este proyecto, la metodología empleada en el desarrollo, que incluye una descripción de los dispositivos y redes neuronales empleadas, los resultados experimentales de rendimiento y consumo energético en las fases de entrenamiento e inferencia, y por último, las conclusiones extraídas de los resultados.

II. Estado del arte

El incremento en potencia de cálculo de los dispositivos actuales y la ingente cantidad de información almacenada de la se que dispone hoy día han propiciado el auge y expansión de la IA más allá de los campos con los que se relacionaba en sus inicios. La proliferación de la IA también ha hecho que sus casos de uso aumenten proporcionalmente en tareas cada vez más complejas, como es la visión artificial o el procesamiento de lenguaje natural, y es precisamente ahí donde el *Deep Learning* gana protagonismo frente a los algoritmos clásicos de IA [5].

Las posibilidades que ofrece el *Deep Learning* continúan hoy día siendo objeto de debate y estudio [6], no obstante, además de ventajas esta tecnología también entrañaba dificultades. El alto coste del desarrollo de una red neuronal desde cero, se requieren modelos cada vez más sofisticados para resolver tareas cada vez más complejas y la necesidad de una gran cantidad de datos que permita a nuestro modelo generalizar correctamente. En este trabajo se hará uso de la capacidad de transferencia de conocimiento [6] de las redes neuronales, técnica que aplica los conocimientos que se adquieren al resolver un problema en un problema diferente, pero relacionado, y métodos como el aumentado artificial de datos [7] para solventar estos inconvenientes.

En trabajos existentes en los que se han desarrollado sistemas *Deep Learning* para su ejecución en dispositivos IoT, como el de un sistema de autenticación para sistemas médicos inteligentes [8] o de control de acceso a un hogar inteligente [9], ambos empleando una Raspberry Pi 3, constatan que, a pesar de obtener buenos resultados en cuanto a precisión, el mayor impedimento es el pobre rendimiento computacional que se consigue al ejecutar los modelos en el dispositivo.

Utilizando el dispositivo Jetson Orin, se emplea la librería TensorRT para optimizar los modelos usando principalmente la técnica de la cuantización [10], que ha demostrado conseguir aceleraciones considerables en los modelos de visión artificial más populares. Por su parte, el dispositivo Jetson dispone de dos aceleradores de *Deep Learning* (DLA) para los procesos de inferencia. En [11] se estudia la posibilidad de emplear un sistema de conducción autónoma en el dispositivo Jetson AGX Xavier, demostrando el buen rendimiento que es posible obtener en parte gracias a que el uso del DLA es capaz de reducir la carga de trabajo de la GPU. La optimización realizada en [11] es diferente a la propuesta en este paper, básicamente se centran en realizar un planificación del uso de los dispositivos para poder optimizar el tiempo de uso de la GPU.

En [12] se hace un análisis exhaustivo sobre cómo afecta la cuantización post-entrenamiento a un transformador y una red neuronal. Se realizan mediciones de consumo energético, además de las de rendimiento, en múltiples dispositivos empleando diferentes *frameworks*. En nuestra propuesta, sin embargo, realizamos los análisis sobre múltiples modelos que han servido como base para la transferencia de conocimiento. Nuestro objetivo es medir si la pérdida de precisión causada por la cuantización es menor debido a la simplificación resultante de la disminución del número de categorías clasificables. Además, en el trabajo citado se menciona el uso de DLAs, pero no se han realizado medidas exhaustivas sobre su rendimiento y consumo, a diferencia de nuestro estudio en el que sí se evalúan de manera individualizada.

En trabajos donde se evalúa la optimización de modelos entrenados mediante aprendizaje por transferencia, como el estudio realizado en [13], que examina múltiples modelos, y la propuesta realizada en [14], que mide la mejora del rendimiento lograda en la tarea de inferencia utilizando un solo dispositivo. En contraste, este trabajo aborda tanto la eficiencia energética de los modelos como su rendimiento, comparando su desempeño en dos dispositivos diferentes. A diferencia de los trabajos citados anteriormente, que se centran exclusivamente en la tarea de inferencia, nuestro estudio valora también la fase de

entrenamiento. En este proyecto se evalua la viabilidad de ejecutar el entrenamiento en un dispositivo IoT, tanto en términos de tiempo como de consumo energético.

III. Metodología

Para el desarrollo de este proyecto se ha elegido como caso de uso la elaboración de un sistema de reconocimiento facial, cuya funcionalidad objetivo será la de distinguir el rostro de una persona concreta frente a cualquier otro rostro, en otras palabras, tendremos un sistema de clasificación binario.

Para entrenar y validar dicho sistema, se dispone de un total de 1.100 imágenes que se distribuyen uniformemente entre las categorías. Estas imágenes tienen una dimensión de 128x128 y contienen un único rostro cada una. Las 550 imágenes de rostros diversos obtenidas de la base de datos '*Flickr-Faces-HQ dataset*' [15], mientras que las 550 restantes del rostro objetivo se han recolectado manualmente, puesto que se tratan de imágenes del rostro de uno de los autores de este documento. La división de las imágenes entre conjunto de entrenamiento, validación y verificación es de 700, 300, y 100 imágenes respectivamente.

En lo que respecta a los dispositivos en los que se ha realizado la experiencia computacional, en la Tabla I se muestran las especificaciones tanto de la NVIDIA Jetson como del ordenador de escritorio usado como referencia. Obsérvese que las especificaciones del DC son superiores a las del resto de los equipos en términos de rendimiento, debe tenerse en cuenta que el consumo de la Jetson está limitado a un máximo de 50W. Por último podemos destacar el *Deep Learning Accelerator* o DLA, acelerador diseñado para mejorar el rendimiento y la eficiencia en la inferencia de modelos de *Deep Learning*.

Tabla I: Especificaciones de los dispositivos

	DC	**Jetson**
CPU	Intel Core i7-1260P 12 núcleos 4,7 GHz	Arm CortexA78AE 12 núcleos 2,2 GHz
GPU	GeForce RTX3080 8704 CUDA cores 272 tensor cores	NVIDIA Ampere 2048 CUDA cores 68 tensor cores
DLA	-	2x NVDLA v2.0 1,6 GHz

A. Redes Neuronales

Para el desarrollo del sistema haremos uso de la plataforma TensorFlow y la librería Keras, empleando el aprendizaje por transferencia con cinco redes neuronales de diferentes características como modelos base, MobileNet-v2 [16], ResNet-50 [17], ResNet-50 [17], Xception [18] y Vgg16 [19]. El la Tabla II podemos ver las características de las redes antes mencionadas según sus implementaciones disponibles en TensorFlow. La profundidad hace referencia al número de capas, mientras que los parámetros son la suma del total de neuronas y las conexiones entre capas.

La red MobileNet se destaca por su capacidad de ser eficiente en términos de capacidad computacional, lo que la hace adecuada para dispositivos con limitaciones de hardware. ResNet50 y ResNet152 se basan en la idea de bloques residuales, permitiendo un entrenamiento más profundo y una mayor precisión en la clasificación de imágenes. Xception, por otro lado, introduce la divisibilidad de las convoluciones, lo que reduce la cantidad de parámetros y operaciones en comparación con otras arquitecturas. Por último, la red VGG16 es conocida por su simplicidad y profundidad, empleando convoluciones de tamaño reducido y múltiples capas, lo que resulta en una alta capacidad de aprendizaje.

Tabla II: Comparativa de redes neuronales

Red	**Profundidad**	**Parámetros**
MobileNet	105	3.538.984
ResNet50	107	25.636.712
ResNet152	311	60.419.944
Xception	81	22.910.480
Vgg16	16	138.357.544

Para el entrenamiento de estas redes se han elegido medidas de precisión y valor de pérdida binarias, debido a que nuestro modelo únicamente categoriza en dos clases. Así mismo, se ha optado por hacer uso de la parada temprana o *early stopping* [20], empleando como criterio de parada la no mejoría del valor de pérdida en un número determinado de iteraciones; de esta forma conseguimos reducir el número de iteraciones del entrenamiento y en consecuencia la carga computacional.

B. Optimización

El proceso de optimización de los modelos se ha llevado a cabo mediante el uso de la librería TensorRT, cuyo proceso podemos resumir en dos fases. En la primera fase se proporciona a TensorRT una definición del modelo, que será optimizado para una GPU destino; en nuestro caso este proceso de optimización implica entre otros la cuantización, más concretamente, la cuantificación post entrenamiento (PTQ). En la segunda fase, se utiliza el modelo optimizado para ejecutar la inferencia. El modelo resultante del proceso de mejora sólo obtendrá el máximo rendimiento si se ejecuta en el mismo dispositivo en el que ha sido optimizado. Las razones principales que impulsan el uso de esta librería son tanto su especialización en GPU de NVIDIA como ser la forma recomendada de uso del acelerador DLA.

Para cada uno de los modelos se han elaborado cuatro versiones optimizadas, en las que se han visto modificadas la anchura de bits de las operaciones con el proceso de cuantización: 32 bits en punto flotante (FP32), 16 bits en punto flotante (FP16) y 8 bits de enteros (INT8). Para la ejecución en el DLA se crea de forma adicional un nuevo modelo optimizado con precisión de 8 bits de enteros, la única diferencia radica en que se especifica el acelerador como responsable de la ejecución, mientras que la GPU actúa como una unidad de respaldo.

C. Medición del consumo energético

Con el fin de estimar el consumo energético de las tareas de entrenamiento e inferencia, se ha empleado la API EML (*Energy Measurement Library*) [21]. Esta herramienta, desarrollada por el Grupo de Computación de Altas Prestaciones (GCAP) de la Universidad de La Laguna, simplifica la medición del consumo energético al presentar una interfaz unificada para distintos tipos de hardware, encapsulando la interfaz RAPL para sistemas Intel y empleando NVML para GPUs de Nvidida entre otros.

El diseño del dispositivo Jetson únicamente permite tomar las medidas de consumo energético de dos grupos de componentes, el primer grupo lo conforman el SoC (System on a Chip), que es el chip principal de la Jetson, excluyendo los sub-bloques principales como CPU, GPU y aceleradores, junto con la GPU, y el segundo la CPU y los distintos aceleradores disponibles en el dispositivo, entre los que se encuentra el DLA. No es posible medir de forma directa e independiente el consumo energético de la CPU y los aceleradores.

IV. Resultados experimentales

Los resultados computacionales se han dividido en dos grupos, el de rendimiento y el de eficiencia, que a su vez distinguen las fases de entrenamiento e inferencia.

A. Rendimiento

Los valores comprendidos dentro de la categoría de rendimiento son la precisión, el valor de la función de pérdida y el tiempo de ejecución para la fase de entrenamiento, y la variación de la precisión, la latencia y la capacidad de procesamiento o *throughput* tras la optimización en la fase de inferencia. Con la precisión medimos cuán exacto es el modelo en la clasificación o predicción de los datos, mientras que el valor de la función de pérdida se usa para cuantificar la discrepancia entre las predicciones del modelo y los valores reales. Por latencia hacemos referencia a la suma del tiempo de copia a la GPU, el tiempo de inferencia y la copia desde la GPU, en cambio, la capacidad de procesamiento o *throughput* hace referencia a la cantidad de imágenes que capaz de procesar la red en un intervalo de tiempo determinado. Adicionalmente se mide la aceleración conseguida por la optimización de los modelos, calculada según el cociente de reducción de la latencia y de aumento del *throughput*.

A.1 Entrenamiento

Los valores de precisión y pérdida de un modelo no se ven afectados según el dispositivo o unidad de procesamiento en que se ejecute la fase de entrenamiento, a diferencia del tiempo de entrenamiento, que está directamente relacionado con la capacidad de procesamiento de la unidad que ejecuta la tarea.

Como puede observarse en la Figura 1, todos los modelos obtienen valores de precisión que superan el 90 por ciento sin diferencias significativas entre ellos; además de valores bajos de pérdida, con la red

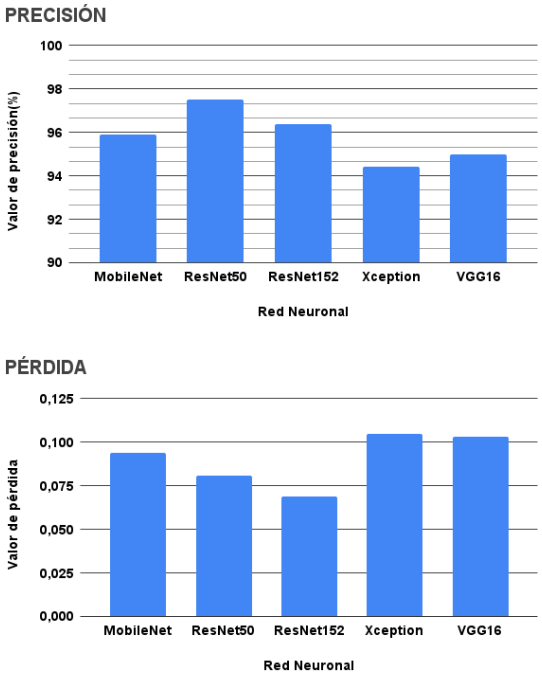


Fig. 1: Valores de precisión y pérdida

ResNet152 destacando sobre el resto por un corto margen, no obstante, para las funciones de entropía cruzada, el tipo de función pérdida que se emplea en este trabajo, los valores para una predicción de gran calidad son menores a 0,02 o incluso 0,05 [22], podemos concluir que los resultados son satisfactorios pero lejos de los valores óptimos. Con estos resultados es posible apreciar con claridad las ventajas del aprendizaje por transferencia, gracias a que podemos conseguir un modelo de gran precisión sin necesidad de desarrollarlo desde cero y empleando un *dataset* relativamente pequeño de 1.100 imágenes; en comparación, el dataset con el que se han entrenado originalmente los modelos, Imagenet [23], cuenta con más de 14 millones de imágenes.

La Figura 2 muestra los resultados obtenidos para el tiempo de entrenamiento con una diferencia de tiempo notoria entre CPU y GPU; con la DC consiguiendo realizar la tarea en menor tiempo en comparación con la Jetson, especialmente al comparar los tiempos conseguidos por las GPUs de ambos dispositivos. No obstante, esta diferencia se reduce considerablemente en la red MobileNet que está diseñada para ser más ligera que sus homólogas

Sin embargo, en la GPU la red MobileNet muestra un tiempo superior a determinadas redes en la GPU del DC, esto se debe a la parada temprana mencionada en la sección anterior, puesto que, a pesar de ser más rápida, necesita más iteraciones que el resto para alcanzar los valores óptimos, lo que desemboca en una mayor duración final únicamente porque los tiempos son considerablemente reducidos de por sí en una GPU de alto rendimiento como la RTX 3080.

Como se observa en la tabla III, la CPU del dispositivo Jetson logra un tiempo aproximadamente el doble del obtenido por la CPU del DC, excepto en la

Red neuronal	Dispositivo	Tiempo en DC	Tiempo en Jetson	Ratio
MobileNet	CPU	192,969	225,397	1,168
	GPU	33,279	72,255	2,171
	Ratio	6,773	3,119	
ResNet50	CPU	631,882	1124,081	1,835
	GPU	26,138	136,3	5,215
	Ratio	43,006	8,247	
ResNet152	CPU	1855,331	3403,608	2,228
	GPU	66,704	168,331	2,524
	Ratio	51,026	20,22	
Xception	CPU	602,647	1342,543	1,779
	GPU	39,053	98,26	2,516
	Ratio	34,377	13,663	
Vgg16	CPU	1931,634	3299,493	1,708
	GPU	28,017	148,25	5,291
	Ratio	117,768	22,256	

Tabla III: Tiempos de ejecución de la fase de entrenamiento

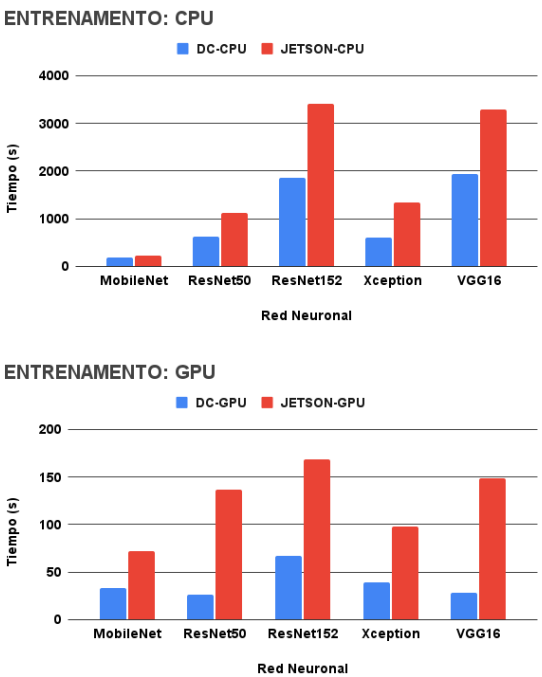


Fig. 2: Tiempos de entrenamiento

red MobileNet donde obtiene una cifra muy similar. Al comparar los tiempos de GPU, se observa que la Jetson presenta un rendimiento notablemente inferior, aproximadamente cinco veces más lento, en las redes computacionalmente más costosas como ResNet152 y Vgg16; mientras que en las demás redes logra realizar el entrenamiento en aproximadamente el doble de tiempo. Los tiempos obtenidos por la GPU, con el modelo más lento necesitando alrededor de tres minutos, demuestran que la NVIDIA Jetson podría ser capaz de realizar el entrenamiento en el mismo dispositivo si los requerimientos del caso de uso lo permiten.

A.2 Inferencia

El resultado del proceso de optimización llevado a cabo por la librería TensorRT es un modelo afinado para la GPU en la que se ha ejecutado el proceso de mejora. Por este motivo, las comparaciones del proceso de inferencia se han realizado únicamente sobre la unidad gráfica RTX 3080 y la GPU integrada en la NVIDIA Jetson. Cada inferencia se ejecuta sobre un lote de 32 imágenes en lugar de imágenes individuales con el objetivo de obtener el máximo rendimiento.

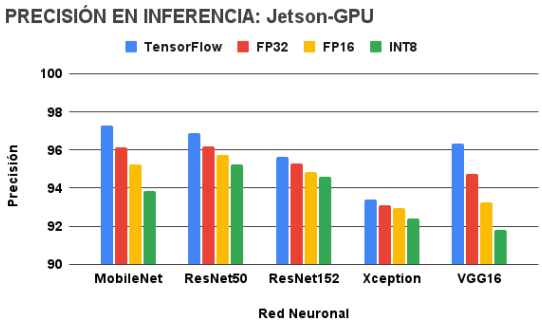


Fig. 3: Comparativa de precisión en GPU de la Jetson tras la optimización

Los resultados obtenidos en la ejecución sobre la GPU del dispositivo Jetson (Figura 3) ponen de manifiesto una reducción de la precisión en torno al 5 por ciento en el peor de los casos. La redes MobileNet y Vgg16 se muestran más susceptibles a la pérdida de precisión, mientras que el resto presentan una disminución notablemente baja. Los resultados no han mostrado diferencias significativas entre ambos dispositivos.

En la figura 4 se muestran los valores de latencia para cada red, modelo y dispositivo. Nótese que que los resultados mantienen la tendencia observada en los tiempos de entrenamiento, con MobileNet nuevamente a la cabeza mostrando la menor latencia, mientras que el dispositivo Jetson obtiene unos valores de latencia aproximadamente del doble de los conseguidos en el DC. Al comparar los modelos optimizados con TensorRT en diferentes precisiones, tamaño en bits de las operaciones, se pone de manifiesto una importante reducción de la latencia incluso con el punto flotante de 32 bits, que es la precisión original del modelo; las operaciones en INT8 son capaces de alcanzar una reducción de en torno al 90 por ciento respecto al modelo original.

La figura 5 muestra los valores de aceleración obtenidos a partir los datos de latencia, dejando patente que los valores obtenidos en la DC son aproximadamente el doble a los obtenidos con la Jetson. La aceleración mejora de forma lineal a medida que la an-

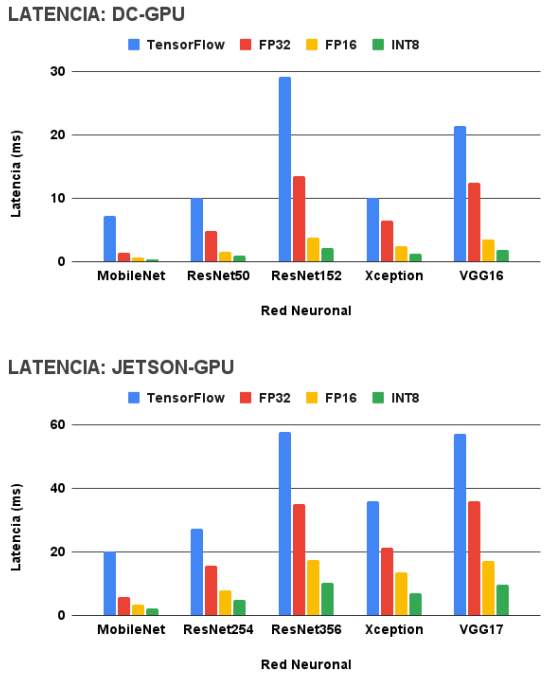


Fig. 4: Latencia en inferencia

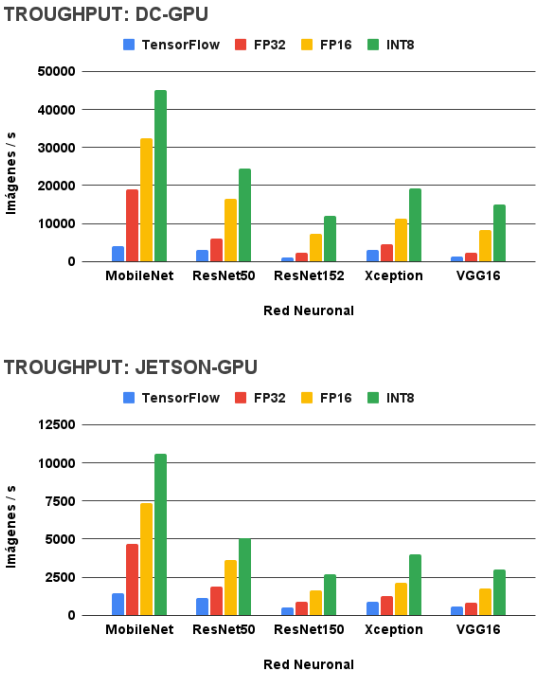


Fig. 6: Capacidad de procesamiento en inferencia

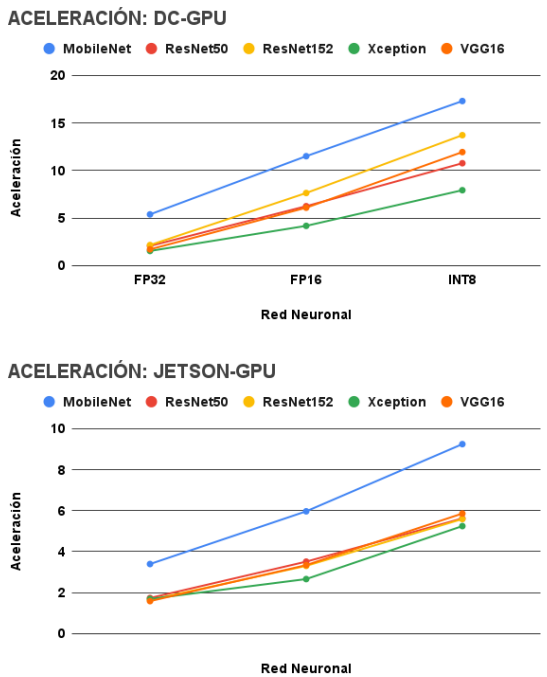


Fig. 5: Aceleración en inferencia

chura en bits de las operaciones disminuye, donde podemos destacar que en la Jetson la mejora del INT8 respecto al FP16 es mayor que en el DC; además de que nuevamente la red MobileNet consigue los mejores valores en contraposición con la red Xception, que tiene la peor optimización con TensorRT. La redes restantes consiguen resultados similares entre sí, tanto en el DC como en la Jetson.

En la figura 6 pueden verse los resultados de la capacidad de procesamiento o *troughput*, que nos muestran un resultado inverso al observado anteriormente, es decir, con un descenso en la precisión de las operaciones se consigue un mayor número de predicciones por segundo. Sin duda alguna, la red MobileNet es la que más se ve beneficiada por el proceso de optimización, seguida por la red ResNet50, mientras que en el resto de los modelos se observa un factor de mejora similar entre ellos. Si en la latencia se observaba una diferencia en la reducción de la latencia aproximada del 50 por ciento entre ambos dispositivos, la diferencia del aumento de rendimiento muestra una mejora aproximada del 450 por ciento en favor del DC. Esta diferencia entre latencia y capacidad de procesamiento podemos asociarla a que el proceso de inferencia se realiza sobre un lote de imágenes en lugar de una única imagen.

En la figura 7 se muestra gráficamente el aumento en la capacidad de procesamiento obtenido con el proceso de optimización con TensorRT. Si bien en la Jetson las aceleraciones son similares a las obtenidas con la latencia, en el DC, con una GPU con capacidad de computación superior, las aceleraciones de ResNet50 y VGG16 consiguen una aceleración superior al de la red MobileNet en la precisión de INT8, la manera en que aumenta el rendimiento a medida que reduce la precisión es notablemente superior en estas dos redes en comparación con el resto. Otro punto a destacar es el cómo la diferencia de aceleración entre ambos dispositivos ya no es cercana al doble, como en la latencia, sino que ronda el 30 por ciento; por último, la red Xception sigue mostrando los peores resultados por un escaso margen.

La figura 8 muestra los datos comparativos de los resultados obtenidos por los modelos optimizados en INT8 usando y sin usar el DLA, ambos en la Jetson. Los resultados muestran que el acelerador reduce el rendimiento de los modelos optimizados con TensorRT de forma significativa. La causa de este po-

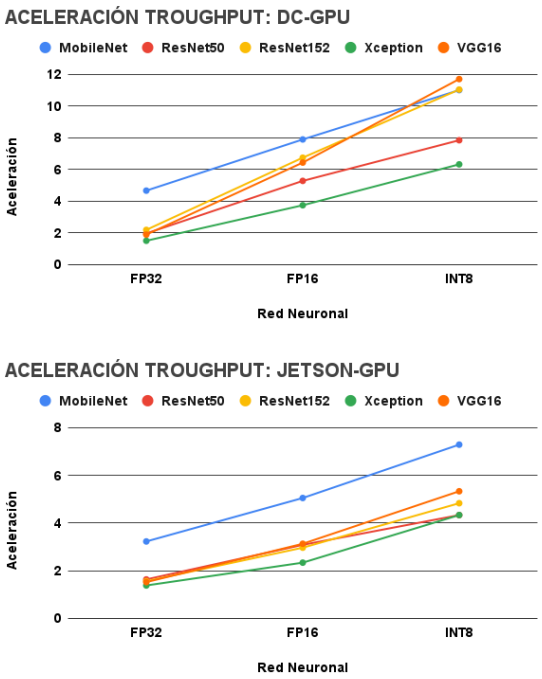


Fig. 7: Aumento en capacidad de procesamiento

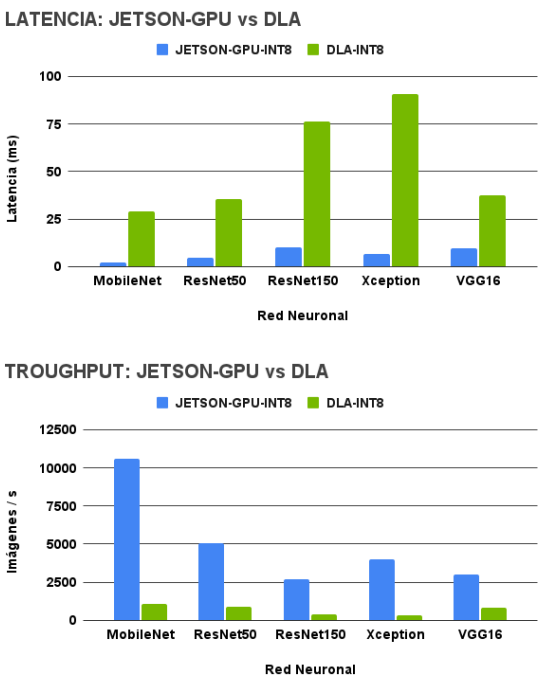


Fig. 8: Comparativa de resultados con DLA

bre rendimiento es la incompatibilidad de gran parte de las capas de las redes pre-entrenadas, la ejecución del modelo el acelerador necesita enviar los datos de vuelta a la GPU de forma constante, lo que repercute negativamente en el rendimiento de forma ostensible.

B. Eficiencia Energética

En la medición de la eficiencia energética valoraremos el consumo del proceso de entrenamiento según que dispositivo y unidad de procesamiento lo está llevando cabo. Al mismo tiempo cuantificamos como afecta la optimización al consumo del proceso de inferencia de cada uno de los modelos. Para la tarea de inferencia, debido a que es un proceso de muy corta duración, se han realizado las mediciones ejecutando un total de 1000 inferencias para posteriormente calcular el consumo individual de cada inferencia en base a la media de los resultados obtenidos.

B.1 Entrenamiento

Como puede observarse en la figura 9, los resultados obtenidos son opuestos a los vistos en la figura 2 con el tiempo consumido por la tarea de entrenamiento, donde se observa que si bien la Jetson necesitaba más tiempo para completar la tarea, el consumo de energía es notablemente menor en comparación con las unidades de procesamiento del DC. Al comparar los resultados de la CPU frente a GPU, podemos afirmar que el procesador gráfico es más eficiente, en gran parte debido a que el tiempo de entrenamiento es considerablemente menor; en especial la diferencia de consumo energético en GPU es significativa a favor de la Jetson, máxime si tenemos en cuenta que el aumento en tiempo es de menor orden que la reducción del consumo energético.

Los resultados de los diferentes modelos muestran que la red Vgg16 consigue una eficiencia notoria si se entrena en un procesador de altas prestaciones, mientras que en dispositivos menos potentes obtendrá el mayor consumo de todas las redes, mostrando un cierto grado de incompatibilidad con los dispositivos IoT, que por norma general no disponen de gran capacidad de cálculo. La GPU de la Jetson manifiesta una mayor eficiencia energética al entrenar modelos de gran cuantía de capas, frente al modelo Vgg16 con pocas capas pero de gran densidad.

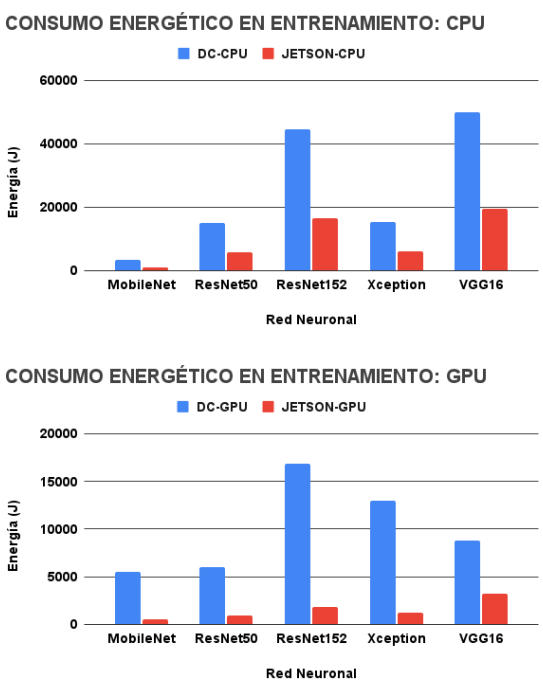


Fig. 9: Comparativa del consumo energético en entrenamiento

B.2 Inferencia

De forma similar que al examinar el rendimiento, en la figura 10 se comparan los valores de consumo energético para cada red, modelo y dispositivo. Siguiendo un patrón similar visto en el análisis de la latencia, tanto en el DC como en la Jetson el proceso de optimización consigue una sustancial reducción del consumo, en especial para las redes más pesadas, como son la ResNet152 y la Vgg16. El modelo de menor consumo en el DC consigue unos valores similares al modelo de mayor consumo en la Jetson, lo que pone de manifiesto la destacable eficiencia del dispositivo. En lo que respecta al comportamiento de los modelos, todos ellos reciben una mejora notoria, no obstante, una vez más la red Xception muestra que el proceso de optimización consigue una mejoría de menor índice que el resto, particularmente apreciable en los resultados obtenidos en la Jetson.

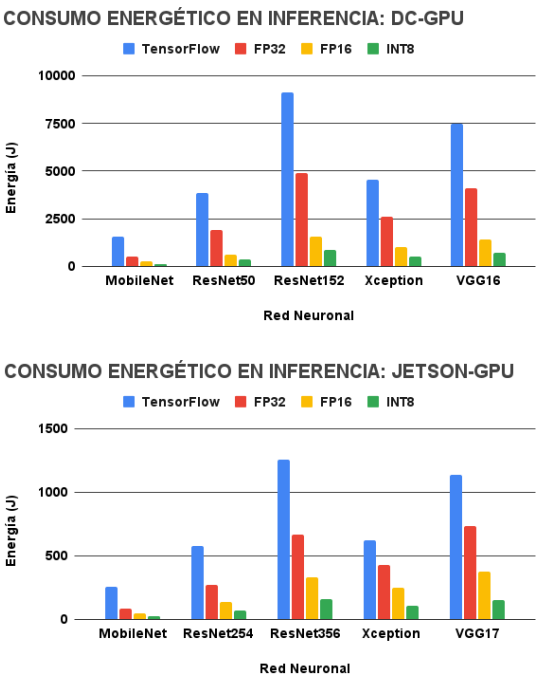


Fig. 10: Comparativa del consumo energético en inferencia

La figura 11 representa los resultados obtenidos por los modelos optimizados con precisión INT8 empleando el DLA con la GPU como respaldo y al emplear únicamente la GPU, ambos en la Jetson. Debido a que no es posible medir el consumo energético de los aceleradores de forma independiente a la CPU, el valor de consumo del DLA es la suma del valor del propio acelerador, la GPU que actúa como respaldo y una sobrecarga añadida por el consumo de la CPU.

Como podíamos esperar, la incompatibilidad de gran parte de las capas de los modelos con el DLA no consigue una mejoría en la eficiencia energética, al contrario, el consumo aumenta con una única excepción. La red Vgg16 muestra un menor consumo energético al usar el acelerador, esto se debe a que la proporción de capas incompatibles con el DLA es considerablemente menor que en los otros modelos, debido a que únicamente se compone de 16 capas.

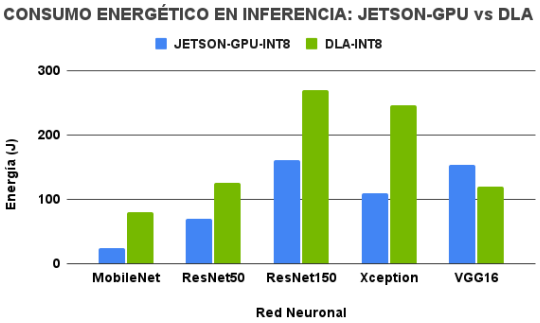


Fig. 11: Comparativa del consumo energético con DLA

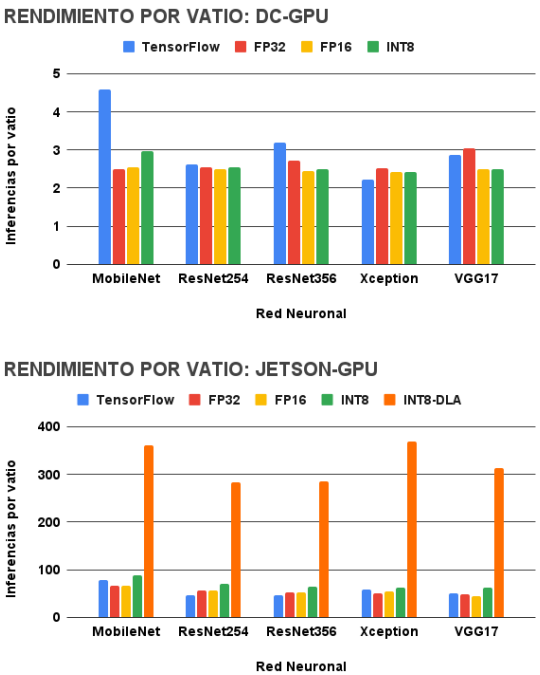


Fig. 12: Comparativa de rendimiento por vatio en inferencia

Para concluir, la figura 12 ilustra el rendimiento por vatio en cada uno de los dispositivos. Las mediciones en el DC muestran un menor rendimiento por vatio generalizado tras el proceso de optimización, lo que indica que la disminución de la latencia y el aumento del *troughput* se consigue a cambio de una reducción de la eficiencia energética. En el dispositivo Jetson se observa, por el contrario, una mejoría generalizada del rendimiento por vatio, en especial con la cuantización a INT8, no obstante, el aspecto más destacable es el notorio rendimiento conseguido por el DLA. El acelerador, aún teniendo un peor desempeño en lo que respecta a latencia y *troughput*, consigue un rendimiento por vatio muy superior a cualquiera de los otros modelos optimizados ejecutados únicamente en la GPU, lo que demuestra mayor eficiencia energética frente al procesador gráfico.

V. Conclusiones

En este trabajo se ha desarrollado un sistema de *Deep Learning* con el objetivo de medir y comparar el rendimiento y la eficiencia energética de las tareas de entrenamiento e inferencia. La inferencia ha sido

optimizada utilizando la librería TensorRT. Las medidas se han obtenido en el dispositivo IoT NVIDIA Jetson AGX Orin, en comparación con un ordenador de propósito general equipado con un procesador Intel i7-1260P y una tarjeta gráfica NVIDIA RTX 3080. Los resultados demuestran la viabilidad de realizar el entrenamiento en el dispositivo IoT, así como la posibilidad de obtener un rendimiento similar al de un procesador con mejores especificaciones, pero con un menor consumo energético gracias al proceso de optimización. El análisis del acelerador de inferencia DLA en su versión 2.0 muestra un desempeño inferior respecto al obtenido por la GPU del dispositivo debido a la incompatibilidad con las capas de las redes neuronales empleadas, sin embargo, exhibe un destacado rendimiento por vatio, superando a las demás unidades de procesamiento en términos de eficiencia energética.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia e Innovación de España con los proyectos PID2019–107228RB–I00, TED2021–131019B–I00 y PDC2022–134013–I00; y por el Gobierno de Canarias con el proyecto ProID2021010012.

Referencias

[1] Zhiliang Zhang, Limei Zhao, and Tao Yang, "Research on the application of artificial intelligence in image recognition technology," *Journal of Physics: Conference Series*, vol. 1992, no. 3, pp. 032118, aug 2021.

[2] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE Access*, vol. 7, pp. 19143–19165, 2019.

[3] Amirsina Torfi, Rouzbeh A. Shirvani, Yaser Keneshloo, Nader Tavaf, and Edward A. Fox, "Natural language processing advancements by deep learning: A survey," 2021.

[4] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He, "A comprehensive survey on transfer learning," 2020.

[5] Sambit Mahapatra, "Why deep learning over traditional machine learning?," `https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063`, Accessed: 2023-02-22.

[6] Terrence J. Sejnowski, *The Deep Learning Revolution*, The MIT Press, 10 2018.

[7] Suorong Yang, Weikang Xiao, Mengcheng Zhang, Suhan Guo, Jian Zhao, and Furao Shen, "Image data augmentation for deep learning: A survey," 2022.

[8] Tahir Hussain, Dostdar Hussain, Israr Hussain, Hussain lSalman, Saddam Hussain, Syed Sajid Ullah, and Suheer Al-Hadhrami, "Internet of things with deep learning-based face recognition approach for authentication in control medical systems," *Computational and Mathematical Methods in Medicine*, vol. 2022, Feb 2022.

[9] Syafeeza Ahmad Radzi, MK Mohd Fitri Alif, Y Nursyifaa Athirah, AS Jaafar, AH Norihan, and MS Saleha, "Iot based facial recognition door access control home security system using raspberry pi," *International Journal of Power Electronics and Drive Systems*, vol. 11, no. 1, pp. 417, 2020.

[10] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer, "A survey of quantization methods for efficient neural network inference," 2021.

[11] Hsin-Hsuan Sung, Yuanchao Xu, Jiexiong Guan, Wei Niu, Shaoshan Liu, Bin Ren, Yanzhi Wang, and Xipeng Shen, "Enabling level-4 autonomous driving on a single $1k off-the-shelf card," 2021.

[12] Dario Kevin Köllner, "Optimization of convolutional neural networks and transformer neural networks using post-training integer quantization," *Technische Hochschule Ingolstadt*, pp. IV, 59, 2023.

[13] Sumaira Manzoor, Eun-Jin Kim, Sung-Hyeon Joo, Sang-Hyeon Bae, Gun-Gyo In, Kyeong-Jin Joo, Jun-Hyeon Choi, and Tae-Yong Kuc, "Edge deployment framework of guardbot for optimized face mask recognition with real-time inference using deep learning," *IEEE Access*, vol. 10, pp. 77898–77921, 2022.

[14] Vidhi Bishnoi and Nidhi Goel, "Tensor-rt-based transfer learning model for lung cancer classification," *Journal of Digital Imaging*, Apr 2023.

[15] Tero Karras and Janne Hellsten, "Flickr-faces-hq dataset," `https://github.com/NVlabs/ffhq-dataset`, Accessed: 2022-02-20.

[16] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[18] François Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR*, vol. abs/1610.02357, 2016.

[19] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, 2015.

[20] Lutz Prechelt, *Early Stopping - But When?*, pp. 55–69, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[21] Alberto Cabrera and Javier Arteaga, "Energy measurement library," 12 2018.

[22] Christophe Pere, "What are loss functions?," `https://towardsdatascience.com/what-is-loss-function-1e2605aeb904`, Accessed: 2022-02-27.

[23] Princeton University Stanford Vision Lab, Stanford University, "Imagenet," `https://www.image-net.org`, Accessed: 2022-02-24.

[24] Shahid Latif, Maha Driss, Wadii Boulila, Zil e Huma, Sajjad Shaukat Jamal, Zeba Idrees, and Jawad Ahmad, "Deep learning for the industrial internet of things (iiot): A comprehensive survey of techniques, implementation frameworks, potential applications, and future directions," *Sensors*, vol. 21, no. 22, 2021.

[25] Ruhul Amin Khalil, Nasir Saeed, Mudassir Masood, Yasaman Moradi Fard, Mohamed-Slim Alouini, and Tareq Y. Al-Naffouri, "Deep learning in the industrial internet of things: Potentials, challenges, and emerging applications," *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11016–11040, 2021.

[26] P. Ambika, "Chapter thirteen - machine learning and deep learning algorithms on the industrial internet of things (iiot)," in *The Digital Twin Paradigm for Smarter Systems and Environments: The Industry Use Cases*, Pethuru Raj and Preetha Evangeline, Eds., vol. 117 of *Advances in Computers*, pp. 321–338. Elsevier, 2020.

[27] Yannis Panagakis, Jean Kossaifi, Grigorios G. Chrysos, James Oldfield, Mihalis A. Nicolaou, Anima Anandkumar, and Stefanos Zafeiriou, "Tensor methods in computer vision and deep learning," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 863–890, 2021.

[28] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015.

Análisis de un acelerador VPU en entornos *Fog* de bajo coste

Víctor Hidalgo[1], Carmen Carrión[2] y Blanca Caminero[2]

Resumen—
El aprendizaje automático ha avanzado enormemente, impulsando a grandes compañías como Google e Intel a desarrollar aceleradores basados en circuitos ASIC. Estos aceleradores se aplican en diversos campos, como mejorar diagnósticos médicos, transcribir textos medievales, predecir el clima extremo y clasificar imágenes. La necesidad de procesar datos complejos en el Internet de las Cosas (IoT) ha impulsado el desarrollo de aceleradores específicos para el borde de la red, como las unidades de procesamiento visual (VPUs). Estos aceleradores permiten ejecutar redes entrenadas cerca de la fuente de datos, reduciendo las latencias y los costes de ancho de banda. En muchos casos, tanto el rendimiento como el consumo de energía son críticos. En este trabajo, se analiza el Neural Compute Stick (NCS) de Intel, un dispositivo basado en VPUs, en un entorno de niebla (fog) con dispositivos económicos. El objetivo es verificar si el rendimiento obtenido y el consumo de energía varían según los modelos de aprendizaje automático empleados.

Palabras clave— **Computación en la niebla, internet de las cosas, aprendizaje automático.**

I. Introducción

LAS técnicas de Inteligencia Artificial (IA) y en concreto las redes neuronales profundas (*Deep Neural Networks*, DNNs) tienen capacidad de aprender y extraer patrones complejos a partir de grandes cantidades de datos, lo que las convierte en herramientas de gran utilidad en campos tales como la visión por computador o el procesamiento del lenguaje natural [1]. Sin embargo, la inferencia o proceso por el que se utiliza un modelo de red neuronal profunda entrenado para realizar predicciones o toma de decisiones sobre nuevos datos de entrada es una tarea intensiva en recursos de cómputo y almacenamiento [2]. La solución tradicional a este problema es transferir todas las operaciones de cálculo a la nube. Pero esta transferencia de las operaciones de inferencia a los grandes centros de datos de la computación cloud no siempre es una solución viable debido a preocupaciones de privacidad, conectividad limitada a Internet o restricciones de tiempo. En este sentido, la computación en la niebla o *Fog computing* surge como una tecnología que busca acercar el procesamiento y la computación de datos al usuario final. Este paradigma ofrece ventajas como menor latencia, mayor privacidad y escalabilidad [3].

En particular, el procesamiento de DNNs en el borde es de especial interés en dominios centrados en el usuario, donde la preservación de la privacidad es fundamental, y en escenarios con conectividad limitada a Internet, como drones, robots y vehículos autónomos [4]. Sin embargo, la implementación de inferencia de DNNs en el borde plantea un gran desafío, debido al conflicto inherente entre la naturaleza intensiva en recursos de las DNNs y los recursos limitados disponibles en los dispositivos de borde. Los dispositivos de borde, con sus capacidades computacionales limitadas y suministro de energía restringido, luchan por cumplir con los exigentes requisitos de recursos de las redes neuronales profundas.

Llevar a cabo la inferencia de DNNs en una arquitectura de la niebla supone un desafío técnico significativo. Así, se desarrollan nuevos modelos DNNs compactos y *frameworks* de desarrollo específicos ligeros tales como Tensorflow Lite [5]. Sin embargo, el uso exclusivo de técnicas de software no puede garantizar una ejecución rápida de las DNNs. Recuérdese que las plataformas hardware de propósito general no están diseñadas específicamente para las DNNs y obtienen una baja eficiencia. Por ello, surgen en el mercado dispositivos aceleradores especializados para realizar inferencias rápidas en el borde, como la TPU (*Tensor Processing Unit*) Coral de Google [6], Jetson Nano de Nvidia [7] y Movidius Neural Computer Stick de Intel [8].

En este trabajo presentamos una evaluación de prestaciones del acelerador Movidius Neural Compute Stick 2 (NCS2) como elemento de inferencia incluido en nodos *Fog* de distintas características. El dispositivo USB NCS2 de Intel proporciona una conexión rápida y sencilla para acelerar DNNs en cualquier plataforma. Está equipado con la unidad de procesamiento de visión (*Visual Processing Unit*, VPU) Intel Movidius Myriad X, que consta de 16 núcleos SHAVE (*Streaming Hybrid Architecture Vector Engine*). Los núcleos SHAVE tienen una arquitectura interna VLIW y están equipados con unidades funcionales SIMD (*Single Instruction Multiple Data*), lo que les permite ejecutar múltiples instrucciones y datos simultáneamente. La VPU admite diferentes precisiones mixtas de forma nativa, incluyendo datos de 32 bits, 16 bits y algunos de 8 bits.

Con el fin de obtener una comprensión más precisa, llevamos a cabo un análisis y una comparación del rendimiento de diversos nodos *Fog*, tanto con el acelerador NCS2 como sin él. Además, utilizamos un medidor de energía para determinar el consumo asociado a cada inferencia. Los experimentos realizados son reproducibles y se pueden ampliar a nuevas plataformas utilizando nuestro proyecto de código abierto en GitHub.

Este trabajo se estructura como sigue: En la Sec-

[1]Escuela Superior de Ingeniería Informática, Universidad de Castilla-La Mancha, e-mail: victor.hidalgo3@alu.uclm.es.
[2]Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, e-mail: {carmen.carrion,mariablanca.caminero}@uclm.es.

ción II se presentan brevemente algunos trabajos relacionados. La Sección III describe el entorno donde se han realizado las pruebas así como los modelos utilizados y las métricas empleadas. La Sección IV presenta los resultados de los experimentos realizados. Finalmente, la Sección V concluye el trabajo y presenta algunas ideas de trabajo futuro.

II. Estado del arte

La irrupción de los aceleradores de IA en el borde ha motivado la aparición de diversos estudios en los últimos años. A continuación se revisan brevemente algunos de los más significativos.

En [9] se analiza el impacto de los entornos de desarrollo de modelos, su pila de software y las optimizaciones implementadas en el rendimiento final. Además, se mide el consumo de energía y el comportamiento de la temperatura de estos dispositivos de borde. Este estudio es muy completo, abordando un amplio abanico de modelos, dispositivos y métricas. Sin embargo, emplea una versión anterior del dispositivo NCS2, objeto de este estudio. Los modelos empleados en [9] debían ser desarrollados en un entorno específico.

Por su parte, en [10] se analiza el rendimiento de diversos dispositivos de computación de borde avanzados para la inferencia de DNNs. Se consideran varias arquitecturas (GPUs, ASICs y FPSoCs[1]), y se evalúan empleando las versiones más actuales de entornos DNN populares y sus herramientas de desarrollo correspondientes. El análisis se centra en las latencias obtenidas al procesar una imagen y no contempla el acelerador NCS2.

El uso de un clúster compuesto de hasta 3 aceleradores NCS2 se explora en [11] con el objeto de acelerar el reconocimiento de caras en escenarios no restringidos, en un contexto de mejora de la seguridad. El estudio se centra en la mitigación de la pérdida de prestaciones del modelo, contemplando también el consumo energético y el empleo de recursos de bajo coste.

Un estudio más reciente [12] realiza una evaluación comparativa del modelo de detección de objetos YOLO (*You Only Look Once*) en tres tipos de aceleradores: NVIDIA Jetson Nano, NVIDIA Jetson Xavier NX y Raspberry Pi 4B con Intel Neural Compute Stick2 (NCS2). La evaluación se centra en el rendimiento obtenido, la exactitud conseguida y la facilidad de adaptación del modelo a los distintos aceleradores, pero no considera el consumo energético.

III. Entornos de pruebas

El entorno de pruebas pretende emular los dispositivos comúnmente presentes en un entorno *fog/edge*. Se consideran los siguientes dispositivos:

- Ordenador con CPU Ryzen 5 3500U (4 núcleos)
- Raspberry Pi 4 Modelo B
- Intel Neural Compute Stick 2 (NCS2)

[1] *Field Programmable Systems-on-Chip*

Estos dispositivos se combinan para formar cuatro escenarios, recogidos en la Figura 1. En los escenarios 1 y 2 (Figuras 1a y 1b) los modelos de IA se ejecutan en el propio ordenador, variando el número de núcleos dedicados a la ejecución del modelo. En los escenarios 3 (Figura 1c) y 4 (Figura 1d) los modelos se ejecutan en el acelerador NCS2 y la diferencia radica en el nodo anfitrión al que se conecta el acelerador y desde el cuál se carga el modelo. En el escenario 3 el nodo anfitrión es el ordenador mientras que en el escenario 4 se emplea una una Raspberry Pi. Se puede comprobar como en los 3 primeros escenarios se emplea un nodo *fog* de coste y prestaciones medias, complementado por el acelerador en el escenario 3. Por contra, en el último escenario se emplea un nodo *fog* de bajo coste complementado con el acelerador.

En todos los casos se emplea el medidor de energía WattsUp? .NET [13] para obtener el consumo energético al realizar las pruebas en cada escenario. De esta manera se busca medir no solo las prestaciones obtenidas al ejecutar los modelos de IA sobre uno u otro hardware, sino también hacerlo en relación a su eficiencia energética.

Hay que comentar que se ha medido el consumo en reposo de los dispositivos para poder descontarlo del consumo total obtenido en cada prueba, y obtener así valores más realistas sobre el incremento de consumo de energía causado por la ejecución de los modelos. Más concretamente, la única diferencia entre la medición del consumo en reposo y la medición durante las pruebas es la ejecución de la aplicación que ejecutará la inferencia y el modelo utilizado. Se evitará tener procesos en segundo plano y se procurará tener las mismas configuraciones tanto en reposo como durante la ejecución, como lo son el brillo de la pantalla, los dispositivos conectados o la batería del dispositivo al máximo.

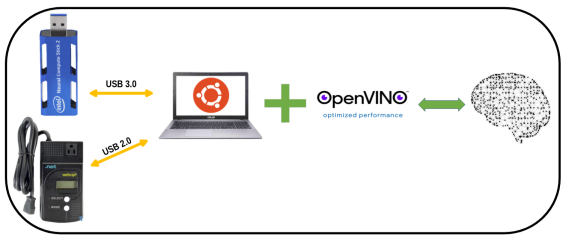
Intel OpenVINO [14] es el software que da soporte al Intel Neural Compute Stick 2. OpenVINO es un conjunto de herramientas de código abierto cuyo propósito principal es el de optimizar y desplegar modelos de *Deep Learning*. De forma más concreta, su objetivo es ofrecer mejores rendimientos para diferentes modelos de visión, audio e idioma. Dichos modelos son resultado del uso de otros *frameworks* muy conocidos para *Deep Learning*, como TensorFlow o PyTorch.

Este software se puede dividir en dos funcionalidades principales:

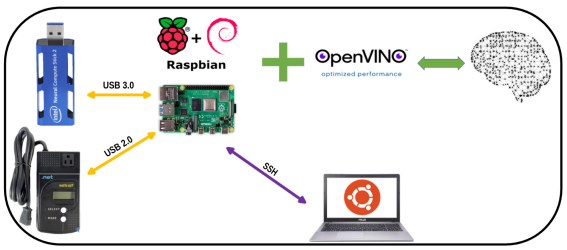
- *OpenVINO Development Tools*: Conjunto de herramientas que facilita el desarrollo y optimización de modelos y aplicaciones para OpenVINO. Dichas herramientas permiten preparar los modelos para que *OpenVINO Runtime* pueda realizar la inferencia sobre ellos.
- *OpenVINO Runtime*: Permite ejecutar la inferencia sobre los modelos. Junto a algunos códigos añadidos (Intel los denomina *plugins*) este módulo es el que ofrece la ejecución de *Deep Learning* sobre un variado número de aceleradores hardware.

(a) Escenario 1: CPU (1 núcleo)

(b) Escenario 2: CPU (4 núcleos)



(c) Escenario 3: CPU + NCS2



(d) Escenario 4: Raspberry Pi 4 + NcS2

Fig. 1: Escenarios de prueba.

Por otro lado en la página web oficial de OpenVINO se proporciona un listado de los modelos disponibles, tanto del tipo *intel* (desarrollados y entrenados por Intel) como *public*. Cada modelo se caracteriza por los siguientes parámetros:

- ***Framework***: Se indica el nombre del *framework* en que fue desarrollado el modelo, como TensorFlow, Caffe o PyTorch, entre otros.
- **Precisión (Porcentaje)**: Se indica el porcentaje de acierto que tiene el modelo en concreto, reflejando su nivel de acierto en las ejecuciones.
- **Complejidad (GFLOPs)**: Valor que indica el número de GFLOPs requeridos para la ejecución del modelo. La complejidad está directamente relacionada con la potencia necesaria para llevar a cabo dicha ejecución.
- **Tamaño (mParams)**: Valor que representa el tamaño que ocupa el modelo, en mParams (*Model Parameters*). Indica la cantidad de parámetros utilizados en la red neuronal, obtenidos durante su entrenamiento. Podemos observar que, con el aumento de este valor, el tamaño que ocupan los modelos en el anfitrión aumenta.

En primer lugar, se van a analizar cada uno de los modelos observando comportamientos comunes a todos ellos, y destacando las anomalías observadas. Seguidamente, y considerando las características mencionadas arriba, se han tratado de seleccionar pares o tríos de modelos que presentan similitudes en todas estas características, a excepción de una de ellas. El objetivo es determinar si la característica diferente afecta en alguna de las métricas obtenidas durante las pruebas.

Con la instalación del *framework* OpenVINO se incluyen aplicaciones de demostración previamente desarrolladas y listas para su compilación y ejecución. Estas aplicaciones permiten probar de manera sencilla el funcionamiento del software y verificar el correcto funcionamiento de los modelos. Entre estas aplicaciones se encuentra `benchmark_app`, una aplicación a la que se le indica mediante parámetros el nombre de un modelo y el tiempo de ejecución. La prueba de rendimiento consistirá en realizar la mayor cantidad de inferencias posibles en el tiempo indicado, obteniendo las siguientes métricas:

- **Tiempo de carga del modelo (ms)**: Este valor indica la cantidad de tiempo, en milisegundos, necesaria para transferir un modelo ubicado en el dispositivo anfitrión al acelerador encargado de la inferencia, hasta que el modelo esté listo para su ejecución. La velocidad de transferencia desde el anfitrión al acelerador, así como el tamaño del modelo, influyen en estos tiempos.
- **Tiempo total de ejecución (ms)**: Esta métrica representa el tiempo total que llevó ejecutar la aplicación `benchmark_app`. Es muy importante tener en cuenta este tiempo, ya que, aunque se haya establecido un tiempo máximo de ejecución para la aplicación, también se considerará el tiempo que se ha tardado en preparar los flujos paralelos de inferencia previos a la ejecución. Aunque esta métrica no se refleja directamente en las gráficas posteriores al análisis, se utiliza para calcular el rendimiento en *frames* por segundos (FPS) del modelo.
- **Latencia (ms)**: La latencia es el tiempo necesario para procesar una única solicitud de inferencia. Esta métrica se ve afectada por la cantidad de flujos de procesamiento paralelo. Una menor cantidad de flujos de procesamiento paralelo suele resultar en una mejor latencia, siendo recomendable utilizar un único flujo en la mayoría de los casos para aprovechar al máximo los recursos hardware y reducir el tiempo de inferencia.
- **Rendimiento (FPS)**: El rendimiento se refiere a la capacidad del flujo de inferencia para procesar datos simultáneamente, y generalmente se suele medir en FPS. Esta métrica se ve afectada por la cantidad de flujos de procesamiento

paralelo empleados. A diferencia de la latencia, en la que se busca minimizar el tiempo de procesamiento de una única solicitud de inferencia, para lograr un mayor rendimiento en la ejecución de tareas de inferencia de DNNs, se busca aumentar el número de flujos de procesamiento paralelo. El objetivo es aprovechar al máximo los recursos del dispositivo que realiza la inferencia, tratando de evitar tiempos de ejecución ociosos. Una mayor cantidad de FPS indica un mejor desempeño del modelo, siendo esta métrica la cantidad de inferencias que se han realizado en un segundo.

Además, se ha elaborado una métrica propia, la **eficiencia**, que permite relacionar el rendimiento con el consumo energético. Así, da una idea de cuantos watios se emplean por inferencia realizada.

IV. Resultados experimentales

En esta sección se resumen algunos de los resultados obtenidos al ejecutar los modelos de IA sobre los cuatro escenarios descritos anteriormente. En [15] se puede consultar detalles adicionales sobre todos los análisis realizados. Además, todos los resultados, así como los diversos *scripts* empleados para su obtención, pueden consultarse en `https://github.com/VictorHidalgoUCLM/NeuralComputeStick2`.

En todos los resultados que se muestran a continuación, cada una de los experimentos se han ejecutado varias veces, obteniendo resultados prácticamente idénticos.

A. Comportamiento general de los modelos

La Figura 2 muestra los resultados del modelo `face-detection-retail-0004` en términos de eficiencia, consumo (vatios) y rendimiento (FPS). Los resultados obtenidos para este modelo siguen las mismas tendencias que el resto de modelos analizados.

En relación al consumo (Figura 2a) puede apreciarse como el escenario 2 es el que mayor consumo ha registrado durante la prueba, mientras que el escenario 4 ha sido el de menor consumo. Esto era esperable, dadas las características hardware de ambos entornos. Sin embargo, el rendimiento obtenido en los escenarios 3 y 4, que incluyen el acelerador NCS2, es similar al obtenido en el mejor caso (el obtenido por el escenario 4) tal como refleja la Figura 2b). Ambas métricas se combinan en la Figura 2c, que muestra la eficiencia obtenida al ejecutar el modelo en los distintos escenarios. Puede apreciarse claramente como los escenarios 3 y 4 (con acelerador) son un orden de magnitud más eficientes que los escenarios 1 y 2 (sin acelerador), es decir, son capaces de realizar un orden de magnitud más de inferencias por watio consumido.

B. Comparativas entre modelos

Se han realizado comparativas entre varias parejas o tríos de modelos similares, que se diferencian en uno solo de los 4 parámetros que los definen. Mostraremos aquí las dos comparativas que han arrojado resultados más significativos: por complejidad y por tamaño de modelo.

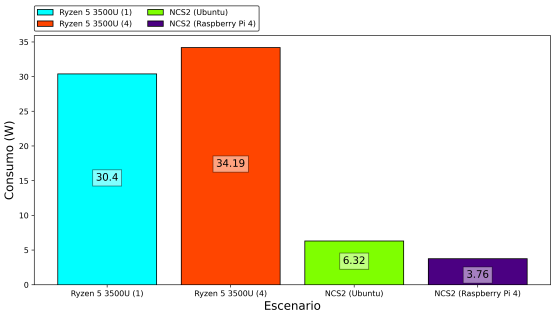
B.1 Comparativa 1: Complejidad del modelo (GFLOPS)

Recordemos que la complejidad de un modelo se mide en GFLOPs y hace referencia a la potencia de cálculo necesaria para ejecutarlo. En esta comparativa se emplean los tres modelos cuyas características se resumen en la Tabla I. Todos los modelos utilizados están clasificados dentro del grupo de procesamiento de imágenes. Los tres modelos comparados tienen las mismas caracteristicas, exceptuando sus requerimientos computacionales.

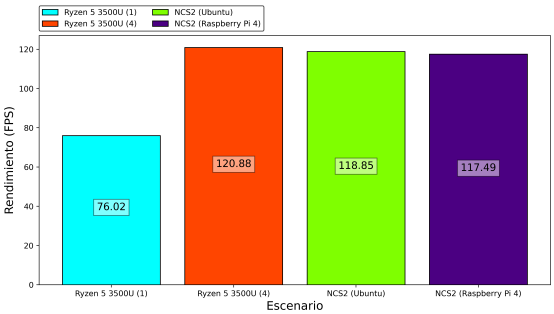
La Figura 3 muestra los resultados en cuanto a consumo, rendimiento y eficiencia para los tres modelos seleccionados. En cada escenario, la variación de consumo energético es poco significativa y no presenta un comportamiento lineal respecto a la complejidad del modelo. En cuanto al rendimiento, destaca el resultado obtenido por el modelo computacionalmente más simple en los escenarios 1 y 2, donde la CPU del ordenador ejecuta la inferencia. En estos escenarios, se muestra una tendencia a un mayor rendimiento en los modelos más sencillos y uno menor en los más complejos. Sin embargo, en los escenarios donde el sistema que realiza la inferencia es el NCS2, observamos un comportamiento muy similar entre los tres modelos. Los altos rendimientos obtenidos en los dos primeros escenarios pueden ser resultado de un mejor aprovechamiento de la arquitectura por parte del modelo `text-image-super-resolution-0001`, ya que no es un comportamiento que se reproduzca en todos los escenarios.

Por su parte la Figura 4 muestra las latencias y el tiempo de carga obtenidos. Las latencias son opuestas al rendimiento en FPS, donde un mayor valor en las latencias supone un menor rendimiento. Los tiempos de carga son muy ilustrativos en esta comparativa, ya que podemos observar en los escenarios 3 y 4, que incluyen el NCS2, que las cantidades de tiempo son coherentes con la complejidad computacional del modelo. Más concretamente, la carga del modelo `single-image-super-resolution-1033` es hasta 5.68 veces la del modelo `text-image-super-resolution-0001`, y hasta 3.41 veces la del modelo `single-image-super-resolution-1032`, siendo estas diferencias más notorias en el cuarto escenario (Raspberry Pi 4 + NCS2). Es importante destacar que en estas comparativas, aunque haya una tendencia al aumento del tiempo de carga con el aumento de la complejidad, no se ha observado el mismo factor de crecimiento en ambas métricas.

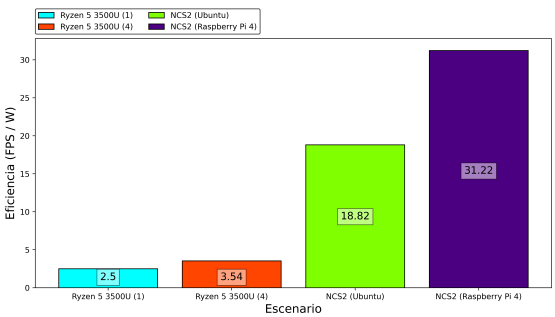
En resumen, los resultados indican que en los escenarios con el NCS2, el rendimiento no se ve afectado en gran medida por la complejidad de los modelos, obteniendo resultados muy similares, mientras que los tiempos de carga se ven muy afectados.



(a) Consumo



(b) Rendimiento



(c) Eficiencia

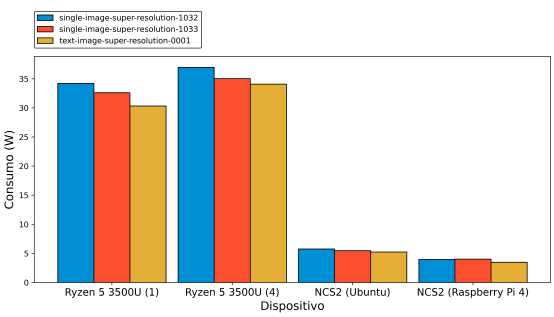
Fig. 2: Resumen de métricas para el modelo `face-detection-retail-0004`

Modelo	*Framework*	GFLOPs	*mParams*
`single-image-super-resolution-1032`	PyTorch	11.654	0.030
`single-image-super-resolution-1033`	PyTorch	16.062	0.030
`text-image-super-resolution-0001`	PyTorch	1.379	0.030

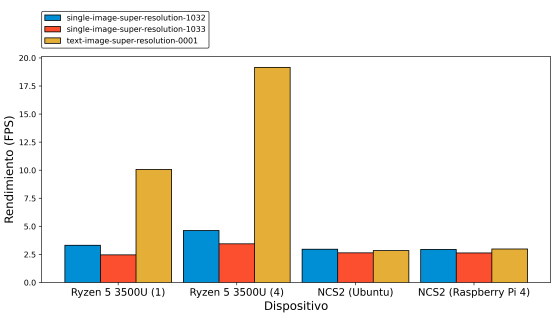
Tabla I: Comparativa 1: complejidad del modelo

Fig. 3: Comparativa 1: consumo, rendimiento y eficiencia según la complejidad del modelo

(a) Consumo



(b) Rendimiento



(c) Eficiencia

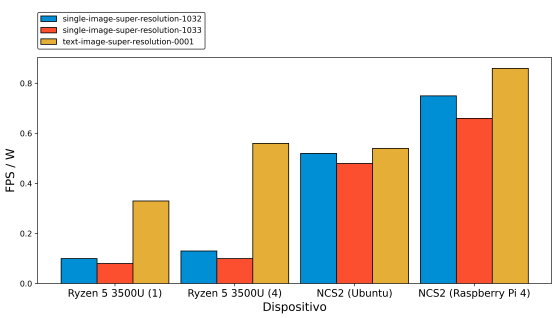
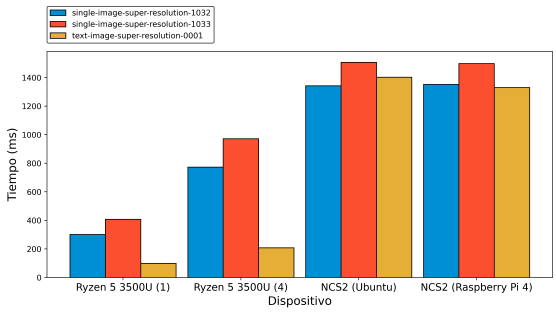
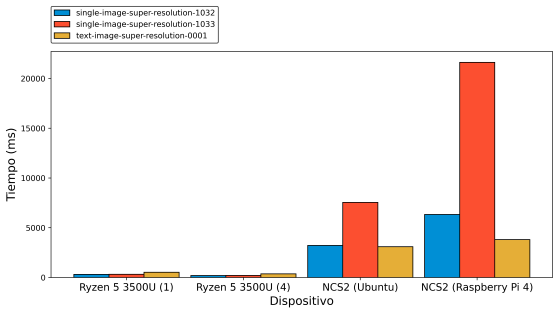


Fig. 4: Comparativa 1: latencia y carga según la complejidad del modelo

(a) Latencias



(b) Cargas

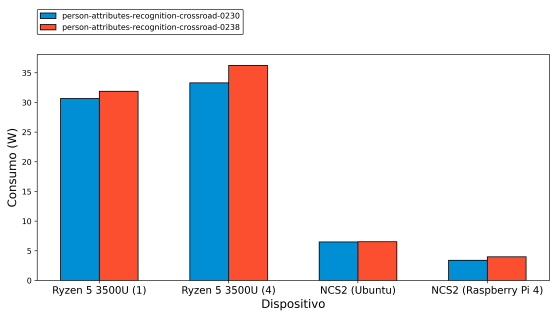


Modelo	*Framework*	GFLOPs	*mParams*
`person-attributes-recognition-crossroad-0230`	PyTorch	0.174	0.735
`person-attributes-recognition-crossroad-0238`	PyTorch	1.034	21.797

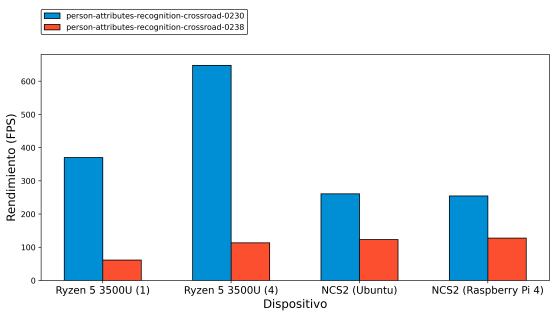
Tabla II: Comparativa 2: tamaño del modelo

Fig. 5: Comparativa 2: consumo, rendimiento y eficiencia según el tamaño del modelo

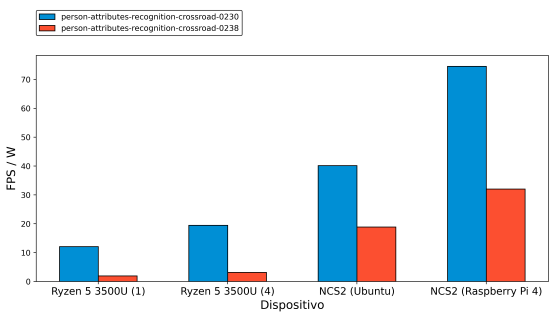
(a) Consumo



(b) Rendimiento



(c) Eficiencia



B.2 Comparativa 2: Tamaño del modelo (*mParams*)

Esta comparativa se centrará en la ejecución de modelos cuya diferencia principal radica en el tamaño que ocupan, expresado en términos de *mParams* (ver Tabla II). Esta prueba contrastará dos modelos algo dispares en cuanto a GFLOPs, pero con una diferencia significativa en cuanto a su tamaño (un modelo es casi 30 veces mayor que el otro).

Aunque estos modelos son dispares tanto en complejidad como en tamaño, en la comparativa anterior hemos determinado que la complejidad de un modelo para los escenarios con el NCS2 no afecta significativamente en el rendimiento obtenido en dichos escenarios. No podemos decir lo mismo para el rendimiento obtenido en los dos primeros escenarios ni en los tiempos de carga de ningún escenario, motivo de que no analicemos ninguno de ellos.

El consumo (ver Figura 5), al igual que en el resto de comparativas, es muy similar para ambos modelos en cada escenario. Sin embargo, en los escenarios 3 y 4 (los que utilizan el NCS2 para ejecutar inferencia) el rendimiento llega a ser el doble en el modelo `person-attributes-recognition-crossroad-0230` respecto al modelo `person-attributes-recognition-crossroad-0238` en ambos escenarios.

Estos resultados nos indican que, al menos en los escenarios con el NCS2, el rendimiento alcanzado por los modelos se ve afectado en gran medida por la can-

tidad de *mParams*. Un alto número de parámetros se traduce en un menor rendimiento, aunque no en la misma medida.

V. Conclusiones

En este trabajo se han desplegado aplicaciones de inteligencia artificial sobre dispositivos que forman parte de una infraestructura de cálculo desplegada en el borde de la red. Más concretamente, con la premisa de ejecutar modelos de Deep Learning en la capa de red más cercana al usuario, Intel ha desarrollado el *framework* OpenVINO, ofreciendo así una plataforma software que hace posible este tipo de funcionalidades en el borde. Se ha analizado en profundidad el funcionamiento de esta plataforma junto al acelerador de visión Neural Compute Stick 2 mediante una serie de pruebas llevadas a cabo en entornos monitorizados, integrando dispositivos aceleradores y de monitoreo, así como el software necesario para el correcto funcionamiento de los sistemas.

En la mayoría de las pruebas realizadas, el escenario compuesto por una Raspberry Pi 4 ejerciendo de dispositivo anfitrión para el Neural Compute Stick 2 ha resultado ser uno de los más eficientes, encontrando un equilibrio entre el rendimiento (FPS) y el consumo energético (vatios). Además, hay que destacar que hacer uso de este escenario como nodo de cómputo en una arquitectura de computación en la niebla es una opción de bajo coste (menor de 300 €) capaz de proporcionar inteligencia artificial en el borde de la red.

Los horizontes de estas tecnologías conjuntas y sus posibles aplicaciones aún se encuentran en proceso de exploración. Sin embargo, ya se ha observado una gran aparición en distintos proyectos debido a las notables mejoras tanto en rendimiento como en eficiencia energética. Con el acercamiento de la IA a los usuarios, se evita la necesidad de recurrir a grandes computadoras o, en su defecto, al uso excesivo de la red o entornos Cloud.

Como ideas para trabajo futuro, comentar que en versiones más recientes del entorno OpenVINO existe la posibilidad de utilizar varios aceleradores Neural Compute Stick 2 con un mismo anfitrión. Esto permitiría disponer de varios de estos dispositivos para poder realizar las tareas de inferencia de forma paralela. Surge también el reto de gestionar ese conjunto de aceleradores de manera inteligente, para intentar asignarlos a las cargas de trabajo que lo requieran, en un contexto que incluya múltiples aplicaciones de IA ejecutándose de manera simultánea en los recursos del *Fog* o del *Edge*.

Agradecimientos

Este trabajo ha sido financiado por el proyecto PID2021-123627OB-C52, con fondos de MCIN/AEI/10.13039/501100011033 y de European Regional Development Fund (ERDF) "A way to make Europe", y por la Ayuda a Grupos Consolidados 2023-GRIN-34056 de la Universidad de Castilla-La Mancha.

Referencias

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[2] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[3] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, New York, NY, USA, 2012, MCC '12, p. 13–16, Association for Computing Machinery.

[4] W. Su, L. Li, F. Liu, et al., "AI on the edge: a comprehensive review," *Artif Intell Rev*, vol. 55, pp. 6125–6183, 2022.

[5] "Tensorflow lite," `https://www.tensorflow.org/lite`, último acceso en junio de 2023.

[6] Google LLC, "Coral: Build beneficial and privacy preserving AI," `https://coral.ai/`, último acceso en junio de 2023.

[7] NVIDIA Corporation, "NVIDIA Jetson Nano," `https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano/`, último acceso en junio de 2023.

[8] Intel Corporation, "Inteligencia visual mejorada en el borde de la red," `https://www.intel.es/content/www/es/es/products/docs/processors/movidius-vpu/myriad-x-product-brief.html`, último acceso en junio de 2023.

[9] Ramyad Hadidi, Jiashen Cao, Yilun Xie, Bahar Asgari, Tushar Krishna, and Hyesoon Kim, "Characterizing the deployment of deep neural networks on commercial edge devices," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 2019, pp. 35–48.

[10] Rancaño, Xalo and Molanes, Roberto Fernández and González-Val, Carlos and Rodríguez-Andina, Juan J. and Fariña, José, "Performance Evaluation of State-of-the-Art Edge Computing Devices for DNN Inference," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, 2020, pp. 2286–2291.

[11] Juan Mas, Teodoro Panadero, Guillermo Botella, Alberto A. Del Barrio, and Carlos García, "CNN Inference acceleration using low-power devices for human monitoring and security scenarios," *Computers & Electrical Engineering*, vol. 88, pp. 106859, 2020.

[12] Haogang Feng, Gaoze Mu, Shida Zhong, Peichang Zhang, and Tao Yuan, "Benchmark analysis of yolo performance on edge intelligence devices," *Cryptography*, vol. 6, no. 2, 2022.

[13] C. Ramseyer, "Watts Up? Pro, Pro ES, and .Net Power Meter Review," `https://www.tweaktown.com/reviews/5947/watts-up-pro-pro-es-and-net-power-meter-review/index.html`, último acceso en mayo de 2023.

[14] Intel Corporation, "OpenVINO Quickstart Guide," `https://docs.openvino.ai/latest/ovms_docs_quick_start_guide.html#doxid-ovms-docs-quick-start-guide`, último acceso en junio de 2023.

[15] Víctor Hidalgo Izquierdo, "Evaluación del Intel Neural Stick en entornos Fog de bajo coste," Trabajo Fin de Grado de Ingeniería Informática, E.S. de Ing. Informática, Universidad de Castilla-La Mancha, Junio, 2023.

Servicio web basado en dockers para ayuda al diagnóstico de cáncer de próstata con técnicas de deep-learning

Antonio Pérez-Peña[1], Luis Muñoz-Saavedra[1], José Manuel Marrón-Esquivel[1], Lourdes Durán-López[1], Juan P. Domínguez Morales[1], Saturnino Vicente-Díaz[1], Alejandro Linares-Barranco[1]

Resumen— **La inteligencia artificial y el aprendizaje profundo están cada vez más desarrollados y mejor integrados en el entorno médico para facilitar y ayudar en las tareas más rutinarias y que mayor tiempo consumen a los doctores. La clasificación de imágenes con redes neuronales convolucionales han demostrado un alto grado de madurez y tasas de efectividad muy elevadas. En este artículo presentamos una solución para que los doctores puedan utilizar estas técnicas de una forma eficiente y sencilla. Centrándonos en la detección del cáncer de próstata clasificando tejido obtenido por biopsia como cancerígeno o no, y ofreciendo el estadío de la enfermedad de acuerdo a la escala de Gleason, el trabajo se centra en la creación de un servicio basado en web que ejecuta el algoritmo solicitado en una máquina específicamente configurada para minimizar los tiempos de ejecución. Esta máquina usa contenedores para facilitar la réplica de la solución en nuevas máquinas o réplicas de esta, y utiliza un gestor de base de datos Django para la gestión de usuarios, imágenes a analizar y diagnósticos obtenidos, de forma que los usuarios puedan consultar estas soluciones cuando lo requieran. Estos resultados se muestran como mapas de calor sobreimpresos en la imagen original utilizando la herramienta gratuita y abierta QuPath. Nuestra solución, la cual incluye además la ejecución en un sistema hardware específicamente diseñado para optimizar la ejecución del sistema, puede ser usada no sólo por experimentados médicos patólogos, sino también por médicos noveles que están en proceso de aprendizaje de esta materia.**

Palabras clave—**Servitización, Deep Learning, cáncer de próstata, patología computacional.**

I. Introducción

El cáncer de próstata es uno de los tipos de cáncer con mayor porcentaje de detección en la actualidad, llegando a ser el segundo tipo de cáncer más detectado en hombres y la quinta causa de muerte a nivel mundial [1].

Uno de los condicionantes más importantes a la hora de determinar el grado de supervivencia a la enfermedad es una detección temprana de la misma, permitiendo así aplicar los tratamientos mas adecuados para cada caso. Para ello, contamos con una serie de métodos con los que tomar muestras del paciente de formas no invasivas, como la tomografía computerizada o la resonancia magnética [2] y, de formas más invasivas, como la extracción de muestras de tejidos mediante biopsias. Este último método nos permite conocer de manera más detallada la morfología celular del tejido extraído, haciendo posible digitalizar dichas muestras mediante escáneres especializados para, posteriormente, poder trabajar sobre la muestra con diferentes métodos computacionales.

Las diferentes pruebas de detección comentadas anteriormente necesitan ser revisadas y analizadas por un conjunto de profesionales, con el fin de detectar posibles indicios de células cancerígenas en el tejido. Esta tarea conlleva en muchos casos, una gran carga de trabajo y de tiempo para el personal dedicado al análisis de dichas muestras, ralentizando en algunos casos el estudio de pacientes y derivando en algunas ocasiones en una detección tardía de la patología. Por ello, en estos últimos años, el desarrollo de herramienta para el análisis y la detección de patologías ha ido en aumento, buscando una solución para ofrecer el apoyo necesario a la hora de analizar y detectar patologías en diferentes muestras histopatológicas.

Un enfoque que está demostrando una gran tasa de acierto y mejora es el uso de algoritmos de inteligencia artificial enfocados en la detección y clasificación de imágenes histopatológicas [3], [4], [5], [6], llegando a dar resultados de detección del cáncer de próstata con un porcentaje de acierto del 99.98 % al hacer uso de la red neuronal PROMETEO [7]. Con estos datos, podemos ver la ventaja que aportan este tipo de tecnologías en el campo de la patología, tanto en el ámbito educativo sirviendo de herramienta para estudiantes y patólogos, como en el ámbito profesional ayudando a agilizar el análisis de muestras médicas y sirviendo de mecanismo de decisión a la hora de diagnosticar aquellos casos que generen ciertas dudas en los profesionales.

Centrándonos en las redes neuronales como una herramienta de apoyo a los estudiantes y especialistas en patología, podemos decir que los análisis realizados por estos últimos están expuestos al sesgo propio de cada patólogo, lo que conlleva un cierto nivel de discrepancia en la decisión tomada a la hora de diagnosticar un caso analizado por varios patólogos. En algunos casos, el nivel de discrepancia puede ser un tanto elevado, por ejemplo, para un conjunto de muestras de cáncer de próstata de diferentes grados de Gleason, se llega a una discrepancia del 47 % en la selección del área anotada como zona con presencia de células cancerígenas [8]. Esta discrepancia se reduce si utilizamos una red entrenada adecuadamente donde, no sólo mejora la selección del área afectada,

[1]Dpto. de Arquitectura y Tecnología de Computadores, Universidad de Sevilla, e-mail: antperpen@alum.us.es

sino que mejora también el acierto a la hora de determinar el grado de Gleason presente en la muestra [8]. Con este tipo de herramientas los doctores que están aprendiendo a diagnosticar, reducirán ese sesgo indicado.

Como ya se ha mencionado anteriormente, los modelos de análisis de imágenes histopatológicas han demostrado ser de gran ayuda para patólogos y estudiantes de patología, pero la mayoría de las veces hacer uso de las herramientas que incluyen algoritmos de detección de células cancerígenas no es una tarea fácil, ya que es necesario un conocimiento avanzado del uso de las tecnologías para instalar y poner en funcionamiento dichos algoritmos. Por este motivo, es necesario el desarrollo de herramientas que extiendan y faciliten el uso de dichas tecnologías para contribuir en el análisis de diferentes tipos de patologías, como es el caso de las aplicación web desarrollada en este trabajo.

La aplicación web desarrollada propuesta en este artículo se centra en permitir el análisis de imágenes histopatológicas de biopsias de próstata, con el objetivo de ofrecer al usuario un mapa de calor con las diferentes regiones celulares con presencia de tejido tumoral y ofreciendo su clasificación en escala de Gleason. Todo esto, buscando un alto nivel de eficiencia y escalabilidad de la propia aplicación, utilizando contenedores basados en la tecnología de dockers para reforzar la portabilidad y aplicando avanzadas tecnologías de desarrollo web para facilitar uso de la misma.

Las principales contribuciones de este trabajo son las siguientes:

- Extender el análisis histopatológico mediante inteligencia artificial.
- Una herramienta fácilmente escalable.
- Usable por patólogos para tareas rutinarias o por aprendices en ámbito universitario o de residencia.

El resto del paper se estructura siguiendo la siguiente organización en secciones: en la sección I encontramos la introducción a este trabajo donde se presenta el problema a tratar y la solución dada. En la sección II se revisa la información de los algoritmos de inteligencia artificial que se emplean para el análisis de las imágenes de tejido. En la sección III se analiza la arquitectura de la aplicación desarrollada, así como el flujo de datos de dicha aplicación. En la sección IV se realizan diversas pruebas sobre la aplicación desarrollada y se analiza las características del servidor empleado. Finalmente, en la sección V se revisan las conclusiones obtenidas con el desarrollo de este trabajo.

II. Deep Learning como ayuda al diagnóstico

Las muestras de tejido obtenido tras una biopsia son procesados en un laboratorio mediante una técnica llamada tinción Hematoxilina-Eosina (H&E), la cual permite mejorar el contraste y resaltar algunas estructuras celulares del tejido [9]. Este proceso de tinción no está estandarizado, por lo que varía entre preparaciones. Esto, sumado al hecho de que cada escáner usado para digitalizar las muestras influye en aspectos como el color, saturación y luminosidad de la imagen, hacen que todo el proceso dé lugar a una gran heterogeneidad entre muestras escaneadas. Esta variabilidad entre las muestras, sumado al hecho de la limitada cantidad de datos públicos disponibles, hacen que desarrollar un sistema inteligente universal para detectar malignidades en imágenes histológicas sea una tarea aún por resolver.

En el proceso de análisis actual, las muestras de tejido que han sido procesadas y digitalizadas dan lugar a imágenes de muy alta resolución, que se denominarán Whole-Slide Images (WSIs) o Tissue Micro-Arrays (TMAs) según cómo fueron adquiridas [10], [11]. Estas imágenes son analizadas por patólogos, que evalúan el tejido y asignan una puntuación de la escala de Gleason [12] en función de los dos patrones de Gleason con mayor presencia en la imagen. Mientras más alta sea la agresividad del cáncer, mayor será el valor del patrón de Gleason identificado (el cual tiene un rango entre 1 y 5). Los dos patrones más predominantes se suman, dando lugar al valor de la escala de Gleason de la imagen completa. Son varios los estudios que han reportado la gran variabilidad interobservadora entre los expertos patólogos a la hora de asignar el valor de la escala correspondiente [13], [14], [15], [8]. Es por ello que el auge de la inteligencia artificial y el Deep Learning han sido de interés para minimizar esta variabilidad y servir como segunda opinión a la hora de tomar una decisión en el análisis.

Las WSIs digitalizadas tienen una muy alta resolución (alrededor de 200k $\times$ 200k píxeles por imagen). Con el hardware actual se hace imposible trabajar con imágenes de tan alto tamaño como entrada a una red neuronal de convolución. Es por ello que se hace necesario particionar las imágenes en subimágenes de menor tamaño, comúnmente denominadas *patches*, que sí pueden ser procesados por las redes neuronales.

De cara a entrenar un sistema basado en redes neuronales de convolución para el propósito de detectar tejido benigno o maligno y su correspondiente patrón de Gleason, diferentes métodos de entrenamiento pueden ser aplicados. En este caso, nos centraremos en métodos de entrenamiento supervisados, que son los más presentes en la literatura. Para poder entrenar una red neuronal por este método, es necesario que las imágenes utilizadas estén etiquetadas. En nuestro caso, esto correspondería a que cada *patch* tenga una clase asignada (benigno, patrón de Gleason 3, 4 ó 5). Para ello un equipo de patólogos ha tenido que, previamente, anotar de forma específica cada una de las WSIs a considerar en el estudio, indicando qué regiones del tejido corresponden a cada una de las clases designadas, lo cual es muy laborioso y costoso en tiempo. Debido a esto, son pocos los datasets públicos que hay en los que se haya etique-

tado cada imagen de forma específica, motivo por el cual se está tendiendo a estudiar otros métodos de entrenamiento que no dependan de ellas.

En nuestro caso, partimos de tres datasets privados obtenidos de los hospitales Virgen de Valme (Sevilla, España), Clínic (Barcelona, España) y Puerta del Mar (Cádiz, España), los cuales han sido etiquetados por patólogos de dichos centros. En total, se han obtenido 938 WSIs de Virgen de Valme provenientes de 199 pacientes distintos, 221 WSIs de Clínic provenientes de 43 pacientes distintos, y 144 WSIs de Puerta del Mar de 18 pacientes distintos. Cabe destacar que, debido a los motivos comentados con anterioridad, no todas ellas fueron etiquetadas por los patólogos. En particular, se obtuvieron 70 WSIs etiquetadas de Virgen de Valme y 80 de Clínic. Para el resto, aquellas correspondientes a pacientes sanos se usaron para obtener *patches* benignos. En total, se obtuvieron alrededor de 30000 *patches* de Virgen de Valme, 17600 de Clínic y otro volumen similar de *patches* benignos de Puerta del Mar, de tamaño 750×750 píxeles a 40× de aumento.

Con esto, se realizaron dos tareas. Por un lado, se diseñó, entrenó y validó una red para diferenciar entre *patches* malignos y benignos. Dicha red fue construida teniendo en cuenta la optimización en tiempo de respuesta, por lo que el diseño fue propio y contenía un total de 9 capas de procesamiento. Los resultados obtenidos de este sistema pueden verse en detalle en los trabajos publicados por Durán-López et al. [7], [16]. Por otro lado, se entrenó y evaluó un sistema basado en DenseNet121 [17] para la clasificación de *patches* entre patrones de Gleason 3, 4 y 5, los cuales son los correspondientes a tejido maligno. Los resultados obtenidos de este sistema pueden verse en detalle en el trabajo publicado por Marrón-Esquivel et al. [8].

Cada uno de los algoritmos anteriormente mencionados es utilizado para procesar de forma automática WSIs completas, de las cuales primero se extraen densamente (sin solapamiento) todos los *patches* de tejido que contienen. Estos *patches* son usados como entrada a los modelos, los cuales generan un mapa de calor, indicando aquellas zonas malignas de la imagen y el patrón de Gleason detectado en cada zona maligna de tejido. Este mapa de calor se genera en formato CSV, el cual puede ser visualizado en el entorno que se desee.

En nuestro caso, hacemos uso de QuPath [18] para representar dichos mapas de calor. QuPath es una herramienta open-source y completamente gratuita que permite abrir imágenes histológicas de gran tamaño y navegar por ellas, además de realizar cierto tipo de procesamiento sobre ellas. A su vez, dispone de un sistema de anotaciones que pueden tanto ser creadas y exportadas (herramienta usada por los patólogos de Clínic para reportarnos las zonas etiquetadas de las WSIs malignas) como representadas. En este sentido, QuPath dispone de un editor y lanzador de *scripts* en lenguaje de programación Groovy, con el que se ha desarrollado un fragmento de código que permite cargar los mapas de calor en CSV generados por los modelos entrenados y representarlos encima de la imagen WSI correspondiente. De esta forma, se tiene una visual completa de la imagen con una capa por encima que puede ser activada/desactivada y cuya opacida puede ser modificada, de cara a ver el resultado de los algoritmos desarrollados.

III. Servitización

Para llevar a cabo el desarrollo de la aplicación, se ha tenido muy en cuenta el enfoque que esta debía tener, orientándose hacia una funcionalidad ágil y simplificada. En primer lugar, el usuario debe tener su propio perfil de trabajo. En segundo lugar, se requiere de una sección donde el usuario pudiese tener una serie de imágenes histopatológicas asociadas a su perfil e ir trabajando con estas. Además, se ofrece la posibilidad de aplicar diferentes algoritmos dependiendo del resultado deseado a la hora de realizar el análisis. En la subsección III-A se describen las diferentes tecnologías utilizadas para el desarrollo del sistema. Por otro lado, en la subsección III-B, se detalla el flujo de funcionamiento del sistema completo en base a las tecnologías presentadas en el anterior apartado.

A. Tecnologías empleadas

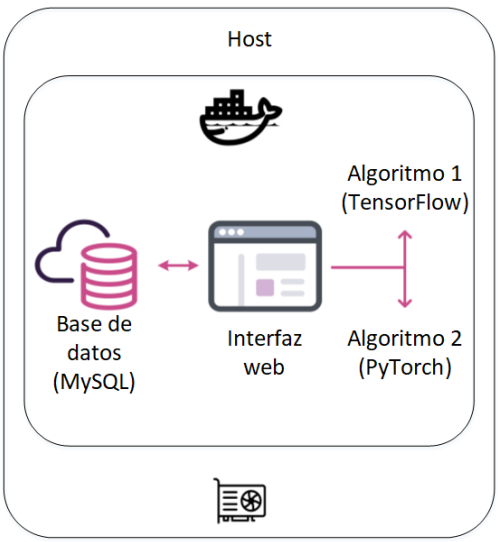


Fig. 1: Esquema general del servicio PROMETEO WEB.

La tecnología utilizada en el desarrollo de la aplicación web se divide en el frontend y el backend. En el primero, se ha utilizado el framework de Django para ofrecer un funcionamiento simple y dinámico y, en el segundo, se ha realizado un desarrollo basado en la tecnología de contenedores para favorecer la portabilidad y escalabilidad de la aplicación desarrollada. Esto nos permite mover la aplicación de una máquina a otra sin tener en cuenta la dependencias de librerías instaladas, permitiendo así un despliegue rápido del sistema completo en nuevos equipos para nuevos centros (véase la figura 1).

Por otro lado, el desarrollo basado en el uso de contenedores implica añadir una capa de abstracción entre la máquina y todo el sistema que compone la

aplicación, dificultando la comunicación entre la aplicación web que ejecuta los diferentes algoritmos de IA para analizar imágenes y los dispositivos hardware, como la tarjeta gráfica, que permiten una aceleración en la ejecución de los algoritmos implementados. Por este motivo, se ha basado el desarrollo sobre contenedores optimizados para favorecer la comunicación entre la aplicación y el hardware del servidor, favoreciendo una ejecución eficiente de la aplicación web.

El desarrollo basado en contenedores aporta otra ventaja muy importante si se compara con el desarrollo de aplicaciones tradicionales. Se trata de la escalabilidad de la aplicación. Este concepto hace referencia a la posibilidad de aumentar la capacidad de procesamiento de la aplicación web en el caso de una demanda creciente, evitando la posible saturación del sistema. Otra de las ventajas con la que cuenta la aplicación desarrollada, es la posibilidad de ejecutarse en servicios de contenedores en la nube, aumentando la capacidad de cómputo y, con ello, acelerando la ejecución de los algoritmos de análisis incluidos en la aplicación. Por este motivo, el desarrollo basado en contenedores se ha realizado utilizando ficheros que automatizan el arranque y la instalación de todas las dependencias, separando la imagen utilizada para crear el contenedor y las aplicaciones que se ejecutan en este.

Al usar la tecnología de contenedores, nos encontramos con que aportan una serie de ventajas como las que se han comentado anteriormente. Pero también su uso conlleva una serie de desventajas, como la pérdida de todos los datos presentes en el contenedor si este se detiene. Por este motivo, el desarrollo de la aplicación web se ha enfocado en mantener en el servidor todos los archivos que necesitan persistencia, como la base de datos de los usuarios, las imágenes histopatológicas, los resultados de los análisis, etc. Con esta medida mantenemos todos los datos guardados en el servidor, evitando que se produzcan perdidas. Sin embargo, con ello surge el problema de la sincronización entre los datos guardados en el servidor y los datos que se encuentran en el contenedor. Para evitar este problema se han empleado los volúmenes, una tecnología que permite tener una jerarquía de ficheros y carpetas compartidas dinámicamente entre el contenedor y el servidor, sincronizando cualquier cambio que se realice en los datos.

B. Flujo de funcionamiento del sistema

Una vez hemos visto la parte más técnica del desarrollo, podemos centrarnos en el funcionamiento a nivel de usuario de la aplicación PROMETEO WEB (véase la figura 2). Comenzamos con un usuario que accede a la aplicación web, el cual se debe registrarse o logarse dependiendo si es su primer acceso al sistema. Después de esto, el usuario tiene la posibilidad de subir una imagen nueva a analizar o acceder al apartado de sus imágenes para consultar el resultado de los análisis realizados, pudiendo relanzar el análisis con otro algoritmo si necesita un segundo resultado, o descargar el mapa de calor resultante para cargarlo en el visor de imágenes histopatológicas que prefiera el usuario.

Al subir una imagen nueva, al usuario se le da la opción de seleccionar con qué algoritmo se va a realizar el análisis 3, pudiendo elegir un algoritmo rápido y que ofrece un mapa con dos clases (indicando si hay o no tejido cancerígeno y a qué zonas afecta), pero con un porcentaje de precisión algo menor; o un algoritmo con un tiempo de inferencia más elevado y que ofrece un mapa con los patrones de Gleason identificados, pero con un porcentaje de acierto superior al comentado anteriormente. Al ofrecer dos opciones de análisis diferentes, se busca dar versatilidad a la aplicación, aportando una opción más ligera para una detección rápida y otra opción algo más pesada para análisis en mayor profundidad.

Una vez se han llevado a cabo los diferentes análisis, la aplicación ofrece una lista de las imágenes analizadas y si ha encontrado o no tejido cancerígeno en la muestra, permitiendo al usuario acceder a cada imagen histopatológica por separado y habilitando la descarga del mapa de calor resultante del análisis (en formato CSV). Con este mapa de calor, el usuario puede localizar las áreas del tejido donde el algoritmo ha localizado la presencia de células cancerígenas, aportando una información de gran valía para un patólogo en su trabajo diario (o a un estudiante patólogo).

Para poder visualizar el mapa de calor comentado anteriormente, es necesario contar con un visor preparado para la carga de imágenes histopatológicas y su respectivo mapa de calor para delimitar las áreas afectadas. Esta no es una tarea sencilla y la mayoría de los visores presentes en la actualidad son privados y con un alto coste económico. Es por este motivo que junto con el resultado del análisis se aporta un script de la herramienta gratuita open-source QuPath[1] [18], con el cual podemos seleccionar de manera automática el mapa de calor y la imagen analizada 4. Esto permite al usuario estudiar de manera cómoda y simplificada los resultados de los análisis realizados en la aplicación PROMETEO WEB, visualizando la imagen y el resultado a diferentes resoluciones.

IV. Resultados experimentales

Una vez hemos visto la arquitectura del sistema, estamos en disposición de presentar una serie de pruebas sobre la herramienta desarrollada para probar su comportamiento a diferentes niveles de carga.

Antes de comenzar con las prueba, necesitamos analizar brevemente la arquitectura del sistema sobre el que se ejecuta la aplicación desarrollada. Este sistema no ha sido seleccionado de manera aleatoria, si no que se han llevado a cabo una serie de pruebas sobre diferentes equipos y, finalmente, se ha seleccionado como mejor candidato un sistema con un procesador AMD Ryzen 5 5600x y una tarjeta gráfica NVIDIA GeForce GTX 1660 [16].

[1] `https://qupath.github.io/`. Accedido el 30 de junio de 2023.

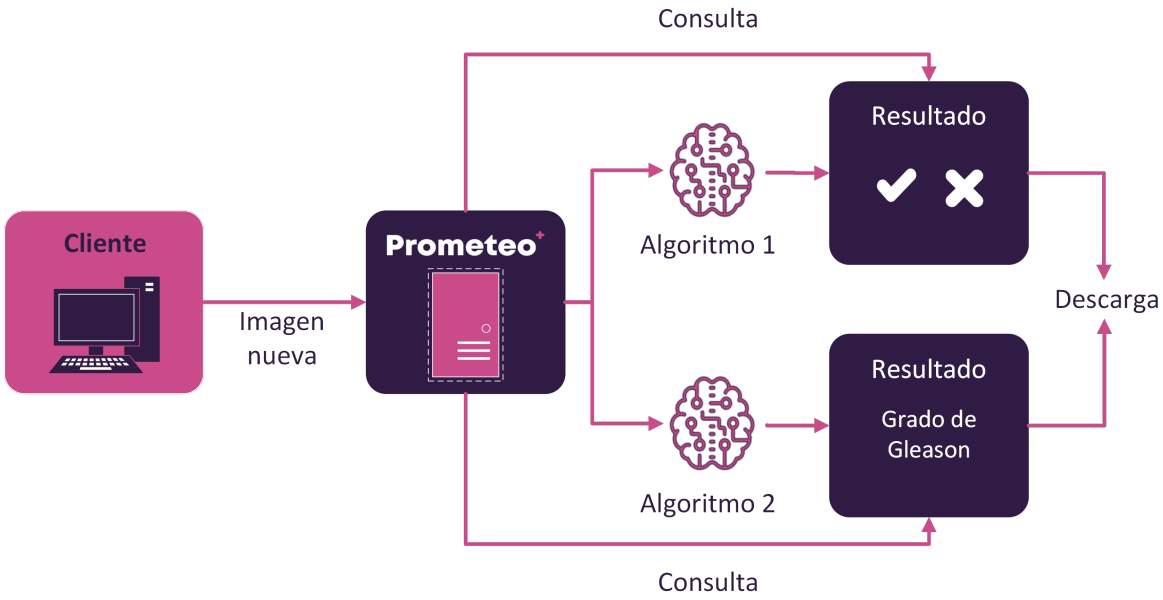


Fig. 2: Flujo de funcionamiento de la aplicación web PROMETEO WEB.

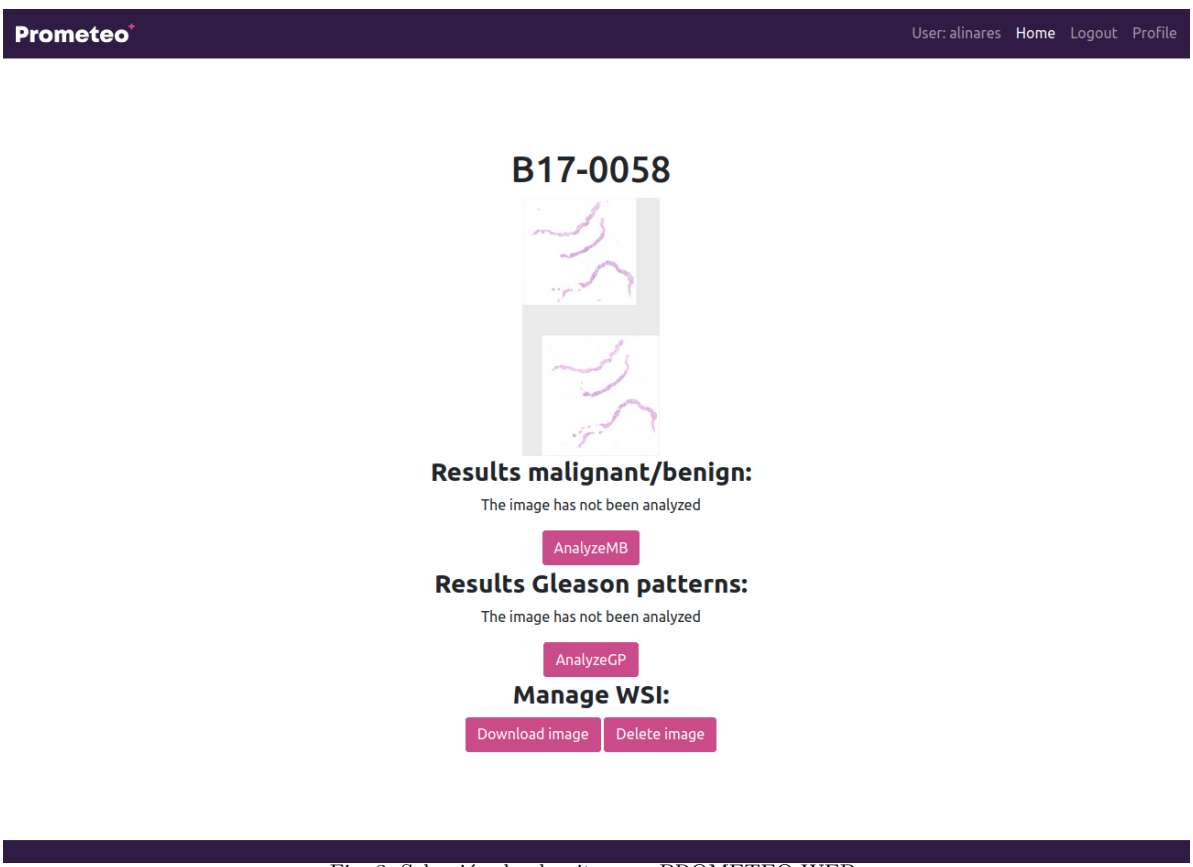


Fig. 3: Selección de algoritmo en PROMETEO WEB.

Por otro lado, es necesario dar una breve introducción a las diferentes fases por las que pasa una imagen a la hora de ser analizada. En primer lugar, se pasa por una fase de preprocesado, donde la imagen se va recorriendo por regiones de píxeles llamadas *patches*, a la vez que se va analizando la presencia de tejido en los píxeles seleccionados y se le aplica un filtrado a dicho conjunto de píxeles. Posteriormente, se realiza la inferencia con la red neuronal de aquellos *patches* que contienen un porcentaje elevado de tejido.

Para las pruebas, se va a utilizar un conjunto de imágenes de tipo WSI de diferentes tamaños sobre los dos algoritmos de inteligencia artificial actualmente disponibles en la aplicación web. Con estas pruebas se busca evaluar las siguientes mediciones:

- **Tiempo de análisis**: cálculo del tiempo necesario para analizar una imagen, dividido en tiempo de preprocesamiento y tiempo de inferencia de la red neuronal.
- **Porcentaje de uso de recursos**: análisis de porcentaje de uso del procesador, la memoria

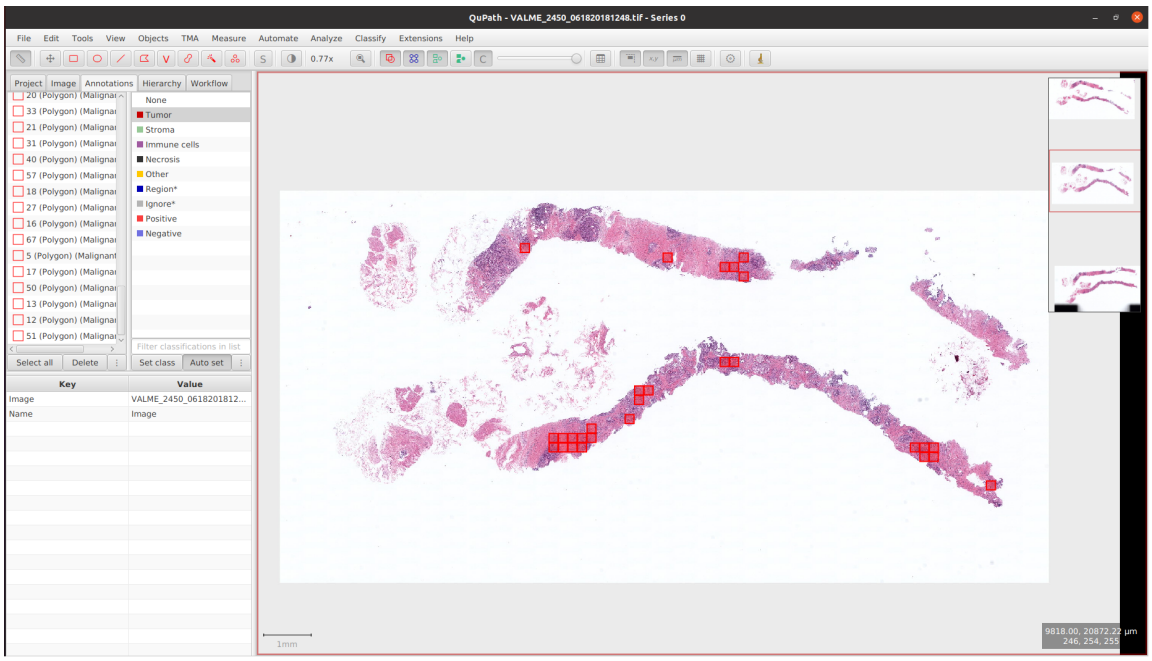


Fig. 4: Visualizado de resultado en la aplicación QuPath [18].

ram y la tarjeta gráfica durante el análisis de las diferentes imágenes.

A. *Algoritmo de detección binario*

Comenzamos realizando las pruebas sobre el algoritmo que analiza si hay presencia o no de células cancerígenas en el tejido digitalizado [7]. Este algoritmo cuenta con una ejecución mucho más rápida, ya que lo que se busca es localizar la presencia de células cancerígenas sin importar el grado de Gleason.

Tras realizar pruebas con diferentes imágenes, podemos ver que el consumo de recursos del sistema es reducido (véase la tabla I), a excepción de la carga en memoria RAM, que alcanza un 69 % de capacidad ocupada en algunos análisis.

Para dichas pruebas, se han analizando 4 imágenes de diferentes tamaños y formatos, obteniéndose una mediciones temporales muy variadas. Se puede destacar la influencia del formato de la imagen ya que, si observamos la tabla I, encontramos la "Imagen 3", que cuenta con unas dimensiones muy parecidas al resto de las imágenes, pero su tiempo de inferencia y de preprocesado se dispara en comparación al resto de las imágenes. Esto es debido mayormente, al formato resultante a la hora de digitalizar la biopsia, además de la posibilidad de una mayor concentración de tejido por *patch* recorrido en dicha imagen.

B. *Algoritmo de detección grados de Gleason*

Después de analizar el rendimiento de 4 imágenes sobre el algoritmo de PROMETEO para la detección de tejido maligno [7], procedemos a la segunda parte de la experimentación, donde evaluamos el rendimiento a la hora de analizar las mismas imágenes que en el caso anterior, pero con el algoritmo que detecta las regiones cancerígenas determinando el grado de Gleason de las mismas (véase la tabla 2II).

Como se puede observar en la tabla 2 II, se aprecia una reducción en el uso de la CPU y de la GPU si se compara con los resultados obtenidos al evaluar el algoritmo PROMETEO (I), aumentando el consumo de memoria RAM con la ejecución del segundo algoritmo.

Por otro lado, se puede observar que han aumentado los tiempos de preprocesamiento para cada imagen debido a la complejidad del procesamiento para el análisis de los grados de gleason. Sin embargo, el tiempo de inferencia es similar entre ambas redes neuronales, siendo el tiempo empleado en el preprocesamiento lo que marca la diferencia temporal entre las dos opciones de análisis disponibles.

Al igual que en el análisis realizado al algoritmo de detección binaria(I), encontramos una relación directa entre el tipo de compresión de la imagen y el tiempo de análisis, destacando la "Imagen 3" que tiene un tiempo de preprocesamiento 3 veces mayor al del resto de las imágenes analizadas.

Una vez vistos los resultados de los análisis realizados sobre los dos algoritmos, podemos saber que el consumo de recursos hardware es reducido en ambos casos, demostrando la capacidad del sistema de ejecutarse sobre equipos con pocos recursos de computación. Por otra parte, se ha demostrado la gran importancia del tipo de compresión usado a la hora de digitalizar las imágenes a analizar, suponiendo en algunos casos, un aumento significativo en el tiempo de análisis.

V. Conclusiones

En este trabajo se presenta un servicio basado en web como apoyo al diagnóstico de cáncer de próstata en imágenes histológicas. El sistema se ha implementado utilizando un gestor de bases de datos Django, el cual se encarga de gestionar todo lo relacionado al control de acceso de usuarios, subida y acceso a

Tabla I: Test de PROMETEO WEB con algoritmo de análisis binario (tejido maligno vs tejido benigno).

	Dimensione (Píxeles)	Número patches	Tiempo preprocesamiento (segundos)	Tiempo inferencia (segundos)	% Uso CPU	% Uso RAM	% Uso GPU
Imagen 1 675,7 MB (.tif)	84.752 x 159.456	24.025	2,50	12,50	17 %	67,4 %	16 %
Imagen 2 553,1 MB (.tif)	81.217 x 151.547	22.578	1,50	7,60	17 %	68,5 %	18 %
Imagen 3 1,5 GB (.bif)	84.400 x 150.480	21.881	37	40	17 %	69,5 %	18 %
Imagen 4 1,1 GB (.tif)	73.648 x 186.976	24.480	6,90	32	17 %	68,5 %	19 %

Tabla II: Test de PROMETEO WEB con algoritmo de identificación de patrones de Gleason.

	Dimensione (Píxeles)	Número patches	Tiempo preprocesamiento (segundos)	Tiempo inferencia (segundos)	% Uso CPU	% Uso RAM	% Uso GPU
Imagen 1 675,7 MB (.tif)	84.752 x 159.456	24.025	116	9.1	13 %	75,6 %	10 %
Imagen 2 553,1 MB (.tif)	81.217 x 151.547	22.578	84,5	7,6	14 %	76 %	12 %
Imagen 3 1,5 GB (.bif)	84.400 x 150.480	21.881	1408.9	68.5	15 %	75 %	10 %
Imagen 4 1,1 GB (.tif)	73.648 x 186.976	24.480	319.3	23.2	14 %	75,5 %	13 %

imágenes con sus correspondientes permisos, además de la gestión de solicitudes de diagnóstico. El servicio web integra dos modelos de redes neuronales de convolución, que permiten analizar las imágenes y reportar resultados tanto de zonas de tejido malignas como de la información relativa a la agresividad del cáncer en dicho tejido (distinguiendo entre grados de Gleason 3, 4 y 5). El resultado del diagnóstico reportado se almacena en archivos de valores separados por comas (CSV), que la propia web permite descargar, junto con el software necesario (QuPath) y un tutorial para visualizar el mapa de calor sobrepuesto sobre la imagen original. Todo el sistema al completo ha sido validado y se han realizado numerosas pruebas y experimentos para demostrar su funcionamiento. El sistema presentado aporta un workflow de trabajo completo, desde la recepción de la imagen escaneada hasta el análisis de la misma, siendo de gran utilidad como herramienta de diagnóstico por su alto porcentaje de acierto y como segunda opinión al patólogo, ayudándolo en casos especialmente complejos y como mecanismo de triaje en la clínica.

Agradecimientos

Esta trabajo ha sido financiado parcialmente por el proyecto I+D+i FEDER Andalucía 2014-2020 DAFNE (US-1381619), con soporte del fondo europeo de desarrollo regional, y por el proyecto MINDROB (PID2019-105556GB-C33) financiado por MCIN/AEI/10.13039/501100011033.

Referencias

[1] Hyuna Sung, Jacques Ferlay, Rebecca L. Siegel, Mathieu Laversanne, Isabelle Soerjomataram, Ahmedin Jemal, and Freddie Bray, "Global cancer statistics 2020: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries," *CA: A Cancer Journal for Clinicians*, vol. 71, no. 3, pp. 209–249, 2021.

[2] Rebecca C Fitzgerald, Antonis C Antoniou, Ljiljana Fruk, and Nitzan Rosenfeld, "The future of early cancer detection," *Nature Medicine*, vol. 28, no. 4, pp. 666–677, 2022.

[3] Gabriele Campanella, Matthew G Hanna, Luke Geneslaw, Allen Miraflor, Vitor Werneck Krauss Silva, Klaus J Busam, Edi Brogi, Victor E Reuter, David S Klimstra, and Thomas J Fuchs, "Clinical-grade computational pathology using weakly supervised deep learning on whole slide images," *Nature medicine*, vol. 25, no. 8, pp. 1301–1309, 2019.

[4] Y Wang, B Acs, S Robertson, B Liu, Leslie Solorzano, Carolina Wählby, J Hartman, and M Rantalainen, "Improved breast cancer histological grading using deep learning," *Annals of Oncology*, vol. 33, no. 1, pp. 89–98, 2022.

[5] Sara Kuntz, Eva Krieghoff-Henning, Jakob N Kather, Tanja Jutzi, Julia Höhn, Lennard Kiehl, Achim Hekler, Elizabeth Alwers, Christof von Kalle, Stefan Fröhling, et al., "Gastrointestinal cancer classification and prognostication from histology using deep learning: Systematic review," *European Journal of Cancer*, vol. 155, pp. 200–215, 2021.

[6] Osamu Iizuka, Fahdi Kanavati, Kei Kato, Michael Rambeau, Koji Arihiro, and Masayuki Tsuneki, "Deep learning models for histopathological classification of gastric and colonic epithelial tumours," *Scientific reports*, vol. 10, no. 1, pp. 1504, 2020.

[7] Lourdes Duran-Lopez, Juan Pedro Dominguez-Morales, Antonio Conde-Martin, Saturnino Vicente Díaz, and Alejandro Linares-Barranco, "Prometeo: A cnn-based computer-aided diagnosis system for wsi prostate cancer detection," *IEEE Access*, vol. PP, pp. 1–1, 07 2020.

[8] José M. Marrón-Esquivel, L. Duran-Lopez, A. Linares-Barranco, and Juan P. Dominguez-Morales, "A comparative study of the inter-observer variability on gleason grading against deep learning-based approaches for prostate cancer," *Computers in Biology and Medicine*, vol. 159, pp. 106856, 2023.

[9] John KC Chan, "The wonderful colors of the hematoxylin–eosin stain in diagnostic surgical pathology," *International journal of surgical pathology*, vol. 22, no. 1, pp. 12–32, 2014.

[10] Navid Farahani, Anil V Parwani, Liron Pantanowitz, et al., "Whole slide imaging in pathology: advantages, limitations, and emerging perspectives," *Pathol Lab Med Int*, vol. 7, no. 23-33, pp. 4321, 2015.

[11] Adel RH Eskaros, Shanna A Arnold Egloff, Kelli L Boyd,

Joyce E Richardson, M Eric Hyndman, and Andries Zijlstra, "Larger core size has superior technical and analytical accuracy in bladder tissue microarray," *Laboratory Investigation*, vol. 97, no. 3, pp. 335–342, 2017.

[12] Andres Matoso and Jonathan I Epstein, "Grading of prostate cancer: past, present, and future," *Current urology reports*, vol. 17, no. 3, pp. 1–6, 2016.

[13] Alastair M Lessells, Rodney A Burnett, S Rosalind Howatson, Stephen Lang, Frederick D Lee, Kathryn M McLaren, E Robert Nairn, Simon A Ogston, Alistair J Robertson, John G Simpson, et al., "Observer variability in the histopathological reporting of needle biopsy specimens of the prostate," *Human pathology*, vol. 28, no. 6, pp. 646–649, 1997.

[14] M McLean, J Srigley, D Banerjee, P Warde, and Y Hao, "Interobserver variation in prostate cancer gleason scoring: are there implications for the design of clinical trials and treatment strategies?," *Clinical oncology*, vol. 9, no. 4, pp. 222–225, 1997.

[15] Eirini Arvaniti, Kim S Fricker, Michael Moret, Niels Rupp, Thomas Hermanns, Christian Fankhauser, Norbert Wey, Peter J Wild, Jan H Rueschoff, and Manfred Claassen, "Automated gleason grading of prostate cancer tissue microarrays via deep learning," *Scientific reports*, vol. 8, no. 1, pp. 1–11, 2018.

[16] Lourdes Duran-Lopez, Juan P. Dominguez-Morales, Antonio Rios-Navarro, Daniel Gutierrez-Galan, Angel Jimenez-Fernandez, Saturnino Vicente-Diaz, and Alejandro Linares-Barranco, "Performance evaluation of deep learning-based prostate cancer screening methods in histopathological images: Measuring the impact of the model's complexity on its processing speed," *Sensors*, vol. 21, no. 4, 2021.

[17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[18] Peter Bankhead, Maurice B Loughrey, José A Fernández, Yvonne Dombrowski, Darragh G McArt, Philip D Dunne, Stephen McQuaid, Ronan T Gray, Liam J Murray, Helen G Coleman, et al., "Qupath: Open source software for digital pathology image analysis," *Scientific reports*, vol. 7, no. 1, pp. 1–7, 2017.

[19] Rawla P., *Epidemiology of Prostate Cancer*, World Journal of Oncology, 2019 Apr.

Aplicaciones de la computación de altas prestaciones

Una propuesta paralela con paso de mensajes para la implementación de un Pipeline en el desarrollo del videojuego SIMON

Mario Rossainz-López[1], Bárbara Sánchez-Rinza[1], Liosbel Cabrera-Hernández[1], Manuel I. Capel-Tuñón[2]

***Resumen*— Se presenta el diseño y desarrollo del patrón de comunicación entre procesos llamado Pipeline como una propuesta de Composición de Objetos Paralelos utilizando una librería de clases particular llamada JPMI (Java Passing Message Interface) para programación paralela con paso de mensajes y poder implementar una versión original y particular del conocido videojuego llamado SIMON con el objetivo, por un lado, de mostrar la utilidad de este diseño dentro de la Programación Paralela Estructurada y por otro lado que ésta propuesta sirva para garantizar un buen rendimiento en la ejecución de aplicaciones de tiempo real donde se requiera llevar a cabo la renderización de imágenes, la interacción del usuario con la aplicación a partir de dichas imágenes y la salida que proporcione el sistema en respuesta a dicha interacción. Un ejemplo de este tipo de aplicaciones son precisamente los videojuegos. El videojuego SIMON consiste en que el jugador tiene que ser capaz de memorizar y repetir una secuencia de colores que es generada por la aplicación a través de un pipeline que define una secuencia de colores. Se compara esta propuesta con otra que utiliza como motor de renderizado OGRE y una biblioteca de hilos llamada boost y ZeroC Ice para la invocación remota de objetos distribuidos. Se comparan los tiempos de ejecución y las aceleraciones de ambas propuestas para identificar que tan parecidas o diferentes son en sus respectivos rendimientos con pruebas de entrenamientos utilizando módulos de IA con secuencias de 500000 colores en un cluster de 2 CPUs Intel Xeon de 8 cores cada uno y 2 nodos, cada uno con 2 tarjetas NVIDIA de 5760 núcleos CUDA cada una y una memoria RAM de 128 GB.**

Palabras Clave— **Programación Paralela Estructurada, CPAN, Pipeline, Paso de Mensajes, JPMI, Videojuego, SIMON**

I. Introducción

La Programación Paralela Estructurada se basa en el modelado, diseño, construcción y desarrollo de patrones de comunicación entre los procesos que se definen en una aplicación, con varios objetivos; uno de ellos es la separación de la interacción que tienen los procesos en su comunicación respecto de la funcionalidad de estos [1]. Al lograrlo podemos obtener un patrón paralelo de comunicación genérico que mediante el uso del paradigma orientado a objetos podemos particularizarlo a la solución de un problema que se pueda resolver con dicho patrón a través de propiedades como el reuso, herencia y polimorfismo. Esto lleva al segundo objetivo que es facilitar al programador novel la programación "automática" de la parte paralela de su propuesta algorítmica, centrando su esfuerzo en el diseño y codificación de los algoritmos secuenciales de su solución [1]. Existen en la actualidad diversos patrones de comunicación que representan modelos de solución en la interacción entre procesos tales como los Pipelines, Farms, Trees, Cube, Hipercube, Mesh, etc. y que son utilizados en áreas y disciplinas distintas. El modelo de las Composiciones Paralelas de Alto Nivel o CPAN pretende ser un modelo genérico para el diseño de patrones de comunicación de procesos que se adapte a un patrón particular en la solución de un problema secuencial que puede ser paralelizable. Un CPAN es una composición de objetos paralelos de tres tipos: Un objeto Manager que representa al CPAN en sí mismo y controla las referencias de un conjunto de objetos (Collector y Stages), que representan los demás componentes del CPAN y cuya ejecución se lleva a cabo en paralelo y debe ser coordinada por el propio manager. Los objetos Stage definen la conexión necesaria entre ellos para implementar la semántica del patrón de comunicación que se pretende definir y el objeto Collector almacena en paralelo los resultados que le lleguen de los objetos Stage que tenga conectados para generar una solución al problema que es pasada al Manager para la comunicación con el usuario (los detalles de su desarrollo se encuentran en [2]). En base a esta idea se crea el CPAN Pipe como aquel patrón de comunicación que implementa un Pipeline de procesos para resolver problemas que se descomponen en una serie de tareas sucesivas de manera que los datos fluyen en una cierta dirección por la estructura de procesos y cada tarea es completada una después de otra. El problema que se resuelve mediante este modelo de CPAN es el desarrollo del videojuego conocido como SIMON que consiste en que el usuario tiene que ser capaz de memorizar y repetir una secuencia de colores que es generada por la aplicación a través de un pipeline que define una secuencia de colores. Actualmente la industria de los videojuegos ha tenido un gran desarrollo y avance en la disciplina computacional, tanto en calidad gráfica como en la incorporación ahora de algoritmos de IA y en el uso de modelos distribuidos y en red, no sólo como aplicaciones de entretenimiento o lúdicas sino también

[1]Benemérita Universidad Autónoma de Puebla, Avenida. San Claudio y 14 Sur, San Manuel, Puebla, Puebla, 72000, México. E-mail: {mario.rossainz, barbara.sanchez}@correo.buap.mx, liosbel.cabrera@gmail.com

[2]DPT. Ingeniería de Software, Colegio de Informática y Telecomunicaciones, Universidad de Granada, ETSIIT Calle Periodista Daniel Saucedo Aranda s/n, 18071 Granada, España. E-mail: manuelcapel@ugr.es

como aplicaciones que ayudan a mejorar las habilidades cognitivas de las personas [3] como lo puede ser el videojuego que aquí se presenta.

Es por eso que el presente escrito centra su esfuerzo en explicar como la programación paralela estructurada a través del modelo del CPAN Pipe puede ser útil para desarrollar una propuesta particular de implementación del videojuego SIMON donde el uso del pipeline resulta ser inherentemente natural utilizarlo en esta propuesta paralela de desarrollo, de manera particular utilizando la programación con paso de mensajes a través de una librería de clases que implementa el álgebra de procesos del CSP de Hoare llamada JPMI.

El presente escrito tiene la estructura siguiente: El primer capítulo es esta introducción, el segundo explica cómo está diseñada y construida la librería de clases JPMI para la programación paralela con paso de mensajes. El tercer capítulo habla del patrón Pipeline como Composición de Objetos Paralelos o CPAN. El capítulo IV explica cómo se implementó el videojuego SIMON a través del CPAN Pipe, junto con la librería de clases JPMI. En el capítulo V se muestran los resultados obtenidos y el análisis de rendimiento de la propuesta comparándola con aquella que se propone en [20] y tener un patrón de que tan buenos son los tiempos de ejecución y la aceleración o Speedup de Cpan Pipe SIMON implementado. Finalmente, en el capítulo VI se muestran las conclusiones de este trabajo.

II. Programación con paso de mensajes en Java

Dada la importancia de contar con herramientas actuales en Java que proporcionen programación con paso de mensajes, el presente trabajo muestra el uso y utilidad de la biblioteca de clases JPMI (Java Passing Message Interface) que implementa el álgebra de procesos CSP de Hoare para mejorar el rendimiento de aplicaciones que se pueden paralelizar mediante este esquema. La biblioteca JPMI proporciona clases para generar procesos, canales de comunicación y composiciones secuenciales, paralelas y alternativas de procesos, para poder comunicarlos y sincronizarlos. Los antecedentes de esta librería se muestran en el proyecto de la Universidad de Twente que derivó en lo que se conoce como CTJ (Communicating Threads for Java) [4],[5] y en el proyecto de la Universidad de Kent que culminó en la propuesta de la librería de clases JCSP (Communicating Sequential Process for Java) [6]. La ventaja de JPMI respecto de CTJ y JCSP es que muestra una versión actualizada de éstas últimas las cuales se muestran ya obsoletas y además el programador cuenta con un conjunto amplio de reglas que ayudan a eliminar las condiciones no deseables del paralelismo durante las fases de diseño e implementación tales como la alternancia estricta, la falta de exclusión mutua, el interbloqueo y la espera infinita. Los procesos se muestran como objetos activos con capacidad de ejecución en sí mismos y con otros procesos al crear una composición de éstos, en tanto que los canales son objetos pasivos que sirven como medio de comunicación entre los procesos que los utilizan. El modelo general de comunicación se representa en la Fig.1. En él se identifican los elementos fundamentales que intervienen en la comunicación en sistemas con paso de mensajes (un proceso emisor, un proceso receptor, un canal de comunicación, el mensaje a enviar/recibir y las operaciones de envío y recepción) [7], [8].

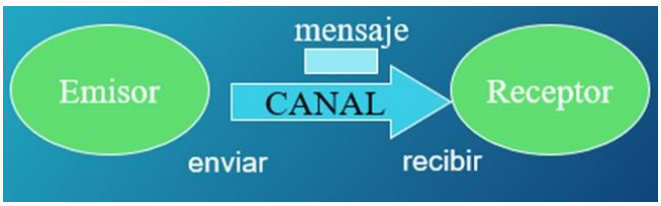


Fig.1. Modelo de Comunicación de Procesos con paso de mensajes

A. Tipos de Comunicación entre procesos [7][8]

- Comunicación Directa: El emisor identifica explícitamente al receptor del mensaje en la operación de envío y viceversa para la operación de recepción por parte del receptor.
- Comunicación Indirecta: No se identifican explícitamente a los procesos emisor y receptor. La comunicación se realiza depositando los mensajes en un almacén intermedio (buzón) que se supone conocido por los procesos interesados en la comunicación.

B. Sincronización entre procesos [7],[8]

- Comunicación Asíncrona. El proceso emisor puede realizar la operación de envío sin que para ello sea necesario que coincida en el tiempo con la operación de recepción por parte del proceso receptor.
- Comunicación Síncrona. Debe darse la coincidencia (cita o encuentro) en el tiempo de las operaciones de envío y recepción por parte de los procesos emisores y receptores.

C. Características de los canales y mensajes [9]

- Flujo de Datos. El flujo de datos que pasa por un canal de comunicación entre dos procesos puede ser unidireccional o bidireccional.
- Capacidad del Canal. Es la posibilidad que tiene el enlace de comunicación de almacenar los mensajes enviados por el proceso emisor cuando éstos no son recogidos de forma inmediata por el proceso receptor.
- Tamaño de los mensajes. Los mensajes pueden ser de longitud fija o variable.
- Canales con tipo o sin tipo. Algunos esquemas de comunicación exigen definir el tipo de datos que va a fluir por el canal por tanto podemos tener canales tipados o no tipados
- Paso por copia o por referencia. La información enviada por el proceso emisor hacia el proceso receptor a través de un canal se realiza efectuando una copia exacta de los datos (mensaje) o simplemente el envío y recepción de la dirección en el espacio de direcciones donde se encuentra el mensaje.

D. La biblioteca de clase JPMI

JPMI (Java Passing Message Interface) es un paquete de clases que implementa el álgebra de Procesos CSP de Hoare y se utiliza para crear procesos, composiciones de procesos y canales de comunicación entre procesos. JPMI, ha de implementar la interface *Jpmi.Proceso* y proporcionar una implementación para su método *run()* el cual contendrá la tarea que el proceso quiere llevar a cabo cuando este método

es invocado por otro dentro de una composición que puede ser de alguna de los tipos permitidos: secuencial, paralela o alternativa. El constructor del proceso especifica los canales de entrada, de salida y parámetros adicionales para inicializar el estado del proceso. El método run() es el único método público que un proceso puede invocar directamente en otro proceso. La Fig.2 muestra su diseño arquitectónico a través de un diagrama de clases en UML.

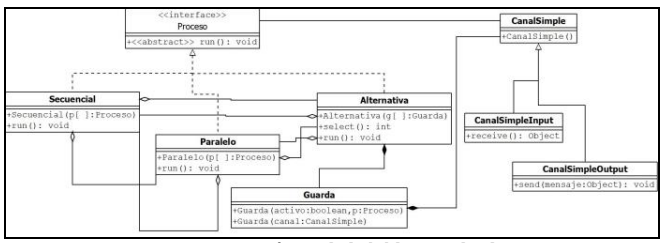


Fig.2. Diseño arquitectónico de la biblioteca de clases JPMI

En JPMI los canales son unidireccionales, de capacidad cero, no tipados (genéricos) y con operaciones de envío y recepción de mensajes los cuales son de longitud variable y pasados por copia. JPMI pretende ser un puente entre la teoría del CSP y su aplicación en JAVA (para los detalles de diseño e implementación, consultar [10]).

III. El Pipeline y su representación como Composición de Objetos Paralelos

El Pipeline es una técnica de procesamiento paralelo aplicable a un amplio rango de problemas que son parcialmente secuenciales por naturaleza. Con este esquema podemos resolver un problema descomponiéndolo en una serie de tareas sucesivas de manera que los datos fluyen en una cierta dirección por la estructura de procesos y cada tarea puede ser completada una después de otra [11]. En un pipeline cada tarea es ejecutada por un proceso tal como lo muestra la Fig.3. A cada proceso que conforma un pipeline se le llama "etapa" o "stage" [12].

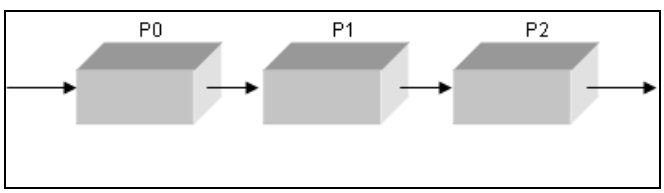


Fig.3. Estructura de un Pipeline

Cada etapa del pipeline contribuye al problema general y pasa la información necesaria a la siguiente etapa con la que está conectado. Este tipo de paralelismo es visto como una "descomposición funcional", pues el problema es dividido en funciones separadas que pueden ser ejecutadas de manera individual e independiente [11],[12]. Un algoritmo que resuelva un determinado problema puede ser formulado como un pipeline si se puede dividir en una serie de funciones que podrían ser ejecutadas por las etapas del pipeline. De esta forma, si un problema puede ser dividido en una serie de tareas secuenciales, el enfoque del pipeline puede proporcionar incremento en la velocidad de ejecución de los siguientes tres tipos de cálculos:

1. Cuando más de una instancia del problema completo puede ser ejecutada en paralelo;
2. O bien una serie de datos pueden ser procesados y cada uno de estos se utiliza en múltiples operaciones;
3. O bien si la información que demanda el siguiente proceso para iniciar su cálculo se pasa después de que el proceso actual haya completado todas sus operaciones internas.

Con esta técnica muchos de los problemas computacionales que se llevan a cabo de forma secuencial pueden ser fácilmente paralelizados como un pipeline (para mayor detalle, consultar [13])

Esta técnica del Pipeline se ha desarrollado como una composición de objetos paralelos o CPAN (acrónimo de Composición Paralela de Alto Nivel) aplicable a un amplio rango de problemas que son parcialmente secuenciales en su naturaleza, de tal forma que el CPAN Pipe garantiza la paralelización de código del algoritmo secuencial utilizando el patrón Pipeline. Un CPAN representa la composición de un conjunto de objetos paralelos de tres tipos: Un objeto Manager que representa al CPAN en sí mismo (Fig.4). El Manager controla las referencias de un conjunto de objetos (un objeto Collector y varios objetos Stage), cuya ejecución se lleva a cabo en paralelo y es coordinada por el propio Manager.

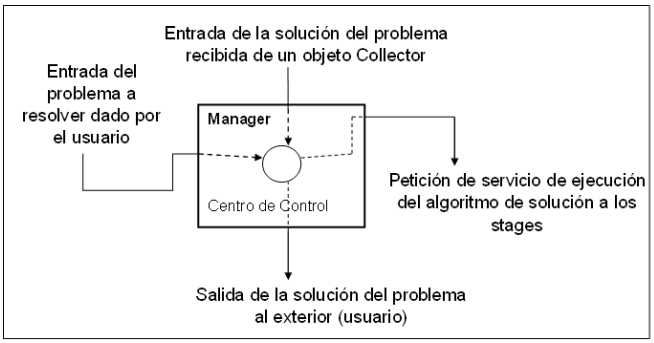


Fig.4. Modelo del componente Manager de un CPAN

Los objetos Stage (Fig.5) son objetos de propósito específico, encargados de encapsular una interfaz tipo cliente-servidor que se establece entre los Stages y los objetos esclavos (objetos que no son activamente participativos en la composición del CPAN, sino que se consideran entidades externas que contienen el algoritmo secuencial que constituye la solución de un problema dado), así como el proporcionar la conexión necesaria entre ellos para implementar la semántica del patrón de comunicación que se pretende definir. En otras palabras, cada Stage debe actuar en paralelo como un nodo del grafo que representa al patrón. Un Stage puede estar directamente conectado al Manager y/o a otros componentes Stage dependiendo del patrón particular del CPAN implementado.

Un objeto Collector (Fig.6) es un objeto encargado de almacenar en paralelo los resultados que le lleguen de los objetos Stage que tenga conectados. Es decir, durante el servicio de una petición, el flujo de control dentro de los Stages de un CPAN depende del patrón de comunicación

implementado. Cuando la composición finaliza su ejecución, el resultado no se regresa directamente al Manager, sino que una instancia de la clase Collector se encarga de almacenar dichos resultados y de enviarlos al Manager que enviará al exterior los resultados, tan pronto como le vayan llegando, sin que exista la necesidad de esperar a que hayan sido obtenidos todos en su totalidad.

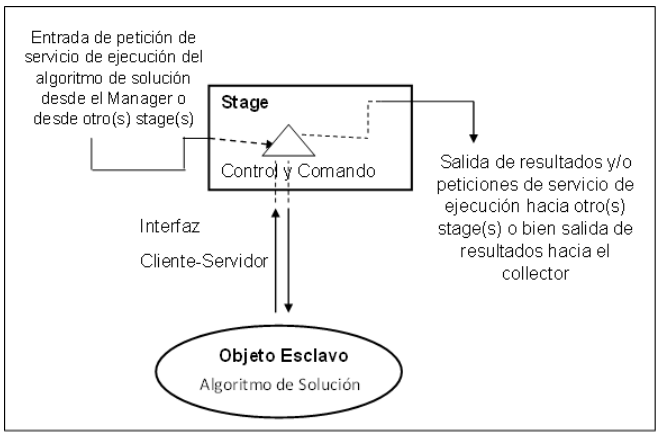


Fig.5. Modelo del componente Stage de un CPAN

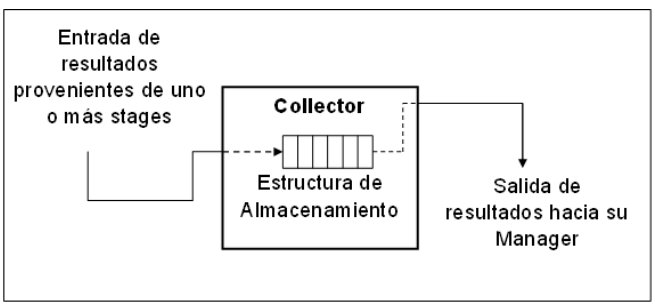


Fig.6. Modelo del componente Collector de un CPAN

Los objetos Manager, Collector y Stages son Objetos Paralelos (PO) que pueden explotar tanto el paralelismo entre objetos (inter-object) como el paralelismo interno de ellos (intra-object) [14].

Un PO tiene una estructura similar a la de un objeto en Java, pero además incluye una política de planificación que especifica la forma de sincronizar una o más operaciones de la clase del objeto susceptibles de invocarse en paralelo [14],[15]. Los objetos paralelos soportan herencia simple con múltiples interfaces, lo que permite derivar una nueva especificación de PO a partir de una que ya existe. Un CPAN cuenta con las siguientes propiedades: Modos de comunicación síncrono, asíncrono y futuro asíncrono entre los objetos paralelos del CPAN, objetos con paralelismo interno, disponibilidad de mecanismos de sincronización; paralelismo máximo, exclusión mutua y sincronización del tipo Productor-Consumidor, disponibilidad de control de tipos genéricos, transparencia en la distribución de aplicaciones paralelas y rendimiento satisfactorio: Programabilidad-Portabilidad-Performance [16]. La Fig.7 muestra el modelo del patrón paralelo de comunicación Pipeline como un CPAN.

La caja que engloba a los componentes representa el CPAN Pipeline encapsulado. Las cajas internas representan objetos compuestos (Collector, Manager y objetos Stages), en tanto que los círculos son los objetos esclavos asociados a los Stages. Las líneas dirigidas dentro del CPAN suponen las comunicaciones existentes entre los objetos citados; por ejemplo, entre el Manager y el Stage_1. Lo mismo sucede entre los Stages y con el objeto Collector. El Manager esta comunicado con el exterior para recibir los datos iniciales del problema y entregar su solución, y cada Stage del Pipeline se asocia con su respectivo Objeto Esclavo los cuales no pertenecen al CPAN, pero son necesarios para indicarle a los Stages el algoritmo secuencial que tienen que ejecutar.

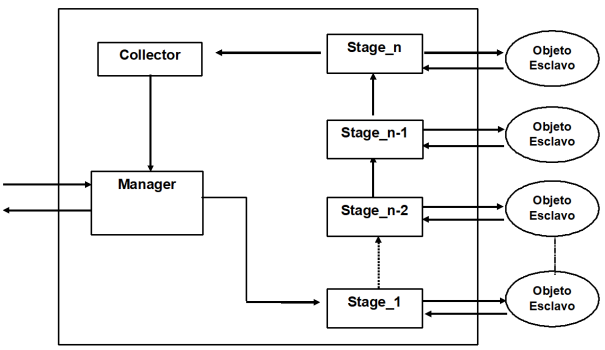


Fig.7. Modelo del Pipeline como una Composición de Objetos Paralelos o CPAN

IV. Implementación del videojuego SIMON como CPAN-Pipe usando JPMI

Un videojuego es una aplicación gráfica en tiempo real con una interacción explícita entre el usuario y el propio videojuego. La noción de tiempo real implica entonces que el videojuego debe hacer que el usuario tenga una sensación continua de realismo cuando juega [17], esto se logra generando un ciclo de 3 pasos:

- El usuario visualiza una imagen renderizada
- El usuario interactúa con la aplicación en función de lo que visualice
- En base a esa interacción la aplicación responde con una salida

Este ciclo debe ejecutarse de forma rápida y constante para que el usuario se sienta inmerso en el videojuego y no tenga la sensación de ver imágenes estáticas. Técnicamente esto significa que el videojuego debe generar una determinada cantidad de imágenes por segundo (frames) con base en la interacción con el usuario y es aquí precisamente donde el paralelismo y la concurrencia pueden ayudar a lograr esa ejecución acelerada y no pausada del videojuego. La propuesta planteada en el presente escrito del uso del CPAN Pipe programado con la librería JPMI puede lograr este cometido. Como un caso de estudio mostramos a continuación una propuesta de paralelización en el diseño e implementación del ya conocido videojuego SIMON que consiste en que el jugador tiene que ser capaz de memorizar y repetir una secuencia de colores que es generada por SIMON (ver Fig.8).

Fig.8. Videojuego SIMON implementado como CpanPipe

Primero creamos el módulo de inteligencia artificial (IA) responsable de la generación de una secuencia de color del videojuego; particularmente se adoptó la idea de [17] de incorporar un algoritmo genético y de aprendizaje profundo que representa el Objeto Esclavo del modelo del CPAN Pipe como una instancia de la funcionalidad a ejecutarse por un Stage del CPAN. Particularmente para la creación del módulo de IA se utilizó el modelo de entrenamiento que define Intel a través de redes neuronales usando la librería Deep Neural Networks y la extensión de Intel para TensorFlow, todo ello encapsulado en el Objeto Esclavo que se asocia a cada Stage del CPAN Pipeline.

A continuación, utilizando la interface "Proceso" de la biblioteca JPMI creamos los objetos paralelos del modelo CPAN Pipe de la Fig.7, particularizados al diseño del videojuego, es decir, el objeto Manager de éste nuevo CPAN Pipe al que llamaremos CpanPipeSIMON es un proceso que, por un primer canal de entrada, recibirá en cada oportunidad para el usuario la secuencia de colores que SIMON defina como el patrón a seguir. Esta secuencia irá generando el Pipeline del modelo según el número de colores que la integren, un objeto Stage del pipeline por cada color en la secuencia generada en el instante de tiempo corriente.
Al principio se genera un primer Stage con dos canales de entrada y uno de salida. El primer canal de entrada esta conectado con el canal de salida del Manager para recibir la secuencia del usuario que al principio del juego será de sólo un color, el definido por ese primer Stage, que por su segundo canal de entrada reciba de su objeto esclavo asociado (Algoritmo genético y de aprendizaje profundo de IA adaptado de [17]) y que dormirá al módulo de IA (del objeto esclavo) entre sus actualizaciones, permitirá controlar el acceso concurrente de cada Stage que forma parte del CPAN conforme se vaya generando la lista de colores y comparará con el color que reciba del Manager para verificar que la secuencia definida por el usuario sea la misma que genera SIMON a través de la secuencia definida (secuencia de Stages conectados uno después de otro generando el Pipeline) y el resultado de esa comparación será enviado al objeto Collector que será el tercer proceso del modelo que recibe por su canal de entrada ese resultado de comparación para dar respuesta al usuario a través del Manager quien por otro canal de entrada recibe ese resultado e informa al usuario de si la secuencia es correcta o no. Nuevamente se repite el mismo proceso generándose un segundo Stage conectado con el primero, y después un tercer Stage conectado con el segundo y así sucesivamente hasta que el usuario erre en la secuencia que tiene que ir siguiendo y que esta siendo dictada por SIMON. El Pipeline que se va generando representa entonces la secuencia de colores creada por SIMON y que el usuario tiene que seguir. Cada proceso Stage pasará al siguiente (a través de los canales de salida y entrada conectados entre los Stages vecinos) una bandera de acierto o error según el usuario vaya avanzando en la generación de dicha secuencia y que será enviada al Collector para que éste formule el resultado final y lo envíe al Manager para que éste a su vez le indique al usuario si puede seguir jugando o no. El juego irá contabilizando la cantidad de colores acertados por el usuario en la generación de la secuencia hasta que éste último falle y servirá para que aprendizaje profundo adoptado de [17] vaya aprendiendo del usuario para después, en un siguiente juego generar una secuencia más complicada de colores. Como se puede ver, el pipeline del modelo es dinámico, va creciendo en tiempo real conforme la secuencia de colores del usuario siga siendo correcta, hasta que éste falle.
Los objetos paralelos del Manager, Stages y Collector se encuentran ejecutándose dentro de una Composición Paralela de Procesos generada a través del uso de la clase Paralelo de la librería JPMI en base al comportamiento que debe tener este nuevo proceso, modelado a través del álgebra del CSP. El nuevo modelo gráfico del CpanPipeSIMON se ilustra en la Fig.9 y corresponde con lo narrado hasta este momento.

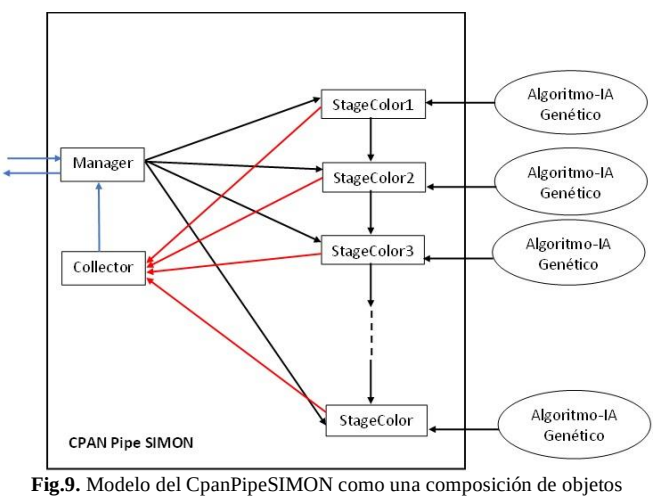


Fig.9. Modelo del CpanPipeSIMON como una composición de objetos paralelos usando la librería JPMI del álgebra CSP

V. Resultados y Análisis del Rendimiento del CpanPipeSIMON

Para medir el rendimiento del videojuego SIMON implementado con el modelo del CPAN, se tomó como referencia comparativa la implementación de [18] de éste mismo videojuego en donde utiliza como motor de renderizado OGRE y una biblioteca de hilos llamada boost y ZeroC Ice para la invocación remota de objetos distribuidos. Ambas propuestas, la de [18] y la que se expone en este escrito se ejecutaron en un cluster con 2 CPUs Intel Xeon de 8 cores cada uno y 2 nodos, cada uno con 2 tarjetas NVIDIA de 5760 núcleos CUDA cada una y una memoria RAM de 128 GB. Para el caso de nuestro modelo, el nodo servidor se alojó en una CPU donde se colocaron los objetos Manager y Collector, mientras que el Pipeline que se genera de forma dinámica junto con los objetos esclavos, se alojaron como clientes en los núcleos CUDA de los GPUs correspondientes. En ambas propuestas se realizó un entrenamiento para llevar a cabo los cálculos de los speedup's correspondientes con secuencias de colores de 500000 elementos los cuales se muestran en la gráfica de la Fig.10.

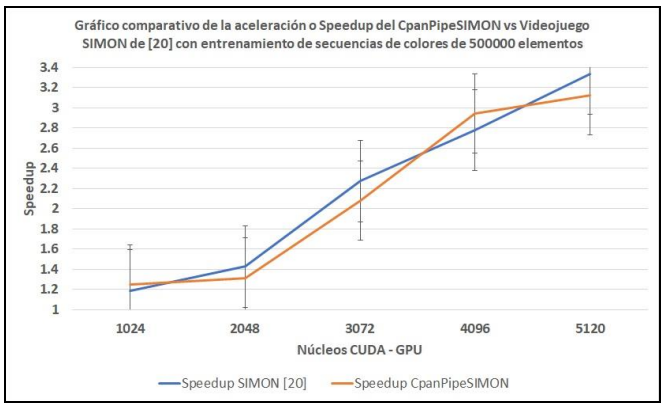


Fig.10. Comparativo de la escalabilidad del Speedup encontrado en el CpanPipeSIMON vs Videojuego SIMON [20] con entrenamiento de secuencias de colores de 500000 elementos

El gráfico muestra la media poblacional de una distribución Normal de Probabilidades, de una serie de entrenamientos de ambas propuestas con secuencias correctas de 500000 colores y el uso de 1024 hasta 5120 núcleos CUDA en incrementos de 1024. Cada color de una secuencia produce un delay de 0.2 segundos en obtenerse. La media de las secuencias generadas de forma secuencial fue de aproximadamente 50 hrs de ejecución, mientras que las medias de los tiempos paralelos de ejecución de las 2 propuestas se muestras en la Tabla I.

Tabla 1. Promedios de los Tiempos de Ejecución en hrs del Videojuego SIMON [20] vs CpanPipeSIMON con entrenamiento de secuencias de colores de 500000 elementos

Núcleos CUDA	*T. Ejec (hrs) SIMON [20]*	*T. Ejec (hrs) CpanPipeSIMON*
1024	42	40
2048	35	38
3072	22	24
4096	18	17
5120	15	16

Como se puede observar tanto en los valores de la Tabla I, como en el gráfico de la Fig.10 el rendimiento del CpanPipeSIMON es casi idéntico al mostrado con la propuesta del videojuego SIMON de [18]. La diferencia de los errores del comparativo en el gráfico de la Fig.10 de los speedup es muy pequeño, pues los tiempos de ejecución de ambas propuestas bajo las mismas condiciones son muy parecidos (ver Tabla I).

VI. Conclusiones

Se ha presentado el diseño de una composición de objetos paralelos para modelar e implementar la estructura de comunicación Pipeline como una Composición Paralela de Alto Nivel o CPAN cuya implementación se llevó a cabo a través del uso de la librería JPMI para programación con paso de mensajes particularmente en el caso de estudio del conocido videojuego SIMON. Con ello quedó demostrada la utilidad de dicha librería al ser usada en la Programación Paralela Estructurada para desarrollar el CPAN propuesto. La implementación particular de este videojuego se realizó tomando como base el modelo CpanPipeSIMON (ver Fig.9) a través de una composición paralela de 3 tipos de procesos: Manager, Collector y Stages para crear un Pipeline dinámico donde cadas Stage del Pipeline representa un color que es generado aleatoriamente por el videojuego en una secuencia que debe ser seguida de forma correcta por el usuario, mediante técnicas de IA de aprendizaje profundo.

El análisis del rendimiento de esta propuesta se hizo mediante una comparación tanto en tiempos de ejecución como en aceleración con la propuesta que plantea [18] y cuyos resultados son mostrados en la Fig.10 y Tabla I e ilustran la similitud que existe entre estas dos implementaciones aun cuando fueron diseñadas y desarrolladas con modelos diferentes en el diseño y codificación de algoritmos. Los rendimientos se consideran buenos dadas las condiciones de entradas y salidas hacia y desde el videojuego y la plataforma de hardware utilizada para su análisis de ejecución y aceleración.

Referencias

[1] McCool M., Robison A.D., and Reinders J. Structured Parallel Programming. Patterns for Efficient Computation. Morgan Kaufmann Publishers Elsevier. USA. (2012).

[2] Rossainz M. and Capel M. Design and implementation of communication patterns using parallel objects. Especial edition, Int. J. Simulation and Process Modelling, Volume 1, Number 17. (2017).

[3] Simone B., López C. Breve historia de los videojuegos. Athenea Digital, Revista de Pensamiento e Investigación Social. Número 14, Universidad Autónoma de Barcelona Barcelona España (2008).

[4] Hiderink J., Broenink J., Bakkers A.: Communicating Threads for Java: University of Twente: Draf-Rev 5.: Netherlands (2000).

[5] Hiderink J., Broenink J., Vervoort W., Bakkers A.: Communicating Java Threads. University of Twente: Netherlands (2000).

[6] Welch P., McEwan A., Schneider S., Ifill W.: Integrating and Extending JCSP: Communicating Process Architectures. IOS Press, (2007).

[7] Fujimoto: Parallel and Distributed Simulation Systems: Wiley-Interscience: USA, (2000).

[8] Palma J.T., Garrido M. C., et al: Programación Concurrente: Thomson. España (2003).

[9] Capel I. M., Rodriguez V. S: Sistemas Concurrentes y Distribuidos. Teorìa y Pràctica: Copycentro Editorial: España (2012).

[10] Rossainz M., Sánchez B., Rangel A., Ballinas A.L., JPMI: Un paquete de clases en Java para la Programación Paralela con Paso de Mensajes.

Modelado, TIC y Sistemas Distribuidos: avances y aplicaciones. Dirección General de Publicaciones. BUAP, México (2019).
[11] Robbins, K. A., Robbins S.: UNIX Programación Práctica. Guía para la concurrencia, la comunicación y los multihilos. Prentice Hall. (1999).
[12] Roosta, Séller: Parallel Processing and Parallel Algorithms. Theory and Computation. Springer (1999).
[13] Rossainz M., Capel M., Carrasco O., Hernández F., Sánchez B. Implementation of the Pipeline Parallel Programming Technique as an HLPC: Usage, Usefulness and Performance. Annals of Multicore and GPU Programming. Volume 4, Number 1, Spain (2018).
[14] Corradi A., Leonardi L.: PO Constraints as tools to synchronize active objects. Pp: 42-53. Journal Object Oriented Programming 10. (1991).
[15] Danelutto, M.; Orlando, S; et al.: Parallel Programming Models Based on Restricted Computation Structure Approach. Technical Report-Dpt. Informatica. Universitá de Pisa (1999).
[16] Rossainz M., Pineda I., Dominguez P.: Análisis y Definición del Modelo de las Composiciones Paralelas de Alto Nivel llamadas CPANs. Modelos Matemáticos y TIC: Teoría y Aplicaciones. Dirección de Fomento Editorial. ISBN 987-607-487-834-9. Pp. 1-19. México. (2014).
[17] Rahman A., Bawiec M., Simon Says. Amazon Web Services. (2023). Recuperado de: https://aws.amazon.com/es/deeplens/community-projects/deeplens_simon_says/
[18] Vallejo D., Martín C. Desarrollo de Videojuegos. Un Enfoque Práctico. Volumen 1. Arquitectura del Motor. Creative Commons License. España (2015).

Estimación de Chl-a en entornos muy antropizados mediante aprendizaje automático y teledetección

José G. Giménez[1], Virginia C. Sánchez[1], Juan-Carlos Cano[1], Carlos T. Calafate[1] y José M. Cecilia[1]

Resumen— **Las lagunas costeras son ecosistemas de gran valor socioeconómico y ambiental. Sin embargo, están sometidas a grandes presiones antropogénicas y ambientales, principalmente debidas al cambio climático, que amenazan su sostenibilidad. Los sistemas de monitorización espacial y temporal de alta resolución son obligatorios para (1) identificar estas amenazas, (2) entender los principales problemas que afectan a estos ecosistemas, y (3) predecir cómo se comportarán estos ecosistemas en el futuro. En este artículo, se presenta un sistema de control basado en el servicio europeo de teledetección Copernicus que permite la monitorización diaria de la clorofila-a (Chl-a) para la laguna del Mar Menor (Sureste de España). Además, se analizan varios modelos de aprendizaje automático (Machine Learning) para adaptar los datos recogidos al contexto particular del Mar Menor, poco profundo y altamente salino. Los resultados son satisfactorios ya que se logra obtener un modelo global con buenas precisiones, concretamente con un valor de R^2 de 0'9 y 0'75 mg/m^3 de Error Medio Absoluto (MAE). Asimismo, este modelo es capaz de describir las floraciones algales que provocan picos de concentración de Chl-a.**

Palabras clave— **Clorofila-a; Teledetección por satélite; Aprendizaje automático; IoT**

I. Introducción

La teledetección mediante fuentes satelitales (SRS), basada en sensores embarcados en satélites que detectan y registran la energía reflejada o emitida, se perfila como un sistema cada vez más extendido para la monitorización de diferentes entornos naturales [1], [2]. Estos sistemas proporcionan imágenes de alta resolución con información geométrica y radiométrica de alta calidad. Además, a partir del procesado de estas imágenes se pueden obtener parámetros biogeofísicos [3], [4], [5], desarrollando diferentes productos para la observación de la tierra, océanos, etc. [6]. Los productos Sentinel [7] y NPP VIIRS [8] de la Agencia Espacial Europea (ESA) ofrecen resoluciones espaciales y temporales cada vez mejores, así como el desarrollo de nuevos servicios.

Estos productos basados en SRS que permiten generar datos biogeoquímicos se desarrollan a escala mundial y a menudo es necesario adaptarlos a contextos más locales o regionales. De hecho, puede haber grandes discrepancias entre los valores obtenidos por satélite y el estado real de las variables de superficie [9]. Esto implica que los datos generados por las herramientas satelitales no son útiles debido a las grandes desviaciones cuando se comparan con las mediciones *in situ* (es decir, datos de referencia). Esto no sólo reduce su utilidad en las aplicaciones, sino también la precisión de sus productos derivados. Por lo tanto, es esencial adaptar estos sistemas SRS para mejorar su fiabilidad en contextos locales/regionales.

En este artículo se proponen diferentes modelos de aprendizaje automático (Machine Learning) y aprendizaje profundo (Deep Learning) para corregir datos de satélite obtenidos del sistema de monitorización oceánica Coopernicus. En concreto, se centra en los datos de clorofila-a (Chl-a) generados a partir de Sentinel-3 (S3). Los modelos se aplican en el contexto regional de la laguna del Mar Menor (Murcia, España); un ecosistema local en riesgo de colapso debido a las altas presiones antropogénicas como la agricultura y el turismo. Chl-a es una medida indirecta de la salud de la laguna ya que debido a la proliferación masiva de algas produce una falta de oxígeno en el agua que provoca una elevada mortalidad de la fauna. Por lo tanto, este artículo se centra en la concentración de Chl-a como indicador crítico del estado de eutrofización de este ecosistema.

Se han propuesto varios trabajos para desarrollar sistemas de monitorización basados en SRS [10], [11], [12], [13] para esta variable. Sin embargo, se asumen la validez de los datos obtenidos por los sistemas satelitales, centrándose en una o dos fuentes satelitales, principalmente productos derivados de Sentinel, sin profundizar en el proceso de validación y/o cobertura de los valores recogidos.

El proceso de modelización es crítico en el actual ecosistema único. Las razones son la baja cantidad y calidad de los datos disponibles *in situ*, el desajuste espacial entre estas mediciones y los píxeles de teledetección, el no tener en cuenta la heterogeneidad intrínseca de las superficies terrestres y oceánicas que liberan nutrientes en la laguna, las carencias y deficiencias de las teorías sobre el problema de la escala en la validación. Por lo tanto, las principales contribuciones de este artículo son las siguientes:

- Evaluar la precisión de los modelos ML y DL centrados en la concentración de Chl-a en el Mar Menor mediante teledetección.
- Encontrar un modelo global para el Mar Menor que permita estimar la concentración de Chl-a y así proporcionar una monitorización continua de la salud de la laguna además de un mejor conjunto de datos para la predicción de Chl-a. Este

[1]Dpto. de Ingeniería Informática, Universidad Politécnica de Valencia, Valencia, España, e-mails: jggimman@doctor.upv.es, {vcassan, jucano, calafate, jmcecilia}@disca.upv.es

modelo también será capaz de detectar picos de concentración de Chl-a.
- Desarrollar un clúster basado en profundidades.

El artículo se estructura de la siguiente manera. En primer lugar, se describen en la Sección II los materiales y métodos empleados. A continuación, en la Sección III, se muestra los resultados cuantitativos ofrecidos por todos los productos basados en SRS para la monitorización de la laguna costera del Mar Menor. Finalmente, en la Sección IV se presentan las conclusiones y futuras líneas de trabajo.

II. Materiales y métodos

Esta sección proporciona una descripción completa de los materiales y métodos utilizados en este artículo. En particular, se presenta brevemente el caso de estudio de seguimiento del Mar Menor, presentando los datos de referencia utilizados para la calibración del modelo. También son introducidas las fuentes de satélite utilizadas y los modelos objeto de estudio.

A. Laguna Costera Mar Menor

Una de las mayores lagunas costeras de Europa y la mayor de la Península Ibérica es el Mar Menor. Consta de 135 km^2 y se encuentra situada en Murcia (sureste de España). Esta laguna con una profundidad media de 3'6 m y máxima de 7 m, es relativamente poco profunda. Está aislada del Mar Mediterráneo por una barrera costera arenosa de 22 km de longitud denominada La Manga. Dicha barrera es atravesada por varios barrancos, lo cual determina la naturaleza semiconfinada de sus aguas y les confiere unas características únicas de salinidad y temperatura. Además de su importancia desde el punto de vista medioambiental, el Mar Menor es también un lugar clave para la economía de la Región de Murcia. Sus condiciones climáticas únicas y sus abundantes recursos naturales han atraído el turismo, los usos de recreativos y la pesca, así como la importancia de la agricultura para la economía local. La cuenca que drena el Mar Menor se designa como el Campo de Cartagena (CC) y es una extensa llanura de más de 1.200 km^2 con una red de arroyos efímeros que recogen las escasas pero intensas precipitaciones [14].

El Mar Menor se ha caracterizado por sus aguas oligotróficas y por su fuerte resistencia a la eutrofización. Las aguas transparentes de la laguna han sido históricamente su principal rasgo definitorio, pero durante los últimos diez años han aparecido tendencias eutróficas [15]. Los aportes con altas concentraciones de nutrientes a la laguna han crecido en las últimas décadas debido a los cambios en las prácticas agrícolas en el CC y a la introducción de cultivos de regadío intensivo. Varios estudios [16], [17] encontraron que la transferencia de flujo desde el acuífero del CC a la laguna del Mar Menor debió verse significativamente impactada por la entrada de nitratos y otros elementos agroquímicos procedentes de la fertilización. Estos aportes provocaron un aumento drástico de la contaminación en la laguna [18] y desencadenaron un proceso de eutrofización, con la consiguiente pérdida de calidad del agua [19]. En 2016, un evento de eutrofización extrema dio lugar a una fuerte floración de fitoplancton [20]. Esto produjo un cambio considerable en la calidad de las aguas, con un aumento sustancial de la turbidez, un cambio en el color del agua y una pérdida de transparencia, con una caída en la profundidad de visibilidad del disco de Secchi a menos de 1 m. Esto suscitó una gran preocupación entre los ecologistas y los que trabajan en la industria del turismo, con importantes repercusiones socioeconómicas [21]. En septiembre de 2019, los niveles de Chl-a aumentaron notablemente, superando incluso el pico observado en 2016. Esto fue debido al agua de escorrentía superficial de la cuenca de drenaje que desembocó en el Mar Menor como resultado de una gran tormenta. Los factores impulsores de la productividad primaria fueron los enormes volúmenes de nitrógeno, fósforo, materia orgánica y sedimentos drenados por dicha escorrentía [22]. En este escenario, se requiere una intervención temprana para reducir la entrada de nutrientes y otros contaminantes en la laguna. Así pues, la creación de un modelo preciso para el estudio de algunos parámetros medioambientales, como el clorofluorocarbono, puede ayudar a las autoridades a investigar las causas y tomar decisiones al respecto.

B. Datos in situ

El conjunto de datos utilizado para analizar los recolectados mediante la teledetección se han obtenido del Gobierno Regional de Murcia (CARM). Esta entidad promovió mediciones periódicas del Mar Menor desde 2016, cuando se hizo patente la degradación de la laguna. En la Tabla I se puede ver una pequeña descripción sobre los datos recogidos. El conjunto de

Tabla I: Seguimiento *in situ* proporcionado por la CARM.

Fuente de datos *in situ*	**CARM**
Fecha de inicio del muestreo	2016-08-02
Fecha de fin del muestro	2022-02-24
Muestras	3.251

datos proporciona medidas casi semanales a diferentes profundidades para doce puntos repartidos a lo largo del Mar Menor (véase la Figura 1). Estos puntos se denominan puntos de medidas *in situ* (ISMP) y representan la heterogeneidad de la laguna, teniendo diferentes características como la profundidad, y distancias a diferentes puntos de interés como la orilla del mar y las ramblas. Entre los datos suministrados por el CARM, se incluyen Chl-a, turbidez, CDOM (Chromophoric Dissolved Organic Matter), oxígeno, salinidad y pH. Este artículo se centra en la Chl-a, ya que está fuertemente relacionado con los episodios de anoxia. La Tabla II muestra los valores de los principales descriptores estadísticos aplicados sobre el conjunto de datos. CARM provee datos filtrados, por lo que no es necesario realizar una búsqueda más limpia ni de valores atípicos. Además, en este artículo los experimentos gestionan los datos por profundidades, lo que significa que los modelos se ajustarán utilizando diferentes profundidades, pero

nunca mezclándolas. Así, las medidas tomadas entre 0 y 1 metro de profundidad aparecen en este artículo como profundidad 0, las tomadas entre 1 y 2 metros son profundidad 1, y así sucesivamente.

Fig. 1: Puntos de seguimiento *in situ* (ISMP) realizados por la Región de Murcia (CARM) en el Mar Menor.

Tabla II: Descripción estadística de los datos de Chl-a proporcionados por CARM.

	Profundidad (m)	**Chl-a** (mg/m^3)
Total valores	6146	6146
Media	2'225	2'651
Desviación típica	1'66	3'865
Valor mínimo	0	0'022
Primer cuartil	1	0'726
Segundo cuartil	2	1'367
Tercer cuartil	4	2'964
Valor máximo	6	28'112

C. Datos SRS

Los datos SRS se están popularizando en los sistemas de vigilancia en tiempo casi real (NRT) para hacer frente a los retos naturales y sociales, como arrojar luz sobre los desafíos globales y regionales, apoyando así una serie de iniciativas (por ejemplo, el Marco de Sendai, el Acuerdo de París, los Objetivos de Desarrollo Sostenible) [23], [24], [25].

En este artículo se analizan los datos de SRS de acceso público proporcionados por los productos satelitales de la ESA. El Servicio Marino Europeo Copernicus (CMS) que proporciona información autorizada gratuita, regular y sistemática sobre el estado del océano Azul (físico), Blanco (hielo marino) y Verde (biogeoquímico), a escala global y regional. Entre los diversos productos que Copernicus ofrece a través de sus diferentes plataformas, este artículo se centra en los datos de Sentinel-3 A y B (S3-AB). La Tabla III describe las misiones S3. Ambos satélites llevan a bordo el espectrómetro de imagen Ocean and Land Colour Instrument (OLCI) que mide la radiación solar reflejada por la Tierra en 21 bandas espectrales con una resolución espacial de 300 m. Es importante señalar que este artículo se centra en estos productos relacionados con S3 porque ofrecen datos diarios de la misma región del mundo alrededor de la misma hora (ver Tabla III) y tener una alta resolución temporal es esencial para el desarrollo de un sistema de vigilancia. Los datos basados en Sentinel se procesan y se dividen en productos dependiendo de la fase del proceso. Así, OLCI ofrece tres niveles de procesamiento diferentes. Este artículo se centra en los datos de nivel 2 (L2) que proporcionan la concentración de Chl-a en el agua medida en mg/m^3. El producto OLCI L2 también proporciona datos adicionales como reflectancias, espesor óptico de aerosoles (AOT), coeficiente de atenuación difusa (Kd940), por mencionar algunos. A pesar de que estos parámetros no se consideran en este artículo, son utilizados por el algoritmo de la ESA para calcular los datos de Chl-a proporcionados. Otra característica importante de los productos OLCI es la resolución por ello es utilizado el producto Water Full Resolution (WFR) que proporciona los datos relacionados con las masas de agua a 300 metros de resolución.

En cuanto a la granularidad temporal, la herramienta de descarga del CMS selecciona los archivos por tiempo, que indica el periodo desde que se adquirió la imagen hasta que se hace pública. En particular, se centra en los archivos NTC (tiempo no crítico) que se refieren a los productos entregados en las 24/48 horas siguientes a la adquisición de datos por satélite, en contraste con los NRT que se refieren a los productos entregados menos de 3 horas después de la adquisición. Los productos NTC contienen datos ajustados y cotejados utilizando información de otras plataformas, por lo que parece más preciso utilizar este en el presente análisis. También es importante señalar que se considera el mejor escenario posible para la cobertura de datos proporcionada por fuentes SRS; es decir, datos NTC en lugar de NRT. De todos modos, un sistema de vigilancia podría proporcionar los datos en NRT y luego reescribirlos con sus datos NTC cuando estén disponibles.

Para obtener los datos históricos de Chl-a procesados de S3-AB, se emplea CREODIAS [26]; una plataforma que contiene los datos generados por Sentinel, Envisat, Lansat y otros satélites EODATA. Su diseño permite a terceros usuarios crear prototipos y construir sus propios servicios y productos de valor añadido. Este servicio permite a los programadores solicitar fácilmente a Sentinel-Hub los produc-

Tabla III: Principales características de los satélites.

Satélite / Instrumento	**Hora de revisión** (día)	**Hora de exploración** (UTM)*	**Resolución en nadir** (m)	**Algoritmo Chl-a**	**Inicio de la misión** (año)	**Fin de la misión** (año)
Sentinel-3A OLCI	1	10:10 ± 1h	∼300	Chl_nn Chl_nomc	2016	2023
Sentinel-3B OLCI	1	10:10 ± 1h	∼300	Chl_nn	2018	2025

tos anteriores de Sentinel. Para este caso, la API devuelve un conjunto de archivos netCDF4 que contienen parámetros relacionados con Chl-a, coordenadas GPS, reflectancias lumínicas, absorción lumínica de la atmósfera, etc.

Los productos basados en Sentinel utilizan dos algoritmos diferentes para obtener la concentración de Chl-a a partir de esta información bruta. Se centra en el Modelo de Transferencia Radiactiva Inversa-Red Neural (IRTM-NN) para estimar a partir de la reflectancia normalizada de la lámina de agua en las bandas OLCI b440 a b674, b709, θs y $\Delta\phi$ (denominada *Chlor_nn*). A diferencia de otros algoritmos de estimación de la Chl-a, el método S3-NN proporciona datos incluso cuando algunas reflectancias proporcionadas en el L2 son negativas. Por tanto, el algoritmo IRTM-NN puede tratar con estos valores, algo que el algoritmo clásico no permite y que obliga a descartar los valores negativos. Esta consideración se toma para hacer coincidir el mayor número posible de días entre los datos S3 y los datos *in situ* con el fin de obtener un conjunto de datos más amplio para poder entrenar y probar los modelos. Del mismo modo, se considera que las reflectancias gemelas S3 A-B proporcionan información de reflectancia similar ya que utilizan el mismo instrumento OLCI y los mismos procesos para obtener los productos L2, por lo que los datos son fusionados de ambas fuentes en un único conjunto de datos.

Cabe mencionar que IRTM-NN se basa en redes neuronales para abordar el problema como un método de regresión múltiple no lineal que reduce drásticamente el tiempo computacional, una vez que la red está adecuadamente entrenada para ajustar los coeficientes. Afortunadamente, CMS ya proporciona el modelo IRTM-NN que produce propiedades ópticas inherentes al agua como el coeficiente de retrodispersión total (BBP443), el coeficiente de absorción total (ATOT443), el coeficiente de absorción de fitoplancton (APH443), el coeficiente de absorción de Material Detrital y Disuelto coloreado (ADG443), la concentración de Chl-a, la concentración Total de Materia Suspendida (TSM). Estos productos S3 también proporcionan un valor de bandera para cada píxel que indica si es válido el tipo de terreno: agua, tierra o nieve. Se remite al lector a [27] para más información. En cualquier caso, en este artículo todos los datos utilizados y asociados a cada ISMP son clasificados por S3 como agua. Los no válidos han sido descartados.

Un análisis inicial muestra que los valores de S3 Chl-a tienen una correlación media con los datos CARM *in situ*. La correlación depende del CARM, lo que significa que el S3 no proporciona resultados fiables para puntos aleatorios. Además, en todo el conjunto de datos Sentinel no hay picos de Chl-a, lo cual implica que su modelo no es capaz de detectar las floraciones de algas que se han producido varias veces durante el período de estudio. En la Figura 2 que toma como referencia los datos de ISMP 6 se puede ver la diferencia de los datos *in situ* y los de Sentinel, así como la no existencia de estos picos para los datos de Sentinel.

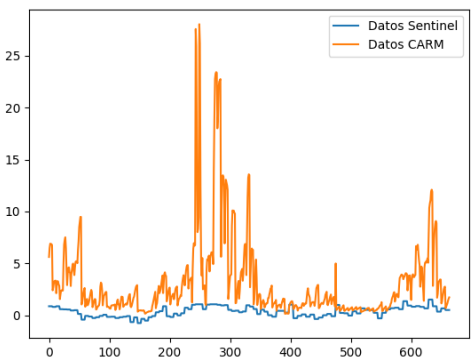


Fig. 2: Comparativa de los datos *in situ* y Sentinel de manera gráfica.

D. Modelos ML para la previsión de Chl-a

La predicción de Chl-a se considera en este artículo como un problema de regresión utilizando los datos del satélite como entrada y medidas *in situ* como salida. El objetivo es encontrar un modelo específico para ajustar los datos SRS obtenidos a través del sistema Copernicus a las medidas de Chl-a obtenidas para el Mar Menor. Se propone el uso de diferentes modelos/algoritmos ML para analizar cuál de ellos proporciona los mejores resultados. Los seleccionados son:

- Regressor Random Forest (RFR): es una técnica de aprendizaje supervisado, basada en un conjunto de árboles de decisión. Para evitar el sobreajuste, RFR utiliza la técnica de bagging. Así, cada árbol se construye sobre diferentes porciones de los datos, ningún árbol ve todos los datos de entrenamiento. Esto implica que cada árbol se entrena con diferentes muestras de datos para el mismo problema. De manera que al combinar sus resultados, unos errores son compensados por otros y se obtiene una predicción que generaliza mejor [28].
- Decision Tree Regressor (DT): son un subtipo de árboles de predicción que se aplican cuando

la variable respuesta es continua. En el entrenamiento de un árbol de decisión en regresión, las instancias, se distribuyen a través de los nodos generando la estructura de árbol hasta llegar a un nodo terminal, la condición para generar o dividir un nodo suele ser la Suma Residual de Cuadrados (RSS). Cuando se quiere predecir una nueva instancia, se recorren los nodos del árbol en función de los atributos de la nueva observación hasta llegar al nodo terminal. Cuando se alcanza este, la salida es la media del atributo de salida de ese nodo de las instancias de entrenamiento [29].

- K-Nearest Neighbors Regressor (KNN): es un método basado en instancias que trata de clasificar o predecir una nueva instancia basándose en las más cercanas a ese valor. Para ello, utiliza diferentes medidas de distancia, en función de los tipos de atributos que componen la instancia. En el caso de la predicción por regresión, el valor devuelto suele ser la media de los valores de los atributos de salida de los k vecinos más cercanos a la instancia dada. Cabe mencionar que el método KNN no requiere entrenamiento, ya que el modelo se crea al mismo tiempo que se infiere una nueva instancia [30].
- Multi-layer Perceptron Regressor (MLP): es un algoritmo de aprendizaje supervisado que aprende de un aproximador de función no lineal para clasificación o regresión. Un número de capas ocultas situadas entre las capas de entrada y salida son el verdadero motor de esta técnica. Las neuronas MLP se entrenan con el algoritmo de aprendizaje de retropropagación. Están diseñadas para aproximar cualquier función continua y pueden resolver problemas que no son linealmente separables [31].
- Convolutional Neural Network (CNN): son un tipo de red neuronal que se compone de una arquitectura de red de DL que aprende directamente de los datos, sin necesidad de extraer características manualmente. En general, las CNN aplicadas en estudios de teledetección se utilizan para analizar imágenes. No obstante, en este artículo se emplea otro enfoque utilizando una capa convolucional 1D. Basamos esta decisión en la relación entre reflectancias adyacentes, que permite a la CNN encontrar la relación adecuada. Entonces una capa encontrará la relación similar a trabajos que prueban diferentes algoritmos clásicos buscando esta relación y sus coeficientes [32].

III. Evaluación y discusión

A. Montaje del experimento

El conjunto de datos utilizado en este artículo contiene los datos de emparejamiento S3-AB e ISMP por día y coordenada. Partiendo de estos, se generan varios conjuntos de datos por ISMP y profundidades que contienen las reflectancias S3-AB y las medidas CARM de Chl-a. El valor de salida es la Chl-a en mg/m^3 y los parámetros de entrada son las reflectancias obtenidas mediante la teledetección. La Tabla IV resume el número de instancias disponibles para cada conjunto de datos.

Tabla IV: Número de instancias por ISMP y profundidad.

	Profundidad (m)						
ISMP	**0**	**1**	**2**	**3**	**4**	**5**	**6**
1	87	87	87				
2	105	105	105	105	105	98	
3	108	108	108	108	108	108	42
4	112	112	112	112	112	30	4
5	102	102	102	102	102	87	
6	103	103	103	103	103	103	46
7	108	109	109	108	106		
8	103	103	103	103	102		
9	96						
10	98	99	99	99	99		
11	104	104	105	103	54		
12	101	101	101	101	101	98	

Para evaluar la precisión de los modelos presentados aquí se utilizan las siguientes estadísticas:

- Coeficiente de determinación (R^2): es la proporción de la varianza total de la variable explicada por la regresión. Esta medida refleja el ajuste de un modelo a la variable que pretende explicar, cuanto más se acerque al valor a uno, mejor será el ajuste del modelo. Esta medida se calcula como:

$$R^2 = \frac{\sigma_{yp}^2}{\sigma_y^2 * \sigma_p^2},$$

  siendo σ_{yp} la varianza de la variable real y de la variable predicha p, σ_y la varianza de y (valor real) y σ_p la varianza de p (valor predicho).
- Error cuadrático medio (MSE): es una medida derivada del cuadrado de la distancia euclídea, es siempre un valor positivo que disminuye a medida que el error se aproxima a cero. Se calcula como:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - p_i)^2,$$

  siendo n el número de instancias, y_i el valor real y p_i el valor predicho.
- Error medio absoluto (MAE): se utiliza para cuantificar la precisión de una técnica de predicción comparando los valores predichos frente a los observados. Se calcula como:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - p_i|,$$

  siendo n el número de instancias, y_i el valor real y p_i el valor predicho.

Se han realizado un total de cuatro experimentos con el objetivo de obtener un modelo adecuado. Inicialmente se utilizan técnicas de ML, pero también el modelo CNN puede mejorar los resultados. Para los experimentos, los algoritmos se ajustan utilizando la mejor optimización de parámetros para obtener los modelos que mejor se ajusten.

A continuación, se resumen los pasos que se siguen para encontrar el modelo:

- El primer experimento realiza una validación cruzada quíntuple aleatoria sobre el algoritmo ML propuesto. Evalúa un modelo local para cada ISMP y profundidad.
- El segundo experimento realiza una validación cruzada de cinco veces aleatoriamente sobre ML utilizando todos los ISMP indistintamente para encontrar un modelo global. Las profundidades se tratan siempre por separado.
- Los experimentos anteriores sugieren la existencia de un clúster. Por lo tanto, se repiten los anteriores excluyendo los ISMP para evaluar un modelo por clúster.

Finalmente se vuelven a realizar todos los experimentos utilizando una CNN Conv1D.

B. Resultados y discusión

B.1 Creación de un modelo local ML para ISMP

Este primer experimento tiene como objetivo entrenar modelos ML para cada estación individualmente. Debido a la heterogeneidad del Mar Menor, cada punto de monitorización (es decir, ISMP) puede tener características diferentes que pueden beneficiarse de la modelización mediante métodos ML. Por lo tanto, la hipótesis principal en esta sección es que cada ISMP requiere un modelo ML diferente para ser entrenado para este punto en particular. Con el objetivo de obtener los mejores resultados se configuran modelos ML utilizando diferentes parametrizaciones, sin embargo no se profundiza en esta parte para no alargar el texto y porque CNN resulta ser el mejor modelo. La Tabla V presenta los mejores resultados para cada ISMP y la profundidad donde es más preciso e indicando la mejor técnica ML.

En la selección de los mejores resultados, se ha tomado como referencia el R^2 más alto. En este primer experimento es importante señalar que los modelos ML para las ISMP 1 y 9 no obtienen buenos resultados, teniendo valores de R^2 en 0 y grandes valores de error tanto para MAE como para MSE. De hecho, estos ISMP son los más someros del Mar Menor y, por tanto, las reflectancias procedentes de teledetección introducen muchos errores, proporcionando una gran disparidad de valores en comparación con la monitorización *in situ*.

Otro aspecto relevante que destacar es que para todas las ISMP excepto la 5, los mejores resultados se obtienen a profundidades someras de 0 a 1 metro o de 1 a 2 metros. En el caso del ISMP 5, los mejores resultados se obtienen a profundidades de 4 a 5 metros. Esto tiene sentido, debido a que está situado al otro lado de la laguna (véase la Figura 1).

Por último, no hay ningún modelo que sea mejor que los demás para todos los ISMP. Las técnicas con mejores resultados son MLP y RFR para 5 ISMP diferentes y DT y KNN para un ISMP diferente. Aunque los resultados de la mayoría de los ISMP, excluidos el 1 y el 9, son satisfactorios, la falta de un único modelo de predicción crea problemas y complejidad para el sistema de predicción. De ahí la necesidad de buscar un modelo global para Chl-a en el Mar Menor. Por lo tanto, a continuación se prueban las diferentes técnicas de ML para todos los ISMP y todas las profundidades.

Tabla V: El modelo de mejor rendimiento para cada estación y su profundidad en la que R^2 es máximo para predecir Chl-a. MAE y MSE se miden en mg/m^3.

ISMP	**Técn.**	**Prof.**(m)	$\mathbf{R^2}$	**MAE**	**MSE**
1	DT	0	0'0	1'526	12'919
2	MLP	0	0'863	1'141	3'473
3	MLP	0	0'762	1'132	4'171
4	MLP	1	0'808	1'1	2'75
5	MLP	4	0'808	1'1	2'75
6	RFR	1	0'864	0'932	2'88
7	MLP	0	0'79	0'943	3'966
8	RFR	0	0'877	0'84	1'589
9	KNN	0	0'0	4'682	53'934
10	RFR	0	0'738	0'919	4'481
11	RFR	1	0'861	1'047	3'1
12	RFR	1	0'887	0'965	2'823

B.2 Creación de un modelo ML global

En este segundo experimento, los ISMP ya no se consideran de forma aislada, por lo que se crea un modelo de predicción global para cada técnica ML. Para estos modelos globales, se crea un nuevo conjunto de datos con datos de todos los ISMP y profundidades juntos. La Tabla VI muestra, para los 4 primeros niveles de profundidad, los resultados de los modelos globales para cada técnica ML. En este experimento se descartan las profundidades superiores a 4 m debido a la baja precisión obtenida en el experimento anterior.

La Tabla VI muestra que el mejor resultado obtenido es con la técnica KNN en profundidades entre 1 y 2 metros para la predicción de Chl-a. El valor R^2 es 0'6919 y el MAE y MSE son 1'1614 y 4'9211 mg/m^3 respectivamente. La siguiente técnica con mejores resultados es DT a la misma profundidad con un R^2 de 0'6707. Las técnicas RFR y MLP son las que obtienen peores resultados, al contrario que los modelos locales. Esto se debe a que las técnicas DT y KNN se ven menos afectadas por el ruido introducido por los ISMP 1, 5 y 9. Para verificar esta teoría de los datos ruidosos, se realiza el mismo experimento utilizando una validación cruzada de 5 veces, pero sin incluir los datos de estos ISMP. Como ya se ha mencionado, los ISMP 1 y 9 tienen datos muy poco profundos y una pequeña cantidad de datos. También se descarta el ISMP 5 porque los mejores resultados se obtuvieron en los modelos locales a grandes profundidades y además, al estar fuera de la laguna, los datos de satélite pueden aportar posible ruido en los resultados.

B.3 Creación de modelo global por agrupación

Teniendo en cuenta las razones anteriores, se supone la existencia de un clúster relacionado con la profundidad máxima de la ISMP. Así, se puede considerar inicialmente dos clústeres: (1) Clúster-0 menos de 2 metros de profundidad (ISMP 1 y 9), (2) Clúster-1 más de 2 metros de profundidad. Así, la

Tabla VI: El modelo de mejor rendimiento para cada estación y su profundidad en la que R^2 es máximo para predecir Chl-a. MAE y MSE se miden en mg/m^3.

Técnica	**Profundidad**(m)	$\mathbf{R^2}$	**MAE**	**MSE**
DT	0	0'553	1'249	7'481
	1	0'671	1'217	6'237
	2	0'264	1'582	10'101
	3	0'266	1'57	8'386
KNN	0	0'662	1'159	6'528
	1	0'692	1'161	4'921
	2	0'48	1'326	5'847
	3	0'409	1'519	8'036
MLP	0	0'587	1'344	6'091
	1	0'596	1'351	7'561
	2	0'578	1'397	7'224
	3	0'526	1'391	5'894
RFR	0	0'644	1'251	6'456
	1	0'631	1'148	5'562
	2	0'61	1'212	5'093
	3	0'555	1'309	4'956

precisión de los modelos ML para el Clúster-1 mejora en general, como se muestra en la Tabla VII. Para las técnicas DT y KNN, la mejora es sólo de unas 3 ó 4 centésimas de ajuste en la referencia al valor R^2, y los errores MAE y MSE disminuyen ligeramente o permanecen iguales. Sin embargo, las técnicas MLP y RFR obtienen resultados mucho mejores, alcanzando valores de ajuste R^2 de aproximadamente 0'8 y errores MAE y MSE de aproximadamente 1 mg/m^3 a profundidades de 0 a 1 metro. La Figura 3 muestra el gráfico de dispersión para RFR con los datos de profundidad 0 en ISMP 6. Este modelo es capaz de detectar cambios de tendencia en los datos de Chl-a, de hecho, RFR introduce el mejor modelo capaz de detectar estos picos.

De estos experimentos de ML se extraen varias conclusiones. La primera es un posible clúster basado en la profundidad de ISMP. También que es posible descartar ISMP 5 ya que se encuentra fuera de la laguna. Por último, los modelos ML globales se ajustan ligeramente peor que los locales. A pesar de ello, los modelos locales no son homogéneos y complican el sistema. Este hecho justifica probar otras técnicas como DL para encontrar el mejor modelo.

Tabla VII: Precisión de los modelos a diferentes profundidades para predecir Chl-a en Clúster-1. MAE y MSE se miden en mg/m^3.

Técnica	**Profundidad**(m)	$\mathbf{R^2}$	**MAE**	**MSE**
DT	0	0'639	1'159	5'733
	1	0'692	1'239	5'776
	2	0'629	1'436	6'458
	3	0'488	1'45	6'417
KNN	0	0'765	1'103	5'041
	1	0'682	1'303	6'215
	2	0'558	1'402	6'882
	3	0'451	1'381	6'077
MLP	0	0'772	1'041	4'012
	1	0'75	1'267	5'543
	2	0'677	1'334	5'829
	3	0'556	1'347	5'790
RFR	0	0'833	0'964	3'647
	1	0'81	1'0783	3'969
	2	0'724	1'296	5'163
	3	0'548	1'314	5'769

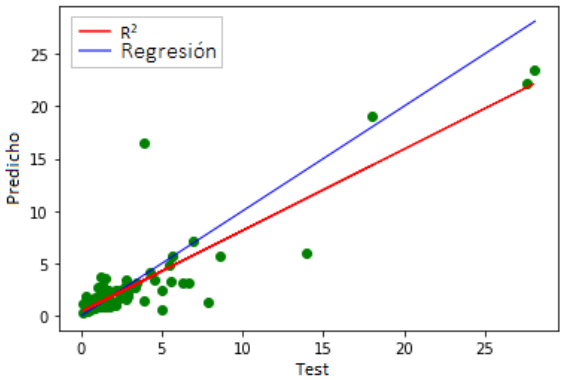


Fig. 3: Diagrama de dispersión CNN para la profundidad 0 en ISMP 6.

B.4 Uso del aprendizaje profundo

A pesar de que la precisión de los modelos ML no es mala, es supuesto que un modelo DL podría funcionar mejor, también sin un clustering por profundidad. Por lo tanto, se repiten los pasos anteriores pero utilizando CNN Conv1D. Aunque esta técnica se utiliza principalmente para la clasificación o detección de imágenes, es sugerido otro enfoque. Para encontrar la relación entre reflectancias adyacentes se establece un kernel de ventanas a 4 para construir la capa. Inicialmente la CNN presenta un buen comportamiento, pero para estaciones aisladas los resultados no son muy significativos comparados con los modelos ML. De forma similar a los modelos ML, la mejor precisión se alcanza utilizando la profundidad 0 de CARM. Pese a que, muestra una mejor precisión que ML cuando utiliza todos los ISMP, es decir, cuando CNN se entrena para predecir los valores de un ISMP completo utilizando el resto de ISMP, las tablas muestran de nuevo como el comportamiento de los ISMP 1, 5 y 9 no tienen relación con el resto. Por lo tanto, la agrupación por profundidad parece necesaria y también justifica la exclusión de ISMP 5.

Teniendo en cuenta estos resultados, se busca un modelo global intentando predecir aleatoriamente cualquier valor de ISMP en el Clúster-1. Así, entrenando el modelo con el 80 % de los datos y utilizando una validación cruzada de 5 proporcionada por el modelo los resultados son R^2 con un valor de 0'89, MSE de 1'95 mg/m^3 y MAE de 0'78 mg/m^3, lo cual muestra un buen rendimiento que supera a los modelos ML anteriores.

Además, la Figura 4 muestra el comportamiento de la CNN en los picos de Chl-a en profundidad 0 en ISMP 6, cuando se producen los fenómenos de floración de algas, lo que indica que este modelo puede representar con bastante exactitud el comportamiento de Chl-a en el Mar Menor.

IV. Conclusiones

Gracias a su ubicuidad, la teledetección ofrece una herramienta muy poderosa para vigilar grandes áreas naturales. Las mejoras en la instrumentación de estos sistemas ofrecen una resolución espacial y temporal cada vez menor, lo que brinda la oportunidad de de-

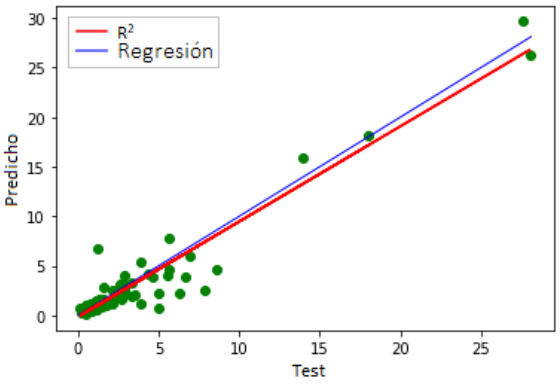


Fig. 4: Diagrama de dispersión RFR para la profundidad 0 en ISMP 6.

Tabla VIII: Resultados de la técnica CNN a profundidad 0 utilizando un modelo global para predecir el Chl-a de toda la estación. MAE y MSE se miden en mg/m^3.

ISMP	R^2	MSE	MAE
1	0'00	15'43	2'08
2	0'90	2'58	0'94
3	0'85	2'60	0'92
4	0'88	1'72	0'74
5	0'00	0'83	0'59
6	0'91	1'80	0'75
7	0'89	2'00	0'77
8	0'90	1'27	0'69
9	0'00	48'82	3'86
10	0'76	4'18	0'88
11	0'87	2'14	0'90
12	0'87	2'81	0'80

sarrollar herramientas de alerta para la conservación de espacios naturales. Sin embargo, los sistemas de satélite son globales y sus señales necesitan ser procesadas para aumentar su precisión en contextos más locales/regionales.

En este artículo se han analizado diferentes modelos de ML y DL para mejorar la precisión de las medidas de Chl-a a partir de las reflectancias del sistema de observación europeo Sentinel-3 en un contexto particular altamente antropizado como es la laguna del Mar Menor. Los resultados muestran que los datos de reflectancias de Sentinel-3 tienen una buena relación con las medidas *in situ* proporcionadas por CARM. Esta buena correlación sugiere que utilizando la teledetección es posible obtener una buena estimación de la concentración de Chl-a en el Mar Menor.

El modelo CNN empleado representa un modelo global que puede manejar bien la información de teledetección en este entorno. Por lo tanto, puede proporcionar un modelo único capaz de describir toda la laguna. No obstante, sigue existiendo una falta de precisión en puntos cercanos a la costa o poco profundos, por lo que habrá que buscar un método mejor para estimar los valores en estos casos. Este hecho parece chocar con las conclusiones de que los mejores resultados del modelo se producen utilizando mediciones en superficie, pero se debe al fenómeno de la reflectancia de la luz en el agua y está fuera del alcance de la presente investigación. Además, sólo se ha tenido en cuenta Chl-a y otros parámetros como la turbidez no. Estos pueden incluirse en como parámetros de entrada para proporcionar un modelo multivariante de la laguna. Dichos parámetros están siendo estudiados para futuros trabajos así como probar el enfoque del modelo CNN en otros ambientes similares lo cual sugiere una dirección interesante.

Agradecimientos

Este trabajo ha sido apoyado por los proyectos TED2021-130890B, financiado por MCIN/AEI/10.13039/501100011033 y por la Unión Europea NextGenerationEU/PRTR, y la Beca Ramón y Cajal RYC2018-025580-I, financiada por MCIN/AEI/10.13039/501100011033, "FSE invierte en tu futuro" y "ERDF Una forma de hacer Europa".

Referencias

[1] Alex M Lechner, Giles M Foody, and Doreen S Boyd, "Applications in remote sensing to forest ecology and management," *One Earth*, vol. 2, no. 5, pp. 405–412, 2020.

[2] Yanbo Huang, Zhong-xin Chen, YU Tao, Xiang-zhi Huang, and Xing-fa Gu, "Agricultural remote sensing big data: Management and applications," *Journal of Integrative Agriculture*, vol. 17, no. 9, pp. 1915–1931, 2018.

[3] Jun Yang, Peng Gong, Rong Fu, Minghua Zhang, Jingming Chen, Shunlin Liang, Bing Xu, Jiancheng Shi, and Robert Dickinson, "The role of satellite remote sensing in climate change studies," *Nature climate change*, vol. 3, no. 10, pp. 875–883, 2013.

[4] DR Sowmya, P Deepa Shenoy, and KR Venugopal, "Remote sensing satellite image processing techniques for image classification: a comprehensive survey," *International Journal of Computer Applications*, vol. 161, no. 11, pp. 24–37, 2017.

[5] Kilian Vos, Kristen D Splinter, Mitchell D Harley, Joshua A Simmons, and Ian L Turner, "Coastsat: A google earth engine-enabled python toolkit to extract shorelines from publicly available satellite imagery," *Environmental Modelling & Software*, vol. 122, pp. 104528, 2019.

[6] CO Justice, JRG Townshend, EF Vermote, E Masuoka, RE Wolfe, Nazmi Saleous, DP Roy, and JT Morisette, "An overview of modis land data processing and product status," *Remote sensing of Environment*, vol. 83, no. 1-2, pp. 3–15, 2002.

[7] Zbyněk Malenovskỳ, Helmut Rott, Josef Cihlar, Michael E Schaepman, Glenda García-Santos, Richard Fernandes, and Michael Berger, "Sentinels for science: Potential of sentinel-1,-2, and-3 missions for scientific observations of ocean, cryosphere, and land," *Remote Sensing of environment*, vol. 120, pp. 91–101, 2012.

[8] Christopher O Justice, Miguel O Román, Ivan Csiszar, Eric F Vermote, Robert E Wolfe, Simon J Hook, Mark Friedl, Zhuosen Wang, Crystal B Schaaf, Tomoaki Miura, et al., "Land and cryosphere products from suomi npp viirs: Overview and status," *Journal of Geophysical Research: Atmospheres*, vol. 118, no. 17, pp. 9753–9765, 2013.

[9] Xiaodan Wu, Qing Xiao, Jianguang Wen, Dongqin You, and Andreas Hueni, "Advances in quantitative remote sensing product validation: Overview and current status," *Earth-Science Reviews*, vol. 196, pp. 102875, 2019.

[10] Manuel Erena, José A Domínguez, Felipe Aguado-Giménez, Juan Soria, and Sandra García-Galiano, "Monitoring coastal lagoon water quality through remote sensing: The mar menor as a case study," *Water*, vol. 11, no. 7, pp. 1468, 2019.

[11] Isabel Caballero, Mar Roca, Juan Santos-Echeandía, Patricia Bernárdez, and Gabriel Navarro, "Use of the sentinel-2 and landsat-8 satellites for water quality monitoring: An early warning tool in the mar menor coastal lagoon," *Remote Sensing*, vol. 14, no. 12, pp. 2744, 2022.

[12] Diego Gómez, Pablo Salvador, Julia Sanz, and José Luis Casanova, "A new approach to monitor water quality in the menor sea (spain) using satellite data and machine learning methods," *Environmental Pollution*, vol. 286, pp. 117489, 2021.

[13] Javier González-Enrique, Juan Jesús Ruiz-Aguilar, Eduardo Madrid Navarro, Rosa Martínez Álvarez Castellanos, Ivan Felis Enguix, José M Jerez, and Ignacio J Turias, "Deep learning approach for the prediction of the concentration of chlorophyll *a* in seawater, a case study in el mar menor (spain)," in *International Workshop on Soft Computing Models in Industrial and Environmental Applications*. Springer, 2023, pp. 72–85.

[14] Javier Senent-Aparicio, Julio Pérez-Sánchez, José Luis García-Aróstegui, Alicia Bielsa-Artero, and Juan Carlos Domingo-Pinillos, "Evaluating groundwater management sustainability under limited data availability in semiarid zones," *Water*, vol. 7, no. 8, pp. 4305–4322, 2015.

[15] Héctor M Conesa and Francisco J Jiménez-Cárceles, "The mar menor lagoon (se spain): A singular natural ecosystem threatened by human activities," *Marine pollution bulletin*, vol. 54, no. 7, pp. 839–849, 2007.

[16] Javier Senent-Aparicio, Adrián López-Ballesteros, Anders Nielsen, and Dennis Trolle, "A holistic approach for determining the hydrology of the mar menor coastal lagoon by combining hydrological & hydrodynamic models," *Journal of Hydrology*, vol. 603, pp. 127150, 2021.

[17] Juan Carlos Domingo-Pinillos, Javier Senent-Aparicio, José Luis García-Aróstegui, and Paul Baudron, "Long term hydrodynamic effects in a semi-arid mediterranean multilayer aquifer: Campo de cartagena in south-eastern spain," *Water*, vol. 10, no. 10, pp. 1320, 2018.

[18] Salvador García-Ayllón, "New strategies to improve comanagement in enclosed coastal seas and wetlands subjected to complex environments: Socio-economic analysis applied to an international recovery success case study after an environmental crisis," *Sustainability*, vol. 11, no. 4, pp. 1039, 2019.

[19] Patricia Jimeno-Sáez, Javier Senent-Aparicio, José M Cecilia, and Julio Pérez-Sánchez, "Using machine-learning algorithms for eutrophication modeling: case study of mar menor lagoon (spain)," *International Journal of Environmental Research and Public Health*, vol. 17, no. 4, pp. 1189, 2020.

[20] Jesús M Mercado, Dolores Cortés, Francisco Gómez-Jakobsen, Candela García-Gómez, Sophia Ouaissa, Lidia Yebra, Isabel Ferrera, Nerea Valcárcel-Pérez, María López, Rocío García-Muñoz, et al., "Role of small-sized phytoplankton in triggering an ecosystem disruptive algal bloom in a mediterranean hypersaline coastal lagoon," *Marine Pollution Bulletin*, vol. 164, pp. 111989, 2021.

[21] Salvador Garcia-Ayllon, "The integrated territorial investment (iti) of the mar menor as a model for the future in the comprehensive management of enclosed coastal seas," *Ocean & Coastal Management*, vol. 166, pp. 82–97, 2018.

[22] Juan Manuel Ruiz-Fernandez, VM León, L Marín-Guirao, F Giménez-Casalduero, J Alvárez-Rogel, MA Esteve-Selma, R Gómez-Cerezo, F Robledano-Aymerich, G González-Barberá, and J Martínez Fernández, "Informe de síntesis sobre el estado actual del mar menor y sus causas en relación a los contenidos de nutrientes," *Projects of Sustainability and Conservation of Mar Menor Lagoon and Its Basin; Universidad de Alicante: Alicante, Spain*, 2019.

[23] Yvonne Walz, Annika Min, Karen Dall, Moses Duguru, Juan-Carlos Villagran de Leon, Valerie Graw, Olena Dubovyk, Zita Sebesvari, Andries Jordaan, and Joachim Post, "Monitoring progress of the sendai framework using a geospatial model: The example of people affected by agricultural droughts in eastern cape, south africa," *Progress in Disaster Science*, vol. 5, pp. 100062, 2020.

[24] Alan Grainger and Junwoo Kim, "Reducing global environmental uncertainties in reports of tropical forest carbon fluxes to redd+ and the paris agreement global stocktake," *Remote Sensing*, vol. 12, no. 15, pp. 2369, 2020.

[25] Steffen M Noe, Ksenia Tabakova, Alexander Mahura, Hanna K Lappalainen, Miriam Kosmale, Jyri Heilimo, Roberto Salzano, Mattia Santoro, Rosamaria Salvatori, Andrea Spolaor, et al., "Arctic observations and sustainable development goals–contributions and examples from era-planet icupe data," *Environmental Science & Policy*, vol. 132, pp. 323–336, 2022.

[26] Radek Malinowski, Stanisław Lewiński, Marcin Rybicki, Ewa Gromny, Małgorzata Jenerowicz, Michał Krupiński, Artur Nowakowski, Cezary Wojtkowski, Marcin Krupiński, Elke Krätzschmar, et al., "Automated production of a land cover/use map of europe based on sentinel-2 imagery," *Remote Sensing*, vol. 12, no. 21, pp. 3523, 2020.

[27] Nima Pahlevan, Brandon Smith, John Schalles, Caren Binding, Zhigang Cao, Ronghua Ma, Krista Alikas, Kersti Kangro, Daniela Gurlin, Nguyen Hà, et al., "Seamless retrievals of chlorophyll-a from sentinel-2 (msi) and sentinel-3 (olci) in inland and coastal waters: A machine-learning approach," *Remote Sensing of Environment*, vol. 240, pp. 111604, 2020.

[28] Leo Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[29] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone, *Classification and regression trees*, Routledge, 2017.

[30] Gavin Hackeling, *Mastering Machine Learning with scikit-learn*, Packt Publishing Ltd, 2017.

[31] Hind Taud and JF Mas, "Multilayer perceptron (mlp)," in *Geomatic approaches for modeling land change scenarios*, pp. 451–455. Springer, 2018.

[32] Shubhi Harbola and Volker Coors, "One dimensional convolutional neural network architectures for wind prediction," *Energy Conversion and Management*, vol. 195, pp. 70–75, 2019.

SkewEngine: Reorganización de mallas regulares para cálculo intensivo

Felipe Romero [1], Luis F. Romero [2], Gerardo Bandera[2]

Resumen— **En muchas aplicaciones, tales como la manipulación de datos hiperespectrales, la exploración de datos de resonancia magnética o la identificación de cuencas visuales en modelos digitales de elevación, es necesario realizar operaciones aritméticas sobre cada punto de una malla de datos que implican al resto de puntos de la misma, lo que puede resultar en un problema computacionalmente intratable. Se presenta en este trabajo SkewEngine, una herramienta diseñada para mejorar el rendimiento de cálculos intensivos en mallas regulares de datos 2D o 3D, como imágenes, volúmenes de datos multiespectrales o modelos digitales de elevación. SkewEngine soluciona este problema mediante la reorganización de la malla en memoria según una dirección espacial preferente, lo que permite una mayor eficiencia en la realización de cálculos intensivos. Se demuestra que SkewEngine ofrece una mejora significativa en la velocidad de los cálculos para una variedad de casos de prueba, lo que sugiere que puede ser una herramienta útil en una mucho más amplia gama de aplicaciones en las que se requiere procesamiento intensivo de datos en mallas regulares.**

Palabras clave— **Paralelismo embarazoso, Jerarquía de Memoria, Mallas estructuradas**

I. Introducción

EXISTEN problemas, en muchos y muy diversos campos, en los que es necesario realizar un cálculo extremadamente intensivo sobre cada uno de los puntos de una malla de datos 2D o 3D, como por ejemplo, sobre cada uno de los píxeles de una imagen, sobre cualquier punto de un mapa, o en los datos de una resonancia magnética.

En algunos casos, la intensidad aritmética es tan alta, que hace que el problema sea intratable, desde el punto de vista computacional. Veamos el ejemplo de la cuenca visual de un punto, en un modelo digital de elevaciones. Imagine que desea conocer qué parte de un territorio es visible desde un determinado lugar del territorio:

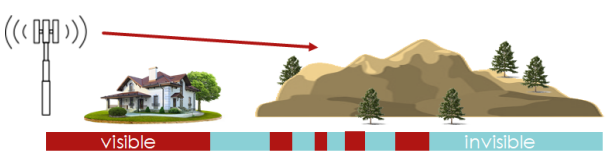


Fig. 1: Ejemplo de cuenca visual.

Como es obvio, para saber si desde un punto A se puede ver B (donde B puede ser cualquier otro punto de la geografía considerada), habría que tener en cuenta la altura del resto de puntos del modelo, ya que cualquiera, a priori, puede ser un obstáculo para la visión. Así, en un modelo digital de elevaciones, o DEM, (de *digital elevation model*), con $N = dimx \times dimy$ datos, la complejidad del problema, usando la notación big-O, sería de orden $O(N^2)$, aunque es evidente que dicha complejidad se puede reducir si sólo se consideran como obstáculos los puntos C que están en la línea A-B. Aún así, la complejidad del problema, $O(N^{1,5})$, es bastante alta. La conocida aplicación Google Earth, por ejemplo, tarda varios segundos en obtener un resultado aproximado.

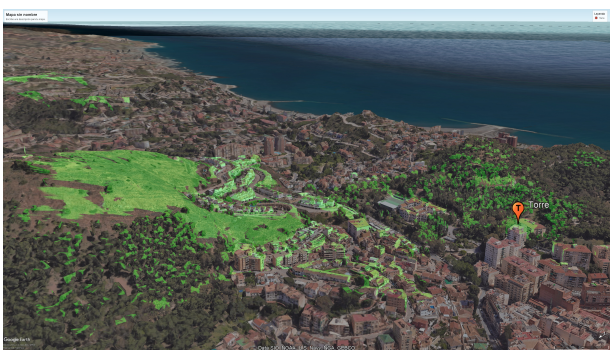


Fig. 2: Cuenca visual obtenida con Google Earth desde un punto de la ciudad de Málaga.

Cuando se desea calcular la cuenca visual no sólo ya de un observador situado en un lugar concreto de un territorio, sino la que se obtendría desde una trayectoria arbitraria que transcurre por el terreno, o la que se observaría desde una región, o incluso desde todo el territorio, donde cualquier punto de un DEM se convierte en observatorio. En este caso, la complejidad se dispara a $O(N^2{,}5)$. Incluso con baja precisión, se requieren meses de CPU para un cálculo de un modelo sencillo.

Afortunadamente, este tipo de problemas pertenece a una categoría en la que los parámetros de estudio están afectados por un decaimiento que depende de la distancia geométrica en la malla de datos, ya sea lineal o cuadrático, siendo esto último lo más frecuente. Son esos problemas en la que la influencia de un punto alejado (como puede ser una mariposa en Singapur), no afecta, o al menos inmediatamente, a lo que sucede en el otro extremo de la geometría (p.e., el Caribe). Este tipo de problemas se suele simplificar mediante un procesamiento en un conjunto discreto de direcciones radiales respecto a un punto de estudio, ya que los radios están más próximos en el punto de origen, y por tanto, la malla es más fina en el punto de estudio.

Sin embargo, en este tipo de problemas, donde la

[1]Dpto. Informática, CEiA3, Universidad de Almería, e-mail: fr@uma.es

[2]Dpto. Arquitectura de Computadores, Universidad de Málaga, e-mail: felipe@uma.es, gbandera@uma.es

localidad espacial de la información es un factor crítico, existe la necesidad de trasladar dicha localidad espacial a los propios datos que representan a la información, y muy en particular, al almacenamiento en memoria de los mismos. Veamos un ejemplo. Imagine que en la imagen de la Fig. 3 (izquierda) se desea aplicar un filtro en todas las direcciones paralelas a la de la flecha negra. Es evidente que cualquier algoritmo de cierta complejidad de los que suelen aplicarse en gráfica computacional, como la FFT, sería mucho más eficiente si los datos estuvieran alineados en memoria, como ocurriría en la imagen deformada de la derecha.

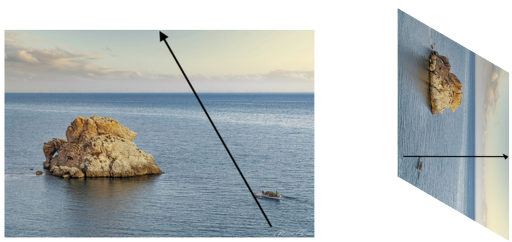


Fig. 3: Reorganización de datos con sesgo

La propuesta de este trabajo consiste en analizar y explotar los beneficios de una reorganización en memoria, pero sobre todo demostrar que, en problemas de muy elevada complejidad, la reorganización de la información mediante una interpolación sesgada de los datos es extremadamente beneficiosa.

A. Antecendentes: el algoritmo SDEM

En 2013, Tabik *et al* [1] publican un algoritmo que considera la dependencia radial en el cálculo de la cuenca visual, para crear un algoritmo que la calcula para todos los puntos de un modelo digital, en un conjunto discreto de direcciones alrededor de cada punto ($s = 360$, normalmente). Es decir, para cada sector angular de un grado, sólo tiene en cuenta los puntos en el eje central del sector, reduciendo la complejidad del problema de $O(N^{2,5})$ a $O(sN^{1,5})$.

```
for pov in 1,N //Point of View loop
    for s in 1,360
        viewshed[pov]= function(s, DEM)
```

Además, mediante una simple inversión de los lazos, el código aprovecha que todos los datos están ya alineados en una determinada dirección para hacer el cálculo de la cuenca visual desde todos los puntos de la línea en una determinada dirección:

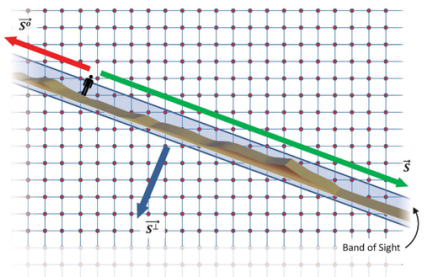


Fig. 4: Cálculo de la cuenca visual a todos los puntos (lazo interno) para un valor (s=22º) del lazo externo.

```
for s in 1,360
    for pov in 1,N
        viewshed[pov]= function(s, DEM)
```

En la siguiente figura, las imágenes de la fila interior muestran cómo, al intercambiar los lazos, se puede aprovechar para aplicar el algoritmo a todos los puntos que se encuentran alineados en la misma dirección del lazo exterior. En la fila superior, se han elegido cuatro puntos del territorio, y se ha calculado la cuenca visual en 4 direcciones alrededor de los puntos, mientras que en la fila de abajo, para cada una de las cuatro direcciones de cálculo se calcula la cuenca visual a los puntos de estudio. Se observa que algunos puntos se aprovechan del alineamiento de los datos empleados en otros cálculos.

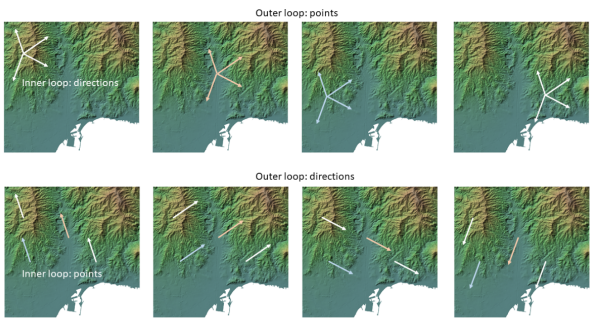


Fig. 5: Intercambio de bucles.

A.1 Almacenamiento sesgado de los datos

Recientemente, Romero *et al* publican en [2] una importante modificación del algoritmo en el que se consideran dos aspectos claves para mejorar aún más el rendimiento: 1) Si, dado un sector, sólo vamos a utilizar el resto de puntos que están en la misma línea ¿por qué no colocar los datos correspondientes en la misma línea de memoria, para aprovechar la localidad espacial? 2) Si, dado un sector, sólo vamos a utilizar el resto de puntos que están en la misma línea (no hay dependencia con otras líneas), ¿por qué no trabajar en paralelo con todas las líneas simultáneamente, usando además, GPUs?

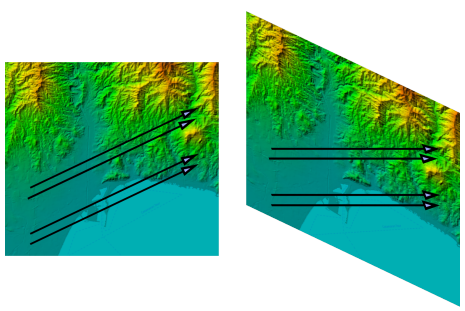


Fig. 6: Almacenamiento de un DEM con sesgo

Nace así el algoritmo sDEM (de skew-DEM, modelo sesgado de elevaciones), que se basa en la idea de colocar los datos en memoria de una forma apropiada para el cálculo en una determinada dirección, ya que el coste de "deconstruir y reconstruir el mapa" merece la pena, teniendo en cuenta la intensidad de los cálculos que se van a realizar sobre los datos en su estado "sesgado".

II. Un motor para optimizar datos en memoria

La propuesta de este trabajo consiste en generalizar esa idea y extenderla a cualquier algoritmo, mediante una plantilla que sea independiente del algoritmo. Para ello, se propone la herramienta *skewEngine*: un código empaquetado en una clase C++ que facilite la parte más tediosa de los muchos algoritmos que pueden aprovecharse de la propuesta. En concreto, el motor sería el encargado de la reorganización de los datos de mallas regulares para que estén alineados en memoria (considerando la necesaria interpolación), y que, al final del algoritmo, reubique los datos a su ubicación original, mediante una interpolación en sentido inverso. En particular, la clase esté diseñada para considerar la siguiente secuencia de etapas:

1. Lectura de la imagen o el modelo
2. Preparación de datos para cada dispositivo (CPU o GPU). Se discute en la sección III.
3. Puesta en marcha de los hilos necesarios
4. Puesta en marcha del iterador (por ejemplo, sobre 360 direcciones). Cada iteración es asignada a un dispositivo. Así, cada CPU/GPU recibirá varios sectores sobre los que va a realizar distintas operaciones.
   a) Preparación de datos (interpolación *skew*)
   b) Procesamiento en CPU o GPU
   c) Restauración de datos (interpolación *deskew*)
5. Recolección de resultados de los dispositivos.

Considerando la anterior estructura, un código sencillo, con OpenMP, que realice operaciones en 360 direcciones, tendría esta forma:

Código 1: Programa básico

```
int main(int argc, char *argv[]) {
    configure(argc, argv);
    omp_set_num_threads(nthreads);
    execute();
    deallocate();
    return 0;
    }

int execute() {
    inData_t inData=prepareData();
#pragma omp parallel default(shared)
    {
        skewer= new skewEngine();
        ...
#pragma omp for schedule(dynamic) nowait
        for (int i = 0; i < 359; i++) {
            // ... in CPU or GPU, do work
        }
#pragma omp critical
        {
            // ... reduce function
        }
        delete skewer;
    }
```

Se propone asimismo que la clase (skewEngine) sirva para cualquier tipo de datos (enteros, flotantes, double, o píxeles, por ejemplo), utilizando plantillas C++, y que se encargue de todo lo relacionado con la preparación de los datos y la recolección de resultados.

A. Operaciones de la clase SkewEngine

La clase, en concreto, realizará sus operaciones en las siguientes etapas:

- Etapa 3: Se creará un objeto skewEngine en cada hilo, al que le corresponde un dispositivo de cálculo. Llamaremos "motores" a dichos objetos.
- En la etapa 4.a, el motor se encarga de"sesgar" los datos de entrada
- En la etapa 4.b, una función externa aplica el algoritmo supuestamente costoso:

  ```
  skewOutput = FuncionCostosa(skewInput);
  ```

  siendo, por ejemplo:

  ```
  FuncionCostosa = cuencaVisualTotal-1D
  FuncionCostosa = radonTransform-1D
  FuncionCostosa = Identity
  ```

- En la etapa 4.c, el motor se encarga de reparar"los resultados sesgados.
- En la etapa 5, una sección crítica recupera la información de los motores, y los destruye.

De esta forma, el kernel de ejecución del código 1 tendría la siguiente estructura:

Código 2: Kernel de ejecución

```
int execute() {
    inData_t inData=prepareData();
#pragma omp parallel default(shared)
    {
        skewEngine<float> *skewer=
            new skewEngine<float>
                (dimx, dimy, inData);
        skewer->kernel= funcionElegida;
#pragma omp for schedule(dynamic) nowait
        for (int i = 0; i < 359; i++) {
            skewer->skew(i);
            skewer->runKernel();
            skewer->deskew(i);
        }
#pragma omp critical
        {
            reduce(skewer,outData);
        }
        delete skewer;
    }
```

Obsérvese que, entre las funciones costosas elegidas. se puede implementar una función identidad, de coste cero. Dicha función tendría una triple función. En primer lugar, como depuración, ya que un volumen de datos deconstruido y reconstruido debe ser casi idéntico. En segundo lugar, sirve para determinar los errores de redondeo implicados en los procesos de interpolación. Finalmente, sirve para estimar el sobrecoste que implica la reorganización de datos, y en consecuencia, para valorar si merece la pena aprovecharse del algoritmo.

III. Implementación de *skewEngine*

Consideremos una imagen o mapa 2D (todo será extrapolable a 3D, de forma anidada, según se explica en la sección V) a la que queremos aplicar el algoritmo. En primer lugar, hay que considerar que:

- Los datos alineados se pueden procesar en doble sentido, por lo que sólo se considerarán 180º direcciones en los cálculos de 1º de precisión..

- Los 180 sectores se clasifican en cuatro bloques, con el objeto de hacer el algoritmo más simple y eficiente.
  Los límites de los cuatro bloques dependen del aspect-ratio de los datos de entrada. En particular, la variable $fAngle = atan(dimy/dimx)$ determina que:

  ```
  set0=[0,fAngle[,
  set1=[fAngle,90[,
  set2=[90,180-fAngle[,
  set3=[180-fAngle,180[
  ```

En la siguiente imagen se muestran, para una imagen, 4 ángulos de cada uno de los conjuntos, así como las 4 posibles situaciones a las que se debe enfrentar *skewEngine*:

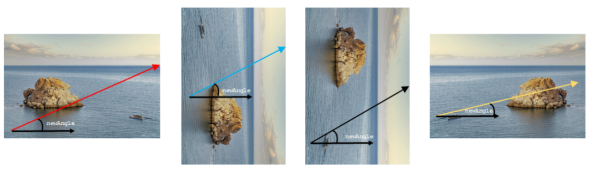


Fig. 7: sDEM.

Antes de proceder al sesgado, se preparan 4 versiones de la imagen o modelo de entrada, a las que podríamos denominar Normal-Normal, Transpose-Normal, Transpose-Mirror y Normal-Mirror, y que se corresponderán con las entradas que necesitaría el algoritmo para los ángulos de cada uno de los conjuntos de sectores. Si disponemos previamente de estas 4 versiones del modelo, el código no necesitará diferenciar la forma en la que se realiza el sesgado y el restablecimiento de los datos:

Código 3: Generación de 4 conjuntos de entrada del algoritmo

```
indata_t prepareData(T *input){
    for(int i=0;i<dimy;i++) {
    int ci=dimy-1-i;
    for (int j = 0; j < dimx; j++) {
        int cj=dimx - 1 - j;
        T val= input [dimx * i +  j];
        input1[dimy * j + ci] = val;
        input2[dimy *cj + ci] = val;
        input3[dimx * i + cj] = val;
        }
    }
    \\ return 4 arrays of type T
}
```

Con estas 4 versiones del modelo, el algoritmo no tiene que distinguir entre signos para la tangentes, ni tipos de acceso, ni está limitado a modelos cuadrados. El procesamiento es similar en los cuatro casos. Consiste, básicamente, en sesgar los datos de entrada, mediante un mecanismo simple de interpolación.

En concreto, será necesario calcular varios parámetros simples, como el ángulo de sesgo ($newAngle$), el desplazamiento que tendrá la última columna ($offset$), las dimensiones verticales y horizontales de la correspondiente versión de entrada (dim_o, dim_i) (subindexados con o por $outer$, i por $inner$) y que

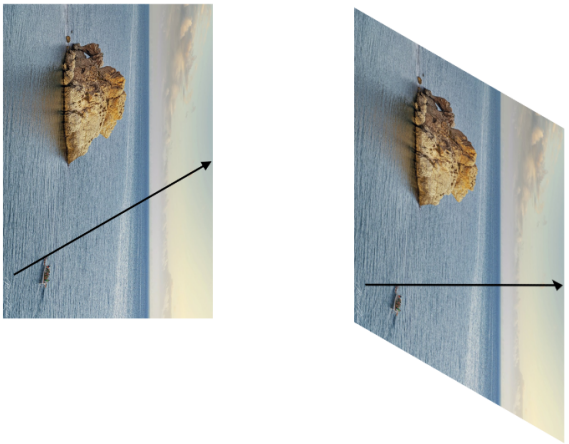


Fig. 8: Sesgado de una imagen, para el sector 113º, perteneciente al conjunto 1

serán los límites de los lazos externos e interno que recorran los datos.

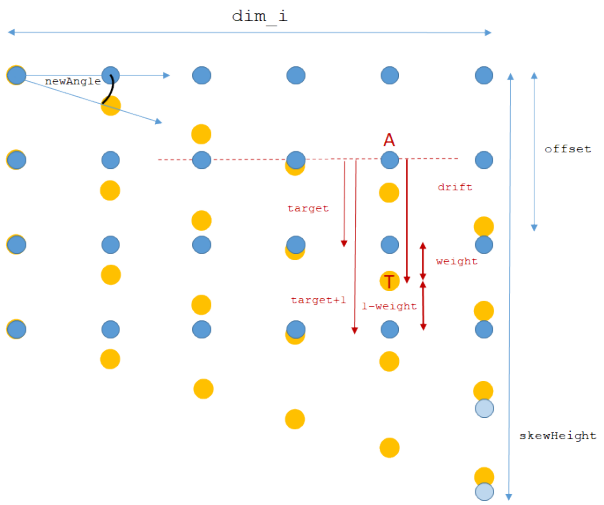


Fig. 9: Representación gráfica de los parámetros necesarios para el sesgado de un modelo

Hay que tener en cuenta que al sesgar los datos de entrada (al pasar del rectángulo al romboide), cada elemento de entrada se va a ubicar en la misma columna, pero en un lugar intermedio entre dos filas del array destino. Por ese motivo, necesitamos calcular un par de vectores que sólo dependen del índice de columna: $target$ y $weight$, siendo $target$ el número de filas que avanza un determinado punto hacia abajo (redondeado a inferior), y $weight$ un factor de ponderación, según la ubicación sesgada del punto, entre $target$ y $target + 1$. En la imagen anterior se representan en rojo dichos parámetros, y que son independientes de la fila.

Asimismo, calcularemos el número de filas que se van a procesar sobre el modelo sesgado, skewHeight, así como los límites de cada fila, que dependen de $newAngle$ y $offset$, y de dim_i, dim_o. En definitiva, estos dos vectores, básicamente, definen la forma del romboide:

Código 4: Cálculo de los límites del sesgado

```
for (int i=0;i< skewHeight; i++){
    first[i]=0;
    last[i]=dim_i;
    if(i<offset)
        last[i]=(i+1)*iskewness+1;
    if(i>dim_o)
        first[i]=(i-dim_o)*iskewness+1;
    }
```

Finalmente, se realiza el sesgado del modelo:

Código 5: Sesgado del modelo

```
for(int i=0;i<=skewHeight*dim_i;i++)
    skewInput[i]=skewOutput[i]=0;
//isT: is transpose
//isM: is mirror
T *source=isT?(isM?input2:input1)
    :(isM?input3:input0);
for (int i = 0; i < dim_o; i++) {
    for (int j = 0; j < dim_i; j++) {
        int row = i + target[j];
        skewInput[row * dim_i + j] +=
            (1.0 - weight[j]) *
            source[dim_i * i + j];
        skewInput[(row + 1) * dim_i + j]
            += weight[j] *
            source[dim_i * i + j];
    }
}
```

A. Computación intensiva tras el sesgado

Una vez preparado los datos, el algoritmo intensivo es un simple bucle, que se puede ejecutar con paralelismo anidado (y embarazoso), ya que cada fila es independiente:

Código 6: Ejecución del kernel

```
for(int i=0;i<skewHeight;i++>)
    skewOutput[i]=
      kernel(skewInput[i],first[i],last[i]);
```

Es importante no olvidar que el espacio se ha deformado, por lo que las distancias que se apliquen en el algoritmo (si se necesitan) tienen un factor de escala 1 en vertical y $1/cos(newAngle)$ en horizontal.

B. Reconstrucción de los datos no sesgados

El proceso de interpolación de los resultados es mucho más simple, ya que no es necesario computar nuevos parámetros. Para ello, definimos un método para la reconstrucción de los datos:

Código 7: Reconstrucción de datos no sesgados

```
skewEngine::deskew()
{
    T *output;
    if(sectorType==0)output=output0;
    if(sectorType==1)output=output1;
    if(sectorType==2)output=output2;
    if(sectorType==3)output=output3;
    for(int i=0;i<dim_o;i++)
     for(int j=0;j<dim_i;j++)
      output[i*dim_i+j]+=
       (1-weight[j])*skewOutput[(i+target[j])
        *dim_i+j]+
        weight[j]*skewOutput[(i+target[j]+1)
        *dim_i+j];
}
```

C. Reducción

Finalmente, cuando una CPU o GPU termina de procesar los sectores que le han sido asignado, tiene que pasar a la fase crítica de reducción. El correspondiente thread puede tener datos de sectores asignados de más de un tipo de los 4 conjuntos, por lo que se apoya en cuatro variables booleanas. En este momento, los datos traspuestos o en espejo recuperan la ubicación original:

Código 8: Reducción para recuperar la ubicación original

```
#pragma omp critical
{
    for (int i = 0; i < dimy; i++){
        int ci=dimy-1-i;
        for (int j = 0; j < dimx; j++) {
            int cj=dimx-1-j;
            if (skewer->has0)outData[i][j]
              += skewer->output0[i*dimx+j];
            if (skewer->has1)outData[i][j]
              += skewer->output1[j*dimy+ci];
            if (skewer->has2)outData[i][j]
              += skewer->output2[cj*dimy+ci];
            if (skewer->has3)outData[i][j]
              += skewer->output3[i*dimx+cj];
        }
    }
}
```

La implementación de *skewEngine* se ha realizado en lenguaje C++, utilizando la librería OpenMP para la distribución de las direcciones de procesamiento entre los distintos núcleos. Normalmente se trabaja con 180 sectores, por lo que el grado de paralelismo es suficiente para que apenas haya desequilibrio de carga en ordenadores de 16 o menos núcleos. El código implementa también la transferencia opcional de los modelos sesgados a una GPU, para que el kernel de un determinado ángulo se pueda ejecutar en la misma. Para GPUs, *skewEngine* se ha programado tanto en CUDA como en OpenCL.

También se ha implementado un kernel unitario, en el que, básicamente, el Código 6 se convierte simplemente en un lazo con la sencilla igualdad $skewOutput[i][j] = skewInput[i][j]$. Además, se han implementado diferentes casos de estudio, y en todos ellos, tan sólo se necesita definir un método de acceso de tipo void, cuyo único argumento es el objeto de la clase *skewEngine* que corresponde al sector. Lo normal es que el programador tan sólo tenga que cambiar la línea $skewer- > kernel = funcionElegida$, en el Código 2, por su propio caso de estudio. Por ejemplo, para la Cuenca Visual Total:

```
skewer->kernel=isGPU?viewshedGPU:vieshedCPU;
```

El código *skewEngine* está disponible públicamente en un repositorio Github [3].

IV. Casos de estudio

Los excelentes resultados obtenidos por el algoritmo sDEM [4] para el cálculo de la cuenca visual total fueron la motivación principal para el desarrollo de la herramienta *skewEngine*. Sin embargo, en sDEM se detectaron algunas deficiencias que se han corregido en este trabajo, y que, básicamente se centran

en que sDEM no hace una preparación de los datos antes del procesamiento, sino que simultáneamente hace el trabajo de sesgado, aplica el algoritmo, y reconstruye los datos originales. Además de ser un código tremendamente complejo y poco exportable a otros casos, sDEM no diferencia previamente entre los 4 conjuntos descritos en el código 3, por lo que incluye numerosas bifurcaciones que ralentizan el código, especialmente en GPUs. La implementación de la cuenca visual total con *skewEngine* ha sido precisamente (tras la obvia implementación de la identidad) el primer caso de estudio considerado. Sin embargo, para comprobar la facilidad de la implementación de otros códigos con el modelo propuesto, hemos elegido otros dos casos adicionales: el filtrado de imágenes borrosas y la transformada Radon. Por otro lado, en la Tabla I se muestran las arquitecturas utilizadas en este trabajo.

TABLA I: Arquitecturas utilizadas en los casos de estudio.

	Máquina 1	Máquina 2	Máquina 3
Nombre	Desktop–PC	Server	HPC node
CPUs	16x Intel	32x Intel	64x Intel
	i7-10700K	Xeon E5-2698 v3	Xeon E5-2698 v4
GPUs	1x NVidia	4x NVidia	8x NVidia
	Ampere RTX4080	Maxwell GTX980	Volta V100

En los siguientes apartados se muestra un breve resumen de los resultados de los casos de estudio. Sin embargo, es muy importante aislar previamente el tiempo empleado en las dos fases en las que está realmente implicada la herramienta *skewEngine*, pues es lo que servirá para identificar las aplicaciones en las que merezca la pena utilizarla:

A. Caso 1: Identidad

Para estimar el coste de las etapas *skew* y *deskew*, se ha utilizado un kernel identitario (los datos se deconstruyen y reconstruyen sin cálculos intermedios), en sus versiones para CPU y GPU. Se han seleccionado 180 sectores, y se ha aplicado a dos conjuntos de datos diferentes: una fotografía RGB de 2122×2122 píxeles, y un DEM de una zona montañosa de 2000×2000 puntos. Para simplificar, se muestran solo los resultados obtenidos, en tiempo de ejecución, para la fotografía, ya que el escalado del tiempo con el tamaño es prácticamente lineal (12,5 %× más lenta la imagen que el DEM). Los resultados se muestran en la Tabla II

TABLA II: Tiempos de ejecución del kernel Identidad

	Máquina 1	Máquina 2	Máquina 3
tiempo CPUs	2.34 s.	1.3 s.	0.45 s.
tiempo GPUs	0.24 s.	0.46 s.	0.10 s.

Teniendo en cuenta la perfecta escalabilidad de la herramienta respecto al número de núcleos o GPUs, se han utilizado sólo 15, 30 y 60 núcleos de los disponibles en las respectivas máquinas, para que sean divisores del número de sectores (180) y descontar así el desequilibrio de carga. Por otra parte, el código puede elegir una CPU o GPU mediante un paradigma de granja de tareas, por lo que los mejores tiempos (siempre con GPUs) podrían reducirse si reciben la colaboración de las CPUs, aunque apenas merece la pena en ninguna de las arquitecturas. En cualquier caso, se observan tiempos muy razonables, especialmente para algoritmos que pueden tardar minutos, horas e incluso días, en conjuntos de datos de tamaños similares.

B. Caso 2: Cuenca Visual Total

Como se ha descrito en la subsección I-A, la cuenca visual total determina la superficie de un territorio que es visible por un observador, calculada para todas las posibles ubicaciones del observador. Dada una ubicación cualquiera del observador, su visibilidad se determina en un conjunto discreto de s direcciones radiantes equitativamente distribuidas. Con *skewEngine*, y dada una dirección discreta, hemos visto que fácilmente se puede calcular la cuenca visual a todos los puntos que estén en la misma línea, reutilizando todos los datos de elevación.

En trabajos previos [5], [4], [6], la mayoría de los modelos de elevación utilizados tienen tamaños de alrededor de 2500x2500 puntos. Estas dimensiones son suficientes para cubrir, por ejemplo, la superficie del Parque Nacional Sierra de las Nieves, con una resolución de 10 metros. Teniendo en cuenta que una línea de datos tiene un tamaño de (a lo sumo) 2 a 4 mil datos de elevación, y que normalmente cada dato es de sólo 2 bytes, lo normal es que toda la información de una línea quepa en la cache L1 de un núcleo. Sin embargo, por la propia naturaleza del algoritmo, el número de operaciones es de cientos de millones de FLOPs por línea, que al ejecutarse sin apenas fallos en la L1, producen unos rendimientos altísimos en todas las arquitecturas empleadas en este trabajo.

Sin entrar en detalles concretos de los resultados, cabe destacar que en el peor de los casos (el ordenador de sobremesa, utilizando sólo las CPUs) el tiempo de ejecución fue de 59 s. para el modelo de 25M de puntos del Parque Sierra de las Nieves, y mejorando un 10 % los resultados de sDEM. Sin embargo, la GPU hizo los cálculos en apenas 3.5 s, mientras que sDEM requiere 10.2 s. Hay que tener en cuenta que las herramientas de cálculo utilizadas en los Sistemas de Información Geográfica, como gdal-viewshed, o GRASS [7], [8] hacen sus operaciones en pocos segundos para el cálculo de ¡una sola cuenca visual, en lugar de 25 millones de ellas!, lo cual supone que nuestro modelo es de 6 a 7 órdenes de magnitud más rápido.

C. Caso 3: Transformada Cepstrum para filtrado de imágenes borrosas por movimiento

Pero el objeto de este estudio no es demostrar el rendimiento de una aplicación como la referenciada en la subsección anterior, sino su utilidad para implementar rápidamente versiones más eficientes de otros algoritmos. Y para ello, hemos elegido dos aplicaciones que requieren un cálculo muy intensivo. La primera de ellas es la transformada Cepstrum local, aplicada a una imagen borrosa por el movimiento.

La transformada Cepstrum es una técnica matemática utilizada para analizar la estructura espectral de una señal en el dominio cepstral. Se define como la transformada de Fourier del logaritmo del espectro de potencia de la señal. Su fórmula se expresa de la siguiente manera:

$$C(\tau) = \mathcal{F}^{-1}\left[\log\left(|\mathcal{F}[x(t)]|^2\right)\right]$$

Donde $x(t)$ representa la señal en el dominio del tiempo y τ es la variable de retardo. El uso de la transformada cepstrum es común en aplicaciones de procesamiento de señales y análisis espectral [9], [10]. En imágenes borrosas por movimiento, el dominio cepstral ayuda a identificar los coeficientes que identifican la dirección e intensidad del movimiento que ha provocado que la imagen esté borrosa. No se ha podido encontrar ningún experimento en la literatura que haya calculado la transformada cepstral centrada en cada uno de los píxeles de una imagen, posiblemente por su elevado coste computacional. El análisis cepstral sólo se aplica a toda la imagen, por lo que sólo se utiliza para detectar el movimiento de la cámara, y no de los diferentes objetos de la cámara, como ocurre en la imagen inferior, en la que distintos objetos se vuelven borrosos en direcciones y velocidades diferentes.

Fig. 10: Imagen borrosa por objetos con diferentes movimientos

Con *skewEngine* se ha implementado la transformada Cepstrum aplicada a cada píxel de la imagen, en 360 direcciones diferentes, y con ventanas de 32, 64 y 128 píxeles de radio. Los tiempos de ejecución oscilan entre 1 y 3 minutos para una imagen de 4Mpixel, aunque lo más sorprendente es que se podido programar en apenas unas horas, gracias a la herramienta.

D. Caso 4: Transformada Radon

la transformada de Radon es una herramienta matemática utilizada en radiología para la reconstrucción de imágenes de objetos a partir de mediciones de rayos X o de otras formas de radiación. En radiología, la transformada de Radon mide la cantidad de radiación que atraviesa el objeto desde todas las direcciones posibles y convierte esta información en una imagen bidimensional o tridimensional del objeto. Este proceso se llama tomografía y se utiliza comúnmente en medicina para visualizar estructuras internas del cuerpo humano y en otras áreas de la ciencia para la inspección de materiales o la investigación de la naturaleza de un objeto. La transformada Radon se define como la integral de la imagen a lo largo de todas las rectas que pasan por un punto fijo del espacio. En otras palabras, para cada dirección, se mide la cantidad de radiación que atraviesa el objeto a lo largo de esa dirección y se integra esta información a lo largo de todas las rectas de esa dirección.

El resultado de la transformada de Radon es una función que representa la suma de las proyecciones a lo largo de cada dirección posible. Esta función se llama sinograma y se utiliza como entrada para la reconstrucción de la imagen. La reconstrucción se realiza mediante una técnica llamada retroproyección filtrada, que consiste en tomar cada proyección del sinograma, rotarla y luego "proyectarla hacia atrás" a lo largo de la dirección correspondiente en el espacio. El resultado de este proceso, tras aplicarlo en todas las direcciones, se acumula para producir la imagen reconstruida del objeto.

La Transformada Radon Local (LRT, por sus siglas en inglés) es una variante de la Transformada de Radon que se utiliza para analizar imágenes en dominios locales y no globales. A diferencia de la transformada de Radon estándar, que utiliza la información de la proyección de la imagen en todas las direcciones posibles, la LRT utiliza una ventana de análisis local en la imagen para calcular la transformada de Radon. Esto permite analizar la imagen de forma más detallada y adaptativa a las características locales de la imagen. La LRT tiene varias aplicaciones, como el análisis de texturas en imágenes, la detección de bordes y la segmentación de imágenes. También se utiliza en campos como la inspección no destructiva de materiales, la visualización médica y la astronomía. La Transformada Radon Local (LRT) tiene un alto costo computacional en comparación con la transformada de Radon estándar. Esto se debe a que la LRT requiere el cálculo de la transformada de Radon para cada ventana local en la imagen, lo que puede ser computacionalmente intensivo, por lo que SkewEngine se presenta como la herramienta más adecuada para ello.

En este trabajo se ha implementado tanto la transformada Radon local como la general o estándar, aunque sólo presentamos datos de comparación con la última, ya que es la única de la que existe un software disponible en la bibligrafía. En particular, hemos comparado nuestros resultados con las dos aplicaciones más utilizadas y eficientes: SciKit [11] (en Python) y Astra Toolbox [12], para Matlab. En ambos casos, el código binario de los respectivos kernel está en C++, y también en CUDA, para el caso de Astra.

La imagen que hemos elegido para la trasformada Radon es una fotografía de 1500×1000 píxeles. Sin embargo, una de las aplicaciones mencionadas internamente extiende las imágenes para que estén circunscritas en un círculo, y éste, a su vez, en un

cuadrado, por lo que hemos preferido crear una imagen ficticia de 2122x2122 píxeles para que la comparación sea en iguales condiciones, y a pesar de que perjudica los resultados de *skewEngine*, que trabaja con los límites ajustados. En todas las aplicaciones, la imagen resultante tras la aplicación del algoritmo ha sido idéntica (utilizando un filtro rampa), como se muestra en la Fig. 11.

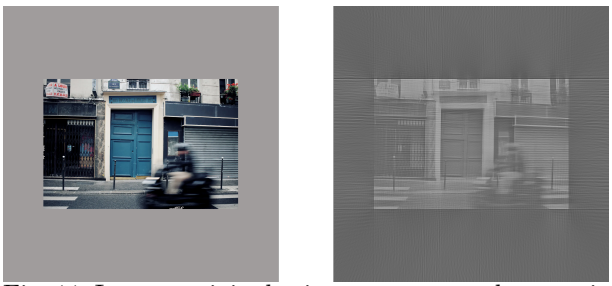


Fig. 11: Imagen original, e imagen restaurada a partir del sinograma

La siguiente tabla muestra un resumen de los resultados de la comparación, en las que se demuestra el alto rendimiento de *skewEngine*, a pesar de que no es precisamente un algoritmo especialmente costoso desde el punto de vista computacional:

TABLA III: Tiempos de ejecución de la transformada Radon

	SKE (Server)	SKE (PC)	Scikit (PC)	Astra (PC)
CPUs	1.46 s.	2.03 s.	10.58 s.	-
GPUs	0.49 s.	0.317 s.	-	0.316 s.

Como se puede observar, Astra Toolbox, que utiliza CUDA, es la que obtiene mejores resultados, y obviamente, esto se debe a que el coste de la reorganización de los datos en memoria, que en este caso supone casi el 70 % del coste total, no merece la pena para una aplicación con tan poco coste computacional. Pero si sorprende que tan sólo se hayan necesitado apenas cuatro líneas de código para su implementación con *skewEngine*:

Código 9: Código de la transformada Radon con *skewEngine*

```
for (int i=start;i<end;i++)
    sum+=skewInput[row *dim_i+i];
for (int i=start;i<end;i++)
    skewOutput[row*dim_i+i]=sum;
```

Poniéndose así de manifiesto no sólo el rendimiento de la herramienta, sino también la altísima productividad en la programación de herramientas que se aprovechen de la misma.

V. Extensión a tres dimensiones

Si el procesamiento intensivo de datos, con una dependencia de todos con todos en cualquier dirección sobre una malla de datos bidimensional es muy costoso en dos dimensiones, su extensión a tres dimensiones puede ser excepcionalmente costoso, por lo que es evidente que cualquier forma de abordar el problema debe partir de dos premisas similares a nuestra anterior propuesta en dos dimensiones.

En primer lugar, el infinito número de direcciones en el que podríamos procesar los datos debe reducirse, de una forma similar a como ya se hizo con la discretización azimutal en 2D con *skewEngine*, y que por defecto, dividía el espacio en 360º.

En segundo lugar, los datos deben alinearse en memoria, e incluso sería altamente recomendable un almacenamiento alineado de la información en memoria secundaria, también usando criterios similares a *skewEngine*. El primer problema, lo hemos resuelto utilizando un elegante y sencillo método de discretización del espacio 3D: la espiral esférica de Fibonacci: un algoritmo que obtiene una nube de puntos cuasi-equitativamente distribuidas en la superficie de una esfera. El siguiente código muestra cómo se calculan las direcciones de procesamiento discretas (ejes de Fibonacci)

Código 10: Cálculo de los ejes esféricos de Fibonacci

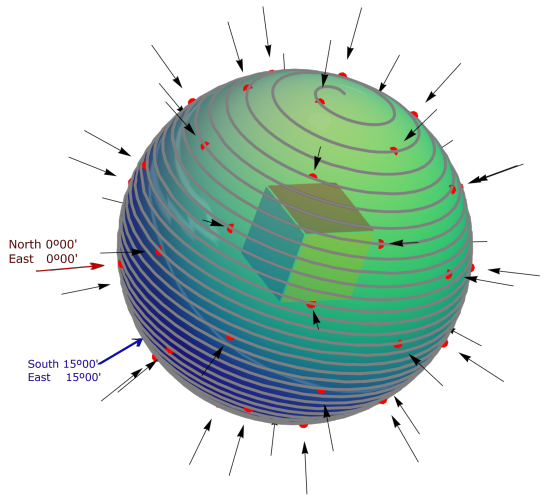
```
double golden= (1+sqrt(5.0))/2;
for (int i=0; i<n; i++) {
    double the= 2*pi*i/golden;
    double phi= acos(1-2*(i+0.5)/n);
    double sphi= sin(phi);
    x= sphi * cos(the);
    y= sphi * sin(the);
    z= cos(phi);
    axes[i]= {x,y,z};
}
```

Como se observa en el código, y haciendo un símil de la esfera con nuestro planeta, las latitudes de los diferentes ejes están equitativamente distribuidos, de forma que si el número de ejes fuera $n = 180$, habría una separación exacta de un grado de latitud entre ejes (latitudes desde 0.5 a 179.5, para ser precisos). Por otra parte, a cada "latitud discreta" le corresponde una longitud exclusiva que depende de la razón áurea. Siguiendo con el símil, podría ser que a la latitud 36.5º le corresponda la longitud de Málaga, mientras que justo la anterior de la serie, a 35.5º, podría coincidir con Tokio.

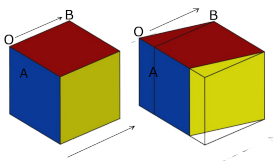
Podemos aprovechar entonces la distribución equitativa de latitudes para resolver el problema del alineamiento de los datos en 3D mediante la misma reutilización del código para *skewEngine* en 2D. De esta forma, los n_x planos de datos (de dimensiones $n_y \times n_z$) serían reproyectados con el algoritmo *skewEngine* generando un cubo que estaría sesgado perpendicularmente respecto al eje z.

Véase con un ejemplo: Una malla tridimensional de datos, como la representada en el interior de la esfera tiene los datos alineados en memoria según la dirección marcada en rojo. Esto es, la arista que separa su cara amarilla de la roja almacena datos consecutivos. A esa situación inicial, la marcamos con latitud y longitud cero.

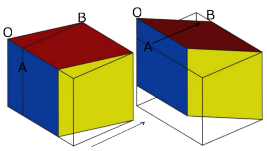
Sin embargo, queremos aplicar un algoritmo intensivo que procese los datos en la dirección marcada por la flecha azul, de latitud -15º y longitud 15º. Para alinear la información, se realizaran dos pasos. En una de ellas se alinearía en latitud, mediante la aplicación del algoritmo *skewEngine* aplicado a cada plano Z, y en la segunda etapa, al cubo regular que



(a) En rojo, eje del alineamiento natural de datos. En azul, alineamiento deseado.



(b) Alineado en longitud



(c) Alineado en latitud

Fig. 12: Algoritmo *skewEngine* en 3 dimensiones aplicado a un eje de la secuencia de Fibonacci

contiene al cubo original sesgado, se le aplicaría el sesgo de latitud (o viceversa):

En [13] ya se utilizó la reproyección de objetos tridimensionales como una etapa de preprocesamiento en el entrenamiento de redes neuronales para el reconocimiento de moléculas. Gracias al alineamiento de la información, en ese trabajo se llegaron a generar cientos de millones de imágenes correspondientes a decenas de miles de moléculas (los componentes principales de los correspondientes fármacos) en apenas unos segundos.

VI. Conclusiones

El correcto alineamiento de datos en la memoria del computador es un factor cada vez más importante en el diseño de algoritmos eficientes que procesen datos estructurados de una forma intensiva. En el caso de mallas regulares de dos o más dimensiones, los algoritmos que realizan operaciones en una dirección concreta que sea diferente a la dirección principal de alineamiento (la que coincide con la secuencia de almacenamiento de los correspondientes datos), tendrán un patrón de acceso a los datos que resulta casi completamente aleatorio, y en consecuencia, provoca innumerables fallos de cache que pueden ser catastróficos, especialmente en el caso de grandes volúmenes de datos, como aquellos generados en un TAC o en radioastronomía.

El re-alineamiento de datos en la dirección correspondiente a las operaciones del algoritmo es una operación relativamente costosa si el volumen de datos es elevado, pero como su complejidad computacional es lineal y por tanto, escala bastante bien, puede merecer la pena el coste de la reorganización de la información, siempre y cuando las operaciones de los algoritmos tengan complejidades superiores. En los experimentos mostrados en este trabajo, correspondientes a tres problemas computacionales de alta y media complejidad, como son la cuenca visual total en modelos digitales de elevación, la parametrización de imágenes borrosas por movimiento, y la transformada Radon bidimensional, se ha demostrado que el re-alineamiento de la información ya merece la pena en algoritmo de media complejidad, superando incluso a los mejores resultados publicados, y produce espectaculares mejoras, de varios órdenes de magnitud, en los problemas de mayor complejidad.

Para facilitar el uso de los algoritmos de interpolación y extrapolación de las mallas en las n direcciones elegidas (que son las etapas que preceden y suceden a cualquier algoritmo que realice su operación en una dirección concreta), se ha elaborado una clase en lenguaje C++, denominada *skewEngine*, que gracias a la explotación del paralelismo embarazoso de la interpolación mediante OpenMP y OpenCL o CUDA, hace que la parte más incómoda de la propuesta de este trabajo sea simple y rápida. Además, *skewEngine* está diseñado de una forma que facilita la implementación de otros algoritmos de cálculo intensivo de datos en mallas regulares usando direcciones arbitrarias de procesamiento.

Los experimentos se han desarrollado mediante cálculos en direcciones equitativamente distribuidas en dos dimensiones (normalmente, en $n = 360$ direcciones, separadas un grado), pero también se presentan las directrices de una implementación de la herramienta *skewEngine* en 3 dimensiones, utilizando, igualmente, un conjunto equitativo de direcciones de búsqueda tridimensional que se calculan utilizando la espiral esférica de Fibonacci. De este último caso, se han publicado resultados preliminares en la que datos no estructurados de miles de moléculas, mediante interpolación numérica, se interpolan a mallas regulares que posteriormente se proyectan en imágenes, y que se han utilizado para el entrenamiento de una red neuronal. Al reconvertir y estructurar la información, se ha conseguido la generación de decenas de millones de imágenes en apenas unos segundos.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia y Tecnología de España mediante el proyecto del Plan Nacional PID2019-105396RB-I00, por la Junta de Andalucía y los fondos FEDER a través del proyecto UMA20-FEDERJA-127, y por la Universidad de Málaga (PIE22-099). Agradecemos asimismo al Servicio de Supercomputación y Bioinformática de la Universidad de Málaga y a la Red Española de Supercomputación por facilitar el acceso a los Supercomputadores Picasso y Loginexa.

Referencias

[1] S. Tabik, E.L Zapata, and L.F Romero, "Simultaneous computation of total viewshed on large high resolution grids," *International Journal of Geographical Information Science*, vol. 27, no. 4, pp. 804–814, 2013.

[2] Andres J. Sanchez-Fernandez, Luis F. Romero, Gerardo Bandera, and Siham Tabik, "Vpp: Visibility-based path planning heuristic for monitoring large regions of complex terrain using a uav onboard camera," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. PP, pp. 1–1, 12 2021.

[3] Felipe Romero, "Skewengine: Mesh reorganization for computing intensive applications," https://github.com/luisfromero/skewEngine, 2023.

[4] A. J. Sanchez-Fernandez, Luis F. Romero, G. Bandera, and S. Tabik, "A data relocation approach for terrain surface analysis on multi-gpu systems: a case study on the total viewshed problem," *International Journal of Geographical Information Science*, vol. 35, no. 8, pp. 1500–1520, 2021.

[5] A.R. Cervilla, S. Tabik, J. Vias, M. Merida, and L.F. Romero, "Total 3d-viewshed map: Quantifying the visible volume in digital elevation models," *Transactions in GIS*, 2016.

[6] Siham Tabik, Luis Felipe Romero, and Emilio López Zapata, "High-performance three-horizon composition algorithm for large-scale terrains," *International Journal of Geographical Information Science*, vol. 25, no. 4, pp. 541–555, 2011.

[7] GDAL/OGR contributors, *GDAL/OGR Geospatial Data Abstraction software Library*, Open Source Geospatial Foundation, 2020.

[8] QGIS Development Team, *QGIS Geographic Information System*, QGIS Association, 2023.

[9] Johann Radon, "On the determination of functions from their integral values along certain manifolds," *IEEE Transactions on Medical Imaging*, vol. 5, no. 4, pp. 170–176, 1986.

[10] Alan V. Oppenheim and Ronald W. Schafer, *Discrete-Time Signal Processing*, Pearson, 2010.

[11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al., "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[12] Wim Aarle, Willem Palenstijn, Jan De Beenhouwer, Thomas Altantzis, Sara Bals, Kees Batenburg, and Jan Sijbers, "The astra toolbox: A platform for advanced algorithm development in electron tomography," *Ultramicroscopy*, vol. 157, 05 2015.

[13] Felipe Romero, Luis F. Romero, and Pilar M. Ortigosa, "Reconocimiento de fármacos mediante inteligencia artificia," in *XXXII Jornadas de Paralelismo*. 2022, Jornadas Sarteco.

Modelo de sincronización para la mitigación de cuellos de botella en flujos de datos

Dante D. Sánchez-Gallegos[1], Jesus Carretero[1], y J. L. Gonzalez-Compean.[2]

Resumen— **En años recientes, las organizaciones han empezado a migrar sus soluciones a ambientes distribuidos como la computación edge-fog-cloud, con el objetivo de reducir el tiempo de respuesta para obtener una respuesta de un sistema, acercando el procesamiento de los datos a las fuentes en donde estos son producidos (e.g., dispositivos de Internet de las cosas). Para ello se despliegan múltiples aplicaciones heterogéneas en las diferentes infraestructuras del entorno edge-fog-cloud para realizar el preprocesamiento, procesamiento, almacenamiento y visualización de los datos. Sin embargo, esta heterogeneidad de aplicaciones e infraestructura produce cuellos de botella en tiempo de ejecución que afectan el rendimiento del sistema. Lo anterior produce retrasos en la entrega de resultados a los usuarios finales, lo cual es crítico en escenarios de toma de decisiones. En este trabajo se presenta un modelo de sincronización para la mitigación de cuellos de botella en sistemas edge-fog-cloud en tiempo de ejecución. Este modelo está inspirado en el principio de Bernoulli, y propone que las aplicaciones más lentas (cuellos de botella) de un sistema sean manejadas como áreas de alta presión, mientras que las aplicaciones más rápidas se manejan como áreas de baja presión. Este manejo diferenciado de aplicaciones, permite al modelo identificar cuellos de botella y mitigarlos mediante el uso de patrones auto-paralelos, esquemas de balanceo de carga y un modelo de gestión de operaciones. Se ha implementado y evaluado un prototipo de este modelo en un estudio de caso basado en el procesamiento de datos satelitales en ambientes edge-fog-cloud. La evaluación de este prototipo reveló que el modelo es capaz de mitigar los cuellos de botella, incrementando el rendimiento del sistema evaluado.**

Palabras clave— **Flujos de datos, procesamiento continuo, manejo de cuellos de botella, patrones de paralelismo**

I. Introducción

En años recientes, se ha incrementado de manera exponencial el volumen de datos manejados y almacenados por organizaciones de diferentes industrias [1]. Por ejemplo, se espera que los datos en la industria de la salud crezcan a un ritmo del 36 % entre los años 2018 y 2025 [1]. Para gestionar estos grandes volúmenes de datos, las organizaciones han empezado a migrar sus sistemas a nuevos paradigmas de computación continua, tales como computación en el edge-fog-cloud [2, 3] y servicios multi-cloud [4, 5]. Estos paradigmas permiten a las organizaciones desplegar múltiples aplicaciones heterogéneas para gestionar el ciclo de vida de sus productos digitales (e.g., fotografías, música, vídeos y contratos, por mencionar algunos ejemplos), procesando grandes cantidades de datos a través de múltiples infraestructuras [6, 7].

[1]Universidad Carlos III de Madrid, e-mail: 100433984@alumnos.uc3m.es, jcarrete@inf.uc3m.es
[2]Cinvestav Tamaulipas, e-mail: joseluis.gonzalez@cinvestav.mx

Para gestionar estas aplicaciones como un único sistema, las organizaciones crean estructuras de procesamiento (p. ej. flujos de trabajo o tuberías) que les permiten interconectar sus aplicaciones mediante el diseño de diferentes patrones, comúnmente abstraídos como grafos acíclicos dirigidos (DAGs, por sus siglas en inglés) [8, 9]. En un DAG, los nodos representan las aplicaciones utilizadas para gestionar y procesar los datos, mientras que las aristas representan el intercambio de datos entre aplicaciones.

Así, en un sistema creado a partir de una estructura de procesamiento, se producen diferentes versiones de los productos digitales a través de un conjunto de etapas definidas por las organizaciones [10]. Cada etapa de este sistema se implementa como una aplicación que recibe una entrada, procesa esta entrada y produce una salida que es entregada a la siguiente etapa en el sistema [11].

En escenarios reales, en un sistema compuesto por múltiples aplicaciones desplegadas en múltiples infraestructuras, se producen cuellos de botella debidos a la heterogeneidad en el rendimiento de estas aplicaciones [12, 13]. Estos cuellos de botella producen tiempos de inactividad en las aplicaciones más rápidas, así como una saturación y problemas de encolamiento en las etapas más lentas. Lo anterior, crea problemas de sincronización en la entrega de datos entre aplicaciones, los que es observado por los usuarios finales como tiempos de respuesta grandes o retrasos en la entrega de datos [14].

En el presente artículo, se presenta un modelo de sincronización para mitigar cuellos de botella en sistemas desplegadas en ambientes de múltiples infraestructuras (p. ej., en ambientes edge-fog-cloud o multi-cloud). En este modelo, los cuellos de botella son mitigados en tiempo de ejecución en dos fases: monitorización, diagnóstico y rectificación.

En la etapa de monitorización, entidades llamadas *centinelas* recolectan diferentes métricas de rendimiento de las aplicaciones de un sistema, tales como el tiempo de respuesta y el volumen de datos procesados. Este proceso de monitorización continuo permite obtener el rendimiento de las aplicaciones, lo que facilita la identificación del cuello de botella en la etapa de diagnóstico.

En la etapa de diagnóstico, un componente llamado *gestor de autoescalado* recibe los datos de rendimiento recolectados por los centinelas y se encarga de identificar los cuellos de botella a partir de una metáfora basada en el principio de Bernoulli. En mecánica de fluidos, este principio es utilizado para describir cómo los cambios en la presión de un flujo afectan la velocidad de este. En una tubería horizon-

tal, la velocidad de un fluido disminuye en las zonas de alta presión de la tubería, mientras que su velocidad aumenta en zonas de baja presión. Nosotros proponemos utilizar este principio como una metáfora que permite la representación del estado de las aplicaciones en un sistema mediante el análisis de su rendimiento. En este modelo, los datos que se intercambian entre las diferentes aplicaciones representan un flujo, mientras que la presión es representada como la cantidad de datos en el búfer de entrada de las aplicaciones y el rendimiento (medido como volumen de datos procesados por unidad de tiempo) de las aplicaciones representa la velocidad del flujo. En este sentido, la etapa con la presión mayor representa un cuello de botella, dado que es aquella que tiene el menor rendimiento. Mientras que las etapas más rápidas (con mayor rendimiento) son aquellas con una presión menor.

Finalmente, en la etapa de rectificación la etapa identificada con la mayor presión es etiquetada como cuello de botella y mitigado por el gestor de autoescalado mediante el uso de un esquema de paralelismo implícito basado en el patrón gestor/trabajador. En este modelo, los trabajadores de un patrón son creados utilizando la tecnología de contenedores virtuales, lo que permite agregar a las aplicaciones la propiedad de portabilidad y agilizar el despliegue de clones de estas. Para calcular el número de trabajadores en un patrón, el gestor de autoescalado analiza el rendimiento de las etapas con un rendimiento próximo al cuello de botella. El número de trabajadores calculado por el gestor de autoescalado es entregado a los centinelas, quienes se encargan de crear clones de la aplicación etiquetada como cuello de botella, y agregarlos como trabajadores de un patrón gestor/trabajador. En este patrón, la carga de trabajo entrante es distribuida entre los trabajadores utilizando un balanceador de carga [15]. Como resultado, la carga en el búfer de entrada del cuello de botella es distribuido y procesado de forma paralela, lo que reduce el tiempo de espera en cola de los datos a procesar, así como el tiempo de servicio de la aplicación identificada como cuello de botella. Lo anterior, reduce la presión de la etapa y aumenta su velocidad, mitigando los efectos del cuello de botella en el sistema.

Se ha implementado un prototipo basado en el modelo propuesto como un estudio de caso enfocado en la gestión y procesamiento de datos satelitales. La evaluación experimental reveló que el modelo es capaz de mitigar los cuellos de botella en un sistema, incrementando su eficiencia.

El resto del artículo se encuentra organizado de la siguiente manera: en la Sección II se presenta el trabajo relacionado con este modelo, en la Sección III se describe el modelo propuesto y los principios de diseño de un prototipo basado en este modelo, en la Sección IV se presenta la evaluación experimental conducida como un estudio de caso basado en el procesamiento de datos satelitales y en la Sección V se presentan las principales conclusiones y trabajo futuro.

II. Trabajo relacionado

En la literatura existen una gran cantidad de herramientas que permite el diseño, despliegue y ejecución de sistemas [16]. Estas herramientas implementan esquemas que buscan mejorar la eficiencia en el procesamiento de los datos, mediante la implementación de patrones paralelos, balanceadores de carga o técnicas de escalamiento vertical y horizontal [17]. Por ejemplo, diferentes gestores de flujos de trabajo como Makeflow [18] o Parsl [19] permiten, en tiempo de diseño, especificar el número de trabajadores paralelos que procesarán los datos en tiempo de ejecución. Mientras que otras herramientas permiten, durante tiempo de ejecución, planificar las tareas para aprovechar eficientemente los recursos disponibles [20]. En este sentido, se puede observar que las herramientas para el diseño de sistemas permiten utilizar estrategias que mejoren la eficiencia de estos durante tiempos de diseño, despliegue y ejecución.

En tiempo de diseño, los diseñadores crean un sistema utilizando un modelo de programación, el cual permite materializar en forma de código la unión de los diferentes componentes del sistema. Dentro de esta codificación, se incluye el número de hilos a desplegar, la cantidad de núcleos físicos a utilizar y el patrón de paralelismo con el que se desplegará el sistema [6,18,19,21]. Soluciones como DagOnStar [21] y Parsl [19] proporcionan un modelo de programación basado en Python, en el cual los diseñadores pueden especificar diferentes parámetros de paralelismo.

En tiempo de despliegue, soluciones como DagOnStar [21], TOPOSCH [22], Kulla [23] y Jenkins [24] permiten a los diseñadores especificar diferentes parámetros que impactan en la eficiencia del sistema. Por ejemplo, la selección de la infraestructura donde se desplegará el sistema o el reservar recursos para las aplicaciones que los diseñadores consideren prioritarias, ya sea porque consideren que requieren más recursos o porque deben de ser ejecutadas antes que el resto de aplicaciones [20].

Además, en la literatura existen soluciones que permiten generar una planificación automática que permita hacer un uso eficiente de los recursos. Por ejemplo, Slurm [25], Condor [26] y ESMS [27] implementan algoritmos de planificación para desplegar las aplicaciones en un conjunto de recursos de computación siguiendo un conjunto de restricciones (p. ej. presupuesto, latencia o requerimientos de calidad de servicio) [20, 28, 29].

Durante tiempo de ejecución, diferentes soluciones permiten identificar y mitigar cuellos de botella en un sistema, sin requerir la intervención de los diseñadores y los usuarios del sistema. Diferentes herramientas en el estado del arte permiten mitigar cuellos de botella utilizando técnicas de paralelismo, gestión de memoria y reducción de la latencia entre aplicaciones. SWITCH [30] incluye un sistema de monitorización para identificar aquellas aplicaciones computacionalmente costosas y realizar su autoesca-

lado. IntMA [31] agrupa las aplicaciones con el objetivo de reducir la latencia entre estas. Gearbox [32] monitoriza las aplicaciones en una tubería computacional para identificar cuellos de botella y mitigarlos utilizando esquemas de paralelismo y técnicas de gestión de memoria.

III. Modelo de sincronización para la mitigación de cuellos de botella

En esta Sección se describe el modelo propuesto para identificar y mitigar cuellos de botella en sistemas edge-fog-cloud compuestos por aplicaciones desplegadas en múltiples infraestructuras. Este modelo incluye las etapas de monitorización, diagnóstico y rectificación para la mitigación de cuellos de botella en un sistema edge-fog-cloud. En esta Sección, además, se describe el diseño de un prototipo basado en el modelo propuesto.

A. Monitorización continua de las aplicaciones en un sistema edge-fog-cloud

El objetivo del modelo presentado en este artículo es identificar cuellos de botella que puedan surgir en sistemas edge-fog-cloud conformados por un conjunto de aplicaciones heterogéneas, desplegadas en múltiples infraestructuras (ver Figura 1). Un sistema edge-fog-cloud se puede modelar como un DAG en el que los nodos del grafo representan las aplicaciones utilizadas para procesar los datos, mientras que las aristas representan el intercambio de datos entre estas aplicaciones. La ecuación 1 muestra el modelo:

$$\begin{aligned} sys = &\{(FD \rightarrow app_1), (app_1 \rightarrow app_2), \\ &\ldots, (app_{n-1} \rightarrow app_n), (app_n \rightarrow DD)\}, \end{aligned} \tag{1}$$

donde FD es la fuente de datos a procesar por el sistema, app es una aplicación en el sistema, n es el número de aplicaciones y DD es el destino de los datos.

En el modelo propuesto en este artículo, durante la etapa de monitorización se despliegan entidades llamadas centinelas encargadas de recolectar métricas de rendimiento (tiempo de servicio, volumen de datos procesados y volumen de datos en el búfer de entrada) de las aplicaciones de un sistema edge-fog-cloud.

Se asigna un centinela para cada aplicación en el sistema. Este centinela se encarga de monitorizar dicha aplicación. Esto activa la monitorización continua de las aplicaciones y del sistema, permitiendo registrar los datos procesados por cada aplicación, así como sus tiempos de servicio y respuesta. El tiempo de servicio representa el tiempo que tarda un contenido/archivo en ser procesado por una aplicación. Este tiempo incluye el tiempo de lectura del contenido desde la fuente, el tiempo de procesamiento de este contenido y el tiempo de escritura de los resultados de la aplicación. En este sentido, dado un contenido que debe ser procesado por una aplicación, el tiempo de servicio (ts) es calculado de la siguiente manera:

$$ts = l + p + e, \tag{2}$$

donde l es el tiempo de lectura, p representa el tiempo de procesamiento y e el tiempo de lectura. Este tiempo se obtiene para cada contenido procesado por cada aplicación y, a partir de este, se obtiene el tiempo de respuesta (tr), que se calcula como la sumatoria de los tiempos de servicios observados para procesar un conjunto de contenidos (ver ecuación 3).

$$tr = \sum_{c \in \mathbf{C}} [ts_c] \, | \mathbf{C} = \{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_n}\} \tag{3}$$

donde c es un contenido perteneciente al conjunto de tamaño n $\mathbf{C}$, y ts_c es el tiempo de servicio que tarda una aplicación en procesar c.

Utilizando el tiempo de servicio, es posible obtener el rendimiento de una aplicación, el cual viene dado por el volumen de datos procesados por unidad de tiempo. Por tanto, el rendimiento (D) de una aplicación es obtenido a partir del tiempo de respuesta de la siguiente manera:

$$D = \frac{tam}{rt} \tag{4}$$

donde tam es el tamaño total de los contenidos procesados y tr el tiempo de respuesta observado para procesar tal volumen de datos.

Estas métricas descritas son obtenidas por los centinelas de forma continua y enviadas a un gestor de autoescalado que se encarga de realizar las etapas de diagnóstico y rectificación del sistema edge-fog-cloud.

B. Esquema de identificación de cuellos de botella basado en el principio de Bernoulli

En este modelo, la identificación de cuellos de botella está basada en una metáfora inspirada en el principio de Bernoulli. En mecánica de fluidos, este principio describe cómo los cambios de presión en una tubería horizontal impacta en la velocidad del fluido, observándose que en zonas de alta presión la velocidad del fluido será menor, mientras que en zonas de baja presión la velocidad de este será mayor. Esta metáfora se implementa en la fase de diagnóstico, en la que diferentes componentes de la ecuación de Bernoulli se mapean con las métricas obtenidas durante la etapa de monitorización.

En este sentido, en este modelo la velocidad de un fluido es representada con el rendimiento, que mide la cantidad de datos que una aplicación puede procesar por unidad de tiempo. En un fluido, el área de la tubería porque la que es transportada impacta directamente en la presión ejercida sobre este y en su velocidad. Por lo anterior, nosotros hemos mapeado el área de la tubería con el tiempo de respuesta de esta, el cual es un indicativo de la eficiencia de una aplicación e impacta directamente sobre el rendimiento de esta. Por su parte, la presión de una tubería es mapeada con el tamaño del búfer de entrada de una aplicación, el cual se ve afectado por tiempo de respuesta de la aplicación. De esta forma, se espera que las aplicaciones más rápidas puedan procesar los datos de entrada a un ratio mayor, evitando que

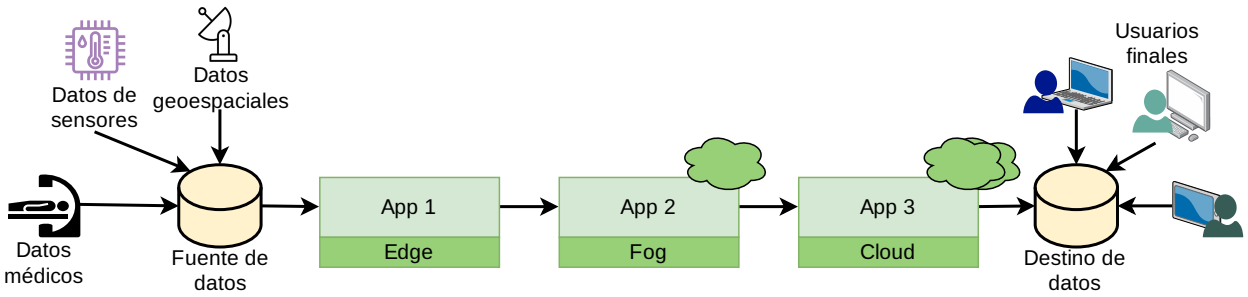


Fig. 1: Ejemplo de un sistema edge-fog-cloud.

crezca su búfer de entrada, disminuyendo la presión sobre la aplicación. Por otro lado, se espera que en las aplicaciones más lentas este búfer de entrada crezca, aumentando la presión sobre la aplicación.

En la Figura 2 se presenta una representación conceptual de cómo el rendimiento, el tiempo de respuesta y el tamaño del búfer de entrada son mapeados con los componentes de velocidad, área y presión de la ecuación de Bernoulli. Cabe resaltar, que originalmente la ecuación de Bernoulli considera un componente gravitacional, el cual no es considerado en este modelo.

El principio de Bernoulli es utilizado para describir la continuidad de la energía en fluidos a través de su recorrido por una tubería. De forma similar, en este modelo se busca conservar la continuidad en el procesamiento de datos mediante la homogeneización del rendimiento de estos, lo cual se puede expresar de la siguiente manera:

$$D_{app_1} \approx D_{app_1} \approx \dots D_{app_n}. \tag{5}$$

Para realizar este proceso de homogeneización, el primer paso es identificar aquella aplicación, o aplicaciones, que es un cuello de botella en un sistema edge-fog-cloud. Para ello, el gestor de autoescalado clasifica las aplicaciones en un sistema según las siguientes categorías:

Cuello de botella. es la etapa con el tiempo de respuesta más alto y, por lo tanto, con el menor rendimiento.

Etapas de alta presión. Son aquellas etapas con tiempos de respuesta cercanos al cuello de botella.

Etapas de baja presión. Son aquellas etapas con el menor tiempo de respuesta.

C. *Esquema de paralelismo implícito para mitigar cuellos de botella*

Una vez identificado el cuello de botella, lo siguiente es mitigarlo para que este no afecte el rendimiento de la estructura de procesamiento. Este proceso es realizado por el gestor de autoescalado en la etapa de rectificación. En este modelo, cada aplicación es manejada utilizando un esquema de paralelismo implícito basado en el patrón gestor/trabajador. En este patrón, una aplicación es clonada n veces, y cada clon es gestionado como un trabajador en el patrón. Mientras que el gestor se encarga de distribuir la carga de trabajo entrante entre los trabajadores. Para lo anterior, el gestor implementa un algoritmo de balanceo de carga.

En este modelo, tanto el gestor como los trabajadores se implementan como contenedores virtuales, lo que permite agilizar el despliegue de las aplicaciones y, por lo tanto, disminuir las tareas de gestión requeridas para escalar una aplicación. Estos contenedores virtuales encapsulan la aplicación, la cual es instalada en el contenedor como un binario o código fuente, así como las dependencias de la aplicación (sistema operativo, librerías y variables de entorno).

Por otro lado, el gestor de los patrones utilizados en este trabajo incluye un balanceador de carga basado en el algoritmo *Two Choices* [33], en el cual para distribuir un contenido, dos trabajadores son seleccionados aleatoriamente y se selecciona aquel trabajador con la menor utilización en su búfer de entrada.

Este patrón permite incrementar la eficiencia de una aplicación, permitiendo aumentar su rendimiento y reduciendo su tiempo de respuesta. En este modelo, la idea es realizar la mitigación continua de los cuellos de botella en un sistema edge-fog-cloud mediante el despliegue de patrones y su autoescalado, para evitar tiempos de ociosidad y acumulación de tareas en el sistema.

Siguiendo con la metáfora basada en el principio de Bernoulli establecida, estos patrones tienen por objetivo reducir el área de la aplicación (tiempo de servicio) para disminuir su presión e incrementar su velocidad. Por lo tanto, en este modelo este es un proceso al que llamamos *despresurización*, el cual es realizado agregando nuevos trabajadores a una aplicación y es exitoso cuando el rendimiento de la aplicación aumenta.

El número de trabajadores que se agrega a un cuello de botella se obtiene mediante un análisis del tiempo de respuesta, rendimiento y tareas encoladas en cola de la aplicación identificada como cuello de botella y aquellas etiquetadas como *etapas de alta presión*.

En este sentido, el primer paso es obtener el tiempo de respuesta medio de las etapas de alta presión, lo cual se obtiene de la ecuación:

$$\hat{TR} = \frac{\sum_{a \in \mathbf{A}} TR_a}{n}, \tag{6}$$

donde $\hat{TR}$ es el tiempo de respuesta medio, $\mathbf{A}$ es el conjunto de aplicaciones etiquetadas como etapas de alta presión, a es una aplicación en este conjunto, TR_a es el tiempo de respuesta de esta aplicación y

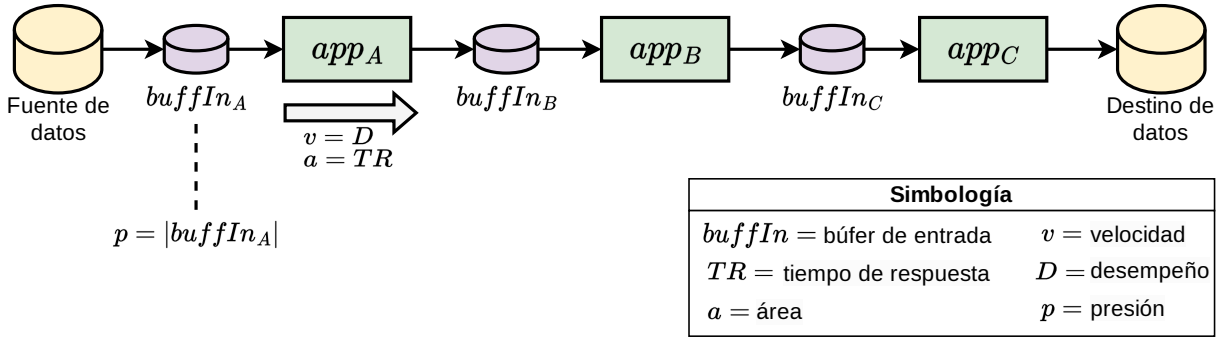


Fig. 2: Representación conceptual de las métricas obtenidas a partir de un sistema edge-fog-cloud y su mapeo con los componentes de la ecuación de Bernoulli.

n es el número de aplicaciones en el conjunto **A**. En este sentido, el número de trabajadores de la aplicación identificada como cuello de botella es obtenido de la siguiente manera:

$$nt = \frac{\hat{TR}}{TR_{cb}}, \tag{7}$$

donde TR_{cb} es el tiempo de respuesta observado en el cuello de botella, y nt es un número entero que representa el número de trabajadores propuesto. El número de trabajadores que se puede desplegar por aplicación se encuentra limitado por los recursos físicos disponibles en la infraestructura, en especial por la cantidad de núcleos físicos disponibles. En este sentido, cuando el número de trabajadores nt propuesto es mayor al número de núcleos físicos nf disponibles, nt será igual a nf.

Una vez que se tiene una nueva configuración en el número de trabajadores, esta es entregada a los centinelas, quienes se encargan de gestionar, en caso de ser necesario, del despliegue de nuevas trabajadores en una aplicación. Este despliegue se realiza mediante la instanciación de nuevos contenedores virtuales y es transparente para el usuario final.

D. *Implementación de un prototipo basado en el modelo propuesto*

En la Figura 3 se presenta la arquitectura en pila de un prototipo diseñado a partir del modelo descrito. En la parte superior de la arquitectura se encuentra el gestor del sistema edge-fog-cloud, el cual incluye una componente de control de acceso basado en tokenización para solo validar que solo usuarios autorizados puedan acceder al sistema, sus aplicaciones, datos y resultados. Posteriormente, se encuentra un proxy encargado de redireccionar las peticiones de los usuarios a las aplicaciones que deben responder su solicitud. El gestor del sistema incluye el gestor de autoescalado, el cual se encarga de identificar cuellos de botella en el sistema y mitigarlos utilizando un esquema de paralelismo implícito.

En la siguiente capa de la arquitectura se encuentran los centinelas encargados de monitorizar las aplicaciones del sistema. Estos centinelas implementan un modelo de comunicación basado en publicación/suscripción (pub/sub) para comunicarse tanto con el gestor de autoescalado como con las aplicaciones. En este escenario, las aplicaciones publican sus

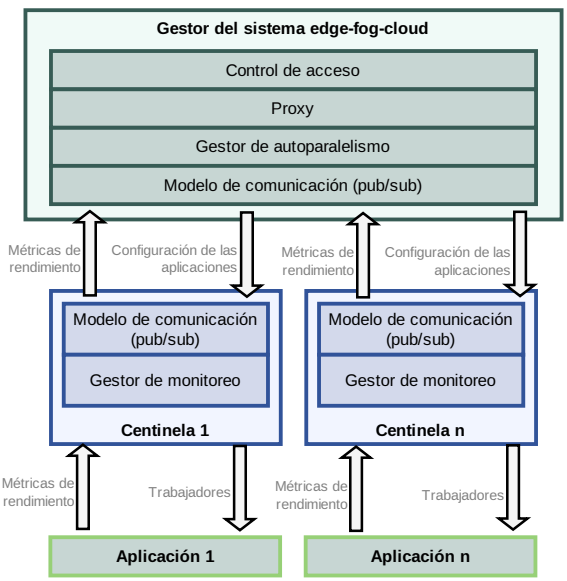


Fig. 3: Arquitectura en capas de un prototipo desarrollado en el modelo propuesto.

resultados de rendimiento y los centinelas se suscriben a estas aplicaciones para consumir estos resultados. De la misma manera, los centinelas y el gestor de paralelismo intercambian datos entre sí, para gestionar la identificación de los cuellos de botella y el despliegue de la nueva configuración propuesta por el gestor.

Los componentes de la arquitectura están desarrollados en el lenguaje de programación C++17. Para gestionar los contenedores virtuales utilizados en los patrones paralelos se utiliza la plataforma Docker. Para habilitar el intercambio de datos entre diferentes aplicaciones desplegadas en diferentes infraestructuras, se utiliza una red de distribución de contenidos llamada Painal [34], implementada del lado del servidor en PHP y del lado del cliente en Java.

IV. Evaluación experimental y resultados

En la presente Sección se presenta la evaluación experimental del método propuesto en este artículo. Para conducir esta evaluación se diseñó un estudio de caso basado en el procesamiento de imágenes satelitales. En la Figura 4 se presenta el diseño del sistema edge-fog-cloud diseñado para este estudio de caso. Dicho sistema está conformado por las siguientes aplicaciones:

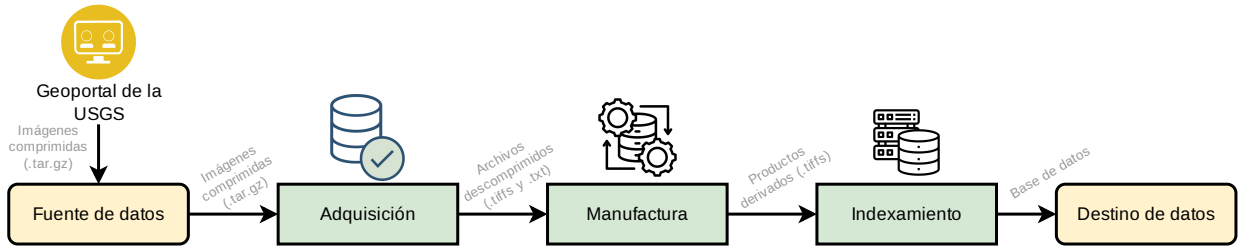


Fig. 4: Representación conceptual del sistema edge-fog-cloud diseñado para conducir un estudio de caso basado en el procesamiento de imágenes satelitales.

Tabla I: Características de la infraestructura utilizada para experimentación.

Equipo	Imágen	Características de hardware		
		Núcleos	RAM (GB)	Almacenamiento (GB)
EC2-Acq	Amazon EC2 VM m5zn.6xlarge	24	32	
EC2-Man	Amazon EC2 VM c4.4xlarge	16	64	600
EC2-Vis	Amazon EC2 VM r5.xlarge	4	32	

Adquisición. Esta aplicación se encarga de descomprimir las imágenes satelitales, las cuales originalmente se encuentran almacenadas en formato *tar.gz*. Además, esta aplicación se encarga de leer los metadatos de las imágenes satelitales y enriquecerlos para obtener metadatos extra, tales como la transformación de las coordenadas originalmente en formato numérico a una dirección en lenguaje natural.

Manufactura. Esta aplicación se encarga de producir productos derivados a partir de las bandas de las imágenes satelitales. Entre estos productos se incluyen índices que indican el nivel de vegetación en la imagen, la cantidad de mantos acuíferos y el área de vegetación quemada (por ejemplo, por un incendio forestal).

Visualización. Esta aplicación se encarga de indexar los resultados de las etapas anteriores en una base de datos MongoDB [35].

A. Infraestructura y conjunto de datos de prueba

En la Tabla I se muestran las principales características de la infraestructura utilizada para conducir el estudio de caso presentado en esta evaluación experimental. La infraestructura está compuesta por tres máquinas virtuales desplegadas en Amazon EC2. Utilizando esta infraestructura, se simuló un escenario en donde las aplicaciones de un sistema edge-fog-cloud se encuentran distribuidas en diferentes infraestructuras.

Para conducir este estudio de caso se utilizó una fuente de datos compuesta por 70 imágenes satelitales LandSat8 [36], adquiridas desde el portal *USGS Earth Explorer* [37]. El volumen total de las imágenes es de 78 GB, con un tamaño promedio de 1.11 GB por imagen.

B. Resultados

El sistema edge-fog-cloud ilustrado en la Figura 4 fue evaluado utilizando el modelo propuesto en este artículo. En el primer experimento de esta evaluación, nosotros registramos las decisiones tomadas por el gestor de autoescalado para gestionar los cuellos de botella en el sistema. En la Figura 5 se muestra en el eje vertical la cantidad de trabajadores creados

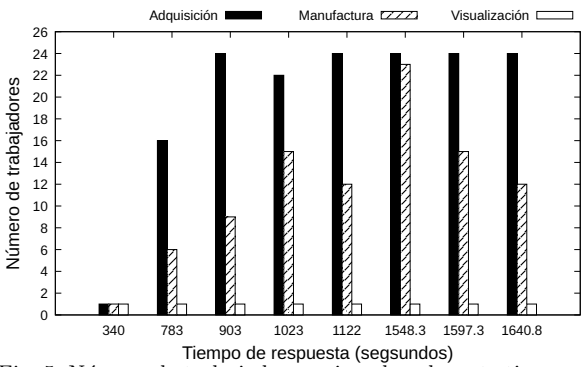


Fig. 5: Número de trabajadores asignados, durante tiempo de ejecución, a cada aplicación en el sistema edge-fog-cloud.

en tiempo de ejecución (eje horizontal) para mitigar estos cuellos de botella. Lo primero que podemos observar en la Figura 5 es que en la configuración inicial todas las soluciones inician con un trabajador en su patrón, lo que representaría que se está ejecutando una versión secuencial de estas. Al avanzar en la ejecución, se observa que las la etapa de adquisición es aquella a la que se agregan una mayor cantidad de trabajadores paralelos, lo que indica que esta etapa es un cuello de botella en el sistema. El número máximo de trabajadores asignados a esta etapa es de 24, lo que es igual a utilizar todos los núcleos físicos disponibles en la infraestructura.

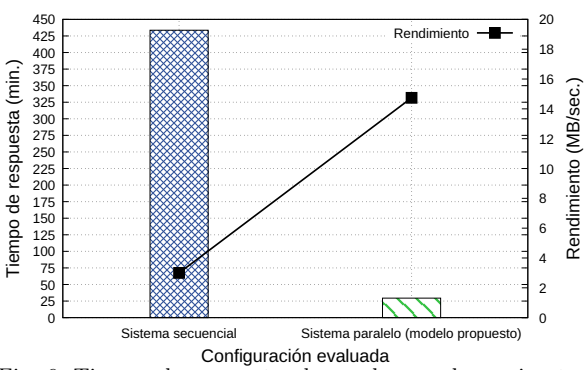


Fig. 6: Tiempo de respuesta observado cuando se ejecuta el sistema edge-fog-cloud con una versión secuencial y la versión paralela implementada con el modelo propuesto.

Para medir el impacto de estas decisiones en el tiempo de respuesta del sistema, se realizó una com-

paración entre una versión secuencial del sistema (sin paralelismo) y una versión implementando el modelo propuesto. En la Figura 6 se muestra en el eje vertical el tiempo de respuesta observado para procesar las 70 imágenes con el sistema en su versión secuencial y su versión paralela implementando el modelo propuesto (eje horizontal). Como se puede observar, la versión secuencial procesó los datos en 433,63 minutos, mientras que la versión paralela hizo el mismo proceso en 29,42 minutos. Esto representa una aceleración de 14.73x, y una ganancia en el tiempo de respuesta del 93.21 %. Esta reducción en el tiempo de respuesta resulta clave en sistemas edge-fog-cloud utilizados en procesos de toma de decisiones, en donde es crítico tener disponibles los datos e información en el menor tiempo posible.

V. Conclusiones

En este artículo se presentó un modelo de sincronización para la mitigación de cuellos de botella en sistemas edge-fog-cloud. Este modelo identifica y mitiga en tiempo de ejecución estos cuellos de botella utilizando una metáfora inspirada en el principio de Bernoulli, en el cual las aplicaciones más lentas son clasificadas como etapas de alta presión, mientras que las más rápidas como de baja presión. En este sentido, el modelo presentado busca homogeneizar el rendimiento de las aplicaciones mediante el escalamiento en tiempo de ejecución de la etapa con la presión más alta (cuello de botella). Para realizar este escalado, las aplicaciones se manejan como patrones de paralelismo basados en contenedores virtuales.

Se ha llevado a cabo un estudio de caso basado en la gestión de imágenes satelitales utilizando un prototipo del modelo propuesto. La evaluación reveló la factibilidad de implementar dicho modelo en escenarios reales, permitiendo la identificación y mitigación automática de los cuellos de botella en el sistema edge-fog-cloud evaluado. Además, en la evaluación experimental se observó como las decisiones tomadas por el modelo logran incrementar la eficiencia del sistema, permitiendo reducir el tiempo de respuesta requerido para procesar un lote de datos. Como trabajo futuro se exploraran escenarios de calidad de servicio (QoS), para permitir que el modelo no solo gestione cuellos de botella, sino que además permita gestionar otros requerimientos de los usuarios.

Agradecimientos

Este trabajo ha sido apoyado por el Ministerio de Ciencia e Innovación bajo el proyecto "New Data Intensive Computing Methods for High-End and Edge Computing Platforms (DECIDE)". Ref. PID2019-107858GB-I00.

Referencias

[1] D. Reinsel, J. Gantz, and J. Rydning, "The digitization of the world: from edge to core," *Framingham: International Data Corporation*, 2018.

[2] Nitinder Mohan and Jussi Kangasharju, "Edge-fog cloud: A distributed cloud for internet of things computations," in *2016 Cloudification of the Internet of Things (CIoT)*. IEEE, 2016, pp. 1–6.

[3] Dante D Sánchez-Gallegos, Alejandro Galaviz-Mosqueda, JL Gonzalez-Compean, Salvador Villarreal-Reyes, Aldo E Perez-Ramos, Diana Carrizales-Espinoza, and Jesus Carretero, "On the continuous processing of health data in edge-fog-cloud computing by using micro/nanoservice composition," *IEEE Access*, vol. 8, pp. 120255–120281, 2020.

[4] Dana Petcu, "Multi-cloud: expectations and current approaches," in *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, 2013, pp. 1–6.

[5] Jiangshui Hong, Thomas Dreibholz, Joseph Adam Schenkel, and Jiaxi Alessia Hu, "An overview of multi-cloud computing," in *Web, Artificial Intelligence and Network Applications: Proceedings of the Workshops of the 33rd International Conference on Advanced Information Networking and Applications (WAINA-2019) 33*. Springer, 2019, pp. 1055–1068.

[6] Dante D Sanchez-Gallegos, JL Gonzalez-Compean, Jesus Carretero, Heidy M Marin-Castro, Andrei Tchernykh, and Raffaele Montella, "Puzzlemesh: A puzzle model to build mesh of agnostic services for edge-fog-cloud," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1334–1345, 2022.

[7] Guadalupe Ortiz, Meftah Zouai, Okba Kazar, Alfonso Garcia-de Prado, and Juan Boubeta-Puig, "Atmosphere: Context and situational-aware collaborative iot architecture for edge-fog-cloud computing," *Computer Standards & Interfaces*, vol. 79, pp. 103550, 2022.

[8] Sarah Cohen-Boulakia, Khalid Belhajjame, Olivier Collin, Jérôme Chopard, Christine Froidevaux, Alban Gaignard, Konrad Hinsen, Pierre Larmande, Yvan Le Bras, Frédéric Lemoine, et al., "Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities," *Future Generation Computer Systems*, vol. 75, pp. 284–298, 2017.

[9] Maria Alejandra Rodriguez and Rajkumar Buyya, "A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 8, pp. e4041, 2017.

[10] Gerardo A Vazquez-Martinez, JL Gonzalez-Compean, Victor J Sosa-Sosa, Miguel Morales-Sandoval, and Jesus Carretero Perez, "Cloudchain: A novel distribution model for digital products based on supply chain principles," *International Journal of Information Management*, vol. 39, pp. 90–103, 2018.

[11] Papa Senghane Diouf, Aliou Boly, and Samba Ndiaye, "Variety of data in the etl processes in the cloud: State of the art," in *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*. IEEE, 2018, pp. 1–5.

[12] José A Joao, M Aater Suleman, Onur Mutlu, and Yale N Patt, "Bottleneck identification and scheduling in multithreaded applications," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 223–234, 2012.

[13] Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Meihui Zhang, "In-memory big data management and processing: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1920–1948, 2015.

[14] Peng Sun, Minlan Yu, Michael J Freedman, and Jennifer Rexford, "Identifying performance bottlenecks in cdns through tcp-level monitoring," in *Proceedings of the first ACM SIGCOMM workshop on Measurements up the stack*, 2011, pp. 49–54.

[15] Jorge Luis Ortega-Arjona, *Patterns for Parallel Software Design*, Wiley Publishing, 1st edition, 2010.

[16] Jun Zhao, Jose Manuel Gomez-Perez, Khalid Belhajjame, Graham Klyne, Esteban Garcia-Cuesta, Aleix Garrido, Kristina Hettne, Marco Roos, David De Roure, and Carole Goble, "Why workflows break—understanding and combating decay in taverna workflows," in *2012 IEEE 8th International Conference on E-Science*. IEEE, 2012, pp. 1–9.

[17] Nelson Mimura Gonzalez, Tereza Cristina Melo de Brito Carvalho, and Charles Christian Miers, "Cloud resource management: towards efficient execution of large-scale scientific applications and workflows on complex infrastructures," *Journal of Cloud Computing*, vol. 6, no. 1, pp. 1–20, 2017.

[18] Michael Albrecht, Patrick Donnelly, Peter Bui, and Douglas Thain, "Makeflow: A portable abstraction for da-

ta intensive computing on clusters, clouds, and grids," in *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, 2012, pp. 1–13.

[19] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin M Wozniak, Ian Foster, et al., "Parsl: Pervasive parallel programming in python," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, 2019, pp. 25–36.

[20] Liang Bao, Chase Wu, Xiaoxuan Bu, Nana Ren, and Mengqing Shen, "Performance modeling and workflow scheduling of microservice-based applications in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 9, pp. 2114–2129, 2019.

[21] Dante Domizzi Sánchez-Gallegos, Diana Di Luccio, Sokol Kosta, JL Gonzalez-Compean, and Raffaele Montella, "An efficient pattern-based approach for workflow supporting large-scale science: The dagonstar experience," *Future Generation Computer Systems*, vol. 122, pp. 187–203, 2021.

[22] Chunming Hu, Jianyong Zhu, Renyu Yang, Hao Peng, Tianyu Wo, Shiqing Xue, Xiaoqiang Yu, Jie Xu, and Rajiv Ranjan, "Toposch: Latency-aware scheduling based on critical path analysis on shared yarn clusters," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020, pp. 619–627.

[23] Reyes, Gonzalez, Sosa, Carretero, and Garcia, "Kulla, a container-centric construction model for building infrastructure-agnostic distributed and parallel applications," *JSS*, p. 110665, 2020.

[24] Valentina Armenise, "Continuous delivery with jenkins: Jenkins solutions to implement continuous delivery," in *Proceedings of the Third International Workshop on Release Engineering*. IEEE, 2015, pp. 24–27.

[25] Mohak Chadha, Jophin John, and Michael Gerndt, "Extending slurm for dynamic resource-aware adaptive batch scheduling," *arXiv preprint arXiv:2009.08289*, 2020.

[26] E Fajardo, Jeffrey Dost, Burt Holzman, T Tannenbaum, J Letts, A Tiradani, B Bockelman, J Frey, and Darius Mason, "How much higher can htcondor fly?," in *JPCS*, 2015, vol. 664, p. 062014.

[27] Sheng Wang, Zhijun Ding, and Changjun Jiang, "Elastic scheduling for microservice applications in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 98–115, 2020.

[28] Anshul Jindal, Vladimir Podolskiy, and Michael Gerndt, "Performance modeling for cloud microservice applications," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019, pp. 25–32.

[29] Changyuan Lin and Hamzeh Khazaei, "Modeling and optimization of performance and cost of serverless applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 615–632, 2020.

[30] Polona Štefanič, Matej Cigale, Andrew C Jones, Louise Knight, Ian Taylor, Cristiana Istrate, George Suciu, Alexandre Ulisses, Vlado Stankovski, Salman Taherizadeh, et al., "Switch workbench: A novel approach for the development and deployment of time-critical microservice-based cloud-native applications," *Future Generation Computer Systems*, vol. 99, pp. 197–212, 2019.

[31] Christina Terese Joseph and K Chandrasekaran, "Intma: Dynamic interaction-aware resource allocation for containerized microservices in cloud environments," *Journal of Systems Architecture*, vol. 111, pp. 101785, 2020.

[32] Miguel Santiago-Duran, JL Gonzalez-Compean, André Brinkmann, Hugo G Reyes-Anastacio, Jesus Carretero, Raffaele Montella, and Gregorio Toscano Pulido, "A gearbox model for processing large volumes of data by using pipeline systems encapsulated into virtual containers," *Future Generation Computer Systems*, 2020.

[33] Felix Garcia-Carballeira, Alejandro Calderon, and Jesus Carretero, "Enhancing the power of two choices load balancing algorithm using round robin policy," *Cluster Computing*, vol. 24, no. 2, pp. 611–624, 2021.

[34] Diana Carrizales-Espinoza, Dante D Sanchez-Gallegos, JL Gonzalez-Compean, and Jesus Carretero, "Fedflow: A federated platform to build secure sharing and synchronization services for health dataflows," *Computing*, vol. 105, no. 5, pp. 1019–1037, 2023.

[35] Kyle Banker, Douglas Garrett, Peter Bakkum, and Shaun Verch, *MongoDB in action: covers MongoDB version 3.0*, Simon and Schuster, 2016.

[36] Nicholas E Young, Ryan S Anderson, Stephen M Chignell, Anthony G Vorster, Rick Lawrence, and Paul H Evangelista, "A survival guide to landsat preprocessing," *Ecology*, vol. 98, no. 4, pp. 920–932, 2017.

[37] USGS, "Earthexplorer - usgs," July 31, 2020.

Entorno de monitorización y reconfiguración dinámica del DVFS para ahorro de energía

Alberto Cascajo, David E. Singh, Jesus Carretero [1]

Resumen— *EpiGraph* **es un simulador paralelo basado en agentes que es capaz de modelar la propagación de virus como la gripe y COVID-19 (incluyendo sus variantes). La simulación tiene en cuenta varios modelos que reproducen, de forma bastante precisa, el entorno donde ocurren las infecciones y contagios. Para la obtención de resultados hay que realizar cientos de simulaciones de diferentes escenarios, lo que implica una gran cantidad de horas de computación. Esto se traduce en un alto consumo energético.** `LIMITLESS` **en cooperación con** *FlexMPI* **ofrecen un entorno de monitorización y planificación ligero y escalable que proporciona una visión holística del sistema. En este trabajo se desarrolla un entorno de despliegue donde el sistema es capaz de encontrar una configuración para aplicar DVFS que mantiene los tiempos de ejecución de las simulaciones mientras se reduce el consumo de energía. La característica más importante es que no requiere modificación alguna por parte de los desarrolladores de las aplicaciones, ya que el framework interactúa con el sistema y no con la aplicación en sí.**

Palabras clave— **DVFS, high-performance computing, simulación, energía.**

I. Introducción

EpiGraph es un simulador paralelo basado en agentes que modela el comportamiento de virus como la gripe o COVID-10, incluyendo sus variantes, y es capaz de predecir la curva de propagación del virus sobre la población. Utiliza varios modelos que, de forma realista, reproducen el entorno donde tienen lugar los contagios, por ejemplo un modelo social muy detallado, modelos de vacunación o de transporte, que ajustan la propagación entre individuos mientras éstos transitan entre estados de un modelo epidemiológico SEIR *(Susceptible, Expuesto, Infectado, Removido)* extendido. Además, *EpiGraph* también considera intervenciones no farmacológicas como el uso de tests, mascarillas y distanciamiento social por parte de la población.

Las propuestas basadas en agentes tienen la capacidad de modelar características y patrones de cada individuo, lo que puede resultar en simulaciones mucho más realistas en comparación a otras alternativas [1] [2]. Una de las características principales de *EpiGraph* es que se basa en datos reales tanto para los individuos como para los patrones de interacción entre ellos. *EpiGraph* utiliza diferentes fuentes de datos heterogéneas como entrada y ajustando los diferentes parámetros del modelo de agentes [3].

Desde 2021, nuestro equipo ha estado proporcionando al Ministerio de Salud de España resultados sobre la incidencia del COVID-19 sobre la población. Proporcionar estos datos estadísticos y significativos requiere realizar distintas simulaciones varias veces. En nuestras pruebas, ejecutamos en torno a 200 mil horas de simulaciones en 110 cores, lo que implica una ingente cantidad de recursos utilizados (especialmente en términos de energía consumida). Esto creó una fuerte motivación para tratar de reducir la energía consumida mediante el uso de un entorno capaz de optimizar la relación *energía consumida vs tiempo de ejecución*.

Por lo tanto, la contribución principal de este trabajo se centra en realizar una optimización en el consumo de la energía, evitando en la medida de lo posible, la manipulación del código de las aplicaciones (en este caso *EpiGraph*. Para ello, se ha diseñado un entorno de despliegue compuesto por un monitor a nivel de sistema que es capaz de gestionar el DVFS de cada nodo en uso, y un monitor a nivel de aplicación que permite obtener métricas de rendimiento de cada aplicación, con grano más fino. Este entorno está formado por dos componentes principales: `LIMITLESS`, que es el monitor encargado de recolectar periódicamente las métricas de rendimiento a nivel de sistema (tales como el consumo de CPU, de memoria, energía, etc.) y modificar el DVFS en base a la política seleccionada, y *FlexMPI*, que es el encargado de proporcionar las métricas de rendimiento a nivel de aplicación, entre las que se incluyen los FLOPS, el tiempo de CPU, tiempo de E/S, etc.).

La estructura de este artículo es la siguiente: Sección II muestra trabajos relevantes relacionados con nuestra propuesta; La sección III describe la organización de la arquitectura del entorno; La sección IV describe de forma más detallada el caso de uso *EpiGraph*; La sección V proporciona una evaluación práctica donde se muestran los beneficios de la adaptación dinámica del DVFS para simulaciones largas que ejecutan diferentes fases (como el caso de uso); Finalmente, la Sección VI muestra las principales conclusiones.

II. Antecedentes

Dynamic Voltage and Frequency Scaling (DVFS) es una técnica utilizada en entornos de computación para optimizar el consumo de energía. Los beneficios principales que se pueden extraer de la literatura [4] [5] se pueden resumir en: (1) reducción del consumo de energía debido a la bajada de la frecuencia de reloj y el voltaje de los componentes como la CPU y la memoria según los requisitos del usuario; (2) menor necesidad de disipación de calor debido a la misma bajada de frecuencia y voltaje, lo que es especialmente útil en entornos HPC donde las plataformas disponen de sofisticados sistemas de refrigeración y

[1]Computer Science and Engineering Department, Universidad Carlos III de Madrid, Spain, e-mail: {acascajo,dexposit,jcarrete}@inf.uc3m.es

disipación del calor, los cuales, a su vez, consumen menos energía; (3) poder ajustar dinámicamente el DVFS puede mejorar la eficiencia energética de los sistemas aumentando o disminuyendo la frecuencia y el voltaje en función de las necesidades que se tenga en cada momento, lo que, a futuro, puede suponer también una vía para conservar el hardware en buen estado, evitando mantener los altos niveles de rendimiento cuando no es necesario.

Los trabajos relacionados con las optimizaciones en el consumo de energía están divididos según dos puntos de vista principales: métodos a nivel de sistema y métodos a nivel de aplicación. Mair et al. [6] cuantifican la eficiencia energética de los supercomputadores utilizando las listas Top500 [7] y Green500 [8]. Los autores proponen una serie de métricas que mide la escalabilidad del sistema y el rendimiento para evaluar la eficiencia energética. Una de las principales conclusiones, la cual pone el foco en el DVFS y los algoritmos relacionados con la energía, es que las plataformas más eficientes (energéticamente hablando) tienen una escala pequeña, confirmando que sistemas de gran escala no suelen tener en cuenta el consumo de energía.

Para poder lidiar con el consumo energético, una de las técnicas más utilizadas es la de reconfigurar el DVFS de los nodos según el perfil de las aplicaciones. En [9] los autores presentan un algoritmo eficiente en cuanto al consumo energético que es capaz de modificar el DVFS de forma dinámica en distintos cores. Los autores identifican grupos de aplicaciones que, por sus características, podrían beneficiarse de su propuesta. Sin embargo, la evaluación y los resultados se obtienen utilizando un conjuto de microbenchmarks y un framework que permite al usuario identificar fases de aplicaciones y modificar, de forma local, el DVFS. `LIMITLESS` crea un perfil de las aplicaciones para evaluar el rendimiento, en lugar de segmentar el código de las aplicaciones para aplicar los cambios (quiere decir que no es invasivo para las aplicaciones, como si otras propuestas), lo que hace nuestra alternativa más general. Por otro lado, los autores también discuten las limitaciones del uso del DVFS en los entornos HPC, destacando una característica que nosotros mismos hemos podido comprobar: el DVFS y el HyperThreading son incompatibles porque dos threads podrían estar compartiendo el mismo core físico, lo que generaría problemas en el rendimiento de la aplicación u otros procesos que estuviesen ejecutándose en la máquina.

Bratek et al. [10] también realizan un estudio sobre los beneficios de utilizar el DVFS de forma dinámica para incrementar el rendimiento de aplicaciones paralelas en sistemas multicore. Proponen una metodología para adaptar la frecuencia y el voltaje de forma dinámica para cada core en uso, dependiendo del trabajo que éste tenga que hacer. Sin embargo, esta optimización requiere un algoritmo más sofisticado para manejar la información que proporcione los datos para la toma de decisiones (tales como los procesos en ejecución, los threads si los hubiera, el perfil de cada uno, qué cores ejecutan qué procesos, etc.). Esto consume muchos recursos y requiere de una continua monitorización de grano muy fino por cada aplicación.

Siguiendo esta línea de investigación, Gupta et al. [11] presentan un algoritmo que es capaz de evaluar diferentes configuraciones de DVFS en base a un modelo de regresión que, según sus resultados, es capaz de ahorrar un 20 % de energía en comparación a sus resultados sin utilizar su algoritmo (caso base). Sin embargo, los autores basan sus conclusiones en resultados obtenidos mediante simulación, no proporcionando datos en entorno real. No obstante, cabe destacar que las comparaciones que realizan de su solución la realizan con otras alternativas similares que también simulan sus resultados, por lo que la comparación podría ser correcta, si bien aún están desarrollando su solución.

En cuanto al uso de modelos de regresión lineal, como el caso del último estudio presentado, a veces son menos precisos a la hora de predecir las métricas de rendimiento que otros algoritmos de aprendizaje automático, sobre todo en casos con una alta variabilidad en las métricas - como suele ser el caso en aplicaciones HPC [12] [13].

Finalmente, merece la pena mencionar que estos estudios como [14] donde los autores demuestran que el rendimiento y del consumo energético también debe ser estudiado en GPUs debido a la creciente demanda de dicho hardware para operar en entornos de machine learning.

III. Arquitectura del entorno

`LIMITLESS` es un monitor a nivel de nodo escalable y ligero que brinda información sobre los recursos disponibles del sistema y el rendimiento de las aplicaciones que se están ejecutando. La figura 1 muestra una descripción general de su arquitectura. Se puede ver que coopera con otros componentes del sistema como el planificador de aplicaciones, y FlexMPI, gracias al que obtiene métricas de rendimiento a nivel de aplicación. `LIMITLESS` engloba cuatro componentes principales: el *monitor de sistema*, *ElasticSearch* como base de datos para series temporales, *Kibana* como visualizador de datos, y *Analytics* que se encarga del análisis de los datos recolectados así como de la toma de decisiones. Nótese que, además de analizar y tomar decisiones sobre el DVFS y la energía, `LIMITLESS` también realiza evaluación de potenciales fallos, notificaciones a los usuarios, predicción del rendimiento, etc. [15].

El componente *Analytics* es el encargado de lidiar con el almacenamiento, la visualización, la comunicación con el planificador de aplicaciones y es responsable de la predicción del rendimiento de las aplicaciones en ejecución. Además, es el encargado de analizar los modelos de rendimiento recolectados para ajustar, de forma dinámica, el DVFS cuando sea necesario.

La figura 1 muestra el flujo de los datos en la arquitectura propuesta. Una vez que se ejecuta una

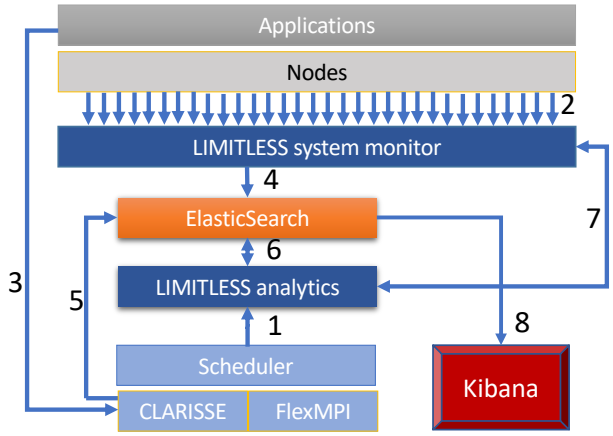


Fig. 1: Arquitectura del sistema: `LIMITLESS` como monitor de sistema en cooperación con FlexMPI como monitor a nivel de aplicación y runtime para maleabilidad.

aplicación, el programador notifica a *analytics* (flecha 1) las características de la aplicación. Desde ese momento, se recopilan dos grupos de métricas de rendimiento simultáneamente: a nivel de nodo para el monitor (flecha 2) ya nivel de aplicación para FlexMPI (flecha 3). Al final de cada intervalo de muestreo, ambos conjuntos de datos son procesados y enviados a la base de datos (flechas 4 y 5). A partir de ese momento, *analytic* inicia la creación de un modelo de rendimiento de la aplicación en base a la información almacenada que se va actualizando (flecha 6). Finalmente, una vez que se genera el modelo de aplicación, se realizan diferentes análisis para identificar las distintas fases que se dan a lo largo de una ejecución y poder ajustar el DVFS en función de su tipología (flecha 7). Durante todo el proceso, Kibana visualiza en un panel de datos (flecha 8) el estado del clúster.

A. Monitor de sistema

El monitor está diseñado para capturar el rendimiento de los nodos y aplicaciones en sistemas de gran escala. La monitorización y recolección de métricas de rendimiento tiene lugar periódicamente, y dicho intervalo de muestreo se puede configurar en un rango de tiempo de horas a segundos, incluyendo valores subsegundo, teniendo en cuenta que hay una relación inversa entre el intervalo de muestreo y la sobrecarga de monitorización. Sin embargo, basándonos en nuestras pruebas, la sobrecarga por la monitorización es inferior a 1 % en términos de CPU hasta 200ms, con una huella en memoria inferior a los 4MB, por lo que consideramos que el monitor tiene un impacto sobre los nodos y las aplicaciones apenas destacable.

El sistema de monitorización se compone de tres elementos principales: un monitor de *`LIMITLESS` Daemon* (LDM) en cada nodo, el cual recopila regularmente métricas de rendimiento; un conjunto de agregadores de *`LIMITLESS` DaeMon* (LDA), los cuales transmiten la información de los LDM a otros agregadores o servidores; y el servidor de *`LIMITLESS` DaeMon* (LDS), encargado de recopilar y almacenar la información de monitorización en ElasticSearch. La implementación sigue una estructura jerárquica, donde los datos fluyen desde los nodos (LDMs) hacia el servidor principal (LDS) y la base de datos. El usuario puede definir la jerarquía, aunque se recomienda diseñarla de manera óptima para que coincida con la topología de la red. Este tipo de monitorización utiliza una arquitectura predefinida basada en redes de superposición *Based-Tree* (TBON), que organiza el flujo de información en forma jerárquica y en estructura de árbol.

B. Integración con FlexMPI

FlexMPI se integra en `LIMITLESS` con dos objetivos principales: (1) proporcionar la capacidad de migrar procesos de aplicaciones (mediante la maleabilidad) desde nodos con un rendimiento deficiente hacia nodos donde puedan ejecutarse sin interferencias, y (2) utilizar los contadores de rendimiento que obtiene para disponer de más información sobre el sistema y las aplicaciones que se están ejecutando. De esta manera, el monitor del sistema proporciona dos niveles de datos: información sobre el rendimiento actual de los nodos reales (nivel de hardware) e información sobre el rendimiento de las aplicaciones que se ejecutan en el clúster (nivel de software).

Mediante este enfoque, es posible incluir información detallada sobre las características de la aplicación. Por ejemplo, se puede registrar la duración de cada fase de entrada/salida con una precisión de milisegundos. Esta información se utilizará para realizar una modelización de aplicaciones más precisa que posteriormente será utilizada por el monitor para mejorar el proceso de planificación de aplicaciones y refinado del DVFS para reducir el consumo de energía de los nodos.

IV. EpiGraph

La figura 2 muestra las distintas fases que tiene que ejecutar cada simulación en *EpiGraph*. Los datos de entrada se obtienen desde distintas fuentes de datos (*Data aquisition*), tanto artículos de investigación como bases de datos públicas y privadas. Estos datos son extremadamente heterogéneos y deben ser procesados en una segunda etapa (*Data pre-processing*). Con los datos procesados, la fase de simulación comienza (*Simulation*). Una vez que la simulación ha terminado, se genera una serie de ficheros de trazas con el estado de cada individuo en cada intervalo de tiempo y en cada lugar. Esta información es realmente útil pues contiene información como las características de cada individuo (estado de salud, edad, profesión, etc.), acciones aplicadas (vacunación, viajes, etc.), entre otros. Para proporcionar análisis inteligibles partiendo de los resultados de la simulación, se necesita una nueva etapa para procesar los datos (*Data post-processing*). Esta etapa genera ficheros de trazas específicos de acuerdo a la información que se desea analizar o mostrar. Finalmente, la última etapa utiliza la información generada para obtener datos estadísticos, resumir los resultados de la simulación y mostrarlos al usuario (*Data analysis*).

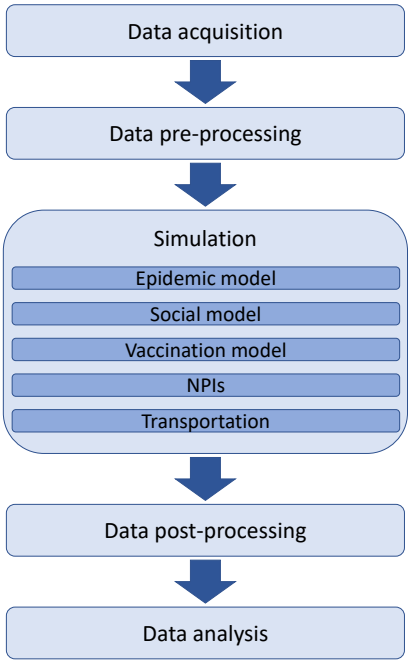


Fig. 2: Flujo de fases de cada simulación en *EpiGraph*.

A continuación, se describen brevemente los modelos utilizados en el simulador, que son: un modelo social, un modelo epidemiológico, un modelo de vacunación, y un modelo de movimiento de la población.

A. Modelo epidemiológico

El modelo epidemiológico es un modelo SEIR estocástico con estados *incubación*, *asintomático*, *sintomático*, *muerto*, *hospitalizado*, *recuperado*, y *estados de vacunados*. De forma diferente a otros modelos analíticos comunes basados en ecuaciones diferenciales, *EpiGraph* decide la duración de los diferentes estados y las transiciones de forma probabilística.

A continuación se ofrece una breve descripción de los distintos estados:

- **Incubación:** Individuo infectado pero sin síntomas, aún no capaz de transmitir el virus.
- **Asintomático:** Individuo que no tiene síntomas pero es capaz de transmitir el virus con una cierta probabilidad.
- **Sintomático:** El individuo muestra síntomas relacionados con la enfermedad.
- **Hospitalizado:** Una cierta fracción de los individuos son hospitalizados. La probabilidad de transitar a este estado está relacionado con la edad del individuo, y a mayor edad, más posibilidades.
- **Estados de vacunado:** Estos sub-estados representan las etapas de aquellos individuos vacunados. Éstos pueden ser susceptibles de contagiarse e infectarse, pero tienen parámetros distintos al estado general dado que la capacidad inmunológica es diferente.
- **Recuperado:** Individuo que ha adquirido la inmunidad.
- **Muerto:** Los individuos que transitan a este estado, también relacionado con la edad, se eliminan de la simulación.

B. Modelo social

Como ha sido comentado, *EpiGraph* es un simulador basado en agentes que maneja individuos con sus respectivos atributos, afiliaciones e interacciones. Consideramos cuatro grupos principales: estudiantes, trabajadores, personas que permanecen en casa, y personas de tercera edad. Para cada grupo, consideramos tres diferentes distribuciones temporales para las actividades de los individuos: aquellas relacionadas con los días de la semana, los sábados, y los festivos (que incluyen los domingos). Dos grafos diferentes de interacciones sociales son utilizados para generar los patrones de contacto entre individuos, además de matrices de contacto extraídas de encuestas públicas que se utilizan para proporcionar información estadística sobre dichos contactos.

Para incrementar el realismo de la mezcla de individuos en la población simulada, hemos dividido el grupo de trabajadores en diferentes profesiones, lo que mejora la capacidad de simular las interacciones entre la población. De la misma forma, el grupo de tercera edad se ha dividido en varios subgrupos para distinguir entre personas que son atendidas en centros especializados, centros de día o en sus propias casas.

C. Modelo de vacunación

En el modelo compartimentado descrito anteriormente, un individuo vacunado que es infectado transita bien a un estado asintomático o, en caso de que la vacuna no haya tenido efecto, a infectado (vacunado), con una probabilidad de riesgo de hospitalización o muerte mayor que aquellos individuos infectados y no vacunados. La eficacia de la vacunación es modelada como la probabilidad de transitar al estado asintomático. Por ejemplo, un individuo vacunado con una vacuna eficaz al 95 % significa que, si es infectado, tiene un 95 % de probabilidad de transitar al estado asintomático. Nótese que *EpiGraph* cuenta con modelos de simulación de vacunas Pfizer-BioNTech, Moderna, Astra-Zeneca y Janssen, además de las estrategias de vacunación establecidas, que definen aspectos como la prioridad de vacunación de los diferentes grupos o el número de dosis.

D. Intervenciones no-farmacológicas - NPIs

El riesgo de infección de un individuo dado depende, además, de dos factores que reducen el riesgo de transmisión de las enfermedades: la vacunación de un individuo susceptible que está en contacto con un infectado, y el uso de intervenciones no-farmacológicas. Esta categoría incluye diferentes fuentes de intervenciones, entre las que merece la pena mencionar: el uso de mascarilla, el distanciamiento social, y el uso de tests para muestreo de la población.

E. Modelo de transporte

El modelo de transporte refleja el movimiento de personas entre ciudades por trabajo, estudios o vacaciones, y está basado en el modelo propuesto por Viboud et al. [16]. Nótese que el movimiento de personas en el interior de una ciudad ya es capturado por el modelo social. Por lo tanto, el modelo de transporte tiene como objetivo mover individuos entre diferentes ciudades, permitiendo la transmisión de las enfermedades en grandes áreas. La información geográfica que maneja *EpiGraph* incluye latitud, longitud, y la distancia entre zonas urbanas, y es obtenida de Google Maps mediante la API Google Distance Matrix [17].

V. Evaluación

Esta sección se enfoca en el entorno de ejecución de las aplicaciones, que incluye el mencionado `LIMITLESS` y el *DVFS* (Dynamic Voltage Frequency Scaling), una técnica para el manejo del consumo energético de la CPU de los nodos. Ejecutar aplicaciones con niveles bajos de frecuencia/voltaje reduce el consumo de energía, pero esto tiene un impacto negativo en el rendimiento de las aplicaciones debido a la reducción en la velocidad de la CPU. Nuestro objetivo es reducir el consumo de energía mientras se mantiene el rendimiento esperado, lo que implica no reducir el DFVS en fases intensivas en CPU. Para hacer esto, diseñamos políticas de consumo energético que utilizan los datos recolectados por `LIMITLESS` para fijar el DVFS en tiempo de ejecución de acuerdo con la fase que esté ejecutando la aplicación.

El DVFS ha sido utilizado de forma extensa para reducir la energía consumida. Aplicaciones como *EpiGraph* combinan fases de CPU, de comunicación y de entrada/salida, lo que implica que el valor óptimo de DVFS debe ser diferente en cada una de esas fases. Además, reducir la frecuencia no necesariamente reduce el consumo energético debido a un incremento del tiempo de ejecución (posiblemente linear), lo que implica consumir energía (quizá menos) pero durante más tiempo. Para poder hacer frente a estos potenciales problemas, necesitamos monitorizar de forma continua la plataforma y las aplicaciones y generar perfiles de las mismas para poder identificar la mejor configuración.

Durante las simulaciones de *EpiGraph*, `LIMITLESS` monitoriza cada ejecución con diferentes valores de DVFS para generar un perfil de la aplicación. El DVFS es configurado en tiempo de ejecución dependiendo de la información recolectada.

La motivación para generar perfiles de aplicaciones es poder identificar si es posible establecer diferentes valores de DVFS en tiempo de ejecución para reducir el consumo manteniendo el máximo rendimiento esperado. Mediante la ejecución de un escenario de propagación de una infección con diferentes valores de DVFS, hemos encontrado que la máxima frecuencia no obtiene el mejor rendimiento a nivel global. Esto corrobora que el valor máximo de DVFS no siempre es la mejor configuración, pues depende de la aplicación. Además, mediante experimentación hemos comprobado que, conociendo el comportamiento de la aplicación, es posible reducir el consumo de energía sin penalizar el tiempo de ejecución.

La evaluación que se muestra a continuación se ha realizado en una plataforma física que consta de ocho nodos de cómputo. Una partición del clúster contiene seis nodos con Intel(R) Xeon(R) E7 con 12 núcleos y 128 GB de RAM en el otro. La segunda partición contiene dos nodos con CPU Intel(R) Xeon(R) Gold 6212U a 2,40 GHz con 24 núcleos y 315 GB de RAM. La conexión entre nodos es Ethernet de 10 Gbps. La E/S se basa en el sistema de archivos paralelos Gluster.

A. Application profiling

La figura 3 muestra la metodología propuesta para evaluar el rendimiento de las aplicaciones con diferentes valores de frecuencia. La primera etapa consiste en la caracterización de la aplicación mediante la recolección de las métricas de rendimiento durante su ejecución. La segunda etapa consiste en almacenar las métricas de rendimiento asociadas a cada test en la base de datos, lo que permite al componente de análisis generar los perfiles. El tercer y el último paso consisten en modificar los valores del DVFS de forma dinámica durante la ejecución de la aplicación para consumir menos energía mientras se mantiene el tiempo de ejecución.

La figura 4 muestra los resultados de la simulación de una ola de infección de la variante Omicron de COVID-19 en Madrid (España), durante 2022. Se puede apreciar una ola de infección desde el día 700 hasta el 820 aproximadamente. Esta figura está estrechamente relacionada con la figura 5, que muestra el tiempo de cómputo por día de simulación. Se puede apreciar cómo la ola de infección (debido al número de agentes infectados) tiene un impacto en el rendimiento, incrementando el tiempo por iteración.

La figura 6 muestra el patrón de E/S que tiene la ejecución de la simulación. Como la principal causa de E/S es el checkpointing, y éste se realiza cada 6 horas, se puede ver que los picos son periódicos, excepto durante la ola de infección, donde hay muchos casos de infección activos).

Las figuras 7 8 y 9 muestran los perfiles de uso de CPU, memoria y consumo de energía durante la simulación. Se ejecutan 16 procesos por nodo, consumiendo prácticamente toda la memoria y manteniendo una utilización de la CPU en torno al 80 %. Inicialmente, el uso de red es mayor debido a la sincronización entre procesos, y posteriormente se esta-

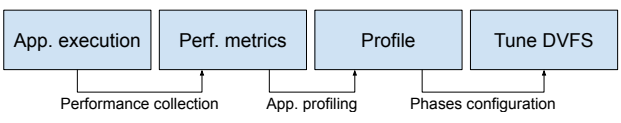


Fig. 3: Metodología para crear perfiles de las aplicaciones basados en los datos de monitorización.

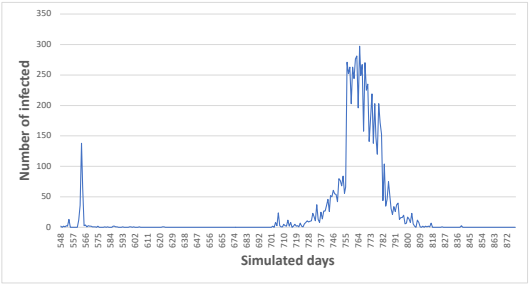


Fig. 4: Número de personas infectadas por día simulado. Esta ola de infección se corresponde con el caso de uso de EpiGraph para el COVID-19 en Madrid.

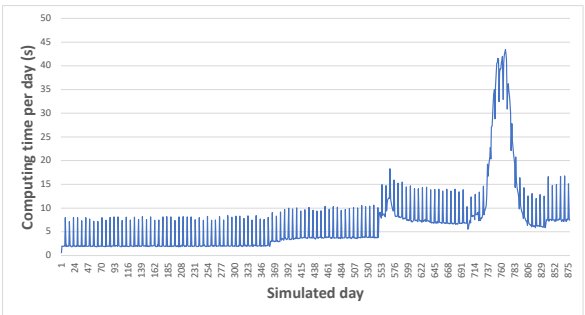


Fig. 5: Tiempo de iteración por día simulado (agregado). Cada día simulado consiste en 1440 iteraciones. Dado que el tiempo por iteración depende del número de individuos infectados, del checkpointing, y de las comunicaciones entre procesos, cada día simulado puede tener un tiempo de iteración diferente.

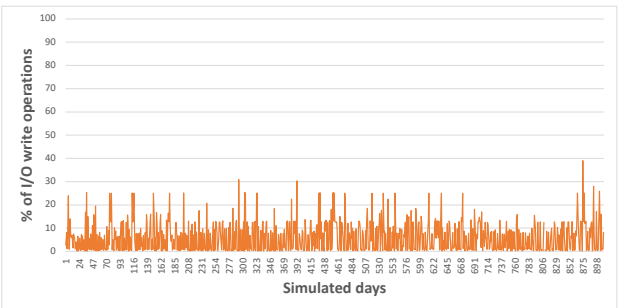


Fig. 6: Patrón de E/S durante el caso de uso de Madrid. La mayoría de las operaciones de E/S se corresponden con operaciones de checkpointing.

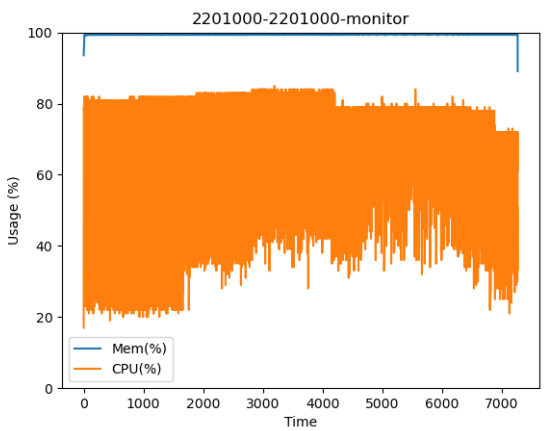


Fig. 7: EpiGraph caso base - Uso de CPU y memoria con un DVFS fijado a frecuencia 2.2GHz.

biliza, decayendo un poco durante la ola de infección. Finalmente, la energía está directamente relacionada con el valor de DVFS seleccionado, que por defecto es 2.2GHz.

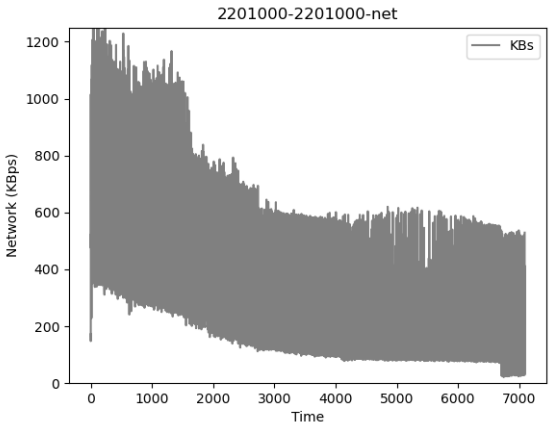


Fig. 8: EpiGraph caso base - Uso de red con un DVFS fijado a frecuencia 2.2GHz.

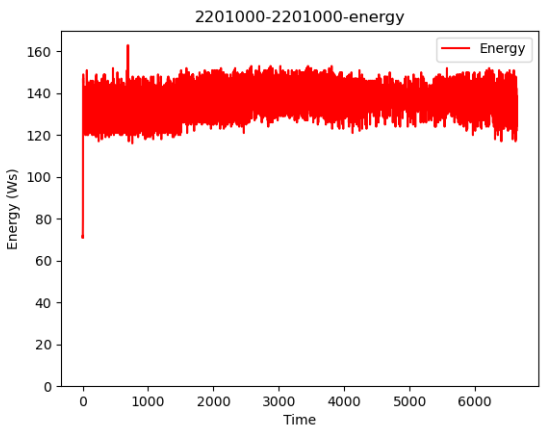


Fig. 9: EpiGraph caso base - Energía consumida con un DVFS fijado a frecuencia 2.2GHz.

B. Resultados

En vista de los resultados del profiling ofrecido por el monitor, podemos ver que el rendimiento de la memoria principal apenas se ve afectada (incluso puede no tener degradación) con bajas frecuencias de reloj, manteniéndose siempre en torno al 100 % en la figura 7. Por ello, se propone una serie de configuraciones de voltaje/frecuencia diferentes para las dos fases identificadas durante la simulación: (1) ola de infección y (2) el resto de la simulación. Estas configuraciones se pueden ver, junto a los resultados obtenidos en la tabla I y la figura 10.

El resultado principal de estos experimentos es que el caso base, aquel que utiliza la configuración de DVFS por defecto, no proporciona la mejor relación entre el tiempo de ejecución y consumo de energía. Ejecutar *EpiGraph* con una frecuencia de 2.2GHz (lo máximo permitido en las máquinas de prueba) requiere un total de 7540 segundos, lo que consume 874805W. Dado que la velocidad de la CPU se ve directamente afectada por el DVFS establecido y la memoria no [18], el checkpointing parece ser el único culpable de la degradación del rendimiento con valores de DVFS altos.

Las figuras 11, 12 y 13 muestran las métricas de rendimiento relacionadas con el test con ID 20 en la

ID	Configuración	T. ejecución (s)	Energía (Ws)
1	1000000-1000000	12029	1055995
2	1000000-1300000	10978	985835
3	1000000-1600000	10254	937910
4	1000000-1900000	9750	907560
5	1000000-2201000	8865	886050
6	1300000-1000000	11107	997202
7	1300000-1300000	9963	915690
8	1300000-1600000	9262	868980
9	1300000-1900000	8758	839613
10	1300000-2201000	7916	821510
11	1600000-1000000	10491	956779
12	1600000-1300000	9383	882455
13	1600000-1600000	8648	833345
14	1600000-1900000	8153	805137
15	1600000-2201000	7336	789528
16	1900000-1000000	10074	939404
17	1900000-1300000	8955	860864
18	1900000-1600000	8313	819558
19	1900000-1900000	7795	788504
20	**1900000-2201000**	**6905**	**766716**
21	2201000-1000000	9440	959051
22	2201000-1300000	8262	877059
23	2201000-1600000	7554	830169
24	2201000-1900000	6973	791633
25	2201000-2201000	7540	874805

Tabla I: Combinación de frecuencia/voltaje para cada una de las fases identificadas en la simulación(pre/post infección, y ola de infección respectivamente). Estos experimentos se han ejecutado tres veces cada uno para evitar discrepancias puntuales.

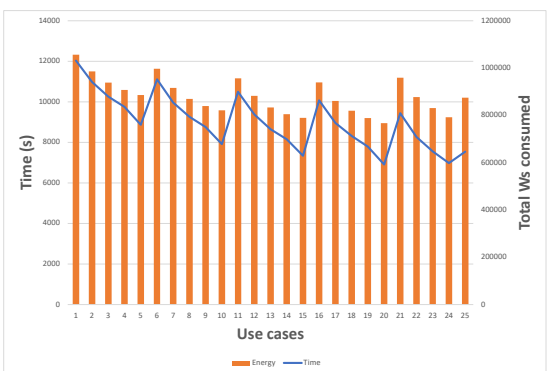


Fig. 10: Combinaciones DVFS - Esta figura muestra, para cada combinación frecuencia/voltaje identificada en la Tabla I, el consumo de energía (barras naranjas) y el tiempo de ejecución (línea azul).

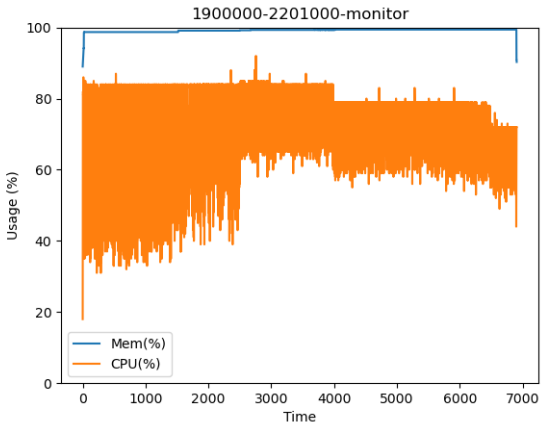


Fig. 11: EpiGraph caso de uso 20 - Utilización de CPU y memoria usando DVFS con una frecuencia de 1.9GHz, excepto entre los días simulados 700 y 836 en que se configuró para 2.2GHz.

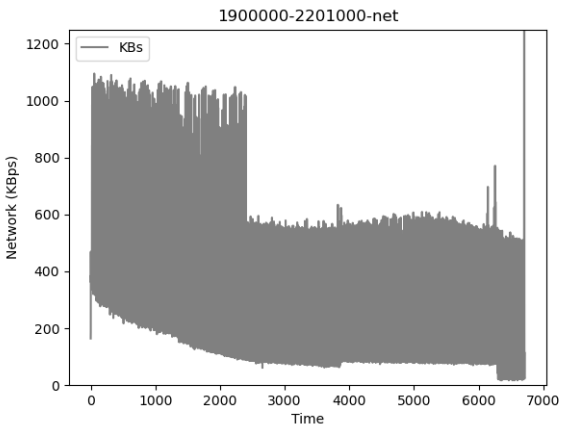


Fig. 12: EpiGraph caso de uso 20 - Utilización de la red usando DVFS con una frecuencia de 1.9GHz, excepto entre los días simulados 700 y 836 en que se configuró para 2.2GHz.

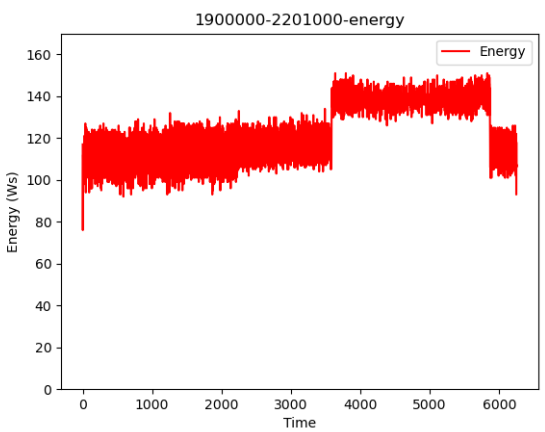


Fig. 13: EpiGraph caso de uso 20 - Consumo de energía usando DVFS con una frecuencia de 1.9GHz, excepto entre los días simulados 700 y 836 en que se configuró para 2.2GHz.

tabla I, que es el que tiene un menor tiempo de ejecución y menor consumo energético. La frecuencia fijada por defecto es de 1.9GHz, mientras que durante la ola de infecciones se utiliza la frecuencia máxima de 2.2GHz, ya que es cuando más uso de CPU se hace y a la vez se reduce la E/S. Esta configuración ejecuta las simulaciones en 6905 segundos con consumos de energía de 766716W, lo que significa una reducción del tiempo de ejecución de un 8,4 %, y una reducción del consumo energético del 12,3 %. Se puede ver que el cambio más significativo se produce en el uso de CPU en la figura 11, donde la CPU es más estable durante la ola de infección con mínimos un 20 % superiores al caso base. Razonamiento que también puede ser aplicado a la figura 12, donde el ancho de banda alcanza valores más altos al inicio de la ejecución. Finalmente, como se puede apreciar en la figura 13, el ajuste dinámico del DVFS basado en modelos permite que el sistema reduzca el consumo energético en aquellas fases en las que no es necesaria la máxi-

ma potencia de la máquina, y manteniendo el mismo consumo en aquellas fases donde sí es necesario. Esto hace que, en la mayoría de casos, la energía consumida por las aplicaciones sea menor que en los casos que no aplican la optimización propuesta. En los casos en los que esta propuesta no se puede aplicar, por ejemplo si se utiliza una aplicación que requiere continuamente la máxima frecuencia, los resultados de consumo energético y tiempo de ejecución son similares (con diferencias despreciables).

VI. Conclusion

En este artículo se presenta una nueva integración en `LIMITLESS` como entorno de monitorización y planificación de aplicaciones en entornos HPC. Aprovechar toda la información recopilada, tanto del sistema como de las aplicaciones, es un reto. Para ello, se ha implementado una política de configuración dinámica del DVFS de los nodos, basando la toma de decisiones en los modelos de rendimiento de las aplicaciones que hay en ejecución. El objetivo de esta integración es mejorar la eficiencia energética de los sistemas HPC, mediante la reconfiguración en tiempo de ejecución del DVFS, para minimizar la energía consumida por componentes como la CPU, la memoria y los sistemas de refrigeración (debido a la menor necesidad de disipación de calor), mientras se mantiene el rendimiento esperado de la aplicación en ejecución. Los resultados principales que se han obtenido muestran que mantener un valor estático para el DVFS no es práctico en términos de energía, y dependiendo del perfil de la aplicación, tampoco en términos de rendimiento o tiempo de ejecución. Hemos verificado que en aplicaciones, o fases de aplicaciones, intensivas en CPU, los mejores resultados en tiempo de ejecución se obtienen con el DVFS en su valor máximo, ya que lo que interesa es realizar el mayor número de operaciones por instante de tiempo. Sin embargo, fases de comunicación o E/S no requieren que el HW esté al máximo, ya que son componentes más lentos (en comparación con la CPU), y por lo tanto una reducción de la frecuencia y del voltaje los afecta mínimamente, incluso puede no perjudicarlos. Las pruebas realizadas siguiendo estos resultados han ofrecido mejoras en el consumo energético de en torno a un 12,3 % a la par que se ha reducido el tiempo de ejecución en un 8,4 % con respecto a la configuración por defecto del DVFS.

Agradecimientos

El presente trabajo ha sido parcialmente financiado por European High-Performance Computing Joint Undertaking (JU) mediante el proyecto .Adaptive multi-tier intelligent data manager for Exascaleçon identificación No 956748 - ADMIRE - H2020-JTI - EuroHPC-2019-1 y la Agencia Española de Investigación con identificación PCI2021-121966.

Referencias

[1] Robert Hinch, William JM Probert, Anel Nurtay, Michelle Kendall, Chris Wymatt, Matthew Hall, Katrina Lythgoe, Ana Bulas Cruz, Lele Zhao, Andrea Stewart, et al., "Openabm-COVID19-an agent-based model for non-pharmaceutical interventions against COVID-19 including contact tracing," *medRxiv*, 2020.

[2] Erik Cuevas, "An agent-based model to evaluate the COVID-19 transmission risks in facilities," *Computers in Biology and Medicine*, vol. 121, pp. 103827, 2020.

[3] Miguel Guzmán Merino, Christian Duran Gonzalez, María Cristina Marinescu, Concepción Delgado Sanz, Diana Gómez Barroso, Jesús Carretero Pérez, and David Expósito Singh, "Data management in epigraph COVID-19 epidemic simulator," *Workshop on 27th International European Conference on Parallel and Distributed Computing (Europar 2021).*, 2021.

[4] Parham Haririan, "Dvfs and its architectural simulation models for improving energy efficiency of complex embedded systems in early design phase," *Computers*, vol. 9, no. 1, 2020.

[5] Sparsh Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *International Journal of Computer Aided Engineering and Technology*, vol. 6, no. 4, pp. 440–459, 2014.

[6] Jason Mair, Zhiyi Huang, David Eyers, and Yawen Chen, "Quantifying the energy efficiency challenges of achieving exascale computing," *Proceedings - 2015 IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2015*, pp. 943–950, jul 2015.

[7] "Top500 list," 2023.

[8] "Green500 list," 2023.

[9] Bilge Acun, Kavitha Chandrasekar, and Laxmikant V. Kale, "Fine-Grained Energy Efficiency Using Per-Core DVFS with an Adaptive Runtime System," *2019 10th International Green and Sustainable Computing Conference, IGSC 2019*, oct 2019.

[10] Pawel Bratek, Lukasz Szustak, Roman Wyrzykowski, Tomasz Olas, and Tomasz Chmiel, "Heterogeneous voltage frequency scaling of data-parallel applications for energy saving on homogeneous multicore platforms," in *Euro-Par 2021: Parallel Processing Workshops: Euro-Par 2021 International Workshops, Lisbon, Portugal, August 30-31, 2021, Revised Selected Papers*, Berlin, Heidelberg, 2021, p. 141–153, Springer-Verlag.

[11] Manjari Gupta, Lava Bhargava, and Indu Sreedevi, "Dynamic voltage frequency scaling in multi-core systems using adaptive regression model," *Proceedings of the 4th International Conference on IoT in Social, Mobile, Analytics and Cloud, ISMAC 2020*, pp. 1201–1206, oct 2020.

[12] Péter Piros, Tamás Ferenci, Rita Fleiner, Péter Andréka, Hamido Fujita, László Főző, Levente Kovács, and András Jánosi, "Comparing machine learning and regression models for mortality prediction based on the hungarian myocardial infarction registry," *Knowledge-Based Systems*, vol. 179, pp. 1–7, 2019.

[13] Mahya Seyedan and Fereshteh Mafakheri, "Predictive big data analytics for supply chain demand forecasting: methods, applications, and research opportunities," *Journal of Big Data*, vol. 7, no. 1, pp. 53, 2020.

[14] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu, "The impact of gpu dvfs on the energy and performance of deep learning: An empirical study," in *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, New York, NY, USA, 2019, e-Energy '19, p. 315–325, Association for Computing Machinery.

[15] Alberto Cascajo, David E. Singh, and Jesús Carretero, "Limitless - light-weight monitoring tool for large scale systems," in *2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2021, pp. 220–227.

[16] Cécile Viboud, Ottar N Bjørnstad, David L Smith, Lone Simonsen, Mark A Miller, and Bryan T Grenfell, "Synchrony, waves, and spatial hierarchies in the spread of influenza," *Science*, vol. 312, no. 5772, pp. 447–451, 2006.

[17] Google LLC, https://developers.google.com/maps/, *Google Maps API*, 2021.

[18] Wen-Yew Liang, Shih-Chang Chen, Yang-Lang Chang, and Jyh-Perng Fang, "Memory-aware dynamic voltage and frequency prediction for portable devices," in *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA '08. 14th IEEE International Conference on*, 09 2008, pp. 229 – 236.

Detectando metilación alelo-específica en clústeres multinúcleo

Alejandro Fernández Fraga[1], Jorge González-Domínguez[1] y María J. Martín[1]

Resumen— **El descubrimiento de la Metilación Alelo-Específica (ASM, Allele-Specific Methylation) es un campo de investigación en biología de gran importancia ya que regula la impronta genómica, que ha sido identificada como la causa de algunas enfermedades genéticas. Sin embargo, el alto coste computacional de las herramientas bioinformáticas desarrolladas para este propósito impide su aplicación a conjuntos de datos de gran escala. Por lo tanto, se requieren herramientas mucho más rápidas para avanzar en este campo de investigación. En este trabajo presentamos *PARamrfinder*, una herramienta paralela que aplica un modelo estadístico para identificar ASM en datos de secuenciación por bisulfito (BS-Seq, Bisulfite Sequencing). Está basada en la herramienta secuencial *amrfinder*, que es capaz de detectar ASM a nivel de región a partir de experimentos de BS-Seq en ausencia de información de Polimorfismos de Nucleótido Único (SNP, Single Nucleotide Polymorphism). *PARamrfinder* proporciona las mismas regiones alelicamente metiladas (AMRs, Allelically Methylated Regions) que *amrfinder*, pero con un tiempo de ejecución significativamente menor gracias a la explotación de las capacidades de cómputo de los clústeres de CPU multinúcleo y las operaciones RMA de MPI para lograr un equilibrado dinámico de la carga de trabajo. Como ejemplo, nuestra herramienta es hasta 567 veces más rápida para experimentos de datos reales en un clúster con ocho nodos, cada uno con dos procesadores de 16 núcleos. El código fuente de *PARamrfinder*, así como un manual de referencia, está disponible en https://github.com/UDC-GAC/PARamrfinder.**

Palabras clave— **Metilación alelo-específica, Computación de altas prestaciones, MPI, OpenMP, RMA, Balanceo dinámico de la carga.**

I. Introducción

La metilación es un proceso epigenético que modifica el ADN al agregar un grupo metilo a un nucleótido del ADN. Este proceso es clave para entender varias funciones biológicas de forma que niveles anormales de metilación pueden indicar la presencia de ciertas enfermedades. Una de esas funciones biológicas es la impronta genómica, un proceso en el que solo se expresa una copia de un gen en un individuo, mientras que la otra copia se suprime, y que se ha determinado como causa de varias enfermedades genéticas, como los síndromes Prader-Willi [1], Angelman [2] o Beckwith-Wiedemann [3].

La expresión de genes improntados está regulada por la metilación alelo-específica (ASM, Allele-Specific Methylation), un tipo particular de metilación que ocurre cuando los patrones de metilación del ADN son asimétricos entre alelos. Debido a esto la detección de regiones alélicamente metiladas (AMRs, Allelically Methylated Regions) ha ganado atención en los últimos años, ya que proporciona información biológica interesante.

amrfinder [4] es una herramienta para detectar ASM en experimentos de secuenciación por bisulfito (BS-Seq, Bisulfite Sequencing), que se ha usado en varios estudios biologicos [5], [6], [7]. El problema principal de esta herramienta es que requiere un tiempo de ejecución elevado para procesar conjuntos de datos grandes, ya que los análisis que lleva a cabo requieren un alto coste computacional.

En este articulo presentamos *PARamrfinder*, una herramienta que acelera la identificación de AMRs en clústeres multicore modernos, utilizando una aproximación híbrida eficiente que combina procesos MPI e hilos OpenMP. *PARamrfinder* obtiene los mismos resultados biológicos altamente precisos que obtiene *amrfinder*, pero en un tiempo mucho menor.

El resto del artículo se organiza de la siguiente manera. La Sección II presenta el estado del arte y el trabajo relacionado. La Sección III introduce algunos conceptos básicos sobre la herramienta original *amrfinder* que son necesarios para entender el objetivo de este trabajo y la implementación de nuestro método. La Sección IV describe la implementación paralela de *PARamrfinder*. La Sección V proporciona la evaluación experimental en términos de tiempo de ejecución y escalabilidad. Finalmente, las conclusiones se presentan en la Sección VI.

II. Trabajo relacionado

En los últimos años se ha puesto un gran interés en el desarrollo de herramientas para detectar ASM en conjutos de datos obtenidos de distintos tipos de tecnologías biológicas, como microarrays que genotipan el ADN convertido por bisulfito [8], tecnologías de captura de baja resolución como secuenciación por dominio de unión a ADN metilado (MBD, Methyl-Binding Domain) [9], secuenciación por inmunidad del ADN metilado (MeDIP, methylated DNA immunoprecipitation) [10] o secuenciación por bisulfito (BS-Seq, Bisulfite Sequencing) [11].

Esta última es la tecnología más utilizada en la actualidad y permite detectar ASM a nivel de nucleótido, aunque la mayoría de las herramientas que se han desarrollado para detectar ASM en datos de BS-Seq lo hacen asociándolos con SNPs heterogéneos. Algunos ejemplos son Bis-SNP [12], Allelonome.PRO [13] o CGmapTools [14]. Estas herramientas que dependen de información genotípica presentan una limitación importante: están ciegas ante ciertas porciones de ASM, ya que la metilación impronta no esta necesariamente asociada a variaciones genotípicas. *amrfinder* [4] supera esta limitación, ya que su modelo es independiente del genotipo, y por lo tanto es am-

[1]CITIC, Grupo de Arquitectura de Computadores, Universidade da Coruña, e-mail: {a.fernandez3,jgonzalezd,mariam}@udc.es

pliamente aplicable para la identificación de ASM en el contexto de la impronta.

Hasta donde nosotros sabemos, no existe ningún trabajo previo centrado en acelerar la identificación de ASM con técnicas de Computación de Altas Prestaciones (HPC). Sin embargo, podemos encontrar en la literatura otras herramientas bioinformáticas que son capaces de explotar eficientemente las capacidades computacionales de clústeres multinúcleo. Algunos ejemplos son *pIQPNNI* [15], que utiliza MPI para inferir árboles filogenéticos de máxima verosimilitud a partir de datos de ADN o proteínas con un gran número de secuencias; *MPIGeneNet* [16], que incluye rutinas MPI y directivas OpenMP para construir redes genéticas; o *ClustalW-MPI* [17], que aplica MPI para reducir el tiempo de ejecución de la alineación de múltiples secuencias de proteínas o nucleótidos. Las herramientas anteriores han sido utilizadas por biólogos para completar experimentos sobre grandes conjuntos de datos en un tiempo razonable, demostrando que una herramienta como *PARamrfinder* puede ser atractiva para la comunidad científica.

III. AMRFINDER

amrfinder [4] es una herramienta disponible públicamente como parte del pipeline *MethPipe*[1] para identificar ASM en mamíferos, en los que la metilación ocurre principalmente en sitios CpG: fragmentos de ADN donde una nucleótido citosina (C) está seguido de una guanina (G). La herramienta identifica estas regiones en conjuntos de datos de BS-Seq, un método por el cual las citosinas no metiladas son convertidas a timinas (T) y las metiladas permanecen como citosinas (C). La herramienta aplica dos modelos, un modelo para un solo alelo que asume que el conjunto de lecturas mapeadas a un intervalo representan un único patrón de metilación, y un modelo de alelo específico que asume que estas lecturas representan dos patrones de metilación distintos, ambos constituidos por aproximadamente la misma proporción de lecturas (ya que los alelos están presentes en proporciones iguales). Los parámetros de ambos modelos se computan iterativamente para ajustarse a los datos, y se compara su verosimilitud para determinar la presencia de ASM.

amrfinder es una herramienta de línea de comandos escrita en C++ que admite varios parámetros de configuración (por ejemplo, el número máximo de iteraciones utilizadas para ajustar los modelos) y obtiene los datos biológicos de archivos de texto. En concreto, la entrada es un genoma de referencia contenido en una serie de archivos *FASTA*, a los que se deben alinear los resultados, y un archivo *epiread*, que contiene las lecturas de entrada en un formato comprimido, indicando solo el estado de metilación para cada sitio CpG.

La Figura 1 muestra un ejemplo de un archivo *epiread*, donde cada fila está dedicada a una lectura, que consiste en tres columnas: la primera columna es el cromosoma de la lectura, la segunda es el orden de numeración del primer CpG en la lectura, y la última es la secuencia de CpG de la lectura.

[1]http://smithlabresearch.org/software/methpipe/

Nombre del cromosoma	Posición del sitio CpG	Secuencia de solo sitios CpG
chr10	62	T N N
chr10	62	N T N T
chr10	62	N T T T
chr10	67	N T N N T N
chr10	68	N T C C N T N
chr10	69	N N T N T T N

Fig. 1: Ejemplo de fichero *epiread*

```
1  procedure AMRsIdentification()
2    for each chrom in epireads_file
3      chrom_epireads := ReadEpireads(chrom,
           epireads_file)
4      for each window in chrom
5        window_epireads := GetCurrentEpireads(
             chrom_epireads, window)
6        if ContainsMinInformation(window_epireads)
7          single_allele_model :=
               TestSingleAlleleModel(window_
               epireads)
8          specific_allele_model :=
               TestSpecificAlleleModel(window_
               epireads)
9          is_allelicaly_methylated :=
               EvaluateModel(single_allele_model,
               specific_allele_model)
10         if is_allelicaly_methylated
11           AddAMR(window, specific_allele_model)
12         end if
13       end if
14     end for
15   end for
16 end procedure
```

Fig. 2: Algoritmo de la fase de identificación de AMRs de *amrfinder*

La herramienta ejecuta varios pasos sobre los datos para transformarlos en las AMRs finales. De ellos, el paso principal es la identificación de las AMRs, ya que constituye el cuello de botella principal de la herramienta. La Figura 2 detalla cómo funciona este paso. Los cromosomas se leen en memoria uno a uno (Línea 2) y, para cada uno, se recorren todas las posiciones de los sitios CpG utilizando una ventana deslizante de ancho fijo (un número fijo de sitios CpG) (Línea 4). La Figura 3 muestra el comportamiento de esta ventana, que se mueve una posición cada vez. Para cada posición, se seleccionan las lecturas limitadas por la ventana actual (Línea 5). Si esa ventana no contiene un número mínimo de lecturas, se descartan esas lecturas (Línea 6). De lo contrario, se procesan utilizando los modelos estadísticos (Líneas 7-9). En caso de que se detecte la presencia de ASM, se añade la región a la lista de AMRs (Líneas 10-11).

Tras este paso, se obtiene una lista de AMRs, a la que se aplicarán varios pasos de post-procesado para obtener las AMRs finales, entre los que destaca el alineamiento de estas al genoma de referencia.

IV. PARAMRFINDER

PARamrfinder es una herramienta paralela que proporciona exactamente los mismos resultados que *amrfinder* (y por lo tanto su alta precisión) pero en un tiempo de ejecución significativamente menor gracias al empleo de MPI y OpenMP para explotar las

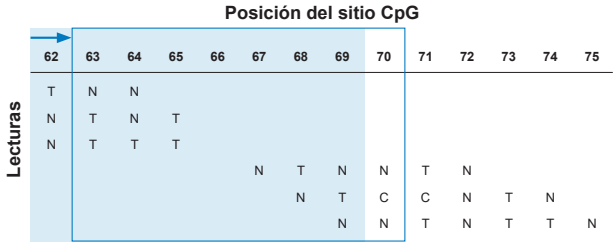


Fig. 3: Ejemplo del comportamiento de la ventana deslizante utilizando un tamaño de 8. Sombreada, la ventana inicialmente en la posición 62. Bordeada, la ventana desplazándose una posición para analizar el siguiente intervalo.

capacidades computacionales de los clústeres multinúcleo. Además, *PARamrfinder* mantiene el mismo mecanismo de configuración que *amrfinder* para simplificar su adopción por aquellos biólogos que ya están familiarizados con la herramienta original. Más información sobre este procedimiento de configuración se puede encontrar en el manual de referencia que está disponible en el repositorio público de *PARamrfinder* [2].

A. *Optimización secuencial de los modelos estadísticos*

Antes de comenzar la implementación paralela, se realizó un análisis del código para asegurarse de que la herramienta original era una base adecuada y efectiva para la nueva herramienta paralela. Sin embargo, este análisis reveló que la implementación original para el ajuste de los modelos estadísticos era ineficiente. En particular, los resultados de algunas operaciones que se ejecutaban a lo largo de esta etapa de ajuste (principalmente el cálculo de logaritmos, una operación particularmente costosa) se estaban descartando y volviendo a calcular varias veces. Para evitar este comportamiento ineficiente, se implementó una nueva técnica, que hemos denominado *ComputeAndStore*, que consiste en realizar todos los cálculos la primera vez que se requieren y almacenar los resultados en un búfer. De esta forma, si se vuelven a necesitar los resultados de estos cálculos, un acceso simple y rápido al búfer es suficiente para obtenerlos.

B. *Problemas de balanceo de carga*

Además del ineficiente método de ajuste, existen varios problemas de balanceo de carga que hacen que una implementación paralela *naive* no sea adecuada para *amrfinder*. Estos problemas hacen que sea un reto conseguir una distribución de datos y carga de trabajo eficiente. El motivo es que diferentes regiones pueden necesitar tiempos de ejecución extremadamente diferentes para su análisis.

En primer lugar, la mayoría de las regiones no tienen suficiente información para que el análisis tenga sentido. Por lo tanto, muchas de ellas serán descartadas sin siquiera tener que ejecutar los modelos computacionalmente costosos (Figura 2, Línea 6). Esto divide las regiones en dos grupos muy distintos. Por un lado, las regiones que no se analizarán y que tienen un tiempo de ejecución asociado cercano a cero, y por otro lado, las regiones que se analizarán y que necesitan un tiempo de ejecución relativamente alto.

Segundo, incluso dentro de las regiones que se analizarán, también existe un gran desequilibrio en la carga de trabajo asociada a cada una de ellas. Este desequilibrio se debe a dos factores:

- La cantidad de información en la región. En cuanto a este factor, no todas las regiones tienen la misma cantidad de lecturas asociadas a ellas. Además, cada lectura no necesita contener información sobre cada sitio CpG en la región. Es decir, la cantidad de información en la región no es una constante, sino una variable que depende del número de lecturas y el número de CpGs representados en ellas. Existe una relación directa entre el número total de lecturas correspondientes a cada uno de los CpGs asociados a una región (cantidad de información) y el tiempo de ejecución de esa región. En consecuencia, el tiempo de ejecución de una región se puede predecir de antemano en función de la cantidad de información que contiene.
- El número de iteraciones que se necesitan para que los modelos estadísticos converjan. Este factor también sigue una relación directa con el tiempo de ejecución de una región. Sin embargo, es impredecible y permanece desconocido hasta que el proceso de ajuste del modelo se completa. Por lo tanto, no es posible estimar el tiempo de ejecución de una región basándose en el número de iteraciones ya que esta información no se conoce de antemano.

Es más, es común que algunos grupos pequeños de regiones contengan una gran cantidad de información (hasta cinco órdenes de magnitud más grande que la mediana). Hemos denominado a estas *regiones elefante*, y sus análisis representan un porcentaje relevante del tiempo de ejecución total del programa. Su gestión es clave en el rendimiento de la implementación paralela ya que pueden generar fácilmente un gran desequilibrio en la carga de trabajo y convertirse en el cuello de botella de la ejecución.

C. *Implementación paralela de la fase de identificación*

PARamrfinder incluye dos niveles de paralelismo para aprovechar las capacidades computacionales de los actuales clústeres multinúcleo cuando se acelera la identificación de AMRs. En primer lugar, se utilizan rutinas MPI para distribuir las regiones entre los diferentes nodos. En segundo lugar, cada proceso MPI crea varios hilos OpenMP que se ejecutan en diferentes núcleos del nodo. Las regiones asignadas a cada proceso se distribuyen entre los hilos OpenMP. Como diferentes regiones pueden tener distintas cargas computacionales, se utiliza una política de planificación dinámica para garantizar un buen balanceo

[2] `https://github.com/UDC-GAC/PARamrfinder`

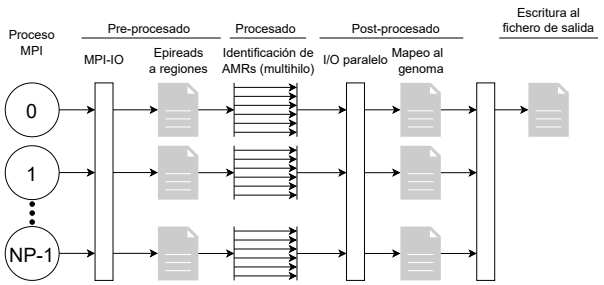


Fig. 4: Flujo de trabajo de *PARamrfinder*

de la carga entre los hilos. La Figura 4 proporciona una visión general gráfica de la implementación paralela de dos niveles aplicada en *PARamrfinder*.

No obstante, debido a los problemas discutidos en la Sección IV-B, una distribución eficiente de la carga de trabajo a nivel de proceso no es trivial. La herramiento original *amrfinder* trabaja cromosoma a cromosoma, es decir, que solo un cromosoma se mantiene en memoria a la vez. Aunque esta es la mejor opción para la herramienta original, no lo es para *PARamrfinder*, ya que necesita obtener información sobre todas las regiones de antemano para hacer una buena distribución de la carga de trabajo. Entonces, la herramienta paralela comienza con una fase de pre-procesamiento para conocer el número total de regiones a analizar, así como la cantidad de información que contienen cada una de ellas. Este pre-procesamiento fue diseñado teniendo como prioridad no introducir un sobrecoste significativo en la ejecución. Para lograr esto, se aplica una especie de *loop fission* al bucle principal del algoritmo que se muestra en la Figura 2 (Línea 2), dividiéndolo en dos partes:

- Un bucle de pre-procesamiento, que elimina las instrucciones computacionalmente costosas del bucle original. En este paso, se filtran las regiones sin suficiente información. Además, para aquellas regiones que pasan el filtro, se calcula y almacena en memoria datos útiles como la cantidad de información en sus límites.
- Un bucle de cómputo, donde se ejecutan los modelos para las regiones que contienen suficiente información.

El bucle de pre-procesamiento proporciona a la herramienta paralela la información necesaria para intentar obtener un equilibrio en la carga de trabajo. Las siguientes secciones describen varias estrategias de balanceo de carga basadas en esta información.

C.1 Distribución por bloques pura

Como primer paso se implemento una aproximación *naive*. Gracias al bucle de pre-procesamiento, el número total de regiones a analizar se conoce de antemano. La herramienta utiliza esta información para distribuir una cantidad equitativa de regiones entre los procesos MPI.

Una distribución por bloques contiguos es más apropiada que una distribución cíclica, ya que las regiones contiguas tienen lecturas contiguas y, por lo

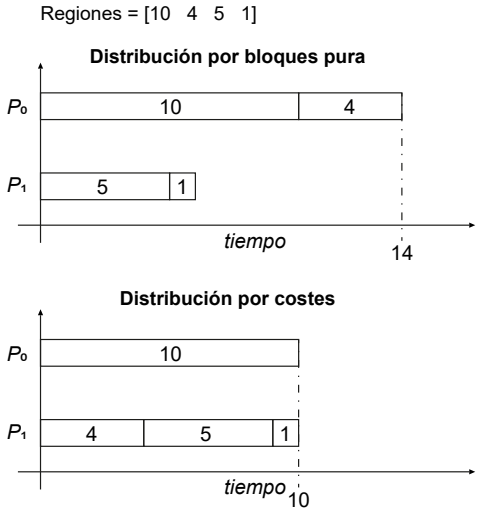


Fig. 5: Ejemplo comparando el rendimiento de la distribución de bloques pura y la basada en costes

tanto, cada proceso MPI solo necesita almacenar un bloque de lecturas. Esto ayuda a mejorar la localidad de datos y evita que los procesos tengan que acceder a bloques de datos que están separados entre sí por un desplazamiento, lo que conllevaría una degradación del rendimiento.

Debido a las razones explicadas en la Sección IV-B, las regiones tendrán diferencias enormes en su coste computacional. Esto genera situaciones en las que, aunque el número de regiones esté distribuido de forma equitativa, la carga de trabajo no lo está.

C.2 Distribución por bloques basada en costes

El enfoque anterior podría presentar, dependiendo del conjunto de datos de entrada, un desequilibrio de carga de trabajo severo a nivel de proceso. Para mejorar el balanceo de la carga, se diseñó una distribución estática basada en el coste de las regiones.

En esta distribución el coste de cada región se asigna según su cantidad de información. Es decir, en lugar de recibir bloques con la misma cantidad de regiones, los procesos reciben bloques de tamaño variable pero con la misma cantidad total de información. De esta manera, se espera que cada proceso MPI tenga una carga de trabajo total similar para distribuir entre sus hilos OpenMP.

La Figura 5 muestra un ejemplo de la comparación del rendimiento de la distribución por bloques pura y la basada en costes, utilizando cuatro regiones con tiempo de ejecución variable (representado por el vector *Regiones*). El primero distribuye dos elementos a cada proceso, sin centrarse en el coste de ejecución, lo que hace que *P1* permanezca inactivo desde $t = 6$ hasta $t = 14$. La distribución basada en costes, sin embargo, tiene en cuenta este factor, dando solo un elemento a *P0* y tres a *P1*, lo que hace que ambos procesos terminen la ejecución en $t = 10$.

Este nuevo enfoque tiene un gran impacto en la reducción del desequilibrio debido a las *regiones ele-*

fante, ya que tiene en cuenta su gran cantidad de información. Sin embargo, no logra distribuciones totalmente equilibradas por dos razones. En primer lugar, la estimación del coste, aunque precisa, no es exacta, por lo que esta fuente de desequilibrio se reduce, pero no se elimina. En segundo lugar, y más importante, el enfoque no tiene en cuenta el desequilibrio causado por el número de iteraciones necesarias para ajustar los modelos (ver Sección IV-B).

C.3 Distribución dinámica

Para hacer frente al desequilibrio de carga de trabajo debido al hecho de que algunas regiones necesitan más iteraciones para converger se diseñó una distribución dinámica.

La idea principal detrás de este enfoque es asignar pequeños bloques de regiones bajo demanda, minimizando la sincronización entre los procesos MPI y los costes de comunicación. Para este propósito, se utilizaron comunicaciones RMA pasivas y tres estructuras fueron asignadas a memoria compartida RMA del proceso raíz:

- Una porción de memoria compartida para almacenar de forma contigua los bloques de regiones.
- Un array compartido para almacenar los resultados de los cálculos.
- Un índice compartido, inicializado a cero, que apunta al siguiente bloque a analizar.

Tras la inicialización, todos los procesos entran en un bucle donde obtienen el índice actual y lo incrementan a la siguiente posición utilizando *MPI_Fetch_and_Op*. Si el índice está dentro de los límites, el proceso obtiene el bloque de regiones de la memoria RMA utilizando *MPI_Get*, calcula sus resultados y los almacena de nuevo en el desplazamiento correcto del búfer de salida RMA utilizando *MPI_Put*.

No obstante, una implementación *naive* de este algoritmo no conduce a una mejora del rendimiento, ya que las *regiones elefante* siguen provocando un desequilibrio de carga de trabajo. En primer lugar, el tiempo de ejecución de los bloques que contienen estas regiones es mucho mayor que el resto, y un solo *bloque elefante* puede llenar todo el tiempo de ejecución de la herramienta en un solo proceso. Además, si alguno de estos *bloques elefante* se calcula al final de la distribución, se crea una situación en la que solo un proceso trabaja, mientras que el resto están inactivos.

Para hacer frente a este problema, la distribución dinámica se realiza en dos pasos. En primer lugar, los bloques que contienen *regiones elefante* se distribuyen utilizando un tamaño de bloque pequeño para asegurar que estas regiones no sobrecarguen a un proceso en particular durante todo su tiempo de ejecución. Además, estas *regiones elefante* se asignan primero a los procesos, ya que son las que más tiempo tardan, y encontrar una al final de la ejecución conduciría a desequilibrios masivos. Finalmente, las regiones restantes se distribuyen entre los procesos utilizando un tamaño de bloque más grande.

El tamaño de los bloques tendrá un gran impacto en el rendimiento de la herramienta. Bloques demasiado pequeños conducen a sobrecargas de comunicación. Por el contrario, bloques demasiado grandes aumentan el desequilibrio potencial entre ellos. Por estas razones, el tamaño del bloque se calcula en tiempo de ejecución para hacer el programa flexible y capaz de manejar diferentes conjuntos de datos y configuraciones. En este cálculo se tiene en cuenta la cantidad de procesos y el coste total del conjunto de datos, garantizando una cantidad mínima de bloques por proceso e intentando dividir cada conjunto de datos en aproximadamente el mismo número de bloques, tanto para asegurar que los conjuntos de datos de bajo coste se distribuyen con una granularidad suficientemente fina, como para asegurar que los conjuntos de datos más caros no pierden tiempo con comunicaciones y sincronizaciones innecesarias que no mejoran el equilibrio. Además, para asegurar que cada bloque tarda un tiempo razonable en procesarse, también se tiene en cuenta la relación entre hilos y procesos, aumentando el tamaño del bloque proporcionalmente al número de hilos utilizados para cada bloque.

D. Entrada paralela

Un análisis de rendimiento de una versión preliminar de *PARamrfinder* señaló que la lectura de la entrada y el bucle de pre-procesamiento se habían convertido en un cuello de botella tras la paralelización de la fase de identificación. Por lo tanto, estas fases fueron rediseñadas para aprovecharse también de la computación paralela.

En primer lugar, se utilizaron funciones MPI-I/O para paralelizar la lectura de datos, permitiendo que cada proceso lea desde un cierto desplazamiento. El formato de entrada (ver Figura 1) es muy apropiado para aprovechar la computación paralela en la fase de pre-procesamiento, ya que cada proceso solo está interesado en un bloque de lecturas consecutivas (filas) que deben ser mapeadas en una serie de regiones consecutivas. Por lo tanto, en *PARamrfinder* cada proceso solo lee un bloque de filas que debe mapear en regiones para ejecutar la fase de pre-procesamiento. Sin embargo, este enfoque presenta cuatro inconvenientes principales:

- No todas las filas tienen la misma longitud.
- El número de filas en un archivo no se conoce de antemano.
- El mapeo de lecturas en regiones no es uno a uno, sino N a N.
- Las regiones al principio y al final de la lista de regiones de un proceso pueden estar duplicadas e incompletas, ya que pueden carecer de información de las lecturas en procesos adyacentes.

El algoritmo de la Figura 6 muestra el pseudocódigo de la implementación paralela para la fase de entrada que resuelve todos los problemas señalados. En primer lugar, *PARamrfinder* obtiene de antemano el

```
1 procedure readCandidateRegions(epireads_file)
2   file_chunk := ReadInputFileToMemoryInParallel(
        epireads_file)
3   reads := ParseFromRawToEpireads(file_chunk)
4   local_regions := IdentifyRegionsFromEpireads(
        reads)
5   last_local_valid_region :=
        FilterLeftoverRegions(local_regions)
6   first_local_valid_region :=
        CommunicateLastValidRegion(last_local_
        valid_region)
7   RemoveInvalidLeadingWindows(local_regions,
        first_local_valid_region)
8   regions := GatherRegionsOnRootProcess(local_
        regions)
9 end procedure
```

Fig. 6: Pseudocódigo de la fase de entrada de *PARamrfinder*

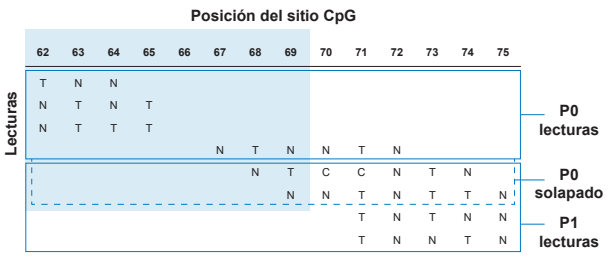


Fig. 7: Ejemplo de distribución de un fichero *epiread* con MPI en la entrada paralela

tamaño del archivo de entrada (en bytes) y luego los distribuye entre los procesos tratando de generar una distribución justa de lecturas (Líneas 2 y 3). Para asegurarse de que cada proceso no pierda ninguna información relacionada con sus regiones, se implementa una técnica de superposición: suponiendo p procesos, el proceso $n \in [0, p\text{-}1]$ lee su bloque y algunos bytes finales adicionales para asegurarse de que puede procesar correctamente todas las regiones (Línea 4), incluso aquellas que pueden compartir información con el proceso n+1. Esto resuelve el problema de las regiones incompletas, pero no el problema de las regiones duplicadas. Para evitar esto, el proceso n elimina sus últimas regiones hasta que encuentra una que puede no estar completa en el proceso n+1 (Línea 5). Luego, el proceso n comparte esta ventana con el proceso n+1 (Línea 6) y el proceso n+1 elimina sus primeras regiones hasta que encuentra una que no está completa en el proceso n (Línea 7). Finalmente, todas las regiones se recopilan en un vector en el proceso raíz (Línea 8) para ser procesadas en la fase de identificación como se explica en la Sección IV-C.

La Figura 7 muestra un ejemplo de la distribución de las lecturas en un archivo *epiread* entre dos procesos utilizando este algoritmo. En la figura, ambos procesos P0 y P1 se asignan un bloque de lecturas. Sin embargo, con este bloque, ninguno de ellos puede identificar correctamente la ventana seleccionada, ya que ambos pierden parte de los datos. La superposición adicional resuelve este problema para P0, ya que ahora tiene todas las lecturas que necesita, pero no para P1, que todavía identifica incorrectamente esta región. P0 debe comunicar a P1 que ha identificado correctamente esta ventana para que P1 la elimine y evitar duplicados.

```
1 procedure mapAMRsToReferenceGenome(amrs, fasta_
      files)
2   local_chrom_mapping :=
        IdentifyChromsOnFastaFiles(fasta_files)
3   chrom_mapping := ShareChromMapping(local_chrom
        _mapping)
4   local_amrs := DistributeAMRsAmongProcesses(
        amrs)
5   MapAMRsToReferenceGenome(local_amrs, chrom_
        mapping)
6   amrs := GatherAMRsOnRootProcess(local_amrs)
7 end procedure
```

Fig. 8: pseudocódigo de la fase de post-procesado de *PARamrfinder*

E. Post-procesado paralelo

Tras la fase de identificación, la herramienta ejecuta varios pasos de post-procesado sobre las AMRs identificadas. Como ya se ha mencionado, el más costoso de estos pasos es el mapeo de estos AMRs al genoma de referencia. De hecho, se convirtió en el principal cuello de botella después de la paralelización de la fase de entrada. Por lo tanto, también se paralelizó, aplicando el algoritmo mostrado el la Figura 8.

El objetivo es distribuir los AMRs entre los procesos y los hilos para que puedan mapearse en paralelo. Sin embargo, la lectura del genoma de referencia es tan costosa como este mapeo, por lo que las operaciones de entrada de los archivos *FASTA* también deben realizarse en paralelo para deshacerse del cuello de botella. Dado que cada proceso trabaja con un subconjunto de los AMRs, solo necesita leer un subconjunto de los cromosomas del genoma de referencia. Sin embargo, la ubicación de cada cromosoma en el archivo de entrada no se conoce de antemano. Por lo tanto, primero se debe escanear el genoma de referencia para identificar la ubicación de cada cromosoma (Línea 2). Si todos los cromosomas se almacenan en un único archivo *FASTA*, cada proceso escaneará un bloque de este. Si cada cromosoma se almacena en un archivo *FASTA* diferente, estos archivos se distribuyen entre los procesos usando un algoritmo *round-robin* y son escaneados en paralelo. Después de este escaneo inicial, los procesos comparten su información (primero comparten el número de cromosomas que han identificado, usando *MPI_Allgather*, y luego comparten la información real, usando *MPI_Allgatherv*), para que todos ellos puedan localizar todos los cromosomas (Línea 3). Estos dos pasos permiten que los procesos tengan acceso directo a cualquier subconjunto particular de cromosomas para que puedan leerlos a memoria sin ningún costo adicional. Una vez logrado este objetivo, las AMRs se distribuyen entre los procesos (Línea 4). A continuación, cada proceso mapea sus AMRs a su subconjunto de cromosomas utilizando todos sus hilos de forma concurrente (Línea 5). Finalmente, los AMRs se reúnen en el proceso raíz (Línea 6) para que se pueda realizar el siguiente paso de post-procesamiento.

Tabla I: Especificaciones de los conjuntos de datos

Conjuntos de datos	Genoma de referencia	Tamaño del fichero epiread	#Lecturas
ERS2586503	hg19	984 MB	52,837,538
ERS4575883	mm10	2.0 GB	117,430,460
ERS7819375	mm38	1.6 GB	97,424,137
ERS208315	hg38	9.1 GB	574,149,340

V. Evaluación experimental

La evaluación experimental de *PARamrfinder* se ha realizado en términos de tiempo de ejecución y escalabilidad, ya que nuestra herramienta proporciona las mismos AMRs que *amrfinder*, cuya alta precisión se ha demostrado en [4]. Todos los tiempos mostrados en esta sección son la media de 3 ejecuciones. Esta sección proporciona una comparación de rendimiento de ambas herramientas en un clúster de ocho nodos con un total de 256 núcleos de CPU (32 núcleos por nodo). Cada nodo tiene dos procesadores Intel Xeon Silver 4216 Cascade Lake-SP de 16 núcleos con soporte para Hyperthreading (hasta dos hilos lógicos por núcleo de CPU) y 256 GB de memoria. Los nodos están interconectados a través de una red InfiniBand EDR de baja latencia y alto ancho de banda. En cuanto al software, tanto *amrfinder* como *PARamrfinder* se han compilado con el compilador GNU GCC v.8.3.0, y este último está vinculado a la biblioteca OpenMPI v.4.0.5.

Para esta evaluación experimental se utilizaron cuatro conjuntos de datos biológicos reales que se denominan según el identificador SRA de su muestra relacionada. El conjunto de datos *ERS2586503* se utiliza para investigar el fenotipo epigenético de los adenomas/pólipos serrados sésiles [18]. El conjunto de datos *ERS4575883* proporciona información relacionada con la detección de interacciones moleculares individuales de factores de transcripción y nucleosomas con ADN in vivo [19]. El conjunto de datos *ERS7819375* aporta células resistentes al tratamiento en el cáncer de mama [20]. Estos tres conjuntos de datos se han obtenido del repositorio público de datos SRA de NCBI [3], mientras que el conjunto de datos *ERS208315* ha sido generado por el Blueprint Consortium a partir de datos de sangre venosa [4].

La Tabla I resume las características de estos conjuntos de datos. Incluye información sobre los genomas de referencia, el tamaño de los archivos *epiread* derivados y el número de lecturas dentro de estos archivos. Nótese que los archivos *epiread* pueden alcanzar varios gigabytes y cientos de millones de lecturas.

A. Experimentos sobre la herramienta secuencial

El primer paso de la evaluación experimental es comprobar el impacto de la optimización secuencial presentada en la sección IV-A. La versión original de *amrfinder* ha sido comparada con la versión optimizada utilizando los parámetros por defecto y cambiando el valor del número máximo de iteraciones para ajustar los modelos (opción *-i* en la línea de comandos). La herramienta se ha probado con un máximo de 10 (por defecto), 100 y 1000 iteraciones, para ver el impacto de esta optimización a medida que la ejecución de los modelos estadísticos gana relevancia. La Tabla II muestra el tiempo de ejecución de la herramienta con y sin la optimización. Algunos tiempos de ejecución no se muestran en la tabla ya que exceden el tiempo máximo de ejecución permitido en el clúster (72 horas). El tiempo de ejecución con la optimización es siempre menor que el de la versión original, logrando una reducción de alrededor de la mitad del tiempo original.

Tabla II: Tiempo de ejecución para la versión original y optimizada de *amrfinder* (en segundos) y aceleración variando el número máximo de iteraciones para ajustar los modelos estadísticos

Conjunto de datos	Versión	-i 10	-i 100	-i 1000
ERS2586503	Original	4,152.08	26,484.37	153,887.21
	Optimizada	1,890.68	11,069.62	63,342.55
	Aceleración	2.20	2.39	2.43
ERS4575883	Original	8,124.49	58,332.44	—
	Optimizada	3,718.81	25,964.10	167,814.11
	Aceleración	2.18	2.25	—
ERS7819375	Original	4,031.03	12,191.98	45,779.46
	Optimizada	2,349.74	6,786.17	25,525.28
	Aceleración	1.72	1.80	1.79
ERS208315	Original	31,215.27	222,083.28	—
	Optimizada	12,898.39	89,434.31	—
	Aceleración	2.42	2.48	—

B. Evaluación de las optimizaciones de I/O paralela

La tabla III muestra los tiempos de ejecución (en segundos) obtenidos por *PARamrfinder* en las fases de pre-procesamiento y post-procesamiento al utilizar las optimizaciones paralelas presentadas en las secciones IV-D y IV-E para un número variable de núcleos, y comparadas con una versión secuencial. Concretamente, la tabla muestra los tiempos de referencia para ambas fases y los tiempos paralelos al utilizar desde un nodo completo (32 núcleos) hasta ocho nodos (256 núcleos). Nótese que el Hyperthreading está activado, permitiendo utilizar hasta 64 hilos por nodo (dos hilos lógicos por núcleo). El número máximo de iteraciones para ajustar los modelos estadísticos se deja por defecto, ya que no afecta a estas fases. Como la estructura de la fase de preprocesamiento ha sido modificada respecto a la herramienta original (ver sección IV-D), tomarla como referencia sería injusto. En su lugar, para este experimento la referencia es el tiempo de ejecución de *PARamrfinder* con un proceso y un hilo. Para la fase de post-procesamiento la referencia es el tiempo de ejecución de esta fase en la herramienta original utilizando la misma configuración.

Estos resultados son satisfactorios, ya que el cuello de botella tanto de la fase de pre-procesamiento como de la de post-procesamiento son eliminados y los tiempos de ejecución se reducen de cientos a menos de cinco segundos en casi todos los casos. No solo eso, sino que hay una excepción positiva entre estos resultados: el tiempo de ejecución de la fase de preprocesamiento del conjunto de datos *ERS208315*, que escala mucho mejor que el resto de casos. Es-

[3] https://www.ncbi.nlm.nih.gov/sra
[4] https://ega-archive.org/datasets/EGAD00001002523

Tabla III: Tiempo de ejecución (en segundos) de las fases de pre-procesamiento y post-procesamiento de *PARamrfinder* variando el número de núcleos

Conjunto de datos	Fase	Tiempo de referencia	32n	64n	128n	256n
ERS2586503	Pre-proc.	134.69	7.65	4.93	3.18	1.81
	Post-proc.	146.56	6.64	3.90	2.52	1.99
ERS4575883	Pre-proc.	236.43	12.88	8.97	5.48	2.88
	Post-proc.	81.44	5.33	3.45	2.16	1.65
ERS7819375	Pre-proc.	202.49	12.80	8.27	5.53	3.07
	Post-proc.	68.99	6.13	3.27	2.06	1.51
ERS208315	Pre-proc.	3,068.30	100.53	53.37	29.1	15.20
	Post-proc.	173.31	11.81	7.46	4.66	3.51

to ocurre porque el tiempo secuencial de esta fase es mucho mayor que el resto y, al paralelizarla utilizando los 256 núcleos, esta fase aún tarda más de 15 segundos en ejecutarse, por lo que las sincronizaciones y sobrecargas introducidas en la versión paralela tienen su impacto diluido.

C. Evaluación de los algoritmos de balanceo de carga

Un punto clave en el rendimiento de *PARamrfinder* es su capacidad para balancear la carga de trabajo. Como se explicó en la sección IV-C, se han implementado tres algoritmos diferentes en la herramienta paralela para distribuir las regiones entre los procesos MPI: distribución por bloques pura, distribución por bloques basada en costes y distribución dinámica.

La Figura 9 muestra las aceleraciones obtenidas por los tres algoritmos al utilizar 256 núcleos, un máximo de 10 y 1000 iteraciones para ajustar los modelos estadísticos, y todos los conjuntos de datos disponibles. Todas las ejecuciones usan Hyperthreading, con dos hilos lógicos por CPU (512 hilos lógicos en total). A partir de ahora, todas las ejecuciones se harán con Hyperthreading. La referencia es el tiempo de ejecución de la herramienta original con la optimización secuencial explicada en la sección IV-A. Como el tiempo de ejecución de referencia del conjunto de datos *ERS208315* no se puede calcular para 1000 iteraciones máximas debido al límite de tiempo de ejecución de 72 horas, se ha estimado un tiempo de ejecución de referencia para ese conjunto de datos asumiendo una aceleración de 32x para la ejecución de *PARamrfinder* con una distribución dinámica usando 32 núcleos (un nodo completo) y un máximo de 1000 iteraciones.

El algoritmo dinámico proporciona el mejor rendimiento en todos los casos, al borde de las aceleraciones superlineales para el máximo de 1000 iteraciones. También es destacable que es el algoritmo más consistente, ya que los otros son más sensibles a las características específicas de los conjuntos de datos. Por lo tanto, este será el algoritmo incluido en la versión final de la herramienta paralela, y el utilizado en los experimentos de escalabilidad presentados en la siguiente sección.

D. Escalabilidad de PARamrfinder

Las pruebas de escalabilidad comenzaron analizando el rendimiento dentro de un nodo, utilizando un proceso por CPU y 2, 4, 8 y 16 núcleos por proceso (32 núcleos en total). Esta configuración de dos procesos por nodo fue elegida porque es la que proporciona el mejor rendimiento en un solo nodo, ya que mejora el ancho de banda de la memoria y la localidad, y se mantendrá al aumentar el número de nodos. La Figura 10 muestra las aceleraciones obtenidas por *PARamrfinder* cuando se utilizan máximos de 10 y 1000 iteraciones para ajustar los modelos estadísticos, todos los conjuntos de datos disponibles, la distribución dinámica y un número variable de núcleos dentro de un nodo. La referencia es nuevamente el tiempo de ejecución de la herramienta original con la optimización secuencial.

Se puede observar que *PARamrfinder* escala bien con el número de núcleos, manteniendo aceleraciones superlineales cuando se llena un nodo para varios conjuntos de datos para un máximo de 1000 iteraciones. También se puede observar que las aceleraciones obtenidas por la herramienta son mucho mayores cuando se especifica un máximo de 1000 iteraciones que cuando se indica solo 10, lo que implica que la herramienta funciona mejor para cargas de trabajo pesadas. Esto se debe a que las fases de preprocesamiento y posprocesamiento de la herramienta ganan relevancia cuando la carga de trabajo es menor, y no escalan tan bien como la fase de identificación con la distribución dinámica. Además, el tiempo de ejecución de la herramienta para un máximo de 10 iteraciones se reduce a menos de dos minutos para la mayoría de los conjuntos de datos, por lo que los pequeños sobrecostes también ganan relevancia en estos tiempos de ejecución reducidos.

La Figura 11 muestra las aceleraciones obtenidas por *PARamrfinder* cuando se utilizan máximos de 10 y 1000 iteraciones para ajustar los modelos estadísticos, la distribución dinámica, todos los conjuntos de datos disponibles y 1, 2, 4 y 8 nodos. La referencia es el tiempo de ejecución de la herramienta paralela en un nodo. Estos resultados demuestran que *PARamrfinder* escala bien con el número de nodos, y que su escalabilidad es consistente para todos los conjuntos de datos.

Esta evaluación experimental demuestra que *PARamrfinder* puede ser útil para los científicos con el fin de reducir drásticamente el tiempo de ejecución necesario para identificar AMRs. La Tabla IV proporciona un resumen de esta reducción de tiempo de ejecución cuando se utiliza un máximo de 1000 iteraciones para ajustar los modelos estadísticos. En ella se puede observar que *PARamrfinder* es más rápido que *amrfinder* incluso cuando se utiliza el mismo hardware (un núcleo). Además, la nueva herramienta paralela permite ejecuciones que no eran posibles con la herramienta original. Por ejemplo, los análisis de los conjuntos de datos *ERS4575883* y *ERS208315*, que antes excedían el máximo de 72 horas permitido por el clúster, ahora se pueden completar en menos de 9 y 45 minutos, respectivamente, usando ocho nodos.

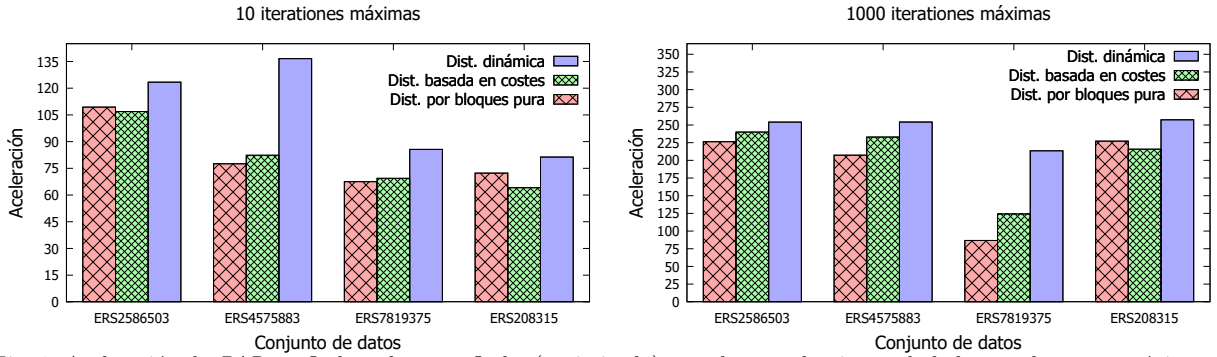


Fig. 9: Aceleración de *PARamrfinder* sobre *amrfinder* (optimizado) usando tres algoritmos de balanceo de carga, máximos de 10 y 1000 iteraciones, los diferentes conjuntos de datos y ocho nodos del clúster (256 núcleos)

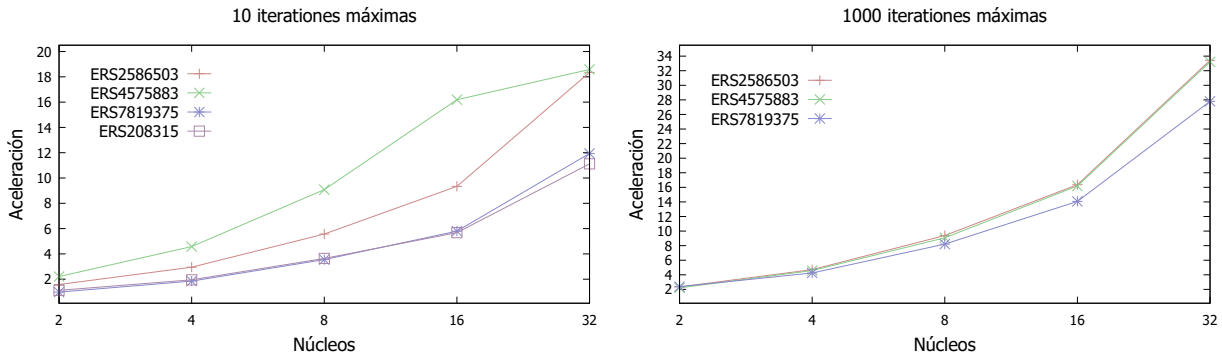


Fig. 10: Aceleración de *PARamrfinder* sobre *amrfinder* (optimizado) usando máximos de 10 y 1000 iteraciones para los diferentes conjuntos de datos variando el número de núcleos en un solo nodo

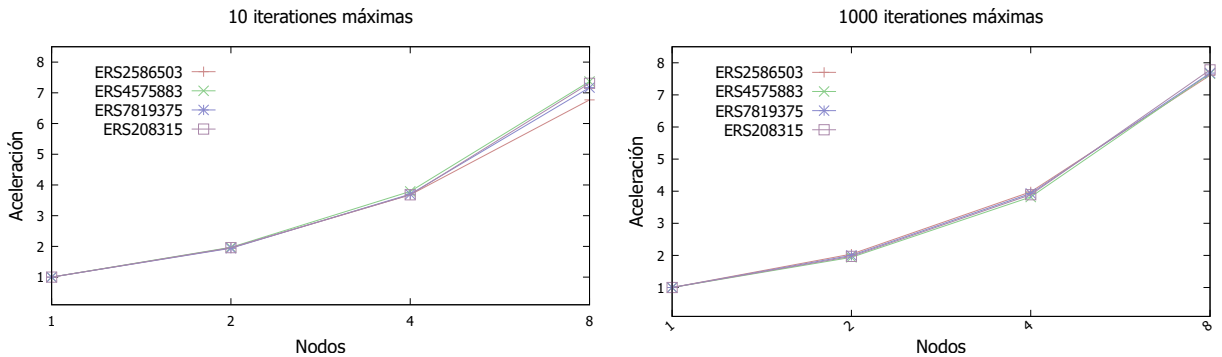


Fig. 11: Aceleración de *PARamrfinder* usando máximos de 10 y 1000 iteraciones para los diferentes conjuntos de datos variando el número de nodos. La referencia es el tiempo de ejecución de PARamrfinder en un nodo completo

Tabla IV: Tiempo de ejecución para distintas versiones de la herramienta (en segundos) para 1000 iteraciones máximas para ajustar los modelos estadísticos

Conjunto de datos	amrfinder	PARamrfinder		
	1 núcleo	1 núcleo(Ht)	1 nodo(Ht)	256 núcleos(Ht)
ERS2586503	123,887	51,474	1,542	202
ERS4575883	—	136,364	3,901	509
ERS7819375	38,779	20,261	728	94
ERS208315	—	—	16,408	2,089

VI. Conclusiones

La metilación del ADN es un campo de investigación muy activo en los últimos años. En concreto, la detección de ASM bajo distintas condiciones biológicas tiene una gran relevancia, ya que puede ayudar a entender la función de la impronta genómica. Sin embargo, estos análisis pueden llevar un tiempo muy elevado para conjuntos de datos grandes o incluso medianos.

Para aliviar este problema, este trabajo presenta *PARamrfinder*, una aplicación paralela que obtiene los mismos resultados biológicos que la popular herramienta *amrfinder*, pero con un tiempo de ejecución significativamente menor gracias a la explotación del hardware de los clústeres multinúcleo modernos. La herramienta emplea una implementación paralela híbrida MPI/OpenMP, que aporta importantes beneficios, como la capacidad de usar *Hyperthreading*, una gestión de memoria eficiente y menos sincronizaciones entre los elementos de procesamiento. Además, es capaz de obtener una gran escalabilidad gracias a su equilibrio dinámico de carga de trabajo tanto a nivel de proceso como de hilo.

La evaluación experimental se ha realizado en un clúster con ocho nodos, cada uno con 32 núcleos (un total de 256 núcleos), usando cuatro conjuntos de datos representativos con datos biológicos reales y diferentes características. *PARamrfinder* es más rápido que *amrfinder* en todos los escenarios, incluso usando los mismos recursos hardware (un núcleo). Su impac-

to es más notable para un gran número de recursos, siendo capaz de reducir una ejecución de varios días (más de 72 horas) a menos de nueve minutos.

Como trabajo futuro planeamos aplicar enfoques paralelos similares a otras etapas del pipeline *MethPipe*, de forma que las diferentes etapas puedan integrarse para aprovechar los recursos de un clúster multinúcleo.

Agradecimientos

Este estudio hace uso de datos generados por el Consorcio Blueprint, el consorcio de investigadores que fue financiado por el Séptimo Programa Marco de Investigación y Desarrollo de la Unión Europea (FP7 / 2007-2013) bajo el acuerdo de subvención n.° 282510 BLUEPRINT.

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación de España (PID2019-104184RB-I00 / AEI / 10.13039/501100011033), el Ministerio de Universidades de España bajo la subvención FPU21/03408, y por fondos de la Xunta de Galicia y FEDER (Centro de Investigación de Galicia acreditación 2019-2022 y Programa de Consolidación de Grupos Competitivos de Referencia, con cargo a las ayudas ED431G 2019/01 y ED431C 2021/30, respectivamente).

Referencias

[1] Robert D Nicholls, Joan HM Knoll, Merlin G Butler, Susan Karam, and Marc Lalande, "Genetic imprinting suggested by maternal heterodisomy in non-deletion Prader-Willi syndrome," *Nature*, vol. 342, no. 6247, pp. 281–285, 1989.

[2] Angela M Mabb, Matthew C Judson, Mark J Zylka, and Benjamin D Philpot, "Angelman syndrome: insights into genomic imprinting and neurodevelopmental phenotypes," *Trends in neurosciences*, vol. 34, no. 6, pp. 293–303, 2011.

[3] Rosanna Weksberg, Adam C Smith, Jeremy Squire, and Paul Sadowski, "Beckwith–Wiedemann syndrome demonstrates a role for epigenetic control of normal development," *Human molecular genetics*, vol. 12, no. suppl_1, pp. R61–R68, 2003.

[4] Fang Fang, Emily Hodges, Antoine Molaro, Matthew Dean, Gregory J. Hannon, and Andrew D. Smith, "Genomic landscape of human allele-specific DNA methylation," *Proceedings of the National Academy of Sciences*, vol. 109, no. 19, pp. 7332–7337, 2012.

[5] Hiroaki Okae, Hatsune Chiba, Hitoshi Hiura, Hirotaka Hamada, Akiko Sato, Takafumi Utsunomiya, Hiroyuki Kikuchi, Hiroaki Yoshida, Atsushi Tanaka, Mikita Suyama, and Takahiro Arima, "Genome-wide analysis of DNA methylation dynamics during early human development," *PLOS Genetics*, vol. 10, no. 12, pp. 1–12, 12 2014.

[6] Catherine Do, Alyssa Shearer, Masako Suzuki, Mary Beth Terry, Joel Gelernter, John M. Greally, and Benjamin Tycko, "Genetic-epigenetic interactions in CIS: A major focus in the post-GWAS era," *Genome Biology*, vol. 18, no. 1, 2017.

[7] Vitor Onuchic, Eugene Lurie, Ivenise Carrero, Piotr Pawliczek, Ronak Y. Patel, Joel Rozowsky, Timur Galeev, Zhuoyi Huang, Robert C. Altshuler, Zhizhuo Zhang, R. Alan Harris, Cristian Coarfa, Lillian Ashmore, Jessica W. Bertol, Walid D. Fakhouri, Fuli Yu, Manolis Kellis, Mark Gerstein, and Aleksandar Milosavljevic, "Allele-specific epigenome maps reveal sequence-dependent stochastic switching at regulatory loci," *Science*, vol. 361, no. 6409, pp. eaar3146, 2018.

[8] Benjamin Tycko, "Allele-specific DNA methylation: beyond imprinting," *Human Molecular Genetics*, vol. 19, no. R2, pp. R210–R220, 09 2010.

[9] Yingying Zhang, Christian Rohde, Richard Reinhardt, Claudia Voelcker-Rehage, and Albert Jeltsch, "Non-imprinted allele-specific DNA methylation on human autosomes," *Genome biology*, vol. 10, pp. 1–11, 2009.

[10] Thomas A Down, Vardhman K Rakyan, Daniel J Turner, Paul Flicek, Heng Li, Eugene Kulesha, Stefan Graef, Nathan Johnson, Javier Herrero, Eleni M Tomazou, et al., "A Bayesian deconvolution strategy for immunoprecipitation-based DNA methylome analysis," *Nature biotechnology*, vol. 26, no. 7, pp. 779–785, 2008.

[11] Qiangwei Zhou, Pengpeng Guan, Zhixian Zhu, Sheng Cheng, Cong Zhou, Huanhuan Wang, Qian Xu, Wingkin Sung, and Guoliang Li, "ASMdb: a comprehensive database for allele-specific DNA methylation in diverse organisms," *Nucleic Acids Research*, vol. 50, no. D1, pp. D60–D71, 10 2021.

[12] Yaping Liu, Kimberly D Siegmund, Peter W Laird, and Benjamin P Berman, "Bis-SNP: combined DNA methylation and SNP calling for Bisulfite-seq data," *Genome biology*, vol. 13, no. 7, pp. 1–14, 2012.

[13] Daniel Andergassen, Christoph P. Dotter, Tomasz M. Kulinski, Philipp M. Guenzl, Philipp C. Bammer, Denise P. Barlow, Florian M. Pauler, and Quanah J. Hudson, "Allelome.PRO, a pipeline to define allele-specific genomic features from high-throughput sequencing data," *Nucleic Acids Research*, vol. 43, no. 21, pp. e146–e146, 07 2015.

[14] Weilong Guo, Ping Zhu, Matteo Pellegrini, Michael Q Zhang, Xiangfeng Wang, and Zhongfu Ni, "CGmapTools improves the precision of heterozygous SNV calls and supports allele-specific methylation detection and visualization in bisulfite-sequencing data," *Bioinformatics*, vol. 34, no. 3, pp. 381–387, 09 2017.

[15] Bui Quang Minh, Le Sy Vinh, Arndt Von Haeseler, and Heiko A Schmidt, "pIQPNNI: parallel reconstruction of large maximum likelihood phylogenies," *Bioinformatics*, vol. 21, no. 19, pp. 3794–3796, 2005.

[16] Jorge Gonzalez-Dominguez and Maria J Martin, "MPI-GeneNet: Parallel calculation of gene co-expression networks on multicore clusters," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 15, no. 5, pp. 1732–1737, 2017.

[17] Kuo-Bin Li, "ClustalW-MPI: ClustalW analysis using distributed and parallel computing," *Bioinformatics*, vol. 19, no. 12, pp. 1585–1586, 2003.

[18] Hannah R Parker, Stephany Orjuela, Andreia Martinho Oliveira, Fabrizio Cereatti, Matthias Sauter, Henriette Heinrich, Giulia Tanzi, Achim Weber, Paul Komminoth, Stephan Vavricka, et al., "The proto CpG island methylator phenotype of sessile serrated adenomas/polyps," *Epigenetics*, vol. 13, no. 10-11, pp. 1088–1105, 2018.

[19] Can Sönmezer, Rozemarijn Kleinendorst, Dilek Imanci, Guido Barzaghi, Laura Villacorta, Dirk Schübeler, Vladimir Benes, Nacho Molina, and Arnaud Regis Krebs, "Molecular co-occupancy identifies transcription factor binding cooperativity in vivo," *Molecular cell*, vol. 81, no. 2, pp. 255–267, 2021.

[20] Ksenija Radic Shechter, Eleni Kafkia, Katharina Zirngibl, Sylwia Gawrzak, Ashna Alladin, Daniel Machado, Christian Lüchtenborg, Daniel C Sévin, Britta Brügger, Kiran R Patil, et al., "Metabolic memory underlying minimal residual disease in breast cancer," *Molecular systems biology*, vol. 17, no. 10, pp. e10141, 2021.

Super-resolución distribuida de imágenes hiperespectrales

Pen Zheng[1], Jin Sun[1], Yang Xu[1], Yi Zhang[1], Zhihui Wei[1], Zebin Wu[1], Javier Plaza[2] y Antonio Plaza[2]

Resumen— **La super-resolución de imágenes hiperespectrales tiene como objetivo fusionar una imagen hiperespectral de baja resolución espacial (LR-HSI) y una imagen multiespectral de alta resolución espacial (HR-MSI) para obtener una imagen hiperespectral de alta resolución espacial (HR-HSI). Este artículo presenta un modelo distribuido para la fusión de imágenes hiperespectrales y multiespectrales (HSI-MSI). Para resolver el problema de la baja eficiencia computacional del método, proponemos una implementación distribuida en Apache Spark para acelerar el proceso de fusión. Los resultados experimentales demuestran que el método propuesto no solo supera a algunos métodos de fusión HSI-MSI de última generación, sino que también logra una tasa de aceleración significativa.**

Palabras clave— **Fusión de imágenes multiespectrales e hiperespectrales, implementación distribuida.**

I. Introducción

EN los últimos años, las imágenes hiperespectrales (HSIs) –que proporcionan abundante información espacial y espectral– han jugado un papel importante en muchas aplicaciones de teledetección. Sin embargo, debido a las limitaciones de los instrumentos de adquisición, no es posible obtener imágenes HSI con alta resolución espacial y espectral simultáneamente. Una posible solución es fusionar una HSI de baja resolución espacial (LR-HSI) y una imagen multiespectral de alta resolución espacial (HR-MSI) de la misma escena para obtener una imagen hiperespectral de alta resolución (HR-HSI) [1]. Este enfoque puede mejorar la resolución espacial de la imagen HSI mientras que se preserva su rica información espectral.

Los métodos de super-resolución se clasifican principalmente en cuatro categorías: sustitución de componentes, análisis de resolución múltiple, enfoques basados en modelos, y enfoques basados en aprendizaje profundo. Sin embargo, los métodos basados en tensores son más costosos en cuanto a sus requerimientos de computación y memoria en comparación con los métodos de factorización de matrices.

Con el volumen rápidamente creciente de las imágenes HSI y la explosión en la demanda de servicios de super-resolución, se ha vuelto un desafío abordar la super-resolución de imágenes HSI. Afortunadamente, la utilización de implementaciones distribuidas puede resolver este problema. La idea fundamental es dividir el problema en pequeños subproblemas y distribuirlos a través de múltiples nodos [2].

Este artículo propone un enfoque novedoso para la fusión distribuida de imágenes hiperespectrales. El método propuesto tiene en cuenta la similitud no local y las características específicas de las imágenes HSI. Las principales contribuciones se resumen a continuación:

1. Proponemos una nueva descomposición tensorial no local para la fusión HSI-MSI que analiza las relaciones entre el dominio espacial, el dominio no local, y el dominio espectral.
2. Implementamos un flujo de procesamiento paralelo y distribuido para acelerar aún más el proceso de fusión. La relación máxima de aceleración lograda por el flujo de procesamiento paralelo puede ser de hasta 14×.

II. Metodología

Denotamos la HSI a procesar como $\mathbf{X} \in R^{W\times H\times S}$, donde W, H y S representan las dimensiones de la imagen (ancho, alto y número de bandas, respectivamente). En consecuencia, la imagen LR-HSI se denota como $\mathbf{Y} \in R^{w\times h\times S}$ y la imagen HR-MSI se denota como $\mathbf{Z} \in R^{W\times H\times s}$, donde $w < W$, $h < H$ y $s < S$. Tanto $\mathbf{Y}$ como $\mathbf{Z}$ se obtienen degradando $\mathbf{X}$. La degradación se puede modelar como:

$$\mathbf{Z} = \mathbf{XSH} + \mathbf{E}_h \tag{1}$$

y:

$$\mathbf{Y} = \mathbf{X} \times_3 R + \mathbf{E}_m \tag{2}$$

donde $\mathbf{S} \in R^{WH\times WH}$ es un operador de desenfoque espacial, $H \in R^{WH\times wh}$ es un operador de submuestreo espacial, $R \in R^{s\times S}$ es la respuesta espectral del sensor multiespectral, y $\mathbf{E}_h$ y $\mathbf{E}_m$ son funciones de ruido Gaussiano. Nuestro objetivo es fusionar las imágenes LR-HSI y HR-MSI para estimar la imagen HR-HSI objetivo.

El concepto de similitud no local ha sido ampliamente utilizado en la comunidad de procesamiento de imágenes. La idea básica es agrupar sub-cubos o *patches* no locales similares en estructura de la imagen como un nuevo grupo. Como resultado, los *patches* que pertenecen a un mismo clúster están correlacionados, lo que es beneficioso para extraer información a partir de ellos.

Recientemente, el concepto de similitud no local se ha extendido al caso de la fusión de imágenes HSI-MSI. De esta manera, la imagen HR-HSI se divide espacialmente en un grupo de sub-cubos superpuestos en la imagen HR-MSI. d_w y d_h representan

[1]School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China.
[2]Hyperspectral Computing Laboratory, Department of Technology of Computers and Communications, University of Extremadura, Cáceres, Spain.

el ancho y alto de cada sub-cubo, respectivamente. Los sub-cubos que son similares se agrupan en grupos de K sub-cubos mediante el método propuesto en [3]. Después de eso, los cubos agrupados se despliegan espacialmente (de 3D a 2D) y se apilan en una cadena 3D que se trunca a un tamaño fijo N. Un fragmento truncado se considera como un tensor de trayectoria no local (NPT), denotado por $\mathbf{X}^k = \mathbf{G}_k\mathbf{X} \in R^{N_k \times d_w d_h \times S}$, donde $\mathbf{G}_k$ es el operador de extracción NPT y el número de sub-cubos en cada NPT es idéntico, excepto que el penúltimo puede ser menor que N.

Llegados a este punto, podemos fusionar el k-ésimo NPT $\mathbf{G}_k\mathbf{Z}$ y $\mathbf{G}_k\mathbf{Y}$ para generar $\mathbf{G}_k\mathbf{X}$. Formalmente, representamos al NPT como

$$\mathbf{G}_k\mathbf{X} = [\![U_k^1, U_k^2, U_k^3, \mathbf{B}_k^1, B_k^2]\!] \tag{3}$$

donde $U_k^1 \in R^{wh \times R_1}, U_k^2 \in R^{N_k \times R_2}, U_k^3 \in R^{S \times R_3}$ denota los diccionarios del dominio espacial, con átomos R_1, el dominio no local, con átomos R_2, y el dominio espectral, con átomos R_3. $\mathbf{B}_k^1$ (como tensor central) establece una conexión entre el dominio espacial y el dominio no local, que tienen una fuerte correlación. B_k^2 establece la conexión del dominio espectral con los demás dominios. Nuestro método permite una definición más flexible de las relaciones entre dos diccionarios. La resolución final del problema de optimización descrito anteriormente se realiza mediante el método *alternating direction method of multipliers* (ADMM) [4], tal y como se muestra en la Figura 1, que resume el método propuesto.

III. Implementación distribuida

El método no local es conocido por su redundancia, lo que aumenta considerablemente la cantidad de cálculo y el consumo de memoria. Otro desafío importante en su implementación es que los métodos basados en tensores son inherentemente más intensivos desde el punto de vista computacional que los basados en matrices. Además, la dimensionalidad y el volumen de los datos de teledetección están aumentando rápidamente, lo que magnifica los problemas que surgen a la hora de gestionar dichos datos.

Para resolver dichos problemas, presentamos una implementación distribuida con idea de acelerar el método propuesto. Para ello, utilizamos Apache Spark [5] (uno de los motores distribuidos más populares para tratar problemas de *big data*). La base arquitectónica de Spark está consituída por los denominados *resilient distributed datasets* (RDDs), un conjunto de elementos de datos de sólo lectura distribuidos entre un grupo de máquinas y que se mantiene garantizando la tolerancia a fallos. Teniendo en cuenta las características del método propuesto, a continuación proponemos una nueva implemtación distribuida para la fusión de datos HSI-MSI y describimos en detalle la optimización a nivel de arquitectura que se ha llevado a cabo para el desarrollo de la implementación distribuida.

La Figura 2 muestra un diagrama de flujo de la implementación paralela realizada. Primero, optimizamos la sobrecarga de la red durante la distribución de los NPT. La extracción y el agrupamiento de bloques no locales, que se puede implementar de forma rápida, se realiza en el nodo *Master* para generar los NPT. Los operadores de extracción $\mathbf{G}$ se denotan como *indices_RDD*. A continuación se ejecuta el operador *mapPartition* de Apache Spark sobre *indices_RDD* y se deja que cada partición extraiga los NPT nuevamente desde $\mathbf{X}$ y $\mathbf{Y}$, que son transmitidos a los nodos *Slave*. Luego se combinan las variables iniciales $U_k^{1^0}, U_k^{2^0}, U_k^{3^0}, U_k^{4^0}, \mathbf{B}_k^{1^0}, B_k^{2^0}$ con la información en *indices_RDD* utilizando el operador *zip*. El resultado de dicha agrupación se denomina *nonlocal_RDD*. De esta manera, la sobrecarga de la red se puede reducir en gran medida. A continuación, las variables en cada partición se actualizan de forma independiente. De esta forma, se pueden paralelizar sin alterar la solución del problema original. Después de actualizar las variables, *nonlocal_RDD* se almacena en caché para la siguiente iteración. Los NPT *nonlocal_RDD* son mapeados y revertidos al formato de tensor, indicado como NPT_RDD. Esta aproximación puede obtener una buena aceleración a medida que aumenta el número de nodos de procesamiento, sin tener en cuenta el consumo de la red. Finalmente, se ejecuta el operador *collect* sobre NPT_RDD y los NPT se recopilan en el $Master$. La función objetivo $\overline{\mathbf{X}}$ se calcula combinando y promediando los NPT.

IV. Resultados experimentales

Para verificar la precisión y la eficiencia computacional de nuestro método, configuramos un cluster Apache Spark con un nodo $Master$ y cuatro nodos $Slave$. Cada nodo está equipado con una CPU de 32 núcleos, 128 GB de memoria y 500 GB de almacenamiento. Además, se emplean dos imágenes HSI, obtenidas sobre la Universidad de Pavía en Italia y sobre un centro comercial en Washington DC. La primera imagen fue capturada por el sensor ROSIS y consta de 103 bandas espectrales y 610 × 340 píxeles, que son 41 MB. La otra imagen fue obtenida por el sensor HYDICE sobre el National Mall de Washington DC en 1995. Consta de 191 bandas espectrales y 1280 × 307 píxeles, que son 51 MB.

En los experimentos, el método propuesto se compara con varios métodos de fusión HSI-MSI de última generación: *coupled nonnegative matrix factorization* (CNMF) [6], *hyperspectral image super-resolution via subspace-based regularization* (HySure) [7], Naive Bayes y Sparse Bayes [8], *coupled sparse tensor factorization* (CSTF) [9] y *nonlocal coupled tensor CP decomposition* (NCTCP) [10]. Los parámetros utilizados en nuestros experimentos se establecen en los valores predeterminados sugeridos en los papers originales en los que los métodos fueron presentados.

Para la implementación paralela, configuramos los parámetros del clúster Spark como $spark.executor.memory = 35GB$ y $executor.cores = 1$. Se utilizan cuatro métricas de calidad, incluida la relación señal-ruido máxima

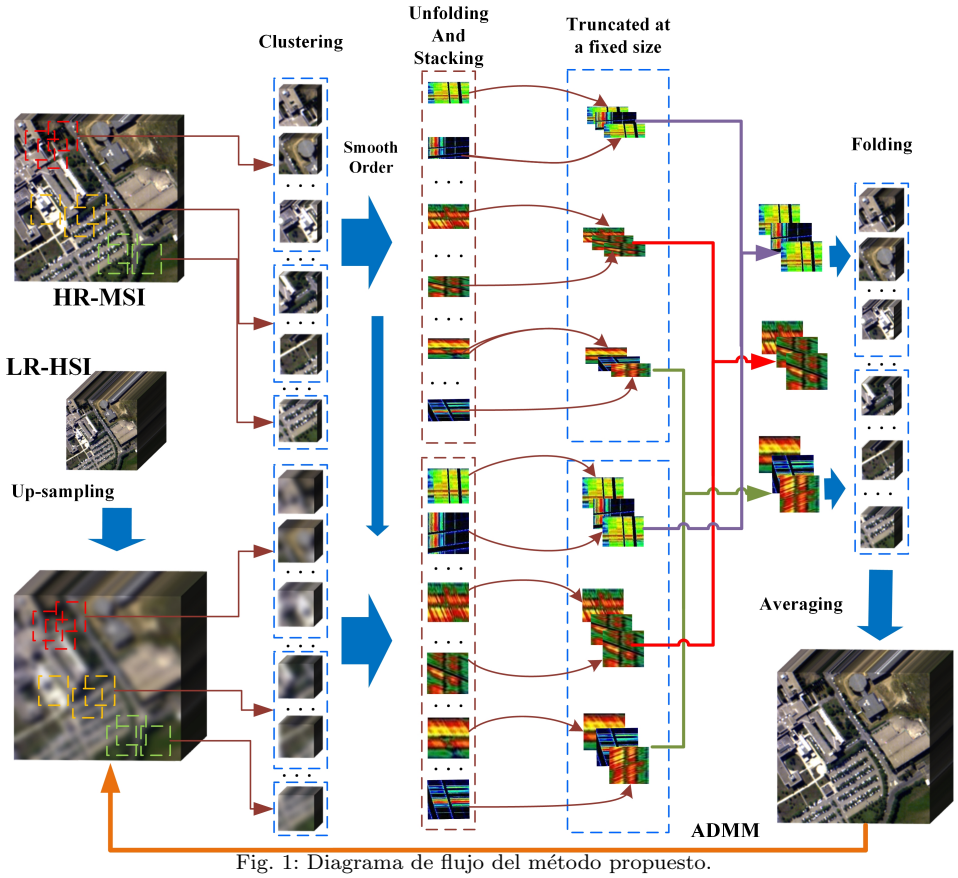


Fig. 1: Diagrama de flujo del método propuesto.

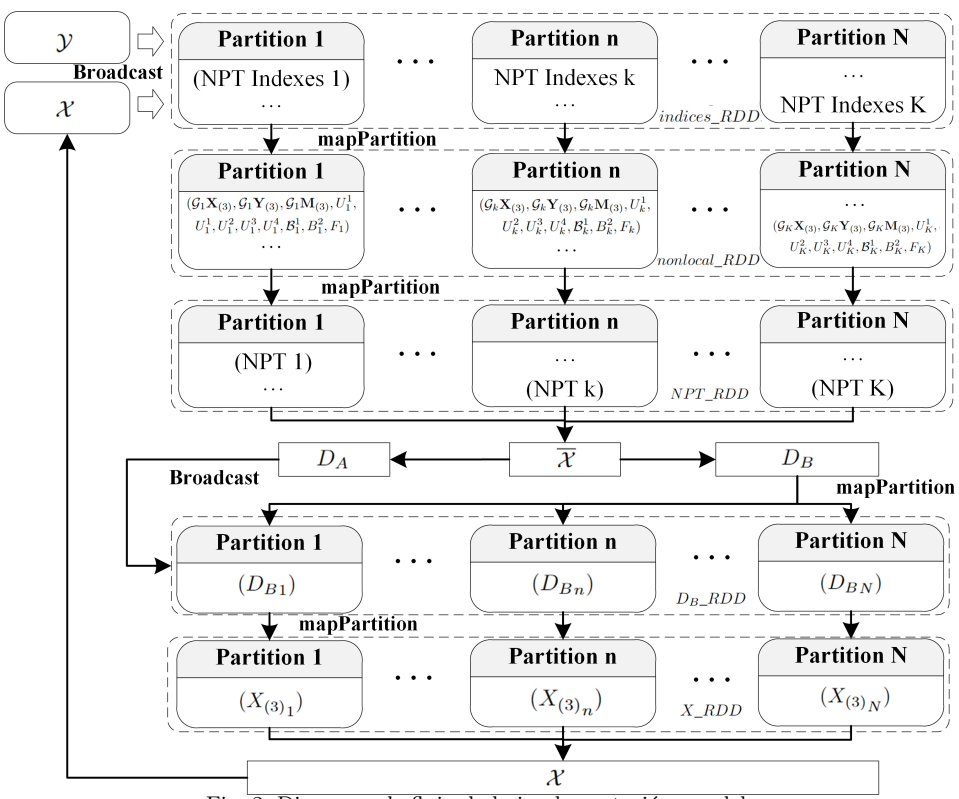
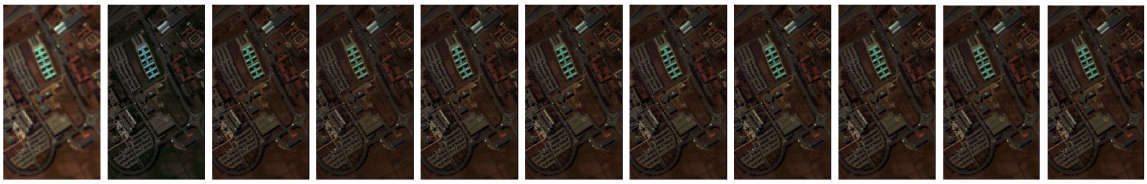


Fig. 2: Diagrama de flujo de la implementación paralela.

(PSNR), la correlación cruzada (CC), el error global adimensional relativo en síntesis (ERGAS) y el ángulo espectral promediado (SAM), para evaluar los resultados experimentales. El paralelismo se analiza en términos del tiempo de procesamiento y el factor de aceleración obtenido.

La Figura 3 muestra las imágenes en falso color del conjunto de datos de la Universidad de Pavía. Los resultados experimentales detallados se presentan en las Tablas I y II, que evalúan el rendimiento de todos los métodos en términos de cuatro métricas cuantitativas. Los mejores valores están resaltados en negri-



(a) LR-HSI (b) HR-MSI (c) Reference (d) CNMF (e) HySure (f) NaiveB. (g) SparseB. (h) CSTF (i) NCTCP (j) NCHTD (k) DNCHTD

Fig. 3: Resultados cualitativos con la imagen obtenida sobre la Universidad de Pavía.

Tabla I: Resultados cuantitativos con la imagen obtenida sobre la Universidad de Pavía. Mejores resultados en negrita.

Método	SAM	PSNR	CC	ERGAS
CNMF	5.85	31.48	0.92	3.84
HySure	5.41	36.61	0.92	3.34
Naive Bayes	5.99	37.76	0.93	3.21
Sparse Bayes	5.93	37.80	0.93	3.17
CSTF	6.06	37.19	0.94	2.95
NPTCP	4.38	37.15	0.96	2.36
Propuesto	**3.64**	**39.60**	**0.97**	**1.99**
Paralelización	3.69	38.89	**0.97**	2.15

Tabla II: Resultados cuantitativos con la imagen obtenida sobre Washington DC Mall. Mejores resultados en negrita.

Método	SAM	PSNR	CC	ERGAS
CNMF	7.38	45.77	0.91	5.09
HySure	5.87	49.59	0.93	4.38
Naive Bayes	4.61	51.23	0.95	3.72
Sparse Bayes	4.47	51.43	0.95	3.56
CSTF	7.91	49.74	0.92	4.65
NPTCP	3.88	**53.03**	**0.97**	**2.79**
Propuesto	**3.55**	52.54	**0.97**	2.80
Paralelización	3.84	52.14	**0.97**	3.09

ta. El modelo propuesto logra los mejores resultados con la imagen sobre la Universidad de Pavia y funciona mejor en términos de CC y SAM con la imagen obtenida sobre el Washington DC Mall. Esto indica que nuestro método obtiene un mejor rendimiento en términos de la representación de características espectrales y morfología. Las versiones distribuidas logran un rendimiento similar porque las estrategias paralelas no alteran la solución del problema original.

Como se muestra en las Tablas III y IV, los tiempos de ejecución del método propuesto disminuyen significativamente a medida que aumenta el número de núcleos (de 1 a 32). La Fig. 4 muestra los factores de aceleración logrados por el método propuesto sobre la imagen de la Universidad de Pavía, mientras que la Figura 5 muestra los factores de aceleración logrados por el método propuesto sobre la imagen del Washington DC Mall. En ambas figuras se aprecia que la implemetación distribuida exhibe una mejora de rendimiento decreciente a medida que aumenta la cantidad de núcleos. Por otra parte, la implementación distribuida se beneficia del hecho de que las tareas paralelas suelen ser independientes entre sí. La relación de aceleración más significativa obtenida es de 14×.

V. Conclusiones

Este artículo presenta un nuevo modelo no local para la fusión de datos HSI-MSI. La similitud no local mejora la calidad del proceso de fusión de imágenes. La relación entre la información espacial y espectral se modela a través de una descomposición más flexible. Además, se desarrolla un método paralelo distribuido eficaz para acelerar el proceso de fusión. Los resultados experimentales obtenidos demuestran que el método propuesto no solo supera significativamente a algunos métodos de fusión HSI-MSI de última generación en términos de la calidad del proceso de fusión, sino que también mejora sustancialmente la eficiencia computacional de dicho proceso.

Agradecimientos

Esta publicación ha sido posible gracias a la financiación proporcionada por la Consejería de Economía, Ciencia y Agencia Digital de la Junta de Extremadura y Fondo Europeo de Desarrollo Regional de la Unión Eropea a través del proyecto GR21040. Este trabajo también ha sido financiado por el Ministerio de Ciencia e Innovación a través del Proyecto PID2019-110315RB-I00 (APRISA), por el programa Horizonte 2020 de la Unión Europea mediante el proyecto 734541 (EOXPOSURE).

Referencias

[1] X. Li, Y. Yuan, and Q. Wang, "Hyperspectral and multispectral image fusion via nonlocal low-rank tensor approximation and sparse representation," IEEE Trans. Geosci. Remote. Sens., vol. 59, no. 1, pp. 550–562, 2021.

[2] Z. Wu, J. Sun, Y. Zhang, Z. Wei, and J. Chanussot, "Recent develop- ments in parallel and distributed computing for remotely sensed big data processing," Proc. IEEE, vol. 109, no. 8, pp. 1282–1305, 2021.

[3] I. Ram, M. Elad, and I. Cohen, "Image processing using smooth ordering of its patches," IEEE Trans. Image Process., vol. 22, no. 7, pp. 2764–2774, 2013.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein et al., "Distributed optimization and statistical learning via the alternating direction method of multipliers," Foundations and Trends in Machine learning, vol. 3, no. 1, pp. 1–122, 2011.

[5] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen et al., "Mllib: Machine learning in apache spark," The Journal of Machine Learning Research, vol. 17, no. 1, pp. 1235–1241, 2016.

[6] N. Yokoya, T. Yairi, and A. Iwasaki, "Coupled nonnegative matrix factorization unmixing for hyperspectral and multispectral data fusion," IEEE Trans. Geosci. Remote. Sens., vol. 50, no. 2, pp. 528–537, 2012.

[7] M. Simoes, J. M. Bioucas-Dias, L. B. Almeida, and J. Chanussot, "A convex formulation for hyperspectral image superresolution via subspace-based regularization," IEEE Trans. Geosci. Remote. Sens., vol. 53, no. 6, pp. 3373–3388, 2015.

[8] Q. Wei, J. M. Bioucas-Dias, N. Dobigeon, and J. Tourneret, "Hyperspectral and multispectral image fusion ba-

Tabla III: Tiempos de ejecución (segundos) del método propuesto con la imagen obtenida sobre la Universidad de Pavía.

Cores	1	2	4	8	16	32
Total	39041	20655	11221	7027	5704	4504
Actualización de factores	34731	17941	9526	5572	4405	3242
Cálculo de $\mathbf{X}$	4085	2492	1460	1228	1064	1030

Tabla IV: Tiempos de ejecución (segundos) del método propuesto con la imagen obtenida sobre el Washington DC Mall.

Cores	1	2	4	8	16	32
Total	99768	51233	26848	14782	11792	7208
Actualización de factores	95399	48442	24742	13085	10443	5974
Cálculo de $\mathbf{X}$	4151	2578	1894	1482	1133	885

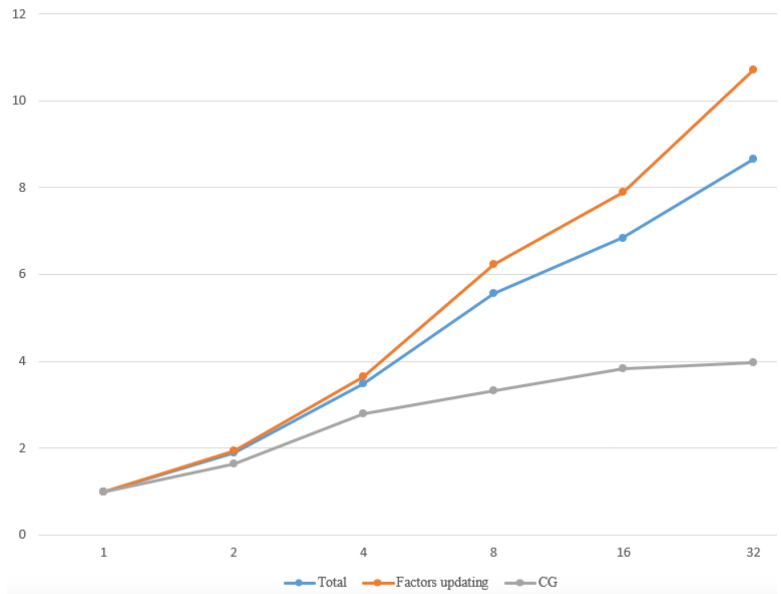


Fig. 4: Factores de aceleración obtenidos por el método distribuido propuesto con la imagen de la Universidad de Pavía.

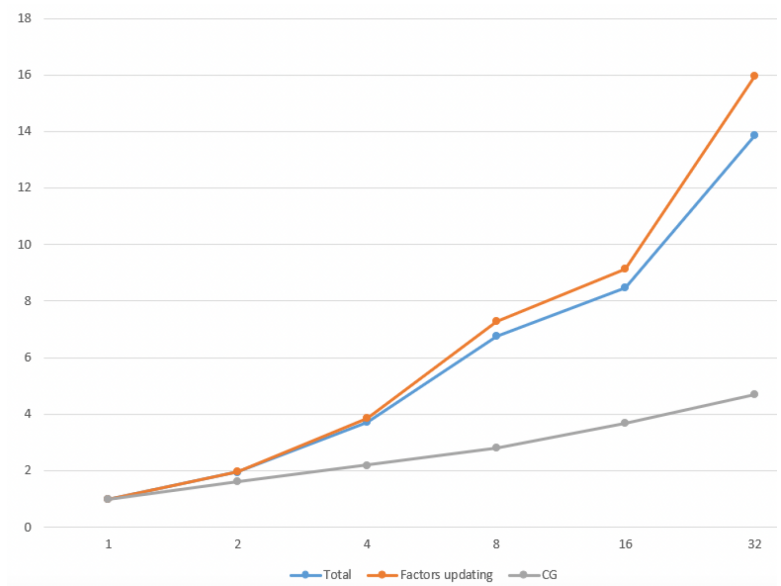


Fig. 5: Factores de aceleración obtenidos por el método distribuido propuesto con la imagen de Washington DC Mall.

sed on a sparse representation," IEEE Trans. Geosci. Remote. Sens., vol. 53, no. 7, pp. 3658–3668, 2015.

[9] S. Li, R. Dian, L. Fang, and J. M. Bioucas-Dias, "Fusing hyperspectral and multispectral images via coupled sparse tensor factorization," IEEE Trans. Image Process., vol. 27, no. 8, pp. 4118–4130, 2018.

[10] Y. Xu, Z. Wu, J. Chanussot, P. Comon, and Z. Wei, "Nonlocal coupled tensor CP decomposition for hyperspectral and multispectral image fusion," IEEE Trans. Geosci. Remote. Sens., vol. 58, no. 1, pp. 348–362, 2020.

Sistema multi-resolución out-of-core para el procesamiento de nubes masivas de puntos en control dimensional

Sergio Constantino Vidal Miramontes [1], Carlos Vázquez Regueiro [2], Álvaro Brage Leira [3], David Díaz Gómez [4], Margarita Amor López [5]

Resumen— **En una amplia gama de industrias, como la naval, la inspección de calidad de los productos manufacturados se ha convertido en una de las principales tareas destinadas a minimizar los problemas de fabricación y los costes de producción. Los sistemas basados en visión artificial permiten obtener un sistema fiable para el control de montaje de piezas en los buques. Sin embargo, necesita datos de muy alta precisión que generan nubes de puntos masivas que implican grandes requerimientos de memoria y computación. En este trabajo se presenta una propuesta de un sistema multi-resolución *out-of-core* que permite el procesamiento de nubes de puntos masivas para el análisis de paneles navales. Esta propuesta se ha comprobado sobre el procedimiento del cálculo del mapa de calor, tanto en una versión multi-hilo como en una versión sobre la GPU (*Graphics Processing Unit*). Los resultados han demostrado que la memoria requerida en estos procesos ha sido reducida, además de permitir la descarga de datos a la GPU de tareas más complejas sin superar la memoria disponible, lo que permite obtener una mejora del rendimiento de 40x de la obtenida con la versión multi-hilo.**

Palabras clave— **OpenMP, CUDA, Nubes de Puntos, Out-of-Core, Control de Calidad**

I. Introducción

El control de calidad se ha convertido en una de las fases más cruciales en el proceso de manufactura de muchos productos en diferentes industrias. Una inspección a tiempo puede reducir considerablemente los errores en la fabricación y los costes de producción en un amplio rango de productos. Las técnicas de visión artificial han sido utilizadas en los últimos años como una de las herramientas de automatización y aceleración de los procesos para la detección de fallos en las líneas de producción [1–3].

Estas técnicas con imágenes 2D, aunque suficientes en algunas industrias, carecen de la precisión necesaria para la naval, que requiere de representaciones tridimensionales, como nubes de puntos [4, 5].

En los últimos años, la empresa naval Navantia ha estado desarrollando herramientas estadísticas de control dimensional para mejorar los procesos de toma de decisiones en la optimización del proceso de manufactura naval [6]. Estas herramientas necesitan datos muy precisos sobre las partes manufacturadas, que se están obteniendo en la actualidad manualmente o mediante algoritmos que presentan una gran carga computacional, lo que requiere una inversión de tiempo muy costosa por parte de los operarios.

Las herramientas de visión artificial utilizadas en este tipo de industrias, que requieren de datos con una precisión muy elevada en forma de nubes masivas de puntos conllevan un elevado requerimiento tanto de memoria como de potencia computacional. Este es un problema que sufren las herramientas de Navantia que procesan nubes con cientos de millones de puntos que llegan a ocupar decenas de *GigaBytes* y que en procesos como el cálculo del mapa de calor llegan a sobrepasar la memoria RAM disponible en CPU o en GPU. Este trabajo presenta una solución a este problema mediante el uso de un sistema multi-resolución *out-of-core* que permite el procesamiento de nubes masivas de puntos.

El uso de este tipo de estructuras en el trabajo con nubes de puntos ya ha sido sujeto de estudio anteriormente [7], tanto en visualización [8, 9], como en procesamiento [10, 11]. Desde el punto de vista de implementaciones out-of-core en GPU existen dos tendencias: basada en particiones [12, 13] y basadas en memoria unificada [14–16]. En el primer caso, se particionan los datos en *chunks* que puedan caber en la GPU y se van procesando en ella iterativamente, mientras que en el segundo se utiliza un espacio de memoria accesible desde la CPU y la GPU de manera coherente, permitiendo la migración *on-demand* de datos mediante fallos de página. Para este trabajo se ha utilizado el primer acercamiento.

Una de las principales aportaciones de este trabajo es la utilización de un sistema multi-resolución *out-of-core* que permite procesar nubes de puntos de gran tamaño en equipos de baja gama. Además, se utiliza este sistema en el cálculo del mapa de calor en la GPU usando CUDA.

El trabajo se estructura de la siguiente manera. La Sección II introduce la herramienta en la que se incluirá el sistema desarrollado en este trabajo. En la Sección III se presenta el diseño e implementación del sistema multi-resolución *out-of-core* y su aplicación al cálculo del mapa de calor. En la Sección IV se muestran los resultados experimentales obtenidos. Finalmente, la Sección V presenta las principales conclusiones y líneas futuras de trabajo.

[1]Grupo de Arquitectura de Computadores (GAC), CITIC Universidad de A Coruña, e-mail: sergio.c.vidal@udc.es
[2]Grupo de Arquitectura de Computadores (GAC), CITIC Universidad de A Coruña, e-mail: carlos.vazquez.regueiro@udc.es
[3]Navantia S.A., S.M.E., e-mail: abrage@navantia.es
[4]Navantia S.A., S.M.E., e-mail: ddiazg@navantia.es
[5]Grupo de Arquitectura de Computadores (GAC), CITIC CEMI-UDC-Navantia Universidad de A Coruña, e-mail: margarita.amor@udc.es

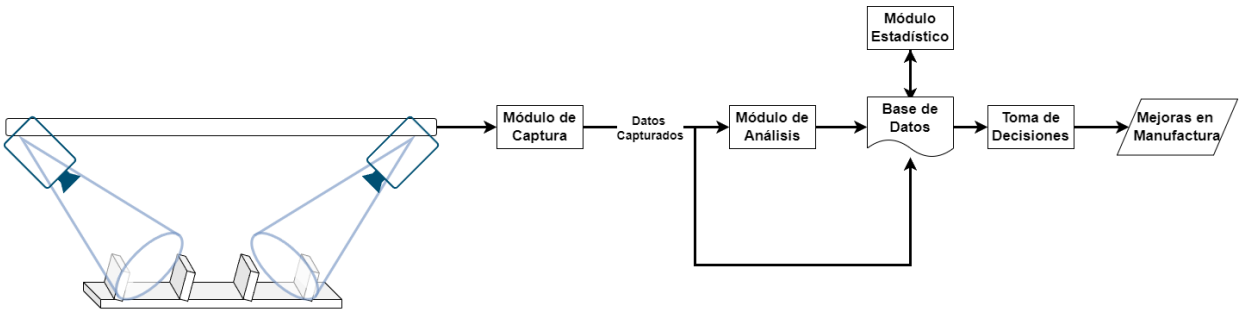


Fig. 1: Representación del sistema actual de captura y procesado

II. Herramienta de control dimensional

El objetivo de la herramienta de control dimensional es su inclusión en la línea de producción industrial unida a hardware especializado para la captura precisa de nubes de puntos que representen a los paneles y permitan su análisis. La integración entre el *hardware* y el *software* utilizado en la herramienta puede observarse en la Figura 1. Gracias a este análisis la herramienta podrá funcionar como fuente de datos para el sistema estadístico y junto a él formar parte de la toma de decisiones por parte de los operarios en la detección de defectos.

En este caso concreto el esfuerzo se ha centrado en un sistema piloto que se encuentra en el Taller de Elaborado de Navantia, donde se escanearán y analizarán paneles similares al mostrado en la Figura 2.

El sistema se diferencia en dos partes principales, hardware y software:

- Sistema hardware: Para la obtención de las nubes de puntos se utilizan dos cámaras 2D inclinadas 45 grados y enfrentadas para capturar las componentes x y z de los paneles. Los sensores se desplazan a lo largo del eje y para poder obtener las imágenes 3D de alta resolución. Esta forma de colocación de las cámaras se ha decidido para evitar las oclusiones debidas a la forma de los paneles.
- Sistema software: El sistema software se divide en dos partes; un sistema que funcionará como la herramienta de captura y análisis inicial de los datos y la herramienta estadística, que utilizará los datos proporcionados por el análisis para identificar posibles defectos e informar de los mismos al operario.

Los algoritmos estadísticos que se utilizarán tienen necesidades muy diferentes, tanto de precisión como de complejidad. En algunos, como la detección de fronteras [17] o la detección de filamentos [18], se puede llegar a tener una complejidad de $O(N^3)$ u $O(N^4)$ pero unas necesidades de precisión no muy elevadas, por lo que es posible usar nubes muestreadas para reducir el número de puntos y por lo tanto la carga computacional de los algoritmos. El sistema de análisis debe ser capaz de adaptar la resolución de la información que le pasa al sistema estadístico dependiendo de las necesidades de los algoritmos que se vayan a ejecutar.

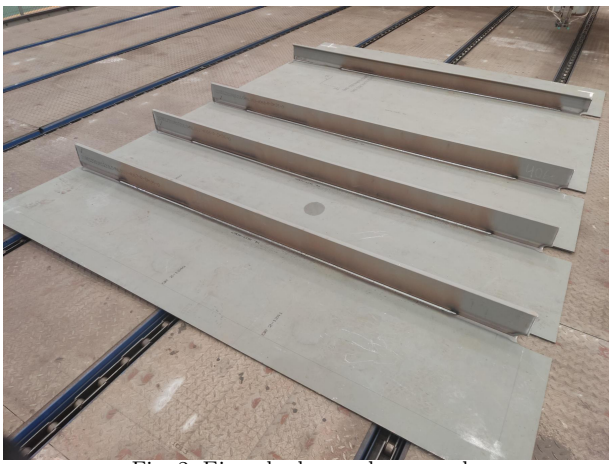


Fig. 2: Ejemplo de panel escaneado

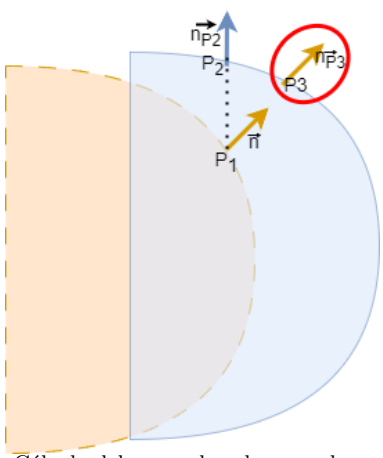


Fig. 3: Cálculo del mapa de calor usando normales

Una parte muy importante del análisis y que ha sido el foco de atención durante el desarrollo de este trabajo es el cálculo del mapa de calor. A partir del modelo en CAD original del panel y la nube de puntos obtenida de las cámaras se calcula una gradación en color para indicar las diferencias entre la captura y el modelo de forma visual. A partir de estos resultados se pueden deducir defectos o errores que hayan afectado a la estructura del panel y que hayan producido hendiduras o protuberancias que alejen a la pieza de su estructura ideal.

Hay dos formas de obtener el mapa de calor implementadas en la herramienta:

- **Cálculo básico del mapa de calor:** El cálculo básico realiza una computación *naive* de distancias de todos los puntos en la nube de entrada

frente a todos los triángulos, planos o centroides del modelo CAD correspondiente. A partir de las distancias se calcula cual es la menor y se utilizará como valor de intensidad en la representación final.

- **Cálculo del mapa de calor usando normales:** El cálculo mediante el uso de normales discrimina parte de los planos previamente mediante el cálculo del ángulo formado por las normales de los puntos y la normal del plano actual del modelo CAD considerado. Si el ángulo es superior a un límite dado el plano queda descartado lo cual reduce el número de planos al que compararlos y aumenta la precisión pues podría ocurrir que un plano esté más cerca de un punto pero que debido al ángulo no sea el que se debe utilizar para el cálculo de distancias pues no es su correlado. Un ejemplo de este caso podría verse en la Figura 3, en el que se puede observar un solapamiento entre el modelo CAD (en azul) y la nube de puntos tomada (en naranja). El punto P1 que se está analizando en la nube de puntos corresponde al punto P3, pero si se utilizase el cálculo básico, únicamente basado en la distancia, se detectaría el punto con la normal perpendicular(P2). Utilizando el ángulo que forman las normales de estos puntos se puede obtener el resultado correcto sin que la distancia sea el único factor de decisión. Si no se encontrase ningún plano con un ángulo adecuado pasaría a utilizarse el método básico.

Los objetivos de nuestra propuesta fueron:

- Solucionar los problemas de memoria tanto en el procesamiento en CPU como en GPU del mapa de calor.
- Ser capaz de proporcionar la información necesaria para el sistema estadístico con la precisión necesaria.

Estas razones son las que llevaron a la idea de desarrollar un sistema multi-resolución out-of-core, con el objetivo de ser capaz de mantener en varios niveles de detalle de la misma nube para distintos algoritmos y que solo se mantenga en memoria la parte de la nube que se está procesando en el momento concreto.

III. Sistema multi-resolución out-of-core

El sistema de multi-resolución out-of-core se ha diseñado teniendo en mente las necesidades especificadas en la anterior sección. La idea es usar una estructura *quadtree*, un árbol en la que cada nodo tiene 4 hijos, y en el que cada nodo contenga una nube que sea representativa de la formada por sus hijos. De esta manera, en cada una de las profundidades del árbol se encontrará la misma nube de puntos con un diferente nivel de detalle, siendo el máximo de profundidad la nube original. Una representación de esta estructura se puede observar en la Figura 4.

Un parámetro fundamental en la estructura es el

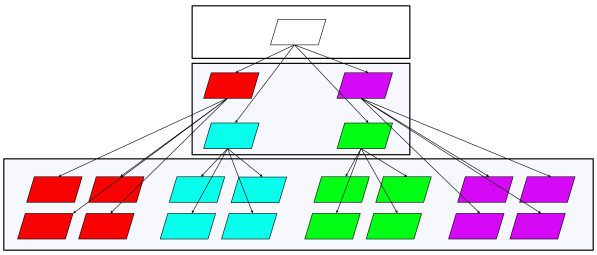


Fig. 4: Estructura *Quadtree* multi-resolución *out-of-core*

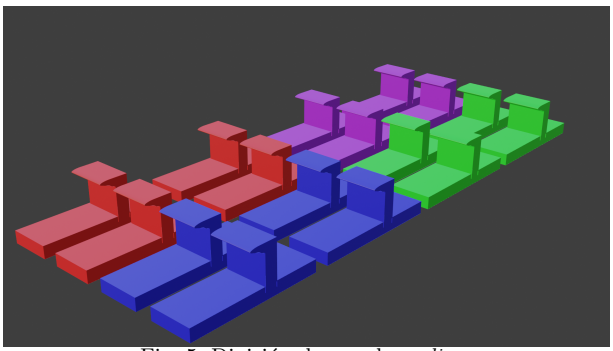


Fig. 5: División de panel en *slices*

número de puntos en cada uno de los *slices* (Ns). En este trabajo, centrándose en el tamaño mínimo adecuado para la visualización se han utilizado 10 millones de puntos, que almacenados ocupan alrededor de 200 MB y consumen alrededor de 2 GB de memoria durante el cálculo del mapa de calor.

Una vez obtenido este parámetro, se debe identificar en cuantos *slices* es necesario dividir la nube para poder cumplir este parámetro de la mejor forma posible. En este caso, al estar lidiando con paneles rectangulares podemos inferir que la división espacial funcionaría para dividir la nube de forma homogénea. Puede haber casos en los que se supere ligeramente el parámetro indicado, sobre todo en *slices* que contengan un refuerzo. Una representación de la división espacial se puede observar en la Figura 5. Podría ocurrir que un *slice* tuviese una mayor cantidad de puntos que otro si el primero contiene un refuerzo y el segundo no lo contiene, pero las pruebas no han mostrado ninguna diferencia significativa en este caso, ni en tiempo de ejecución de la operación completa ni en el consumo de memoria de la misma.

El siguiente paso es el proceso iterativo de creación de los niveles de profundidad superiores del árbol. Para ello, se toman los 4 nodos más cercanos y la nube resultante de su combinación se muestrea aleatoriamente hasta que se pueda mantener el tamaño máximo de nodo y continúa iterativamente hasta que se llegue a la raíz del árbol.

El método de muestreo no debe ser escogido a la ligera pues dependiendo de las necesidades del sistema puede ser necesario una precisión mínima o que los puntos tomados sean originales. El método fue decidido tras una serie de conversaciones con el equipo estadístico de la línea de Inspección y Control de Calidad del CEMI (Centro Mixto de Investigación[1]) que indicaron que el muestreo aleatorio sería

[1]https://cemi.udc.es/

suficiente siempre que se mantenga la estructura del panel. Por lo tanto, el método de muestreo escogido es el muestreo estratificado o *Stratified Sampling*. Para conseguir esto se tomará un porcentaje de puntos de cada *slice* relativo a su tamaño con respecto a la suma de todos [19]. La idea es intentar evitar el caso extremo de que el muestreo aleatorio escoja una cantidad demasiado elevada de puntos de un solo *slice* y que la nube resultante no sea representativa de todos los nodos hijos. De esta manera y teniendo en cuenta que los puntos estarán distribuidos de forma uniforme en el panel se puede obtener un nivel de detalle inferior pero representativo de la nube original.

A. Cálculo del mapa de calor en out-of-core

Una vez se ha transformado la nube original en un *quadtree* multi-resolución se pueden aplicar operaciones específicas a estas estructuras que puedan paliar el consumo de memoria gracias a la subdivisión de la nube original. En este trabajo, el foco ha sido para el cálculo del mapa de calor, para el cual se ha utilizado el máximo nivel de detalle disponible.

Con el fin de restringir el uso de memoria al mínimo el objetivo es mantener únicamente un nodo en memoria al mismo tiempo. Esto es posible gracias a que el mapa de calor es una operación punto a punto entre la nube y el modelo CAD original de la misma. El proceso comienza con la obtención de todos los nodos en la profundidad máxima del árbol multi-resolución y se comienza a realizar un proceso iterativo sobre los mismos.

Para el procedimiento se carga la nube correspondiente al nodo en memoria y se comienza a iterar sobre sus puntos calculando las distancias respectivas a cada punto. Esta iteración sobre los puntos puede ser en CPU, con la ayuda de OpenMP para paralelizar el proceso, o en GPU, utilizando CUDA para reducir considerablemente el tiempo de ejecución. Al finalizar la ejecución se presentan dos opciones para lidiar con la salida, la salida estándar o la salida *out-of-core*.

El uso de la salida estándar almacenará las nubes de puntos resultantes en memoria principal, lo cual, teniendo en cuenta que la mayor parte del coste del algoritmo en memoria viene del almacenamiento de resultados podría causar los mismos problemas que se han intentado evitar con la implementación de este sistema, por lo que se recomienda la siguiente posibilidad.

La salida out-of-core irá generando los nodos de la profundidad máxima del árbol de resultados conforme se va iterando por los nodos de entrada. De esta forma, tomando como ejemplo el cálculo del mapa de calor en un momento dado el máximo de memoria que se puede consumir en una ejecución es el tamaño máximo de *slice* multiplicado por tres, la entrada, los puntos en que se ha conseguido obtener una distancia aceptable y los puntos del CAD correspondientes a cada una de las entradas, sumado a las normales de estos últimos que se han ido calculando, que actualmente se almacenan de forma completa en disco.

Tabla I: Nubes de puntos utilizadas en las pruebas del procedimiento

Nube	Tamaño	Ancho (mm)	Alto (mm)	Nº de puntos (millones)
Panel 1	195 MB	600	600	>9.9
Panel 2	1.56 GB	4475	2700	>119

Tras finalizar todos los nodos de entrada, se aplicará el proceso de creación de árbol a las salidas del procedimiento para poder realizar una visualización de las mismas que no consuma una gran cantidad de memoria.

IV. Resultados experimentales

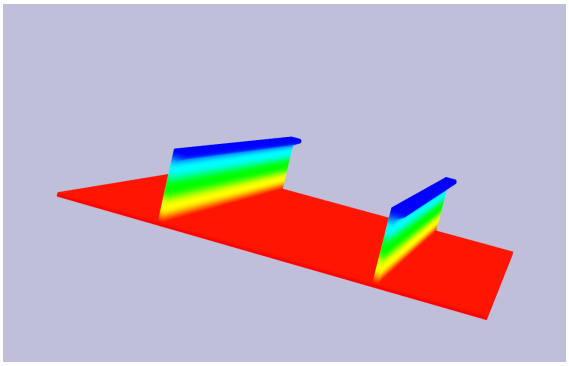
Para realizar las pruebas de rendimiento se han utilizado los paneles descritos en la Tabla I. La primera nube representa un pequeño panel de prueba que no supone una gran carga (ver Figura 6a). Mientras que la segunda nube es obtenida de escanear un panel real y contiene una gran densidad de puntos (ver Figura 6b). Utilizando estos dos tipos de nubes se espera obtener un resultado representativo de las capacidades del sistema en un entorno real. Las pruebas se han realizado en un equipo HP ZBook Studio x360 G5, que dispone de un procesador Intel Core i7-9750H con 6 cores a 2.60 GHz, 16 GB de RAM y una tarjeta gráfica NVIDIA Quadro P1000 con 4 GB de memoria. El sistema operativo utilizado es Windows 10 y la versión de CUDA es la 11.6.

Para la realización de las pruebas se ha utilizado el cálculo del mapa de calor pues es una parte clave del proceso y sus resultados son vitales para realizar la toma de decisiones, por lo que su eficiencia ha sido considerada una prioridad. Para ello anteriormente en el proyecto se desarrolló una implementación mediante CUDA que era capaz de realizar el procedimiento básico del mapa de calor. Como parte de este trabajo se ha desarrollado una implementación básica utilizando OpenMP para el procedimiento *out-of-core* y ha sido adaptada para *in-core* a efectos de comparación. Además, se ha extendido esta implementación al cálculo usando normales, tanto en CPU como en GPU.

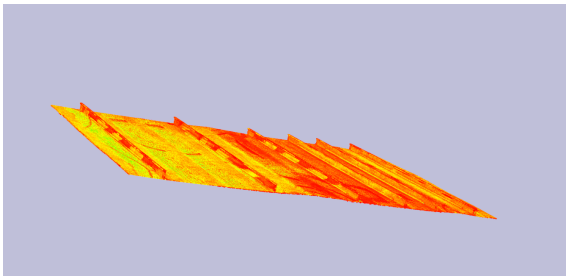
A. Cálculo de mapa de calor básico

La Tabla II muestra los resultados obtenidos en el cálculo básico del mapa de calor.

El algoritmo realiza las comparaciones con los triángulos que forman el modelo original utilizando un método punto a punto, por lo que en este caso el algoritmo se considera vergonzosamente paralelo. En los resultados se pueden observar los problemas principales del cálculo del mapa de calor, su coste en tiempo y su coste en memoria. El coste en tiempo es obvio, pues el cálculo en un panel de 1.6 GB tarda 2 h en procesarse, algo que es un tiempo demasiado elevado para la toma de decisiones en el taller. El siguiente problema, el coste en memoria, es el obstáculo que se ha intentado subsanar en este trabajo con el sistema *out-of-core*. La herramienta estará instalada en el Taller de Elaborado en equipos portátiles con recursos limitados, por lo que la necesidad de 16 GB de memoria RAM no permite procesar nubes de



(a) Panel 1



(b) Panel 2

Fig. 6: Paneles utilizados durante las pruebas

Para

Tabla II: Resultados del cálculo básico del mapa de calor

		CPU (1 hilo)	CPU (6 hilos)		CUDA	
Nube	**Memoria (GB)**	**Tiempo (s)**	**Tiempo (s)**	**SpeedUp**	**Tiempo (s)**	**SpeedUp**
Panel 1	2.1	996	548	1.82x	6.5	84.3x
Panel 2	16	13324	6736	1.97x	-	-

Tabla III: Resultados del cálculo básico del mapa de calor en *out-of-core*

		CPU (6 hilos)		CUDA	
Nube	**Memoria (GB)**	**Tiempo (s)**	**SpeedUp**	**Tiempo (s)**	**SpeedUp**
Panel 1	2	630	0.87	8.5	74.12x
Panel 2	5	6830	0.98x	223.21	30.59x

puntos obtenidas de paneles reales.

Para solucionar el problema del tiempo de ejecución de la aplicación se realizó una implementación en CUDA del cálculo del mapa de calor. Se pudo reducir en gran medida el tiempo que consume el cálculo para el primer panel, obteniendo una aceleración de 84x. El problema de memoria ocurre al procesar nubes de paneles reales, que al tener un mayor tamaño y por tanto requisitos mayores de memoria superan la memoria disponible de la GPU de un portátil.

Los resultados de la implementación *out-of-core* pueden observarse en la Tabla III. El Panel 1 debido al tamaño máximo de nodo no muestra mejoras claras, pues el número de puntos del Panel 1, como se indica en la Tabla I es menor que 10 millones. Debido a esto se observa que solo hay leves cambios en el tiempo consumido, aumentándolo debido a la necesidad de obtener las nubes de disco duro, y ningún cambio en la memoria.

Esto cambia en el Panel 2, donde sí que se observa una gran reducción de la memoria consumida, siendo la indicada en las tablas el máximo a lo largo del proceso. Se ha reducido la memoria consumida por el cálculo en CPU en 11 GB, lo cual permitiría que la herramienta realizase el cálculo del mapa de calor en todo tipo de equipos sin importar sus capacidades de memoria.

Respecto al cálculo en GPU, los resultados del Panel 1 no han sufrido grandes cambios pues se realiza de la misma forma que en la implementación anterior debido a su tamaño. Sin embargo, los resultados del Panel 2 muestran que la nube resultante puede procesarse en la GPU sin problemas, a diferencia de la operación sin utilizar la estructura (ver Tabla II), en la que la memoria consumida era tan elevada que era imposible de procesar. El coste de este cambio es una subida del tiempo de GPU ideal pues el mismo *kernel* se está creando y ejecutando varias veces para evitar llenar la memoria. Aún con este aumento de las comunicaciones, se consigue acelerar 30x la ejecución *out-of-core* sin ningún problema debido a la memoria limitada de la GPU.

B. Cálculo de mapa de calor avanzado

Se ha realizado la implementación del cálculo del mapa de calor utilizando normales en CUDA y se ha adaptado la misma para poder utilizar el sistema multi-resolución desarrollado.

Los resultados de la implementación *in-core* pueden observarse en la Tabla IV.

Estos resultados muestran que esta forma de obtener el mapa de calor es aún más costosa que el cálculo básico estudiado anteriormente, consumiendo aproximadamente el doble de tiempo en el Panel 1 y en el Panel 2, aunque la diferencia en la memoria consumida no es tan elevada. Esto es debido a la necesidad de discriminar los planos según su ángulo antes del procesamiento de cada punto lo cual aumenta el trabajo necesario por punto y hace que el problema sea más complejo.

Se ha implementado, por lo tanto, una versión del problema en CUDA, que consigue una aceleración de 156x respecto a la versión multi-hilo con su versión *in-core* en el Panel 1. Esta implementación tiene mayores limitaciones en tamaño de la nube de entrada a las encontradas en el cálculo básico debido a la necesidad de transferir las normales y las ecuaciones de los planos hacia la GPU, a diferencia del anterior. Debido a esto, se debe utilizar el sistema *out-of-core* para el procesamiento de nubes de puntos de gran tamaño en CUDA, cuyos resultados se pueden observar en la Tabla V.

A diferencia del cálculo básico, la solución *out-of-core* consigue en este caso mejores resultados de tiempo que la versión *in-core*, con una aceleración de 1.07x, seguramente debido a la mayor necesidad de memoria que necesita este cálculo y que, por lo tanto, requiere una mayor virtualización de memoria en la versión *in-core*. Al mantener solo un *slice* en memoria en cada momento, se elimina esta necesidad de virtualización y la mejora obtenida es capaz

Tabla IV: Resultados del cálculo con normales del mapa de calor

		CPU (1 hilo)	CPU (6 hilos)		CUDA	
Nube	**Memoria (GB)**	**Tiempo (s)**	**Tiempo (s)**	**SpeedUp**	**Tiempo (s)**	**SpeedUp**
Panel 1	2.1	1869	1064	1.76x	6.8	156.47x
Panel 2	16	30318	14232	2.13x	-	-

Tabla V: Resultados del cálculo con normales del mapa de calor en *out-of-core*

		CPU (6 hilos)		CUDA	
Nube	**Memoria (GB)**	**Tiempo (s)**	**SpeedUp**	**Tiempo (s)**	**SpeedUp**
Panel 1	2.6	1229	0.86x	9.2	133.58x
Panel 2	6	13193	1.07x	332.51	39.74x

de superar el coste de las transferencias desde disco duro.

Finalmente, la implementación *out-of-core* en GPU permite el procesamiento para nubes de cualquier tamaño y obtiene una aceleración de 39.74x respecto a la versión multihilo *out-of-core*. Gracias a esta implementación, se ha conseguido reducir el tiempo de cálculo del mapa de calor desde más de 3 horas hasta 5 minutos.

V. Conclusiones

Se ha desarrollado un sistema multi-resolución *out-of-core* que permite el procesado de distintos niveles de detalle de una misma nube de puntos sin importar su tamaño y ha sido integrada en el proceso de control de calidad de paneles para reducir considerablemente la memoria consumida por el procesamiento de nubes masivas.

Se ha adaptado el cálculo del mapa de calor tanto en CPU como en GPU para que con la estructura desarrollado sea ejecutable en equipos y GPUs con grandes limitaciones de memoria sin realizar la virtualización. Se ha demostrado su posible uso en GPUs de baja gama, como las de un portátil, lo que permite reducir el tiempo de ejecución y conseguir aceleraciones de alrededor de 31x respecto a su versión multi-hilo.

Se ha completado la implementación en GPU del cálculo del mapa de calor utilizando las normales en GPU, tanto *in-core* como *out-of-core*, obteniendo una aceleración de 133x en el mejor de los casos y acelerando incluso la versión multi-hilo en CPU debido a una mejor gestión de la memoria.

Como trabajo futuro se considera la posibilidad de implementar más operaciones utilizando la estructura desarrollada para el análisis de nubes de puntos, sobre todo aquellas con altos costes de memoria como la extracción de filamentos o fronteras.

Agradecimientos

El presente trabajo ha sido financiado por el Centro Mixto de Investigación Navantia-UDC (IN853C 2022/01), financiado por GAIN (Xunta de Galicia) y ERDF Galicia 2021-2027; y apoyado por el Ministerio de Ciencia e Innovación de España (PID2019-104184RB-I00), por el Gobierno de Galicia y por fondos FEDER bajo el Programa de Consolidación de Unidades Competitivas de Investigación (ref. ED431C 2021/30). Todos los recursos industriales utilizadas en este artículo, como los datos de paneles navales, han sido provistos por Navantia.

Referencias

[1] Majid Mirbod, Ali Ghatari, Saber Saati, and Maryam Shoar, "Industrial parts change recognition model using machine vision, image processing in the framework of industrial information integration," *Journal of Industrial Information Integration*, vol. 26, pp. 100277, 09 2021.

[2] Carlos A. Escobar and Rubén Morales-Menéndez, "Machine learning techniques for quality control in high conformance manufacturing environment," *Advances in Mechanical Engineering*, vol. 10, no. 2, pp. 1687814018755519, 2018.

[3] Longfei Zhou, Lin Zhang, and Nicholas Konz, "Computer vision techniques in manufacturing," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 53, no. 1, pp. 105–117, 2023.

[4] Yu He, Kechen Song, Qinggang Meng, and Yunhui Yan, "An end-to-end steel surface defect detection approach via fusing multiple hierarchical features," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 4, pp. 1493–1504, 2020.

[5] Yulong Zong, Jin Liang, Huan Wang, Maodong Ren, Mingkai Zhang, Wenpan Li, Wang Lu, and Meitu Ye, "An intelligent and automated 3D surface defect detection system for quantitative 3D estimation and feature classification of material surface defects," *Optics and Lasers in Engineering*, vol. 144, pp. 106633, 2021.

[6] Margarita Amor, David Deibe, and Álvaro Brage, "HePnP: Herramienta para el procesamiento de nubes de puntos," March 2022, Número de asiento registral: 03/2022/764.

[7] Shuai Yuan, Shuai Zhu, Dong-Shuang Li, Wen Luo, Zhao-Yuan Yu, and Lin-Wang Yuan, "Feature preserving multiresolution subdivision and simplification of point clouds: A conformal geometric algebra approach," *Mathematical Methods in the Applied Sciences*, vol. 41, no. 11, pp. 4074–4087, 2018.

[8] David Deibe, Margarita Amor, and Ramón Doallo, "Supporting multi-resolution out-of-core rendering of massive lidar point clouds through non-redundant data structures," *International Journal of Geographical Information Science*, vol. 33, no. 3, pp. 593–617, 2019.

[9] Diego Teijeiro, Margarita Amor, Ramón Doallo, and David Deibe, "Interactive Visualization of Large Point Clouds Using an Autotuning Multiresolution Out-Of-Core Strategy," *The Computer Journal*, 2022, bxac179.

[10] Rémi Cura, Julien Perret, and Nicolas Paparoditis, "Implicit LOD for processing, visualisation and classification in point cloud servers," *ArXiv, abs/1602.06920*, 2016.

[11] Linlin Ge and Jieqing Feng, "Out-of-core outlier removal for large-scale indoor point clouds," *Graphical Models*, vol. 122, pp. 101142, 2022.

[12] Wei Han, Daniel Mawhirter, Bo Wu, and Matthew Buland, "Graphie: Large-scale asynchronous graph traversals on just a GPU," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017, pp. 233–245.

[13] Kang-Wook Chon, Eunjeong Yi, and Min-Soo Kim, "Sgminer: A fast and scalable GPU-based frequent pattern miner on SSDs," *IEEE Access*, vol. 10, pp. 62502–62519, 2022.

[14] Design Guide, "Cuda C programming guide," 2013, NVIDIA *July 23* (2013),31.

[15] Dan Negrut, Radu Serban, Ang Li, and Andrew Seidl, "Unified Memory in CUDA 6: A Brief Overview and Related Data Access/Transfer Issues," 08 2018.

[16] Yang Xia, Peng Jiang, Gagan Agrawal, and Rajiv Ramnath, "End-to-end lu factorization of large matrices on gpus," in *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, New York, NY, USA, 2023, PPoPP '23, p. 288–300, Association for Computing Machinery.

[17] Alberto Rodríguez-Casal and Paula Saavedra-Nieves, "A fully data-driven method for estimating the shape of a point cloud," *ESAIM: PS*, vol. 20, pp. 332–348, 2016.

[18] Seppo Pulkkinen, "Ridge-based method for finding curvilinear structures from noisy data," *Computational Statistics & Data Analysis*, vol. 82, pp. 89–109, 2015.

[19] Mohammad Sultan Mahmud, Joshua Zhexue Huang, Salman Salloum, Tamer Z. Emara, and Kuanishbay Sadatdiynov, "A survey of data partitioning and sampling methods to support big data analysis," *Big Data Mining and Analytics*, vol. 3, no. 2, pp. 85–101, 2020.

Ajuste automático de modelos gEUD para planificación en IMRT

J.J. Moreno, S. Puertas-Martín, J.L. Redondo, P.M. Ortigosa, E.M. Garzón[1],

Resumen— **La Radioterapia de Intensidad Modulada (IMRT) es un tratamiento contra el cáncer que aplica altas dosis de radiación de forma precisa en el cuerpo del paciente. Los modelos basados en la Dosis Uniforme Equivalente Generalizada (gEUD) proporcionan planes de irradiación con excelente cobertura dosimétrica de las zonas tumorales y gran protección de los tejidos sanos. Sin embargo, estos modelos requieren un extenso conjunto de parámetros de difícil ajuste. En este trabajo proponemos una estrategia de optimización de dos capas: En la capa inferior optimizamos un modelo gEUD con un algoritmo de descenso por gradiente. En la capa superior ajustamos los parámetros de ese modelo gEUD, utilizando una metaheurística basada en un algoritmo genético multiobjetivo. Para evaluar la metodología propuesta se ha utilizado un caso real de un paciente con un cáncer de Cabeza y Cuello (H&N). Nuestra implementación ha sido capaz de proporcionar, de forma autónoma, un conjunto de planes alternativos y de compromiso, que han sido posteriormente validados independientemente como clínicamente aceptables. Esta estrategia podría facilitar el trabajo de los radiofísicos, proporcionándoles alternativas no exploradas y reduciendo el tiempo requerido para proveer un plan de calidad a cada paciente.**

Palabras clave— **IMRT, Algoritmos genéticos, Optimización multiobjetivo**

I. Introducción

DURANTE la preparación de un plan de Radioterapia, nos encontramos dos objetivos: Por una parte, depositar la dosis prescrita en los tejidos tumorales. Por otra, preservar en la medida de lo posible al paciente. En la mayoría de los casos, estos dos objetivos son contradictorios, pues los tumores suelen encontrarse en zonas de difícil acceso, rodeados de órganos y tejidos sanos. Consecuentemente, la generación de planes de calidad es un problema multiobjetivo [1], [2], [3], en el que existe un equilibrio entre cobertura dosimétrica tumoral y supervivencia de los órganos en riesgo.

Para los problemas de optimización basados en la Dosis Uniforme Equivalente Generalizada (gEUD), las soluciones óptimas pueden ser calculadas eficientemente con métodos de gradiente [4]. Sin embargo, es necesario que un planificador experto defina un conjunto de parámetros para cada Volumen Blanco de Planificación (PTV) y Órgano de Riesgo (OAR). Normalmente, el planificador comienza con un conjunto de parámetros recomendados en la literatura, los cuales ajusta para cada paciente en un proceso de prueba y error, con el objetivo de satisfacer las prescripciones de cada paciente. La calidad del plan final depende por lo tanto en gran medida del conocimiento y las habilidades del planificador.

[1]Dpto. de Informática, Universidad de Almería, e-mail: juanjomoreno,savinspm,jlredondo,ortigosa,gmartin@ual.es

A pesar de las diferencias entre los criterios físicos y biológicos, ambos son utilizados concurrentemente en la práctica clínica. En este trabajo proponemos combinar estos dos tipos de criterios en un esquema común. Nuestro objetivo es generar, de forma automática y para cada paciente, un conjunto de planes con diferentes prioridades, que puedan utilizarse como puntos de partida de calidad para acelerar y facilitar el trabajo de los planificadores.

II. Modelo de optimización gEUD para IMRT

la Dosis Uniforme Equivalente Generalizada (gEUD) es un criterio biológico que estima los efectos de la radiación, indicándonos cuál es la dosis uniforme de radiación que causa el mismo efecto que la dosis no-uniforme actual [5], [6].

En el caso de gEUD, los efectos de la radiación en cualquier Región de Interés (ROI), referida en la fórmula como estructura s, se pueden evaluar de la siguiente manera:

$$gEUD_s(x, a_s) = \left(\frac{1}{|M_s|} \sum_{j \in M_s} d_j(x)^{a_s} \right)^{\frac{1}{a_s}} \quad (1)$$

donde $|M_s|$ es el número de voxels de la estructura s, $d_j(x) = D_j x$ es la dosis de radiación depositada en el voxel j de la estructura s por el mapa de fluencia x y el parámetro a_s representa el efecto de la radiación en la estructura. Normalmente se obtienen valores iniciales de este parámetro en la literatura, los cuales suelen ser después ajustados experimentalmente para cada paciente.

Siguiendo [6], podemos obtener planes clínicamente aceptables calculando el máximo de la función $F(x, \phi)$, construida a partir de la función gEUD de la siguiente forma:

$$F(x, \phi) = \begin{array}{l} \prod_{t \in T} \frac{1}{1 + \left(\frac{EUD_t^0}{gEUD_t(x, a_t)} \right)^{n_t}} \cdot \\ \prod_{r \in R} \frac{1}{1 + \left(\frac{gEUD_r(x, a_r)}{EUD_r^0} \right)^{n_r}} \end{array} \quad (2)$$

donde EUD_t^0 es la dosis prescrita para el PTV con índice t, EUD_r^0 es la dosis uniforme máxima en la región de interés con índice r, n_r y n_t expresan la importancia de las prescripciones en las correspondientes estructuras y ϕ representa el conjunto de parámetros que definen F, es decir, ϕ es una instancia de parámetros $n_t, n_r, a_t, a_r, EUD_r^0$ donde $t \in T, r \in R$.

Con estas definiciones, proponemos un esquema en el que, en cada iteración:

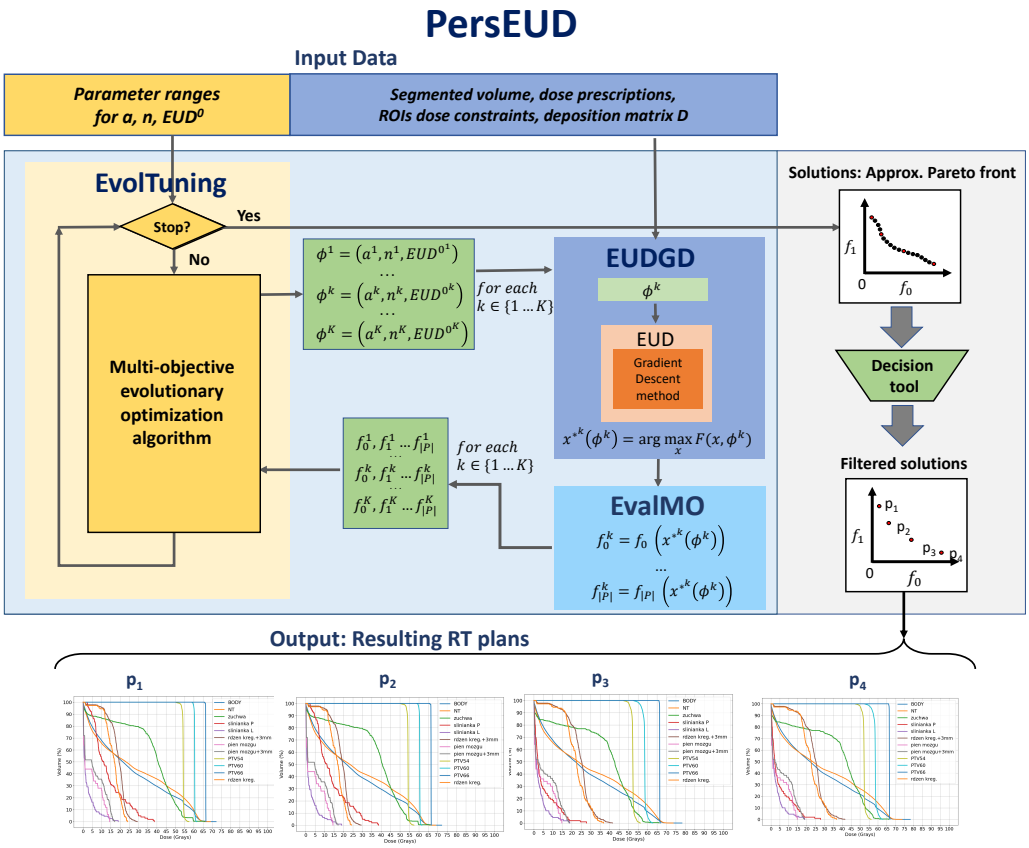


Fig. 1: Diagrama de flujo de PersEUD.

1. Cada colección de parámetros gEUD ϕ es evaluada con respecto a los valores de los $|P+1|$ criterios $f_0(x^*(\phi)), \ldots, f_{|P|}(x^*(\phi))$, los cuales son calculados tras el descenso por gradiente.
2. Un algoritmo genético de optimización multi-objetivo navega el espacio Φ en busca de colecciones ϕ, $\phi \in \Phi$ tal que las tuplas $f_0(x^*(\phi)), \ldots, f_{|P|}(x^*(\phi))$ sean no-dominadas.

El proceso acaba cuando la diferencia entre las tuplas en iteraciones sucesivas es insignificante. El resultado es una colección de tuplas $f_0(x^*(\phi)), \ldots, f_{|P|}(x^*(\phi))$ que constituyen una aproximación del frente de Pareto.

A partir de este esquema, hemos desarrollado e implementado un sistema, llamado PersEUD, cuyo esquema de funcionamiento se representa en la Figura 1. El módulo genético multi-objetivo EvolTuning produce K colecciones de parámetros ϕ, mutuamente no-dominados en términos de las funciones $f_0, f_1, \ldots, f_{|P|}$. Por cada ϕ^k, $k \in K$, el módulo de descenso por gradiente EUDGD busca el $x^*(\phi^k)$ que maximiza la función (2) y el módulo EvalMO utiliza la deposición de dosis resultante para evaluar las funciones $f_0, f_1, \ldots, f_{|P|}$. Estas evaluaciones vuelven al módulo EvolTuning, que las utiliza para direccionar la búsqueda de la nueva generación de parámetros ϕ.

Para comunicarnos entre los diferentes módulos, hemos implementado un enrutador de mensajes basado en la libería ZeroMQ [7].

El módulo EvolTuning está implementado en Java y utiliza el framework jMetal [8]. Este framework proporciona implementaciones de calidad de diferentes algoritmos metaheurísticos. Aunque en este trabajo hemos utilizado el algoritmo MOEA/D, este sistema de módulos permite intercambiar el algoritmo evolutivo con mínimos cambios.

Para los módulos EUDGD y EvalMO hemos desarrollado dos versiones: Una versión en CUDA [9] para GPUs NVIDIA y una versión OpenMP [10] para CPUs multinúcleo. Estas dos versiones pueden utilizarse indistintamente o incluso simultáneamente, gracias a que el enrutador reparte la carga entre todas las instancias de EUDGD disponibles. Así, podemos explotar totalmente todos los recursos disponibles de una plataforma.

III. Resultados experimentales

Este esquema ha sido probado en varios casos de pacientes reales, en este trabajo mostraremos uno de los más interesantes. En este caso, nos centramos en reducir la dosis depositada tanto en las glándulas salivares como en la médula espinal, definiendo un problema tri-objetivo de compromiso.

Para la experimentación, hemos usado un caso real de un paciente de Cabeza y Cuello, comúnmente llamados en la comunidad H&N por sus siglas en inglés, que ha sido tratado con nueve haces (beams) de radiación. En este caso, se han delineado tres PTVs con diferentes prescripciones de dosis (66 Gy, 60 Gy y 54 Gy). El más importante de los PTVs es el de mayor dosis prescrita (66 Gy), pues la mayoría de células clonogénicas se encuentran en esa zona. Los otros dos PTVs son tratados profilácticamente, por lo que su cobertura dosimétrica no es tan crítica. Adicio-

Tabla I: Restricciones de dosis prescritas por la radiofísica para cada región.

	Restricciones de dosis			
Región de Interés	LB	$\overline{LB}$	$\overline{UB}$	UB
Tejido Normal	-	-	-	74.25
Mandíbula	-	-	-	70.00
Glándula Salival D.	-	-	26.00	-
Glándula Salival I.	-	-	26.00	-
Médula Espinal +3mm	-	-	-	50.00
Tronco Encefálico +3mm	-	-	-	60.00
PTV 54	48.60	52.92	55.08	59.40
PTV 60	54.00	58.80	61.20	66.00
PTV 66	59.40	64.67	67.32	72.60

Tabla II: Parámetros gEUD del plan. Negro: Parámetros fijos. Azul: Rangos de EvolTuning. Verde: Parámetros optimizados.

	Parámetros gEUD		
Región de Interés	EUD_s^0	a_s	n_s
Tejido Normal	74.25	40.00	5.00
Mandíbula	70.00	10.00	5.00
Glándula Salival D.	$[0{,}5, 26]$: 4,76	$[1, 100]$: 1,01	$[1, 100]$: 18,25
Glándula Salival I.	$[0{,}5, 26]$: 3,79	$[1, 100]$: 1,31	$[1, 100]$: 11,17
Médula Espinal +3mm	$[0{,}5, 50]$: 1,80	$[1, 50]$: 1,33	$[1, 100]$: 12,79
Tronco Encefálico +3mm	60.00	10.00	5.00
PTV 54	54.00	$[-100, -1]$: −65,55	$[1, 100]$: 54,11
PTV 60	60.00	$[-100, -1]$: −87,12	$[1, 100]$: 57,66
PTV 66	66.00	$[-100, -1]$: −33,27	$[1, 100]$: 18,62

nalmente, se han delineado cinco Órganos en Riesgo (OARs): Médula Espinal +3mm, Tronco Encefálico +3mm, Mandíbula y Glándula Salival Izquierda y Derecha. Finalmente, se ha definido una estructura llamada "Tejido Normal"que contiene todo el tejido saludable del paciente, excluyendo los OARs anteriormente definidos. Con esta distribución, el modelo de deposición de dosis contiene 30265 beamlets. El cuerpo del paciente se representa con un volumen tomográfico obtenido por CT de 512x512x107, el cual es voxelizado y segmentado en 94647 voxels. La Tabla I muestra las prescripciones y restricciones para cada ROI del paciente y la Tabla II muestra los parámetros optimizados por PersEUD para el plan representado.

La Figura 2 muestra la dosis depositada en dos cortes del cuerpo del paciente. En general, la dosis media en las glándulas salivares es 13.54 Gy (siendo el máximo permitido 26 Gy), la dosis máxima en la espina dorsal +3mm 24.06 Gy (permitido 50 Gy). Todos los demás órganos en riesgo cumplen sus restricciones de dosis, mientras que los PTVs reciben la cobertura dosimétrica suficiente para que el plan sea clínicamente aceptable.

Para la validación de este plan, lo hemos comparado con un plan diseñado manualmente por una radiofísica para el mismo paciente. La Figura 3 muestra los histogramas dosis-volumen del plan generado automáticamente (P3, líneas continuas) y manualmente (R2, líneas discontinuas). En esta comparativa se puede observar que la radiofísica obtiene mejor cobertura dosimétrica en los PTVs, a costa de depositar mayor dosis en los órganos de riesgo, especialmente en los tres órganos (las dos glándulas salivares y la médula espinal) en los que nos hemos centrado.

IV. Conclusiones

Los modelos basados en gEUD tienen el potencial de crear mejores planes que los basados en criterios físicos, pues consideran el efecto biológico de la radiación en las células. Sin embargo, el uso de criterios biológicos en las prácticas clínicas actuales es obstaculizado por la dificultad de la conversión entre los parámetros de entrada y los planes resultantes. Para mejorar esta situación, hemos propuesto un sistema de ajuste automático de parámetros basado en métodos de optimización.

La característica principal del sistema propuesto es que consiste en dos capas de optimización: En la capa superior, un algoritmo genético de optimización multiobjetivo trabaja en el espacio de parámetros del modelo gEUD. En la capa inferior, un algoritmo exacto de descenso por gradiente optimiza el plan dado un conjunto de parámetros gEUD.

Los resultados experimentales muestran la viabilidad del sistema propuesto: Los planes resultantes cumplen las prescripciones dadas por los profesionales médicos y son competitivos en calidad si los comparamos con los obtenidos por sistemas de planificación comerciales. El uso de estos sistemas puede ayudar a mejorar los tratamientos, proporcionar una segunda opinión a los profesionales médicos y reducir el tiempo necesario para la planificación manual.

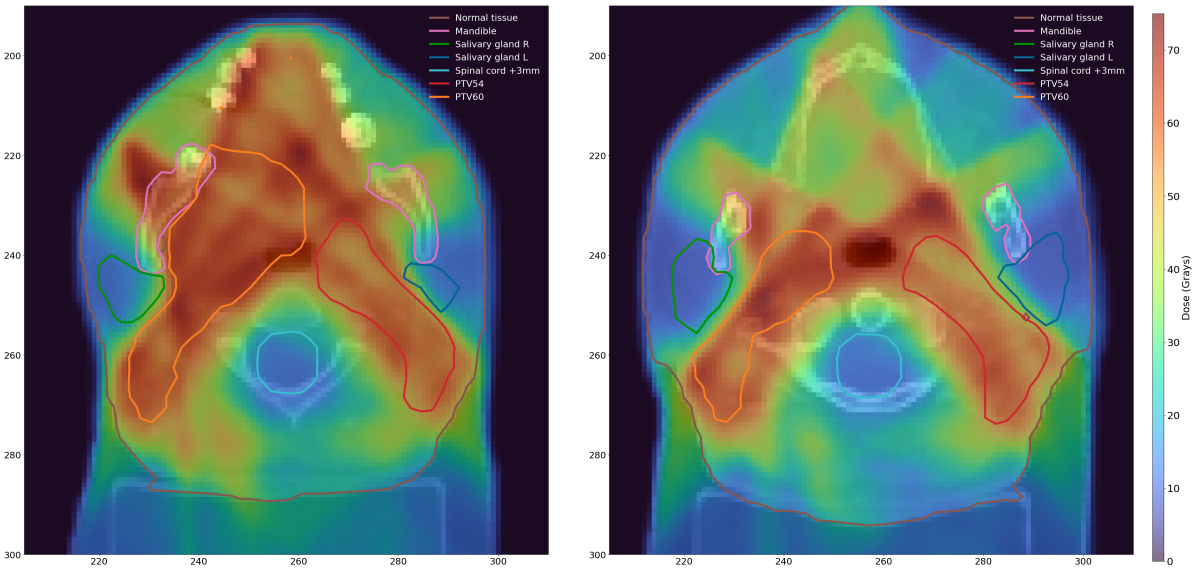


Fig. 2: Dosis depositada en $Z = 75$ (izquierda) y $Z = 91$ (derecha).

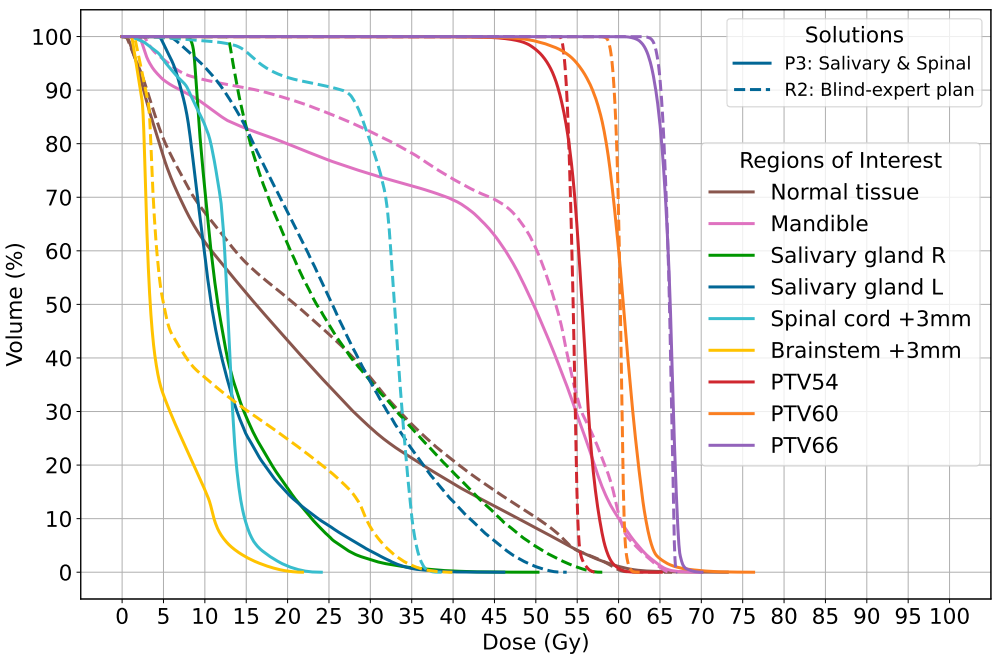


Fig. 3: DVH del plan automático (P3) comparado con el plan obtenido por el experto (R2).

Agradecimientos

Esta publicación es parte del la proyecto de I+D+i PID2021-123278OB-I00, TED2021-132020B-I00 financiado por MCIN/ AEI/10.13039/501100011033/ y por "FEDER Una manera de hacer Europa".

Referencias

[1] S. Breedveld, D. Craft, R. van Haveren, and B. Heijmen. Multi-criteria optimization and decision-making in radiotherapy. *European Journal of Operational Research*, 277(1):1 – 19, 2019.

[2] B. Cho. Intensity-modulated radiation therapy: a review with a physics perspective. *Radiation Oncology Journal*, 36(1):1–10, 2018.

[3] M. Ehrgott, C. Guler, H.W. Hamacher, and L. Shao. Mathematical optimization in intensity modulated radiation therapy. *Annals of Operations Research*, 175(1):309–365, 2010.

[4] B. Choi and J. O. Deasy. The generalized equivalent uniform dose function as a basis for intensity-modulated treatment planning. *Physics in Medicine and Biology*, 47(20):3579–3589, oct 2002.

[5] A Niemierko. Reporting and analyzing dose distributions: a concept of equivalent uniform dose. *Medical physics*, 24(1):103–110, 1996.

[6] A. Niemierko Q. Wu, R. Mohan and R. Schmidt-Ullrich. Optimization of intensity-modulated radiotherapy plans based on the equivalent uniform dose. *Int. J. Radiation Oncology Biol. Phys.*, 52(1):224–235, 2002.

[7] ZeroMQ. Ømq - the guide. `https://zguide.zeromq.org/`, 2022. Last Accessed on 2022/08/09.

[8] J. J. Durillo and A. J. Nebro. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[9] NVIDIA. Cuda toolkit documentation v11.7.1. `https://docs.nvidia.com/cuda/index.html`, 2022. Last Accessed on 2022/08/09.

[10] OpenMP. The openmp api specification for parallel programming. `https://www.openmp.org/`, 2022. Last Accessed on 2022/08/09.

[11] M. Ehrgott, C. Guler, H. W. Hamacher, and L. Shao. Mathematical optimization in intensity modulated radiation therapy. *Annals of Operations Research*, 175(1):309–365, Mar 2010.

GPU-accelerated Spatio-Temporal Reconstruction of OCT Volumes

Arturo Vicente-Jaén[1,2], Juan Mompeán[1,2], Juan L. Aragón[1] and Pablo Artal[2]

Resumen— **This work presents the design, analysis and development of VolRec, a volumetric 4D reconstruction system for OCT (Optical Coherence Tomography) data. VolRec is able to obtain and process more than 25 high resolution OCT data volumes per a second. This way, VolRec allows OCT systems to perform real-time OCT measurements that could help doctors and physicians in lots of different cases, e.g., during surgical interventions or during exams to restless patients, such as kids.**

To achieve this real-time performance it is necessary to address the limitations of current OCT systems, which are restricted by both the acquisition and the processing of massive amounts of data. In order to overcome those limitations we propose an approach based on a 4D volumetric reconstruction. Our technique allows to obtain high-resolution volumes by increasing the acquisition speed while reducing the acquisition resolution without sacrificing image quality.

To this end, we use parallel programming mechanisms such as OpenMP (for CPUs) and CUDA to exploit the computing capabilities of modern GPUs. Our real-time volumetric reconstruction algorithm efficiently achieves a very high performance by exploiting the vast amount of data-level parallelism inherent to OCT data volumes.

Palabras clave— **parallelism, CUDA, OpenMP, GP-GPU, OCT**

I. Introduction

Optical Coherence Tomography (OCT) [1, 2] is a common technique of imaging diagnostic for a wide variety of ocular purposes and ophthalmic procedures. It is able to obtain 3D deep ocular volumes from human eyes in a fast non-invasive way and with high quality. Thus, OCT systems are very useful for the detection of multiple types of ocular pathologies from cataracts to retinal damage [3].

OCT systems have improved over the years in both speed and quality since they were invented 30 years ago [4]. However, their throughput is not enough yet to perform real-time visualization of the data, which can be preferred to an off-line visualization in most of the situations. A trade-off between speed and image quality can significantly increase the performance, however, our goal is to minimize the quality loss while increasing the processing speed.

For this reason, we decided to update our OCT system, developed by authors in [5, 6], to make it capable of performing real-time measurements. This OCT system is shown in Figure 1, together with its basic operating diagram.

A detailed analysis of both the OCT's data acquisition system and previously implemented algorithms has been performed to identify the main bottlenecks and the critical parts that should be optimized. Typically, OCT systems are capable of taking one or even less 3D data volumes per second, although this number might greatly vary depending on the system configuration. When an OCT system is designed, some trade-offs are done between 3D volumes per second and resolution. Usually, throughput is sacrificed to keep the maximum resolution.

In the OCT system evaluated in this work, resolution was originally maximized, capturing a large volume of 300x300x8192 *voxels*, although smaller resolutions are also possible. Therefore, the typical 3D volume is composed of 700 million voxels whose acquisition takes around 2 seconds. Then, the processing of the 3D volume is performed and the resulting 3D image is displayed.

The goal of the proposed VolRec approach is to keep the maximum resolution but increasing the throughput during acquisition, which is only possible by reducing the resolution of the acquired 3D volumes. Subsequently, some intelligent processing is needed in order to improve the quality of the displayed 3D image and recover the original (maximum) resolution. During this processing, advanced techniques are applied to generate a full-resolution and high-quality 3D volume from the available data. Note that, to achieve which is usually known as real-time imaging (i.e., 25 data volumes per second), it is needed a processing rate of 17 GigaVoxels/sec at the very least.

Thus, our approach consists of applying the classical algorithm (dispersion compensation, linearization, FFT, ...) but using lower resolution data sets, that can be acquired in real time and then dynamically reconstruct a full-resolution final volume by leveraging not only 3D spatial interpolation but also temporal interpolation, and hence our 4D reconstruction approach, by using data from previous volumes. The use of data from previous volumes to *approximate* current data is a novel approach for OCT systems but well known in the fields of video compression and real-time graphics rendering, commonly referred to as frame-to-frame temporal coherence, meaning that in a video sequence two consecutive frames exhibit minor changes.

By exploiting temporal coherence in low-resolution OCT data volumes, it is possible to recover and reconstruct high-resolution volumes with fine details in real time.

[1]Dpto. de Ingeniería y Tecnología de Computadores, University of Murcia. e-mail: {arturo.vicentej,jlaragon}@um.es

[2]Laboratorio de Óptica, University of Murcia. e-mail: {juan.mompean,pablo}@um.es

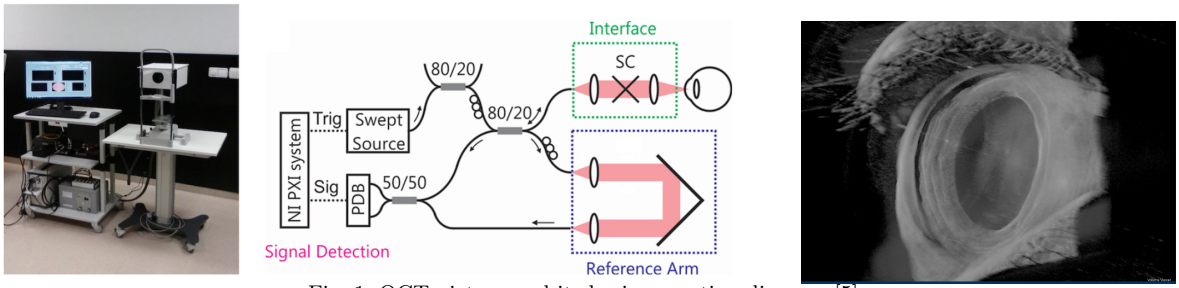


Fig. 1: OCT picture and its basic operating diagram [5].

II. Background and Related Work

A. Background

As it is shown in the diagram of Figure 1, the OCT system that we are improving is based on a light source, called swept source, whose output is split into both the reference arm and the sampling arm. A 2-axis galvanometer scanner is used in the sampling arm to scan the target (typically a human eye) and acquire its corresponding 3D data volume.

Later, the system combines the light from both beams at the exit of their respective arms, and a high-speed photo-detector is responsible of capturing the raw data. Finally, the photo-detector sends the raw data to the computer, where the processing of the 3D volume is done.

Before providing an insight about how this processing is performed, it is important to introduce some basic terminology about how the different entities of an OCT volume are named. There are three levels in a volume, the smallest one is called a *voxel*, a 0-dimensional entity corresponding to each of the individual data points captured by the OCT system (it is the equivalent to a pixel in a 2D image). The next level corresponds to an *A-scan*, a 1-dimensional entity that corresponds to a *line* of voxels along the Z axis (in the evaluated OCT, each *A-scan* contains 8192 voxels). In the next level, a *B-scan* is a 2-dimensional entity that refers to a horizontal cut of the volume, i.e., a 2D *plane* composed of a set of *A-scans* (in the evaluated OCT, each *B-scan* corresponds to a 2D plane of 300x8192 voxels). Lastly, the whole volume, a 3-dimensional entity, is usually referred to as a *C-scan*, but we will call it (3D) volume for simplicity.

The processing of an OCT volume basically consists of performing a Fast Fourier Transform (FFT) per *A-scan*. As there are many different 1-dimensional *A-scans* in a volume, this involves a lot of computation but very independent, so it could be easily parallelized, and GPUs are good hardware platforms to be used thanks to their powerful compute capabilities, as it will be further explained in Section III-A.

B. Related Work

The huge amount of data obtained by OCT systems is very well suited for parallelism, as most of the processing can be done in parallel for each voxel or for each *A-scan*. Being an inherently parallel task has attracted the interest of many researchers who have achieved impressive results trying to speed up the OCT processing. Despite this, existing research works have focused on accelerating the traditional OCT processing algorithm but have not tried to speed up the whole system's throughput. That causes a low throughput when using high-resolution volumes with lots of voxels, which limits the system ability to perform real-time visualization.

In order to achieve a real-time procedure when computing very high resolution 3D images, we developed a system that combines the classic approach with volumetric 4D reconstruction. To the best of our knowledge, no other previous work has developed an OCT processing algorithm based on parallel 4D volumetric reconstruction.

Next, we are going to review some of the aforementioned works that are focused on accelerating the classical FFT approach and which are the basis of our research. To compare the performance obtained by the discussed studies, their reported results are summarized in Table I. Each study has different configurations and small variations on the processing they perform. In a attempt to fairly compare their performance, the number of processed voxels per second has been selected as the comparison metric.

All the state-of-the-art works aim to improve sequential performance by exploiting parallelism through the use of diverse hardware architectures.

One the one hand, [9] used FPGA programmable hardware for computation, so its performance is not the best. However, this approach can lead to very energy-efficient systems.

On the other hand, most of the published works use GPUs [8, 11, 13, 14]. As expected, more recent publications using newer hardware obtain much better results than older ones. Particularly, [7] surpasses the 1000 MVo/s barrier and [11] achieves over 2500 MVo/s. The work in [12] reports impressive results with processing speeds higher than 5000 MVo/s when using 2 NVIDIA's TITAN RTX. Multiple GPU systems are also common in the literature. The system presented in [10] features the use of two GPUs for different purposes, one for computation and another for rendering and plotting the final images. Something similar is done in [7].

As it is shown in Table I, all previous research execute a similar processing algorithm, therefore, the obtained performance differences depend on the computing power of the underlying hardware device. This computing power together with the physical

Tabla I: Comparative analysis of previous related work.

Research work	Processing Unit	Performance
Zhang *et al.* [7]	GPU GeForce GTX 580	120 MVo/s
Cho *et al.* [8]	GPU GeForce GTX 480	110 MVo/s
Choi *et al.* [9]	20 x FPGA	120 MVo/s
Wieser *et al.* [10]	GPU GeForce GTX 690	1070 MVo/s
Mompeán *et al.* [11]	GPU GeForce GTX 1080Ti	2519 MVo/s
Britten *et al.* [12]	GPU 2 xNVIDIA TITAN RTX	5000 MVo/s

limitations of the acquisition/scanning systems are the main limiting factors of the classic approach.

In order to overcome the aforementioned limitations, our approach is able to use volumetric reconstruction out of low-resolution volumes to obtain full-resolution volumes with high accuracy.

III. Proposed Solution

The proposed algorithm is called VolRec and performs a novel and fast 4D spatio-temporal reconstruction from low-resolution volumes (typically of 60x60x8192 voxels), instead of using the full-resolution ones (300x300x8192 voxels) that were previously preprocessed using the classical FFT approach, aimed at reconstructing high-resolution data volumes in real time. Therefore, the proposed approach allows to generate 3D data volumes at a high rate (in terms of volumes per second) without compromising the quality of the output.

Our VolRec algorithm is split in two main stages:

- The first stage *preprocesses* raw data of smaller resolution volumes.
- The second stage *reconstructs* the low-resolution preprocessed volumes into the final full-resolution ones. To do that, it applies both interlace and interpolation operations.

A. First Stage: Preprocessing

This preprocessing algorithm generates low-resolution volumes from the raw data extracted from the optical system and it is completely based on a FFT. However, it is needed to do some initial work in order to prepare the data for the FFT operations. In particular, an operation to subtract the mean of each *A-scan* is applied to center the data. Then, it performs an FFT for each *A-scan*, with a depth size of 8192 voxels. That means that a full-resolution volume typically needs 90000 (300x300) FFTs whereas a low-resolution volume only performs 3600 (60x60) FFTs. Finally, other operations to enhance visualization are applied: a logarithm scaling, a flipping of both horizontal and vertical axes, to finish with a range-based pixel normalization.

B. Second Stage: Volumetric Reconstruction

The proposed algorithm generates reconstructed full-resolution volumes from the smaller preprocessed ones. To accomplish this, it initially merges different preprocessed low-resolution volumes that were obtained in the previous epochs into a high-resolution one with mixed information. Subsequently, it applies a convolution kernel to the mentioned interlaced volume in order to obtain a final processed volume.

This convolution kernel relies on both time and space interpolation to give more weight to newer and closer voxels. This approach efficiently and successfully creates a full-resolution volume combining information from previous lower-resolution ones.

Let us explain in more detail how the 4D spatio-temporal reconstruction is performed.

- **Interlacing**: this operation creates, in each timestamp, a full-resolution volume using lower resolution volumes from the current epoch and previous ones.
  The merging of data is done by applying both horizontal and vertical offsets. For this purpose, as we mentioned before, we empirically decided to obtain from the OCT system volumes 5 times smaller than the maximum resolution in x and y dimensions (or equivalently, 25 times smaller in area). In other words, the optical system needs to be configured to obtain low-resolution volumes of size 60x60x8192 with an offset depending on the timestamp instead of a full-resolution volume of 300x300x8192.
  Overall, this size reduction *potentially* allows to acquire and preprocess the smaller volume 25 times faster than when using full-resolution ones, however, at the expense of reducing data quality, in case no treatment is done to approximately recover the original full resolution.
  As it is shown in Figure 2, *A-scans* are obtained by having a 5-voxel offset in X and Y dimensions for each epoch (recall, that an *A-scan* is a line of voxels in the Z axis; for clarity, the Z dimension is not shown in Figure 2).
  Finally, an interlacing operation is applied to compose the full-resolution volume from a set of 25 low-resolution ones. This interlacing operation consists of inserting in the correct X-Y offsets (positions from 0 to 24 in the Figure) the *A-scans* from the newly acquired low-resolution volumes to compose a final full-resolution volume with information from previous epochs. Therefore, a full-resolution volume is the result of spatially interlacing the last 25 acquired lower-resolution volumes.
- **Processing**: Lastly, we apply a spatio-temporal convolution kernel of size 3x3x3 to the full-resolution interlaced volume. Kernel weights ha-

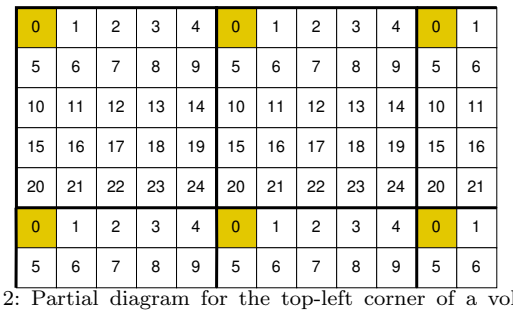


Fig. 2: Partial diagram for the top-left corner of a volume showing the acquisition epochs of each *A-scan*. The yellow *A-scan* is the next to be updated.

ve been carefully selected in a way that for a given voxel more relevance is given to the neighbour voxels that are newer and closer. The pseudocode of the applied convolution kernel is shown in Algorithm 1.

Algorithm 1 Pseudocode for the spatio-temporal convolution kernel.

```
1:  procedure ProcessVolume(entryV, exitV, epoch)
2:      for e ← 1 a entryV.size() do
3:          curr ← entryV[e]
4:          sum ← 0
5:          weight ← 0
6:          neighs ← initNeigh(curr, entryV)
7:          for i ← 1 a n.size() do
8:              n ← neighs[i]
9:              tempW = calcTempW(n, epoch)
10:             spatialW = calcSpatialW(n)
11:             combW ← comb(tempW, spatialW)
12:             sum ← sum + n * combW
13:             weight ← weight + combW
14:         end for
15:         exitV[e] ← sum/weight
16:     end for
17: end procedure
```

On the one hand, *spatial weights* are configured using a 3D Gaussian distribution, that gives much more importance to closer voxels.
On the other hand, *temporal weights* are assigned to every voxel of each *A-scan* depending on how old it is in each epoch, giving less relevance to *A-scans* coming from older volumes. Note that these time-based weights are dynamic and must change in each epoch for each *A-scan*. That causes some issues that have to be solved, as we will explain later. Note also that these temporal weights are the same for every 5x5 block depicted in Figure 2, which represents these temporal weights distribution for each *A-scan*.

IV. Experimental Framework

For testing the CUDA variants we use a NVIDIA 1080TI GPU with 3584 CUDA-cores and 11 GiB of VRAM memory with 484.4 GB/s peak bandwidth. This is a high-end GPU with many more cores, memory and speed than the vast majority of commercial GPUs.

In addition to that, an Intel Xeon Gold 5218 processor at 2.30 GHz with 64 cores has been used to evaluate an OpenMP implementation of the proposed algorithm. This massive CPU allows to obtain

Algorithm 2 Pseudocode for Non-Cumulative Processing of VolRec

```
1:  procedure Non-Cumulative Processing
2:      epoch ← 0
3:      mergedV ← initializeV()
4:      loop
5:          currRaw ← getCurrRaw()
6:          preproV ← preprocess(currRaw)
7:          mergedV ← interl(mergedV, preproV)
8:          exitV ← convolution(mergedV, epoch)
9:          show(exitV)
10:         epoch ← epoch + 1
11:     end loop
12: end procedure
```

fast OpenMP results and can be a good reference point when comparing with the CUDA variants. To ensure that the CPU version is properly optimized, the FFTs have been calculated using the FFTW3 library [15].

As experimental input set, we use a full-resolution volume which was obtained with the OCT device presented in Section I. This volume is used to simulate eye movements, such as rotations that gives the appearance of a real human eye in movement. This allows us to test the behaviour of VolRec in a realistic setting.

V. Accelerating VolRec

CUDA is a powerful programming model to extract all the available parallelism in NVIDIA GPUs. However, it is not enough to simply translate from C code to a naive CUDA version. For this reason, some specific optimizations must be applied in order to achieve the most efficient and fast implementation possible. In addition, all applied optimizations work together to obtain a significant reduction of the execution time. To better understand the impact of each optimization, the performance results have been recorded after applying each one of them.

Next, we explain the different optimizations applied to the proposed VolRec algorithm to make it capable of achieving the desired real-time processing speed.

The preprocessing stage has been parallelized similarly to previous works, such as, the ones presented in [7, 11, 13].

Regarding the final processing phase that involves the full-resolution volumetric reconstruction, Figure 3 summarizes the speedup obtained for each one of the implemented versions which are described next.

- **Naive CUDA version.** The naive version for this second stage of the VolRec algorithm simply maps the sequential C code of the convolution kernel into CUDA code. Thus, the interlacing operation is still performed in a sequential manner in the CPU.
  This way, each CUDA thread must calculate one exit voxel. This calculation consists of loading every neighbour of the current voxel plus the temporal and the spatial weights. The value of each voxel is multiplied by its corresponding temporal and spatial weights and the output vo-

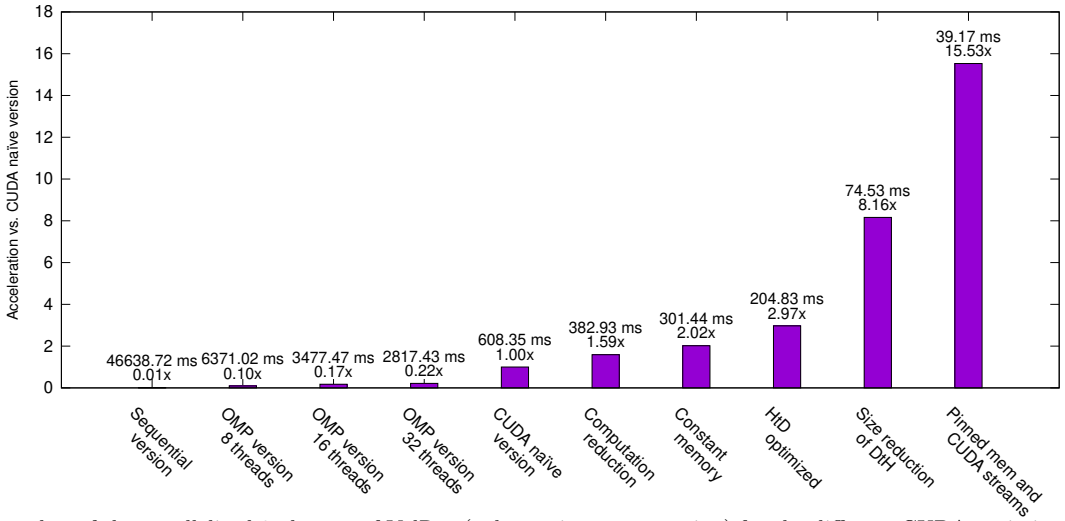


Fig. 3: Speedup of the parallelized 2nd stage of VolRec (volumetric reconstruction) for the different CUDA optimizations (over the naive CUDA version).

xel value is obtained.

Additionally, for each volume it is necessary to copy the whole high resolution volume from both GPU to CPU and vice versa.

This naive version computes each volume in 608.35 ms, which means a rate of 1.64 volumes per second. It reaches a 4.54× speedup versus the fastest OpenMP version.

- **Reducing the amount of computation.** We realised that the previous naive version was mainly limited by the compute power of the GPU hardware. This limitation appears because the naive version calculates both temporal and spatial weights for each neighbour of each result voxel. However, voxels of each *A-scan* share the same temporal weight in each specific epoch. Thus, temporal weights for each *A-scan* can be calculated just one time per epoch instead of 4096 (one per voxel in an *A-scan*) in order to reduce computational complexity.

  As it can be noted, the naive version for this second stage is doing lots of repetitive and unnecessary operations that can be optimized. For that, we decide to precalculate the temporal weights with a CUDA kernel and then reuse them for every voxel in each *A-scan* before launching the processing for each epoch.

  Therefore, the current version calculates just $300 * 300$ instead of $300 * 300 * 4096 * 27$ temporal weights, which means we require much less computational power for processing a full-resolution volume.

  This version obtains each volume in 382.93 ms. Hence, it can process 2.6 volumes per second, reaching a 1.59× speedup over the naive version.
- **Use of constant memory for precalculated weights.** The previous version, even though is able to reduce the computing cost, it is still limited by the memory access latency because many threads try to access the same global memory locations simultaneously in order to retrieve the precalculated temporal and spatial weights. Moreover, global memory banks in GPUs have a bad multi-cast behaviour that makes impossible to load the same position by different threads of the same warp at exactly the same time.

  The solution to reduce global memory latency problems is to replace it by constant memory, which has much better multi-cast performance, to store both types of weights.

  This new optimization yields a 2.02× speedup versus the naive version. Thus, it can process each volume in just 301.44 ms which represents a processing rate of 3.32 volumes per second.
- **Host-to-device copies optimization.** In the previous versions, only the convolution kernel is executed in the device. That way, on each epoch we copy from host to device the whole high-resolution merged volume, which has a size of $300 * 300 * 4092 * 4$ bytes ≈ 1.4 GiB. Hence, the above versions spend a large portion of their execution time copying data back and forth between the CPU and the GPU.

  A way to solve this problem is by moving the interlacing operation to the GPU. That allows to interlace in a parallel and faster way. It also allows to copy just the low-resolution preprocessed volume to the device on each epoch. These smaller volumes, as we explained in preceding Sections, have a size of $60 * 60 * 4096 * 4$ bytes ≈ 0.056 GiB, which is 25 times smaller than the full-resolution volume. Hence, host-to-device copies can reach a theoretical 25× speedup.

  Overall, this version is able to process a volume in just 204.83 ms. Thus, it can process 4.88 volumes per second, achieving a speedup of 2.97× over the naive version.
- **Device-to-host copies optimization.** In contrast with the previous optimization, the full-resolution volume must be copied from the device to the host. As each voxel is represented with a `float` data type, a whole volume occupies $300 * 300 * 4096 * 4$ bytes ≈ 1.4 GiB.

  In an attempt to reduce the excessive amount of

data being copied, the data type has been changed to `uint8_t` before doing the copy. This aims to reduce 4 times the data size and the resulting copies can theoretically be done 4 times faster. Nonetheless, quality is not reduced when applying this optimization as the volume codification when saving the results is 8 bits.
Therefore, this optimization is able to process each volume in just 75.53 ms, which corresponds to more than 13 volumes per second. This translates to a 8.16× speedup over the naive version.

- **Pinned memory and CUDA streams.** Finally, we use pinned memory and CUDA streams to process and copy data in parallel from different *B-scans*. Thus, we overlap copies from a portion of the volume with compute done in a different portion of the volume to minimize the processing time. This final optimization is able to process each volume in 39.17 ms, which means that we can process more than 25 volumes per second, therefore, achieving the desired real-time processing. In addition, it gets a 15.53× speedup over the naive version.

Summarizing, both preprocessing and processing stages are configured using a pipeline that overlaps different stages (both copy and compute) from volumes in different epochs. This allows to obtain and process each volume in real time with no more than one tenth of second of latency.

VI. Quality and Accuracy of VolRec

A key part of this novel approach to improve the processing speed of OCT systems is to be able to maintain a certain level of quality. In order to make a fair comparison between them, we use two different metrics: *Peak Signal-to-Noise Ratio* (PSNR) and *Structural Similarity Index* (SSIM).

PSNR is a classical metric which has demonstrated to be a valid quality indicator for videos and images [16]. Thus, it is widely used in image comparison and engineering in general. It uses a logarithmic scale, ranging from zero (for two completely different images) to infinite (for two completely identical images).

SSIM [17] is a modern metric that calculates differences between two images according to dissimilarities in the structural data. Hence, SSIM gives much importance to differences between groups of pixels that are closely related in terms of luminosity, variance and reciprocity. This index uses a range between zero (for two completely different images) to one (for two completely identical images).

A. Quality of the Preprocessing Stage

For this stage, we compare preprocessed volumes using both the CUDA and the CPU versions, and we measure the SSIM and PSNR metrics comparing both versions. Note that the sequential CPU version is the reference point and it is considered the correct result.

Table II reports both quality metrics, SSIM and PSNR, comparing a volume generated with our parallel GPU version against the same volume generated with the sequential CPU version. It can be noted that both GPU and CPU versions generate a very similar output full-resolution volume, in terms of the SSIM metric (very close to 1) and achieve a very high PSNR. This confirms that the developed CUDA algorithm for the preprocessing stage is correct and with a extremely high fidelity.

Tabla II: Comparison of GPU and CPU generated volumes just considering the preprocessing stage.

SSIM	PSNR (dB)
0.9999999622	93.3129405421

B. Quality of the Volumetric Reconstruction Stage

To measure the quality of this second and more critical stage, we have created a data set that is used to evaluate the parallelized GPU algorithm. Due to physical and mechanical limitations of the scanning part of the OCT system, it is impossible to scan volumes at the maximum resolution in real time, which takes around 2 seconds each, being much slower than common eye movements. Therefore, we have emulated the movement of an eye using a statically acquired volume at full resolution, and later making small rotations, in order to generate a sequence of full-resolution volumes representing a moving eye.

In order to better understand the values obtained by the similarity metrics in this analysis, we launch some tests for the same volume with 3 rotations: $0.1°, 1°$ and $10°$. Results are reported in Table III and provide a reference point. It can be noticed that even a very small rotation of $0.1°$ results in a significant change in the SSIM metric, which clearly drops from almost 1 to around 0.5. The PSNR is less sensitive to such slight changes, going from about 93 dB (when comparing 2 equal volumes) to 22 dB for a $0.1°$ rotation. On the other hand, a clearly noticeable rotation of $1°$ directly kills the SSIM index, dropping it bellow 0.10, whereas the PSNR falls bellow 20 dB.

Tabla III: Comparison between volumes with different rotation angles.

Rotation (°)	SSIM	PSNR (dB)
0.1	0.4907529	22.76333
1	0.0994467	19.48928
10	0.0893903	18.53576

The previously created dataset represents a low-speed movement which simulates an eye rotating from $-0.5°$ to $0.5°$ with a step of $0.01°$. This movement is very common in a real-case scenario. The full-resolution generated volume, using both VolRec and the raw preprocessed data, is shown in Figure 4c whereas the corresponding similarity metrics are reported in Table IV.

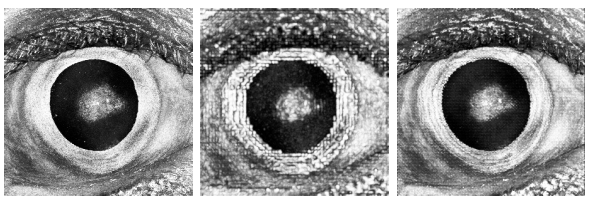
In addition, comparing the snapshots of Figure 4, it can be noted that the generated full-resolution volume (Figure 4c) does not exhibit any noticeable artifact, because one convolution kernel is enough for reducing ghosting without creating any grid effect.

Overall, the volume generated thanks to VolRec is visually similar to the ideal one (Figure 4a) and much better than the low-resolution volume (Figure 4b), which is the best that is possible to generate in real time without VolRec.

These results show that VolRec behaves very well in common situations during examinations with real patients, such as low-speed eye movements.

Tabla IV: Similarity for a low-speed movement.

Rotation (°)	SSIM	PSNR (dB)
Preprocess	0.13314	20.27044
VolRec	0.40078	23.61127



(a) Ideal – full resolution (b) Preprocessed – low resolution (c) Result after VolRec

Fig. 4: Frontal projections of the evaluated 3D volumes.

VII. Conclusions

This work demonstrates that it is possible to process and generate high-resolution 3D volumes for OCT systems by using the proposed VolRec in GPUs, which is able to generate more than 25 full-resolution volumes per second by carefully applying spatio-temporal interlacing and convolution in order to dynamically reconstruct OCT volumes from low-resolution ones.

Experimental results showed that the most optimized CUDA version obtains 71.92× speedup over a parallel OpenMP version in a CPU, and more than 1000× over the sequential version.

We believe that the proposed algorithm will help to improve worldwide OCT systems by developing more versatile systems that can be used in real time in lots of situations such as ocular surgeries or ocular exams for restless patients as kids. Obtained results are top quality for both high- and low-speed movements what proves that VolRec exhibits a good behaviour even in unfavorable situations.

To conclude, the proposed VolRec algorithm is a powerful tool that reaches a processing rate of 18 GigaVoxels/sec, which is more than 7 times faster than the baseline approach while providing a reconstruction fidelity close to ideal full-resolution volumes.

Acknowledgements

This work has been supported by Grants PLEC2022-009214, PDC2021-121575-I00 and TED2021-130233B-C33 funded by MCIN/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR. As well as by Grant LCF/TR/CC21/52490004 funded by Fundación la Caixa.

References

[1] David Huang, Eric A. Swanson, Charles P. Lin, Joel S. Schuman, William G. Stinson, Warren Chang, Michael R. Hee, Thomas Flotte, Kenton Gregory, Carmen A. Puliafito, and James G. Fujimoto, "Optical coherence tomography," *Science*, vol. 254, no. 5035, pp. 1178–1181, 1991.

[2] Wolfgang Drexler, James G Fujimoto, et al., *Optical coherence tomography: technology and applications*, vol. 2, Springer, 2015.

[3] A L Wong, C K-S Leung, R N Weinreb, A K C Cheng, C Y L Cheung, P T-H Lam, C P Pang, and D S C Lam, "Quantitative assessment of lens opacities with anterior segment optical coherence tomography," *British Journal of Ophthalmology*, vol. 93, no. 1, pp. 61–65, 2009.

[4] Johannes F De Boer, Rainer Leitgeb, and Maciej Wojtkowski, "Twenty-five years of optical coherence tomography: the paradigm shift in sensitivity and speed provided by fourier domain oct," *Biomedical optics express*, vol. 8, no. 7, pp. 3248–3280, 2017.

[5] Alberto de Castro, Antonio Benito, Silvestre Manzanera, Juan Mompeán, Belén Cañizares, David Martínez, Jose María Marín, Ireneusz Grulkowski, and Pablo Artal, "Three-Dimensional Cataract Crystalline Lens Imaging With Swept-Source Optical Coherence Tomography," *Investigative Ophthalmology & Visual Science*, vol. 59, no. 2, pp. 897–903, 02 2018.

[6] Ireneusz Grulkowski, Silvestre Manzanera, Lukasz Cwiklinski, Juan Mompeán, Alberto de Castro, Jose Maria Marin, and Pablo Artal, "Volumetric macro- and micro-scale assessment of crystalline lens opacities in cataract patients using long-depth-range swept source optical coherence tomography," *Biomed. Opt. Express*, vol. 9, no. 8, pp. 3821–3833, Aug 2018.

[7] Wolfgang Wieser, Wolfgang Draxinger, Thomas Klein, Sebastian Karpf, Tom Pfeiffer, and Robert Huber, "High definition live 3d-oct in vivo: design and evaluation of a 4d oct engine with 1 gvoxel/s," *Biomed. Opt. Express*, vol. 5, no. 9, pp. 2963–2977, Sep 2014.

[8] Nam Hyun Cho, Unsang Jung, Suhwan Kim, Woonggyu Jung, Junghwan Oh, Hyun Wook Kang, and Jeehyun Kim, "High speed sd-oct system using gpu accelerated mode for in vivo human eye imaging," *J. Opt. Soc. Korea*, vol. 17, no. 1, pp. 68–72, Feb 2013.

[9] Donghak Choi, Hideaki Hiro-Oka, Kimiya Shimizu, and Kohji Ohbayashi, "Spectral domain optical coherence tomography of multi-mhz a-scan rates at 1310nm range and real-time 4d-display up to 41 volumes/second," *Biomed. Opt. Express*, vol. 3, no. 12, pp. 3067–3086, Dec 2012.

[10] Kang Zhang and Jin U. Kang, "Real-time intraoperative 4d full-range fd-oct based on the dual graphics processing units architecture for microsurgery guidance," *Biomed. Opt. Express*, vol. 2, no. 4, pp. 764–770, Apr 2011.

[11] Juan Mompeán, *Desarrollo de sistemas de tiempo real mediante el uso de aceleradores gráficos y específicos aplicados a la óptica visual humana*, Ph.D. thesis, Universidad de Murcia, 2020.

[12] Anja Britten, Philipp Matten, Jakob Weiss, Michael Niederleithner, Hessam Roodaki, Benjamin Sorg, Nancy Hecker-Denschlag, Wolfgang Drexler, Rainer A. Leitgeb, and Tilman Schmoll, "Surgical microscope integrated mhz ss-oct with live volumetric visualization," *Biomed. Opt. Express*, vol. 14, no. 2, pp. 846–865, Feb 2023.

[13] Haiyi Bian, Jingtao Wang, Chengjian Hong, Lei Liu, Rendong Ji, Suqun Cao, Ahmed N. Abdalla, and Xinjian Chen, "Gpu-accelerated image registration algorithm in ophthalmic optical coherence tomography," *Biomed. Opt. Express*, vol. 14, no. 1, pp. 194–207, Jan 2023.

[14] Xiaofeng Deng, Kaiyuan Liu, Tiepei Zhu, Dayou Guo, Xiaoting Yin, Lin Yao, Zhihua Ding, Juan Ye, and Peng Li, "Dynamic inverse snr-decorrelation oct angiography with gpu acceleration," *Biomed. Opt. Express*, vol. 13, no. 6, pp. 3615–3628, Jun 2022.

[15] Matteo Frigo and Steven G. Johnson, "The design and implementation of FFTW3," *Proc. of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, Special issue on "Program Generation, Optimization, and Platform Adaptation".

[16] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of psnr in image/video quality assessment," *Electronics Letters*, vol. 44, pp. 800–801(1), June 2008.

[17] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

Implementación paralela del algoritmo HSI-MSER para el registro de imágenes hiperespectrales

Daniel del Castillo[1,2], Álvaro Ordóñez[3], Dora B. Heras[1,2] y Francisco Argüello[2]

Resumen— **El registro de imágenes es una tarea destinada a alinear imágenes como un paso anterior a otros tipos de procesado como son la detección de cambios o el seguimiento de objetos. En el caso de imágenes de teledetección hiperespectrales tenemos una gran cantidad de información tanto espacial como espectral para cada imagen. Esto ha propiciado la aparición de métodos de registro específicos para este tipo de imágenes que no solo usan las bandas RGB, como sucede en los algoritmos más clásicos, sino que además aprovechan la información espectral contenida en todas las bandas, que pueden llegar a ser del orden de un centenar. No obstante, para algunos de estos algoritmos, especialmente los basados en detección de características como líneas, puntos o regiones, el tiempo de ejecución sigue siendo una barrera para su uso en aplicaciones que requieren un procesado especialmente rápido. Uno de estos algoritmos es *Hyperspectral Image-Maximally Stable Extremal Regions* (HSI-MSER), que trata de encontrar regiones comunes en las imágenes explotando la información disponible en las bandas espectrales para conseguir un mejor registro. En este artículo se presenta una versión paralela del algoritmo HSI-MSER sobre una arquitectura heterogénea de bajo coste. Ha sido diseñada para explotar eficientemente las arquitecturas de la CPU y GPU usando OpenMP y CUDA, respectivamente. Como resultado, para imágenes hiperespectrales de teledetección disponibles en la bibliografía estado del arte se consiguen aceleraciones de hasta 7× respecto a la versión secuencial de HSI-MSER.**

Palabras clave— **Registro de imágenes, hiperespectral, teledetección, CUDA, GPU, OpenMP**

I. Introducción

El registro de imágenes es una operación crucial a la hora de trabajar con imágenes de teledetección de superficie terrestre capturadas por drones o satélites. Se puede definir como un proceso que toma dos imágenes que capturan información de la misma área geográfica y modifica la escala, la rotación y el desplazamiento de una de ellas para que se alinee con la otra. Esta técnica puede ser necesaria para diferentes propósitos. El más habitual es realizar un análisis de cambios sobre imágenes de la misma área capturadas en distintos momentos, a diferentes alturas o con equipos distintos.

Es importante destacar que existe una gran variedad de algoritmos de registro. La mayoría de ellos se pueden clasificar en dos tipos: basados en área o basados en características. Los métodos basados en área, como los basados en la *Fourier-Mellin transform* (FMT) [1], trabajan directamente con las intensidades buscando correlaciones para realizar el alineamiento. Por el contrario, los métodos basados en características, como *Speeded Up Robust Features* (SURF) [2], buscan ciertos elementos dentro de cada imagen, como líneas o puntos. Esto hace que tengan un coste computacional mayor que los basados en intensidad, pero, sin embargo, consiguen una mayor tolerancia a cambios en la imagen [3]. Esto permite registrar imágenes con mayores diferencias de escala entre ellas, lo que resulta de gran utilidad para ciertas aplicaciones cuando se ha de trabajar con imágenes obtenidas, por ejemplo, a alturas muy diferentes.

En la actualidad, cada vez son más los sistemas que incorporan sensores hiperespectrales que permiten obtener mapas detallados del terreno con una alta resolución espectral, lo que hace necesario disponer de técnicas de registro que hagan un uso eficiente y eficaz de toda la información que contienen. HSI-MSER [4] es un algoritmo para el registro de dos imágenes hiperespectrales basado en la detección de características, en este caso, regiones de interés. El método explota eficientemente la información espectral disponible en distintas bandas espectrales e incorpora este tipo de información en el descriptor que se construye para cada región detectada. Esto permite registrar imágenes con diferencias de escala de hasta 15×.

A pesar de ser más tolerantes a cambios, los métodos basados en características presentan como desventaja un mayor tiempo de ejecución, ya que se basan en una extracción profunda de características como, por ejemplo, regiones con determinadas propiedades sobre ambas imágenes. Las características son a continuación descritas mediante vectores de gran longitud para realizar después un proceso de emparejamiento entre las dos imágenes que requiere una búsqueda exhaustiva. Esta desventaja se hace notar especialmente a la hora de trabajar con imágenes hiperespectrales, ya que una primera aproximación exigiría buscar características sobre todas las bandas espectrales de ambas imágenes. Sin embargo, las empresas y usuarios que hacen uso de este tipo de imágenes no siempre tienen acceso a supercomputadores o infraestructuras computacionales costosas. Además, resulta mucho más conveniente en muchos

[1]Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, e-mail: {d.delcastillo,dora.blanco}@usc.gal.

[2]Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, e-mail: francisco.arguello@usc.gal.

[3]Universidade da Coruña, Grupo Integrado de Ingeniería, CITIC, Elviña, 15071 A Coruña, e-mail: alvaro.oiglesias@udc.gal

casos tratar este tipo de imágenes directamente en el punto de recogida de los datos para poder ofrecer a los expertos información que facilite una toma de decisiones rápida. Es por ello que se necesitan algoritmos especialmente diseñados para explotar el paralelismo del hardware comúnmente disponible sobre el terreno. En la actualidad este podría, por ejemplo, consistir en un sistema multinúcleo provisto de una o varias GPUs. Existen en la literatura diferentes algoritmos diseñados para su ejecución en este tipo de arquitecturas paralelas, pero ninguno de ellos está basado en registro mediante detección de regiones de interés [5], [6].

En este artículo se presenta una primera implementación paralela del algoritmo HSI-MSER [4], que realiza registro de imágenes hiperespectrales basado en la detección de regiones delimitadas mediante elipses, para su ejecución en un sistema heterogéneo basado en un procesador multicore y GPU. El algoritmo ha sido desarrollado en OpenMP y CUDA para obtener una explotación eficiente de la arquitectura.

El resto del artículo tiene la siguiente estructura. La sección II explica las bases de este trabajo y el algoritmo original HSI-MSER. La sección III describe la implementación híbrida en CPU y GPU desarrollada. En la sección IV se pueden encontrar una evaluación de los resultados conseguidos, analizando no solamente la eficiencia en cuanto a ahorro de tiempo de computación, sino también la calidad en cuanto a capacidad de registro de los algoritmos paralelos desarrollados. Por último, en la sección V se indican los retos alcanzados, así como vías de trabajo futuro.

II. Algoritmo original

En esta sección se describe el algoritmo HSI-MSER para el registro de dos imágenes hiperespectrales de teledetección, para luego explicar el diseño del algoritmo paralelo mediante OpenMP y CUDA.

Maximally stable extremal regions (MSER) [7] es un algoritmo de detección de características que extrae regiones que persisten al analizar la imagen con diferentes umbrales de gris. Estas regiones son invariantes a las transformaciones de la imagen, lo que las hace ideales para el registro. *Scale Invariant Feature Transform* (SIFT) [8] es un conocido detector y descriptor de puntos clave basado en la construcción de un espacio de escala gaussiano. HSI-MSER adapta MSER para su uso en imágenes hiperespectrales, utilizando una adaptación de SIFT como descriptor espacial de regiones e introduciendo un método para seleccionar qué bandas espectrales utilizar en función de su entropía, denominado *Entropy-Based Band Selection* (EBS). HSI-MSER además introduce un descriptor espectral a cada región detectada y propone un método exhaustivo para el cálculo de la transformación geométrica que al ser aplicada sobre una imagen la alinea con la otra. Todo esto ha permitido a este algoritmo aprovechar la información presente en las distintas bandas de una imagen hiperespectral para mejorar la precisión de registro.

El algoritmo HSI-MSER está compuesto por 6 fases, como se puede ver en la Figura 1, que son explicadas a continuación.

Selección de bandas. Las imágenes hiperespectrales están formadas por cientos de bandas contiguas que, dada su proximidad en cuanto a la longitud de onda asociada a ellas, contienen información redundante en algunos casos. Habitualmente se aplica un método de extracción de las bandas estadísticamente más relevantes que permita realizar el registro sobre un número de ellas que no supere la decena. Tal como se puede observar en la Figura 1, la primera fase del algoritmo consiste en aplicar el método de selección de bandas llamado *Entropy-Based Band Selection* (EBS) [4]. Este método tiene la peculiaridad de tener en cuenta las dos imágenes, en lugar de escoger las bandas de cada imagen por separado. EBS selecciona las N bandas con mayor entropía que están separadas por al menos D bandas, cuando estas están ordenadas por longitud de onda. De esta forma consigue seleccionar bandas con una alta entropía, pero que también difieren sustancialmente en longitud de onda.

Detección de regiones. En esta segunda fase, se analizan cada una de las bandas escogidas en ambas imágenes en busca de regiones de interés. Para ello se usa una adaptación del algoritmo MSER [7]. Este algoritmo procesa todos los píxeles de la imagen en orden de intensidad y busca regiones que mantengan su forma al ir aumentando el umbral de nivel de gris. De esta forma se obtienen regiones que existen en distintas bandas espectrales. Después, estas regiones son definidas como elipses.

Descripción de las regiones. En esta fase se describen las regiones que se detectaron durante la fase anterior. A cada región se le asigna un descriptor, es decir, una secuencia de números que resumirá distintas características de la misma. Las mismas regiones, pero extraídas en diferentes imágenes y bajo cambios de iluminación o transformaciones geométricas distintas, deben dar lugar a un descriptor similar. Esto nos permitirá emparejar las regiones de ambas imágenes en la siguiente fase. HSI-MSER propone un descriptor compuesto por información tanto espacial como espectral.

El algoritmo SIFT [8] es usado para construir la parte espacial del descriptor. Como SIFT fue diseñado para describir puntos y no regiones, en HSI-MSER se emplea una adaptación que tiene en cuenta el área de cada región. Para computar el descriptor primero se calculan los ángulos dominantes de cada región usando el gradiente de su superficie y una ventana circular con ponderación gaussiana. Una vez se ha determinado el ángulo dominante se puede construir el descriptor, compuesto por 128 parámetros. Estos valores se normalizan para reducir la influencia de cambios en la iluminación o el brillo.

Además de este descriptor espacial, también se tiene en cuenta la información espectral de cada región. Para ello se usa la firma espectral del centro de la elipse de todas las bandas, no solo de las escogidas durante la selección de bandas.

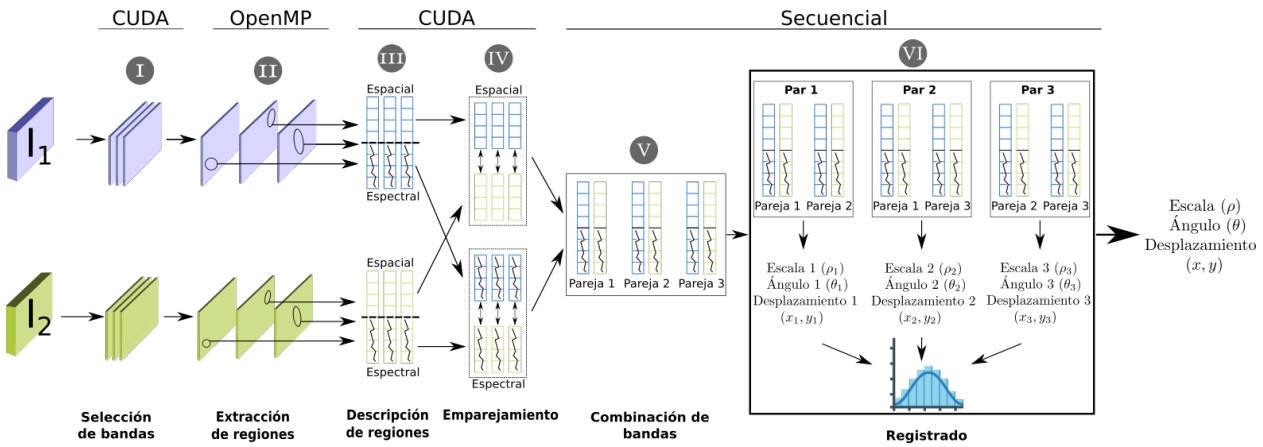


Fig. 1: Plataforma de computación paralela utilizada en cada una de las fases del algoritmo HSI-MSER. Adaptado de [4].

Emparejamiento de regiones. Una vez que las regiones han sido descritas, tratamos de encontrar parejas entre las regiones de ambas imágenes. Estos emparejamientos se hacen de forma independiente para cada banda de la imagen, pero se tiene en cuenta la información espectral del resto de bandas, como se explicó en la subsección anterior. Para calcular la similitud entre dos regiones que determina si dos regiones están emparejadas se usa la distancia euclídea para la parte espacial y la similitud coseno para la parte espectral. Únicamente se considera un emparejamiento si estas distancias están dentro de unos parámetros establecidos [4].

Combinación de las bandas. En este paso se combinan todas las parejas de regiones provenientes de todas las bandas. Esto permite usar en el registro regiones que solo aparecen en algunas frecuencias del espectro.

Registro. Para registrar las imágenes, HSI-MSER realiza una búsqueda exhaustiva basada en histogramas [4]. Para cada pareja se computan el ángulo, la escala y el desplazamiento correspondientes, tal y como se puede ver en la Figura 1. Los distintos ángulos se guardan en un histograma que representa los 360°, divididos en 72 contenedores con un tamaño base de 5° y un solapamiento de 2.5° a cada lado, llevando el total a 10°de tamaño. Una vez que este histograma se ha construido, las escalas del contenedor con más elementos se ordenan para obtener la pareja o parejas que representan la mediana. Estas parejas de regiones son las que se emplean para calcular la transformación geométrica que nos permite alinear las imágenes hiperespectrales.

III. Algoritmo paralelo propuesto para registro mediante HSI-MSER

En esta sección, se presenta la implementación paralela de HSI-MSER en CUDA y OpenMP para explotar arquitecturas heterogéneas. Se ha escogido para cada fase del algoritmo el paradigma de computación más adecuado con el fin de maximizar rendimiento.

En el Algoritmo 1 se presenta el pseudocódigo de la implementación paralela de HSI-MSER. Para cada fase, se indica con < y > que partes han sido parale-

lizadas en CUDA y con # las partes que lo han sido con OpenMP. A continuación se explican los detalles asociados a cada fase.

Selección de bandas. Esta fase se paralelizó en la GPU usando el modelo de programación CUDA. La naturaleza completamente paralela del cálculo de entropía hace que este proceso sea realizable en la arquitectura de una GPU de manera eficiente [9]. Para el cómputo de la entropía de cada banda (línea 1, Alg. 1) se usan varias funciones de la biblioteca CUB [10]. En concreto, `DeviceReduce::Min` y `DeviceReduce::Max` se utilizan para calcular de forma eficiente los valores máximo y mínimo de cada banda. Una vez se tienen estos valores, se usa `DeviceHistogram::HistogramEven` para calcular el histograma. Por último, se realiza una reducción de los valores del histograma, haciendo uso de las operaciones a nivel de *warp* para mejorar su rendimiento. Cuando ya se tienen los valores para la entropía de cada banda, se procede a seleccionar las bandas en CPU (línea 2, Alg. 1). Para ello se ordenan las bandas y se procesan siguiendo lo explicado en la Subsección II.

Extracción de regiones. MSER se basa en el algoritmo *Union find*. Este algoritmo tiene una complejidad de $O(m \cdot \alpha(m, n))$, donde n es el número de elementos, m es el número de operaciones de búsqueda que se necesitarán y α es la función inversa de Ackerman [11]. m es mayor o igual que n, pero tiene una relación de carácter lineal ($m = kn$). Esta complejidad se considera como cuasi lineal debido a la lentitud con la que crece la función inversa de Ackerman.

En la literatura ya se han publicado técnicas que se pueden aplicar para paralelizar este algoritmo [12], incluso en la GPU [13], [14]. Sin embargo, MSER, a pesar de estar basado en *Union find*, presenta varias diferencias respecto a este. Una de ellas es que los píxeles deben ser procesados por orden de intensidad creciente, lo que dificulta la paralelización de la construcción del árbol, ya que cada píxel es colocado en el árbol respecto a las posiciones de los píxeles anteriores. Si se trata de dividir la imagen para realizar la construcción del árbol en paralelo, ya sea por posición física de cada píxel o por intensidad, se consiguen árboles inconexos. En [15] se propone una

Algorithm 1 Pseudocódigo de la implementación híbrida en OpenMP y CUDA

Entrada: Imagen de referencia $\mathbf{I}_1$ e imagen objetivo $\mathbf{I}_2$.
Salida: Escala ρ, ángulo θ y desplazamiento (x, y).

Fase I. Selección de bandas.
< Cálculo de la entropía para cada banda >
Selección de las bandas con base en la entropía

para cada banda b seleccionada:
Fase II. Detección de regiones.
Realizar particiones para cada imagen según el número de hilos disponibles
Paralelizado con OpenMP
Detectar regiones por separado en cada partición

Fase III. Descripción de regiones.
< Calcular la orientación dominante de cada región >
< Computar una ventana gaussiana a partir de la banda >
< Computar el descriptor de cada región >

Fase IV. Emparejamiento de regiones.
< Calcular las distancias entre los descriptores espaciales de las regiones de ambas imágenes >
para cada región r de I_1^b:
Si La menor distancia desde r a una región de I_2^b cumple ciertos parámetros
Si La similitud espectral cumple ciertos parámetros
Guardar la pareja

Fase V. Combinación de bandas.
Combinar las parejas encontradas en distintas bandas en un único vector

Fase VI. Registro.
Computar una transformación geométrica para cada una de las parejas
Generar un histograma de ángulos
Escoger el rango de ángulos más repetido y usar la pareja mediana para el registro final

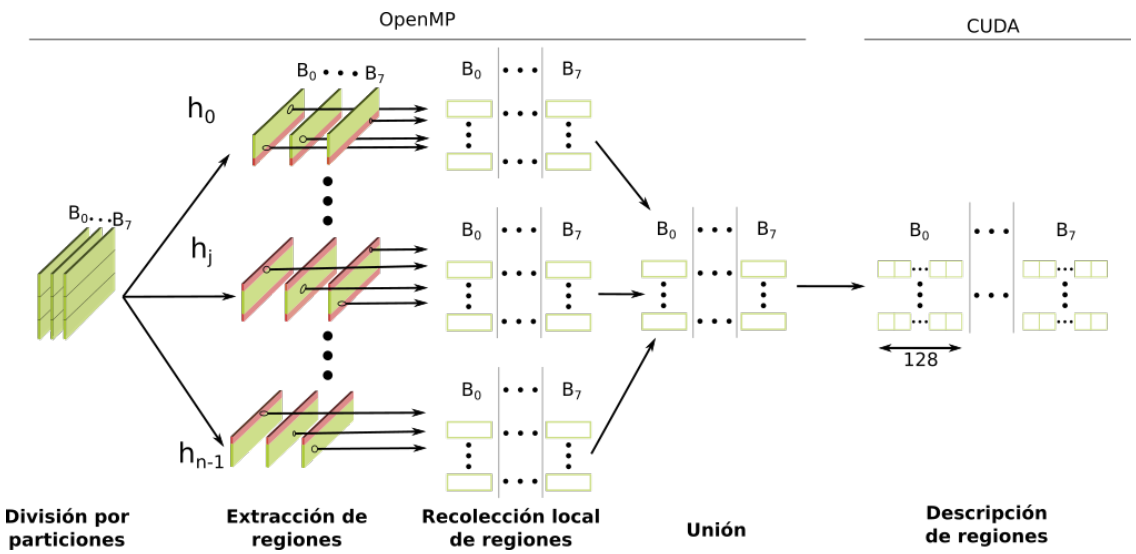


Fig. 2: Esquema de paralelización de la fase de extracción de regiones.

paralelización de MSER que fusiona estos árboles resultantes, pero que no se almacena el área de cada región, dato necesario para convertir estas regiones en elipses.

La Figura 2 muestra en detalle cómo ha sido finalmente paralelizada esta etapa. Primero, se divide la imagen horizontalmente en un número de particiones igual al número de hilos disponibles (línea 4, Alg. 1). Luego, como se puede ver en la Figura 2, se buscan las regiones en cada partición concurrentemente (línea 5, Alg. 1) haciendo uso de OpenMP [16]. Como la detección de regiones no modifica la imagen, la memoria puede ser compartida y no hay necesidad de comunicación explícita entre hilos. Cada hilo

de cómputo guarda localmente las regiones que encuentra, hasta que, finalmente, se unen todas. Para evitar que no se detecten incorrectamente regiones que estén situadas justo en los límites de una partición, se ha añadido un solapamiento entre dichas particiones. Este solapamiento se ha ilustrado en la Figura 2 con el color rojo. Un 20 % de solapamiento significa que cada hilo procesa un 120 % respecto a la versión sin solape, compartiendo un 10 % con cada uno de los hilos vecinos.

Debido a la extracción de regiones por particiones, ha sido necesario ajustar el parámetro de *min_area* de HSI-MSER para que tenga en cuenta el tamaño de la imagen completa y no solo el de la partición. Este parámetro determina cuál es el tamaño mínimo de una región para ser considerada. Su valor representa un porcentaje respecto al tamaño total de la imagen, es decir, si el valor es un 10 %, cada región tiene que tener al menos el 10 % de la imagen para ser considerada. La idea detrás de este cálculo es que, imágenes de menor tamaño es probable que tengan regiones más pequeñas. El parámetro *min_area* ha sido ajustado para que detecte regiones del mismo tamaño que si se procesara la banda completa.

Descripción de las regiones. El proceso de descripción de cada región se ha paralelizado en la GPU, tal y como aparece en la Figura 2. Para ello se han implementado tres *kernels* CUDA. En concreto, el cómputo del histograma de los ángulos (línea 6, Alg. 1), la generación de la ventana de ponderación gausiana (línea 7, Alg. 1) y el cómputo final de los descriptores (línea 8, Alg. 1). Estos *kernels* se caracterizan por una alta carga de operaciones aritméticas y trigonométricas. Además, excepto en el caso de la ventana gaussiana, es necesario el uso de instrucciones atómicas. Estos *kernels* han sido analizados y optimizados siguiendo la metodología *Assess, Parallelize, Optimize, Deploy (APOD)* [17] y haciendo uso de la herramienta de *profiling* NVIDIA Nsight Compute. Se han aplicado las optimizaciones recomendadas por NVIDIA [17] para obtener el mejor rendimiento en la GPU, por ejemplo, minimizar la transferencia de datos entre la memoria principal y GPU, escoger el número de hilos por bloque y el número de bloques óptimo por cada kernel, y maximizar la ocupancia de los multiprocesadores evitando divergencias dentro de los *warps* de cada *kernel*, entre otras.

Emparejamiento de regiones. Para el emparejamiento se ha usado una aproximación de la distancia euclídea [9]. Este método traduce el cálculo de las distancias entre las regiones de una misma banda (línea 9, Alg. 1) en operaciones matriciales. Las matrices son ideales para su cómputo en GPU, ya que se procesan repitiendo la misma operación y conforman un conjunto de datos sin dependencias. Esto nos permite obtener una gran aceleración en esta etapa. Además, se ha empleado la biblioteca cuBLAS [18] siguiendo la implementación descrita en [9]. Una vez se tienen las distancias entre los descriptores espaciales, se evalúa la distancia espectral para aquellos que cumplan ciertos parámetros establecidos por el algoritmo HSI-MSER original y se guarda la pareja si la similitud espectral también es satisfactoria (líneas 10-13, Alg. 1).

Combinación de las bandas y registro. Ambas fases se ejecutan secuencialmente en la CPU debido a su bajo coste computacional, tal y como se mostrará en la siguiente sección (líneas 14-17, Alg. 1).

IV. Resultados

En esta sección se presentan las condiciones de experimentación y las imágenes utilizadas, así como los resultados obtenidos tanto en términos de tiempos de ejecución y rendimiento como en términos de capacidad de registro.

El algoritmo ha sido ejecutado sobre un ordenador con un procesador Intel Core i7 9700k de 3.6 GHz con 8 cores, 32 GB de RAM y una GPU NVIDIA GeForce RTX 3050. Por otro lado, se han usado g++ 11.3 y nvcc 12.1 como compiladores y Ubuntu 22.04.2 LTS como sistema operativo. Las versiones de las bibliotecas de NVIDIA como CUB y cuBLAS vienen determinadas por la versión 12.1 de CUDA. Los tiempos de ejecución y aceleraciones son el resultado de promediar 10 ejecuciones independientes. Por último, se han usado las opciones O3z arch=native"para generar código optimizado para la arquitectura correspondiente. El algoritmo HSI-MSER se ha ejecutado con los valores por defecto [4] a excepción del parámetro *min_area* tal y como se explicó en la sección III.

Se han empleado 14 imágenes para realizar los experimentos de este trabajo. Las características de cada una de ellas pueden encontrarse en el artículo del método original [4] y están disponibles para su descarga en [19]. Las imágenes *Pavia University* y *Pavia Centre* han sido tomadas por el sensor *Reflective Optics System Imaging Spectrometer* (ROSIS). Las restantes han sido capturadas por el *Airborne Visible / Infrared Imaging Spectrometer* (AVIRIS) [20]. Para *Jasper Ridge*, *Santa Barbara Front*, *Santa Barbara Line*, *Baraboo Hills* y *Crown Point* se cuenta con dos imágenes que han sido capturadas en distintos años y, por lo tanto, presentan cambios de escala, rotación, desplazamiento e incluso distorsión. Nos referiremos a estas imágenes como casos reales. La escala, ángulo y desplazamientos de referencia entre estas imágenes están disponibles en [4]. Para trabajar con las imágenes sin pareja, se genera una imagen objetivo aplicándole una escala y un ángulo a la original, lo que nos permite estudiar el registro bajo condiciones controladas.

A. Tiempos de ejecución y aceleraciones

La Tabla I presenta los tiempos de ejecución de la implementación original secuencial y de la implementación paralela propuesta en este trabajo utilizando distinto número de hilos para los casos reales. La versión CUDA ejecuta la selección de bandas, la descripción de regiones y el emparejamiento en la GPU, lo que le da una ventaja considerable frente a la versión secuencial. Las versiones CUDA + OpenMP, además

Tabla I: Tiempos de ejecución (segundos) de la implementación secuencial y de la implementación paralela propuesta con distinto número de hilos en los casos reales.

Imagen	Secuencial	CUDA	CUDA + OpenMP 2	CUDA + OpenMP 4	CUDA + OpenMP 8
Santa Barbara Front	38.24	7.22	5.54	4.64	4.27
Crown Point	39.96	11.38	7.05	5.78	5.21
Jasper Ridge	50.41	12.35	7.79	5.98	5.33
Santa Barbara Line	36.69	12.00	7.65	5.91	5.14
Baraboo Hills	36.07	12.00	7.32	5.91	4.99
Media	40.27	10.45	6.26	4.86	4.22

de ejecutar las partes mencionadas en la GPU, ejecutan la detección de regiones usando 2, 4 y 8 hilos con la ayuda de OpenMP. La Figura 3 muestra las aceleraciones de las implementaciones paralelas frente la implementación secuencial.

Los resultados muestran que gracias a la paralelización de la selección de bandas, la descripción de regiones y el emparejamiento en GPU, el tiempo medio se reduce de un valor promedio de 40.27s a 10.45s. Esto hace que se consiga una aceleración promedio de 4×. Si además añadimos la paralelización de la detección de regiones en OpenMP, en el caso con 8 hilos, se consigue una aceleración cercana a 10×. Para esta versión, el tiempo promedio para registrar dos imágenes hiperespectrales es de 4.22s, lo que constituye una mejora notable respecto a los 40.27s del método secuencial.

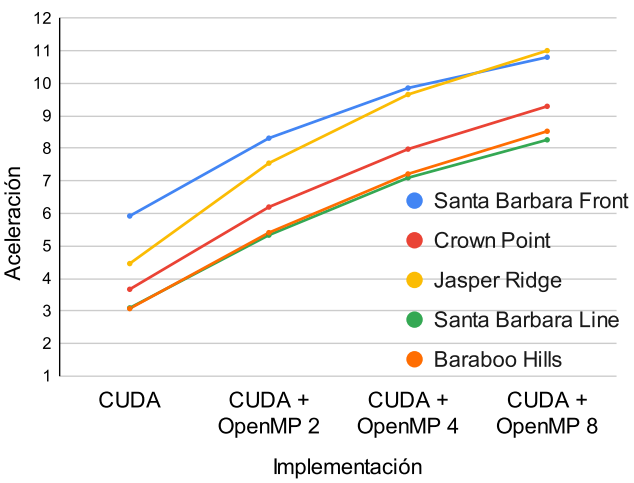


Fig. 3: Aceleraciones de la implementación paralela respecto a la secuencial en distintas imágenes. Se utilizan los tiempos de la Tabla I.

B. *Capacidad de registro*

Para comprobar cómo ha afectado la paralelización a la capacidad de registro, se han llevado a cabo varios experimentos. Se ha seguido el mismo procedimiento propuesto en el artículo original de HSI-MSER [4]. Este método se basa en ir aumentando y disminuyendo la escala original de cada imagen a la vez que se aplican rotaciones con el fin de aumentar el conjunto de imágenes de prueba. La escala se va aumentando en incrementos de 0.5× y se reduce en fracciones (1/2, 1/3, 1/4...). Para cada escala se aplica una rotación de 0 a 355°, en incrementos de 0.5°. Es decir, para cada incremento de escala estamos probando 72 rotaciones. Se considera que una escala ha sido correctamente registrada cuando se registran todos los ángulos asociados a la misma. Esto hace un total de 2592 casos probados por imagen.

La Tabla II resume el rango de escalas registradas con éxito para cada una de las imágenes utilizando las distintas versiones del algoritmo. Se indica entre paréntesis el número de escalas correctamente registradas para todos los 72 ángulos. Como se puede observar, el número de escalas correctamente registradas disminuye al aumentar el número particiones. El algoritmo secuencial es capaz de registrar una media de 19 escalas en comparación con las 10.89 de la versión más eficiente en términos de cómputo (CUDA + OpenMP 8). Al aumentar el número de hilos, aumenta el número de particiones horizontales en las que se dividen las bandas en la fase de extracción de regiones. Esta división de datos entre hilos deteriora la capacidad de detectar regiones en los bordes, reduciendo el número de emparejamientos correctos y, por tanto, impidiendo realizar un registro adecuado en casos con una gran diferencia de escala.

Habiendo estudiado los tiempos de ejecución, aceleraciones y el número de escalas correctamente registradas en comparación la implementación secuencial original, se selecciona la implementación CUDA + OpenMP 4 como una solución de compromiso entre aceleración y capacidad de registro.

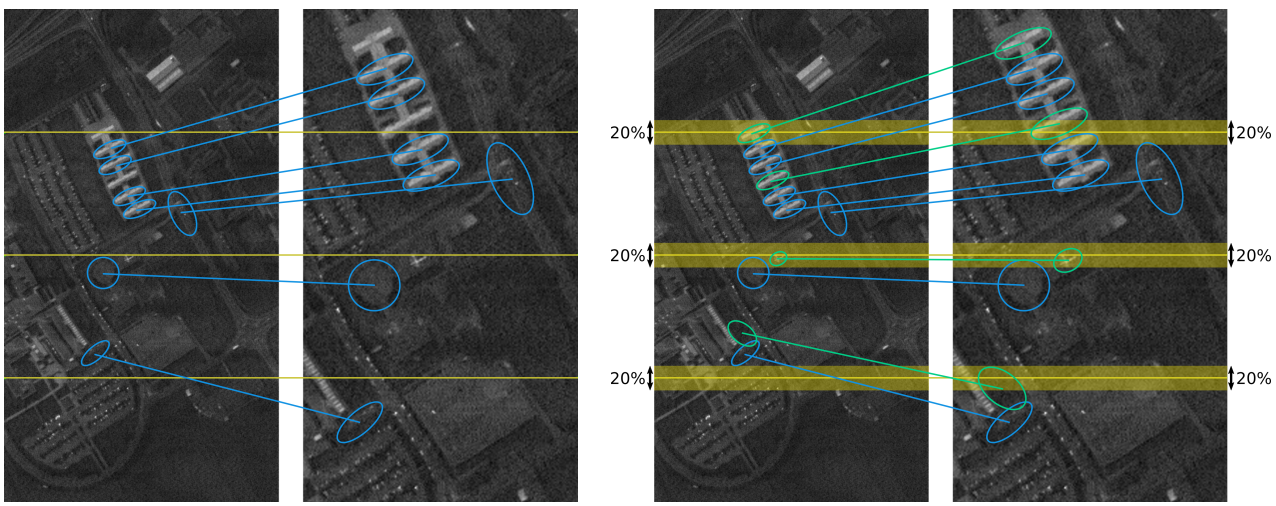
C. *Influencia de la adición de solapamiento de datos entre particiones sobre la calidad de registro*

Como se explicó en la Sección III, cada banda de la imagen se divide horizontalmente en tantas particiones como número de hilos disponibles. Cada hilo extrae regiones de forma paralela en una partición distinta. Para evitar la pérdida de regiones entre las particiones, efecto analizado en la sección anterior, se añade un solapamiento, es decir, una región frontera añadida a los datos que se asignan a cada hilo.

La Tabla III muestra el número de escalas correctamente registradas para distintos tamaños de solapamiento (0 %, 10 %, 20 % y 30 % respecto al tamaño de la partición asignada a cada hilo) con la implementación CUDA + OpenMP 4, que fue seleccionada como decisión de compromiso. Se puede observar que el número de escalas correctamente registradas incrementa al aumentar el solapamiento. Sin embargo, podemos ver que un solapamiento del 30 % obtiene los mismos resultados que el de 20 %, una media de 15.44 escalas en comparación con las 13.44 obte-

Tabla II: Capacidad de registro. Rango y número de escalas registradas con éxito para cada imagen para las distintas implementaciones del algoritmo.

Imagen	Secuencial [4]	CUDA + OpenMP 2	CUDA + OpenMP 4	CUDA + OpenMP 8
Pavia University	1/7× - 12.0× (29)	1/5× - 10.5× (24)	1/5× - 9.0× (21)	1/5× - 7.5× (18)
Pavia Centre	1/8× - 15.0× (36)	1/8× - 13.0× (32)	1/7× - 10.0× (25)	1/6× - 7.5× (19)
Indian Pines	1/3× - 8.0× (17)	1/3× - 6.5× (14)	1/2× - 5.0× (10)	1/1× - 4.0× (7)
Salinas	1/5× - 7.0× (17)	1/5× - 6.5× (16)	1/4× - 7.0× (16)	1/3× - 5.5× (12)
Jasper Ridge	1/3× - 7.0× (15)	1/2× - 3.5× (7)	1/2× - 3.0× (6)	1/2× - 3.0× (6)
Santa Barbara Front	1/3× - 7.0× (15)	1/2× - 7.0× (14)	1/2× - 5.5× (11)	1/2× - 5.5× (11)
Santa Barbara Line	1/5× - 7.0× (17)	1/4× - 6.0× (14)	1/4× - 5.0× (12)	1/3× - 4.0× (9)
Baraboo Hills	1/2× - 4.0× (8)	1/1× - 3.5× (6)	1/1× - 4.0× (7)	1/1× - 3.0× (5)
Crown Point	1/7× - 6.0× (17)	1/5× - 6.0× (15)	1/4× - 5.5× (13)	1/4× - 4.5× (11)
Media	19.00	15.78	**13.44**	10.89



(a) Sin solapamiento. (b) 20 % de solapamiento.

Fig. 4: Influencia de la adición de solapamiento en la detección de regiones en *Pavia University* con 4 particiones.

nidas sin solapamiento. Es por ello que se selecciona la implementación CUDA + OpenMP 4 con 20 % de solapamiento.

Los efectos del solapamiento se muestran en la Figura 4. En esta figura se han representado varias parejas de regiones sobre la imagen de *Pavia University* con una diferencia de escala de 1.75×. El área abarcada por el solapamiento está marcada en amarillo, en este caso, un 20 %. En la figura se representa cómo ciertas regiones, pintadas en color verde, no serían detectadas sin la adición de solapamiento al quedar divididas por las separaciones entre particiones.

D. Número de parejas y regiones

En la sección anterior hemos visto como el solapamiento nos permite mejorar el número de escalas correctamente registradas, haciendo que la implementación paralela obtenga resultados más próximos a la implementación secuencial en cuanto a capacidad de registro de escalas. En esta sección se analizan los motivos de esta mejora.

En la Tabla IV se comparan la implementación secuencial y las implementaciones seleccionadas, la implementación CUDA + OpenMP 4 con y sin solapamiento del 20 %, en base a diferentes métricas. Para esta comparación se ha registrado cada caso real sin aplicar ningún escalado o rotación adicional. En la primera fila de cada imagen se muestra el número de parejas de regiones encontradas durante la fase de emparejamiento. Se puede observar, que el número de parejas disminuye entre la versión secuencial y la versión con cuatro particiones. Esto es esperable, ya que como se explicó anteriormente, las particiones pueden dividir regiones en dos, dando lugar a su pérdida o dos regiones de menor tamaño, que luego pueden ser filtradas por el propio algoritmo o pueden no ser fiables. Sin embargo, se puede ver que con un 20 % de solapamiento se consigue recuperar gran parte del número de parejas de la versión secuencial. Este mismo efecto lo vemos en la segunda y tercera fila, donde se muestran el número de regiones extraídas para cada imagen. El mayor número de regiones obtenido en algunas imágenes se debe a que la misma o similar región es obtenida en las diferentes particiones debido al solapamiento.

En la cuarta fila, se muestra el porcentaje de parejas correctas. Para calcular esta métrica, se mide la distancia entre las regiones que conforman cada pareja tras aplicarle a la región de la imagen a registrar la transformación geométrica de referencia [4]. De esta manera, si la distancia tras aplicar la escala, ángulo y desplazamientos de referencia es mayor que 2 píxeles, la pareja es considerada como incorrecta. Se puede ver que el porcentaje de parejas correctas

Tabla III: Capacidad de registro. Rango y número de escalas registradas con éxito para cada imagen para distintos valores de solapamiento en la versión CUDA + OpenMP 4.

Imagen	0 %	10 %	20 %	30 %
Pavia University	1/5× - 9.0× (21)	1/4× - 10.0× (22)	1/4× - 11.0× (24)	1/4× - 11.0× (24)
Pavia Centre	1/7× - 10.0× (25)	1/7× - 10.5× (26)	1/6× - 11.5× (27)	1/6× - 11.5× (27)
Indian Pines	1/2× - 5.0× (10)	1/2× - 5.0× (10)	1/2× - 6.5× (13)	1/2× - 6.5× (13)
Salinas	1/4× - 7.0× (16)	1/4× - 7.0× (16)	1/4× - 7.0× (16)	1/4× - 7.0× (16)
Jasper Ridge	1/2× - 3.0× (6)	1/2× - 5.0× (10)	1/2× - 5.0× (10)	1/2× - 5.0× (10)
Santa Barbara Front	1/2× - 5.5× (11)	1/2× - 7.0× (14)	1/3× - 7.0× (15)	1/3× - 7.0× (15)
Santa Barbara Line	1/4× - 5.0× (12)	1/4× - 6.0× (14)	1/4× - 6.5× (15)	1/4× - 6.5× (15)
Baraboo Hills	1/1× - 4.0× (7)	1/1× - 4.0× (7)	1/1× - 4.0× (7)	1/1× - 4.0× (7)
Crown Point	1/4× - 5.5× (13)	1/3× - 6.0× (13)	1/3× - 5.5× (12)	1/3× - 5.5× (12)
Media	13.44	14.67	**15.44**	15.44

está, por lo general, en torno al 75 %. Este porcentaje mejora ligeramente al añadir solapamiento, indicando que se recuperan más regiones correctas que incorrectas con su adición. En el caso de *Crown Point*, el porcentaje de parejas correctas es considerablemente menor debido a la distorsión no lineal que presenta. Para poder evitar esto, se necesitan métodos capaces de calcular transformaciones geométricas con mayores grados de libertad que escala, ángulo y desplazamiento.

En la quinta fila se indica el tiempo de ejecución total (media de 10 ejecuciones del registro). Por último, se ha añadido el número de escalas que se registran para cada uno de estos casos reales si se varían escalas y ángulos de manera exhaustiva. Al comparar las distintas implementaciones, se puede observar que existe una relación clara entre el número de parejas detectadas y el número de escalas registradas. Cuando la diferencia de escala es demasiado grande, el registro falla por no contar con parejas para calcular la transformación o porque las pocas parejas obtenidas son incorrectas. Gracias al solapamiento del 20 %, se recuperan regiones y parejas que se encontraban en los bordes de las particiones, consiguiendo aumentar el número de escalas registradas.

No obstante, la versión con solapamiento requiere de un mayor tiempo de ejecución, ya que al añadirle un solapamiento a cada partición, estamos aumentando el espacio donde debemos buscar regiones. Por consiguiente, tal y como se ha explicado, se obtiene un mayor número de regiones, que deriva en un aumento del tiempo de cómputo de las siguientes etapas (descripción, emparejamiento, combinación y registro). En el caso más desfavorable, para las imágenes *Jasper Ridge*, se necesitan 5.72s frente a los 5.22s de la versión sin solapamiento, pasando de registrar 10 a 6 escalas, respectivamente. Si lo comparamos con la versión secuencial, esta necesita 43.95s.

E. Aceleración por etapas

Las Tablas V y VI presentan los tiempos de ejecución y aceleraciones para las implementaciones secuencial y la seleccionada (CUDA + OpenMP 4 con 20 % de solapamiento) para el caso real de mayor tamaño (*Baraboo Hills*) y para el de menor tamaño (*Santa Barbara Front*). Como se puede observar, las fases más costosas son la detección, descripción y emparejamiento de regiones. En ellas es donde se ha conseguido mejores aceleraciones, destacando especialmente la fase de emparejamiento que consigue una aceleración próxima a 21×. La aproximación de la distancia euclídea en cálculos matriciales para el emparejamiento de las regiones, nos permite explotar adecuadamente el modelo de paralelismo y arquitectura de la GPU. Las otras dos fases que se ejecutan en GPU, selección de bandas y descripción de regiones, han obtenido aceleraciones de hasta 2.68× y 16.19×, respectivamente. La fase de detección de regiones está paralelizado usando OpenMP y consigue una aceleración de hasta 4.41× usando 4 hilos y solapamiento del 20 %. En el cómputo total, se obtiene una aceleración de hasta 7.87×. Recordemos que se podría conseguir una aceleración mayor aumentando el número de hilos que se usan en la detección en el caso de que no fuera necesario registrar este tipo de escalas.

Si comparamos los resultados entre ambas imágenes, se puede ver que en la detección de regiones se produce una aceleración mayor en *Baraboo Hills*, debido a su mayor tamaño, que mejora la eficiencia de la paralelización en esa fase. Sin embargo, en la descripción pasa justo lo contrario. Esto se debe a que *Santa Barbara Front*, a pesar de ser la imagen más pequeña de las utilizadas, se obtienen una gran cantidad de regiones (ver Tabla IV). Este desacoplamiento entre el tamaño y el número de regiones que se detectan es lo que produce que *Santa Barbara Front* sea de las imágenes que más aceleración experimentan, como se vio en la Figura 3, a pesar de ser la más pequeña.

V. Conclusiones

En este artículo se ha presentado una versión paralela del algoritmo HSI-MSER de registro de imágenes hiperespectrales basado en características. Este algoritmo adapta MSER y SIFT para su uso en imágenes hiperespectrales, aprovechando la información espectral de forma eficiente.

La implementación propuesta paraleliza el algoritmo haciendo uso los distintos núcleos disponibles en la CPU así como de la GPU, obteniendo una considerable reducción de los tiempos de ejecución cuando la comparamos con la versión secuencial original. La implementación propuesta ha sido evaluada con

Tabla IV: Número de parejas de regiones, porcentaje de parejas correctas, número de regiones, número de escalas registradas correctamente y tiempos de ejecución de la versión secuencial y CUDA + OpenMP 4 sin solapamiento y con 20 % de solapamiento.

Imagen		Secuencial	CUDA + OpenMP 4	CUDA + OpenMP 4 + 20 %
Jasper Ridge	Parejas	253	187	236
	Regiones imagen 1	32538	33379	37879
	Regiones imagen 2	32625	33387	38128
	Porcentaje de parejas correctas	0.742	0.778	0.786
	Tiempo (s)	43.95	5.22	5.72
	Número de escalas registradas	15	6	10
Santa Barbara Front	Parejas	618	531	630
	Regiones 1	27601	28578	32843
	Regiones 2	31688	32618	37379
	Porcentaje de parejas correctas	0.727	0.732	0.756
	Tiempo (s)	33.70	3.89	4.28
	Número de escalas registradas	15	11	15
Santa Barbara Line	Parejas	977	825	988
	Regiones 1	24420	25335	28861
	Regiones 2	25703	26812	30261
	Porcentaje de parejas correctas	0.773	0.764	0.774
	Tiempo (s)	33.78	5.17	5.64
	Número de escalas registradas	17	12	15
Baraboo Hills	Parejas	133	110	137
	Regiones 1	24413	25317	28876
	Regiones 2	22223	23077	26261
	Porcentaje de parejas correctas	0.756	0.771	0.769
	Tiempo (s)	32.83	5.00	5.38
	Número de escalas registradas	8	7	7
Crown Point	Parejas	693	572	704
	Regiones 1	27937	28764	33281
	Regiones 2	27412	28283	32607
	Porcentaje de parejas correctas	0.219	0.24	0.24
	Tiempo (s)	36.11	5.01	5.49
	Número de escalas registradas	17	13	12

Tabla V: Tiempos de ejecución en segundos y aceleración de la versión CUDA + OpenMP 4 con un 20 % de solapamiento respecto a la versión secuencial para *Baraboo Hills*.

Fase	Secuencial (s)	CUDA + OpenMP 4 + 20 % (s)	Aceleración
Leer las imágenes	0.92	0.92	—
Copiar imágenes a GPU	—	0.28	—
Selección de bandas	0.32	0.12	2.68×
Detección de regiones	8.29	1.88	4.41×
Descripción de regiones	14.99	1.02	14.66×
Emparejamiento	8.30	0.40	20.94×
Registro	0.01	0.01	—
Total	32.83	5.38	6.10×

imágenes accesibles públicamente para las que hay datos disponibles en la literatura. Para la evaluación se han tenido en cuenta tanto la aceleración respecto a la implementación secuencial original como la capacidad de registro. Se ha comprobado que, si bien al ejecutar la versión paralela del registro se producen

Tabla VI: Tiempos de ejecución en segundos y aceleración de la versión CUDA + OpenMP 4 con un 20 % de solapamiento respecto a la versión secuencial para *Santa Barbara Front*.

Fase	Secuencial (s)	CUDA + OpenMP 4 + 20 % (s)	Aceleración
Leer las imágenes	0.48	0.48	—
Copiar imágenes a la GPU	—	0.16	—
Selección de bandas	0.17	0.10	1.59×
Detección de regiones	3.43	1.05	3.28×
Descripción de regiones	18.59	1.15	16.19×
Emparejamiento	10.97	0.52	21.26×
Registro	0.06	0.06	—
Total	33.70	4.28	7.87×

pérdidas de capacidad de registro, especialmente para imágenes con diferencias de escala considerables, es posible mitigar este efecto mediante el solapamiento de datos entre hilos. De este modo, se consigue registrar hasta 7.87× más rápido con la versión con 4 hilos y solapamiento, en comparación con el algoritmo secuencial, sin una pérdida sustancial de capacidad de registro para diferencias de escala extremas entre imágenes.

Como trabajo futuro, se propone investigar en mayor grado la paralelización de la fase de detección de regiones. Se estudiará la posibilidad de construir el árbol basado en el algoritmo *Union Find* de forma paralela. Además, será necesario probar los algoritmos en entornos de producción reales que exigen el registro de imágenes de mayor tamaño obtenidas en mayor variedad de condiciones de captura.

Agradecimientos

El presente trabajo ha sido financiado por MCIN/AEI/10.13039/50110001103 a través de los proyectos TED2021–130367B–I00, FJC2021-046760-I y PID2019–104834GB–I00, por la Xunta de Galicia - Consellería de Cultura, Educación, Formación Profesional e Universidades mediante las ayudas 2019-2022 ED431G-2019/04 y ED431C-2022/16, y por el Fondo Europeo de Desarrollo Regional (FEDER). También se ha recibido financiación de la Junta de Castilla y León (proyecto PROPHET–II, VA226P20).

Referencias

[1] Xiaoxin Guo, Zhiwen Xu, Yinan Lu, and Yunjie Pang, "An application of Fourier-Mellin transform in image registration," in *Proceedings of the The Fifth International Conference on Computer and Information Technology*, USA, 2005, CIT '05, p. 619–623, IEEE Computer Society.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool, "SURF: Speeded up robust features," in *European Conference on Computer Vision*, 2006.

[3] Àlvaro Ordóñez, Dora B. Heras, and Francisco Argüello, "Comparing area-based and feature-based methods for co-registration of multispectral bands on GPU," in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, 2021, pp. 1575–1578.

[4] Álvaro Ordóñez, Álvaro Acción, Francisco Argüello, and Dora B. Heras, "HSI-MSER: Hyperspectral image registration algorithm based on MSER and SIFT," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 12061–12072, 2021.

[5] Antonio Plaza, David Valencia, Javier Plaza, and Pablo Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *Journal of Parallel and Distributed Computing*, vol. 66, no. 3, pp. 345–358, 2006.

[6] Álvaro Ordóñez, Dora Blanco Heras, and Francisco Argüello, "Multi-GPU registration of high-resolution multispectral images using HSI-KAZE in a cluster system," *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*, pp. 5527–5530, 2022.

[7] Jiri Matas, Ondrej Chum, Martin Urban, and Tomás Pajdla, "Robust wide-baseline stereo from maximally stable extremal regions," *Image and vision computing*, vol. 22, no. 10, pp. 761–767, 2004.

[8] David G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, nov 2004.

[9] Álvaro Ordóñez, Francisco Argüello, Dora B. Heras, and Begüm Demir, "GPU-accelerated registration of hyperspectral images using KAZE features," *The Journal of Supercomputing*, vol. 76, no. 12, pp. 9478–9492, Dec 2020.

[10] "NVIDIA: CUB library," 2023, Accessed 22 May 2023.

[11] Robert Endre Tarjan, "Efficiency of a good but not linear set union algorithm," *J. ACM*, vol. 22, no. 2, pp. 215–225, apr 1975.

[12] Richard J Anderson and Heather Woll, "Wait-free parallel algorithms for the union-find problem," in *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, 1991, pp. 370–380.

[13] Stefano Allegretti, Federico Bolelli, Michele Cancilla, and Costantino Grana, "A block-based union-find algorithm to label connected components on GPUs," in *Image Analysis and Processing–ICIAP 2019: 20th International Conference, Trento, Italy, September 9–13, 2019, Proceedings, Part II 20*. Springer, 2019, pp. 271–281.

[14] Jun Chen, Qiang Yao, Houari Sabirin, Keisuke Nonaka, Hiroshi Sankoh, and Sei Naito, "An optimized union-find algorithm for connected components labeling using GPUs," *arXiv preprint arXiv:1708.08180*, 2017.

[15] Hailiang Xu, Siqi Xie, and Fan Chen, "Fast MSER," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 3380–3389.

[16] OpenMP, "The OpenMP API specification for parallel programming," `https://www.openmp.org/`, [En línea, accedido: 07-Jun-2023].

[17] "CUDA Best Practices Guide," `https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html`, [En línea, accedido: 18-Jun-2023].

[18] NVIDIA, "cuBLAS library," `https://developer.nvidia.com/cublas`, [En línea, accedido: 07-Jun-2023].

[19] Álvaro Ordóñez, Francisco Argüello, and Dora B. Heras, "Hyperspectral image repository," `https://gitlab.citius.gal/hiperespectral/RegistrationRepository`, [En línea, accedido: 07-Jun-2023].

[20] "AVIRIS database," `https://aviris.jpl.nasa.gov/dataportal`, [En línea, accedido: 07-Jun-2023].

[21] Sabyasachi Maiti and Amit K. Bhattacharya, "Shoreline change analysis and its application to prediction: A remote sensing and statistics based approach," *Marine Geology*, vol. 257, pp. 11–23, 2009.

[22] Amy Lowe, Nicola Harrison, and Andrew P. French, "Hyperspectral image analysis techniques for the detection and classification of the early onset of plant disease and stress," *Plant Methods*, vol. 13, no. 1, pp. 80, Oct 2017.

Modelado del Departamento de Urgencias para la gestión del staff durante epidemias: chikungunya.

Ramona Galeano[1], Dolores Rexachs[1], Alvaro Wong[1], Remo Suppi[1], Emilio Luque [1], Eva Bruballa [2], Francisco Epelde [3]

Resumen— **Los departamentos de urgencias en sistemas de salud, son sistemas complejos, donde una de sus misiones es dar soporte en momento de emergencias y desastres. Estos episodios varían el funcionamiento y requieren su adaptación en función del tipo de emergencias. Una de estas situaciones son las epidemias que alteran su funcionamiento, ya que ocurren de forma asíncrona y cambian el flujo habitual de funcionamiento, y por ello resulta muy útil y es importante disponer de herramientas que ayuden en la toma de decisiones. La capacidad de representar, simular y predecir con mayor precisión el desempeño de los departamentos de urgencias será invaluable para los administradores del hospital y los responsables de la toma de decisiones para resolver problemas de gestión. Para este fin, se parte de un simulador basado en agentes que analiza la gestión de un servicio de urgencias. En el presente trabajo se describe y muestra como adaptar el modelo original para un simulador de un servicio de urgencias que simula el comportamiento cotidiano, incorporando una nueva funcionalidad para que que ayude a controlar y gestionar epidemias como la chikungunya. El objetivo es ayudar en la toma de decisiones a los médicos y administradores para la gestión del departamento de urgencias, evaluar soluciones para reducir el tiempo de espera de los pacientes y la gestión de recursos como médicos y enfermeras.**

Palabras clave— **chikungunya, simulation, agent-based model.**

I. Introducción

La s enfermedades trasmitidas por vectores ocurren cuando el agente que produce la enfermedad es trasmitida al humano por un portador animado no humano denominado vector. En la cadena de transmisión intervienen entonces tres factores: un hospedero, por lo general una persona enferma, un vector invertebrado que propaga la enfermedad, generalmente un artrópodo y el agente biológico que puede ser un virus, una bacteria o un parásito [1].

La transmisión por vectores se refiere a la forma en que ciertas enfermedades infecciosas se propagan de un organismo a otro a través de organismos vivos, conocidos como vectores. En el caso de las enfermedades transmitidas por vectores, el vector actúa como intermediario en la transmisión de la enfermedad, en este caso los mosquitos actúan como vectores.

[1]Computer Architecture and Operating Systems, Universitat Autónoma de Barcelona (UAB). Spain
, e-mail: RamonaElizabeth.Galeano@autonoma.cat ,Dolores.Rexachs@uab.cat,Alvaro.Wong@uab.cat, emilio.luque@uab.es,Remo.Suppi@uab.cat
[2]Escuela Universitaria de Informatica, Escuelas Universitarias Gimbernat, e-mail: eva.bruballa@eug.es
[3]Medicine Department, Hospital Universitari Parc Tauli, (UAB), e-mail: fepelde@tauli.cat

La transmisión directa de enfermedad de persona a persona ocurre cuando un individuo infectado transmite directamente el patógeno a otra persona. Esto puede ocurrir a través de diferentes mecanismos, como el contacto directo de piel a piel, el contacto con fluidos corporales infectados (como la saliva, la sangre, las secreciones respiratorias) o a través de la transferencia de patógenos mediante la tos, los estornudos o las gotas respiratorias expulsadas al hablar.

La transmisión indirecta de enfermedad ocurre a través de objetos y como hemos visto la transmisión por vectores que es a través de un mosquito que pica a la persona infectada y luego a una persona sana, en el caso de la chikungunya la transmisión es de forma indirecta, por vectores.

Para el caso de estudio hemos elegido la chikungunya, la chikungunya es un virus transmitido a través de la picadura de mosquitos Aedes aegypti, no se transmite de persona a persona, el virus necesita un vector que es el mosquito para su transmisión. La chikungunya ha causado millones de casos de la enfermedad en distintas partes del mundo, generalmente en zonas tropicales y templadas. La enfermedad suele ser debilitante y causa grandes epidemias y a la vez graves consecuencias económicas a los paises [2].

En Paraguay en el año 2023 se manifestó un brote de chikungunya lo que comportó una alerta sanitaria, 2401 casos de pacientes hospitalizados de los cuales 69 graves con meningo-encefalitis entre enero y febrero de ese año [3].

El grupo de edad más afectado es el de 20 a 39 años, concentrando el 32 por ciento de los casos registrados según los datos publicados por el ministerio de salud hasta febrero del año 2023. El cincuenta y siete por ciento son mujeres, los casos se concentran en determinadas zonas [3].

Algunos de los síntomas de los pacientes con chikungunya son fiebre y dolor en las articulaciones de moderado a intenso, erupción temprana (erupciones en la piel) y además, cefalea, conjuntivitis y dolor corporal difuso o artritis simétrica con inflamación y rigidez. Las señales de advertencia son somnolencia, dolor abdominal, vómitos, sangrado de las mucosas y debilidad extrema [3].

El chikungunya es en algunos casos incapacitante en cuanto al dolor, especialmente el dolor en las articulaciones y una persona infectada puede no caminar en un momento dado a causa de la enfermedad. En

ocasiones, una vez superado el cuadro viral, la persistencia de algunas secuelas en las articulaciones puede prolongarse durante un periodo prolongado [3].

Los casos más graves a menudo ocurren en recién nacidos, personas mayores de 65 años y aquellos con condiciones médicas subyacentes. Los signos y síntomas adicionales de la enfermedad incluyen artritis, con articulaciones que a menudo muestran sensibilidad e hinchazón, tenosinovitis, sarpullido y mialgia, particularmente en la parte baja de la espalda y los músculos de las piernas. Además de estas características clínicas, las manifestaciones cardíacas y neurológicas graves y, en algunos casos, la muerte, se han asociado con la infección por chikungunya [2].

Los pacientes con síntomas moderados y que en general no van a requerir hospitalización. En personas con otras patologías, personas mayores o con sistemas inmunológicos debilitados, el chikungunya puede provocar complicaciones y enfermedad grave. En casos de enfermedad grave, es posible que sea necesario hospitalizar a los pacientes de chikungunya.

Los continuos factores antropogénicos que promueven la aparición y propagación del chikungunya y otras enfermedades transmitidas por Aedes aegypti sugieren que la epidemia mundial actual continuará propagándose a áreas previamente no afectadas durante algún tiempo [4].

Es importante ayudar a la gestión de los servicios de urgencias cuando se prevé una epidemia ocasionado por una enfermedad contagiosa trasmitida por vectores. En ese caso es importante predecir como va a ser la llegada de pacientes, y disponer de una herramienta que de soporte a la gestión de recursos.

Dentro del grupo de investigación se dispone de un simulador del departamento de urgencias de un hospital, que ha sido validado y configurado para un hospital en España y se desea adaptar y extender su funcionalidad para simular el departamento de urgencias y ayude a la toma de decisiones cuando existen motivos fundados de la llegada de una epidemia local, de una enfermedad viral de transmisión indirecta por vectores que tiene potencial de ocasionar una pandemia.

Como hipótesis de partida se considera adaptar el simulador [5], ya validado, con los datos de otro hospital (en este caso a un hospital de Paraguay), que permita modelar el funcionamiento de los departamentos de urgencias, ayude a planificar y gestionar epidemias como la chikungunya, que pueda ayudar a los médicos y administradores como un sistema de apoyo a la toma de decisiones. El objetivo es comprender mejor el proceso y evaluar soluciones para reducir el tiempo de espera del paciente y ayude a la gestión de los diferentes recursos como camas, doctores, enfermeras.

El documento está organizado de la siguiente manera. En la Sección II se describen los trabajos relacionados y en la sección III se presenta la adaptación del modelo del departamento de urgencias. La sección IV describe la metodología utilizada. La sección V los datos de entrada experimentales y finalmente en la sección VI las conclusiones y el trabajo futuro.

II. Trabajos relacionados

A continuación se muestran diversos trabajos sobre la propagación y análisis del chikungunya que tienen en cuenta como afecta esta enfermedad al volumen de entrada de pacientes en el servicio de urgencias de un hospital:

En el trabajo de Ortigoza et al. [6], se propuso el uso de autómatas celulares definidos en una malla triangular no estructurada para modelar y simular la propagación de chikungunya. Los autómatas celulares se derivaron de un modelo SEIR (el nombre del modelo proviene de las siglas de los cuatro grupos principales en los que se dividen las personas en el modelo: Susceptibles (S), Expuestos (E), Infectados (I) y Recuperados (R) personas que no pueden volver a infectarse. Este modelo se utiliza para comprender y predecir la propagación de enfermedades infecciosas en una población). Las probabilidades de transmisión de humano a vector y de vector a humano se definieron globalmente utilizando una relación entre las tasas de transmisión y la proporción de mosquitos a humanos; localmente, las probabilidades se modificaron por la cantidad de humanos y mosquitos infectados dentro del vecindario. Experimentos numéricos demostraron que la movilidad humana aumenta el tamaño máximo del número de humanos infectados y reduce el tiempo para alcanzar este máximo.

En el trabajo de Osorio et al. [7], se presenta un modelo de simulación basado en ecuaciones diferenciales ordinarias no lineales, utilizando el término de incidencia, la capacidad vectorial de Aedes aegypti y el efecto sinusoidal de la temperatura sobre las probabilidades de transmisión del virus a las personas. En este trabajo se determinó y simuló la capacidad vectorial, el umbral epidémico y el número de reproducción en términos de temperatura y tiempo. Además, se simuló la población infectada y la población de mosquitos hembra portadores del virus.

El trabajo de Maneerat et al. [8], presentó un modelo que tiene como objetivo producir datos estadísticos sobre el comportamiento de los mosquitos y la dinámica de la población que son difíciles de obtener a través de estudios de campo, como las densidades de población en diversas condiciones geográficas y climáticas. El modelo simula mosquitos adultos como agentes que interactúan con su entorno local. Este último proporciona recursos para su desarrollo biológico y también puede limitar sus comportamientos de huida o puesta de huevos. Las variaciones en los entornos ambientales, como el uso de la tierra y el clima, permiten explorar la dependencia de la dinámica de la población de mosquitos en el contexto.

Un estudio de comportamientos de mosquitos simulados revela la capacidad del modelo para producir el ciclo de vida realista del mosquito. También se explora la distancia de vuelo de la cohorte de mosquitos en varias ciudades y paisajes. Este último re-

presenta un barrio en desarrollo procesado a través de un Sistema de Información Geográfica (SIG). Los resultados iniciales revelan una relación significativa entre la topología urbana, las densidades humanas y el vuelo de los mosquitos adultos [8].

En el estudio de Ruiz Moreno et al. [9], se desarrolló un modelo basado del virus por parte de un individuo. Este estudio combina un modelo estocástico basado en el clima de la dinámica de la población de mosquitos con un modelo epidemiológico para identificar ventanas temporales de riesgo epidémico. Este modelo se ejecutó con datos de temperatura de diferentes lugares para estudiar la sensibilidad geográfica del potencial epidémico. Indican que en lugares con marcadas variaciones estacionales de temperatura, una temporada de riesgo de epidemia coincidía con la época del año en que las poblaciones de mosquitos sobreviven y crecen.

Los resultados del trabajo de Ruiz Moreno et al. [9], sugiere fuertemente que en el caso de Chikungunya en los EE.UU, inicialmente surgirían regiones endémicas y epidémicas, definidas principalmente por factores ambientales que controlan los ciclos anuales de población de mosquitos. Estas regiones deben ser identificadas para planificar diferentes medidas de intervención. Además, reduciendo la proporción vector, humano puede reducir la probabilidad y la magnitud de los brotes en regiones con fuertes patrones estacionales de temperatura. Este es el primer modelo que considera el riesgo de Chikungunya en los EE. UU. y se puede aplicar a otras enfermedades transmitidas por vectores.

El trabajo de Dommar et al. [10], se desarrolló un modelo basado en agentes en el que cada individuo está representado explícitamente, y las poblaciones de vectores están vinculadas a las estimaciones de precipitación en un entorno tropical. El modelo se implementa tanto en redes libres de escala como regulares. Se analiza la transmisión espacio temporal de chikungunya y se investigan los propagadores silenciosos asintomáticos dentro de la población en el contexto de la implementación de restricciones de viaje durante un brote. La prevención del movimiento de individuos sintomáticos es un mecanismo insuficiente para detener la propagación de la enfermedad, que puede transmitirse fácilmente a los nodos vecinos a través de individuos subclínicos. El modelo fue desarrollado con NetLogo.

El trabajo de Orosco et al. [11], se centra en desarrollar un modelo híbrido basado en agentes para comprender la dinámica de transmisión de enfermedades como la chikungunya, teniendo en cuenta algunas características geoespaciales. Este estudio ha demostrado que un modelo ABM híbrido ayuda a simular la propagación del virus de la chikungunya. Y aunque todavía hay muchas suposiciones y simplificaciones sobre el comportamiento real de las enfermedades para validar y verificar los modelos, se puede suponer el comportamiento general de este tipo de virus ya que, según esta investigación, la cantidad de humanos en estados susceptibles, expuestos, infectados, y recuperado se asemeja al comportamiento de un brote de virus real.

El trabajo de Stephan Karl et al. [12], podría aplicarse a otras enfermedades virales transmitidas por vectores, como el chikungunya y el dengue, también propagado por Aedes aegypti y, por reparametrización del submodelo vectorial, los virus del chikungunya propagados por Aedes albopictus.

El trabajo de Jindal et al. [13], proporcionó un mecanismo adecuado para el análisis cuantitativo de las epidemias transmitidas por mosquitos. El estudio de caso demostró cómo el modelo podría rastrear la evolución de la infección con precisión, dado que se proporcionan datos de una epidemia anterior en un lugar similar. En dicho trabajo se presentó un modelo generalizado basado en agentes que supera algunas limitaciones al integrar información geográfica (SIG) y datos censales para dar cuenta del movimiento espacial de infecciones y datos climáticos para capturar la naturaleza temporal de una epidemia. Captura las interacciones desorganizadas de huéspedes y vectores a microescala mediante el modelado explícito de cada humano y mosquito para simular las complejas trayectorias de los brotes de enfermedades (incluso aquellas que aún no han ocurrido) y permite probar la eficacia de varias políticas de salud pública.

En épocas de epidémia, como es el caso de la chikungunya, algunos casos pueden ser muy graves. Es por ello que es necesario remodelar el sistema sanitario para que los poco graves no lleguen al hospital (atención primaria) y reservar el hospital para los casos mas graves que pueden ser muchos.

Algunos de nuestros trabajos en el área de urgencias son:

- Desarrolo de un simulador para urgencias con la participación del equipo de urgencias del Hospital Parc Tauli e identificación de los agentes activos, los agentes pasivos y el entorno, y se crea una simulación usando NetLogo [14].
- Optimización del desempeño del departamento de emergencias y búsqueda para encontrar la configuración óptima del personal de emergencia, un problema multidimensional y multiobjetivo y propuesta de un índice para minimizar la estancia del paciente en el departamento de emergencias. Los resultados obtenidos utilizando esquemas alternativos de Monte Carlo y kmeans fueron prometedores [15].
- Definición de un marco de aplicación basado en capas para descubrir el conocimiento de un sistema de departamento de emergencias mediante la simulación, comportamientos a nivel micro de sus componentes [5].
- Adaptación del uso de una herramienta de simulación para para diseñar y realizar ensayos clínicos virtuales y estudiar la transmisión por contacto de MRSA entre pacientes hospitalizados [16].

La presente propuesta se diferencia de los trabajos antes mencionados en que todos estos trabajos se

centran en la propagación de la enfermedad dentro y fuera del hospital, en nuestro caso nos centramos la gestión de los recursos humanos y materiales de los departamentos de urgencias en casos de epidemias. El simulador del modelo de gestión de epidemias del departamento de urgencias permitirá a los gerentes analizar y evaluar cuando ocurre o se prevé que va ocurrir un evento excepcional como una epidemia causada por un virus que se transmite a través de vectores como son los mosquitos, para verificar posibles soluciones como agregar más personal médico, camas, laboratorios y otros dispositivos. Y también evaluar la efectividad de diferentes combinaciones de escenarios.

III. Adaptación del modelo del departamento de urgencias

Esta sección presenta la adaptación de un modelo para el servicio de urgencias como caso de estudio se ha elegido la epidemia de chikungunya. El objetivo general es proponer un modelo que permita ampliar la funcionalidad del simulador para adaptarlo a los cambios en el funcionamiento del servicio de emergencias cuando se presentan situaciones excepcionales, como la epidemia de chikungunya, de tal manera que ayude en la planificación y gestión del servicio.

El primer paso del trabajo consiste en realizar un modelo conceptual del funcionamiento del sistema, a partir del cual se elabora el modelo computacional que permitirá simular el sistema. Las extensiones al diseño original se continuarán desarrollando sobre el entorno de simulación Netlogo [17].

A. Agentes activos

Durante la recogida de la información, son los agentes activos en representación de las personas y entidades que actúan por su iniciativa.

- Pacientes: Son los individuos en el sistema, se tendrá en cuenta su nivel de gravedad y se añadirá una variable de estado si está infectado o no, se hizó el test o no, tipo de arbovirosis.
- Personal de admisiones: Personal que atiende al paciente en primera instancia para solicitar una cita y que actualiza sus datos y/o solicita las apertura o consulta de su historia clínica.
- Doctor: Interactúan con los pacientes para diagnosticarlos y tratarlos, se tendrán en cuenta su nivel de experiencia en pacientes con chikungunya y clasificarlos en *junior y senior*.
- La enfermera de triaje: Es quien llama al paciente para realizar la preconsulta. Identifica el nivel de gravedad del paciente, se le activa un sensor para saber el tiempo de estancia de este paciente para observar cuanto tiempo se encuentra en el servicio si se requiere priorizar o no en la atención.
- Personal de Laboratorio: Son las personas que realizan las pruebas complementarias si es necesario.
- Enfermera: atienden al paciente, toman muestras y las envían al laboratorio correspondiente.

B. Variables de estado

Los agentes se desplazan de un lugar a otro interactuando con otros agentes. Durante este tiempo, cada agente cambia de estado debido a las interacciones. Una máquina de estados representa este comportamiento, y para ello se ha escogido una máquina de Moore probabilística.

C. Salida

Cada estado puede tener una salida diferente. La salida de un simulador basado en agentes incluye la información de estado (sensores) del departamento de emergencias; algunas de las salidas son la longitud de estancia (LOS), el tiempo de espera (Lowt) para cada etapa (por ejemplo, tiempo de espera para solicitud de servicio: wtrs, tiempo de admisión: ta, tiempo de espera en admisión: wta, tiempo de espera en enfermería: wtn, tiempo de atención de enfermería: twnc, tiempo de espera en tratamiento médico:wtdt, tiempo de atención en tratamiento médico: twmd, entre otros), destino, edad, nivel de agudeza, infectados, sintomáticos, vacunados. De esta forma, el simulador nos proporciona directamente información sobre el comportamiento del departamento simulado, sino ésta se obtiene a través del análisis cruzado de diferentes escenarios de simulación (ejemplo de estas salidas son el porcentaje de los diferentes recursos como doctores, enfermeras, personal de admisión, etc.).

D. Agentes pasivos

Los individuos pasivos no actúan solos sino que reaccionan a las acciones de los individuos activos. Estos agentes no tienen la misma complejidad que los agentes activos, ya que no son entidades que se muevan por el departamento. Un ejemplo de un agente pasivo es un sistema informático.

IV. Método

Los departamentos de urgencias pueden ser muy cambiantes y fluidos, los pacientes pueden llegar llegar de una manera muy inesperada en cualquier momento. Es posible predecir las probabilidades estadísticamente pero hay un nivel alto de incertidumbre en caso de que sucedan situaciones inesperadas. Una subida inesperada de los pacientes puede afectar todo el comportamiento del departamento de urgencias.

Tras la llegada del paciente al departamento de urgencias y el registro realizado por el personal de admisión, en función de la gravedad de su situación de triaje, se categoriza a los pacientes teniendo en cuenta su nivel de gravedad. Hay cinco valores diferentes, el nivel 1 es la situación más crítica y el nivel 5 es la más leve.

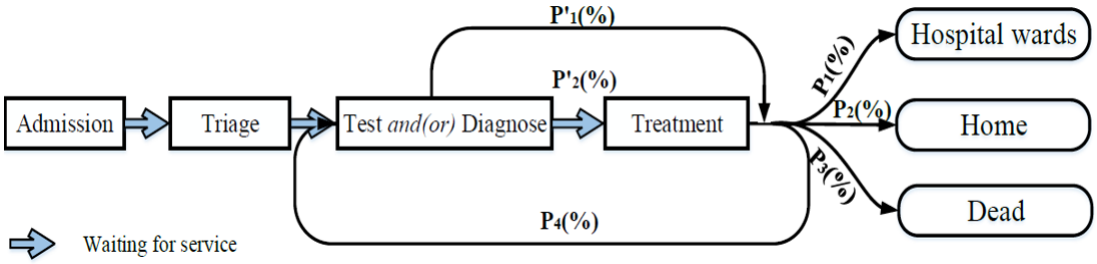


Fig. 1: Tiempo en el departamento de urgencias

Los servicios cuentan con diferentes áreas: área de ingresos, área de triaje, área de diagnóstico-tratamiento, salas de espera, etc.

El departamento de urgencias de este hospital tiene dos áreas, A y B. El área A está ocupado por los pacientes más urgentes y está formado por *careboxes*. Los pacientes atendidos en zona A permanecerán en su propio box de cuidados durante todo el proceso de diagnóstico y tratamiento. El área B es para pacientes con un nivel de gravedad de 4 y 5, hay diversas salas de atención en los que los médicos y las enfermeras interactúan con los pacientes, y una gran sala de espera en la que permanecerán todos los pacientes mientras no tienen interacción con el personal del departamento de urgencias. Estas dos áreas (A, B) tienen su propio médico y enfermera, que trabajan por separado pero comparten los mismos recursos de servicio, por ejemplo, rayos X, tomografía computarizada, ecografías, laboratorios para realizar análisis, etc.

Después del triaje, los pacientes con nivel de gravedad diagnosticada 1, 2 y 3 son tratados por separado y asignados al área A, y los pacientes 4 y 5 se tratan en el área B.

El primer paso para adaptar el simulador al hospital será especificar la configuración concreta del entorno, será preciso especificar el número de boxes del área A y B, el número de salas de espera, de laboratorios para realizar análisis,rayos X.

El escenario adoptado para esta etapa inicial es simular los pacientes que transitan por el servicio de urgencias, teniendo en cuenta el aumento del las entradas de pacientes debido al chikungunya. Las áreas y tipos de agentes activos representados en esta simulación son pacientes, personal de admisión, enfermeras de triaje, médicos, pruebas de laboratorio, ambulancia y salas de cuidados.

Las amplias variedades de cada combinación de valores representan una simulación de escenario diferente. Los valores de f constituyen el espacio de parámetros. Y su combinación pueden generar muchos escenarios diferentes (1).

Es importante destacar que además de número de personas que trabajan en el servicio, (nadmissions, ntriagenursing,...), a cada tipo se le puede asignar un tiempo medio de atención al paciente, en principio se han previsto tiempos diferentes para el personal con experiencia y el personal sin experiencia. El segundo paso para ajustar el simulador es especificar estos parámetros de entrada.

Los agentes se caracterizan por una máquina de Moore probabilística. La función f y f´ son la función de densidad de probabilidad. Estas funciones se sintonizarán mediante el análisis de datos históricos reales en el proceso de ajuste. Este trabajo se debe realizar en el proceso de configuración porque los diferentes departamentos de urgencias tienen características diferentes, y es una parte del simulador que permite calibrarlo o sintonizarlo a partir del modelo general. Por lo tanto las ecuaciones (2) y (4), cada paciente mostrará un comportamiento diferente durante la ejecución de el modelo debido a la distribución de probabilidad y sus propias diferencias. Pero la propiedad estadística de los agentes reflejarán su comportamiento común [18].

Donde LOS es la duración de la estancia del paciente y la age es la edad del paciente, que también tiene gran influencia en la probabilidad de transición de estado. Level es el nivel de gravedad del el paciente y TOT es el tipo de servicio de prueba o diagnóstico por médico. Las funciones f y f' son la densidad de probabilidad función.

Para generalizar el proceso a todos los pacientes, la distribución de probabilidad durante la simulación decidirá el siguiente estado. El modelo de distribución de probabilidad se basó en datos estadísticos del servicio de urgencias. La Figura 1 indica el proceso general durante la estadía del paciente; P1(%), P2(%), P3(%) y Pn (%) representan la probabilidad de la siguiente transición de estado por separado, las ecuaciones (2), (3), (4), (5), muestran las ecuaciones. Todas las probabilidades siguen distribuciones de probabilidad y la probabilidad de la función de densidad de la distribución se decide por varios parámetros clave basados en el análisis estadístico de la decisión del médico y el comportamiento del paciente; un proceso de ajuste estima el valor de estos parámetros a partir de datos históricos reales del departamento de emergencias especificado.

$$\begin{aligned} n_{scenarios} = & n_{admissions} * n_{timeadmissions} * \\ & n_{triagenursing} * n_{timetriagenursing} * \\ & n_{nursing} * n_{timenursing} * \\ & n_{doctor} * n_{timedoctor} \\ & n_{laboratory} * n_{timelaboratory} * \end{aligned} \tag{1}$$

$$Pi = f(LOS, age, level) \tag{2}$$

$$\sum_{i=1}^{n} P_i = 100\,\% \tag{3}$$

$$P' = f'(TOT, age, level) \tag{4}$$

$$\sum_{i=1}^{n} P'_i = 100\,\% \tag{5}$$

En resumen, el método de adaptación requiere 3 pasos:

- Definir el entorno, especificando el número de boxes, y salas de servicios (laboratorios, salas de rayos) que afectan directamente a la atención al paciente.
- Definir el personal que va estar implicado directamente en la atención a los pacientes (personal de admisión, médicos y enfermeras), en otras versiones del simulador se ha añadido personal, por ejemplo en el caso de de querer analizar enfermedades por contagio se deberá tener en cuenta el personal de limpieza y el tiempo que tarda en preparar los boxes, o los camilleros que pueden ser vectores de transmisión.
- Calibrar o sintonizar las probabilidades que afectan al servicio en función de los datos históricos, se elegirá el 80 % de los datos para calibrar y el 20 % restante para verificar.

V. Datos de entrada experimentales

Entre enero y febrero de 2023, el Hospital Nacional de Itauguá, al igual que otros hospitales de Paraguay debido al aumento de casos de chikungunya, se ha saturado la disponibilidad de camas a pesar de contar con un elevado número de plazas (200 camas). En el Hospital Nacional de Itauguá se debe contemplar que hay un número elevado de pacientes con otras patologías como diabetes, problemas de hipertensión y lesiones graves por accidentes de tránsito que llegan de diferentes partes del país. En la Tabla I muestra los valores de los parámetros de la configuración de los recursos humanos para representar el departamento de emergencia simulado.

Tabla I: Representación cuantitativa del departamento de emergencias del Hospital Nacional de Itauguá

Valor de la configuración de RRHH	
Interpretation	***Número***
Personal de junior de admisión	3
Personal senior de admisión	2
Enfermera de triaje junior	2
Enfermera de triaje senior	4
Enfermera junior area A	4
Enfermera senior área A	5
Enfermera junior area B	5
Enfermera senior área B	4
Laboratorio junior exterior	6
Laboratorio senior Interno	3
Doctor junior área A	6
Doctor Senior área A	3
Doctor Junior área B	5
Doctor Senior área B	4

[a]Valor de la configuración de RRHH

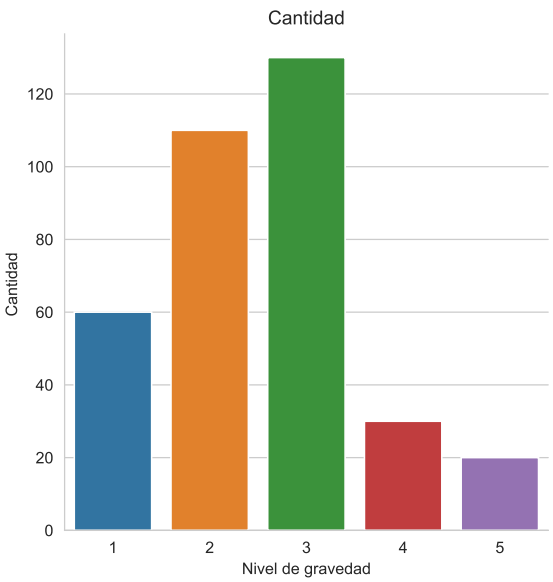


Fig. 2: Número de pacientes de acuerdo al nivel de gravedad

El modelo de llegada de pacientes incluye pacientes con distintos niveles de gravedad L (nivel de agudeza). Para hacer una verificación y validación cuantitativa, se construyó un modelo de llegada de pacientes de acuerdo con los datos reales del Hospital Nacional de Itagua. Con este criterio se ha preparado un conjunto de datos de entrada sintéticos para el simulación basados en la distribución de los pacientes que llegan al hospital. Esta distribución incluye pacientes con chikungunya con diferentes niveles de gravedad (4), la distribución general de pacientes simulados por edades y la distribución de los niveles de gravedad de los pacientes.

Una de las entradas del simulador es que la distribución de distintos niveles de agudeza L (agudeza) entre los pacientes que llegaron se obtuvo a través de análisis estadísticos de los datos reales y la distribución total de edad de los pacientes, como se muestra en la Figura 2 y la Figura 3. Como se puede observar en estas Figura 2, Figura 3, el nivel de severidad 2,3

acumula mayor número de pacientes, como también el sexo femenino acumula mayor número de pacientes en el rango de 15 a 100 años de edad comparado con el sexo masculino. En la Figura 4, se puede observar el número de internados por chikungunya entre las últimas semanas del 2022 y las primeras 16 semanas del 2023, teniendo su pico en la semana número 6 del 2023.

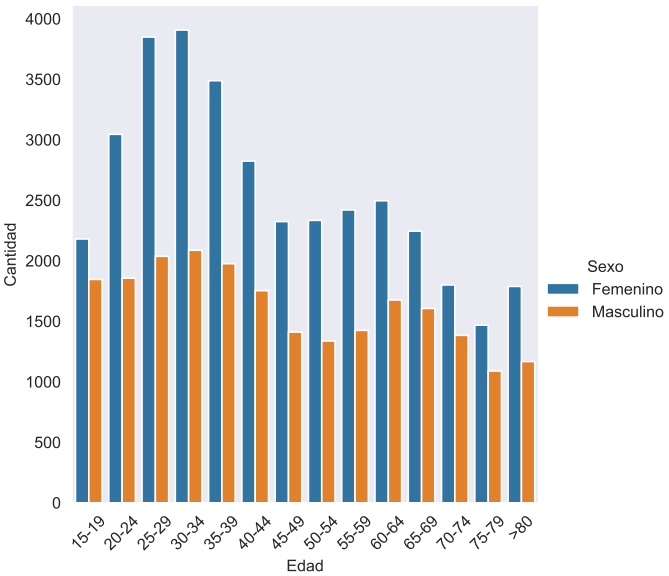


Fig. 3: Número de pacientes de acuerdo a la edad y el sexo

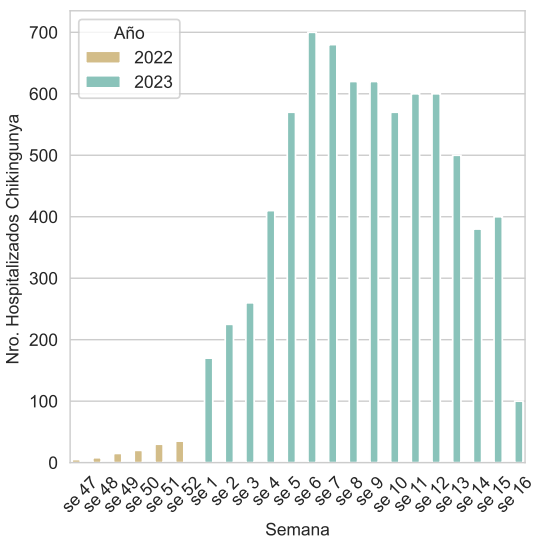


Fig. 4: Número de internados de chikungunya entre finales del 2022 y 2023

VI. Conclusiones

Para concluir, este artículo presenta un modelo del departamento de emergencias basado en agentes, que utiliza Netlogo como plataforma de simulación. Se ha propuesto un método para adaptar el simulador a otro entorno para dar soporte a la gestión del servicio de urgencias en caso de epidemias. Para aplicar el método propuesto se ha seleccionado un caso de estudio, se ha aplicado adaptando el simulador a un hospital de Paraguay y se ha preparado para dar soporte a la gestión del servicio en el caso de una epidemia ocasionada por chikungunya, preparando el simulador y generando unos datos sintéticos a los que se añaden a la llegada de pacientes genérica del hospital los pacientes infectados por chikungunya.

El modelo basado en agentes nos permite diseñar un plan de contingencia prévio a la llegada de una posible pandemia, permitiendo adelantarnos en las modificaciones del sistema sanitario tanto en estructuras como en personal para dar mejor servicio a la población. Para probar posibles usos del simulador, solucionar el creciente problema de saturación de llegadas de pacientes, y qué hacer para aumentar camas y recursos humanos como médicos y enfermeras en estos casos.

Para la simulación se ha preparado un conjunto de datos sintéticos de entrada, por lo que se ha tenido que generar la distribución de los pacientes que llegan con chikungunya con diferentes niveles de gravedad y distribución general de las edades de los pacientes simulados.

Las características de flexibilidad y adaptabilidad de este modelo genérico permiten que la simulación del departamento de urgencias se adapte a diferentes escenarios sin modificar significativamente el modelo subyacente al establecer los parámetros de entrada. Esta metodología facilita que los investigadores puedan centrar su esfuerzo en comprender el comportamiento del departamento de urgencias. Como trabajo futuro, se pretende ampliar el modelo y generar diferentes escenarios de la chikungunya en el servicio de urgencias.

Agradecimientos

El presente trabajo ha sido financiado mediante la Agencia Estatal de Investigacion (AEI), Spain and the Fondo Europeo de Desarrollo Regional (FEDER) UE, under contract PID2020-112496GB-I00 and partially funded by the Fundacion Escuelas Universitarias Gimbernat (EUG).

Referencias

[1] Javier Vargas Herrera, "Prevención y control de la malaria y otras enfermedades trasmitidas por vectores en el perú," *Rev. Peru. Epidemiol*, vol. 11, pp. 18, 2003.

[2] Thomas E Morrison, "Reemergence of chikungunya virus," *Journal of virology*, vol. 88, no. 20, pp. 11644–11647, 2014.

[3] "Casos de chikunguña suman cinco veces más que el histórico registrado, alerta Salud Pública - Nacionales - ABC Color," .

[4] Lyle R Petersen and Ann M Powers, "Chikungunya: epidemiology," *F1000Research*, vol. 5, 2016.

[5] Zhengchun Liu, Dolores Rexachs, Emilio Luque, Francisco Epelde, and Eduardo Cabrera, "Simulating the micro-level behavior of emergency department for macro-level features prediction," in *2015 Winter Simulation Conference (WSC)*. IEEE, 2015, pp. 171–182.

[6] Gerardo Ortigoza, Fred Brauer, and Iris Neri, "Modelling and simulating chikungunya spread with an unstructured triangular cellular automata," *Infectious Disease Modelling*, vol. 5, pp. 197–220, 2020.

[7] Steven Raigosa Osorio, Eliécer Aldana Bermúdez, Anibal Muñoz Loaiza, and Quindío-Colombia Armenia, "A simulation model for the chikungunya with vectorial capacity," *Appl. Math. Sci*, vol. 9, pp. 6953–6960, 2015.

[8] Somsakun Maneerat and Eric Daudé, "A spatial agent-based simulation model of the dengue vector aedes aegy-

pti to explore its population dynamics in urban areas," *Ecological Modelling*, vol. 333, pp. 66–78, 2016.

[9] Diego Ruiz-Moreno, Irma Sanchez Vargas, Ken E Olson, and Laura C Harrington, "Modeling dynamic introduction of chikungunya virus in the united states," *PLoS Neglected Tropical Diseases*, vol. 6, no. 11, pp. e1918, 2012.

[10] Carlos J Dommar, Rachel Lowe, Marguerite Robinson, and Xavier Rodó, "An agent-based model driven by tropical rainfall to understand the spatio-temporal heterogeneity of a chikungunya outbreak," *Acta tropica*, vol. 129, pp. 61–73, 2014.

[11] Rafael Mateus C, Susana Alvarez Zuluaga, Mariajose Franco Orozco, Paula Alejandra Escudero Marín, et al., "Modeling the propagation of the dengue, zika and chikungunya virus in the city of bello using agent-based modeling and simulation," Tech. Rep., Center for Open Science, 2021.

[12] Stephan Karl, Nilimesh Halder, Joel K Kelso, Scott A Ritchie, and George J Milne, "A spatial simulation model for dengue virus infection in urban areas," *BMC infectious diseases*, vol. 14, no. 1, pp. 1–17, 2014.

[13] Akshay Jindal and Shrisha Rao, "Agent-based modeling and simulation of mosquito-borne disease transmission," in *Proceedings of the 16th conference on autonomous agents and multiagent systems*, 2017, pp. 426–435.

[14] Manel Taboada, Eduardo Cabrera, Ma Luisa Iglesias, Francisco Epelde, and Emilio Luque, "An agent-based decision support system for hospitals emergency departments," *Procedia Computer Science*, vol. 4, pp. 1870–1879, 2011.

[15] Eduardo Cabrera, Emilio Luque, Manel Taboada, Francisco Epelde, and Ma Luisa Iglesias, "Abms optimization for emergency departments," in *Proceedings of the 2012 winter simulation conference (WSC)*. IEEE, 2012, pp. 1–12.

[16] Cecilia Jaramillo, Dolores Rexachs, Francisco Epelde, and Emilio Luque, "Virtual clinical trials: A tool for the study of transmission of nosocomial infections," *Procedia Computer Science*, vol. 108, pp. 109–118, 2017.

[17] Uri Wilensky and William Rand, *An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo*, Mit Press, 2015.

[18] Zhengchun Liu, Eduardo Cabrera, Dolores Rexachs, and Emilio Luque, "A generalized agent-based model to simulate emergency departments," in *Sixth Int Conf Adv Syst Simul*, 2014, pp. 65–70.

Arquitecturas del subsistema de memoria y almacenamiento secundario

Diseño de Memorias On-Chip para Aceleradores CNN Alimentados a Baja Tensión

Yamilka Toca-Díaz[1], Nicolás Landeros Muñoz[2], Rubén Gran-Tejero[1] y Alejandro Valero[1]

Resumen— **Reducir agresivamente la tensión de alimentación (V_{dd}) por debajo de la tensión mínima de operación (V_{min}) es una solución eficaz para conseguir un ahorro de energía sustancial. Desafortunadamente, operar a tensiones tan bajas supone un reto debido al elevado número de fallos permanentes como resultado de las variaciones en el proceso de fabricación de los nodos tecnológicos actuales.**

Este trabajo caracteriza el impacto de los fallos permanentes en la precisión de un acelerador para la inferencia de redes neuronales convolucionales (CNN) con memorias de activaciones *on-chip* alimentadas con una V_{dd} por debajo de V_{min}. Basándose en estas observaciones, este artículo propone un par de técnicas microarquitectónicas de bajo coste basadas en volteo y redirección de contenidos, las cuales son transparentes al programador y no dependen de las características específicas de cada red neuronal, con el objetivo de garantizar la precisión del acelerador a pesar de la presencia de fallos como consecuencia de trabajar con $V_{dd} < V_{min}$.

Los resultados experimentales muestran que las técnicas propuestas mantienen la precisión original durante la inferencia de la red CNN con un impacto mínimo en el rendimiento (menos de un 0,05 %) y reducen el consumo de energía en un 11,2 % y un 46,7 % con respecto a un acelerador convencional operando con tensiones de alimentación V_{min} y nominal, respectivamente.

Palabras clave— **Aprendizaje automático, eficiencia energética, fallos permanentes, precisión de red.**

I. Introducción

Las variaciones en el proceso de fabricación de los nodos tecnológicos CMOS actuales imponen unos márgenes de funcionamiento conservadores que comprometen la eficiencia energética de un sistema informático. Un ejemplo es la tensión de alimentación del transistor (V_{dd}). Esta tensión se fija de forma conservadora por encima de la tensión mínima de operación (V_{min}) impuesta por el transistor peor fabricado para garantizar un funcionamiento fiable de todos los transistores. Sin embargo, este sobreescalado de V_{dd} resulta en un desperdicio de energía, puesto que los transistores mejor fabricados podrían funcionar con una tensión de alimentación inferior, y la energía estática y dinámica escalan lineal y cuadráticamente con V_{dd}, respectivamente.

Como cualquier otro circuito digital, los aceleradores para la inferencia de Redes Neuronales Convolucionales (CNN) se ven afectados por variaciones en el proceso de fabricación. Estos aceleradores suelen incluir estructuras de almacenamiento *on-chip* relativamente grandes y de gran consumo energético para las activaciones y los pesos de la red neuronal. Estas estructuras de memoria se implementan normalmente con celdas típicas SRAM consistentes en 6 transistores (6T) por bit y resultan susceptibles a las variaciones en el proceso de fabricación mencionadas con anterioridad.

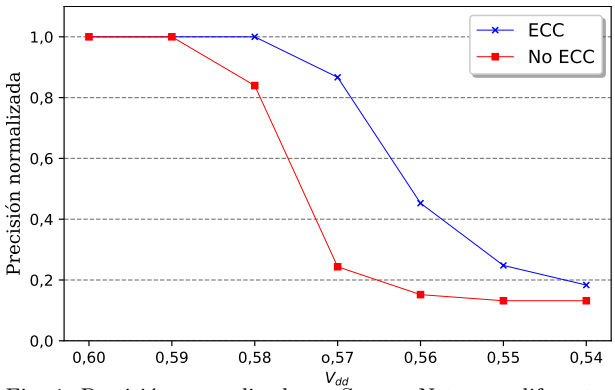


Fig. 1: Precisión normalizada en SqueezeNet para diferentes tensiones de alimentación.

Centrándose en el ahorro energético, una solución eficaz, conocida en inglés como *Dynamic Voltage Scaling* (DVS), consiste en aproximar V_{dd} hacia V_{min} con una frecuencia fija en distintos componentes del microprocesador, inlcuyendo a las grandes memorias *on-chip* [17, 18]. Subescalar V_{dd} de forma agresiva por debajo de V_{min} supone un reto debido al elevado número de fallos permanentes que aparecen en las celdas de memoria vulnerables, lo cual las obliga a mantener un valor lógico determinado [21]. En este sentido, la cobertura de los Códigos de Corrección de Errores (ECC) convencionales es bastante limitada y requiere grandes capacidades de almacenamiento, codificadores/decodificadores complejos y un elevado consumo de energía para garantizar un funcionamiento fiable del sistema. [13, 28].

La Figura 1 ilustra la afirmación anterior con la pérdida relativa de precisión de la red SqueezeNet cuando la V_{dd} de la memoria de activaciones *on-chip* de un acelerador CNN resulta inferior a V_{min} (0,6 V). Los resultados diferencian entre una memoria de activaciones desprotegida o protegida con la técnica ECC, particularmente corrigiendo un fallo y detectando dos fallos, con una granularidad de palabra de 16 bits. La precisión mejora significativamente con ECC; sin embargo, disminuye gravemente con

[1]Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, España, e-mail: {yamilka,rgran,alvabre}@unizar.es.
[2]Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, Italia, e-mail: nicolasignacio.landeros@polimi.it

tensiones de alimentación inferiores a 0,57 V.

En el contexto de los aceleradores CNN, algunos trabajos anteriores han propuesto diferentes mecanismos para contrarrestar el elevado número de fallos como consecuencia de $V_{dd} < V_{min}$, tales como el reentrenamiento redes neuronales bajo fallos [32], ajustar V_{dd} para cada capa de la red en tiempo de ejecución [31], o mejorar el algoritmo del proceso de compilación de una FPGA para evitar celdas defectuosas [21].

Las propuestas anteriores dependen del programador o requieren una caracterización fuera de línea de la aplicación CNN para adaptar el mecanismo. Por el contrario, el presente trabajo propone un par de mecanismos microarquitectónicos de bajo coste que resultan transparentes al programador y no dependen de las características específicas de la aplicación. Basándose en el impacto que los bits defectuosos causan en la precisión de las aplicaciones CNN, el enfoque propuesto transforma la representación de las activaciones con un número de fallos bajo y proporciona un almacenamiento alternativo y fiable para las activaciones con un número de fallos elevado.

Los resultados experimentales muestran que los mecanismos microarquitectónicos propuestos reducen el consumo medio de energía de las memorias de activaciones en un 11,2 % con respecto a las memorias de un acelerador convencional alimentado a V_{min}, sin pérdida de precisión y con un impacto mínimo en el rendimiento del sistema (menos de un 0,05 %).

El resto de este artículo se organiza como sigue. La Sección II introduce los antecedentes de este trabajo. La Sección III presenta un estudio de caracterización de fallos. La sección IV detalla las técnicas microarquitectónicas propuestas. La Sección V se refiere a la evaluación experimental. La Sección VI describe los trabajos relacionados, y por último, la Sección VII concluye este trabajo.

II. Antecedentes

Esta sección resume la arquitectura del acelerador CNN y el modelo de fiabilidad utilizado en este trabajo para evaluar las técnicas propuestas.

A. Arquitectura Básica del Acelerador CNN

La arquitectura básica del acelerador CNN utilizada en el presente trabajo se inspira en modelos de acelerador del estado-del-arte tanto de la academia como de la industria, como es el caso de DaDianNao [5] o la TPU de Google [10], para acelerar el proceso de inferencia de las aplicaciones CNN. La Figura 2 muestra la organización del hardware, que consta de una Matriz de 16×16 Elementos de Procesamiento (EP), memorias *on-chip* para reducir el coste de los accesos a la memoria externa, *dispatchers* para cada memoria y una unidad de control. El almacenamiento intermedio *on-chip* incluye un par de memorias de activaciones de 2 MiB (destacadas en azul en la figura) y una memoria de pesos de 2 MiB. La capacidad de cómputo y almacenamiento se dimensiona de acuerdo con el dominio de los sistemas embebidos [29].

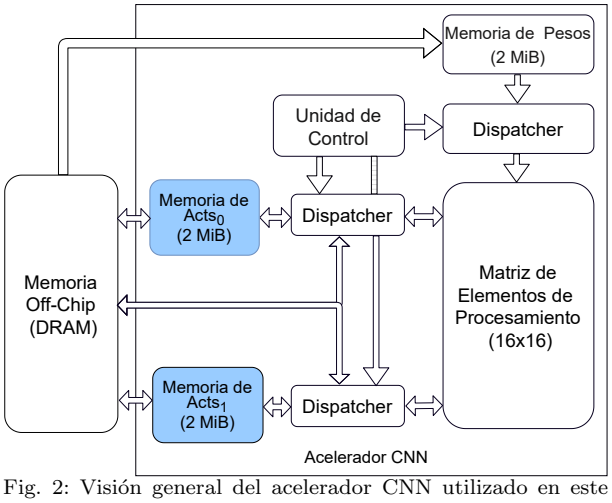


Fig. 2: Visión general del acelerador CNN utilizado en este trabajo.

La matriz EP constituye un procesador sistólico con elementos de procesamiento interconectados mediante una malla 2D. Cada PE computa de forma independiente productos escalares de 16 bits en coma fija mediante sumas parciales con un operando procedente de una memoria de activaciones, que actúa como memoria de entrada, y otro operando procedente de la memoria de pesos. El flujo de datos en la matriz EP corresponde al enfoque estacionario de salida descrito en SCALE-Sim [22].

Al igual que en el acelerador EIE [6], las memorias de activaciones intercambian sus funciones tras la obtención de cada capa de la red. De esta manera, una memoria de activaciones determinada almacena capas pares y la memoria homóloga almacena las capas impares. Por otro lado, la memoria de pesos almacena los pesos que se emitirán en el orden adecuado hacia la matriz EP por parte del *dispatcher* .

De forma similar a modelos de acelerador anteriores [11,16,23], las activaciones y pesos ocupan 16 bits y se representan en aritmética de coma fija, ajustando el número de bits enteros y fraccionarios a los requisitos de cada aplicación CNN en tiempo de ejecución (véase la Sección III).

El tamaño relativamente pequeño de las memorias de activaciones implica volcar a memoria *off-chip* aquellas activaciones de capas que superen los 2 MiB. Estas memorias están dispuestas como memorias *scratchpad* y diseñadas para proporcionar 16 activaciones (32 bytes por ciclo) al procesamiento paralelo en la matriz EP. Por último, los *dispatchers* se controlan por parte de la unidad de control, la cual explota información de control de la capa procesada.

B. Modelo de Fiabilidad

Este trabajo emplea un modelo de fiabilidad basado en una plataforma de hardware real consistente en una FPGA Xilinx VC707 de 28 nm [21]. El modelo de fiabilidad se obtene a partir de un test de la memoria *on-chip* trabajando con diferentes modos de operación de bajo consumo con V_{dd} por debajo

Tabla I: Características principales de las aplicaciones CNN estudiadas.

Aplicación	Profundidad	Tamaño medio de capa	Representación de activación
AlexNet [14]	5×Conv, 3×FC, 3×MaxPooling	153 KiB	4 bits enteros, 4 bits fraccionarios
DenseNet [9]	120×Conv, 1×FC, 5×Pooling	254 KiB	3 bits enteros, 5 bits fraccionarios
MobileNet [7]	28×Conv, 1×Pooling	340 KiB	4 bits enteros, 9 bits fraccionarios
SqueezeNet [8]	26×Conv, 4×Pooling	431 KiB	6 bits enteros, 4 bits fraccionarios
VGG16 [24]	13×Conv, 3×FC, 5×MaxPooling	1.32 MiB	3 bits enteros, 8 bits fraccionarios
ZFNet [30]	5×Conv, 3×FC, 3×MaxPooling	324 KiB	4 bits enteros, 6 bits fraccionarios

de V_{min} (0,6 V). Esta FPGA incluye una memoria *on-chip* con una capacidad de almacenamiento de 4 MiB que se corresponde con el almacenamiento de activaciones del acelerador CNN utilizado. En particular, este trabajo asume el modo de funcionamiento de bajo consumo en el que $V_{dd} = 0{,}54$ V. Ajustar V_{dd} por debajo de este límite no es posible puesto que la FPGA deja de funcionar. Véase la siguiente sección para una caracterización detallada de los patrones de fallos en las activaciones.

Trabajos anteriores han identificado que las activaciones son más vulnerables a los fallos que los pesos [15, 19]. Esto se debe principalmente a que las activaciones suelen abarcar un mayor rango de valores, lo que puede implicar mayores desviaciones absolutas en caso de fallo. Por tanto, el modelo de fiabilidad utilizado sólo se centra en las memorias de activaciones. En este sentido, el acelerador CNN dispone de dominios de tensión dedicados para lógica combinacional y secuencial, manteniendo el resto de componentes hardware con una V_{dd} elevada para evitar fallos, tal y como se describe en trabajos previos académicos [4] y dispositivos comerciales [2].

III. Estudio de Caracterización

Hemos escogido una serie de aplicaciones CNN de referencia con diferentes requisitos computacionales y de memoria. La Tabla I resume las principales características de estas CNN. La profundidad en número de capas varía significativamente entre la red neuronal más pequeña (AlexNet) y la más grande (DenseNet), así como el tamaño medio de las capas, desde cientos hasta miles de KiB. La columna de más a la derecha se refiere al número necesario de bits para la parte entera y fraccionaria en la representación de las activaciones a fin de evitar pérdidas de precisión con respecto a la precisión top-1 suponiendo una respresentación en coma flotante de 32 bits (IEEE-754).

La mayoría de aplicaciones requieren más de 8 bits para representar las activaciones. Por tanto, en este trabajo se asume que los bits correspondientes a la fracción se extienden para representar activaciones con 16 bits. Todas las CNN ejecutan un conjunto de datos de histología de cáncer colorrectal para la clasificación de imágenes [12]. Todos los resultados presentados en este trabajo se promedian para la inferencia de 750 imágenes. Véase la Sección V-A para más detalles acerca del entorno experimental.

El impacto de los fallos en la precisión de una aplicación CNN depende principalmente de la posición de los bits afectados. Es decir, para un tipo de datos de coma fija, la desviación en la magnitud de una activación defectuosa escala exponencialmente con la importancia del bit afectado. De esta manera, se definen tres tipos de activaciones según la ubicación de los bits defectuosos:

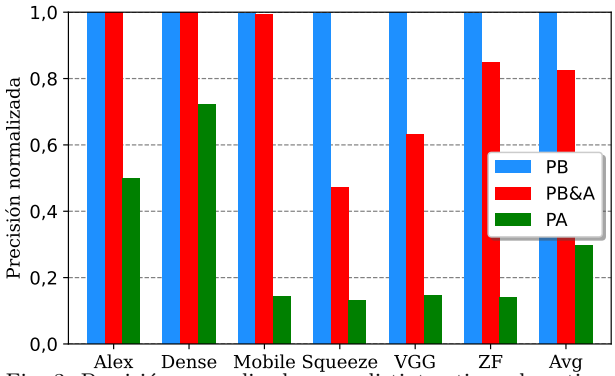


Fig. 3: Precisión normalizada para distintos tipos de activaciones defectuosas: Parte Baja (PB), Parte Alta (PA) y Parte Baja & Alta (PB&A).

- Parte Baja (PB). Estas activaciones sólo contienen fallos en el byte menos significativo. Las activaciones de parte baja representan el 0,45 % del total de activaciones.
- Parte Alta (PA). Estas activaciones sólo presentan fallos en el byte más significativo. El 0,45 % de las activaciones presentan este patrón de fallos.
- Parte Baja & Alta (PB&A). Las activaciones con este patrón muestran fallos tanto en los bytes menos significativos como en los más significativos. Estas activaciones representan el 0,0035 % de todas las activaciones.

Los datos muestran una distribución simétrica para las categorías PB y PA (0,45 %). En total, las tres categorías suman casi un 1 % de activaciones defectuosas en la memoria de activaciones. Tal y como se indica en la Sección II-B, el modelo de fiabilidad utilizado en el presente trabajo corresponde a una plataforma de hardware real. La Figura 3 muestra el impacto de los distintos tipos de activaciones defectuosas en la precisión de las aplicaciones CNN estudiadas.

Las activaciones PB no comprometen la precisión de ninguna CNN puesto que la desviación de la magnitud resultante es reducida. Según la Tabla I, los bits menos significativos sólo almacenan bits fraccionarios para todas las aplicaciones. Por el contrario, las activaciones PA afectan en gran medida a la precisión debido a que los fallos afectan a los bits de más peso de la activación. En el caso de MobileNet, SqueezeNet, VGG16 y ZFNet, la precisión se degrada hasta el valor más bajo posible (0,125), correspondiente a una clasifiación aleatoria (un octavo de la

precisión normalizada según las ocho categorías diferentes del conjunto de datos). En general, para las activaciones PA, la precisión normalizada varía desde un 12,5 % hasta un 72,2 % (DenseNet), con una media del 29,7 %.

La categoría PB&A también supone un impacto notable en la precisión a pesar de ser afectar solamente a un 0,0035 % de las activaciones. Por ejemplo, en SqueezeNet y VGG16, la precisión decae hasta un 47,1 % y 63,2 %, respectivamente. Estas son las aplicaciones que más memoria demandan, lo que significa que están más expuestas a fallos en la memoria de activaciones. Además, SqueezeNet require 6 bits para representar la parte entera de las activaciones, con lo que un fallo en esa parte incrementa la magnitud del error.

IV. Propuestas Microarquitectónicas

La observación en la que se fundamentan nuestras propuestas microarquitectónicas es la siguiente: fallos en la parte más significativa de la activación provocan un mayor error en la magnitud representada, lo cual se traduce en una pérdida de precisión mayor durante la inferencia. El objetivo de estas técnicas es minimizar el impacto o corregir los fallos en la parte alta de las activaciones. La primera de las técnicas hace frente a las activaciones de tipo PA, mientras que la segunda técnica realiza lo propio para las activaciones PB&A.

A. Técnica de Volteo

El objetivo de esta técnica es minimizar el peso de los bits defectuosos en las activaciones PA. Asumiendo una representación de datos *little-endian* y activaciones PA con N bits, los fallos están contenidos en las posiciones de bit desde $N/2$ hasta $N-1$. Tras aplicar una operación de volteo a la activación, el bit que ocupa la posición i-ésima pasa a ocupar la posición $(N-1-i)$-ésima. Por ejemplo, para $N=16$, el bit 12 se convierte en el bit 3 y viceversa. Con la técnica de volteo, las activaciones PA se transforman en activaciones PB, en las que los fallos tan sólo se localizan en bits desde 0 hasta $N/2-1$.

Para diferenciar entre activaciones volteadas y no volteadas, el diseño propuesto incluye un bit de control f por activación. Este bit se establece durante un test de memoria posterior a la fabricación para diferentes valores de tensión de alimentación, de manera similar a los mapas de bits defectuosos utilizados por parte de las técnicas de detección/corrección de errores para diferenciar entre contenidos defectuosos y fiables [25].

La Figura 4 muestra cómo se rediseña el puerto de lectura de una memoria de activaciones para realizar la operación de volteo en activaciones seleccionadas. El puerto incluye 16 multiplexores 2:1, de acuerdo con el tamaño de bloque de una operación de lectura en número de activaciones. Estos multiplexores son accionados por el bit de control f_i, revertiendo el orden de una activación volteada cuando sea necesario ($f_i = 1$). Obsérvese que el puerto de escritura (no representado en la figura) requiere el mismo número de multiplexores para almacenar las activaciones en el orden correcto.

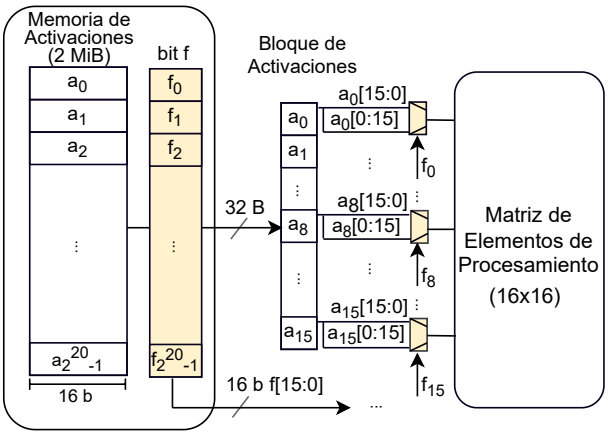


Fig. 4: Técnica de volteo propuesta consistente en 16 multiplexores 2:1 de 16 bits en el puerto de lectura de una memoria de activaciones. El bloque de lectura comprende las activaciones desde a_0 hasta a_{15}. Los componentes añadidos aparecen resaltados en color amarillo.

B. Técnica de Redirección

La técnica de volteo no elimina el impacto de las activaciones PB&A en la precisión de las redes. La segunda propuesta consiste en una pequeña cache, denominada cache de redirecciones, que almacena una copia fiable de dichas activaciones. En concreto, la cache de redirecciones sólo almacena el byte de mayor peso de esas activaciones, mientras que el byte de menor peso, a pesar de la presencia de fallos, se obtiene de la memoria de activaciones, puesto que los bits defectuosos del byte de menor peso no degradan la precisión de las CNN (véase la Sección III).

De forma similar a los bits f de la propuesta basada en volteo, el mecanismo de redirección requiere un bit de control p por activación para determinar si una activación solicitada se encuentra en la cache de redirecciones. Estos bits de control también se establecen durante un test de memoria posterior a la fabricación.

La cache de redirecciones almacena el byte más significativo de las activaciones PB&A que se reparten a lo largo de todo el espacio de direccionamiento de la memoria de activaciones. Para eliminar los fallos de conflicto, la cache de redirecciones se organiza como una cache asociativa por conjuntos de 5 vías, con una capacidad total de 1280 bytes. Esta capacidad es suficiente para almacenar todas las activaciones PB&A.

La Figura 5 muestra los componentes principales de la propuesta de diseño, distinguiendo los componentes relativos a volteo y redirección en amarillo y verde, respectivamente. La combinación de las dos técnicas requiere sustituir los multiplexores 2:1 por 4:1, accionados por los bits p_i y f_i con tres configuraciones distintas: ni volteo ni redirección (00), volteo (01) o redirección (10).

La cache de redirecciones es capaz de proporcionar el byte de mayor peso de una activación de tipo

Tabla II: Potencia estática (leakage), energía dinámica y área de la memoria de activaciones alimentada a 0,6 V (DVS) y 0,54 V (baseline). La tabla también muestra el sobrecoste de la propuesta de volteo y redirección (V+R) aplicada sobre la memoria de activaciones alimentada a 0,54 V, así como de la cache de redirecciones.

	Memoria de activaciones			Cache de redirecciones
	DVS	Base	V+R	
Potencia estática (mW)	315	269,8	288,5	4,1
Energía dinámica por lectura (pJ)	83,8	64,5	71,3	1,5
Energía dinámica por escritura (pJ)	66,4	48	53,7	1,7
Área (mm^2)	6,207		7,129	0,016

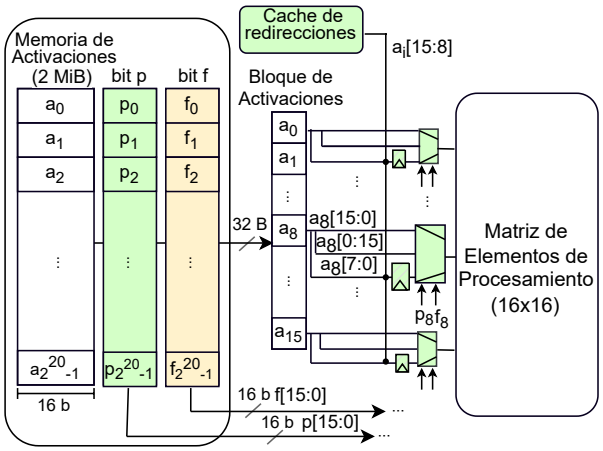


Fig. 5: Técnicas de volteo y redirección propuestas para el puerto de lectura de la memoria de activaciones. El bloque leído incluye las activaciones desde a_0 a a_{15}. Los componentes destacados verde se refieren a la propuesta de redirección.

PB&A por cada ciclo de procesador. Por ende, si en un bloque de activaciones (desde a_0 hasta a_{15}) se encuentran varias activaciones PB&A, se necesita el mismo número de ciclos para proveer los valores correctos desde la cache de redirecciones. Nótese que esto implica una penalización de 16 ciclos en el peor caso. No obstante, la cache de redirecciones se utiliza de manera puntual dado el pequeño número de activaciones PB&A. Prueba de ello es que, para cualquiera de las aplicaciones CNN, los resultados experimentales muestran menos de un 0,05 % de degradación en el rendimiento del sistema.

El byte más significativo procedente de la cache de redirecciones se concatena con el byte menos significativo procedente de la memoria de activaciones y se almacena temporalmente antes de suministrarlo a la matriz de elementos de procesamiento.

Finalmente, en cuanto al puerto de escritura de la memoria de activaciones, se mantiene el diseño con multiplexores 2:1 para distinguir entre valores con y sin volteo durante la escritura. En el caso de activaciones PB&A, el byte más significativo se escribirá en la entrada correspondiente de la cache de redirecciones.

C. Sobrecoste de Potencia Disipada, Energía Consumida y Área

El principal sobrecoste de nuestra propuesta corresponde a los bits de control f y p. Para una memoria de activaciones de 2 MiB, estos bits de control suponen 256 KiB adicionales de almacenamiento. Las memorias convecionales incluyen bits de control de estas características, por ejemplo, para distinguir entre contenidos fiables y defectuosos [26]. A pesar de ello, y de manera conservadora, este trabajo contabiliza el sobrecoste de los dos bits de control.

La Tabla II muestra resultados de potencia estática o *leakage*, energía dinámica y área. *Dynamic Voltage Scaling* (DVS) y *baseline* (Base) se refieren a memorias de activaciones alimentadas de forma segura a 0,6 V e insegura a 0,54 V, respectivamente, mientras que la etiqueta $V{+}R$ alude a las técnicas propuestas de volteo y redirección aplicadas a una memoria de activaciones alimentada a 0,54 V. La sobrecarga de V+R consiste en los multiplexores, *latches* y bits de control necesarios. Por último, la columna de la derecha se refiere a la sobrecarga de la cache de redirecciones. Los bits de control y la cache de redirecciones se alimentan a $V_{min} = 0,6$ V para evitar fallos. Todos los resultados se han obtenido con CACTI-P [3] para un nodo tecnológico de 32 nm y un tipo de dispositivo ITRS de bajo consumo. Como se ha observado, el consumo de potencia y energía de V+R se mantiene entre los resultados de los modos de funcionamiento de 0,6 V y 0,54 V, mientras que la sobrecarga de área supone alrededor de un 15 % de la memoria de activaciones.

Por último, nótese que los multiplexores 4:1 se encuentran en el camino crítico de la operación de lectura. Sin embargo, este trabajo asume que su latencia es insignificante.

V. Evaluación Experimental

Esta sección describe el entorno de simulación utilizado para obtener resultados experimentales. A continuación, se presentan los resultados, incluyendo el impacto en la precisión de las redes CNN y el consumo de energía.

A. Entorno de Simulación

Hemos extendido el entorno de simulación TensorFlow 2.5.0 [1] para modelar el flujo de datos del acelerador CNN descrito en la Sección II-A, incluyendo las técnicas propuestas de volteo y redirección. El modelo de flujo de datos establece una simulación precisa en ciclos, en la cual las latencias de acceso de la cache de redirecciones y las memorias de activaciones son uno y tres ciclos, respectivamente, para una frecuencia de procesador de 1 GHz [3]. La latencia de una suma parcial y acumulación en un EP se asume en un ciclo. En la Sección III se describen las aplicaciones CNN y el conjunto de datos de entrada.

B. Impacto en la Precisión

La Figura 6 muestra la precisión normalizada de los distintos mecanismos de tolerancia a fallos en me-

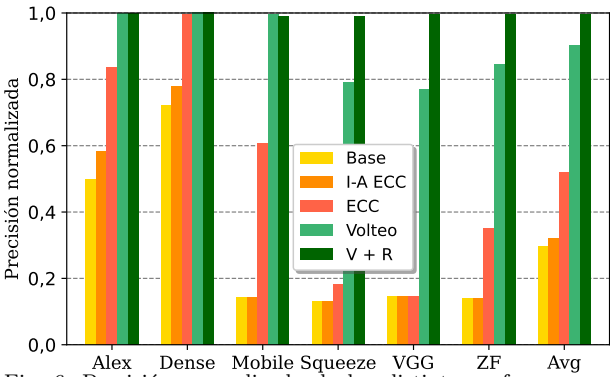


Fig. 6: Precisión normalizada de los distintos enfoques con respecto a un modo de funcionamiento convencional sin fallos ($V_{dd} \geq V_{min}$).

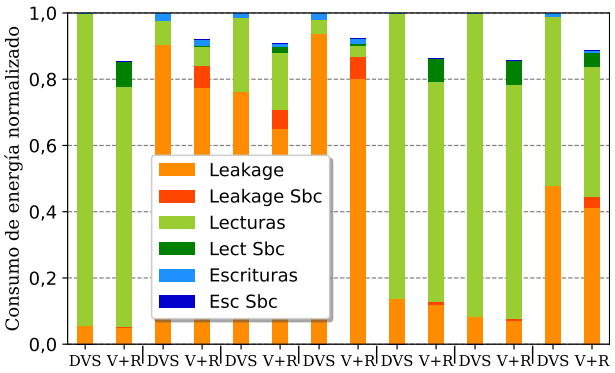


Fig. 7: Consumo de energía normalizado de una memoria de activaciones alimentada a 0,54 V con las técnicas de volteo y redirección propuestas respecto a una memoria de activación convencional alimentada a $V_{min} = 0{,}6$ V.

morias de activaciones alimentadas a 0,54 V con respecto a un modo de funcionamiento sin fallos con V_{dd} por encima de V_{min}. *Base* se refiere a un acelerador con memorias sin ninguna protección contra fallos. *I-A ECC* implementa una protección iso-area ECC con respecto a nuestra propuesta, es decir, con un coste en número de bits de control equivalente, siendo capaz de corregir un fallo y detectar dos fallos por cada bloque de activaciones de 8 bytes. La etiqueta *ECC* se refiere a una mayor protección capaz de corregir un fallo por cada palabra de 2 bytes, siendo necesarios 5 bits ECC para una activación de 16 bits. *V* se refiere únicamente a la técnica de volteo, mientras que *V+R* se aplica a las técnicas de volteo y redirección conjuntamente.

La precisión se ve gravemente afectada en el modelo base, cayendo a un 30 % de media. Los resultados son coherentes con los valores alcanzados por las activaciones PA comentados en la Sección III. Debido al elevado número de fallos con el subescalado de la tensión, los enfoques basados en ECC experimentan una mejora marginal de la precisión con respecto al modelo base, particularmente el modelo I-A ECC, alcanzando un 34 % de media. Por otro lado, convertir las activaciones PA en activaciones de PB mediante la técnica de volteo mejora la precisión al menos hasta un 77 % para cada *benchmark*. Sin embargo, la presencia de activaciones PB&A impide que la técnica de volteo alcance la precisión original (sin fallos). La combinación de las técnicas de volteo y redirección supera esta limitación, obteniendo casi la precisión original en todas las aplicaciones.

C. Consumo de Energía

El objetivo final de este trabajo es reducir el consumo de energía de las memorias de activaciones reduciendo tensión de alimentación agresivamente por debajo de la tensión mínima y a su vez preservando la precisión de la aplicación. Esta sección cuantifica el ahorro energético de los mecanismos de volteo y redirección teniendo en cuenta la sobrecarga de potencia y energía descrita en la Sección IV-C.

La Figura 7 muestra el consumo de energía normalizado de una memoria de activaciones que implementa nuestra propuesta V+R alimentada a una tensión de 0,54 V frente a una memoria de activaciones convencional alimentada a una tensión segura de 0,6 V mediante DVS. Los resultados diferencian entre energía estática o *leakage* y energía dinámica. A su vez, la energía dinámica se divide en operaciones de lectura y escritura en la memoria de activaciones. La etiqueta *Sbc* se refiere a la sobrecarga de las técnicas propuestas para cada tipo de energía.

La contribución de *leakage* sobre el consumo de energía total varía entre las CNN y depende principalmente del patrón de acceso a memoria. Es decir, aplicaciones como DenseNet con una reutilización elevada de las activaciones dentro de la matriz EP realizan menos accesos a la memoria de activaciones. En consecuencia, este tipo de aplicaciones mitigan la contribución del gasto de energía dinámica. Para todas las CNN estudiadas, el sobrecoste de *leakage* de V+R se compensa con el ahorro de *leakage* proporcionado por la reducción de V_{dd}.

La reducción de V_{dd} tiene un mayor impacto en el ahorro de energía dinámica, ya que este tipo de consumo crece cuadráticamente con la tensión de alimentación. En este sentido, las aplicaciones con una reutilización pobre de las activaciones (mayor peso del consumo dinámico), como VGG16, obtienen una mayor reducción del consumo total de energía. Al igual que el *leakage*, el ahorro dinámico es mayor que la sobrecarga dinámica de la propuesta gracias a la reducción de V_{dd}. Obsérvese que las lecturas son más frecuentes que las operaciones de escritura en todas las CNN analizadas, puesto que las activaciones de una capa determinada suelen leerse varias veces en función del número de filtros aplicados a los datos de entrada, mientras que sólo se escriben una vez.

En resumen, el ahorro total de energía de V+R oscila entre un 7,7 % (SqueezeNet) y un 14,5 % (AlexNet), con una media de 11,2 %. Este ahorro de energía puede parecer relativamente bajo; sin embargo, recuérdese que V_{dd} sólo se reduce de 0,6 V a 0,54 V, no siendo posible reducir aún más la tensión de alimentación debido a que la plataforma de hardware real deja de funcionar (véase la Sección II-B). En comparación con un acelerador convencional alimentado con tensión V_{dd} nominal (0,9 V), el ahorro medio de energía alcanza un 46,7 %.

VI. Trabajos Relacionados

La presente sección clasifica los trabajos anteriores en propuestas centradas en aceleradores CNN alimentados con V_{dd} por debajo de V_{min} y técnicas de redirección para microprocesadores de propósito general.

A. Aceleradores CNN

Las técnicas del estado-del-arte para aceleradores CNN se centran en determinar qué parámetros de las redes neuronales tienen un impacto mayor sobre el clasificador de la red neuronal. Una vez identificados, se intenta protegerlos para evitar grandes desviaciones en la precisión al reducir la tensión de alimentación durante el proceso de inferencia.

Zhang *et al.* caracterizan qué capas de las CNN son más susceptibles a los fallos, y para cada capa definen una tensión de alimentación diferente para minimizar el impacto en la precisión [31].

Salami *et al.* también identifican las capas más vulnerables de la CNN y modifican el algoritmo de colocación en el proceso de compilación de una FPGA para asegurarse de que esas capas no se almacenen en bloques de memoria defectuosos [21]. De forma similar, el presente trabajo identifica las celdas de memoria defectuosas mediante un test de memoria posterior a la fabricación del dispositivo. Sin embargo, a diferencia de Salami *et al.*, nuestra propuesta es independiente de la arquitectura de la red neuronal y no altera el algoritmo de colocación de la FPGA.

Zhang *et al.* exponen el proceso de entrenamiento de la red neuronal a los fallos permanentes con el fin de recalcular un conjunto de pesos que oculte el impacto de los fallos en la precisión de la red durante el proceso de inferencia [32]. Sin embargo, al igual que la solución anterior, esta propuesta se debe adaptar a la arquitectura específica de la red neuronal y requiere la intervención del programador.

B. Técnicas de Redirección para Procesadores de Propósito General

En el contexto de los procesadores superescalares CPU, las entradas ociosas de estructuras de memoria, como las cache de trazas, se han explotado como entradas de redirección que almacenan réplicas fiables de contenidos de cache L1 defectuosos [20]. Esta solución complica el diseño y la verificación del procesador, puesto que la gestión de la coherencia de memoria se propaga a dichas estructuras de redirección.

El concepto de redirección también se ha utilizado en los bancos de registros de la GPU. GR-Guard explota registros muertos que contienen datos obsoletos para almacenar datos útiles provenientes de los registros defectuosos [26]. Los registros muertos se identifican en tiempo de ejecución con la ayuda del compilador y modificaciones en el repertorio de instrucciones. Como alternativa, Valero *et al.* aprovechan la observación de que los registros son comprimibles en tiempo de ejecución, y asignan datos comprimidos a registros defectuosos, evitando el uso de celdas defectuosas [27].

VII. Conclusiones

Este trabajo ha explorado la posibilidad de reducir drásticamente la tensión de alimentación de las memorias de activaciones en los aceleradores para la inferencia de redes CNN con el objetivo de ahorrar energía. Para lidiar con el impacto en la precisión de la inferencia en CNN's como consecuencia de la aparición de fallos permanentes al reducir la tensión de alimentación, este trabajo ha propuesto un par de mecanismos microarquitectónicos de bajo coste basados en técnicas de volteo y redirección de contenidos. Al contrario que las propuestas del estado-del-arte, estas técnicas resultan transparentes al programador y no dependen de las características específicas o de la arquitectura de cada red neuronal.

Los resultados experimentales han mostrado que, comparado contra un acelerador convencional, la propuesta de acelerador mejorado con las técnicas de volteo y redirección reduce el consumo medio de energía de las memorias de activaciones en un 11,2 %, mientras se mantiene la precisión original de las aplicaciones con un impacto despreciable en el rendimiento del sistema (menos de un 0,05 % para cada aplicación).

Agradecimientos

Este trabajo ha sido financiado por MCIN/AEI/10.13039/501100011033 (proyecto PID2019-105660RB-C21) y por el Departamento de Ciencia, Universidad y Sociedad del Conocimiento del Gobierno de Aragón (ayuda a grupo de investigación de referencia T5820R).

Referencias

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *CoRR*, abs/1603.04467, 2016.

[2] ARM Limited. ARM11 MPCore™ Processor. Revision: r2p0. Technical Reference Manual, 2008.

[3] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas. CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *ACM Transactions on Architecture and Code Optimization*, 14(2), 2017.

[4] A. Chatzidimitriou, G. Panadimitriou, D. Gizopoulos, S. Ganapathy, and J. Kalamatianos. Assessing the Effects of Low Voltage in Branch Prediction Units. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pages 127–136, 2019.

[5] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. DaDianNao: A Machine-Learning Supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622, 2014.

[6] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 243–254, 2016.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, abs/1704.04861, 2017.

[8] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5MB Model Size. *CoRR*, abs/1602.07360, 2016.

[9] F. N. Iandola, M. W. Moskewicz, S. Karayev, R. B. Girshick, T. Darrell, and K. Keutzer. DenseNet: Implementing Efficient ConvNet Descriptor Pyramids. *CoRR*, abs/1404.1869, 2014.

[10] N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

[11] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos. Stripes: Bit-Serial Deep Neural Network Computing. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1–12, 2016.

[12] J. N. Kather, C.-A. Weis, F. Bianconi, S. M. Melchers, L. R. Schad, T. Gaiser, A. Marx, and F. G. Zöllner. Multi-Class Texture Analysis in Colorectal Cancer Histology. *Nature Scientific Reports*, 6, 2016.

[13] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe. Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 197–209, 2007.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

[15] N. Landeros Muñoz, A. Valero, R. G. Tejero, and D. Zoni. Gated-CNN: Combating NBTI and HCI Aging Effects in On-Chip Activation Memories of Convolutional Neural Network Accelerators. *Elsevier Journal of Systems Architecture*, 128:1–13, 2022.

[16] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo. UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision. *IEEE Journal of Solid-State Circuits*, 54:173–185, 2019.

[17] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose, and V. J. Reddi. Safe Limits on Voltage Reduction Efficiency in GPUs: A Direct Measurement Approach. In *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 294–307, 2015.

[18] J. Leng, Y. Zu, and V. J. Reddi. GPU Voltage Noise: Characterization and Hierarchical Smoothing of Spatial and Temporal Voltage Noise Interference in GPU Architectures. In *Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture*, pages 161–173, 2015.

[19] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler. Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017.

[20] D. J. Palframan, N. Kim, and M. H. Lipasti. iPatch: Intelligent Fault Patching to Improve Energy Efficiency. In *Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture*, pages 428–438, 2015.

[21] B. Salami, O. S. Unsal, and A. Cristal Kestelman. Comprehensive Evaluation of Supply Voltage Underscaling in FPGA on-Chip Memories. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, pages 724–736, 2018.

[22] A. Samajdar, Y. Zhu, P. N. Whatmough, M. Mattina, and T. Krishna. SCALE-Sim: Systolic CNN Accelerator. *CoRR*, abs/1811.02883, 2018.

[23] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. In *Proceedings of the ACM/IEEE 45th Annual International Symposium on Computer Architecture*, pages 764–775, 2018.

[24] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1556, 2014.

[25] J. Tan and X. Fu. Mitigating the Susceptibility of GPGPUs Register File to Process Variations. In *Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium*, pages 969–978, 2015.

[26] J. Tan, S. L. Song, K. Yan, X. Fu, A. Marquez, and D. Kerbyson. Combating the Reliability Challenge of GPU Register File at Low Supply Voltage. In *Proceedings of the 25th International Conference on Parallel Architectures and Compilation Techniques*, pages 3–15, 2016.

[27] A. Valero, D. Suárez-Gracia, and R. Gran-Tejero. DC-Patch: A Microarchitectural Fault Patching Technique for GPU Register Files. *IEEE Access*, 8:173276–173288, 2020.

[28] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S. Lu. Trading off Cache Capacity for Reliability to Enable Low Voltage Operation. In *Proceedings of the ACM/IEEE 35th Annual International Symposium on Computer Architecture*, pages 203–214, 2008.

[29] A. Yazdanbakhsh, K. Seshadri, B. Akin, J. Laudon, and R. Narayanaswami. An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks. *CoRR*, abs/2102.10423, 2021.

[30] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. *Springer Lecture Notes in Computer Science*, 8689, 2014.

[31] J. Zhang, K. Rangineni, Z. Ghodsi, and S. Garg. ThUnderVolt: Enabling Aggressive Voltage Underscaling and Timing Error Resilience for Energy Efficient Deep Learning Accelerators. In *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference*, pages 1–6, 2018.

[32] J. J. Zhang, T. Gu, K. Basu, and S. Garg. Analyzing and Mitigating the Impact of Permanent Faults on a Systolic Array Based Neural Network Accelerator. In *Proccedings of the IEEE 36th VLSI Test Symposium*, pages 1–6, 2018.

Plataforma de simulación para evaluar la interferencia de caché en sistemas de criticidad mixta en tiempo real

Tamara Lugo García[1], Javier Fernández Muñoz[2] y Jesús Carretero Pérez[2]

Resumen— **La ejecución de aplicaciones en tiempo real en un sistema multinúcleo provoca imprevisibilidad en la estimación del tiempo de ejecución de las tareas en el peor de los casos (WCET), debido principalmente a las interferencias en el uso simultáneo de recursos compartidos, entre ellos, la caché compartida. Las herramientas de simulación de caché son esenciales para comprender y evaluar las interferencias y estudiar nuevos algoritmos para mitigar sus efectos en el tiempo de ejecución de las tareas. En este artículo mostramos los detalles de diseño e implementación de un entorno de simulación de caché basado en trazas para procesadores multinúcleo. Nuestro entorno incluye una solución de generación de trazas que puede adaptarse fácilmente a cualquier plataforma hardware. Las trazas se recopilan utilizando direcciones de memoria virtual lo que permite reproducirlas en entornos independientes del hardware. Además, nuestro simulador incluye un planificador de tareas en tiempo real y permite definir características del sistema de memoria caché tales como el total de niveles de caché y las políticas empleadas en cada nivel. El objetivo final del simulador es proporcionar un entorno para evaluar las interferencias de caché en cargas de trabajo de criticidad mixta.**

Palabras clave— **Sistemas en tiempo real, arquitectura multinúcleo, simulación de caché, interferencias de caché, criticidad mixta.**

I. Introducción

En un sistema multinúcleo las interferencias en los recursos compartidos provocan imprevisibilidad en la estimación del peor tiempo de ejecución posible (WCET) de las tareas. Los sistemas de tiempo real requieren una estimación precisa y fiable del WCET lo que implica la implementación de técnicas para conseguir un comportamiento más predecible de los recursos compartidos. Dasari et al.[1] y Löfwenmark et al.[2] identificaron como mayor fuente de imprevisibilidad en sistemas multinúcleo: las cachés, las redes de interconexión y la memoria principal.

El uso de una memoria caché compartida puede reducir el tiempo medio de ejecución de una tarea, sin embargo, en el peor de los casos, la memoria caché compartida provoca importantes penalizaciones de tiempo debido a las interferencias. Las principales técnicas para reducir las interferencias en el uso de la memoria caché compartida incluyen técnicas de particionamiento [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13] y bloqueo de la caché [14], [15], [16], [17], [18].

Las herramientas de simulación de arquitecturas informáticas son esenciales para implementar y evaluar nuevos algoritmos de reducción de interferencias y pueden ser útiles para comprender el comportamiento de los programas y encontrar cuellos de botella. Un simulador de caché puede ser un simulador simplemente funcional (un simulador que simula lógicamente las operaciones que realiza la arquitectura para ejecutar un programa sin tener en cuenta el tiempo) o uno que simule también el tiempo de ejecución (que tiene en cuenta el tiempo que se tarda en ejecutar cada operación). En 2020, Brais et al.[19] presentaron un análisis detallado de los simuladores de caché implementados, evidenciando una tendencia importante al uso de simuladores basados en trazas [20].

Los simuladores basados en trazas utilizan como entrada un archivo de trazas generado a partir de la ejecución o emulación de un programa evitando la sobrecarga de ejecutar un programa de entrada. Son flexibles, ya que pueden reflejar los pasos reales de ejecución del programa con un uso realista de la memoria. Sin embargo, siguen presentando varios problemas. En primer lugar, las trazas de direcciones suelen ser grandes y las simulaciones suelen ser lentas. En segundo lugar, como el espacio de variables a considerar es grande (núcleos de procesador, políticas de caché, cachés privadas o compartidas entre todos los núcleos, etc.), la mayoría de los simuladores basados en trazas se ajustan a parámetros predefinidos y sencillos, lo que restringe el alcance de la simulación. En tercer lugar, la propia generación de trazas es engorrosa y suele adaptarse a una plataforma hardware, sobre todo debido al uso de direcciones de memoria física.

En este artículo, mostramos los detalles de diseño e implementación de un entorno de simulación de caché basado en trazas para procesadores multinúcleo que aborda algunos de los retos anteriores. Nuestro entorno incluye una solución de generación de trazas que puede adaptarse fácilmente a cualquier plataforma. Las trazas se recopilan utilizando direcciones de memoria virtual lo que permite reproducirlas en entornos independientes del hardware. Además, nuestro simulador permite definir hasta tres niveles de cachés y diferentes políticas para cada nivel de caché. El objetivo final del simulador es proporcionar un entorno para evaluar las interferencias de caché y reducir el tiempo de ejecución en cargas de trabajo de criticidad mixta.

En este documento se detalla el trabajo realizado a

[1]Dpto. de Informática,Universidad Carlos III de Madrid, e-mail: tlugo@ing.uc3m.es.

[2]Dpto. de Informática,Universidad Carlos III de Madrid, e-mail: {javier.fernandez,jesus.carretero}@uc3m.es

lo largo del desarrollo de esta artículo, y se estructura como sigue: En la sección II se presentan algunos antecedentes para una mejor comprensión de las ideas de este trabajo, junto con trabajos relacionados. En la sección III se muestran los principales conceptos del simulador propuesto, generación de trazas de ejecución, configuración, y detalles de funcionamiento. En la sección IV se discuten los experimentos realizados para comprobar la viabilidad del entorno de trabajo propuesto en el estudio del comportamiento de las interferencias de caché. Finalmente, la sección V describe las principales conclusiones de este trabajo.

II. Background

Como se ha mencionado anteriormente, el uso de la caché compartida puede reducir el tiempo medio de ejecución pero puede provocar importantes penalizaciones de tiempo debido a las interferencias. Los investigadores han estado proponiendo técnicas novedosas para reducir esta interferencia, por ejemplo, mediante mecanismos de particionamiento y bloqueo de la caché. El simulador de caché basado en trazas implementado, a diferencia de otros simuladores, además de evaluar el funcionamiento normal de la caché, permite probar diferentes configuraciones de técnicas que incluyen bloqueo y particionamiento de caché sin tener que ejecutar el programa real cada vez.

A. Simuladores de caché basados en trazas

La ventaja fundamental de los simuladores basados en trazas es el uso de trazas de ejecución como entrada en lugar de la ejecución real del programa. De este modo se evita la sobrecarga y la complejidad de tratar con dichas ejecuciones pero se siguen reflejando los pasos de ejecución reales del programa utilizando tantas configuraciones como sea necesario. Sin embargo, las trazas de ejecución suelen ser muy grandes y difíciles de almacenar y gestionar. Además, las trazas deben generarse a partir de ejecuciones reales, lo que añade otro nivel de complejidad.

Una simulación de cache basada en trazas consta generalmente de tres estados. Primero, la determinación de la secuencia de accesos a memoria realizados por la aplicación, en segundo lugar la eliminación de datos innecesarios o redundantes de las trazas, y en tercer lugar el procesamiento de la traza en el programa que simula el comportamiento de un sistema de memoria caché real[21].

Entre los simuladores de caché basados en trazas implementados en la bibliografía podemos encontrar DineroIV[22], CASPER[23], Moola[24], drcachesim[25], vCSIMx86[26] y SMPCache[27]. El principal inconveniente de estos simuladores es que sólo permiten una configuración limitada de la arquitectura simulada. La mayoría de ellos tienen algunos parámetros que no pueden ser modificados (como el número de núcleos del procesador, las políticas de caché, cachés privadas o compartidas entre todos los núcleos, etc.), que restringen el alcance de la simulación.

B. Técnicas implementadas para reducir la interferencia de caché

La interferencia de caché se produce cuando un bloque de datos almacenado en la caché es desalojado mientras todavía está siendo utilizado por el programa. Las interferencias de caché se pueden clasificar en Intratask, Intracore e Intercore. La interferencia Intratask se produce cuando una tarea expulsa sus propias líneas de caché. La interferencia Intracore se produce cuando dos o más tareas que se ejecutan en el mismo núcleo desalojan mutuamente las líneas que necesitan. Por último, la interferencia Intercore se produce en la caché compartida cuando una tarea que se ejecuta en un núcleo desaloja líneas que son necesarias para una tarea que se ejecuta en otro núcleo [10], [28], [2].

La mayoría de las técnicas fundamentales orientadas a reducir las interferencias y mejorar el WCET utilizan una o varias de las siguientes características: particionamiento hardware de los bancos (ways) de caché, bloqueo hardware de líneas o bancos de caché y particionamiento software mediante el coloreado de páginas. El particionamiento de la caché se centra en dividir el espacio de caché y asignar una partición de memoria a una tarea o núcleo determinado. Por otra parte, el bloqueo de la caché evita que sean reemplazadas las líneas o bancos de caché marcadas como bloqueadas.

En sistemas de tiempo real en plataformas multicore una tendencia cada vez más significativa es el uso de cargas de trabajo de criticidad mixta. Estas cargas de trabajo mezclan aplicaciones críticas con distintos grados de criticidad y aplicaciones no críticas. El problema fundamental del uso de sistemas de criticidad mixta en plataformas multinúcleo es garantizar que se cumplen los requisitos de tiempo real y seguridad de las tareas críticas cuando se combinan tareas críticas con tareas no críticas en la misma plataforma hardware, garantizando al mismo tiempo el mejor uso posible de los recursos hardware[29].

En los últimos años, se ha dedicado un gran esfuerzo de investigación a evitar, o al menos mitigar las interferencias para que el tiempo de ejecución pueda ser más determinista. Chisholm et al.[30] intentan lograr un equilibrio entre las restricciones de tiempo real y el uso eficiente de la caché compartida para cargas de trabajo de criticidad mixta en plataformas multinúcleo. En este artículo, proponen un doble particionamiento de la caché a nivel de bancos y a nivel de conjuntos (Particionamiento de bancos y conjuntos). Cada núcleo dispone de un registro de bloqueo que permite la partición por bancos y realizan la partición por conjuntos mediante el coloreado de páginas. Las tareas no críticas de todos los núcleos comparten los mismos bancos y conjuntos en la caché compartida de último nivel. Las tareas críticas que se ejecutan en el mismo núcleo tienen su propio espacio privado en la caché. Los colores se reparten a partes iguales entre las tareas críticas que se ejecutan en

distintos núcleos. Para determinar los bancos asignados a tareas críticas y tareas no críticas proponen un método de Programación Lineal que garantiza el cumplimiento de las condiciones de planificabilidad.

Otra propuesta, como la de Bakita et al. [31] incorporan el uso seguro de lectura múltiple simultánea. Para el control de interferencias, añaden al particionamiento combinado de bancos y conjuntos un novedoso mecanismo de aislamiento de caché multinivel. Consideran un conjunto de tareas de tres niveles de criticidad y determinan el tamaño de las particiones utilizando el algoritmo implementado en [30]. En esta variante los bancos de caché asignadas a las tareas de criticidad A y B que se ejecutan en un núcleo se dividen entre los hilos mediante el coloreado de páginas. En la caché L2 privada, el aislamiento entre hilos dependerá del número total de colores disponibles. Para garantizar el aislamiento entre hilos en las cachés L2 y L3, cualquier decisión de color en la caché L2 se aplica también a la caché L3.

El bloqueo de caché para reducir las interferencias en sistemas multicore de tiempo real fue utilizado por Mancuso et al.[32] con la implementación de una técnica de bloqueo por colores. Esta técnica se basa en el coloreado de páginas y el bloqueo en caché de las páginas de memoria más frecuentemente utilizadas. Su implementación requiere un perfil de memoria de la aplicación que permita identificar las páginas más frecuentemente accedidas. El algoritmo implementado asigna a cada página frecuentemente utilizada una posición de caché dada por un color y banco determinado. Una vez cargada la página desde memoria principal en la posición previamente asignada se bloquea mediante un mecanismo de bloqueo hardware basado en bancos garantizando que sean aciertos de caché el resto de accesos a estas páginas. Awan et al.[33] emplean la técnica de bloqueo por colores[32] en un sistema de criticidad mixta para evitar las interferencias Intracore e Intercore en la caché compartida. Proponen un método dinámico que utiliza dos modos de funcionamiento (bajo y alto). Las tareas críticas tienen que considerar dos estimaciones WCET diferentes (baja y alta). Las tareas no críticas sólo tienen una estimación WCET. El sistema comienza en el modo bajo y cambia al modo alto cuando el WCET obtenido para las tareas críticas supera la estimación baja. Cuando el modo es alto, el sistema recupera las páginas de caché asignadas a las tareas no críticas para redistribuirlas entre las tareas críticas. El análisis supone que para cada tarea ya se ha determinado el número de páginas frecuentemente utilizadas que están bloqueadas en la caché en cada modo.

III. Simulador de caché propuesto

En este artículo presentamos un nuevo simulador de caché funcional basado en trazas. El objetivo de este simulador es evaluar las interferencias de caché en sistemas de criticidad mixta y probar e investigar nuevas técnicas que mejoren el rendimiento. Para lograr este objetivo se utilizan características como el particionamiento hardware y software, el bloqueo y la gestión multinivel del sistema de caché. Con este simulador pueden evaluarse aplicaciones reales utilizando sus trazas de ejecución. Además de emular el comportamiento normal de un sistema de caché y contar con mecanismos para la implementación de técnicas de última generación, el simulador incluye algoritmos para la realización del perfil de memoria de las aplicaciones y un planificador de tareas de criticidad mixta.

A. Generación de trazas de ejecución y perfiles de memoria de las aplicaciones

Las trazas para este simulador se generan a partir de la ejecución de aplicaciones reales. Para aquellas aplicaciones que puedan tener diferentes flujos de ejecución se considera aquella con el peor tiempo de ejecución. Las trazas de ejecución pueden obtenerse utilizando un depurador en línea como GDB. En este trabajo hemos desarrollado un script en Python que se ejecuta dentro del entorno GDB. Este script es capaz de ejecutar la aplicación paso a paso a nivel de ensamblador. Entonces, para cada instrucción, puede detectar si la instrucción es una lectura o escritura en memoria y en qué dirección de memoria se está leyendo o escribiendo. El resultado es una traza que muestra, para cada operación de memoria, si es una operación de acceso a instrucciones o datos, la dirección de la operación y si es una lectura o una escritura.

Todas las direcciones incluidas en las trazas de ejecución son direcciones virtuales. Esto permite múltiples ejecuciones de la misma traza como si hubiera diferentes aplicaciones en ejecución. Las direcciones virtuales se traducen a direcciones físicas cada vez que el simulador ejecuta una operación de memoria a partir de la traza. Cada proceso tiene su propia tabla de páginas. La tabla de páginas contiene el marco de página física asignado a la página virtual identificada por un número de página. Los marcos de páginas físicas se asignan a páginas virtuales de cada proceso cuando se accede por primera vez a la página. La tabla de páginas se borra y los marcos de página se liberan una vez terminada la ejecución del proceso. Las mismas trazas de la ejecución real de una aplicación pueden considerarse aplicaciones diferentes y tendrán un uso diferente de las líneas de caché porque tienen tablas de páginas independientes.

Al simulador se le añade un algoritmo para realizar un perfil de memoria de las aplicaciones a partir de las trazas de ejecución. Del perfil puede conocerse con exactitud el total de bytes de memoria que utiliza la aplicación durante su ejecución, así como información relativa a las páginas virtuales frecuentemente utilizadas. Este perfil se realiza sumando todos los accesos de memoria a cada página virtual y a cada bloque en una página. A continuación, las páginas se ordenan por el número de accesos. Esta información es necesaria para la toma de decisiones en la asignación de recursos de caché a las aplicaciones, así como para la implementación de técnicas avanzadas.

B. Configuración y parámetros

El simulador propuesto en este artículo es altamente configurable. Por lo tanto, es capaz de simular múltiples arquitecturas diferentes. Algunas características de las arquitecturas que se pueden configurar son las siguientes:

- Número de núcleos.
- Número de niveles de caché en la jerarquía de memoria.
- Acceso exclusivo o compartido por nivel de caché.
- Cachés inclusivas o exclusivas.
- Tamaño en bytes de cada nivel de caché.
- Tamaño de una línea de caché en bytes.
- Número total de bancos por conjunto en cada nivel de caché.
- Políticas por nivel de caché.
- Tamaño de dirección física y tamaño de página virtual.

C. Comportamiento y detalles de funcionamiento

El comportamiento del simulador puede descomponerse en tres etapas principales: la ejecución en paralelo de diferentes trazas de aplicaciones, la traducción de direcciones virtuales a físicas y el funcionamiento real de una jerarquía de cachés. Este entorno es capaz de simular la ejecución en paralelo de una aplicación diferente en cada uno de los núcleos. Este comportamiento se simula asignando una traza de aplicación para cada núcleo y ejecutando una instrucción de la traza en cada núcleo en cada ronda de ejecución. Las aplicaciones asignadas al mismo núcleo se ejecutan según indique el planificador.

El simulador también se encarga de traducir las direcciones virtuales a direcciones físicas. Se almacena una tabla de páginas para cada tarea que se ejecuta en el simulador. Cuando se solicita una nueva página virtual por primera vez, el simulador selecciona un marco de página aleatorio que no se haya utilizado antes en ninguna otra tarea. Los marcos de página utilizados se almacenan en un contenedor para comprobar que no se volverán a utilizar en otra tarea.

Cada nivel de caché en el simulador se implementa utilizando un modelo asociativo por conjuntos. Parámetros de la caché como el número de bancos por conjunto, el número de conjuntos y el tamaño de cada bloque de caché pueden configurarse a demanda. Cada nivel de caché puede ser privada para cada núcleo o compartida entre todos los núcleos. Además, pueden dividirse entre caché solo de instrucciones o datos, o combinar ambos tipos de valores.

Los fallos y aciertos de caché siguen el comportamiento de una jerarquía de caché real, en función de la política de escritura, política de reemplazo, etc. asignada a la caché. En el caso de un fallo en la caché de datos privada de un determinado núcleo, el simulador implementa el protocolo MESI y se busca el bloque deseado en las demás cachés de datos del mismo nivel antes de solicitarlo a la caché de siguiente nivel. EL protocolo MESI es un protocolo de coherencia de caché para garantizar que los datos almacenados en caché sean coherentes entre múltiples núcleos, fundamental en aplicaciones que pueden migrar de un núcleo a otro.

El simulador tiene implementado dos políticas de sustitución: política de sustitución pseudoaleatoria y política de sustitución del menos recientemente utilizado. Además implementa la política de escritura retardada. Si el bloque de caché que se va a sustituir tiene activado el Dirty bit, se realiza una escritura retardada en el siguiente nivel de memoria antes de realizar la sustitución. Otras políticas de sustitución, así como otras políticas de escritura pueden ser añadidas sin gran dificultad.

D. Características implementadas para reducir las interferencias

Este simulador implementa tres características que son muy útiles para implementar las técnicas más avanzadas para mitigar las interferencias y mejorar el WCET de las aplicaciones críticas. Estas características son: Particionamiento hardware de los bancos, Bloqueo de bancos/líneas y Coloreado de páginas.

La función de particionamiento hardware de los bancos de caché limita los bancos que puede utilizar un núcleo o tarea específica. Se implementa utilizando un registro especial en cada núcleo que indica qué bancos de la caché compartida de último nivel puede utilizar este núcleo.

La función de bloqueo de bancos puede implementarse en cualquier nivel de caché. En el simulador el bloqueo se implementa utilizando un registro especial que indica qué bancos de la caché no pueden sustituir sus líneas de caché correspondientes. El simulador también permite el bloqueo a nivel de línea. En este caso se utiliza otro registro especial para indicar qué líneas de la caché deben bloquearse y no podrán ser reemplazadas.

La función de coloreado de páginas es un mecanismo de particionamiento software implementado en la traducción de direcciones virtuales a direcciones físicas. Los colores de caché que utilizará una aplicación depende de la asignación de marcos de página físicos a cada página virtual. El total de colores de caché disponibles dependerá de los parámetros de la caché. El simulador implementa el coloreado de páginas asignando marcos de página de forma aleatoria para determinados colores de caché. Esta versión de la función es capaz de seleccionar un marco de página aleatorio que pertenezca a un color o conjunto de colores predeterminado. Cada tarea que se ejecuta en un núcleo puede configurar el registro de color de este núcleo en cualquier momento. Este registro se utiliza para seleccionar un marco de página aleatorio perteneciente a estos colores cada vez que este núcleo accede a una nueva página virtual.

E. Planificador de tareas en tiempo real

El planificador implementado se incluye en el amplio ámbito de investigación relativo a la asignación

de recursos en sistemas de criticidad mixta iniciado por Vestal et al.[34]. Este considera que en la planificación de sistemas de criticidad mixta los test de planificabilidad para las tareas de menor criticidad sean modificados para incorporar estimaciones del tiempo ejecución en el peor de los casos (WCET) menos pesimistas para tareas de mayor criticidad. Con lo cual, múltiples estimaciones del WCET debe asignarse a cada tarea, una estimación para evaluar la planificabilidad en su propio nivel de criticidad y otras estimaciones para evaluar la planificabilidad de tareas de menor nivel de criticidad.

En nuestra arquitectura se consideran tareas de 4 niveles de criticidad: nivel A,B,C y D. Las tareas en cada nivel de criticidad emplean un planificador independiente con su política de planificación determinada.

Las tareas de nivel A son las tareas de mayor criticidad. Estas tareas son particionadas entre todos los núcleos por lo que se requiere un planificador para tareas de nivel A en cada núcleo. Las tareas de nivel B tienen la segunda mayor prioridad y también están particionadas entre todos los núcleos. El contenedor de tareas de nivel A en cada núcleo y el contenedor de tareas de nivel B en cada núcleo emplean un planificador Rate Monotonic [35] (menor periodo implica mayor prioridad). Además se considera el algoritmo heurístico Best Fit (seleccionar el núcleo que mejor se ajuste), aplicando la condición de planificabilidad conocida como Increasing Period, para repartir las tareas entre los núcleos. Las tareas de nivel C son tareas de tiempo real de menor prioridad, que poseen suaves restricciones temporales. También son particionadas entre todos los núcleos y utilizan un planificador Earliest Deadline First. Por último, las tareas de nivel D son tareas no críticas y sin restricciones temporales. Estas tareas no estás asignadas a los procesadores sino que están planificadas según el principio del mejor esfuerzo.

IV. Evaluación

Para evaluar el potencial del simulador hemos seleccionado tres de las técnicas propuestas en la bibliografía para mitigar la interferencia en sistemas de criticidad mixta. Su implementación requiere del empleo de muchas de las características añadidas al simulador de caché: particionamiento, bloqueo y gestión multinivel de caché. Las técnicas seleccionadas son las siguientes:

- Particionamiento combinado por conjuntos y bancos: Requiere que el simulador gestione el particionamiento hardware de los bancos de caché L2 y el particionamiento software de los conjuntos de caché L2 a través del coloreado de páginas.
- Gestión multinivel de la caché: Requiere que el simulador gestione el particionamiento software (coloreado de páginas) integrando ambos niveles de caché.
- Bloqueo por colores: Requiere que el simulador gestione el bloqueo hardware de los bancos y el particionamiento software (coloreado de páginas) para la caché de nivel 2. Requiere además perfil de memoria de las aplicaciones críticas.

A. Entorno de evaluación

Como banco de pruebas se utilizan conjuntos de tareas que pertenecen a la suite de benchmarks de tiempo real SNU. La suite conocida como SNU Real-Time Benchmarks contiene fuentes en C que incluyen programas de cálculo y algoritmos como:

- Búsqueda binaria(bs).
- Ordenación por inserción(insertsort).
- Ecuaciones lineales simultáneas por descomposición LU(ludcmp).
- Inversión matricial(minver).
- Versión no recursiva del algoritmo de ordenación rápida(qsortexam).

Al conjunto de tareas de tiempo real se le añaden además dos aplicaciones intensivas en memoria, diseñadas para crear suficientes interferencias al ejecutarse en paralelo en distintos núcleos. Estas aplicaciones son:

- Filtro gaussiano: La huella de memoria de esta aplicación combina el acceso secuencial en un array 2D con el acceso recurrente a un pequeño conjunto de datos.
- Suma de matrices: La huella de memoria es el acceso secuencial de varios datos de una matriz.

Para esta evaluación vamos a simular la arquitectura de un ARM Cortex R-82 de 4 núcleos, con las siguientes jerarquías de caché:

- L1 cache: Dos cachés L1 asociativas separadas para datos e instrucciones privadas de cada núcleo. Los parámetros de ambas incluyen 4 bancos por caché, 32 KB de tamaño de caché y 64 bytes de tamaño de línea.
- L2 cache: Una caché L2 asociativa compartida entre todos los núcleos. Los parámetros de la caché L2 incluyen 8 bancos,128 KB de tamaño de caché y 64 bytes de tamaño de línea.

Estas tareas se combinan para crear un entorno simulado de criticidad mixta. Se asignan diferentes criticidades a las tareas de tiempo real y las tareas intensivas en memoria son no críticas (criticidad D). Finalmente el conjunto a ejecutar se compone de 8 tareas de criticidad A, 6 tareas de criticidad B, 6 tareas de criticidad C y 4 tareas de criticidad D. Las aplicaciones de criticidad A se ejecutan cada 1 ms, las aplicaciones de criticidad B en un período de 5 ms y las aplicaciones de criticidad C en un período de 10 ms. Las aplicaciones de criticidad D se ejecutan siempre que exista un núcleo disponible. EL simulador se evalúa en una ventana de ejecución de 30 ms.

Al ejecutar el simulador se obtiene un fichero de salida por cada núcleo con las estadísticas de caché para cada ejecución de cada una de las tareas que se ejecuten en ese núcleo. La Figura 1 muestra las

estadísticas de una ejecución de la tarea con identificador 1 en el Núcleo 1. El fichero de salida incluye las estadísticas de fallas y aciertos para cada una de las cachés, la tasa de fallos y aciertos, una estimación del tiempo promedio que lleva acceder a la memoria principal para leer o escribir datos (AMAT), así como una estimación del tiempo de ejecución de la tarea, aspectos necesarios para medir el rendimiento de la caché.

```
App:    1

L1_D_1_accesses:        821
L1_D_1_hit_cache:       816
L1_D_1_miss_cache:      5
L1_D_1_writebacks:      0

L1_I_1_accesses:        1898
L1_I_1_hit_cache:       1891
L1_I_1_miss_cache:      7
L1_I_1_writebacks:      0

L2_accesses:    12
L2_hit_cache:   0
L2_miss_cache:  12
L2_writebacks:  0

miss_rate_L1_D:         6.090134e-03
miss_rate_L1_I:         3.688093e-03
miss_rate_L2:   1

AMAT_Datos(ciclos):     1.669915e+00
AMAT_Datos(microsegundos):      1.001949e-03

AMAT_Instrucciones(ciclos):     1.405690e+00
AMAT_Instrucciones(microsegundos):      8.434141e-04

Execution_time(ciclos):         3218
Execution_time(microsegundos):  1.930800e+00
```

Fig. 1: Fichero salida Núcleo 1

En la implementación de los algoritmos para mitigar las interferencias se consideran como tareas críticas las tareas de Nivel A y Nivel B, y se consideran como tareas no críticas las tareas de Nivel C y D. Las Figuras 2,3 y 4 proporcionan información sobre la media y la variabilidad del tiempo de ejecución, fallas en la caché L1 de Datos y fallas en la caché L2 del total de ejecuciones de cada tarea en función de su nivel de criticidad. Debido a que la interferencia que sufren las aplicaciones críticas en la caché L1 de Instrucciones no es de gran magnitud no se incluyen en este artículo sus estadísticas.

B. *Discusión de los resultados al implementar el particionamiento combinado por conjuntos y bancos*

Esta técnica implica el particionamiento hardware de los bancos de caché L2 entre tareas críticas (nivel A y B) y tareas de menor criticidad (nivel C y D), y el particionamiento software(coloreado de páginas) para distribuir los colores de caché L2 en los bancos asignados a las tareas críticas entre todos los núcleos.

Cuatro bancos de la caché L2 se utilizan para tareas críticas y las restantes para tareas no críticas. Cada núcleo que ejecuta una aplicación crítica sólo puede utilizar 1 de los 4 colores. Por el contrario, los núcleos que ejecutan una aplicación no crítica pueden utilizar todos los colores en los bancos correspondientes.

Los resultados en la Figura 2 muestran una disminución del tiempo de ejecución de las tareas de criticidad A (Figura 2a) y B (Figura 2b) al implementar el algoritmo de particionamiento combinado de conjuntos y bancos (SetWay) en comparación a cuando no se implementa ningún algoritmo para mitigar las interferencias (SinAlgoritmo). Al asignar espacios de caché privados para las tareas críticas que se ejecutan en cada núcleo se reduce la interferencia de las tareas no críticas sobre las tareas de mayor criticidad en la caché L2 compartida y esto se traduce en una disminución de las fallas de caché L2 (Figura 4a,Figura 4b) y consecuentemente una reducción del tiempo de ejecución de las tareas críticas. Esta técnica no considera la interferencia en las cachés L1 privadas para cada núcleo, lo que se evidencia en el comportamiento de las fallas en la caché L1 en la Figura 3a y Figura 3b. Al aplicar esta técnica (SetWay) las fallas de caché L1 muestran un comportamiento similar a la ejecución de las aplicaciones sin la implementación de ningún algoritmo de mitigación de interferencias (SinAlgoritmo). En las cachés L1 las aplicaciones críticas sufren la interferencia de aplicaciones de menor criticidad que se ejecutan en el mismo núcleo y comparten las cachés L1 privada.

En el caso de las aplicaciones de tiempo real no críticas (aplicaciones de Nivel C) comparten el mismo espacio de caché L2 con las aplicaciones de Nivel D intensivas en memoria, por lo que su rendimiento no mejora al aplicar esta técnica (SetWay) y las estadísticas de tiempo de ejecución (Figura 2c), fallas de caché L1 (Figura 3c) y fallas de caché L2 (Figura 4c) no varían significativamente en comparación a cuando las aplicaciones se ejecutan sin implementar ningún algoritmo (SinAlgortimo).

C. *Discusión de los resultados al implementar el manejo multinivel de la caché*

Esta técnica implica el particionamiento software (coloreado de páginas) en ambos niveles de la caché, de forma que las aplicaciones críticas y las aplicaciones de menor criticidad utilicen un conjunto diferente de colores en ambos niveles de la caché. La técnica de gestión de caché multinivel asigna espacios privados de caché L1 y L2 a las tareas críticas y no críticas que se ejecutan en cada núcleo.

Los resultados en la Figura 2 muestran una disminución del tiempo de ejecución de las tareas de Nivel A (Figura 2a) y B (Figura 2b) al implementar el algoritmo de manejo multinivel de la caché (Multilevel) en comparación a cuando no se implementa ningún algoritmo para mitigar las interferencias (SinAlgortimo). Esta técnica mitiga la interferencia de las tareas no críticas sobre las tareas de mayor criticidad en todos los niveles de caché y esto se traduce en una reducción de las fallas de caché L1 (Figura 3a y Figura 3b) y las fallas de caché L2 (Figura 4a y Figura4b) de las aplicaciones de nivel A y B y consecuentemente una reducción del tiempo de ejecución de las tareas críticas.

En el caso de las aplicaciones de tiempo real no críticas (aplicaciones de Nivel C) comparten el mismo espacio de caché L1 y L2 con las aplicaciones de Nivel D intensivas en memoria, por lo que su rendi-

miento no mejora al aplicar esta técnica (Multilevel) y las estadísticas de tiempo de ejecución (Figura 2c), fallas de caché L1 (Figura 3c) y fallas de caché L2 (Figura 4c) no varían significativamente en comparación a cuando las aplicaciones se ejecutan sin implementar ningún algoritmo (SinAlgortimo).

D. Discusión de los resultados al implementar el bloqueo por colores

Esta técnica implica el bloqueo a nivel de bancos de caché L2. Asigna colores específicos a las páginas deseadas y las bloquea. Requiere realizar un perfil de memoria de las aplicaciones críticas para ordenar las páginas según la frecuencia de acceso. El número de bancos a bloquear se obtiene contabilizando cuántas páginas se requiere bloquear en total. Considerando que se requiere que sean aciertos al menos el 80 por ciento de los accesos de las aplicaciones críticas (aplicaciones de criticidad A o B) a la caché L2, se bloquean las 2 páginas más frecuentemente utilizadas de cada aplicación crítica en la caché, quedando solo el 12.5 por ciento de la caché L2 disponible para páginas no bloqueadas y aplicaciones no críticas.

A partir del perfil de memoria y una vez decididas las páginas a bloquear en la caché L2 es posible determinar el peor tiempo de acceso a memoria de la aplicación. Para ello se consideran fallas todos los accesos a las cachés L1 y fallas de caché L2 a todos los accesos a las páginas no bloqueadas.

Para las aplicaciones de Nivel A el bloqueo de las 2 páginas más frecuentemente utilizadas se traducen en el acierto de caché L2 para más del 90 por ciento de los accesos a memoria de estas aplicaciones, a excepción de la tarea 3 y 7 donde el bloqueo de estas páginas se traduce en el acierto de caché L2 del 80 por ciento de los accesos. El bloqueo de las 2 páginas más frecuentemente utilizadas de las aplicaciones de Nivel B representa el acierto de caché L2 para más del 90 por ciento de los accesos de las tareas 1,3 y 5, y el 87 por ciento para las tareas 2,4 y 6. Con lo cual, en el peor de los casos para las tareas de Nivel A, por accesos a líneas no bloqueadas en la caché L2, puede estimarse que las tareas 1 y 5 pueden tener 4 fallas, las tareas 2 y 6 pueden tener 200 fallas, las tareas 3 y 7 pueden tener 794 fallas y las tareas 4 y 8 pueden tener 261 fallas. En el peor de los casos para las tareas de Nivel B, por accesos a líneas no bloqueadas, puede estimarse que las tareas 1,3 y 5 pueden tener 410 fallas de caché L2 y las tarea 2, 4 y 6 pueden tener 4040 fallas de caché L2.

El análisis anterior evidencia que el bloqueo garantiza un comportamiento predecible para las aplicaciones críticas. Sin embargo, limita considerablemente el espacio disponible para el resto de accesos a líneas no bloqueadas y líneas de aplicaciones no críticas que comparten el mismo espacio de caché. La Figura 4a y Figura 4b muestran que con la implementación de esta técnica (ColoredLockdown) el comportamiento de las fallas de caché L2 para las aplicaciones críticas (Nivel A y B) se mantienen por debajo del peor de los casos posibles en cada caso, este comportamiento puede ser o no mejor que en la ejecución sin la implementación de ningún algoritmo (SinAlgoritmo) en función de la magnitud de la interferencia que sufra la aplicación por compartir recursos con otras aplicaciones de menor criticidad en este caso.

Esta técnica no considera la interferencia en las cachés L1 privadas para cada núcleo, lo que se evidencia en el comportamiento de las fallas en la caché L1 en la Figura 3a y Figura 3b. Al aplicar esta técnica (ColoredLockdown) las fallas de caché L1 muestran un comportamiento similar a la ejecución de las aplicaciones sin la implementación de ningún algoritmo de mitigación de interferencias (SinAlgoritmo). En las cachés L1 las aplicaciones críticas sufren la interferencia de aplicaciones de menor criticidad que se ejecutan en el mismo núcleo y comparten las cachés L1 privada.

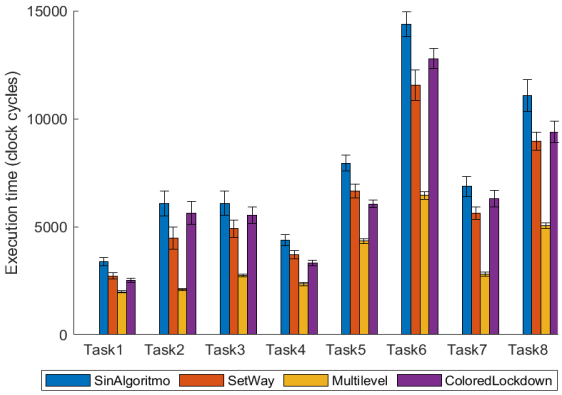
En el caso de las aplicaciones de tiempo real no críticas (tareas Nivel C) se reduce significativamente el espacio de caché L2 disponible que además comparten con las aplicaciones de Nivel D intensivas en memoria, por lo que su rendimiento puede ser igual o deteriorarse al aplicar esta técnica (ColoredLockdown) y las estadísticas de tiempo de ejecución (Figura 2c), fallas de caché L1 (Figura 3c) y fallas de caché L2 (Figura 4c) no mejoran en comparación a cuando las aplicaciones se ejecutan sin implementar ningún algoritmo (SinAlgortimo).

V. Conclusiones

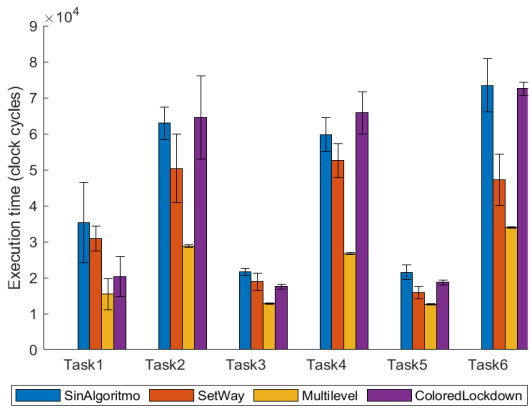
Este artículo implementa un simulador de caché basado en trazas para evaluar la interferencia de la caché en cargas de trabajo de criticidad mixta en tiempo real. El entorno del simulador propuesto en este artículo es altamente configurable y es capaz de simular muchas arquitecturas diferentes. Este trabajo también implementa una solución ingeniosa para obtener las trazas de entrada de las aplicaciones. La solución implementada puede adaptarse fácilmente a cualquier plataforma hardware.

Las pruebas realizadas con el simulador demuestran su potencial para detectar interferencias cuando las tareas acceden simultáneamente al recurso de caché compartido. Las funciones de particionamiento hardware de los bancos, bloqueo basado en bancos/líneas y coloreado de páginas implementadas en el simulador permitieron la ejecución de técnicas de última generación. La implementación de estas técnicas demuestra la minimización de las interferencias y, en consecuencia, la reducción de las pérdidas de caché. El análisis muestra que este simulador de caché es una herramienta útil para estudiantes e investigadores que evalúan las interferencias en sistemas de tiempo real.

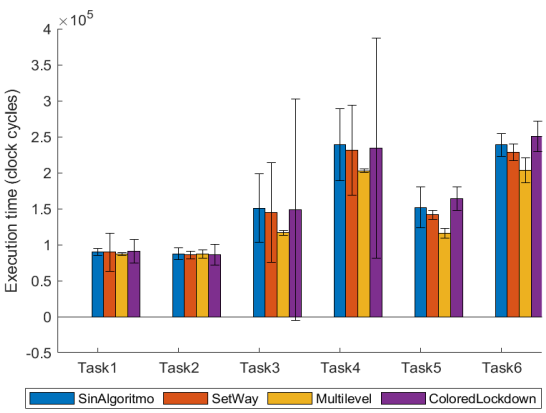
Como futuras líneas de investigación, se plantea la implementación de algoritmos que permitan minimizar las interferencias y utilizar la caché de forma más eficiente en comparación con las técnicas implementadas en la literatura.



(a) Tareas de Nivel A

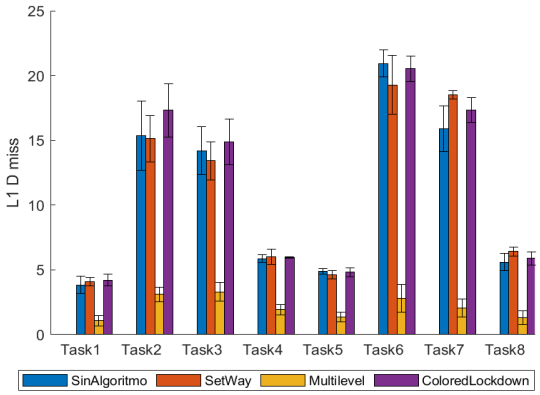


(b) Tareas de Nivel B

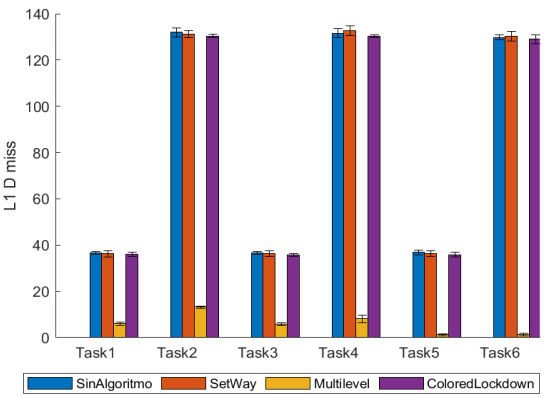


(c) Tareas de Nivel C

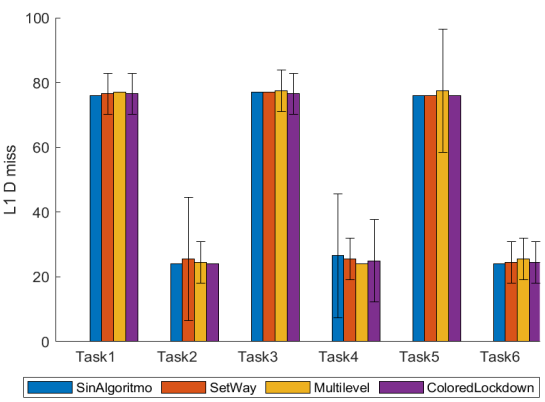
Fig. 2: Evaluación tiempo de ejecución de las tareas



(a) Tareas de Nivel A



(b) Tareas de Nivel B



(c) Tareas de Nivel C

Fig. 3: Estadísticas de fallas en la caché L1 de Datos

Reconocimientos

Este trabajo ha sido fianciado parcialmente por el gobierno de la Comunidad de Madrid con el doctorado industrial IND2019/TIC-17261 "Nuevas técnicas de desarrollo de software de tiempo real embarcado para plataformas. MPSoC de próxima generación."

Referencias

[1] Dakshina Dasari, Benny Akesson, Vincent Nelis, Muhammad Ali Awan, and Stefan M Petters, "Identifying the sources of unpredictability in cots-based multicore systems," in *2013 8th IEEE international symposium on industrial embedded systems (SIES)*. IEEE, 2013, pp. 39–48.

[2] Andreas Löfwenmark and Simin Nadjm-Tehrani, "Understanding shared memory bank access interference in multi-core avionics," in *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

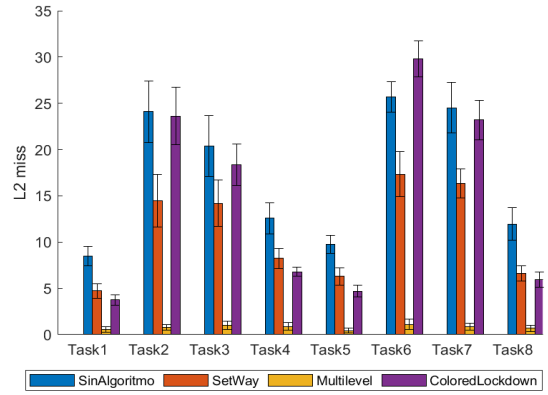
[3] Sparsh Mittal, "A survey of techniques for cache partitioning in multicore processors," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–39, 2017.

[4] Myoungjun Lee and Soontae Kim, "Time-sensitivity-aware shared cache architecture for multi-core embedded systems," *The Journal of Supercomputing*, vol. 75, no. 10, pp. 6746–6776, 2019.

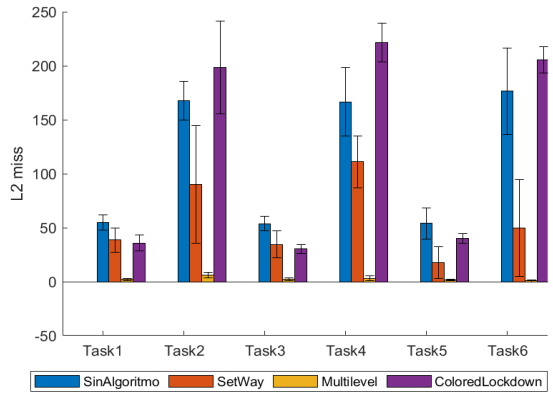
[5] CAT Intel, "Improving real-time performance by utilizing cache allocation technology," *Intel Corporation, April*, 2015.

[6] Lucia Pons, Vicent Selfa, Julio Sahuquillo, Salvador Petit, and Julio Pons, "Improving system turnaround time with intel cat by identifying llc critical applications," in *European Conference on Parallel Processing*. Springer, 2018, pp. 603–615.

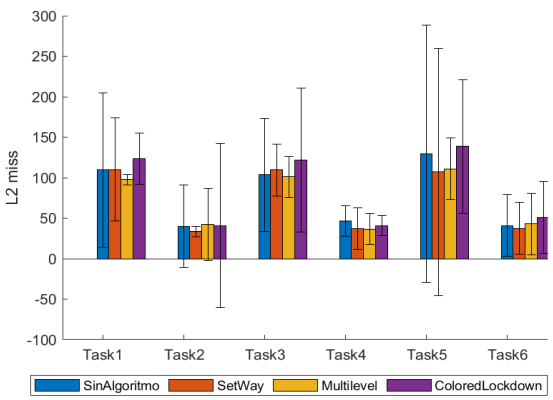
[7] Zenepe Satka and Hena Hodžić, "Minimizing the unpredictability that real-time tasks suffer due to inter-core cache interference.," 2020.



(a) Tareas de Nivel A



(b) Tareas de Nivel B



(c) Tareas de Nivel C

Fig. 4: Estadísticas de fallas en la caché L2

[8] Meng Xu, Robert Gifford, and Linh Thi Xuan Phan, "Holistic multi-resource allocation for multicore real-time virtualization," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[9] Robert Gifford, Neeraj Gandhi, Linh Thi Xuan Phan, and Andreas Haeberlen, "Dna: Dynamic resource allocation for soft real-time multicore systems," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 196–209.

[10] Hyoseung Kim, Arvind Kandhalu, and Ragunathan Rajkumar, "A coordinated approach for practical os-level cache management in multi-core real-time systems," in *2013 25th Euromicro Conference on Real-Time Systems*. IEEE, 2013, pp. 80–89.

[11] Hyoseung Kim and Ragunathan Rajkumar, "Real-time cache management for multi-core virtualization," in *2016 International Conference on Embedded Software (EMSOFT)*. IEEE, 2016, pp. 1–10.

[12] Yoojin Lim and Hyoseung Kim, "Cache-aware real-time virtualization for clustered multi-core platforms," *IEEE Access*, vol. 7, pp. 128628–128640, 2019.

[13] Tomasz Kloda, Marco Solieri, Renato Mancuso, Nicola Capodieci, Paolo Valente, and Marko Bertogna, "Deterministic memory hierarchy and virtualization for modern multi-core embedded systems," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 1–14.

[14] Abu Asaduzzaman, Vidya R Suryanarayana, and Fadi N Sibai, "On level-1 cache locking for high-performance low-power real-time multicore systems," *Computers & Electrical Engineering*, vol. 39, no. 4, pp. 1333–1345, 2013.

[15] Abhik Sarkar, Frank Mueller, and Harini Ramaprasad, "Static task partitioning for locked caches in multicore real-time systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 1, pp. 1–30, 2015.

[16] Wenguang Zheng, Hui Wu, and Chuanyao Nie, "Integrating task scheduling and cache locking for multicore real-time embedded systems," *ACM Sigplan Notices*, vol. 52, no. 5, pp. 71–80, 2017.

[17] Alexy Torres Aurora Dugo, Jean-Baptiste Lefoul, Felipe Gohring De Magalhaes, Dahman Assal, and Gabriela Nicolescu, "Cache locking content selection algorithms for arinc-653 compliant rtos," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–20, 2019.

[18] T. Zhang, W. Zheng, Y. Xiao, and G. Xu, "A dynamic instruction cache locking approach for minimizing worst case execution time of a single task," *IEEE Access*, vol. 8, pp. 208003–208015, 2020.

[19] Hadi Brais, Rajshekar Kalayappan, and Preeti Ranjan Panda, "A survey of cache simulators," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–32, 2020.

[20] Philippe Owezarski and Nicolas Larrieu, "A trace based method for realistic simulation," in *2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577)*. IEEE, 2004, vol. 4, pp. 2236–2239.

[21] Richard A Uhlig and Trevor N Mudge, "Trace-driven memory simulation: A survey," *ACM Computing Surveys (CSUR)*, vol. 29, no. 2, pp. 128–170, 1997.

[22] Jan Edler, "Dinero iv: Trace-driven uniprocessor cache simulator," *http://www. cs. wisc. edu/˜ markhill/DineroIV*, 1994.

[23] Ravi Iyer, "On modeling and analyzing cache hierarchies using casper," in *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003.* IEEE, 2003, pp. 182–187.

[24] Charles F Shelor and Krishna M Kavi, "Moola: Multicore cache simulator," in *30th International Conference on Computers and Their Applications CATA-2015*, 2015.

[25] Derek Bruening and Saman Amarasinghe, *Efficient, transparent, and comprehensive runtime code manipulation*, Ph.D. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering . . . , 2004.

[26] Hui Kang and Jennifer L Wong, "vcsimx86: a cache simulation framework for x86 virtualization hosts," *Stony Brook University*, 2013.

[27] M Vega-Rodriguez, R Martin, and F Gallardo, "Smpcache: Simulator for cache memory systems on symmetric multiprocessors, january 2006," *URL http://arco. unex. es/smpcache/. http://arco. unex. es/smpcache.*

[28] Giovani Gracioli, Ahmed Alhammad, Renato Mancuso, Antônio Augusto Fröhlich, and Rodolfo Pellizzoni, "A survey on cache management mechanisms for real-time embedded systems," *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, pp. 1–36, 2015.

[29] Alan Burns and Robert I Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–37, 2017.

[30] Micaiah Chisholm, Bryan C Ward, Namhoon Kim, and James H Anderson, "Cache sharing and isolation tradeoffs in multicore mixed-criticality systems," in *2015 IEEE Real-Time Systems Symposium*. IEEE, 2015, pp. 305–316.

[31] Joshua Bakita, Shareef Ahmed, Sims Hill Osborne, Stephen Tang, Jingyuan Chen, F Donelson Smith, and James H Anderson, "Simultaneous multithreading in mixed-criticality real-time systems," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 278–291.

[32] Renato Mancuso, Roman Dudko, Emiliano Betti, Marco

Cesati, Marco Caccamo, and Rodolfo Pellizzoni, "Real-time cache management framework for multi-core architectures," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2013, pp. 45–54.

[33] Muhammad Ali Awan, Konstantinos Bletsas, Pedro F Souto, Benny Akesson, and Eduardo Tovar, "Mixed-criticality scheduling with dynamic redistribution of shared cache," *arXiv preprint arXiv:1704.08876*, 2017.

[34] Steve Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE international real-time systems symposium (RTSS 2007)*. IEEE, 2007, pp. 239–243.

[35] Omar U Pereira Zapata and Pedro Mejıa Alvarez, "Edf and rm multiprocessor scheduling algorithms: Survey and performance evaluation," *Seccion de Computacion Av. IPN*, vol. 2508, 2005.

Análisis seguro y automático del reúso de datos en aplicaciones de tiempo real estricto sobre arquitectura x64

Álvaro Moreno Martín Viveros[1], Juan Segarra Flor[2], Rubén Gran Tejero[2] y Adrià Armejach Sanosa[1]

Resumen— **El análisis del reúso de datos presente en una aplicación es una de las técnicas que ha demostrado ser efectiva para obtener mejores resultados en el cálculo del tiempo de ejecución en el peor caso en sistemas con memorias cache de datos. Como prueba de concepto de esta metodología han sido desarrolladas herramientas como *polygaz*, una utilidad que permite extraer de forma segura y fiable una clasificación del reúso de datos presente en binarios compilados sobre arquitectura ARMv7. Esta información independiente de la memoria cache utilizada permite, posteriormente, realizar un cálculo más preciso de el tiempo de ejecución en el caso peor sobre el sistema de tiempo real que se busque estudiar. En este artículo se pretende explorar la funcionalidad de la herramienta al analizar binarios compilados sobre otra arquitectura mayoritaria (x64) y que presenta una filosofía distinta a ARMv7 (CISC vs RISC). Además, se estudia como puede afectar a los resultados en el reúso de datos obtenido emplear compiladores distintos (*clang* y *gcc*) a la hora de generar binarios para la misma aplicación de alto nivel. De esta manera, se ha ampliado la herramienta para permitir el análisis de la nueva arquitectura, y se ha planteado un entorno experimental donde se ha analizado un subconjunto de la colección TACLeBench usando *gcc* y *clang* con dos niveles de optimización (O0 y O2). Los resultados obtenidos muestran cómo la metodología de análisis aplicada permite también capturar el reúso de datos en la arquitectura x64 de forma efectiva. Además, muestran el efecto de utilizar compiladores diferentes pudiéndose identificar diferencias en cómo se manifiesta el reúso de datos capturado entre ambas soluciones. Por último, a través de una estimación sencilla de los previsibles aciertos y fallos en base al reúso capturado se calcula para ambos compiladores una reducción del tiempo en el peor caso de ejecución que supone alrededor del 20 % del tiempo invertido en un sistema pesimista que solo predice fallos. Con optimizaciones este valor se estima que rondaría el 10 %.**

Palabras clave— **Reúso de datos, x64, WCET.**

I. Introducción

Dentro de la creciente variedad de sistemas informáticos que se pueden encontrar hoy en día, los sistemas de tiempo real ocupan una parte cada vez más importante. Habiendo numerosos ejemplos de su uso en sectores tan diversos como el de la automoción, la aviónica, la robótica o la medicina. Un sistema de tiempo real, por definición, consta de un número de tareas cuya funcionalidad debe ser cumplida atendiendo a unas restricciones temporales determinadas con el fin de garantizar un buen funcionamiento del sistema. Para garantizar el correcto cumplimiento de estas restricciones, se debe realizar la planificación de las tareas del sistema en base al tiempo en el peor caso de ejecución (WCET), el cual debe ser calculado formalmente. La mayor parte de métodos de análisis de WCET se basan en el estudio del flujo de control del programa y la interacción de este con el hardware del sistema. De tal forma que esta información pueda ser utilizada para construir un modelo de Programación Lineal Entera (ILP) que permita resolver el problema [1].

Dentro de este análisis, la interacción entre el programa y el hardware que compone el sistema resulta ser el punto más complicado de estudio. Esto es debido a la latencia variable que las estructuras microarquitectónicas introducen con el fin de conseguir un mayor rendimiento. Especialmente crítico es el caso de la jerarquía de memoria, donde la presencia de memorias cache permite aprovechar las características de reúso inherentes del programa para reducir la latencia y consumo energético que suponen los accesos a memoria a la hora de servir datos e instrucciones al procesador.

Atendiendo a este problema, se han planteado diversos estudios abordando el cálculo del tiempo de ejecución en el peor caso en memorias cache de instrucciones. Sin embargo, el análisis de las memorias cache de datos es un problema más complejo de resolver [2], [3]. Esta complejidad se puede observar de manera clara en el flujo de un programa, por ejemplo, el recorrido de direcciones de memoria en bucles, el acceso a las variables locales almacenadas en la pila durante una llamada a una función o la resolución de direcciones de memoria en tiempo de ejecución. Por lo tanto, el análisis de aciertos y fallos en el acceso a datos almacenados en memoria cache para el peor caso de ejecución introduce una dificultad mayor respecto al mismo cálculo para el acceso a instrucciones. Además, teniendo en cuenta que en estudios previos se ha estimado que el acceso a datos tiene un peso de alrededor del 50 % en el cálculo de WCET [4], se puede deducir la importancia de plantear una solución a este problema.

Para realizar el análisis de WCET sobre sistemas que constan de memorias cache de datos se han planteado principalmente dos metodologías: (1) realizar un análisis centrado en la localidad de jerarquías específicas de memorias cache [2] y (2) orientar el análisis sobre las características de reúso de datos inherentes del programa [5].

[1]Dpto. de Ciencias de la Computación, Barcelona Supercomputing Center, e-mail: alvaro.moreno@bsc.es.

[2]Dpto. Informática e Ingeniería de Sistemas, I3A, Universidad de Zaragoza, e-mail: {jsegarra,rgran}@unizar.es.

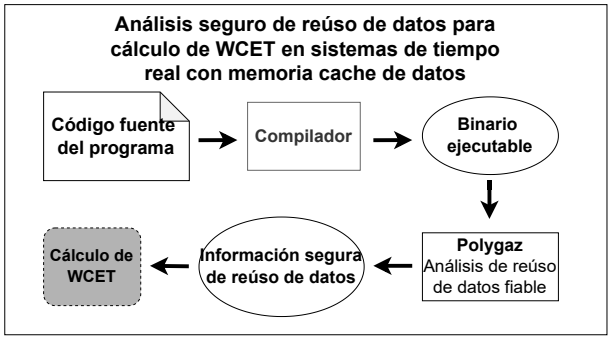


Fig. 1: Flujo del método de análisis: desde código fuente hasta integración en el cálculo de WCET.

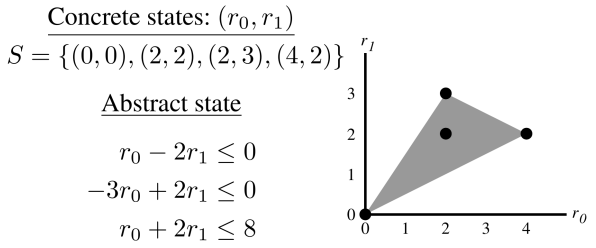


Fig. 2: Dominios concreto (puntos sólidos) y abstracto (zona sombreada)

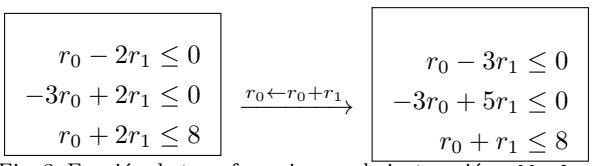


Fig. 3: Función de transferencia para la instrucción `add r0,r1`

El trabajo desarrollado en este artículo pretende continuar la línea de investigación orientada a capturar el reúso de datos presente en el programa. Esto permitirá demostrar que esta metodología permite caracterizar de forma segura el reúso de datos presente en una aplicación, con independencia del sistema sobre el que se esté trabajando, ya que previamente solo se había estudiado su efectividad en sistemas con arquitectura ARMv7. Con este objetivo, se ha extendido la funcionalidad de la herramienta presentada en [5], *Polygaz*, para permitir el análisis de binarios compilados sobre arquitecturas x64 (disponible en https://gitlab.com/m4515/x64-polygaz) y se ha aplicado sobre el banco de pruebas TACLeBench [6]. En este caso, adicionalmente, se ha buscado analizar los efectos que el uso de compiladores diferentes tendría sobre el reúso capturado, estableciendo para esta finalidad una comparación entre binarios compilados con GCC [7] y Clang [8] para los niveles de optimización O0 y O2 en cada uno de ellos.

Las secciones restantes de este artículo quedan organizadas de la siguiente manera. En la Sección II se relata brevemente los conceptos más importantes del trabajo previo en el que se ha basado este artículo. En la Sección III se detallan los cambios aplicados para habilitar el análisis sobre x64. La Sección IV muestra en detalle el entorno experimental y los resultados obtenidos. Por último, en la Sección V se despliegan las conclusiones extraídas.

II. Contexto

El método de análisis utilizado ha sido probado, en [5], como un método general para la extracción segura de los patrones de acceso a datos existentes en las instrucciones load/store de un programa, así como, el reúso existente entre estos accesos.

Para ello este tipo de análisis se sustenta sobre una serie de ideas claves que vale la pena desarrollar para una mejor comprensión de este artículo. En primer lugar, como se puede observar en la figura 1, el análisis se plantea sobre el programa en código binario. Esto permite que las transformaciones aplicadas por el compilador sobre los patrones de acceso a memoria de datos del programa sean visibles al análisis, permitiendo así el estudio sobre diferentes compiladores y niveles de optimización.

En segundo lugar, cabe destacar la utilización de Interpretación Abstracta como forma de garantizar que la captura de patrones de acceso a memoria, y por tanto el reúso entre ellos, es segura en todos los posibles casos. Esta idea se enfoca sobre el hecho de que una dirección de memoria utilizada en una instrucción load/store ha debido ser previamente almacenada en otro registro (o una posición de memoria). Para ello se define un dominio abstracto que conecta como estados definidos por inecuaciones las transformaciones lineales que sufren estos elementos concretos (registros y posiciones de memoria vistas como un almacenamiento único e ilimitado). Los diferentes estados abstractos dentro de este dominio son actualizados por unas funciones de transformación, denominadas funciones de transferencia, que son aplicadas en cada instrucción encontrada al recorrer iterativamente los diferentes flujos de ejecución del programa. Estos dos conceptos pueden verse ilustrados primero en la figura 2 [5], dónde se pueden ver los distintos estados arquitecturales que dos registros han podido tomar en un punto de la ejecución del programa (puntos solidos), y las inecuaciones obtenidas a partir de las transformaciones lineales aplicadas sobre los registros por las instrucciones que representan el dominio abstracto (zona sombreada). De aquí se puede deducir que cualquier patrón capturado en el dominio abstracto se cumple en el dominio concreto sustituyendo por los valores almacenados en cada uno de los registros. Además, en la figura 3 [5] se puede observar un ejemplo de cómo la función de transferencia asociada a la instrucción `add r0,r1` transformaría los estados del dominio abstracto. Por último, se caracteriza el reúso siguiendo la teoría general del reúso de datos en bucles [9] en presencia de memorias cache. De este modo, para instrucciones load/store escalares se hablará de *reúso temporal* cuando estas presenten una dirección que apunte a una línea de cache (o bloque) ya ac-

```
1  int A[100][100]; int b, c, d;
2  b=1;             //1: primer uso en acceso escalar
3  for (register int i=0; i<100; i++){
4   c=              //2: auto-reúso temporal escalar
5    A[i][5]+       //3: auto-reúso espacial largo
6    b;             //4: reúso temporal de grupo escalar, reúsa 1
7   A[5][i]=        //5: auto-reúso espacial corto
8    A[i][5]+       //6: reúso temporal de grupo vectorial, reúsa 3
9    c;             //7: reúso temporal de grupo escalar, reúsa 2
10 }
```

Fig. 4: Ejemplo de clasificación de reúso en código C.

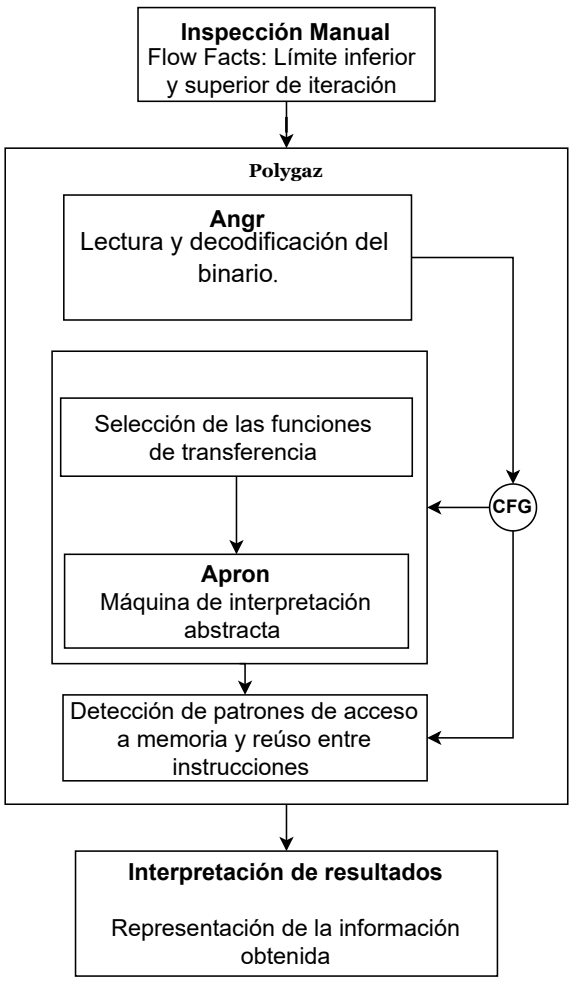


Fig. 5: Representación esquemática de Polygaz

cedida previamente. A su vez, se indentificará como *auto-reúso temporal* si la línea ya había sido accedida por esa misma instrucción, o como *reúso temporal de grupo* si el acceso previo fue provocado por una instrucción distinta. Por otro lado, en el caso de instrucciones load/store que constituyen el patrón de acceso a un vector, se hablará de *auto-reúso espacial corto* cuando el patrón de acceso al vector haga que cada acceso consecutivo a memoria pueda servirse desde la misma línea de cache. En cambio, si el recorrido del vector está construido de tal forma que esto no se pueda dar se considerará *auto-reúso espacial largo*. Finalmente, si dos instrucciones distintas presentan el mismo patrón de acceso a un vector se catalogará también como *reúso temporal de grupo*.

En la figura 4 aparece ilustrado un ejemplo sobre la clasificación de los diferentes tipos de reúso considerados en un fragmento de código de alto nivel.

III. Análisis de reúso de datos en arquitectura x64

Para conseguir llevar a cabo el análisis de reúso de datos sobre ejecutables de arquitectura x64 ha sido necesario ampliar la funcionalidad de la herramienta Polygaz. En la figura 5 se muestra un esquema modular que describe su flujo de trabajo.

Esta utilidad necesita, en primer lugar, recibir información previa al análisis sobre el dominio de iteración de los bucles que se han encontrado dentro del programa a analizar. Esto se consigue mediante una primera fase de exploración manual de cada uno de los binarios considerados en el experimento[1]. Esta información es empleada posteriormente para estimar el número de veces que es ejecutado cada uno de los accesos a memoria en el caso peor. Después de esta fase previa, la herramienta utiliza *angr* [10], [11], [12] para la lectura del binario y su decodificación en un lenguaje intermedio basado en micro-operaciones básicas. Posteriormente, todos los posibles caminos de ejecución presentes en el grafo de flujo de control del programa (CFG) son recorridos y cada una de estas micro-operaciones básicas son interpretadas definiendo una función de transferencia que actualiza en la máquina de interpretación abstracta, implementada con la biblioteca *apron* [13], el estado del programa. A partir de este estado abstracto se puede finalmente extraer la información correspondiente a los patrones de acceso a memoria y el reúso existente entre instrucciones aplicando las ideas integradas en la herramienta en el estudio original.

Las modificaciones realizadas para conseguir el análisis sobre x64 han sido las siguientes. Primero, se tuvieron que definir los registros de propósito general de la arquitectura x64 (aquellos en los que se espera que se computen las direcciones de memoria) para la inicialización de las variables del estado abstracto. En segundo lugar, al ampliar el campo de análisis de la herramienta desde una arquitectura de 32 bits a una arquitectura de 64 bits, surgen nuevos posibles accesos a memoria que presentan un tamaño mayor a los que ya existían implementados. Estos nuevos accesos han tenido que ser decodificados dentro de la herramienta, añadiendo de esta manera la posibilidad de interpretar accesos a variables de tipo entero de 64 bits, a variables de coma flotante de 64 bits y accesos vectoriales de 128 bits. Además de los accesos a memoria también se han tenido que interpretar las nuevas micro-operaciones obtenidas a partir de

[1]El compilador también podría realizar esta tarea de manera automatizada.

la decodificación de las nuevas instrucciones de la ISA x64 encontradas. En total se ha añadido la posibilidad de interpretar 66 nuevas micro-operaciones. Estas micro-operaciones comprenden tanto operaciones aritmético-lógicas enteras y de coma flotante de 64 bits como operaciones vectoriales, de conversión entre tipos de datos (más comunes en x64 por el uso de segmentos en los registros de propósito general) y operaciones geométricas (desplazamientos, multiplicaciones...) de 64 bits.

IV. Resultados experimentales

En esta sección se muestra la evaluación de los resultados derivados de aplicar el método de análisis de reúso de datos sobre arquitectura x64. En primer lugar, se introduce brevemente el entorno de trabajo sobre el que se ha puesto en práctica el experimento junto al banco de pruebas seleccionado. En segundo lugar, se valoran los resultados obtenidos con relación a la clasificación del reúso de datos capturado en cada ejecutable por el analizador. Por último, se realiza una estimación simplificada del tiempo ejecución en el peor caso que suponen los accesos a datos de memoria en cada caso de prueba aplicando la información obtenida en el análisis.

A. Entorno experimental y caracterización del banco de pruebas

Para la realización del análisis se ha utilizado como banco de pruebas un subconjunto de la colección TACLeBench [6]. En este trabajo no se han tenido en cuenta aquellos programas que cuentan con funcionalidad recursiva o paralela. De entre los 50 programas de prueba restantes, en este artículo se han escogido 38 de ellos, listados en la tabla I, ya que se ha considerado que este conjunto es lo suficientemente representativo para ilustrar el propósito del trabajo. Una vez seleccionado el banco de pruebas se han compilado todos los fuentes utilizando *gcc-9.4.0* y *clang-10.0.0* para x64 con dos niveles diferentes de optimización (-O0 y -O2).

En la tabla I por cada programa, compilador y nivel de optimización aparecen representados, en primer lugar, bajo la primera columna, “Instrucciones load/store dominante”, el número de instrucciones load/store que suponen el primer acceso a contenido de memoria que ha podido ser reusado posteriormente. En la columna se muestra tanto el número absoluto, como el porcentaje respecto del número total de instrucciones load/store encontradas para cada compilador (*gcc* y *clang*) y nivel de optimización (O0 y O2) probados. En el nivel de optimización O0 se encuentra, en general, un número mayor de load/store dominantes ya que estos binarios utilizan variables temporales almacenadas en la pila que suponen accesos innecesarios a memoria, los cuáles son reducidos por las técnicas de optimización en -O2. Esta afirmación, como se puede observar, se cumple en un mayor término en los binarios compilados con *gcc* respecto de aquellos para los que se ha utilizado *clang*. Mientras que en el nivel O0 (sin optimizaciones) los resultados entre ambos compiladores son prácticamente similares, en el nivel O2 se aprecian diferencias notables en programas como: *ndes*, *filterbank* o *minver*. Donde se ha comprobado que las optimizaciones aplicadas por *clang* O2 en términos de desenrollado de bucles y vectorización son más agresivas afectando de esta forma a los resultados, p.e. aparecen desenrollados completos de bucles transformando accesos a vector en un número elevado de accesos a variables escalares.

Bajo la columna “ Accesos estimados en el peor caso” se muestra una estimación del número de accesos dinámicos (teniendo en cuenta los realizados en cada iteración de un bucle) a datos en memoria para cada binario en el peor caso de ejecución posible (número máximo de iteraciones en todos los bucles). Este valor esta representado como un número absoluto para los binarios O0 y como el porcentaje de accesos encontrados respecto de O0 para los binarios O2. De este modo, se puede observar fácilmente como, en general, en los binarios optimizados el número de accesos a memoria se ve significativamente reducido. Además, cabe destacar que como se puede ver en la mayor parte de los binarios O0, *clang* emplea un mayor número de accesos a memoria en sus soluciones que *gcc*, ya que como se ha podido comprobar utiliza más la pila para la declaración de variables temporales incluyendo de esta forma más instrucciones load/store. Por ejemplo, en el programa *fft* dónde la diferencia es significativamente notable se encuentran un total de 104 instrucciones estáticas adicionales.

B. Clasificación de reúso

En las figuras 6 y 7 se muestra la cantidad de reuso clasificado por su tipología para cada binario analizado con *gcc* y *clang* respectivamente. Las gráficas se descomponen en base al tipo de dato que ha sido accedido en memoria (escalar, vector o no lineal) por las instrucciones load/store encontradas. Estos accesos se han denominado “escalar” cuando el patrón de acceso correspondiente a la instrucción determina que se está accediendo a un dato de tipo escalar. “vector” cuando se trata de una instrucción con un patrón de acceso que recorre una estructura de datos en memoria. Y “no lineal” cuando no se ha conseguido construir una expresión lineal para determinar la dirección de memoria accedida en el load/store durante el proceso de interpretación abstracta quedando oculta esta información al analizador. Para mayor realismo, en el análisis se ha considerado un tamaño fijo de 64 bytes para los datos accedidos, igual al tamaño de línea de una cache L1 de Intel. De esta forma, los accesos a variables escalares se asume que traen de memoria principal la línea de cache donde se encuentra el dato accedido, en lugar de únicamente los bytes especificados por la instrucción load/store. Para cada tipo de acceso, a su vez, se fragmenta el color de relleno de las barras teniendo en cuenta el tipo de reúso específico detectado. El color de relleno se ha escalado de tal modo que los tipos de reúso que se

Tabla I: **Caracterización del banco de pruebas**

	Instrucciones load/store dominantes				**Accesos estimados en el peor caso**			
	gcc		**clang**		**gcc**		**clang**	
Nombre	***O0***	***O2***	***O0***	***O2***	***O0***	***O2***	***O0***	***O2***
adpcm_enc	22 (3 %)	35 (9 %)	23 (3 %)	30 (4 %)	240 783	0 %	276 190	0 %
audiobeam	95 (5 %)	75 (10 %)	89 (4 %)	79 (10 %)	1 021 651	17 %	1 082 201	17 %
binarysearch	3 (4 %)	2 (13 %)	5 (6 %)	1 (8 %)	412	32 %	488	26 %
bsort	4 (5 %)	3 (27 %)	3 (4 %)	14 (25 %)	246 943	16 %	257 342	12 %
cjepg_transupp	21 (3 %)	13 (4 %)	22 (3 %)	36 (3 %)	36 746 346	13 %	40 809 350	14 %
cjpeg_wrbmp	80 (12 %)	26 (11 %)	80 (12 %)	13 (5 %)	167 738	24 %	176 247	23 %
complex_updates	11 (9 %)	8 (20 %)	14 (10 %)	22 (16 %)	1258	38 %	1700	26 %
cosf	25 (5 %)	22 (9 %)	16 (3 %)	33 (12 %)	52 620	35 %	54 624	48 %
countnegative	6 (8 %)	4 (21 %)	6 (6 %)	1 (5 %)	7658	26 %	15 811	21 %
deg2rad	4 (13 %)	2 (33 %)	4 (11 %)	2 (22 %)	3631	0 %	3635	0 %
dijkstra	18 (9 %)	8 (10 %)	18 (8 %)	9 (9 %)	3 011 142 206	20 %	3 014 846 507	40 %
duff	7 (7 %)	2 (8 %)	8 (7 %)	6 (14 %)	4337	9 %	4841	5 %
fft	12 (6 %)	13 (15 %)	17 (6 %)	17 (20 %)	36 962 344	57 %	142 178 459	15 %
filterbank	5 (4 %)	11 (11 %)	7 (5 %)	201 (2 %)	4 176 141	26 %	4 186 942	29 %
fir2dim	14 (7 %)	4 (7 %)	12 (4 %)	8 (4 %)	4442	22 %	6267	51 %
fmref	462 (15 %)	86 (7 %)	467 (15 %)	77 (8 %)	2 761 201	14 %	2 869 150	12 %
g723_enc	40 (3 %)	30 (4 %)	50 (3 %)	46 (6 %)	1 499 506	30 %	1 639 291	14 %
h264_dec	17 (2 %)	7 (6 %)	41 (3 %)	10 (6 %)	230 498	17 %	311 527	13 %
huff_dec	19 (3 %)	19 (8 %)	34 (5 %)	6 (4 %)	410 566	36 %	465 399	19 %
iir	5 (6 %)	4 (11 %)	6 (5 %)	10 (6 %)	1167	27 %	1454	27 %
insertsort	5 (5 %)	4 (9 %)	7 (8 %)	3 (8 %)	1717	22 %	1808	20 %
isqrt	7 (6 %)	5 (14 %)	7 (5 %)	3 (4 %)	578 594	5 %	686 706	4 %
jfdctint	18 (6 %)	14 (31 %)	20 (6 %)	14 (6 %)	2988	13 %	3245	11 %
lift	44 (8 %)	17 (8 %)	32 (6 %)	27 (7 %)	1 775 960	13 %	2 102 423	13 %
ludcmp	6 (3 %)	9 (12 %)	8 (3 %)	16 (14 %)	5143	20 %	5408	31 %
matrix1	4 (6 %)	1 (7 %)	4 (5 %)	9 (8 %)	6339	43 %	15 784	33 %
minver	8 (3 %)	12 (14 %)	11 (3 %)	48 (18 %)	2568	16 %	2672	48 %
ndes	35 (5 %)	25 (11 %)	34 (5 %)	82 (10 %)	60 107	32 %	67 344	25 %
petrinet	39 (5 %)	10 (2 %)	40 (5 %)	10 (2 %)	1537	59 %	1554	53 %
pm	51 (3 %)	39 (7 %)	70 (4 %)	67 (6 %)	10 573 751	33 %	11 360 834	21 %
prime	8 (6 %)	2 (8 %)	8 (5 %)	1 (8 %)	550	4 %	598	2 %
rad2deg	4 (13 %)	2 (33 %)	4 (11 %)	2 (22 %)	3621	0 %	3625	0 %
rijndael_dec	147 (7 %)	105 (12 %)	156 (8 %)	182 (16 %)	2 969 828	41 %	2 996 913	52 %
rijndael_enc	155 (8 %)	104 (11 %)	154 (8 %)	135 (16 %)	2 385 671	46 %	2 399 773	43 %
sha	41 (3 %)	20 (7 %)	46 (3 %)	28 (4 %)	2 253 965	20 %	2 904 561	14 %
st	26 (6 %)	10 (14 %)	24 (5 %)	8 (10 %)	106 651	11 %	127 557	27 %
statemate	16 (2 %)	15 (2 %)	20 (2 %)	9 (1 %)	79 868	72 %	76 425	91 %

prevee que presenten un acierto en el acceso a memoria tengan colores más oscuros, mientras que aquellos con colores más claros normalmente supondrían un fallo. Por último, el caso promedio obtenido para el banco de pruebas se posiciona en el margen izquierdo.

Ambas figuras 6 y 7 muestran como las transformaciones aplicadas por los dos compiladores transforman de manera importante la distribución del tipo de reúso obtenido. Las versiones sin optimizaciones (O0) presentan, generalmente, un mayor número de variables temporales innecesarias y accesos a memoria replicados. Estos accesos son eliminados por las técnicas de optimización. Por ejemplo, manteniendo las variables en registros en lugar de acceder constantemente a la pila. Se puede observar, entonces, como en gran parte de los programas los accesos escalares se ven reducidos en el nivel O2. También, en el caso de programas como *lift*, *jfdctint*, *complex_updates* o *matrix1* se puede apreciar el efecto de la diferencia en la agresividad de las optimizaciones entre compiladores comentada previamente. Especialmente, en el caso de *lift* se ve como en los resultados de *gcc* el número de accesos escalares se ve decrementado al compilar con O2 y se detecta en su lugar más reúso espacial. En el caso de *clang* en cambio el desenrollado agresivo que se plantea sobre los bucles del binario O2 hace que no solo el número de accesos escalares se vea reducido en menor medida, sino que aumente respecto de O0 y además no se detecte ningún tipo de reúso espacial.

También se puede observar como el nivel de optimización O0 oculta en la mayoría de casos el reúso espacial, el cual es clasificado como no lineal. Esto es

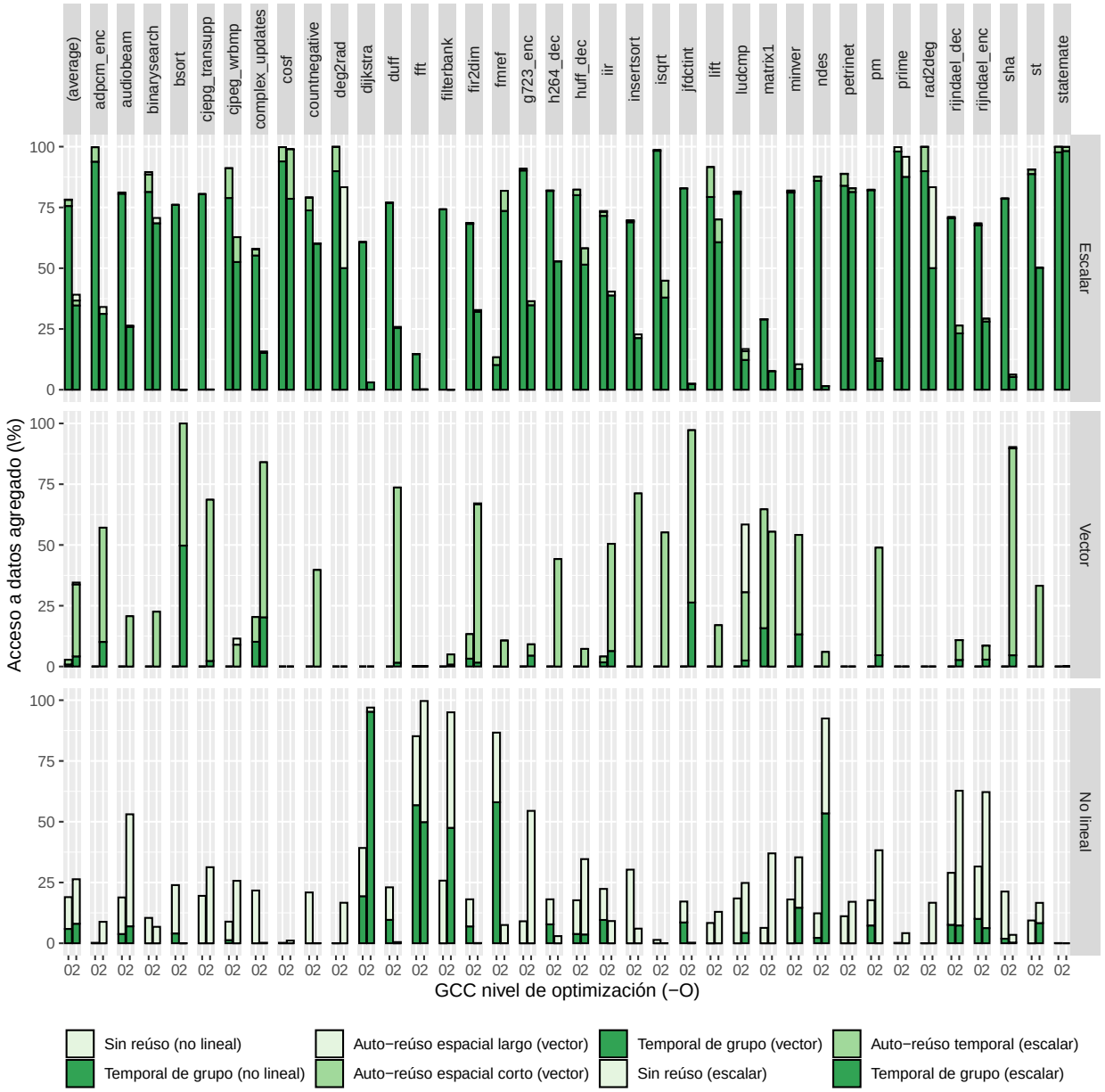


Fig. 6: **Porcentaje de de tipos de reúso encontrados en cada binario para el compilador gcc.**

debido a que en las versiones sin optimizaciones normalmente la variable que se corresponde con el iterador de un vector se almacena y recupera constantemente de memoria. Provocando que si se produce una instrucción store en medio de esta secuencia, todas las posiciones de memoria pasen a ser consideradas por el analizador en un estado de incertidumbre que evita detectar correctamente el patrón de acceso al recuperar el valor del iterador. Este comportamiento es especialmente notable en *clang* dónde no se ha detectado reúso espacial en ningún binario O0, mientras que en *gcc*, aunque de forma igualmente escasa, sí que se ha conseguido la detección en algunos programas como *complex_updates*, *fir2dim* o *matrix1*.

En el nivel de optimización O2 en ambos casos se observa también una tendencia de incremento en el reúso no lineal capturado. Las optimizaciones aplicadas por los compiladores sobre los binarios con arquitectura x64 se ven implementadas con nuevas instrucciones (p.e uso de instrucciones vectoriales al vectorizar bucles) que, a su vez, son decodificadas en nuevas micro-operaciones en lenguaje intermedio. La mayoría de nuevas micro-operaciones añadidas (un 80 %) en la herramienta no son representables con una expresión lineal. Por lo tanto, el uso de instrucciones que impliquen usar estas nuevas micro-operaciones para construir los patrones de acceso a memoria puede ocultar información al analizador. Un ejemplo claro se puede ver en el programa *dijkstra*, en el que gran parte del reúso escalar encontrado en las versiones O0, además de ser eliminado por las optimizaciones también queda oculto a la herramienta siendo capturado como no lineal.

C. Estimación de WCET

Por último, se evalúa mediante una estimación simple como podría afectar la información de reúso capturada anteriormente al cálculo del tiempo de eje-

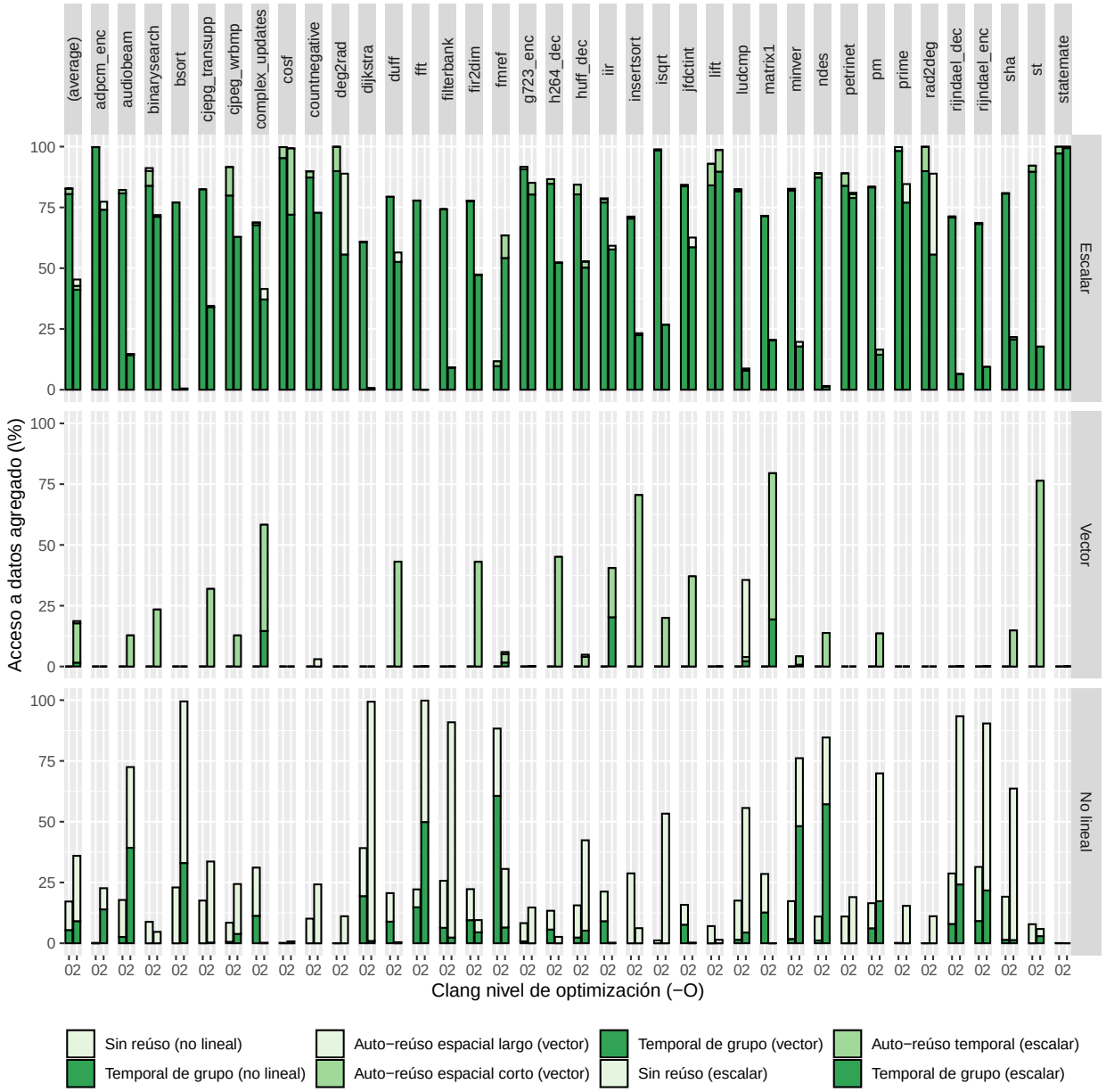


Fig. 7: **Porcentaje de de tipos de reúso encontrados en cada binario para el compilador clang.**

cución en el peor caso. Para ello, primero se valora el número de aciertos y fallos que cada tipo de reúso de datos catalogado en la clasificación debería provocar en base a los siguientes principios:

- Aquellas instrucciones de memoria clasificadas como "sin reúso" (escalar y no lineal) deben ser consideradas como fallos en el peor caso.
- Las que presentan un reúso "temporal de grupo"(escalar, vector y no lineal) deberían resultar siempre en un acierto. Para simplificar la estimación propuesta no se tiene en cuenta que el bloque accedido podría llegar a ser expulsado de la memoria cache entre la ejecución de las instrucciones que lo reusan.
- Los accesos escalares con "auto-reúso temporal" deberían suponer un fallo en el primer acceso al dato dentro del bucle, y un acierto en las sucesivas ejecuciones de la instrucción (asumiendo que el bloque no es expulsado de memoria entre accesos).
- De los accesos a vector clasificados como "auto-reúso espacial corto" (p.e. un recorrido secuencial de un vector accediendo los elementos sucesivos de cada linea de cache) cabe esperar una alta tasa de acierto, la cual dependerá del tamaño de los elementos que conformen el vector accedido, el tamaño de la linea de cache y el patrón de acceso aplicado.
- Los accesos a vector clasificados como "auto-reúso espacial largo" (p.e. el recorrido de las columnas de una matriz de mayor tamaño que la memoria cache y habiendo sido esta almacenada por filas) deben ser considerados, generalmente, como fallos. Excepto en el caso de que la estructura de datos pueda ser almacenada por completo en la memoria cache, dónde se obtendrían resultados equivalentes a las instrucciones que presentan "auto-reúso espacial corto"

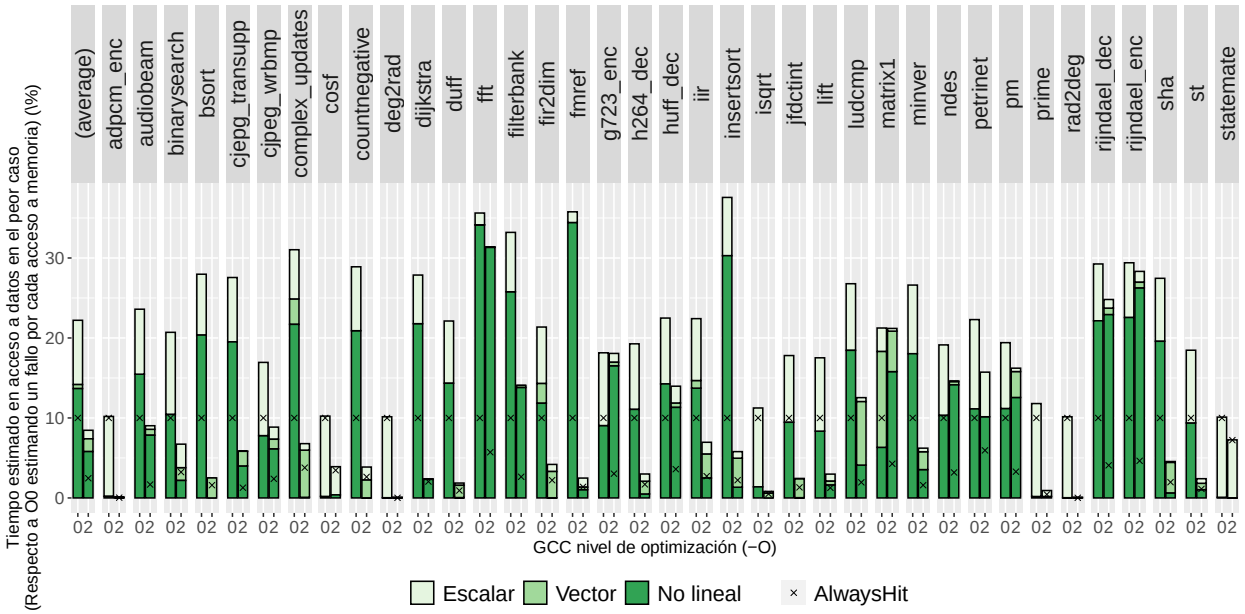


Fig. 8: **Estimación aproximada para el compilador gcc del tiempo en el peor caso de ejecución destinado a instrucciones de acceso a memoria respecto a O0 estimando un fallo por cada acceso a memoria.**

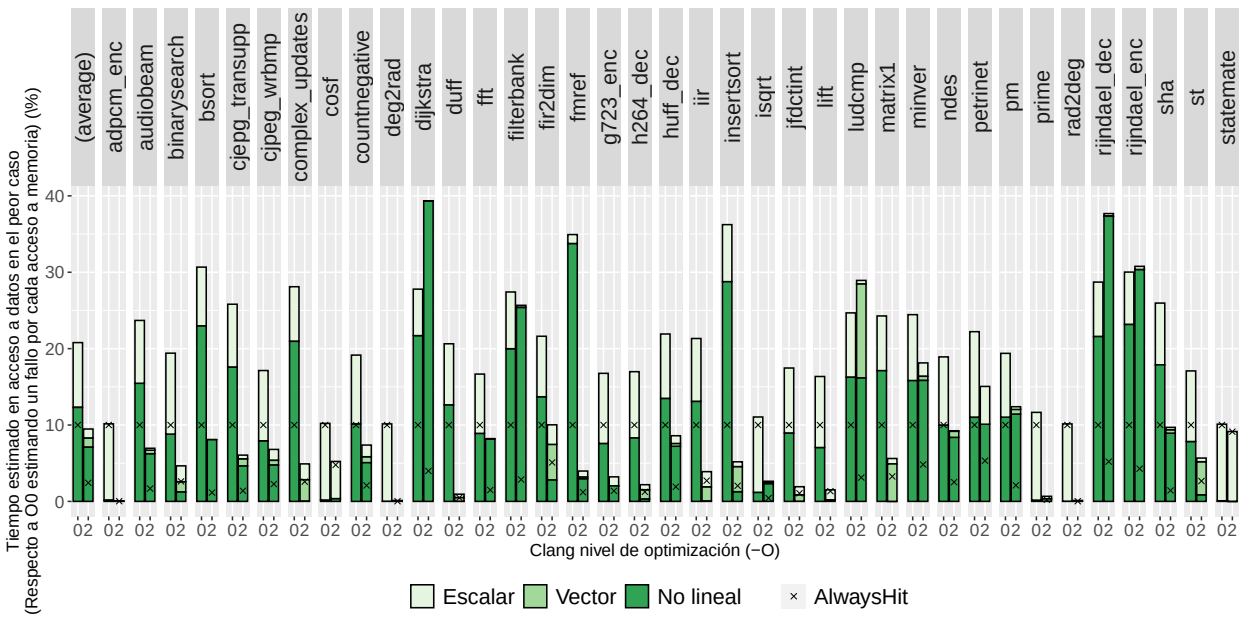


Fig. 9: **Estimación aproximada para el compilador clang del tiempo en el peor caso de ejecución destinado a instrucciones de acceso a memoria respecto a O0 estimando un fallo por cada acceso a memoria.**

De esta forma, para mostrar los resultados se estima que aquellos accesos clasificados como "sin reúso" y "auto-reúso espacial largo" resultarán siempre en un fallo, mientras que las instrucciones clasificadas con reúso "temporal de grupo" se resolverán con un acierto. En el caso de el "auto-reúso temporal (escalar)" se ha considerado el primer acceso como un fallo y aquellos que le suceden en el resto de iteraciones del bucle como acierto. Por último, para el "auto-reúso espacial corto", se asume una línea de cache capaz de almacenar 8 elementos, contando, por tanto, un fallo cada 8 accesos. Con estas premisas se pretende acercar la estimación de forma razonable a la ejecución de una aplicación sobre un procesador con una única memoria cache L1 de datos pequeña, capaz de lanzar a ejecución una instrucción por ciclo y con latencia de ejecución de instrucciones de un ciclo.

En las figuras 8 (*gcc*) y 9 (*clang*) se muestra, en base a la estimación descrita previamente, el tiempo de ejecución en el peor caso, desglosado en base al tipo de acceso a dato. Estos valores se interpretan con respecto al peor caso posible del binario O0 donde todos los accesos se interpretarían como fallos. Se ha dado un coste de un solo ciclo a los accesos a datos que suponen un acierto y un coste de 10 ciclos a los que suponen un fallo. De esta forma, las barras etiquetadas con O0 muestran la reducción en el tiempo de ejecución obtenida simplemente aplicando la estimación planteada sobre la clasificación de reúso obtenida. Las barras etiquetadas con O2 muestran además los efectos que las optimizaciones de ambos compiladores han tenido sobre la reducción en el tiempo de

ejecución. También, sobre cada barra se marca con una × el límite (inzalcanzable) que marcaría el caso ideal dónde todas las instrucciones se resuelven con un acierto al realizar el acceso al dato.

Como se puede observar en ambas figuras, independientemente del compilador utilizado, la reducción en el tiempo en el peor caso de ejecución es significativa comparado con aplicar la opción pesimista de asumir que todos los accesos a datos son fallos en un binario O0. En promedio, tanto en los binarios O0 como los binarios O2, ambos compiladores muestran un resultado muy similar (diferencia del 1-2 %) en el tiempo de ejecución en el peor caso con respecto al caso de asumir siempre fallo en todos los accesos en el binario O0. En O0 estos tiempos se sitúan alrededor del 20 % y en O2 alrededor del 10 % respecto del caso pesimista donde se asume que todos los accesos en el binario O0 son un fallo. De esta manera se puede ver como, en general, las optimizaciones deberían reducir el tiempo requerido para los accesos a datos al eliminar accesos a datos innecesarios. No obstante, se puede observar como en los binarios O2 de *clang* *dijkstra*,*rijndael_dec*,*rijndael_enc* y *ludcmp* esta reducción no se ve reflejada por la dificultad que ha tenido el analizador para resolver los patrones de acceso en sus instrucciones incrementando significativamente el reúso no lineal "sin reúso" y el número de fallos estimado respecto a O0 por tanto.

V. Conclusiones

En este artículo se ha presentado la ampliación de la herramienta polygaz para permitir capturar de forma segura el reúso presente en programas compilados sobre arquitecturas x64. Además, se ha puesto a prueba la herramienta implementada y se ha evaluado el comportamiento para binarios obtenidos por dos compiladores (*gcc* y *clang*) y niveles de optimización (O0, O2). El análisis se ha aplicado sobre un subconjunto de los programas del banco de pruebas TACLeBench (38) el cual está adaptado para la experimentación con aplicaciones de tiempo real. Como resultado del experimento se ha obtenido la clasificación del reúso de datos presente en los 152 binarios analizados (38 por cada compilador y nivel de optimización). Se ha observado que existen diferencias en el reuso expuesto entre ambos compiladores debido a sus diferentes estrategias de generación de código y optimización. Por ejemplo, en el nivel de optimización O0 se ha encontrado que el compilador *gcc* genera soluciones donde es más previsible conseguir capturar reúso espacial. Mientras que en *clang* ha quedado oculto en todos los programas analizados, en *gcc* se ha podido capturar en 5 de ellos. Estas diferencias también existen en el nivel de optimización O2, donde *clang* muestra una técnica de optimización más agresiva a la hora de aplicar desenrollado y vectorización para eliminar y transformar bucles. Encontrándose por tanto una disminución del reúso escalar y aumento del reúso espacial más ligera que en *gcc* en programas como *lift* o *complex_updates*. A su vez, se ha calculado que en promedio en el nivel O2 se detecta más reúso de datos escalar en *clang* (45.35 %) que en *gcc* (39.1 %), una mayor cantidad de reúso no lineal en *clang* (35.95 %) que en *gcc* (26.33 %) y finalmente un mayor reúso en accesos a vector esta en *gcc* (34.54 %) que en *clang* (16.67 %). Además, se han utilizado estos resultados para estimar el tiempo ejecución en el peor para un procesador sencillo con un solo nivel de cache de datos. Consiguiéndose como resultado que en la arquitectura x64 el tiempo dedicado a accesos a datos en el peor caso de ejecución supone alrededor de un 20 %, en comparación al tiempo de acceso a datos obtenido en un sistema que solo estima fallos en las instrucciones de acceso a memoria. Al aplicar optimizaciones este porcentaje de reducción se ve disminuido hasta alrededor de un 10 %.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación a través de los contratos PID2019-105660RB-C21 y PID2019-107255GB-C21 (MCIN/AEI/10.13039/501100011033), además de por el Gobierno de Aragón (grupo T5820R).

Referencias

[1] Luis C Aparicio, Juan Segarra, Clemente Rodríguez, and Víctor Viñals, "Improving the wcet computation in the presence of a lockable instruction cache in multitasking real-time systems," *Journal of Systems Architecture*, vol. 57, no. 7, pp. 695–706, 2011.

[2] Mingsong Lv, Nan Guan, Jan Reineke, Reinhard Wilhelm, and Wang Yi, "A survey on static cache analysis for real-time systems," *Leibniz Transactions on Embedded Systems*, vol. 3, no. 1, pp. 05–1, 2016.

[3] Jan Reineke, Daniel Grund, Christoph Berg, and Reinhard Wilhelm, "Timing predictability of cache replacement policies," *Real-Time Systems*, vol. 37, pp. 99–122, 2007.

[4] Juan Segarra, Clemente Rodriguez, Ruben Gran, Luis C Aparicio, and Victor Vinals, "Acdc: small, predictable and high-performance data cache," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 2, pp. 1–26, 2015.

[5] Juan Segarra, Jordi Cortadella, Rubén Gran Tejero, and Victor Vinals-Yufera, "Automatic safe data reuse detection for the wcet analysis of systems with data caches," *IEEE access*, vol. 8, pp. 192379–192392, 2020.

[6] Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener, "Taclebench: A benchmark collection to support worst-case execution time research," in *16th International Workshop on Worst-Case Execution Time Analysis*, 2016.

[7] The GNU Foundation, "Gcc, the gnu compiler collection," `https://gcc.gnu.org/`, 1986, Access Date: 10-2021.

[8] The LLVM Foundation, "Clang: a c language family frontend for llvm," `https://clang.llvm.org/`, 2003, Access Date: 10-2021.

[9] Michael E Wolf and Monica S Lam, "A data locality optimizing algorithm," in *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, 1991, pp. 30–44.

[10] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna, "Sok: (state of) the art of war: Offensive techniques in binary analysis," 2016.

[11] Nick Stephens, John Grosen, Christopher Salls, Audrey Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna, "Driller: Augmenting fuzzing through selective symbolic execution," 2016.

[12] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna, "Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware," 2015.

[13] Bertrand Jeannet and Antoine Miné, "Apron: A library of numerical abstract domains for static analysis," in *International Conference on Computer Aided Verification*. Springer, 2009, pp. 661–667.

Soporte Eficiente de Memoria Compartida Distribuida en Sistemas Multi-FPGA para Procesos de Inferencia de Redes Neuronales

David Rodríguez Agut[1], Rafael Tornero[1] y Jose Flich[1]

Resumen— **Hoy en día, las redes neuronales convolucionales (CNN) se han convertido en una herramienta común en una amplia gama de aplicaciones. Su gran precisión y eficiencia contrastan con sus requisitos computacionales, lo que lleva a la búsqueda de plataformas de hardware más eficientes para poder procesarlas. Las FPGA son adecuadas por su flexibilidad, eficiencia energética y baja latencia. Sin embargo, la creciente complejidad de las CNN requiere de dispositivos con mayor capacidad de cómputo, lo que justifica el uso de plataformas multi-FPGA. En este artículo, presentamos una plataforma multi-FPGA con soporte para memoria compartida distribuida para la inferencia de CNNs. Nuestra solución, a diferencia de trabajos anteriores, permite combinar diferentes estrategias de paralelismo de modelos aplicadas a CNNs, gracias al soporte de memoria compartida distribuida. Para una configuración de cuatro FPGA, la plataforma reduce el tiempo de ejecución de convoluciones 2D en un factor de 3,95 en comparación con una sola FPGA. La inferencia de modelos CNN se mejora en factores que oscilan entre 3,63 y 3,87.**

Palabras clave— **Multi-FPGA, Redes Neuronales, Aprendizaje Profundo.**

I. Introducción

En los últimos años, las FPGA han ido ganando popularidad no sólo como plataformas de prototipado, sino también como aceleradores hardware. Uno de los campos en los que se ha explorado el uso de las FPGA es el de las redes neuronales profundas (DNN). Las DNN se utilizan en una amplia gama de aplicaciones, como: conducción autónoma, clasificación de imágenes, procesamiento del lenguaje natural, visión por ordenador, etc.

Entre las DNN, las redes neuronales convolucionales (CNN) destacan por su gran precisión en muchas tareas aunque a cambio de un elevado coste computacional. Las CNN se componen principalmente de capas convolucionales, cada una de las cuales es un operador de convolución de alta dimensión. Las FPGAs destacan por su baja latencia y reducido consumo de energía en comparación con las CPUs/GPUs y flexibilidad en comparación con los ASICs [1–3]. Aunque se ha explorado el uso de FPGAs en el entrenamiento de redes neuronales [4], su uso se ha centrado principalmente en la inferencia con aritmética de punto fijo o basada en enteros, donde pueden conseguir menor latencia y menor consumo energético.

Originalmente, la mayoría de aceleradores basados en FPGA estaban centrados/restringidos a diseños de una sola FPGA [5–7]. Sin embargo, con el aumento del tamaño de las CNN y la perspectiva de que los modelos sean cada vez más grandes, una única FPGA se queda corta con los crecientes requisitos en términos de recursos. Para superar la limitación de recursos, y al mismo tiempo obtener un mejor rendimiento, en los últimos años se han explorado las plataformas multi-FPGA como solución. Pero con el uso de plataformas multi-FPGA surgen nuevos retos, como la forma de utilizar eficientemente todos los recursos disponibles, o cómo aumentar el rendimiento global sin afectar a la latencia o al consumo energético.

Para la ejecución de redes neuronales en FPGA se pueden seguir dos enfoques. En el primero, el modelo completo puede sintetizarse y optimizarse. Así se obtiene el mejor rendimiento posible, ya que los pesos de los filtros de convolución pueden codificarse en el chip utilizando BRAM. Sin embargo, la ejecución de distintos modelos requiere la reprogramación del dispositivo que suele durar cientos de milisegundos. Además, los modelos de gran tamaño pueden no caber en el chip. En segundo lugar, el diseño de un acelerador genérico (Kernel) puede implementarse y programarse desde el host, e iterar sobre él para inferir un modelo completo. El acelerador puede adaptarse a los recursos de la FPGA y no es necesario reprogramar el dispositivo para ejecutar diferentes modelos.

Cuando se utilizan múltiples FPGAs, la asignación de memoria externa es crítica. Cada acelerador tiene acceso a su propia memoria local, pero para intercambiar datos son necesarias transferencias de memoria explícitas. Para evitar estas transferencias, proponemos una plataforma multi-FPGA para la ejecución efectiva de procesos de inferencia de modelos CNN. El enfoque se basa en el paradigma de programación de memoria compartida distribuida. Nuestra plataforma implementa un espacio de direcciones de memoria compartida distribuido entre todos los dispositivos FPGA, siguiendo una arquitectura NUMA. De este modo, se evitan las transferencias explícitas de memoria entre FPGAs. En cada FPGA implementamos un acelerador genérico que puede programarse para trabajar con diferentes esquemas de particionado para un conjunto de capas específicas de la red neuronal. Por lo tanto, se pueden programar varios aceleradores para ejecutar el mismo conjunto de capas de forma concurrente y cooperativa. Gracias al uso del espacio de direcciones de memoria compartida, también es posible utilizar simultáneamente distintos esquemas de particionado. Esto es clave pa-

[1]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: {drodagu,ratorga,jflich}@disca.upv.es.

ra permitir aceleraciones de rendimiento cercanos al ideal.

II. Trabajos relacionados

En los últimos años varios trabajos han explorado el concepto de conectar varias FPGAs con una determinada topología para procesar DNNs. En [8] los autores proponen un clúster de FPGAs en *pipeline* para aumentar el rendimiento de las aplicaciones CNN mediante el uso de un algoritmo de programación dinámica para mapear las diferentes capas del modelo en las FPGAs. En su solución utilizan siete FPGAs, una utilizada como controlador, formando una topología en anillo. Cada FPGA tiene un acelerador personalizado para un conjunto específico de capas. Como resultado, el diseño carece de flexibilidad, ya que las FPGA están muy acopladas entre sí, por lo que para ejecutar diferentes modelos es necesario reconfigurar las FPGA cada vez. Ampliando su trabajo original en [9] proponen un algoritmo dinámico para mapear DNNs en plataformas multi-FPGA asimétricas.

En ambos trabajos, su objetivo es mapear un modelo DNN en una plataforma multi-FPGA capa a capa para aumentar el rendimiento. Siguiendo el mismo enfoque en [10–14] los autores también proponen diferentes técnicas y algoritmos para mapear eficientemente una DNN en una plataforma multi-FPGA.

Otros trabajos siguen un enfoque diferente. En [15] se propone un marco para particionar CNNs en clusters FPGA. La atención se centra en reducir la latencia mediante el empleo de diferentes esquemas de partición de capas, así como en aliviar el uso de memoria fuera del chip haciendo uso de los enlaces inter-FPGA. Sin embargo, debido a la implementación de su solución, el empleo de diferentes esquemas de particionado entre capas conlleva transferencias de datos que, en consecuencia, va en detrimento de su solución. Por lo tanto, se ven obligados a utilizar la misma estrategia de partición entre todas las FPGAs para no afectar a la latencia como resultado del intercambio explícito de datos entre FPGAs. Otros trabajos tienen como objetivo reducir el tiempo de inferencia de RNNs. En [16,17], el proyecto Microsoft Brainwave pretende reducir la latencia mediante la asignación de pesos en las FPGA. En [18] los autores también se centran en reducir la latencia en RNNs particionando las capas y segmentado el cálculo. En nuestro trabajo, permitimos la combinación de diferentes esquemas de particionado.

III. Conceptos Previos

La convolución se define como:

$$Input(I \times HI \times WI) \circledast Filter(O \times I \times KH \times KW) = Output(O \times HO \times WO)$$

La entrada se define como un tensor tridimensional $I \times HI \times WI$ donde I representa el número de canales de entrada (*feature maps*) y HI y WI representan el alto y la ancho de cada canal, respectivamente. Del mismo modo, la operación de convolución produce un tensor tridimensional de salida $O \times HO \times WO$. Normalmente, cada par de entrada-salida tiene un filtro $KH \times KW$ asociado. Por lo tanto, se utilizan $O \times I$ filtros en una convolución. Los filtros se aplican en cada subconjunto $KH \times KW$ de entradas y se desplazan vertical y horizontalmente sobre los canales de entrada.

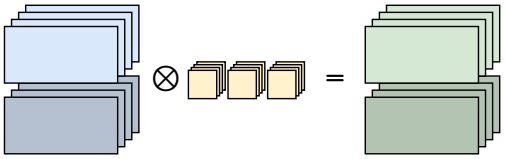
Una convolución puede dividirse en partes que se ejecutan en paralelo, dando lugar a diferentes esquemas de distribución de las activaciones de entrada, pesos y *feature maps*. Principalmente, podemos aplicar esquemas de partición por *batch*, filas, columnas, canales de salida y canales de entrada. La partición por *batches* se utiliza principalmente en procesos de entrenamiento. En la inferencia en tiempo real, el tamaño de *batch* se establece a uno.

En el particionado por filas (de entrada) (IRP, Fig. 1a), todos los canales de entrada se dividen en varias particiones, cada una con un conjunto disjunto de filas. Cada partición realiza la convolución utilizando los mismos filtros, por lo que los filtros se comparten entre particiones. Cada convolución en cada partición produce un subconjunto de filas de salida para todos los canales de salida. Dependiendo del *stride* horizontal de la convolución, algunas filas deben compartirse entre particiones adyacentes, por lo que no se trata de un esquema de particionado perfecto. El particionado IRP requiere que los datos de entrada, aunque particionados, sean accedidos por diferentes aceleradores. En un sistema de memoria no compartida, esto implicará copias de datos explícitas. En el particionado por columnas, los canales se dividen en conjuntos de columnas en lugar de filas. Este esquema tiene las mismas implicaciones que el particionado por filas.

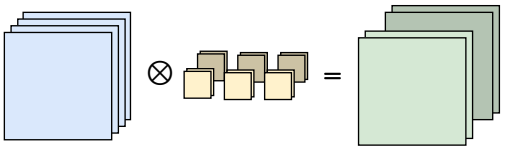
Con el particionado por canal de salida (OCP, Fig. 1b), los *feature maps* de salida se particionan/calculan en paralelo. Los filtros se particionan porque pertenecen a canales de salida diferentes. Sin embargo, los datos de entrada se comparten y, por tanto, cada acelerador necesita leer los mismos datos. Por último, en el particionado por canal de entrada, la entrada se divide en conjuntos de canales y para cada partición se calcula el conjunto completo de canales de salida. De forma similar a OCP, los filtros se particionan por canal de entrada. Sin embargo, la salida producida por cada partición debe ser compuesta por una capa adicional para agregar todas las particiones.

IV. Diseño de la plataforma

Nuestro diseño base (Fig. 2) cuenta con cuatro FPGAs Xilinx Kintex Ultrascale interconectadas siguiendo una topología de anillo bidireccional. Cada FPGA tiene dos módulos QSFP+, y cuatro cables ópticos para la comunicación con sus dos FPGA vecinas. A nivel de diseño, se han utilizado los IP cores Chip2Chip [19] y Aurora [20] de Xilinx para la comunicación entre FPGAs. Estos IP permiten la comunicación *board-to-board* siguiendo el protocolo AXI4. Además, cada FPGA incluye un módulo de memoria DDR4 de 2 GB y su correspondiente controlador de



(a) Particionado por filas



(b) Particionado por canal de salida
Fig. 1: Esquemas de partición de capas.

memoria. Este controlador gestiona los accesos locales a la memoria y no ha sido modificado en nuestro diseño. Además, una FPGA está conectada al *host* por PCIe a través del IP core de PCIe. Cada FPGA incluye también un acelerador. Todos los componentes del diseño están interconectados y configurados a través de una infraestructura de interconexión AXI. Como resultado, para la aplicación que se ejecuta en el *host* todo el diseño aparece como una única plataforma con cuatro aceleradores y 8 GB de memoria.

A. Soporte para memoria distribuida y compartida

Para el soporte de memoria compartida distribuida, cada FPGA incluye una jerarquía de interconexiones AXI (AXI SmartConnect IP). En $FPGA_1$, $FPGA_2$ y $FPGA_3$ se utilizan cuatro módulos; mientras que en $FPGA_0$ se utilizan seis. Hasta tres módulos enrutan el tráfico entrante desde una FPGA vecina, que puede acceder a la memoria local o reenviar el tráfico a otra FPGA (utilizando esta FPGA como en tránsito/intermediaria), desde el acelerador local o desde el módulo PCIe en el caso de la $FPGA_0$. Estos módulos AXI tienen diferentes salidas dependiendo de los patrones de comunicación permitidos. Cada salida puede entregar los datos directamente al destino final: memoria local, o llegar a un nuevo módulo AXI que entrega los datos al dispositivo FPGA vecino correspondiente. La interconexión entre módulos AXI no es una topología totalmente conectada, ya que algunos enlaces no están implementados.

Todos los módulos se han configurado para soportar memoria compartida distribuida. Cada acelerador de cualquier FPGA puede acceder a todas las memorias utilizando el mismo espacio de direccionamiento/direcciones de memoria. La tabla I muestra la configuración de las interfaces AXI en cada FPGA con el rango de direcciones y las rutas utilizadas para acceder a cada memoria por todos los dispositivos. Físicamente, las FPGAs están conectadas siguiendo un anillo bidireccional. Sin embargo, internamente, las FPGAs están agrupadas lógicamente por filas. Cada FPGA puede acceder a las DDR de sus vecinas directas a través de los enlaces inter-FPGA. Pero para acceder a una DDR situada en una FPGA no vecina, tiene que pasar por su vecina en la otra fila. Por ejemplo, $FPGA_0$ tiene que pasar por $FPGA_2$ para acceder a DDR_3. Esto significa también que $FPGA_0$ y $FPGA_2$ tienen que compartir el enlace entre $FPGA_2$ y $FPGA_3$ si ambos intentan acceder a DDR_3 al mismo tiempo.

Los enlaces entre FPGAs están configurados para funcionar con un ancho de banda efectivo de lectura y escritura de 2,2 GB/s en modo dúplex. El ancho de banda está limitado por los módulos QSFP+, ya que admiten una velocidad de transferencia máxima de 10 Gb/s por carril/*lane*, mientras que los transceptores admiten transferencias de datos de hasta 16,375 Gb/s, y por el módulo de Chip2Chip que, en tándem con Aurora, sólo puede configurarse para utilizar tres carriles/*lanes* como máximo. Por tanto, cada enlace está configurado para utilizar tres carriles/*lanes* a 10 Gb/s cada *lane*.

B. Software de Soporte

La plataforma está expuesta al *host* como un dispositivo PCIe mapeado en memoria con dos regiones de memoria: una para controlar los aceleradores en cada FPGA, y otra para mapear los diferentes bancos de memoria del dispositivo (DDRs) a través de un espacio de direcciones unificado. Para proporcionar acceso a esta plataforma hemos desarrollado una biblioteca similar a OpenCL, inspirada en el software OpenCL de Xilinx y compuesta por tres capas: un API, un *runtime* y una capa de abstracción de hardware (HAL).

La Figura 3 ilustra la jerarquía de capas de la biblioteca. De arriba a abajo, el API presenta dos abstracciones principales a la aplicación en el *host*, denominadas *device* y *buffer*. Estas abstracciones son similares a las respectivas abstracciones de la especificación OpenCL. En el proceso de inicialización de la plataforma se crea una instancia de la abstracción de dispositivo para cada dispositivo de la plataforma. A través de esta abstracción, el desarrollador de la aplicación puede controlar los *kernels* programados en el dispositivo físico al que se ha adjuntado la instancia. La abstracción de *buffer* representa un área de memoria situada en la memoria del dispositivo, y el API proporciona funciones de alto nivel para realizar transferencias eficientes de datos de memoria a memoria desde la memoria del *host* a los *buffers* asignados en algunos de los bancos de memoria del dispositivo. La capa de *runtime* se sitúa entre el API y el HAL. Proporciona los servicios necesarios para implementar el API, encargándose de la gestión global de la memoria de la plataforma de forma flexible. El HAL proporciona un interfaz de alto nivel que oculta los detalles de la plataforma física a la capa de ejecución y se basa en el controlador XDMA de Xilinx [21] para implementar los detalles de comunicación de bajo nivel con la plataforma hardware. Proporciona dos modos de acceso al hardware. En primer lugar, permite leer y escribir registros de datos de 32 o 64 bits utilizando direcciones de registro de 32 bits. En segundo lugar, permite programar el DMA de la plataforma para realizar transferencias

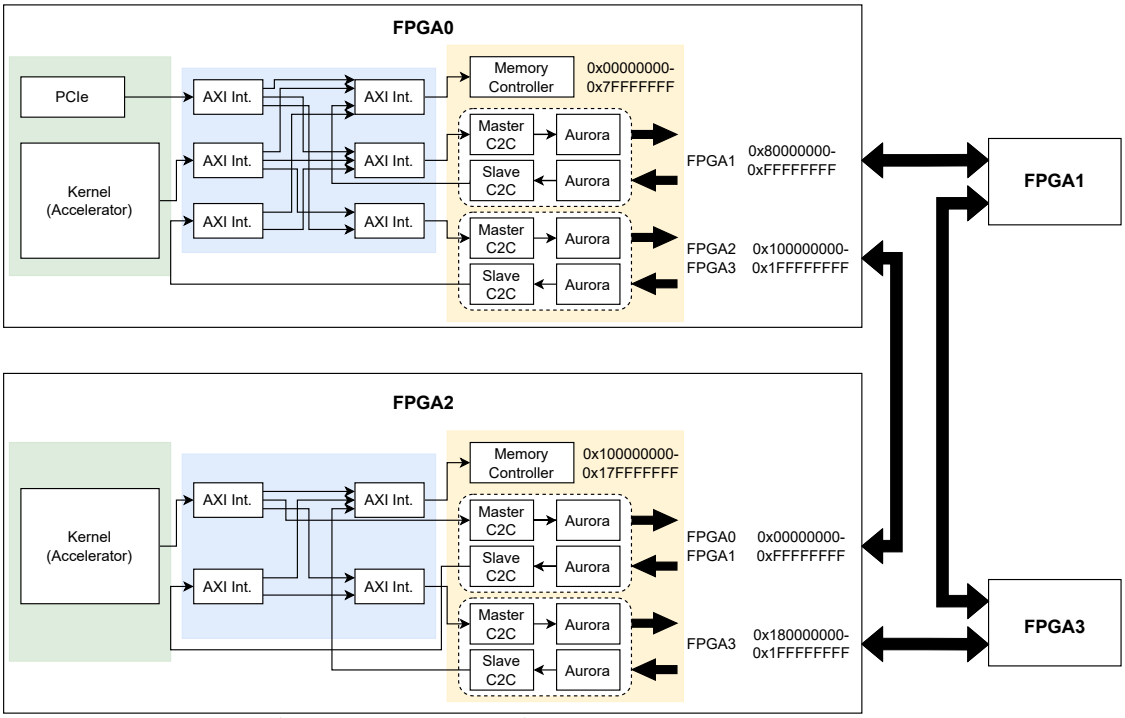


Fig. 2: Arquitectura Multi-FPGA con memoria compartida y disytibuida

Rango de direcciones	$FPGA_0$	$FPGA_1$	$FPGA_2$	$FPGA_3$
0x000000000 0x07FFFFFFF	Local	$FPGA_1 \rightarrow FPGA_0$	$FPGA_2 \rightarrow FPGA_0$	$FPGA_3 \rightarrow FPGA_1 \rightarrow FPGA_0$
0x080000000 0x0FFFFFFFF	$FPGA_0 \rightarrow FPGA_1$	Local	$FPGA_2 \rightarrow FPGA_0 \rightarrow FPGA_1$	$FPGA_3 \rightarrow FPGA_1$
0x100000000 0x17FFFFFFF	$FPGA_0 \rightarrow FPGA_2$	$FPGA_1 \rightarrow FPGA_3 \rightarrow FPGA_2$	Local	$FPGA_3 \rightarrow FPGA_2$
0x180000000 0x1FFFFFFFF	$FPGA_0 \rightarrow FPGA_2 \rightarrow FPGA_3$	$FPGA_1 \rightarrow FPGA_3$	$FPGA_2 \rightarrow FPGA_3$	Local

Tabla I: Configuración de las rutas en la plataforma multi-FPGA.

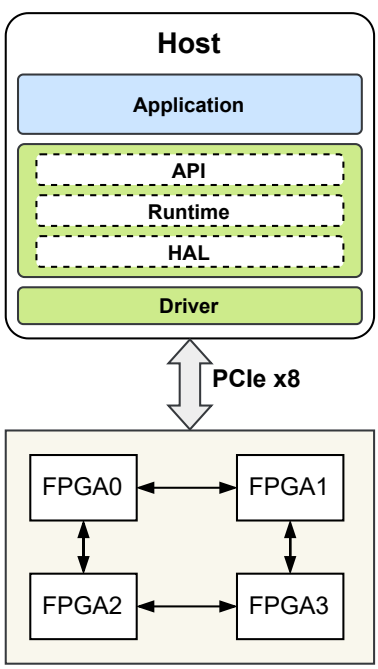


Fig. 3: Jerarquia de capas en el host

de memoria eficientes entre la memoria del *host* y la del dispositivo.

C. Evaluación de los enlaces entre FPGA

Una vez definida la plataforma con soporte para memoria compartida distribuida, necesitamos definir la estrategia de asignación y mapeo tanto de los datos como de la carga de trabajo del acelerador. Hay que tener en cuenta que, dependiendo de la ubicación de los datos, los aceleradores pueden necesitar acceder a memorias no locales e incluso pudiendo congestionar algunos enlaces. Antes de definir esta política de asignación, debemos evaluar el ancho de banda alcanzable por los enlaces FPGA a FPGA y analizar los requisitos de lectura y escritura de los distintos tipos de datos utilizados en el proceso de inferencia.

Para comprobar el impacto en el rendimiento de los enlaces, la Tabla II muestra el rendimiento relativo alcanzado al ejecutar una convolución 2D con una entrada $256 \times 256 \times 256$. El acelerador se ha probado ejecutándose en cada FPGA (K_i cuando se ejecuta en $FPGA_i$) y los datos se han ubicado (entrada, salida y filtros) en cada memoria DDR (DDR_i). El acceso remoto a las DDR vecinas tiene un impacto insignificante en el rendimiento en comparación con la memoria local, por lo que ambas son comparables. Sin embargo, acceder a una DDR remota no vecina (por ejemplo, $K_0 \leftrightarrow DDR_3$) conlleva una penalización de alrededor del 9-11 %. Por tanto, el acceso a memorias remotas no vecinas debe minimizarse y controlarse.

Otro aspecto importante que hay que analizar son los requisitos de lectura y escritura del acelerador para los distintos tipos de datos (entrada, filtros y salidas). De hecho, las operaciones de convolución

	K_0	K_1	K_2	K_3
DDR_0	1.00	0.99	0.99	0.90
DDR_1	0.99	1.00	0.90	0.99
DDR_2	0.99	0.91	1.00	0.99
DDR_3	0.88	0.99	0.99	1.00

Tabla II: Rendimiento relativo de una convolución $256 \times 256 \times 256$ en diferentes FPGA y DDR.

tienen una gran intensidad aritmética, ya que los pesos se reutilizan en toda la entrada. Esto sugiere que los requisitos de ancho de banda de lectura de los pesos son mucho menores que los requisitos de ancho de banda de lectura de la entrada y de escritura de la salida. Para confirmarlo, hemos reproducido el experimento anterior, pero en este caso hemos asignado los datos de entrada y salida en la memoria DDR local donde se ejecuta el acelerador. Los pesos se colocan en la DDR más distante (por ejemplo, k_0 ejecutándose y accediendo a los datos de entrada y salida desde DDR_0 y los pesos en DDR_3). El rendimiento del acelerador al realizar la convolución es igual al rendimiento óptimo conseguido cuando todos los datos están en local. Esto significa que la ubicación de los pesos no afecta al rendimiento final. Como experimento adicional, colocamos los datos de entrada y salida en la DDR remota y los pesos en la DDR local. En este caso, logramos el impacto más significativo acercándonos a 0,9 de rendimiento relativo.

D. Algoritmo de asignación/mapeo

Hemos desarrollado un algoritmo de asignación (Listado 1) para determinar qué *kernels* se ejecutarán y qué trabajo realizarán (que filas leer de los canales de entrada y que salida calcular). El algoritmo utiliza esquemas de particionado de canal de salida (OCP) y filas de entrada (IRP) al mismo tiempo (lo denominamos partición híbrida, HP), o sólo OCP. El algoritmo (Listado 1) se ejecuta antes de realizar la inferencia para cada capa del modelo.

Algorithm 1 Algoritmo de asignación.

1: $n_k \leftarrow min(4, \frac{O}{CPO})$
2: **if** $H \geq row_{th}$ & $n_k = 4$ **then** ▷ HP
3: $rows_{\{k_0,k_1\}} \leftarrow 0 \ldots \frac{H}{2} - 1$
4: $rows_{\{k_2,k_3\}} \leftarrow \frac{H}{2} \ldots H - 1$
5: $OC_{\{k_0,k_2\}} \leftarrow 0 \ldots \frac{O}{2} - 1$
6: $OC_{\{k_1,k_3\}} \leftarrow \frac{0}{2} \ldots O - 1$
7: ASSIGN BUFFER(...)
8: LAUNCH KERNEL($k_{th}, rows, OC, \ldots$)
9: **else** ▷ OCP
10: $OC_{\{k_{th}\}} \leftarrow \frac{O}{n_k} \times k_{th} \ldots \frac{O}{n_k} \times (k_{th} + 1) - 1$
11: ASSIGN BUFFER(...)
12: LAUNCH KERNEL($k_{th}, OC, \ldots$)
13: **end if**

El algoritmo toma como umbral el número de filas (row_{th}) para decidir si se utiliza HP u OCP (línea 2). Este valor se ha obtenido experimentalmente de la plataforma y depende del tipo de datos utilizado y del ancho de banda del enlace. Para nuestras pruebas se ha fijado en 56. El número de canales de salida determina cuántos aceleradores se lanzarán (línea 1). Suponemos que se proporcionan suficientes canales de salida para dividir la carga de trabajo entre dos aceleradores en el caso de HP (líneas 5-6).

Con HP la configuración de entrada (líneas 3-4) y salida (líneas 5-6) de cada acelerador es diferente (los datos de entrada pueden distribuirse en diferentes memorias DDR). En este caso, el *runtime* utiliza DDR_0 y DDR_2 para almacenar los datos de entrada ubicando cada mitad en una DDR. Por lo tanto, cada acelerador procesará $I \times \frac{H}{2} \times W$ de los datos de entrada y calculará la mitad de los canales de salida para esa parte de los datos de entrada: $\frac{O}{2} \times \frac{H}{2} \times W$. Observe que la asignación de las DDR la realiza el *runtime*.

Además, K_0 y K_1 trabajarán en tándem con DDR_0, así como K_2 y K_3 con DDR_2. Dado que el acceso a filtros y sesgos tiene un impacto insignificante en el rendimiento, como se ha comentado antes, se ubicarán en DDR_0. Por otro lado, con OCP el algoritmo asignará $\frac{O}{n_k}$ canales de salida a cada acelerador. Observe que n_k determina el número de aceleradores a utilizar. Los pesos no se comparten, por lo que se pueden distribuir entre todas las memorias. La función ASSIGN_BUFFER (líneas 7 y 11) configura los *buffers* para salidas y pesos, y los asigna a las memorias.

La Figura 4 muestra el resultado producido para el modelo VGG16 como ejemplo. Se puede ver en las siete primeras capas se utiliza HP, mientras que las capas restantes sólo utilizan OCP. Obsérvese cómo los cuatro *kernels* se ejecutan de forma concurrente para todas las capas, logrando la máxima utilización. En las siete primeras capas, dos grupos de *kernels* trabajan en los mismos canales de salida, pero cada uno de ellos calcula filas de salida diferentes. El *runtime* divide los datos entre DDR_0 y DDR_2, y los *kernels* son los que escriben en la memoria DDR correcta. Por ejemplo, en la primera capa todos los *kernels* leen los datos de entrada de DDR_0 pero dos de los *kernels* escriben los resultados en DDR_2. En las capas siguientes ya se tiene en cuenta esta división, por lo que K_2 y K_3 operan en DDR_2. Obsérvese que en la séptima capa todos los *kernels* escriben en DDR_0, ya que la capa siguiente no utiliza HP. A partir de esa capa, todos los *kernels* trabajan simultáneamente en diferentes intervalos de canales de salida, explotando así sólo OCP.

V. Resultados

Para evaluar la plataforma hemos ejecutado múltiples inferencias y calculado el tiempo medio para cada caso. El tiempo de inferencia se ha medido desde el *host*. Para realizar las pruebas hemos utilizado el acelerador HLSinf [22] configurado para utilizar tipo de datos FP32. Aunque es recomendable utilizar tipos de datos de coma fija en FPGAs, el acelerador consigue un mejor rendimiento con FP32. Además, utilizar FP32 nos permite estresar los enlaces entre FPGAs. Para la aplicación del lado del *host* hemos

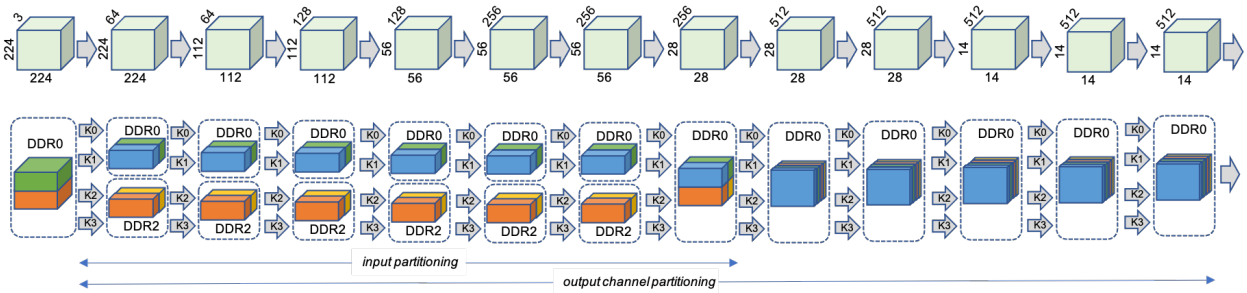
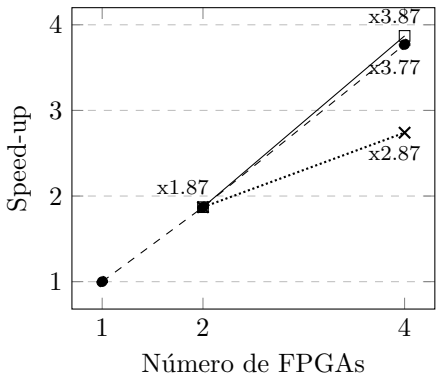


Fig. 4: Resultado de ejecutar el algoritmo para la VGG en la plataforma multi-FPGA.

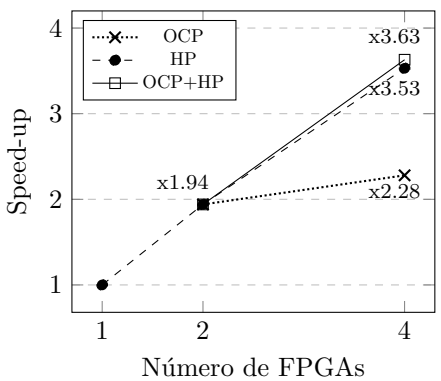
utilizado la librería EDDL [23] para entrenar e inferir modelos de redes neuronales, de forma similar a TensorFlow. Se han utilizado los modelos VGG16, *Hourglass neural network for human pose estimation* (HG) y *Residual network* (Resnet50). Comparamos los resultados de aumento de rendimiento utilizando particionado de canal de salida (OCP) con todos los aceleradores accediendo a la misma DDR, con particionado híbrido (HP) y una combinación de ambas estrategias utilizando el algoritmo de mapeo/asignación descrito en la sección anterior. Los datos de entrada son $3 \times 224 \times 224$ píxeles.

En primer lugar, analizamos el rendimiento al utilizar OCP. En esta prueba, cada acelerador tiene sus pesos y sesgos en su memoria local (K_i en DDR_i), mientras que los datos (entrada y salida) se mantienen en DDR_0. Así, todos los aceleradores acceden a DDR_0 al mismo tiempo. La Figura 5 muestra el rendimiento conseguido en comparación con una sola FPGA para los tres modelos de CNN analizados. Los resultados con OCP se corresponden con la línea de puntos. Como se puede observar, con dos FPGAs el aumento de velocidad es casi lineal, sin embargo, con cuatro FPGAs la plataforma no obtiene un rendimiento óptimo. Como ya se ha comentado, la causa de esta caída es la sobrecarga en uno de los enlaces, así como el acceso a memoria no vecina sin ningún mecanismo para controlarlo.

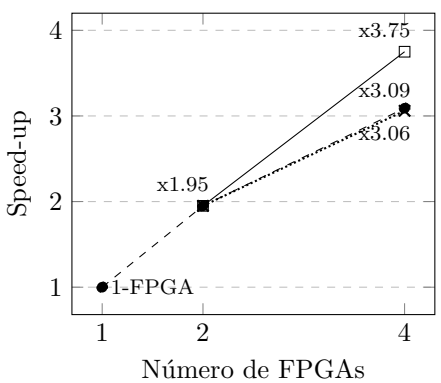
Para evitar esta pérdida de prestaciones, utilizamos particionado híbrido (HP), combinando IRP con OCP. De esta forma, en lugar de tener los datos de entrada en una DDR a la que acceden todos los aceleradores, los datos de entrada se distribuyen en dos DDR (DDR_0 y DDR_2). Al igual que con OCP, los pesos y sesgos se ubican en la memoria local de cada acelerador. El rendimiento conseguido con HP se muestra en la figura 5 (línea discontinua). En los modelos VGG y HG los resultados han mejorado significativamente. Sin embargo, en el modelo Resnet el uso de HP se tradujo en ganancias insignificantes. Aunque hemos abordado la causa de la pérdida de rendimiento, la plataforma sigue sin lograr resultados óptimos.



(a) VGG.



(b) HG.



(c) Resnet.

Fig. 5: Resultados obtenidos con varias FPGAs y diferentes esquemas de particionado.

Para aumentar más el rendimiento, hemos analizado el rendimiento obtenido con ambas estrategias. En la tabla III se muestra el rendimiento conseguido para cada capa del modelo VGG al utilizar OCP o HP. Como se puede observar, algunas capas se comportan mucho mejor con HP mientras que otras lo hacen con OCP. Las capas que funcionan mejor con OCP se corresponden con capas con un gran número de canales de salida y pocas filas por canal, mientras que las capas con un gran número de filas por canal se comportan mejor con HP. Esto motiva el algoritmo que hemos introducido anteriormente.

Configuración	OCP	HP	OCP vs HP
3x224x224	2,80	3,70	0,90
64x224x224	1,75	3,94	2,19
64x112x112	1,78	3,88	2,11
128x112x112	2,04	3,90	1,86
128x56x56	2,45	3,70	1,25
256x56x56	2,45	3,84	1,39
256x56x56	3,76	3,84	0,08
256x28x28	3,85	3,70	-0,16
512x28x28	3,87	3,73	-0,14
512x28x28	3,87	3,72	-0,14
512x14x14	3,91	3,43	-0,48
512x14x14	3,91	3,43	-0,48
512x14x14	3,81	3,43	-0,38

Tabla III: Rendimiento capa a capa con OCP y HP para la VGG16.

Así pues, aplicando el algoritmo de mapeo/asignación algunas capas se ejecutan con HP y otras con OCP. Los datos también se ubican estratégicamente en DDR_0 o se dividen en DDR_0 y DDR_2 en función del algoritmo de asignación. La Figura 5 muestra el rendimiento conseguido al utilizar el algoritmo (línea densa). Con el uso de ambas estrategias de particionado aumentamos el rendimiento con cuatro FPGAs. Con VGG y HG ganamos un 10 % en rendimiento mientras que con ResNet la ganancia es del 66 %.

Como se puede observar en los resultados, todavía hay margen de mejora. De hecho, VGG, HG y ResNet alcanzan factores de aceleración de 3,87, 3,63 y 3,75, respectivamente. Tras medir los anchos de banda de enlace y los requisitos de ancho de banda de memoria de los distintos modelos, confirmamos que la plataforma no introducía ningún cuello de botella. Así pues, nos centramos en la eficiencia del acelerador. La tabla IV muestra el rendimiento obtenido por cuatro modelos sintéticos diferentes compuestos por una única capa de convolución 2D, cada uno con un tamaño de convolución diferente. Ejecutamos la inferencia de esos modelos en una FPGA y comparamos con el tiempo de ejecución teórico para obtener la eficiencia. El tiempo de ejecución teórico se obtiene mediante la fórmula $H \times W \times \frac{I}{CPI} \times \frac{O}{CPO}$ ciclos, siendo CPI y CPO el número de canales de entrada y salida procesados en paralelo por el acelerador.

Configuración	Eficiencia
16x16x16x16	0.39
64x64x64x64	0.93
128x128x128x128	0.97
512x512x256x256	0.98

Tabla IV: Eficiencia del acelerador con diferentes tamaños de convolución definidos como $I \times 0 \times H \times W$.

Así pues, la razón de la pérdida de rendimiento se debe a algunas ineficiencias internas del acelerador. Para convoluciones pequeñas, el acelerador no alcanza el máximo rendimiento. Por el contrario, la plataforma gestiona grandes convoluciones con un factor de eficiencia cercano al factor ideal.

VI. Conclusión

En este articulo hemos presentamos una plataforma multi-FPGA con memoria distribuida compartida para la inferencia de CNNs. Gracias al soporte de memoria distribuida compartida, podemos combinar diferentes estrategias de particionado para lograr una aceleración cercana a la lineal con hasta cuatro FPGAs. Además, hemos ideado un algoritmo de mapeo preliminar que tiene en cuenta los parámetros de cada capa de un modelo dado y aplica diferentes estrategias de particionado para obtener un rendimiento óptimo. La plataforma dispone de recursos de comunicación suficientes para aumentar el número de FPGAs, y el algoritmo de mapeo y la biblioteca pueden mejorarse para lograr un mayor rendimiento y soportar cargas de trabajo más complejas.

Agradecimientos

Este proyecto ha recibido financiación de la European High-Performance Computing Joint Undertaking (JU) en virtud del acuerdo de subvención nº 955558 (proyecto eFlows4HPC). La JU recibe apoyo del programa de investigación e innovación Horizonte 2020 de la Unión Europea, y de España, Alemania, Francia, Italia, Polonia, Suiza y Noruega.

Referencias

[1] Eriko Nurvitadhi et al., "Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC," in *2016 Int. Conf. on Field-Programmable Technology*, Xi'an, China, Dec. 2016, pp. 77–84, IEEE.

[2] Jiantao Qiu et al., "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," in *Proc. of the 2016 ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*. Feb. 2016, pp. 26–35, ACM.

[3] Jason Cong et al., "Understanding Performance Differences of FPGAs and GPUs," in *2018 IEEE 26th Annual Int. Symp. on Field-Programmable Custom Computing Machines*, Boulder, CO, USA, Apr. 2018, IEEE.

[4] Tong Geng et al., "A Framework for Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters with Work and Weight Load Balancing," in *2018 28th Int. Conf. on Field Programmable Logic and Applications (FPL)*, Dublin, Ireland, Aug. 2018, pp. 394–3944, IEEE.

[5] Chen Zhang et al., "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proc. of the 2015 ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*. 2015, pp. 161–170, ACM.

[6] Yufei Ma et al., "Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks," in *Proc. of the 2017 ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, Monterey California USA, Feb. 2017, pp. 45–54, ACM.

[7] Naveen Suda et al., "Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks," in *Proc. of the 2016 ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, Monterey California USA, Feb. 2016, pp. 16–25, ACM.

[8] Chen Zhang et al., "Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster," in *Proc. of the 2016 Int. Symp. on Low Power Electronics and Design*, San Francisco Airport CA USA, Aug. 2016, pp. 326–331, ACM.

[9] Wentai Zhang et al., "An Efficient Mapping Approach to Large-Scale DNNs on Multi-FPGA Architectures," in *2019 Design, Automation & Test in Europe Conf. & Exhibition*. Mar. 2019, pp. 1241–1244, IEEE.

[10] Junzhong Shen et al., "Scale-out Acceleration for 3D CNN-based Lung Nodule Segmentation on a Multi-FPGA System," in *Proc. of the 56th Annual Design Automation Conf. 2019*, Las Vegas NV USA, June 2019, ACM.

[11] Saman Biookaghazadeh et al., "Toward Multi-FPGA Acceleration of the Neural Networks," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 17, no. 2, pp. 1–23, Apr. 2021.
[12] Junnan Shan et al., "CNN-on-AWS: Efficient Allocation of Multikernel Applications on Multi-FPGA Platforms," *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 2, Feb. 2021.
[13] Ruihao Li et al., "Improving CNN performance on FPGA clusters through topology exploration," in *Proc. of the 36th Annual ACM Symp. on Applied Computing*, Virtual Event Republic of Korea, Mar. 2021, ACM.
[14] Tobias Alonso et al., "Elastic-DF: Scaling Performance of DNN Inference in FPGA Clouds through Automatic Partitioning," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 2, June 2022.
[15] Weiwen Jiang et al., "Achieving Super-Linear Speedup across Multi-FPGA for Real-Time DNN Inference," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 5s, pp. 1–23, Oct. 2019.
[16] Eric Chung et al., "Serving DNNs in Real Time at Datacenter Scale with Project Brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, Mar. 2018.
[17] Jeremy Fowers et al., "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *2018 ACM/IEEE 45th Annual Int. Symp. on Computer Architecture (ISCA)*, Los Angeles, CA, June 2018, pp. 1–14, IEEE.
[18] Yuxi Sun et al., "FiC-RNN: A Multi-FPGA Acceleration Framework for Deep Recurrent Neural Networks," *IEICE Transactions on Information and Systems*, vol. E103.D, no. 12, pp. 2457–2462, Dec. 2020.
[19] Xilinx, "Chip2Chip," https://docs.xilinx.com/r/en-US/pg067-axi-chip2chip.
[20] Xilinx, "Aurora 64B/66B," https://docs.xilinx.com/r/en-US/pg074-aurora-64b66b.
[21] Xilinx, "Xilinx DMA IP Reference drivers," https://github.com/Xilinx/dma_ip_drivers.
[22] Jose Flich et al., "Efficient Inference Of Image-Based Neural Network Models In Reconfigurable Systems With Pruning And Quantization," in *2022 IEEE International Conference on Image Processing (ICIP)*, Bordeaux, France, Oct. 2022, pp. 2491–2495, IEEE.
[23] H. D. project, "European Distributed Deep Learning (EDDL) Library," https://github.com/deephealthproject/eddl.

Sistema distribuido de archivos *ad hoc* para cargas de trabajo intensivas en datos en aplicaciones HPC

Genaro Sanchez-Gallegos[1], Javier Garcia-Blas[1], Cosmin Petre[1] y Jesus Carretero[1]

Resumen—
La creciente demanda de procesamiento de información por parte de las nuevas aplicaciones intensivas en datos está ejerciendo una alta presión sobre el rendimiento y la capacidad de los sistemas de almacenamiento de altas prestaciones. Por lo tanto, los nuevos avances en las tecnologías de almacenamiento tienen como objetivo satisfacer estas demandas. Sin embargo, depender únicamente de estos dispositivos de almacenamiento no es rentable, lo que lleva a la necesidad de jerarquías de almacenamiento de varios niveles. Esta aproximación se basa en el movimiento de los datos en función de su uso. Para resolver este problema, se han propuesto como solución los sistemas de archivos *ad hoc*, los cuales utilizan el almacenamiento disponible de los nodos de computación, como la memoria y el almacenamiento persistente, para crear un sistema de archivos temporal que se adapte al comportamiento de la aplicación en entornos HPC.

Este trabajo presenta el diseño, implementación y evaluación de un sistema distribuido de archivos *ad hoc* para cargas de trabajo intensivas en datos en aplicaciones HPC llamado Hercules. Este trabajo se centra en el nuevo modelo de comunicación basado en el framework Unified Communication X (UCX). Esta solución aprovecha las capacidades de los protocolos RDMA (incluidos Infiniband y Onmipath), memoria compartida y transferencias *zero-copy*. Los resultados preliminares de la evaluación muestran la escalabilidad de nuestra solución al contar con una mayor disponibilidad de recursos frente a otros sistemas de ficheros.

Palabras clave— **E/S HPC, Uso intensivo de datos, Almacenamiento en memoria**

I. Introducción

El uso intensivo de datos se ha convertido en una característica común de las aplicaciones científicas y de ingeniería. Debido a ello, actualmente los desarrolladores de sistemas de Computación de Alto Rendimiento (HPC, por sus siglas en inglés) se encuentran interesados en acelerar las operaciones de entrada/-salida (E/S), lo cual se ve reflejado en el rendimiento general del sistema. Para lograr esto, existen enfoques dedicados a mejorar los sistemas de almacenamiento (*e.g.*, NVMe y memoria persistente), lo cual no resulta factible para los sistemas de ficheros tradicionales, debido a que estos pudieran no explotar todas las ventajas que traen consigo los nuevos sistemas de almacenamiento.

Por otro lado, los sistemas de E/S en HPC (que han sido diseñados para aplicaciones que acceden a unos pocos archivos muy grandes de forma mayoritariamente secuencial) no son lo suficientemente flexibles para adaptarse al comportamiento dinámico de las aplicaciones, en las cuales, se espera una mayor cantidad de operaciones de lectura que de escritura, el uso de un gran número de archivos pequeños o el acceso aleatorio a los mismos, lo cual degrada considerablemente su rendimiento. Asimismo, los sistemas de ficheros deben ser capaces de gestionar datos a través de infraestructuras heterogéneas de forma transparente, y dicha gestión debe añadir la menor sobrecarga posible a las aplicaciones [1].

Hoy en día, muchas cargas de trabajo de datos están impulsadas por el aprendizaje automático (ML) y otras técnicas de análisis de datos que se basan en *workflows* (Apache Spark), paquetes de ML (Horovod, TensorFlow) y bibliotecas específicas de dominio que tradicionalmente no se han utilizado en HPC. Como demostraron Chowdhury *et al.* [2], las aplicaciones de ML están dominadas principalmente por un gran número de archivos pequeños y operaciones POSIX de lectura y búsqueda. Estos patrones de E/S no funcionan bien en los actuales sistemas de E/S sobre sistemas HPC, que han sido diseñados para aplicaciones que acceden a unos pocos archivos muy grandes de forma secuencial. Nuestra hipótesis es que esas aplicaciones pueden acelerarse reduciendo el cuello de botella de E/S inducido por el sistema de ficheros, y que facilitar el almacenamiento de datos temporales en un sistema de ficheros *ad hoc* puede afectar significativamente al rendimiento global.

Este trabajo presenta el diseño, implementación y evaluación de un sistema distribuido de almacenamiento *ad hoc* en memoria (HERCULES[1]), una propuesta para mejorar la E/S tanto en sistemas HPC tradicionales como en sistemas de análisis de datos de alto rendimiento (HPDA). El diseño arquitectónico sigue un modelo de diseño cliente-servidor, en el que el propio cliente será el responsable del despliegue de las entidades servidoras. La capa cliente (*frontend*) es la encargada de tratar la explotación de la localidad de los datos junto con la implementación de múltiples patrones de E/S que proporcionan diversas políticas de distribución de datos.

En un trabajo anterior, presentamos una versión preliminar de nuestro sistema de ficheros *ad hoc* [3], donde proponíamos dos tipos de despliegues: un despliegue "application-attached" restringido a los nodos de la aplicación, y un despliegue "application-detached" considerando nodos *offshore*. Identifica-

[1]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: {fjblas,jcarrete}@inf.uc3m.es, {gesanche,cpetre}@pa.uc3m.es

[1]Disponible en https://gitlab.arcos.inf.uc3m.es/admire/hercules.git.

mos algunas limitaciones en esta implementación anterior: en primer lugar, utilizaba ZeroMQ como mecanismo de comunicación, el cual ofrece varias alternativas de transporte como *in-process*, *inter-process*, TCP y *multicast*. Sin embargo, carece de soporte para redes de alto rendimiento y limita su portabilidad; en segundo lugar, esa versión preliminar ofrecía una API basada en librerías, sin ofrecer una interfaz compatible con POSIX, claramente extendida y utilizada por la comunidad HPC, sino solo una interfaz orientada a objetos. En la versión presentada en este artículo hemos abordado ambos puntos débiles, sustituyendo la capa de comunicación por UCX y proporcionando una biblioteca compatible con POSIX. En comparación con los trabajos relacionados, los puntos fuertes de esta nueva versión de HERCULES son el uso de recursos de memoria principal, el soporte completo de POSIX y la portabilidad de red.

El resto del documento se organiza como sigue. La sección II muestra trabajos relacionados con HERCULES. La sección III describe la arquitectura de HERCULES. En la Sección IV, describimos el diseño de nuestro sistema de ficheros paralelo basado en POSIX. La sección V se centra en la integración de HERCULES y UCX. La sección VI muestra los resultados de rendimiento obtenidos en la fase de evaluación. Finalmente, la sección VII concluye el artículo y enumera futuros trabajos.

II. Trabajos relacionados

La memoria persistente es una tecnología de almacenamiento de alta velocidad que permite reducir/eliminar copias desde la memoria al disco, disminuyendo la sobrecarga de operaciones E/S a nivel de software. En la literatura se encuentran soluciones tales como GPFS [4], Lustre [5] y SPMFS [6], las cuales correspondes con sistemas de ficheros paralelos de propósito general que proporcionan almacenamiento persistente en sistemas HPC. Sin embargo, estas soluciones no pueden ser modificadas o ajustadas a una aplicación una vez que han sido desplegadas.

La creciente complejidad de los accesos de E/S en plataformas HPC resultante de añadir nuevas capas y dispositivos de E/S, genera un aumento de la latencia de las operaciones de E/S, lo cual dificulta el rendimiento de las aplicaciones. Así, los casos de uso actuales han potenciado el incremento de sistemas de almacenamiento de baja latencia utilizando dispositivos de almacenamiento en memoria locales o remotos como una aproximación factible al problema [7]. Tal ha sido el impacto de estos sistemas de almacenamiento [8] que en los últimos años se han implementado múltiples soluciones, como bases de datos relacionales en memoria, bases de datos NoSQL en memoria, sistemas de caché en memoria y sistemas de procesamiento de datos en memoria.

Una alternativa distinta son los sistemas de ficheros *ad hoc* [9], los cuales son una solución viable debido a que aprovechan los recursos locales de almacenamiento y se adaptan a las necesidades de cada aplicación [10], [11]. Estos sistemas permiten crear sistemas de archivos temporales que se ajustan al entorno HPC según el perfil de la aplicación o las sugerencias de los usuarios.

GekkoFS [10] es un sistema de archivos de alto rendimiento, paralelo y distribuido para entornos HPC. Su arquitectura basada en niveles mueve los datos entre diferentes niveles de almacenamiento, equilibrando el rendimiento y la rentabilidad. Además, soporta operaciones de E/S paralelas y distribuidas, lo que mejora su rendimiento general.

FlexMPI [12] es una herramienta que proporciona un balanceo de carga dinámico en aplicaciones que hacen uso MPI. Esta solución permite utilizar los mecanismos de control de CLARISSE [13] basándose en dos estrategias de optimización diferentes, con el objetivo de mejorar tanto la E/S de la aplicación y el rendimiento general del sistema. Las aplicaciones se analizan utilizando los componentes de monitorización de CLARISSE y FlexMPI, los cuales envían los datos a un marco externo que recopila la información de todas las aplicaciones ejecutadas en el sistema, teniendo en cuenta los rendimientos de CPU, comunicación y E/S.

Ceph [14] presenta BlueStore, un sistema de archivos distribuido que implementa un back-end de almacenamiento desplegado directamente en dispositivos de almacenamiento en bruto. BlueStore trabaja mediante una técnica clave-valor ordenada, con transacciones de escritura anticipada en registros. Para hacer frente a la lentitud de las operaciones de metadatos (*e.g.*, enumeración de directorios (*readdir*) en directorios de gran tamaño), dentro de esta solución se distribuyen los objetos entre directorios, se ordena el contenido de algunos directorios seleccionados y se mantienen los directorios con un tamaño reducido dividiéndolos cuando crece el número de entradas. BlueStore implementa la especificación de su propia interfaz con llamadas básicas al sistema (*open, mkdir* y *pwrite*). Los datos se almacenan mediante un asignador de espacio que se conecta con un servicio de metadatos para determinar el espacio disponible.

Reda *et. al.* [1] propone tres técnicas enfocadas en reducir la latencia E/S producida en sistemas de almacenamiento, y su flexibilidad en el escalamiento al añadir/reducir nodos, las cuales resultan de interés, ya que una buena implementación resultará en menores tiempos de respuesta y un mayor rendimiento de la aplicación, los cuales son afectados principalmente por la heterogeneidad en la infraestructura y el estado de la red. Las técnicas propuestas son: 1) realizar la gestión de metadatos en el mismo lugar en donde se ejecutan los clientes, permitiendo obtener una mayor localidad, implicando añadir carga extra al nodo; 2) sincronizador con múltiples políticas atendiendo las peticiones de acuerdo a su posible contribución en la generación de cuellos de botella; y 3) reducir la contención entre subprocesos mediante colas siguiendo el estilo PEPS (FIFO, por sus siglas en inglés) y la asignación de costes a las peticiones.

III. Sistema de ficheros *ad hoc* HERCULES

En la Fig. 1 se muestra el diseño arquitectural de HERCULES, basado en el modelo de diseño cliente-servidor. HERCULES en un sistema de ficheros *ad hoc* que es desplegado junto con cada una de las aplicaciones en donde se realiza la gestión de los metadatos y datos. HERCULES se incluye en cada una de las aplicaciones como una capa front-end adicional, la cual puede ser configurada para ajustarse a las necesidades de E/S de datos de manera independiente.

En el presente trabajo, HERCULES proporciona flexibilidad en términos de despliegue; para lograrlo, se cuenta con múltiples servidores distribuidos en donde cada uno de ellos cuentan con un gestor de tareas y un conjunto de subprocesos. El gestor de tareas distribuye la carga de trabajo entrante entre los subprocesos con el objetivo de balancear la carga de trabajo en un escenario multiproceso. Las principales entidades que conforman el diseño son los procesos clientes (HERCULES front-end), los nodos de metadatos (1 a M) y datos (1 a K) (HERCULES back-end).

Abordando la interacción entre estos componentes, el front-end de HERCULES se comunica exclusivamente con los servidores de metadatos cada vez que se realiza una operación relacionada con esta categoría, tales como *create* y *open*. Las operaciones relacionadas con los datos serán gestionadas directamente por un servidor de datos, el cual es calculado en el lado del front-end, por lo que no es necesario el mapeo global de datos en todos los niveles.

Además, HERCULES ofrece a la aplicación un conjunto de políticas de distribución de datos a nivel del conjunto de datos, aumentando el conocimiento de la aplicación sobre la ubicación de los mismos. Un conjunto de datos se define como una colección de elementos con un tamaño fijo que se distribuyen entre los servidores de datos de una única instancia de HERCULES, de acuerdo con una política específica de distribución. Como resultado, el sistema de almacenamiento obtendrá beneficios en términos de distribución de datos en el lado del cliente, tales como la explotación de la localidad y el balanceo de carga.

Por lo tanto, las aplicaciones desplegadas haciendo uso de HERCULES pueden determinar parámetros críticos como la política de distribución de datos, que se establece a nivel de conjunto de datos y establece cómo se distribuyen los bloques de datos entre los nodos de almacenamiento. Actualmente, HERCULES cuenta con las siguientes políticas: ROUND ROBIN, BUCKETS, HASHED, CRC16bits, CRC64bits y LOCAL. Esta política de distribución puede ser configurada para cada despliegue *ad hoc* del sistema, en donde se establece para todos los conjuntos de datos. La política de distribución de datos ROUND ROBIN se ha utilizado para los resultados mostrados en el presente trabajo.

Por otro lado, se ha integrado el framework de comunicación UCX [15], diseñado para redes de HPC, debido a que nos ha permitido explotar el ancho de banda de red presente en las diferentes infraestructuras en donde se ha desplegado HERCULES. Asimismo, UCX tiene soporte de múltiples dispositivos de red (*e.g.*, Infiniband, Omni-Path y Ethernet), modelos de programación (*e.g.*, MPI, OpenSHMEM y PGAS), además de CUDA y memoria compartida para una comunicación intra-nodo. UCX ofrece dos API de red: UCT y UCP. UCT es una capa de transporte de bajo nivel para el acceso a los recursos de red de hardware de forma eficiente y UCP es una API de alto nivel que implementa varias interfaces de comunicación.

IV. Diseño del sistema de ficheros

La figura 2 muestra las diferentes abstracciones diseñadas en HERCULES, desde la perspectiva de la aplicación hasta la disposición final en memoria. Como se muestra en la figura, una ruta de archivo tipo UNIX se traduce a una entidad de conjunto de datos, asignando la ruta montada a un URI de espacio global (vista de conjunto de datos).

Cada archivo o directorio se asigna a un conjunto de datos de HERCULES compuesto por una lista de bloques de tamaño fijo (vista física). HERCULES permite el acceso a bloques por desplazamiento. Así, la capa *frontend* puede solicitar acceso a un bloque concreto en un desplazamiento específico. Las solicitudes enviadas contienen la operación deseada (lectura/escritura), el URI del conjunto de datos, el bloque al que se desea acceder y el desplazamiento. En el caso de las operaciones de escritura, la solicitud también contiene el tamaño a escribir. Esta aproximación optimiza las transferencias de datos reduciendo la cantidad de datos transmitidos.

La versión actual soporta completamente todo el estándar POSIX (*i.e.*, *open*, *close*, *write*, *rm*, *mkdir*, etc.) y también las llamadas a la librería *libc* como *fopen* y *fwrite*. En las siguientes subsecciones, cubriremos los siguientes aspectos del sistema de archivos que se ejecuta sobre HERCULES: manipulación de memoria, replicación de datos y gestión de metadatos.

A. Gestión dinámica de la memoria

Para eliminar la sobrecarga de asignaciones de memoria dinámica, HERCULES proporciona un *pool* de memoria por hilo de escucha. Esta reserva de memoria se basa en una solución sin bloqueos (*lock-free*). Este *pool* se asigna en la inicialización del *backend* y ofrece bloques de memoria alineados de tamaño fijo. En caso de que el *pool* de memoria esté totalmente utilizado, implementamos una política de sustitución de bloques LRU, que reenvía los bloques al almacenamiento persistente. Los bloques se asignan inicialmente en el espacio reservado de páginas grandes con el objetivo de reducir el tamaño de la tabla TLB y, por tanto, la latencia de acceso a la memoria.

B. Mecanismo de replicación

HERCULES ofrece un gestor de replicación gestionado en el lado del cliente. Cuando se crea un

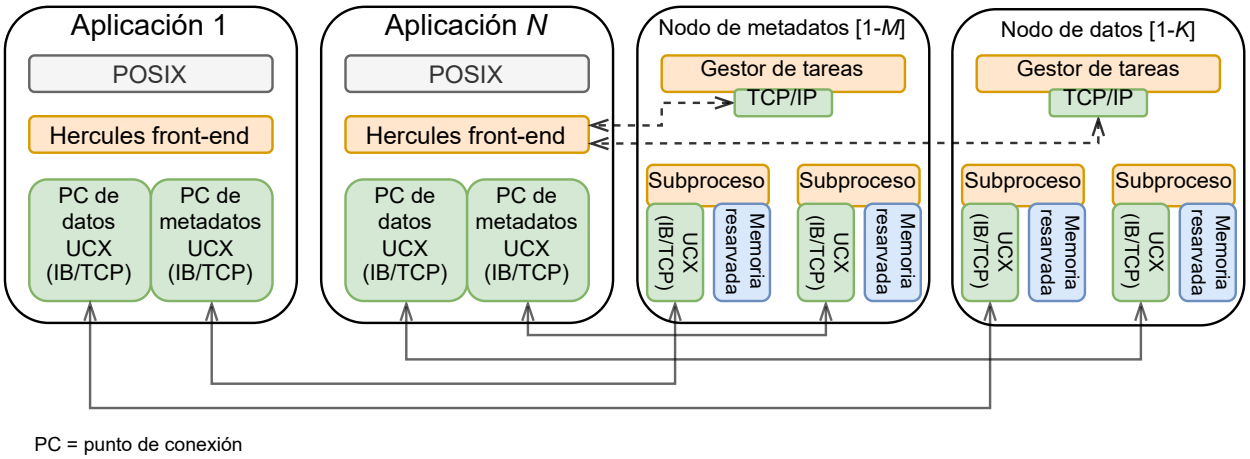


Fig. 1: Arquitectura de HERCULES siguiendo el modelo de diseño client-servidor, en donde los clientes (aplicaciones) se comunican con los servidores de metadatos y datos mediante el framework UCX/UCP.

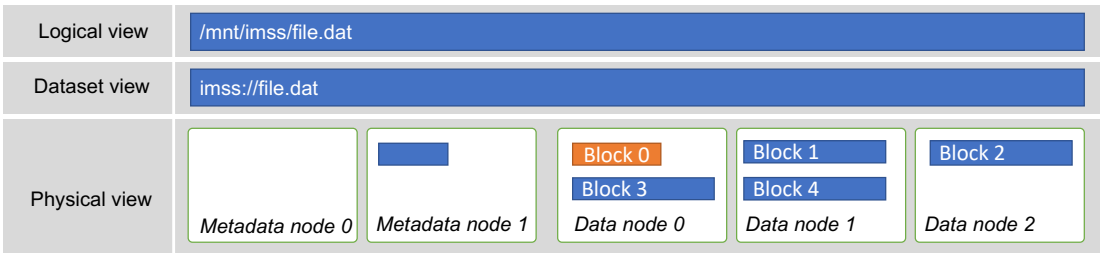


Fig. 2: Representación lógica, datos y física de un archivo en HERCULES. La ruta de datos se traduce de forma lógica a forma de conjunto de datos. Posteriormente, el archivo se divide en múltiples bloques bajo tres nodos de datos. La información de metadatos entre nodos se almacena en el nodo de metadatos 0.

conjunto de datos, la aplicación puede especificar el factor de replicación necesario para los datos. En las operaciones de escritura y lectura, las solicitudes de datos se envían a un subconjunto de *backends* determinado por el factor de replicación. Este subconjunto se crea aplicando políticas basadas en la distancia máxima entre las réplicas. Una vez que HERCULES detecta un fallo en uno o más *backends*, marca este *backend* específico como caído o roto y utiliza el siguiente de la lista. Eventualmente, la copia rota será restaurada si el nodo de datos es alcanzable. Para reducir la sobrecarga de comunicación introducida por el uso de transmisiones múltiples, las operaciones de escritura se consignan de forma asíncrona. HERCULES garantiza un sólido modelo de coherencia gestionado por los canales de comunicación UCX, que proporcionan una cola de mensajes que garantiza el orden de recepción en el *backend*. En el futuro, tenemos previsto optimizar la replicación de datos utilizando operaciones colectivas asíncronas, como *gather* y *broadcast*. Estos paradigmas de comunicación ya están soportados por la biblioteca de comunicación colectiva unificada (UCC).

C. Metadata

HERCULES se basa en dos niveles de metadatos. El primer nivel está compuesto por múltiples entidades/servidores de metadatos distribuidas que gestionan la información entre nodos relacionada con cada conjunto de datos (por ejemplo, tipo de conjunto de datos, política de distribución de bloques y factor de replicación). La biblioteca en el lado del cliente (*frontend*) selecciona el servidor de metadatos más adecuado en función de la política de distribución de datos (ROUND ROBIN, CRC, BUCKETS, etc.). Este mecanismo pretende aliviar la sobrecarga de la gestión de metadatos, especialmente en aplicaciones con un gran número de archivos pequeños. Es importante destacar que es factible una mezcla de servidores de metadatos coasignados y distribuidos.

El segundo nivel está compuesto por los metadatos de cada archivo/carpeta, que se almacenan en el *Bloque 0* en cada conjunto de datos, incluidos los metadatos tradicionales de tipo POSIX (*struct stat*). Los primeros bloques se distribuyen como cualquier otro bloque de datos y se accede a ellos aplicando funciones *hash* al URI del conjunto de datos, evitando así la necesidad de una nueva capa de servidores de metadatos y maximizando el paralelismo. Si se aplica la replicación, para garantizar la coherencia de los datos, la replicación del *Bloque 0* se orquesta en la capa del *frontend*.

Los metadatos de primer nivel se almacenan internamente como una estructura de datos opaca que representa un árbol binario equilibrado proporcionado por la biblioteca *GLib*. El árbol se equilibra automáticamente a medida que se añaden pares clave/-valor, la búsqueda de claves es $O(\log_n)$, donde n es el número de pares clave/valor en el árbol. Por lo tanto, dado nuestro enfoque distribuido de la gestión de metadatos, podemos suponer una complejidad temporal $O(\log_n / meta)$, donde $meta$ es el número de servido-

res de metadatos. Es importante señalar que el servidor de metadatos se calcula en el lado del *frontend*, utilizando nuestras políticas de distribución configurables, por lo que existe un margen de optimización en este aspecto.

V. Mecanismo de comunicación

Esta sección describe el modelo de comunicación proporcionado por HERCULES, basado en Unified Communication X (UCX). Los principales componentes de la capa de comunicación son los *workers* proporcionandos por UCX. Un *worker* UCX abstrae los recursos de red como son la interfaz de red y los protocolos de comunicación. Los *worker* UCX también representan recursos de comunicación virtuales que pueden agregar múltiples dispositivos, lo que permite a HERCULES aprovechar las ventajas de las comunicaciones multi-transporte. Esto permite la entrega de datos en múltiples interfaces de red en paralelo (*network bounding*), sin necesidad de ningún ajuste especial.

Por cada *frontend* de HERCULES (ver la Aplicación [1,$\cdots$,N] de la Fig. 1) se generan dos *workers* de UCX (etiquetados como *PC de datos UCX*), los cuales establecen una comunicación punto a punto [16] con los nodos de metadatos y datos. La consistencia de los datos entre puntos de comunicación es gestionada por UCX; mediante UCX/UCP se elige el protocolo de transferencia adecuado y se fragmenta el mensaje cuando es necesario. Como trabajo futuro, proponemos implementar un mecanismo para la consistencia de datos entre servidores de datos y metadatos.

Los nodos de metadatos y datos realizan el despliegue de un conjunto de subprocesos, así como de un gestor de tareas. Los subprocesos son creados con la finalidad de controlar la transferencia de datos y la gestión de la memoria reservada (*e.g.*, *seek*, colector de basura y el escalado de memoria), permitiendo explotar el uso de la red. Por su parte, el gestor de tareas permite atender las peticiones para el establecimiento de conexión de nuevos HERCULES front-end. Debido a que estas peticiones no suelen ser demandantes a nivel de red, la comunicación se establece mediante el protocolo TCP/IP.

Cuando se inicializa la capa del *frontend*, la biblioteca cliente solicita al hilo despachador del *backend* (gestor de tareas) la dirección del trabajador UCX. Esta dirección se utiliza para la creación de puntos de conexión en ambos lados, representando una conexión de un trabajador local a un trabajador remoto (véase la sección III). Todas las peticiones *get* y *set* se envían a través de estos puntos de conexión finales.

El *backend* de almacenamiento mantiene una lista de puntos de conexión activos, la cual se actualiza a la llegada de la primera solicitud y se destruye una vez que la biblioteca cliente libera y finaliza la ejecución de la aplicación. Los puntos de conexión se almacenan en caché y se utilizan para futuras solicitudes, lo que reduce el coste de creación. Por último, este mecanismo elimina la necesidad de tratar con un ID global, dado que, UCX genera identificadores universalmente únicos (UUID) para cada trabajador UCX.

De forma similar a MPI, HERCULES explota el paso de mensajes basado en etiquetas de múltiples formas. En primer lugar, tanto los mensajes de petición como los de datos brutos se etiquetan con valores diferentes, lo que facilita el orden de los mensajes en la recepción. UCX asegura el orden en la recepción de mensajes, emulando el modelo de consistencia POSIX. En segundo lugar, el uso de mensajes basados en etiquetas elimina la necesidad de tratar con una lista de puntos de conexión, ya que UCX ofrece un enfoque similar al de *MPI_Probe*. Por último, este mecanismo proporciona una alternativa viable a RPC, dado que las etiquetas identifican también la operación del mensaje.

VI. Resultados experimentales

En esta sección se analizan los resultados obtenidos tras evaluar el rendimiento de HERCULES mediante dos evaluaciones de escalabilidad (débil y fuerte). El hardware utilizado para llevar a cabo los experimentos consiste en un clúster de 64 nodos que ejecutan Ubuntu 20.04.5 LTS. Cada nodo está equipado con dos procesadores Intel Xeon CPU E5-2697 v4 de 16 núcleos con un total de 32 núcleos físicos y una velocidad de reloj por núcleo de 2,6 GHz. La topología de red se crea con tres conmutadores que conforman una red fat-tree de dos niveles. Todos los nodos están conectados a través de una red Intel Omni-path con un rendimiento máximo de 100 Gbps por nodo. El software empleado es UCX 1.14, OpenMPI 4.1 y la biblioteca *glib*. UCX expone la red OPA utilizando la librería *ibverbs*, alcanzando un ancho de banda similar en comparación con la instalación OpenMPI nativa. El sistema cuenta con un sistema de ficheros global compuesto por el sistema BeeGFS, el cual cuenta con un único servidor como backend de almacenamiento. El sistema de almacenamiento está basado en discos SSD y red Omni-path, permitiendo al sistema de almacenamiento alcanzar la máxima tasa de transferencia disponible. Se ha decidido comparar con este sistema ya que no permite modificaciones por parte de los usuarios finales.

Los resultados experimentales se obtuvieron utilizando el software IOR (una solución ampliamente utilizada para medir el rendimiento de E/S a escala) e IO500 [17]. Las métricas de evaluación mostradas en este documento corresponden al valor medio de cinco ejecuciones consecutivas. Esta sección muestra los resultados de un despliegue con 16 clientes estáticos distribuidos en 8 nodos (2 clientes por nodo), realizando un escalamiento de 1 a 16 servidores de datos. Todas las ejecuciones se basan en accesos secuenciales, las cuales representan el patrón de aplicaciones referenciadas en la motivación de este trabajo.

La evaluación experimental se ha dividido en dos etapas. En la primera se busca medir el rendimiento de HERCULES fijando 16 clientes que se encuentran constantemente realizando operaciones de escritura y

lectura mientras se aumenta el número de servidores de datos, buscando demostrar la capacidad de escalamiento con la que cuenta nuestra solución. Como segunda etapa, se muestra la comparación entre distintas configuraciones de HERCULES y BeeGFS; el escenario propuesto ha sido el aumentar gradualmente el número de clientes (de 16 hasta 256) mientras se mantiene un número fijo de servidores, lo cual nos permitirá observar la sobrecarga que pudiera añadirse a las soluciones cuando estas se encuentran recibiendo una mayor cantidad de peticiones.

A. *Evaluación de escalabilidad débil*

En este primer experimento, cada cliente realiza la escritura y lectura de 100 MB en un fichero compartido, sumando un total de 16 GB por experimento. La Fig. 3 muestra los resultados de la evaluación de escalabilidad débil realizada para HERCULES, divida en operaciones de escritura (ver Fig. 3a) y lectura (ver Fig. 3b). En el eje horizontal se muestra el número de servidores de datos disponibles, el cual aumenta desde 1 hasta un máximo de 16 servidores. Por otro lado, el eje vertical representa la tasa de transferencia medida en MB por segundos.

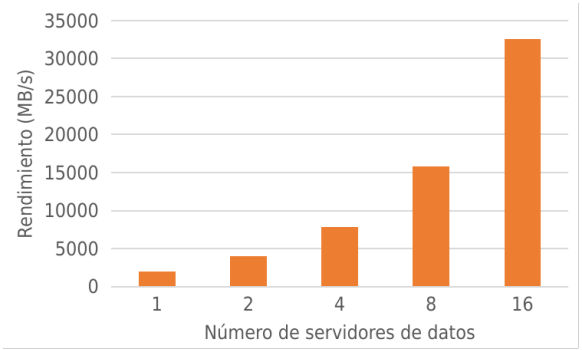
En la Fig. 3a se observa que para las operaciones de escritura, HERCULES presenta una escalabilidad exponencial, siendo la configuración con 16 servidores de datos la que ha obtenido el mejor rendimiento, alcanzando los $31,77$ GB/s, representando una aceleración del $16,80x$ con respecto a la configuración con menor rendimiento (1 servidor de datos con un rendimiento de $1,89$ GB/s).

Por otro lado, la Fig. 3b muestra el rendimiento obtenido para las operaciones de lectura de datos. En este caso, se observa un menor cambio al pasar de 8 a 16 servidores de datos (ganancia del $2,25\,\%$) en comparación con el comportamiento observado en las operaciones de escritura (ganancia del $105,72\,\%$). En contraste con la configuración más lenta (1 servidor de datos con un rendimiento de $11,04$ GB/s), la configuración con 16 servidores de datos representa una aceleración del $3,39x$ con un rendimiento del $37,44$ GB/s.

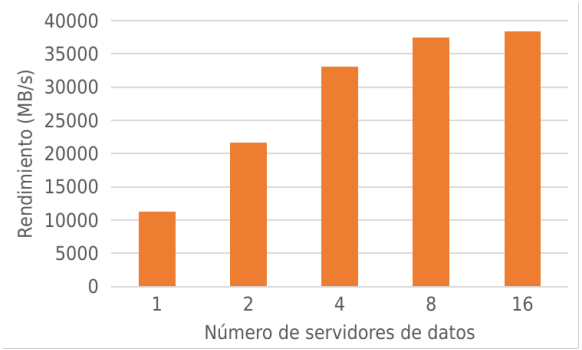
B. *Evaluación de escalabilidad fuerte*

Para este conjunto de pruebas, se ha definido un tamaño de fichero de 10 GBytes, el cual es repartido entre los 16 clientes desplegados, obteniendo así la cantidad datos que cada cliente debe escribir y leer. La Fig. 4 muestra los resultados de la evaluación de escalabilidad fuerte realizada para HERCULES, divida en operaciones de escritura (ver Fig. 4a) y lectura (ver Fig. 4b). En el eje horizontal se muestra el número de servidores de datos disponibles, el cual aumenta desde 1 hasta un máximo de 16 servidores. Por su parte, el eje vertical representa el rendimiento obtenido por prueba medido en MB por segundos.

Los resultados obtenidos en esta prueba muestran un comportamiento similar al obtenido en la escalabilidad débil. En la Fig. 4a se presentan los resultados para las operaciones de escritura, en donde considerando el peor y mejor caso ($1,87$ GB/s para 1 servidor de datos y $30,32$ GB/s para 16 servidores de datos respectivamente) se ha obtenido una aceleración del rendimiento del $16,18x$. Por otra parte, al contrastar el mismo escenario para las operaciones de lectura, la Fig. 4b muestra una aceleración del $3,21x$.



(a) Operaciones de escritura.



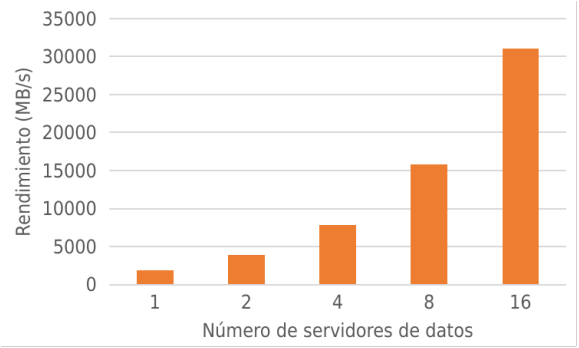
(b) Operaciones de lectura.

Fig. 3: Evaluación de la escalabilidad débil de HERCULES fijando 16 clientes y aumentando el número de servidores de datos disponibles.

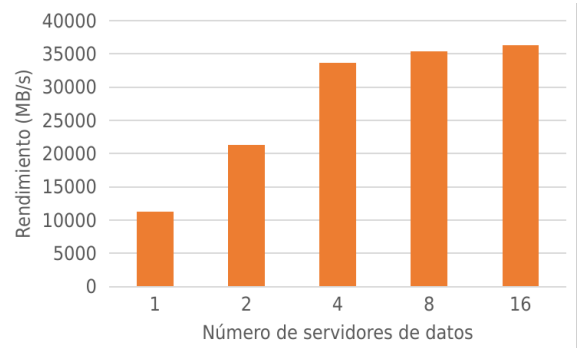
C. *Comparación del rendimiento en acceso a datos de HERCULES y BeeGFS*

En este apartado, se muestran los resultados obtenidos tras realizar la comparación de la escalabilidad fuerte entre HERCULES (solución presentada en este artículo) y BeeGFS. En este caso, se ha establecido un tamaño de fichero de 1 GB, el cual se divide de acuerdo al número de clientes que sean desplegados para cada una de las 5 pruebas (de 16 a 256 clientes). Para HERCULES, se han establecido tres configuraciones, cuya diferencia se destaca en la cantidad de servidores de datos (SD) que se han desplegado (4, 8 y 16).

La Fig. 5 muestra los resultados de la comparación previamente mencionada. En la Fig. 5a se relacionan los rendimientos obtenidos para las operaciones de escritura; para todas las configuraciones, HERCULES ha obtenido un rendimiento promedio superior en comparación con el obtenido por BeeGFS. De manera de ejemplo, la solución cuyo rendimiento se ajusta al de BeeGFS ha sido *HERCULES-4SD* (4 servidores de datos desplegados) con 7,00 GB/s, mientras que BeeGFS ha gestionado $4,95$ GB/s, lo cual representa una aceleración del $1,41x$ a favor de HERCULES.



(a) Operaciones de escritura



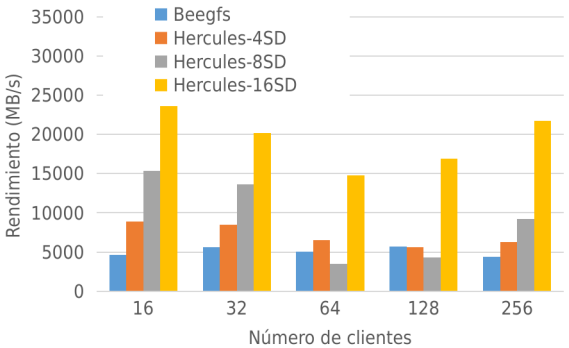
(b) Operaciones de lectura

Fig. 4: Evaluación de la escalabilidad fuerte de HERCULES fijando 16 clientes y aumentando el número de servidores de datos disponibles.

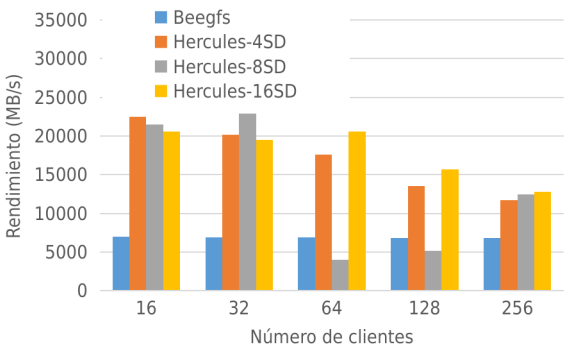
La mejor configuración ha sido $HERCULES - 16SD$ con un pico en rendimiento de 23,08 GB/s cuando se despliegan 16 clientes, mientras que cuando se despliegan 256 clientes se obtiene un rendimiento aceptable de 21,23 GB/s, saturando el ancho de banda máximo proporcionado por la red del sistema.

Un comportamiento observado para todas las configuraciones de HERCULES es la degradación de su rendimiento cuando se pasa de 16 a 32 y posteriormente 64 clientes, sin embargo, se observa que esta tendencia cambia para el caso de 128 y 256 clientes. Con base en estos resultados y como trabajo futuro, resultaría interesante estudiar el comportamiento de HERCULES al realizar variaciones en el tamaño de bloque y el tamaño de transferencia realizado, ya que para las pruebas mostradas en este apartado se ha configurado IOR para realizar una transferencia de datos igual al tamaño de bloque, siendo este valor igual a la cantidad de datos que debe escribir cada cliente. Como ejemplo de dicha configuración se muestra la Tabla I, en la cual se han colocado el número de clientes y sus correspondientes tamaños de bloque y de transferencia utilizados (representados en MB). Al multiplicar el tamaño de bloque por el número de clientes se obtiene el tamaño final del fichero escrito (1024 MB o 1 GB para todos los casos). Es importante distinguir que existen dos tamaños de bloques: el primero de ellos es el utilizado por IOR cuando realiza el intercambio de datos, y el segundo el establecido internamente en HERCULES para la gestión de los conjuntos de datos (el cual se ha establecido en 512KB para todos los casos).

En la Fig. 5b se observan los resultados para las operaciones de lectura. En términos general, BeeGFS ha obtenido un rendimiento promedio de 6,71 GB/s, mientras que HERCULES ha obtenido un rendimiento promedio de 16,70 GB/s, 12,89 GB/s y 17,39 GB/s para HERCULES-4SD, HERCULES-8SD y HERCULES-16SD respectivamente. Estos resultados muestran el efecto de la sobrecarga generada al añadir una mayor cantidad de clientes para HERCULES, sin embargo, en la prueba con 256 clientes (la de menor rendimiento) se ha logrado obtener una ganancia de rendimiento del 72,47 % con HERCULES-4SD, 83,24 % con HERCULES-8SD y 87,60 % con HERCULES-16SD respecto a BeeGFS.



(a) Operaciones de escritura.



(b) Operaciones de lectura.

Fig. 5: Evaluación de la escalabilidad fuerte de HERCULES contra BeeGFS fijando el número de servidores disponibles y escalando el número de clientes que realizan operaciones de lectura y escritura.

Tabla I: Tamaño de bloque y de transferencia utilizados para la prueba de IOR de acuerdo a la cantidad de clientes desplegados.

Número de clientes	Tamaño del bloque (MB)	Tamaño de transferencia (MB)	Tamaño del fichero (MB)
16	64	64	1024
32	32	32	1024
64	16	16	1024
128	8	8	1024
256	4	4	1024

D. *Comparación del rendimiento en acceso a metadatos de HERCULES y BeeGFS*

Como última prueba, se ha evaluado el rendimiento del acceso a metadatos haciendo uso del paquete de pruebas IO500. El temporizador *stonewall* se ha configurado en 30 y 90 segundos respectivamente. En

Tabla II: Resultados de la prueba IO500 comparando BeeGFS con HERCULES utilizando un único nodo de datos/metadatos en diferentes nodos de computación.

	BeeGFS (30 seg.)	**HERCULES (30 seg.)**	**BeeGFS (90 seg.)**	**HERCULES (90 seg.)**
find	1.056	8.120	8.088	23.538
mdtest-hard-write	31.062	34.565	92.322	73.179
mdtest-easy-stat	16.162	24.667	40.760	25.439
mdtest-hard-stat	9.860	8.332	32.482	22.165
mdtest-easy-delete	23.052	10.329	59.737	50.579
mdtest-hard-read	23.953	18.432	77.337	53.956
mdtest-hard-delete	14.648	19.887	48.321	60.104

la tabla II se muestran los resultados obtenidos comparando BeeGFS y HERCULES. HERCULES se ha desplegado de forma distribuida con servidores de metadatos y servidores de datos remotos, de tal forma que mediante esta configuración se ha pretendido emular el despliegue actual de BeeGFS.

Los resultados obtenidos indican que HERCULES alcanza un rendimiento similar. Observamos una pequeña degradación del rendimiento en el caso *find*, motivada principalmente por el uso del árbol binario *GLib* como índice. Como trabajo futuro, este componente puede sustituirse aplicando la inserción masiva de espacios de nombres para cargas de trabajo de creación intensiva y el almacenamiento en caché de metadatos coherentes sin estado en la capa de front-end [18].

VII. Conclusiones

Este artículo presentó el diseño, implementación, y evaluación de HERCULES, un sistema de archivos *ad hoc* para cargas de trabajo intensivas de datos en aplicaciones HPC. Esta solución cuenta con políticas de distribución de datos a nivel del conjunto de datos, lo cual permite trabajar con distintas estrategias para la gestión de los bloques de datos en los sistemas de almacenamiento. Además, HERCULES integra el framework UCX, el cual ofrece una abstracción genérica que ofrece la gestión de múltiples interfaces por la cual se realiza la transferencia de datos.

Los resultados preliminares de la evaluación muestran la factibilidad de nuestra solución, y se ha demostrado el aumento del rendimiento obtenido cuando se dispone de recursos para ampliar el sistema. Como trabajo futuro, se planea añadir soporte para operaciones de memoria compartida mediante el framework UCX, lo cual permitirá realizar operaciones intra-nodo y explotar la localidad de los datos.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto EuroHPC "Adaptive multi-tier intelligent data manager for Exascale" bajo la subvención 956748 - ADMIRE - H2020-JTI-EuroHPC-2019-1 y por la Agencia Española de Investigación con cargo a la subvención PCI2021-121966. Este trabajo también ha sido parcialmente financiado por el proyecto del Ministerio de Ciencia e Innovación español "New Data Intensive Computing Methods for High-End and Edge Computing Platforms (DECIDE)", ref. PID2019-107858GB-I00.

Referencias

[1] Waleed Reda, *Accelerating Distributed Storage in Heterogeneous Settings*, Ph.D. thesis, KTH Royal Institute of Technology, 2022.

[2] Fahim Chowdhury, Yue Zhu, Todd Heer, Saul Paredes, Adam Moody, Robin Goldstone, Kathryn Mohror, and Weikuan Yu, "I/O Characterization and Performance Evaluation of BeeGFS for Deep Learning," in *Proceedings of the 48th International Conference on Parallel Processing*. 2019, ICPP '19, Association for Computing Machinery.

[3] Javier Garcia-Blas, David E. Singh, and Jesus Carretero, "IMSS: In-Memory Storage System for Data Intensive Applications," in *High Performance Computing. ISC High Performance 2022 International Workshops*, Hartwig Anzt, Amanda Bienz, Piotr Luszczek, and Marc Baboulin, Eds., Cham, 2022, pp. 190–205, Springer International Publishing.

[4] Frank Schmuck and Roger Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, USA, 2002, FAST '02, p. 19–es, USENIX Association.

[5] Peter J Braam and Philip Schwan, "Lustre: The intergalactic file system," in *Ottawa Linux Symposium*, 2002, pp. 3429–3441.

[6] Yang Yang, Qiang Cao, Jie Yao, Yuanyuan Dong, and Weikang Kong, "Spmfs: A scalable persistent memory file system on optane persistent memory," in *50th International Conference on Parallel Processing*, 2021, pp. 1–10.

[7] Jian Yang, Joseph Izraelevitz, and Steven Swanson, "Orion: A distributed file system for non-volatile main memory and RDMA-capable networks," in *17th USENIX Conference on File and Storage Technologies (FAST 19)*, 2019, pp. 221–234.

[8] Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Meihui Zhang, "In-memory big data management and processing: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1920–1948, 2015.

[9] André Brinkmann, Kathryn Mohror, Weikuan Yu, Philip Carns, Toni Cortes, Scott A. Klasky, Alberto Miranda, Franz-Josef Pfreundt, Robert B. Ross, and Marc-André Vef, "Ad Hoc File Systems for High-Performance Computing," *Computer Science and Technology*, vol. 35, no. 1, pp. 4–26, 2020.

[10] M. Vef, N. Moti, T. Süß, T. Tocci, R. Nou, A. Miranda, T. Cortes, and A. Brinkmann, "GekkoFS - A Temporary Distributed File System for HPC Applications," in *2018 IEEE International Conference on Cluster Computing*, 2018, pp. 319–324.

[11] Francisco José Rodrigo Duro, Fabrizio Marozzo, Javier García Blas, Jesús Carretero Pérez, Domenico Talia, and Paolo Trunfio, "Evaluating data caching techniques in DMCF workflows using Hercules," 2015.

[12] David E. Singh and Jesus Carretero, "Combining malleability and I/O control mechanisms to enhance the execution of multiple applications," *Journal of Systems and Software*, vol. 148, pp. 21–36, 2019.

[13] Florin Isaila, Jesus Carretero, and Rob Ross, "CLARISSE: A Middleware for Data-Staging Coordination and Control on Large-Scale HPC Platforms," in *2016 16th*

IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2016, pp. 346–355.

[14] Abutalib Aghayev, Sage Weil, Michael Kuchnik, Mark Nelson, Gregory R Ganger, and George Amvrosiadis, "File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 353–369.

[15] Pavel Shamis, Manjunath Gorentla Venkata, M. Graham Lopez, Matthew B. Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman, Gilad Shainer, Richard L. Graham, Liran Liss, Yiftah Shahar, Sreeram Potluri, Davide Rossetti, Donald Becker, Duncan Poole, Christopher Lamb, Sameer Kumar, Craig Stunkel, George Bosilca, and Aurelien Bouteiller, "UCX: An Open Source Framework for HPC Network APIs and Beyond," in *IEEE 23rd Annual Symposium on High-Performance Interconnects*, 2015, pp. 40–43.

[16] Nikela Papadopoulou, Lena Oden, and Pavan Balaji, "A performance study of ucx over infiniband," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 345–354.

[17] Konstantinos Chasapis, Jean-Yves Vet, and Jean-Thomas Acquaviva, "Benchmarking Parallel File System Sensitiveness to I/O Patterns," in *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2019, pp. 427–428.

[18] Kai Ren, Qing Zheng, Swapnil Patil, and Garth Gibson, "IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 237–248.

[19] Florin Isaila, Javier Garcia-Blas, Jesus Carretero, Rob Ross, and Dries Kimpe, "Making the case for reforming the I/O software stack of extreme-scale systems," *Advances in Engineering Software*, vol. 111, pp. 26 – 31, 2017, Advances in High Performance Computing: on the path to Exascale software.

[20] Emilia Rosti, Giuseppe Serazzi, Evgenia Smirni, and Mark S Squillante, "Models of parallel applications with large computation and i/o requirements," *IEEE Transactions on Software Engineering*, vol. 28, no. 3, pp. 286–307, 2002.

Evaluación de rendimiento del sistema de ficheros paralelo Expand Ad-Hoc en MareNostrum 4

Diego Camarmas-Alonso, Félix García-Carballeira, Alejandro Calderón-Mateos y Jesús Carretero[1]

Resumen— **Durante los últimos años las aplicaciones utilizadas en el campo de la ciencia están evolucionando hacia el análisis masivo de datos a través de *workflows* debido al crecimiento de áreas como la Inteligencia Artificial y el *big data*.**

Sin embargo, el mayor cuello de botella cuando se ejecutan este tipo de aplicaciones se encuentra en las operaciones de E/S. Para tratar de solventar este problema se está desarrollando el sistema de ficheros paralelo Expand Ad-Hoc. Permite crear particiones virtuales ad-hoc para incrementar el rendimiento de E/S en entornos de supercomputación. El objetivo de este trabajo es presentar una evaluación de este sistema de ficheros sobre el supercomputador MareNostrum 4. En la evaluación de Expand Ad-Hoc llevada a cabo en MareNostrum 4, se ha podido comprobar que su rendimiento y escalabilidad es globalmente superior al del sistema de ficheros paralelo GPFS.

Palabras clave— **Sistema de ficheros paralelo y distribuido, Expand Ad-Hoc, MareNostrum 4, GPFS.**

I. Introducción

En los últimos años las aplicaciones, principalmente en el campo de la ciencia, y muy especialmente en las crecientes áreas de Inteligencia Artificial o *big data*, han evolucionado hacia el tratamiento masivo de datos a través de flujos de trabajo, también conocidos como *wokflows*. Cada vez son más aplicaciones en estas crecientes áreas, como por ejemplo aplicaciones que permiten diagnosticar enfermedades mediante el análisis de imágenes de gran resolución, previsión meteorológica, simulación de turbulencias o dinámicas moleculares, entre otras.

El grupo de investigación ARCOS está desarrollando el sistema de ficheros Expand Ad-Hoc [2], que es un sistema de ficheros distribuido y paralelo que ha sido diseñado y optimizado para adaptarse a estas nuevas necesidades.

En [1] se presentó un prototipo inicial y una evaluación de rendimiento preliminar de Expand Ad-Hoc. En este nuevo trabajo se va a presentar la versión estable de este sistema de ficheros y una evaluación de rendimiento más completa.

En la versión estable de Expand Ad-Hoc se añade la interceptación de nuevas llamadas al sistema que no habían sido interceptadas en el prototipo inicial. Además, se añaden diversas optimizaciones que ayudan a incrementar el ancho de banda de E/S y se simplifica del despliegue de los servidores ad-hoc para mejorar la experiencia de usuario.

[1]Departamento de Informática, Universidad Carlos III de Madrid (UC3M), e-mail: {dcamarma,fgcarbal,acaldero,jcarrete}@inf.uc3m.es

[2]https://github.com/xpn-arcos/xpn

En la evaluación del prototipo inicial de Expand Ad-Hoc se utilizó un cluster heterogéneo con un máximo de 4 nodos de cómputo. Para esta nueva evaluación se va a utilizar el supercomputador MareNostrum 4 con un máximo de 128 nodos de cómputo homogéneos. Lo que nos va a permitir estudiar mejor aspectos como la escalabilidad del sistema de ficheros, entre otros.

Además, para esta nueva evaluación se ha ampliado la batería de pruebas del *benchmark* IOR para incluir una evaluación cuando los ficheros son compartidos por los procesos. También, se ha utilizado el *benchmark* DLIO para evaluar las prestaciones de E/S de Expand Ad-Hoc cuanto se ejecutan aplicaciones de Inteligencia Artificial, dado que este tipo de aplicaciones son muy intensivas en el uso de datos.

El resto del documento se estructura de la siguiente forma: la Sección II muestra el Estado del arte; la Sección III describe las principales características de Expand Ad-Hoc; la Sección IV presenta los resultados de la evaluación de Expand Ad-Hoc en MareNostrum 4. Y, por último, la Sección V contiene las principales conclusiones y los trabajos futuros.

II. Estado del arte

En la actualidad existen diferentes soluciones de almacenamiento, de las cuales, las que están relacionadas con la problemática que se trata en este trabajo son los sistemas de ficheros paralelos y, particularmente, los sistemas de ficheros ad-hoc.

Un sistema de ficheros paralelo es un tipo de sistema de ficheros distribuido que almacena la información de los ficheros entre varios servidores, proporcionando acceso paralelo a estos ficheros [2]. Habitualmente estos sistemas de ficheros utilizan un conjunto de nodos de almacenamiento compartido que se corresponde con un subconjunto del total de nodos que hay en el supercomputador. Esto hace que se generen cuellos de botella en la E/S de datos en estos nodos, lo que provoca que disminuya el rendimiento global de las aplicaciones [3]. Algunos ejemplos conocidos de sistemas de ficheros paralelos son GPFS [4], Lustre [5] y BeeGFS [6].

El sistema de ficheros paralelo GPFS [4] tiene como origen el sistema de archivos Tiger Shark que era un proyecto del Centro de Investigación Almaden de IBM en 1993. Aunque su diseño inicial estaba planeado para aplicaciones multimedia de alto rendimiento, sus características resultaron ser muy adecuadas para la computación científica y, por ello,

es muy común su uso como sistema de ficheros *backend* en los supercomputadores. Una de sus principal característica es que dispone de una caché de datos a nivel de cliente que permite realizar las operaciones de E/S de forma asíncrona y para garantizar la coherencia de datos utiliza un gestor distribuido de cerrojos. Además, GPFS ofrece un acceso paralelo a los ficheros y que permite añadir o eliminar unidades de almacenamiento del clúster GPFS durante su funcionamiento, sin necesidad de realizar más acciones.

Por otro lado, está el sistema de ficheros Lustre [7] que se inició como un proyecto de investigación en la Universidad Carnegie Mellon (CMU) en el año 1999. Tras la aparición de los clúster Beowulf [8] con sistema operativo Linux como alternativa a los costosos supercomputadores, Lustre se convirtió en la solución de almacenamiento para la computación en clúster a gran escala. Además, la expansión de su uso también fue propiciado debido a que utiliza la semántica POSIX y permite accesos concurrentes de lectura y escritura en los ficheros que almacena.

BeeGFS [6] es un sistema de ficheros paralelo que empezó a desarrollarse en 2005 y presentó su primera versión beta en 2007. Está diseñado especialmente para su uso en HPC y también basado en la interfaz POSIX. Además, según los autores, está diseñado para maximizar el rendimiento y la escalabilidad. Para ello, además de dividir los ficheros en bloques y distribuir estos entre todos los nodos de almacenamiento, como hacen el resto de sistemas de ficheros paralelos, también distribuyen los metadatos para balancear la carga.

Un sistema de ficheros ad-hoc permite virtualizar dinámicamente el almacenamiento en los nodos de cómputo teniendo en cuenta los recursos disponibles en los nodos de cómputo del supercomputador y los requerimientos de la aplicación que se va a ejecutar. Esto permite acercar el almacenamiento a la aplicación a través de la localidad de los datos, reduciendo así sobrecarga en el sistema de almacenamiento *backend* del supercomputador que es distribuido y compartido entre todos nodos cómputo [9]. Algunos ejemplos conocidos de sistemas de ficheros ad-hoc son BurstFS [10], GekkoFS [11] y BeeOND [12].

BurstFS [10] es uno de los primeros sistemas de ficheros ad-hoc que se conocen. Está especialmente diseñado y optimizado para ser desplegado como sistema de ficheros local. Sin embargo, este sistema de ficheros tiene el principal inconveniente de que cada vez que se quiere utilizar se tiene que llevar a cabo el despliegue y la inicialización de los servidores de metadatos, requiere de la implementación de mecanismos de redundancia y el cumplimiento total de POSIX [13].

Otro ejemplo de sistemas de ficheros ad-hoc lo constituye GekkoFS [14]. GekkoFS se empezó a desarrollar en el año 2017 y se encuentra actualmente en desarrollo activo. Como características principales, este sistema de ficheros hace uso de la base de datos RocksDB [15] para almacenar los metadatos y la interfaz RPC Mercury [16] para las comunicaciones. Sin embargo, este sistema de ficheros ad-hoc, según describen los autores, no es POSIX completo [11], por lo que algunas llamadas de POSIX no pueden ser utilizadas sobre los ficheros que almacena, como es el caso del *rename*, entre otras. Además, este sistema de ficheros tiene otros problemas relacionados con el uso de múltiples dependencias de otros paquetes de software (RocksDB, Mercury, etc.) y versiones específicas de estos. Lo que provoca que su instalación sea más compleja y su rendimiento dependa de la sobrecarga de dichos componentes y su correcto enlazado con los *drivers* del hardware del supercomputador (por ejemplo, no poder utilizar los *drivers* de Omni-Path u otras implementaciones de InfiniBand).

Por último, está BeeOND [12] que es otro sistema de ficheros ad-hoc que permite desplegar diferentes instancias del sistema de ficheros BeeGFS [6] de forma dinámica, aprovechando las ventajas de este. Esto puede ser útil en escenarios como la nube o cuando se tiene como sistema de ficheros *backend* en el supercomputador BeeGFS. Aunque para uso personal puede ser gratuito, para su uso profesional BeeOND precisa de pago, a diferencia de Expand Ad-Hoc o GekkoFS.

III. Expand Ad-Hoc

Expand Ad-Hoc [1] es un sistema de ficheros paralelo basado en servidores ad-hoc diseñado especialmente para la ejecución de flujos de trabajo de aplicaciones intensivas en el uso de datos. A continuación, se van a describir brevemente sus características principales.

A. Diseño de Expand Ad-Hoc

El diseño de este sistema de ficheros, como se puede ver en la Figura 1, se basa en el uso de un conjunto de servidores de datos y clientes ad-hoc se comunican entre sí utilizando MPI, lo que facilita su uso en los entornos HPC.

Estos clientes y servidores ad-hoc pueden ser desplegados en los nodos de cómputo donde ejecuta la aplicación, permitiendo aprovechar la localidad de los datos, o en nodos de cómputo diferentes.

Además, cabe destacar que Expand Ad-Hoc utiliza el almacenamiento local de los nodos de cómputo (HDD, SSD, memoria compartida, etc.) lo que permite utilizar los servicios de gestión de datos proporcionados por el sistema operativo, como puede ser POSIX.

B. Distribución de datos y archivos

Expand Ad-Hoc virtualiza el almacenamiento local de los nodos de cómputo generando una o varias particiones genéricas paralelas. Para ello, cada nodo de cómputo que ejecuta un servidor ad-hoc proporciona uno o más directorios que se combinan para generar estas particiones distribuidas.

En cuanto al almacenamiento de los ficheros, como se puede ver en la Figura 2, estos, en primer lugar, son divididos en bloques del mismo tamaño, siendo

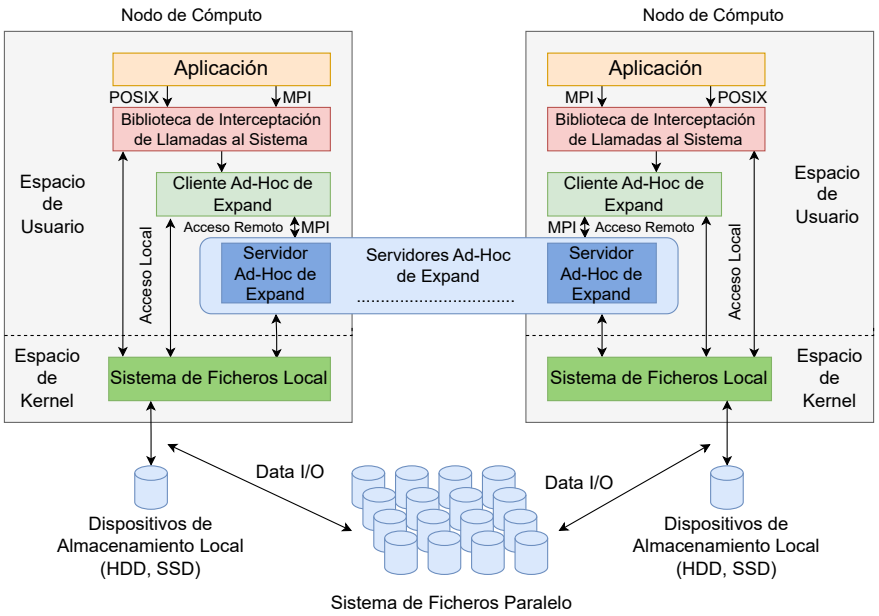


Fig. 1: Diseño de la arquitectura de XPN Ad-Hoc.

este tamaño el *blocksize* de la partición de Expand Ad-Hoc donde se van a almacenar. Después, se distribuyen dichos bloques equitativamente entre todos los servidores ad-hoc que forman la partición virtual utilizando el patrón de reparto *round-robin* (de forma análoga a un RAID 0 de discos) generando un subfichero por nodo de cómputo. Esta metodología de almacenamiento facilita el acceso paralelo a ficheros y es totalmente transparente para los usuarios de Expand Ad-Hoc.

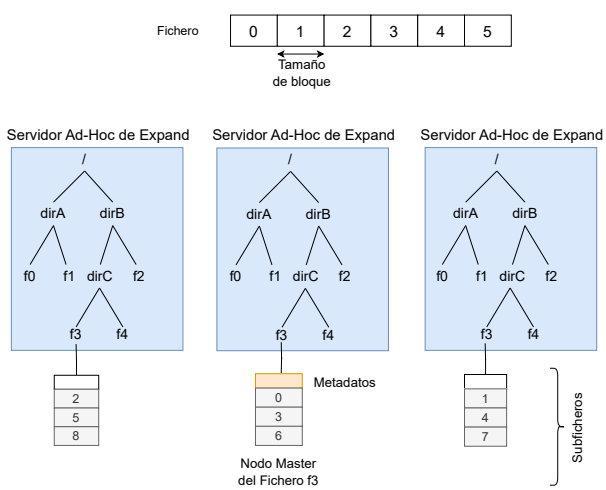


Fig. 2: Estructura de archivos y la asignación de directorios en Expand.

C. Gestión de metadatos

Respecto a la gestión de los metadatos de los ficheros, Expand Ad-Hoc no utiliza ningún tipo de gestor de metadatos, si no que cada uno de los subficheros tiene una pequeña cabecera al comienzo.

Sin embargo, a pesar de que todos los subficheros tengan este espacio reservado al comienzo para guardar metadatos, solo el subfichero almacenado en el nodo maestro de la partición contendrá esta información (ver Figura 2). Para determinar cuál es el nodo maestro entre todos los que forman parte de la partición, se convierte el nombre del fichero en un número aplicando sobre el nombre una función *hash*. De esta forma se reparte la carga de acceder al nodo maestro entre todos los nodos de almacenamiento disponibles.

D. Acceso paralelo

Expand Ad-Hoc hace uso de un manejador de fichero virtual para llevar a cabo todas las operaciones los ficheros que almacena. Esto permite optimizar las operaciones de E/S, ya que se dividen las operaciones de los usuarios en varias operaciones que se llevan a cabo de forma paralela en los servidores ad-hoc involucrados.

E. Localidad de datos

Como se ha indicado en las líneas anteriores, los servidores ad-hoc de Expand Ad-Hoc pueden ser desplegados en los mismos nodos de cómputo en los que van a ejecutar las aplicaciones del *workflow*. Esto permite aprovechar la localidad de los datos cuando los datos a los que desea acceder la aplicación se encuentran almacenados en el servidor ad-hoc que ejecuta en este mismo nodo de cómputo.

Cuando se da esta situación, Expand Ad-Hoc es capaz de detectarlo y acceder a los datos directamente, como se puede ver en la Figura 1, utilizando el sistema de ficheros local, sin tener que comunicarse con el servidor de datos. Lo que permite optimizar el acceso a los datos, ya que se evita la latencia y sobrecarga que se producen con los accesos remotos a datos.

F. Biblioteca de interceptación de llamadas al sistema para las aplicaciones POSIX ya existentes

Por último, Expand Ad-Hoc ofrece una biblioteca de interceptación de llamadas al sistema que permite utilizar este sistema de ficheros sin tener que modificar el código fuente de las aplicaciones ya existentes, que en muchas ocasiones no es posible.

Esta biblioteca intercepta la mayoría de las llamadas al sistema POSIX que se realizan a lo largo de la ejecución de una aplicación. Cuando se intercepta una llamada al sistema, la biblioteca, en primer lugar, comprueba que el fichero con el que se trabaja esté almacenado en Expand Ad-Hoc, en cuyo caso se llamará a la función correspondiente de la API de este sistema de ficheros. Sin embargo, si el fichero no se encuentra almacenado en Expand Ad-Hoc se ejecutará la llamada al sistema de la `libc.so` correspondiente.

Para hacer uso de esta biblioteca de interceptación se utiliza `LD_PRELOAD` que permite cargar esta biblioteca antes que el resto, incluida la `libc.so`, sin necesidad de permisos de superusuario.

IV. Evaluación

Con la evaluación presentada en esta Sección, por un lado, se pretende medir el rendimiento ofrecido por Expand Ad-Hoc cuando se realizan escrituras y lecturas de fichero por una aplicación paralela, y también el ancho de banda de E/S obtenido durante la ejecución de un entrenamiento de *Deep Learning*. Por otro lado, en esta Sección se compara el rendimiento de Expand Ad-Hoc respecto a GPFS, que es un sistema ficheros paralelo que habitualmente se utiliza en los supercomputadores como sistema de ficheros *backend*.

Para esta evaluación se ha decidido utilizar, en primer lugar, el *benchmark* de código abierto IOR [3]. Este *benchmark* es muy utilizado para la evaluación de los sistemas de ficheros paralelos y distribuidos ya que permite simular diferentes cargas de E/S (operaciones de lectura/escritura, acceso a fichero compartido/individual, etc.) y obtener el ancho de banda ofrecido por el sistema de ficheros.

Además, también se ha utilizado el *benchmark* desarrollado por Argonne Leadership Computing Facility DLIO [4] [17], que permite medir el rendimiento de un sistema de ficheros a través de la emulación del comportamiento de E/S de las aplicaciones científicas de *Deep Learning*.

Para llevar a cabo esta evaluación se ha utilizado el supercomputador MareNostrum 4 [18], que dispone de 3456 nodos de cómputo Lenovo ThinkSystem SD530 distribuidos en 48 racks. Concretamente cada nodo dispone de:

- Dos procesadores Intel Xeon Pentium 8160 24C con 24 procesadores de 2.1 GHz.
- Un Intel SSD DC S3520 Series con 240 GiB de almacenamiento disponible.
- 96 GiB (2 GiB per core) de memoria principal.
- Una red de 100 Gb Intel Omni-Path Full-Fat Tree.

Por lo que, en total este supercomputador dispone de 165888 procesadores y 384.75 TiB de memoria principal.

Para la evaluación realizada en este trabajo, se ha usado la configuración por defecto recomendada en cada uno de los sistemas de ficheros. En el caso del almacenamiento:

- Expand Ad-Hoc está configurado con un tamaño de bloque de 512 KiB, lo que permite comparar ambos sistemas de ficheros en las mismas condiciones.
- Expand Ad-Hoc usa el dispositivo SSD y la memoria compartida (SHM) disponible en cada uno de los nodos como almacenamiento subyacente. Para la memoria compartida (SHM) en ambos casos se usa el directorio `/dev/shm` como sistema de ficheros.

En el caso de comunicación en red:

- Expand Ad-Hoc usa la configuración por defecto de MPI en el supercomputador (sin ningún parámetro adicional).

A. Evaluación con IOR

Para evaluar Expand Ad-Hoc con el *benchmark* IOR, se han ejecutado diferentes pruebas de rendimiento para obtener el ancho de banda de escritura y lectura en accesos paralelos a un fichero. Además, las pruebas de rendimiento realizadas en este trabajo se basan en las realizadas en [19] para la evaluación de GekkoFS.

Las configuraciones que se han usado para las evaluaciones con IOR son las siguientes:

- Nodos de cómputo: 1, 2, 4, ... hasta un total de 128 nodos.
- Almacenamiento local: SSD y memoria compartida (SHM).
- Tamaño de transferencia usado en IOR: 64 KiB, 512 KiB y 1 MiB.
- Procesos cliente por nodo de computación: 8.
- Operaciones: lectura y escritura en paralelo sobre un fichero compartido y un fichero por proceso.
- Tamaño escrito por cada proceso cliente: 4 GiB (lo que supone un total de 4 TiB de fichero en la configuración máxima).
- Sistemas de ficheros: GPFS 4.2.2.0 y Expand Ad-Hoc 2.2.2.

Por último, cabe destacar que para todas las pruebas realizadas se ha utilizado un servidor Expand Ad-Hoc por nodo de cómputo (de 1 hasta 128). Y cada una de las pruebas se han ejecutado como trabajo individual con los nodos de cómputo en exclusividad mediante el sistema de colas SLURM (*Simple Linux Utility for Resource Management*) [20].

Las Figuras 3 y 4 muestran los resultados obtenidos en la evaluación llevada a cabo utilizando IOR

[3] https://github.com/hpc/ior
[4] https://github.com/argonne-lcf/dlio_benchmark

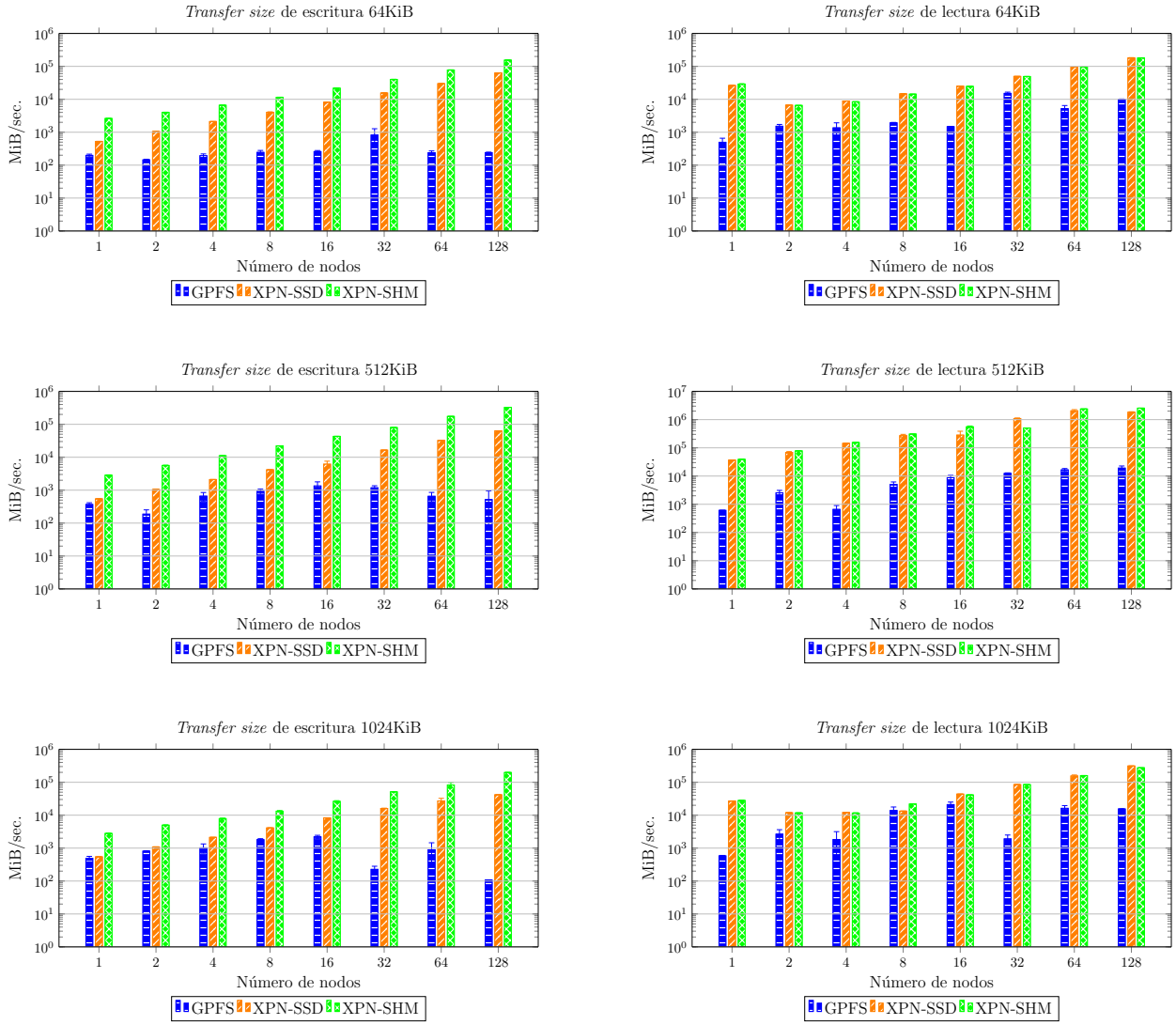


Fig. 3: GPFS vs. Expand Ad-Hoc. Ancho de banda (MiB/seg.) escribiendo y leyendo con diferentes tamaños de transferencia (64KiB, 512KiB y 1MiB), nodos de cómputo (1, 2, 4, 8, 16, 32, 64 y 128), con 8 procesos cliente por nodo y **fichero compartido**. Resultados en escala logarítmica.

con un fichero compartido y un fichero por proceso, respectivamente. Concretamente, se muestran el ancho de banda (MiB/segundo en escala logarítmica) cuando se escribe y leen datos usando como tamaños de transferencia (*transfer sizes*) 64 KiB, 512 KiB y 1024 KiB en GPFS, así como Expand Ad-Hoc con SSD (XPN Ad-Hoc - SSD) y con memoria compartida (XPN Ad-Hoc - SHM).

En los resultados de rendimiento obtenidos con el *benchmark* IOR se puede observar que cuando se utiliza un fichero compartido (ver Figura 3) entre todos los procesos, Expand Ad-Hoc usando un disco SSD obtiene mucho mejor rendimiento de escritura y lectura que GPFS. En el caso de utilizar memoria compartida como almacenamiento local en Expand Ad-Hoc, se obtienen los mejores resultados.

En cuanto a los resultados obtenidos cuando se utilizan ficheros individuales por proceso (ver Figura 4), el sistema de ficheros GPFS ofrece en escrituras un ancho de banda mayor que los sistemas de ficheros ad-hoc. Este comportamiento se debe a que, como se ha comentado anteriormente, GPFS dispone de una caché de datos para realizar de forma asíncrona las operaciones de E/S, mientras que en el caso de Expand Ad-Hoc las operaciones se realizan directamente en disco. Además, al tratarse de ficheros individuales (sin que exista concurrencia) se evita la sobrecarga del servicio de cerrojos distribuido. No obstante, cuando el número de nodos de cómputo incrementa el ancho de banda de ambos sistemas de ficheros la diferencia disminuye. Esto se debe a que Expand Ad-Hoc también incrementan el número de servidores de almacenamiento, permitiendo una mejor escalabilidad. Por otro lado, en el caso de las lecturas Expand Ad-Hoc obtiene mejores anchos de banda que el resto en la gran mayoría de los casos.

Como resumen de la evaluación llevada a cabo con el *benchmmark* IOR, se puede afirmar que Expand Ad-Hoc globalmente obtiene mejores resultados que GPFS ya que al utilizarse el almacenamiento local de los nodos permite escalar el número de servidores de almacenamiento y, además, aprovechar la localidad de los datos. Todo esto se consigue gracias al uso de MPI, que permite aprovechar todo el rendimiento de la red, y de dejar que las aplicaciones sean las encargadas de aplicar un protocolo de coherencia si lo

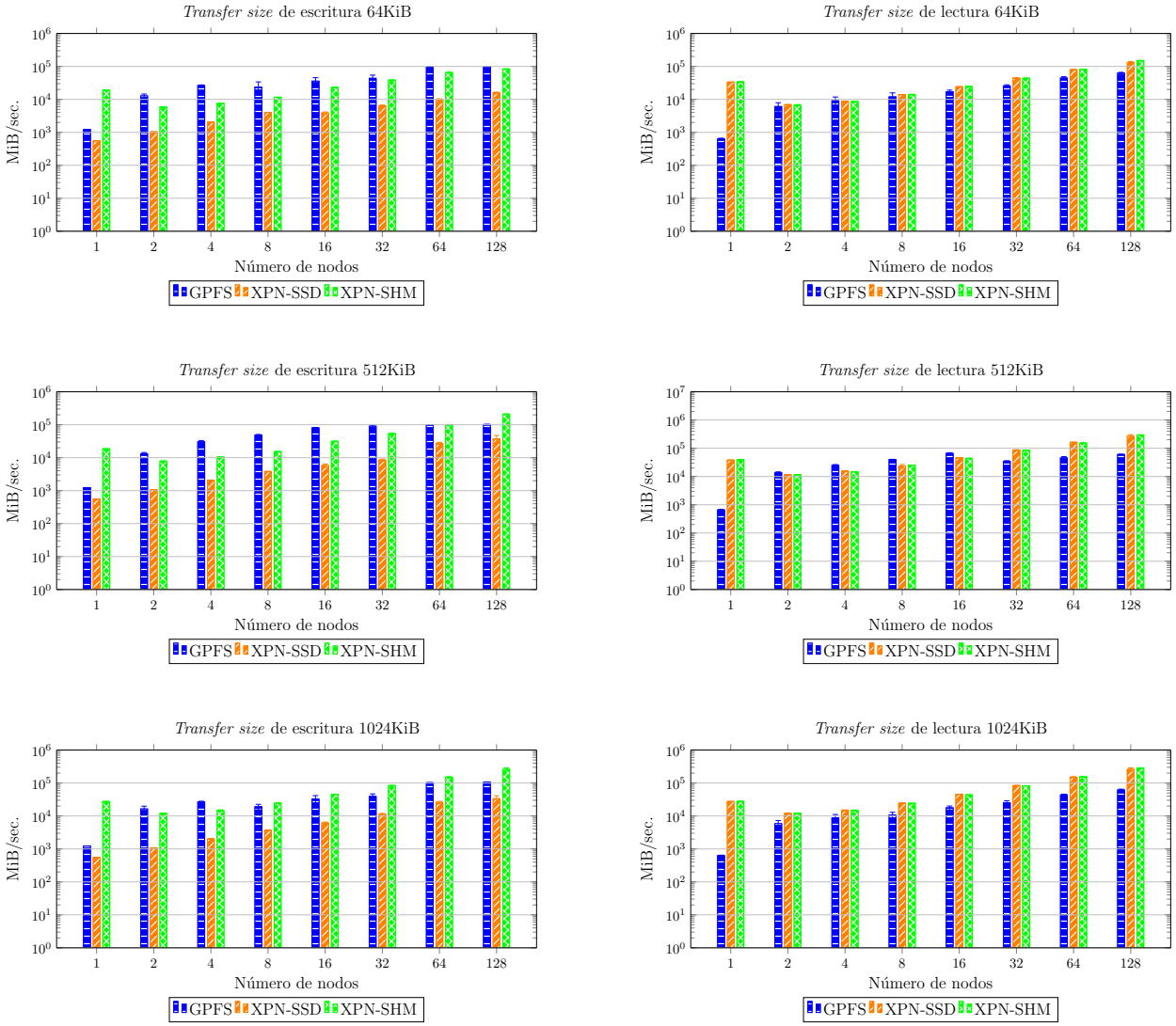


Fig. 4: GPFS vs. Expand Ad-Hoc. Ancho de banda (MiB/seg.) escribiendo y leyendo con diferentes tamaños de transferencia (64KiB, 512KiB y 1MiB), nodos de cómputo (1, 2, 4, 8, 16, 32, 64 y 128), con 8 procesos cliente por nodo y un **fichero individual por proceso**. Resultados en escala logarítmica.

necesitan para evitar esta sobrecarga cuando no sea necesario. Por ejemplo, al usar ficheros compartidos en los que cada proceso escribe en una parte distinta del fichero, es decir, ningún bloque del fichero es modificado por más de un proceso.

B. Evaluación con DLIO

Por otro lado, se ha querido evaluar el rendimiento que ofrece Expand Ad-Hoc cuando se utiliza este con aplicaciones de *Deep Learning*. Para ello, como se mencionó anteriormente, se ha utilizado el *benchmark* DLIO con diferentes cargas de trabajo (*workloads*).

Las configuraciones que se han utilizado para estas evaluaciones son:

- Nodos de cómputo: 16, 32 y 64 nodos.
- Almacenamiento local: SSD y memoria compartida (SHM).
- Cargas de trabajo (con su configuración por defecto):
  - CosmoFlow: red CNN 3D para estudiar el universo. Tamaño del *dataset*: 65 GiB. Número de *epoch*: 4.
  - ResNet50: clasificación de imágenes 3D. Tamaño del *dataset*: 147 GiB. Número de *epoch*: 1.
  - UNET3D: segmentación de imágenes médicas 3D. Tamaño del *dataset*: 36 GiB. Número de *epoch*: 10.
- Sistemas de ficheros: GPFS 4.2.2.0 y Expand Ad-Hoc 2.2.2.

Al igual que en la evaluación llevada a cabo con el *benchmark* IOR, se ha utilizado un servidor Expand Ad-Hoc por cada nodo de cómputo y, también, se han utilizado estos nodos de cómputo en exclusividad utilizando el sistema de colas SLURM.

En la Figura 5 se puede ver el ancho de banda (MiB/segundo) de E/S obtenido durante el entrenamiento realizado por DLIO para cada una de las cargas de trabajo sobre GPFS y Expand Ad-Hoc utilizando como almacenamiento local SSD (XPN Ad-Hoc - SSD) y memoria compartida (XPN Ad-Hoc - SHM).

En los resultados, se puede ver que el ancho de

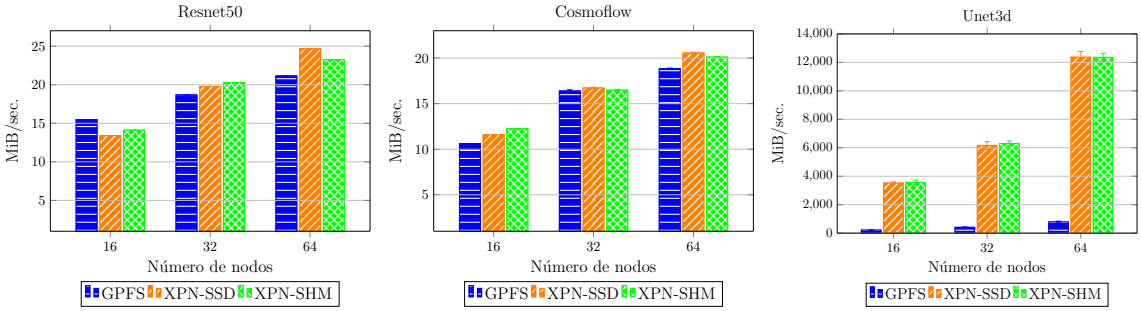


Fig. 5: GPFS vs. Expand Ad-Hoc. Ancho de banda (MiB/seg.) del entrenamiento realizado por DLIO y 16, 32 y 64 nodos de cómputo.

banda obtenido por el sistema de ficheros Expand Ad-Hoc es ligeramente mayor, de forma general, que el ofrecido por GPFS cuando se ejecutan las cargas de trabajo CosmoFlow y ResNet50, mientras que es mucho mayor cuando se ejecuta el *workload* UNET3D.

Este comportamiento se debe a que Expand Ad-Hoc aprovecha la localidad de los datos, ya que en el caso de UNET3D se realizan 10 *epoch* sobre el mismo *dataset*, mientras que en el caso de CosmoFlow y ResNet50 se realizan 4 y 1 *epoch*, respectivamente. Es decir, Expand Ad-Hoc está diseñado para utilizar el almacenamiento local de los nodos de cómputo para guardar los datos que van a necesitar estos nodos de cómputo durante la ejecución de una aplicación. Esto evita tener que obtener los datos del sistema de almacenamiento *backend*, procedimiento que, de forma general, es más costoso ya que este sistema de ficheros es compartido por todos los nodos de cómputo y son accesos remotos. Por lo tanto, cuanto mayor es el número de *epochs* realizados, mayor es el número de accesos evitados al sistema de almacenamiento *backend* ya que se realizan de forma local en los propios nodos, de ahí a la mejora de rendimiento obtenida en UNET3D respecto a GPFS.

V. Conclusiones

En este artículo se ha presentado el sistema de ficheros paralelo Expand Ad-Hoc para entornos HPC y su evaluación sobre el supercomputador MareNostrum 4 utilizando los *benchmarks* IOR y DLIO.

Como se ha podido ver en las evaluaciones presentadas en este trabajo, el diseño de Expand Ad-Hoc utilizando servidores de datos basados en MPI y el aprovechamiento de la localidad permite obtener una mayor escalabilidad y un mayor ancho de banda cuando se realizan operaciones de E/S sobre uno o más ficheros de forma simultánea respecto a un sistema de ficheros paralelo utilizado como *backend*, como es el caso de GPFS.

Como principales líneas de trabajos futuros se tiene el diseño de soporte de maleabilidad, así como su evaluación de rendimiento. Además, también se quiere dotar a Expand Ad-Hoc de tolerancia a fallos basada en replicación. También, se quiere probar Expand Ad-Hoc con el *benchmark* DLIO incrementando el número de *epochs* en las diferentes cargas de trabajo, así como probar Expand Ad-Hoc con aplicaciones reales.

Agradecimientos

Este trabajo ha sido parcialmente apoyado por el proyecto del Ministerio de Ciencia e Inoovación español “Expand: High Performance Storage System for HPC and Big Data Environments”. Ref. TED2021-131798B-I00.

También parcialmente financiado por el proyecto *Adaptive multi-tier intelligent data manager for Exascale (ADMIRE)* del programa *European Union’s Horizon 2020 JTI-EuroHPC research and innovation programme*. Ref. H2020-JTI-EuroHPC-2019-1, Project no. 956748.

Y parcialmente apoyado por el proyecto EuroHPC “Adaptive multi-tier intelligent data manager for Exascale” bajo la subvención 956748 - ADMIRE - H2020-JTI-EuroHPC-2019-1 y por la Agencia Española de Investigación bajo la subvención PCI2021-121966.

Los autores agradecen los recursos informáticos de MareNostrum 4 y el apoyo técnico prestado por el Barcelona Supercomputing Center (IM-2023-1-0012).

Referencias

[1] Diego Camarmas-Alonso, Felix Garcia-Carballeira, Alejandro Calderon Mateos, and Jesus Carretero Perez, “Sistema de almacenamiento Ad-Hoc para entornos HPC basado en el sistema de ficheros paralelo Expand,” in *XXXII Jornadas de Paralelismo (JP21/22)*. July 2022, Disponible en Zenodo.

[2] Viacheslav Dubeyko, “Comparative analysis of distributed and parallel file systems’ internal techniques,” 2019.

[3] Bing Xie, Sarp Oral, Christopher Zimmer, Jong Youl Choi, David Dillow, Scott Klasky, Jay Lofstead, Norbert Podhorszki, and Jeffrey S. Chase, “Characterizing output bottlenecks of a production supercomputer: Analysis and implications,” *ACM Trans. Storage*, vol. 15, no. 4, jan 2020.

[4] Frank Schmuck and Roger Haskin, “GPFS: A Shared-Disk file system for large computing clusters,” in *Conference on File and Storage Technologies (FAST 02)*, Monterey, CA, Jan. 2002, USENIX Association.

[5] Jaehyun Han, Deoksang Kim, and Hyeonsang Eom, “Improving the performance of lustre file system in hpc environments,” in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, 2016, pp. 84–89.

[6] Frank Herold, Sven Breuner, and Jan Heichler, “An introduction to beegfs,” *ThinkParQ, Kaiserslautern, Germany, Tech. Rep*, 2014.

[7] Peter Braam, “The lustre storage architecture,” *CoRR*, vol. abs/1903.01955, 2019.

[8] Daniel A. Reed, “Beowulf clusters: From research curiosity to exascale,” in *Proceedings of the 20 Years of*

Beowulf Workshop on Honor of Thomas Sterling's 65th Birthday, New York, NY, USA, 2014, Beowulf '14, p. 28–33, Association for Computing Machinery.

[9] André Brinkmann, Kathryn Mohror, Weikuan Yu, Philip Carns, Toni Cortes, Scott A Klasky, Alberto Miranda, Franz-Josef Pfreundt, Robert B Ross, and Marc-André Vef, "Ad hoc file systems for high-performance computing," *Journal of Computer Science and Technology*, vol. 35, no. 1, pp. 4–26, 2020.

[10] Teng Wang, Kathryn Mohror, Adam Moody, Kento Sato, and Weikuan Yu, "An ephemeral burst-buffer file system for scientific applications," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 807–818.

[11] Marc-André Vef, Nafiseh Moti, Tim Süß, Markus Tacke, Tommaso Tocci, Ramon Nou, Alberto Miranda, Toni Cortes, and André Brinkmann, "Gekkofs: A temporary burst buffer file system for hpc applications," *Journal of Computer Science and Technology*, vol. 35, no. 1, pp. 72–91, 2020.

[12] Zoya Masih, "On demand file systems with beegfs," 2023.

[13] IEEE-SA Standards Board, "Ieee standard for information technology-portable operating system interface (posix)-part 1: System application program interface (api)- amendment d: Additional real time extensions [c language]," Tech. Rep., IEEE, 1999.

[14] Marc-André Vef, Nafiseh Moti, Tim Süß, Tommaso Tocci, Ramon Nou, Alberto Miranda, Toni Cortes, and André Brinkmann, "Gekkofs-a temporary distributed file system for hpc applications," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2018, pp. 319–324.

[15] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm, "Rocksdb: Evolution of development priorities in a key-value store serving large-scale applications," *ACM Trans. Storage*, vol. 17, no. 4, oct 2021.

[16] Jerome Soumagne, Dries Kimpe, Judicael Zounmevo, Mohamad Chaarawi, Quincey Koziol, Ahmad Afsahi, and Robert Ross, "Mercury: Enabling remote procedure call for high-performance computing," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2013, pp. 1–8.

[17] H. Devarajan, H. Zheng, A. Kougkas, X.-H. Sun, and V. Vishwanath, "Dlio: A data-centric benchmark for scientific deep learning applications," *21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, , no. 81–91, 2021.

[18] BSC, "MareNostrum specification," 2023, Accessed Jan. 2, 2023. [Online].

[19] Marc-Andre Vef, Nafiseh Moti, Tim Süß, Markus Tacke, Tommaso Tocci, Ramon Nou, Alberto Miranda, Toni Cortes, and André Brinkmann, "GekkoFS - A temporary burst buffer file system for HPC applications," *J. Comput. Sci. Technol.*, vol. 35, no. 1, pp. 72–91, 2020.

[20] SLURM, "Slurm workload manager," 2023, Accessed Jan. 2, 2023. [Online].

Diseño de un controlador de memoria principal híbrido para aplicaciones de Big Data

Manel Lurbe [1], Miguel A. Avargues [1], Salvador Petit [1],
Maria E. Gómez [1] y Julio Sahuquillo [1]

Resumen— **La tecnología de memoria DRAM ha sido la dominante en las implementaciones del subsistema de memoria. En los últimos años, e impulsada principalmente por las enormes demandas de memoria de las aplicaciones de big data, la tecnología NVRAM ha emergido como una tecnología de memoria más densa, permitiendo el diseño de nuevas jerarquías de memoria híbridas DRAM/NVRAM que combinan múltiples tecnologías de memoria para equilibrar capacidad, latencia, coste y persistencia.**

En el diseño de jerarquías de memoria híbridas se están aplicando dos enfoques principales: el enfoque de espacio de direcciones híbrido DRAM/NVRAM, que confía en el programador o el sistema operativo para elegir la tecnología de memoria en la que debe almacenarse cada página de memoria; y el enfoque de espacio de direcciones (único) NVRAM, en el que se necesita una tecnología más rápida (por ej., DRAM convencional) que actúe como caché de la NVRAM para aumentar las prestaciones. Este enfoque presenta retos arquitectónicos como la organización de metadatos.

En contraste con los enfoques existentes, este trabajo propone un controlador de memoria que aprovecha tecnologías de memoria recientes como eDRAM y MRAM para mejorar las prestaciones de la lenta memoria NVRAM cuando actúa como memoria principal. La propuesta consiste en una jerarquía de caché de dos niveles en el controlador de memoria: una caché de sectores SRAM y una caché (x)RAM. La caché (x)RAM, mucho más densa, ayuda a reducir significativamente el número de accesos a la NVRAM. Los resultados experimentales demuestran que la implementación de la caché (x)RAM con eDRAM o MRAM es el método más eficaz. Además, la eDRAM es capaz de mejorar la penalización en caso de fallo de la caché SRAM hasta en un 50 % y un 80 %, y las prestaciones globales del sistema en un 15 % y un 23 %.

Palabras clave— **gem5, NVMain, Memoria principal, DRAM, NVRAM**

I. Introducción

La jerarquía de memoria [1] ha desempeñado un papel clave en las prestaciones en los sistemas de cómputo desde los primeros procesadores, los cuales sólo implementaban un nivel de memoria principal, y memoria secundaria como cintas y discos. A finales de los años 70, se introdujeron las memorias caché [2] para reducir el tiempo medio de acceso a la memoria principal. Las memorias caché reducen este tiempo gracias a la explotación de las localidades, espacial y temporal, que presentan los datos [3, 4]. La localidad temporal se refiere a que los mismos datos son accedidos de nuevo en un tiempo cercano dentro de ejecución de un programa, y la localidad espacial se refiere a que los datos situados en direcciones de memoria cercanas tienen alta probabilidad de ser accedidos. A medida que ha ido aumentando la distancia entre la velocidad del procesador y la memoria principal, se han introducido varios niveles de memorias caché, lo que se conoce como jerarquía de caché. Los procesadores actuales suelen implementar tres niveles de caché en el chip.

Hoy en día, las tecnologías SRAM y DRAM son las tecnologías de memoria dominantes. La SRAM es más rápida y se utiliza habitualmente en los niveles superiores de la jerarquía para implementar cachés dentro del chip. En cambio, la tecnología DRAM, mucho más densa pero lenta, se utiliza para implementar la memoria principal, donde se requiere más capacidad, ya que esta tecnología presenta un coste por GB reducido en comparación con el coste de la SRAM. Las prestaciones de la memoria principal tienen un gran impacto en las prestaciones del procesador, ya que una de las funciones clave de esta memoria es contener la mayor parte de los datos que utilizan las aplicaciones en ejecución. Si los datos requeridos no se encuentran en memoria principal, es necesario acceder al almacenamiento secundario, varios órdenes de magnitud más lento, lo que se traduce en importantes penalizaciones en las prestaciones globales del sistema.

En el pasado, se han publicado muchos trabajos para hacer frente al conocido muro de memoria [5,6]. Sin embargo, en la última década han aparecido dos tendencias principales que han exigido revisar el diseño de la memoria principal. Por un lado, el hecho de que aumente el número de núcleos en los procesadores multinúcleo implica que más accesos a memoria compiten por el acceso a la memoria principal; por tanto, es necesario aumentar tanto la capacidad como el ancho de banda para permitir un mayor paralelismo a nivel de memoria (MLP) y servir más peticiones en paralelo. En otras palabras, el muro de la memoria sigue creciendo, especialmente en las cargas de trabajo de big data con grandes demandas de memoria, cuyas necesidades ya superan varios TB en las aplicaciones actuales [7], lo que excede la capacidad memoria DRAM soportada por los procesadores convencionales existentes.

Para hacer frente a estas tendencias, el subsistema de memoria DRAM ha mejorado significativamente en términos de capacidad de almacenamiento y ancho de banda de memoria con cada nueva generación. Desde 1985, la memoria DRAM se ha multiplicado por dos (en Mbits/chip) cada año y medio. Sin embargo, esta tendencia se ha ralentizado desde principios de este siglo y se ha convertido en un reto para una tecnología de nodo inferior a 10nm, ya que la

[1]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, contacto: malursem@gap.upv.es

implementación se vuelve poco fiable [8,9]. Además, los procesadores de servidor actuales suelen implementar varios controladores de memoria multicanal para permitir el soporte de MLP más elevados.

En consecuencia, las condiciones han cambiado, ya que en los últimos años han surgido nuevas tecnologías de memoria como la NVRAM (RAM no volátil) [10], ambas más densas y eficientes energéticamente que la DRAM. El objetivo principal es utilizarlas para implementar memorias principales mucho más grandes [11]. Sin embargo, presentan un importante inconveniente, ya que son más lentas y presentan una durabilidad finita ante las operaciones de escrituras [12–15]. Dado que cada tecnología presenta sus ventajas e inconvenientes, los arquitectos de computadores pueden centrarse en el diseño de nuevas jerarquías de memoria principal fuera del chip utilizando múltiples tecnologías, lo que se conoce como memorias principales híbridas. El diseño híbrido persigue elegir *lo mejor de cada mundo*; por ejemplo, usar DRAM para velocidad y NVRAM para mayor capacidad de almacenamiento. Existe un espacio de diseño abierto en el que los arquitectos informáticos desempeñan un papel clave mezclando distintas tecnologías de memoria para equilibrar: capacidad, latencia, coste y durabilidad.

En resumen, las enormes demandas de capacidad de memoria principal, algunas superiores a 1 TB, abren una era para que los arquitectos informáticos alcancen objetivos elementales como mejor escalabilidad, durabilidad y menor consumo energético. Para ello, este artículo se centra en un novedoso diseño de memoria principal que combina múltiples tecnologías.

A diferencia de los enfoques existentes, la solución propuesta consiste en implementar una jerarquía de caché de dos niveles combinando distintas tecnologías en el controlador de memoria. De esta forma se ataca el cuello de botella en las prestaciones que supone la alta latencia de acceso de las memorias NVRAM, mitigando la congestión y mejorando las prestaciones.

El resto del documento está organizado como sigue. La sección II presenta algunos trabajos relacionados. La sección III describe la propuesta del presente trabajo. La sección IV presenta los simuladores empleados en el presente trabajo. Además, describe los componentes que se modelan en cada simulador y sus parámetros de diseño. La sección V describe las cargas de trabajo empleadas en el estudio de la implementación y sus parámetros de lanzamiento. La sección VI presenta los resultados experimentales y, finalmente, la sección VII expone las conclusiones finales.

II. Estado Del Arte

Las soluciones existentes se basan en sistemas de memoria híbridos construidos con tecnologías DRAM y NVRAM, y pueden agruparse según la forma en que esté organizado el espacio de direcciones de la memoria principal: espacio de direcciones híbrido o espacio de direcciones NVRAM.

Espacio de direcciones híbrido. En estas soluciones la NVRAM y la DRAM comparten el mismo espacio de direcciones, es decir, cada una de ellas almacena un subconjunto diferente de las páginas de memoria física. Este enfoque es el seguido en algunos trabajos [16–24], sin embargo, requiere modificaciones tanto en el controlador de memoria como en el sistema operativo. Las políticas de gestión de este subsistema de memoria híbrido pueden afectar significativamente las prestaciones del sistema. Por lo tanto, se requiere un esfuerzo significativo para optimizar cuidadosamente dichas políticas. El trabajo en [16] propone combinar PRAM (phase change random access memory) y una solución híbrida hardware-software de baja sobrecarga para gestionar la memoria híbrida realizando el intercambio/migración de páginas.

Panthera [17] propone combinar NVRAM y DRAM en el contexto del procesamiento de big data y una técnica de gestión de memoria para la colocación y migración eficiente de datos. Los autores de [18] proponen un modelo analítico basado en Markov para estimar las prestaciones y la vida útil de las memorias híbridas DRAM-NVRAM en diversas cargas de trabajo. El modelo propuesto calcula los porcentajes de éxito y la vida útil de varias configuraciones de memoria principal híbrida. Este modelo analítico puede ayudar a los diseñadores a ajustar las configuraciones de memoria híbrida para mejorar las prestaciones y/o la vida útil. El trabajo en [25] propone la gestión de memoria híbrida basada en la utilidad (UH-MEM), un nuevo mecanismo de gestión de páginas para varias memorias híbridas, que estima sistemáticamente la utilidad (es decir, el beneficio en las prestaciones del sistema) de migrar una página entre diferentes tipos de memoria, y utiliza esta información para guiar la colocación de datos. Los autores de [19] proponen un diseño de memoria híbrida junto con una política de colocación de páginas basada en hardware. La política se basa en el controlador de memoria para supervisar los patrones de acceso, migrar páginas entre DRAM y NVRAM, y traducir las direcciones de memoria procedentes de los núcleos. Periódicamente, el sistema operativo actualiza sus asignaciones de páginas basándose en la información de traducción utilizada por el controlador de memoria.

Espacio de direcciones NVRAM. Las soluciones de este enfoque organizan la memoria principal híbrida de forma jerárquica, donde la NVRAM ocupa el nivel inferior, por lo que la NVRAM tiene todo el espacio de direcciones de la memoria principal y la DRAM actúa como caché de la NVRAM [26, 27]. El trabajo en [26] propone una nueva política de almacenamiento en caché que mejora las prestaciones de la memoria híbrida y la eficiencia energética. La política realiza un seguimiento de los fallos de búfer de fila de las filas utilizadas recientemente en NVRAM, y almacena en una caché DRAM las filas que se prevé que incurran en frecuentes fallos de búfer de fila. Este trabajo utiliza memoria DRAM convencional y al-

macena en caché DRAM filas enteras de bancos de memoria NVRAM. La propuesta en [27] introduce la gestión eficiente de los metadatos (por ejemplo, etiquetas) para los datos almacenados en caché en DRAM. Este artículo utiliza un pequeño búfer para almacenar en caché los metadatos de las filas a las que se ha accedido recientemente. La propuesta también se adapta dinámicamente para elegir la mejor granularidad de datos.

El enfoque del *espacio de direcciones NVRAM* es más práctico que el *híbrido*, ya que no requiere cambios en el software. Por lo tanto, nuestra propuesta sólo se centra en este enfoque. A diferencia de las propuestas previas, como [26, 27], nuestra propuesta implementa una caché mucho más pequeña que puede integrarse dentro del mismo procesador, suponiendo que la tecnología de caché seleccionada sea compatible con CMOS. Además, no requiere de un manejo especial de los metadatos y de implementar una caché más pequeña con tecnologías alternativas.

III. Propuesta

En esta sección se presenta la propuesta del presente trabajo para abordar los requisitos de memoria de las aplicaciones de big data y hacer frente a los principales retos comentados que impone un controlador de memoria híbrido.

Los principales componentes del controlador de memoria propuesto son los controladores de la NVRAM, DRAM y una memoria caché integrada. La NVRAM [11, 28] es más densa y lenta que la DRAM, y se utilizará como espacio de direcciones de la memoria principal. La jerarquía de caché en chip se amplía fuera del procesador a través del controlador de memoria. Este controlador contiene múltiples controladores para cada tipo de medio, por ejemplo, para DRAM y NVRAM. Mantenemos la DRAM o memoria DRAM convencional, pero se utilizará como caché de la NVRAM. Además, se introduce una caché de sectores más pequeña pero más rápida, construida con tecnología SRAM, entre las cachés del procesador y los soportes de memoria, para ayudar a reducir la latencia. La caché dentro del controlador se implementa como una caché de sectores. De este modo, se amortizan las largas latencias a las tecnologías de memoria más lentas, ya que un sector consta de múltiples bloques de caché (4 en nuestro diseño).

El diseño propuesto en este documento se basa en dos observaciones importantes que encontramos durante el proceso de implementación: i) la caché DRAM es accedida con frecuencia, y ii) un tamaño relativamente pequeño (en comparación con los tamaños tradicionales de las memorias DRAM comerciales) parece ser adecuado para capturar una fracción importante de los accesos a la NVRAM. Estos resultados nos motivan a centrar la investigación en el uso de tecnologías de memoria alternativas, como memorias magnéticas (MRAM) [29,30] o DRAM integrada (eDRAM) [31, 32], que son más rápidas y consumen menos energía que la DRAM convencional.

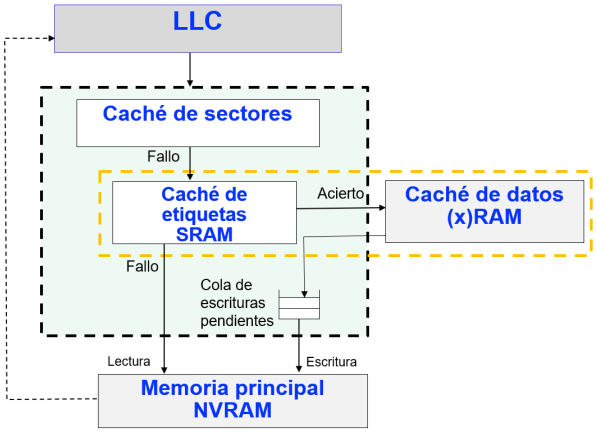


Fig. 1: Diagrama del controlador de memoria híbrido propuesto (HMC o Hybrid Memory Controller).

El enfoque ideado implementa cuatro componentes principales: la caché de sectores del HMC o simplemente caché de sectores, el soporte para los medios DRAM/eDRAM/MRAM, la caché de etiquetas (TAG cache) y el soporte NVRAM. La Figura 1 muestra los componentes diseñados y el flujo de datos ascendente-descendente.

Uno de los principales problemas a la hora de utilizar la DRAM convencional como caché es la ubicación de la caché de etiquetas. Si se coloca en la misma DRAM, es necesario realizar un acceso para buscar si el bloque de destino se encuentra en la caché. Como la DRAM es destructiva y relativamente lenta, esto significa que se puede perder un tiempo precioso independientemente de si el sector de destino está o no en la DRAM. Para evitar este inconveniente, un enfoque alternativo podría ser implementarlo en una caché integrada (es decir, colocada en la placa controladora de memoria). Sin embargo, este diseño supone un reto si se usa memoria convencional, sobre todo debido a la gran capacidad (decenas de GB) de los DIMM DRAM.

Para hacer frente a estos retos, exploramos diseños propios de DRAM con una capacidad menor, de ahora en adelante *in-house* DRAM. La idea clave es reducir la caché de etiquetas para implementarla en una caché SRAM mucho más rápida. De ahora en adelante, denominamos caché de datos y caché de etiquetas a las cachés que mantienen los datos y las etiquetas, respectivamente.

La cuestión que se plantea es cómo de grande debe ser la caché de datos. En principio, la caché debería construirse con una tecnología lo suficientemente densa como para almacenar una gran cantidad de datos. En este trabajo, exploramos en primer lugar la memoria DRAM convencional (DRAM Cache). Sin embargo, como mostrarán nuestros resultados experimentales, a esta caché DRAM se accede con frecuencia y, por tanto, un tamaño relativamente pequeño es suficiente para capturar una parte significativa de los accesos a la NVRAM.

Mejoramos el rendimiento del controlador de memoria con dos opciones de diseño principales respecto a las soluciones existentes. Por un lado, el hecho de que se acceda con frecuencia a la caché DRAM

significa que las principales mejoras de rendimiento vendrán de acelerar esta caché. Por otro lado, como se necesita una caché relativamente pequeña, se pueden utilizar otras tecnologías compatibles con CMOS (por ejemplo, eDRAM) o más densas (por ejemplo, MRAM) para integrarlas en la placa controladora de memoria. De este modo, se puede ahorrar una importante cantidad de hardware, como el canal de memoria y el zócalo DIMM, así como consumo de energía, en comparación con el uso de módulos DRAM convencionales. Por tanto, en este trabajo consideramos MRAM, eDRAM e *in-house* DRAM eficientemente adaptadas a las necesidades del controlador propuesto.

IV. Simuladores

A continuación vamos a presentar los simuladores utilizados para llevar a cabo todos los experimentos y el modelado de los componentes. El entorno de simulación se ha construido en con dos simuladores principales, gem5 y NVMain. El primero se ha utilizado para modelar la parte del procesador (incluyendo las memorias caché), y el segundo para modelar el subsistema de memoria principal y las cachés de etiquetas y datos. NVMain se puede compilar conjuntamente con gem5, lo que significa que en un solo binario podemos utilizar tanto gem5 21.0.1.0 como NVMain. A continuación, resumimos las principales características de cada simulador.

El simulador gem5 es un simulador modular dirigido por eventos. Este simulador ofrece una gran flexibilidad en la configuración del sistema, gracias al uso de scripts Python que inicializan los componentes del sistema y actúan como fuente de configuración del mismo. Esto permite configuraciones complejas de procesadores y jerarquías de caché. También dispone de dos modos de trabajo distintos, simulación completa del sistema (FS) y emulación de llamadas al sistema (SE). En el modo FS, gem5 simula un sistema físico completo desde el arranque, de forma similar a una máquina real. Por otro lado, el modo SE ofrece una forma rápida de simular la ejecución un binario abstrayendo el sistema operativos y el hardware subyacente.

NVMain es un simulador de memoria principal ciclo a ciclo que modela tanto memorias no volátiles como DRAM. Además de combinarse con gem5, NVMain puede compilarse de forma autónoma, lo que permite simular el subsistema de memoria mediante trazas. Este simulador permite la implementación de componentes del sistema de memoria principal, mediante la extensión de clases base definidas en C++. También ofrece soporte para sistemas de memoria híbridos. Además, el modelo de memoria puede modificarse mediante el uso de archivos de configuración, que definen los parámetros de temporización, la geometría de la memoria, y el controlador de memoria.

A. Componentes modelados

Para evaluar la propuesta se modela el sistema completo, desde el núcleo del procesador hasta los niveles inferiores de la jerarquía de memoria, pasando por los niveles de la jerarquía de caché.

Con el simulador gem5 se modela la parte del procesador, incluyendo el núcleo del procesador, las cachés L1 y L2 y la caché de sectores del HMC. No se modela ninguna la L3, ya que este estudio se centra en el subsistema de memoria principal y no en chip del procesador. Sin embargo, se ha modelado una caché L2 lo suficientemente grande como para capturar la mayoría de los accesos a la memoria como en los procesadores reales que incluyen una caché L3.

La caché de sectores también se modela en gem5. Dado que NVMain es un simulador de memoria principal, se utiliza para implementar el controlador de memoria principal que controla las distintas tecnologías de memoria (por ejemplo, DRAM o NVRAM). La figura 2 presenta los principales componentes del sistema de forma jerárquica.

Finalmente, el controlador de memoria incluye los controladores de las tecnologías de memoria conectadas, así como las cachés de etiquetas y datos.

B. Parámetros del sistema

En la evaluación de la propuesta se utilizan los parámetros de configuración del sistema que se muestran en la tabla I. Los tiempos de acceso (latencia) se obtienen con la herramienta CACTI [33] en ns para un nodo tecnológico de 22 nm y se traducen a ciclos de procesador para una frecuencia de 2GHz. El ta-

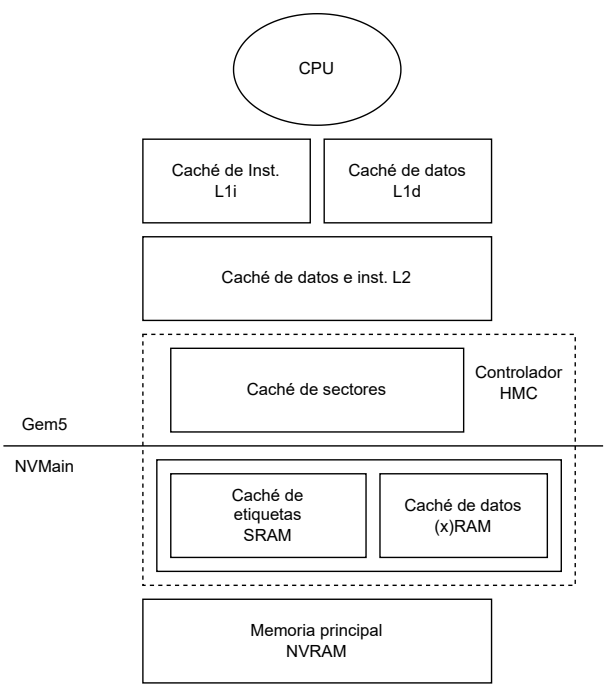


Fig. 2: Herramientas de simulación utilizadas y componentes del sistema modelados en cada una de ellas. Gem5 se muestra en la parte superior de la figura y NVMain en la parte inferior. En el caso de la DRAM convencional, los datos de la (x)RAM se encuentran fuera del controlador HMC, en un módulo DRAM.

Tabla I: Parámetros de configuración del simulador gem5 para un nodo de 22 nm. Los ciclos se han calculado para una frecuencia de 2 GHz.

Componente	Parámetro	Valor
CPU	Tipo	TimingSimpleCPU
	Frecuencia	2.0 GHz
Caché de datos L1d	Tamaño, Tam. bloque, Asociatividad	32 KB, 64 B, 8-vías
	Latencia Etiqueta/Datos	3/3 cycles
Caché de inst. L1i	Tamaño, Tam. bloque, Asociatividad	32 KB, 64 B, 8-vías
	Latencia Etiqueta/Datos	3/3
Caché de datos e inst. L2	Tamaño, Tam. bloque, Asociatividad	2 MB, 64 B, 16-vías
	Latencia Etiqueta/Datos	11/11 cycles
	Política de escritura	Write-back
Caché de Sectores	Tamaño, Tam. bloque, Asociatividad	8 MB, 256 B, 16-vías
	Latencia Etiqueta/Datos	ciclos 17/17
	Política de escritura	Write-back
Caché de datos (x)RAM	Canales de memoria	2
	Espacio de almacenamiento	64MB

maño de la caché de sectores y de la caché (x)RAM (donde x se refiere a la tecnología específica de la caché de datos) se ha escalado al tamaño del problema de las cargas de trabajo para obtener valores representativos. Con esta herramienta, se han podido modelar los distintos tiempos de acceso a la memoria utilizando como memoria base un DIMM 2933-DDR4 de 4 GB, que es la memoria que denominamos DRAM convencional. En cuanto a la DRAM *in-house*, se reduce el tamaño de la DRAM convencional de 4 GB a 64 MB. Por último, los valores de acceso a la MRAM se modelaron según [34].

La tabla II resume las principales características de las tecnologías estudiadas para evaluar la propuesta en ciclos de bus de memoria. Se muestran los tiempos de acceso en caso de que se falle o se accede en el búfer de fila (RB). La tabla también especifica los valores del retardo de precarga de fila (t_{RP}), el retardo de fila a columna (t_{RCD}) y la latencia de columna (CL). La fila inferior cuantifica los fallos y aciertos en el búfer de fila en ciclos de procesador (ciclos) para un procesador de 2 GHz. Es importante tener en cuenta que el tiempo de acesso al medio dependerá de la tasa de acierto eb el búfer de fila. Para aplicaciones con alta localidad espacial, la tasa de acierto en el búfer de fila será alta, acercando así el tiempo de acceso al tiempo de acierto en el búfer de fila. Por otro lado, para aplicaciones con baja localidad espacial, la tasa de acierto en el búfer de fila será bastante pobre, aumentando así el tiempo de acceso.

V. Cargas de trabajo

Se han utilizado tres cargas de trabajo principales para evaluar la propuesta; a continuación se resumen las principales características de cada una de ellas.

Redis. Esta carga de trabajo es una estructura de almacenamiento en memoria que puede utilizarse como base de datos, caché o broker de mensajes. Redis ofrece diferentes tipos de estructuras de datos como hashes, listas, conjuntos, conjuntos ordenados, etc. Soporta de forma nativa la replicación de datos, la política de reemplazo LRU y múltiples niveles de persistencia en disco. Redis se compone de dos pruebas principales: "redis-benchmark" y "memtier_benchmark". El primero es una utilidad incluida durante la instalación de Redis que ofrece una forma de simular órdenes como si fueran realizadas por N clientes al mismo tiempo para un total de M peticiones.

Memtier. Se trata de una herramienta de *benchmarking* para probar instancias tanto de Memcached como de Redis. Algunas de sus características son el ratio de lectura/escritura configurable y la posibilidad de elegir el patrón de acceso a los datos.

MySQL. Actualmente, MySQL es uno de los sistemas de gestión de bases de datos (SGBD) más conocidos. Su código es abierto y ofrece una gran variedad de características, como la restauración y recuperación ante fallos. Lo utilizan gigantes tecnológicos como Google, Facebook o Twitter. Utilizamos la herramienta de evaluación comparativa "sysbench" para realizar los experimentos con el servidor MySQL.

Los parámetros que se han empleado en los experimentos para cada uno de los benchmarks descritos en esta sección están detallados en la tabla III.

VI. Resultados experimentales

En esta sección se presentan y analizan los resultados experimentales obtenidos variando la configuración del sistema (procesador y tecnologías de memoria) y las cargas de trabajo presentadas anteriormente.

Se comparan y analizan las diferencias de prestaciones entre las cuatro tecnologías de memoria utilizadas como caché de datos (x)RAM, es decir, DRAM convencional, DRAM *in-house*, eDRAM y MRAM.

A. Análisis de la latencia de la memoria en la HMC

El objetivo principal de nuestra propuesta es reducir la penalización por fallo del HMC. Se pueden distinguir dos penalizaciones por fallo en el HMC diferentes dependiendo de si el acceso acierta o no en la caché de etiquetas. En el primer caso, el sector será servido por la (x)RAM caché, y en el segundo por la memoria principal, NVRAM.

Por lo tanto, la latencia media dependerá de la tasa de aciertos de la caché de etiquetas. Es decir, cuanto mayor sea el porcentaje de aciertos, más solicitudes serán atendidas por la caché (x)RAM y, por tanto, la latencia media será menor.

La tabla IV presenta los resultados de latencia para Redis, MySQL y Memtier en ciclos de procesador. Las latencias se desglosan en función de si hay un acierto en la caché de etiquetas (latencia de la caché (x)RAM) o un fallo (latencia de NVRAM). Las latencias medias se presentan en ambos casos. Notese que en caso de acierto en la memoria intermedia, la latencia es menor que en caso de fallo. Comparando la latencia media (latencia media) de estas tres aplicaciones, se puede observar que MySQL es la que presenta el valor más alto (un 21 % y un 12 % mayor que Redis y Memtier, respectivamente).

Tabla II: Especificaciones de los distintos modelos de memoria. Ciclos se refiere a los ciclos de procesador.

	Comm. DRAM	In-house DRAM	eDRAM	MRAM	NVRAM
Busqueda de etiqueta (ns/ciclos)	4 / 8				0
Tiempo de acceso de lectura (ns)					
fallo en el búfer de filas (ns)	43	16.5	10	10	176.2
acierto en el búfer de filas (ns)	14	5.5	3	3	43
Tiempo de acceso de escritura (ns)					
fallo en el búfer de filas (ns)	43	16.5	10	50	176.2
acierto en el búfer de filas (ns)	14	5.5	3	16	43
Frecuencia de la memoria	3800 MHz				488
Frecuencia del procesador	2000 MHz				
ciclos (fallo/acierto en el búfer de filas)	86/28	33/11	20/6	20/6R 100/32W	353/86

Tabla III: Valores de los parámetros utilizados en las cargas de trabajo estudiadas. Instancias se refiere a lanzar varios benchmarks sobre el servidor para aumentar el estrés.

Redis	
Parámetro	**Valor**
# clientes	200
# peticiones/test	45000
Tam. petición	2000 B
# instancias	2

Memtier	
Parámetro	**Valor**
# entradas	100000
# hilos	8
# instancias	3

MySQL	
Parámetro	**Valor**
# hilos	4
# clientes	15
Tam. datos	1024

Evaluación holística

En esta sección se analizan las prestaciones justo antes y después de la caché de sectores. Para ello mostramos la latencia de fallo de la L2 (la LLC en nuestro procesador) y la latencia de fallo del HMC. Se persiguen dos objetivos principales: i) comprobar la eficacia de la caché sectorial y de cada componente individual del HMC, y ii) comprobar hasta qué punto la (x)RAM y la NVRAM contribuyen a las prestaciones finales del sistema. Para ser exhaustivos y facilitar el análisis, mostramos métricas adicionales como la tasa de aciertos del HMC y la tasa de aciertos de la caché de etiquetas (es decir, la tasa de aciertos de la caché (x)RAM), así como el MPKI del HMC (fallos por kilo-instrucciones).

Las tablas V, VI y VII presentan los resultados. Se puede apreciar que las prestaciones del controlador de memoria, cuantificado en términos de latencia de fallo de HMC, varía significativamente entre aplicaciones. Esta latencia es superior a 100 ciclos para todas las aplicaciones estudiadas en la caché DRAM convencional, que es la tecnología con las peores prestaciones, y entre 67 y 80 ciclos en el resto de tecnologías (x)RAM. Estos resultados demuestran que se pueden conseguir importantes mejoras de las prestaciones con tecnologías de memoria alternativas con respecto al uso de DRAM convencional. Por ejemplo, puede observarse que la tecnología MRAM mejora las prestaciones de la DRAM convencional en un 51 % (101/67), 77 % (122/69) y 52 % (108/71) en Redis, MySQL y Memtier, respectivamente.

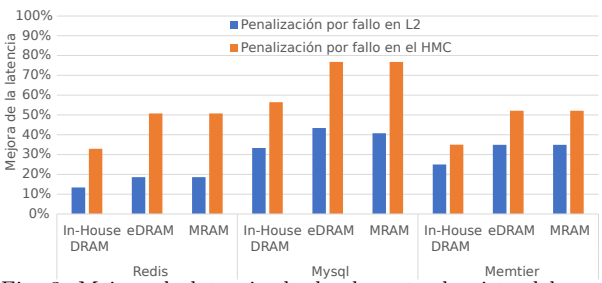


Fig. 3: Mejora de latencia desde el punto de vista del procesador (azul) y del controlador de memoria (naranja) con respecto al modelo de memoria DRAM convencional.

Para evaluar nuestra propuesta utilizamos como referencia un sistema sin la caché (x)RAM. Nos referiremos a este sistema como modelo NV-S y a nuestra propuesta como NV-S-D.

La figura 3 muestra las mejoras de latencia tanto en la penalización por fallo de L2 (LLC) como en la penalización por fallo del HMC. Los resultados se muestran para cada tecnología de memoria alternativa y se comparan con la DRAM convencional utilizada como referencia. Puede observarse que tanto la eDRAM como la MRAM presentan valores similares. Las prestaciones pueden aumentar hasta un 77 % en la penalización por fallo de HMC, pero la penalización por fallo de L2 es mucho menor debido a la eficiencia de la caché de sectores.

Cabe destacar que la penalización por fallo de L2 no es demasiado alta, ya que siempre está por debajo de 85 ciclos (véanse los valores de las tablas anteriores). La razón es doble. Por un lado, la caché de sectores presenta una alta tasa de aciertos, especialmente en Redis y MySQL donde siempre está por encima del 75 % y 55 %, respectivamente. Por otro lado, apenas se accede a la NVRAM, ya que la caché de etiquetas de la (x)RAM presenta un excelente porcentaje de aciertos, superior al 82 % en todas las aplicaciones. Además, en Redis y MySQL este valor supera el 94 %.

Análisis de la latencia de memoria por componente

Para analizar en qué medida los distintos componentes del subsistema de memoria contribuyen a las prestaciones del sistema, hemos analizado su contribución al AMAT (es decir, el tiempo medio de acceso

Tabla IV: Latencia en caso de fallo en la caché del HMC para los benchmarks estudiados. Estos valores incluyen la búsqueda de etiquetas (8 ciclos), el tiempo de acceso y la contención.

Benchmark	Memoria	Tecnología de la caché de Datos			
		Commodity DRAM	In-house DRAM	eDRAM	MRAM
Redis	Cache (x)RAM (ciclos)	70	49	41	42
	NVRAM (ciclos)	256	242	242	242
	Latencia media (ciclos)	101	76	67	67
MySQL	Cache (x)RAM (ciclos)	89	51	43	44
	NVRAM (ciclos)	254	242	242	246
	Latencia media (ciclos)	122	78	69	69
Memtier	Cache (x)RAM (ciclos)	81	53	45	46
	NVRAM (ciclos)	258	252	255	255
	Latencia media (ciclos)	109	80	71	72

Tabla V: Estadísticas de Redis para cada tecnología. Todas las métricas de latencia se expresan en ciclos de procesador (ciclos).

Estructura de memoria	REDIS / NV-S-D	comm. DRAM	DRAM	eDRAM	MRAM
Caché L2 (LLC)	Latencia de fallo de L2	51	45	43	43
Caché de Sectores	Tasa de aciertos del HMC	75.93	75.52	75.32	75.41
	HMC MPKI	1.81	1.83	1.85	1.84
	Latencia de fallo del HMC	101	76	67	67
Caché (x)RAM	Tasa de aciertos etiquetas (%)	94.87	94.99	95.00	95.02
	Latencia de la (x)RAM	70	49	41	42
NVRAM	Lectura / Escritura (%)	49.82 / 50.18	49.85 / 50.15	49.84 / 50.16	49.83 / 50.17
	Latencia de la NVRAM	256	242	242	242

Tabla VI: Estadísticas de MySQL para cada tecnología.

Estructura de memoria	MySQL / NV-S-D	commDRAM	DRAM	eDRAM	MRAM
Caché L2 (LLC)	Latencia de fallo de L2	76	57	53	54
Caché de Sectores	Tasa de aciertos del HMC	57.19	57.48	57.56	56.62
	HMC MPKI	1.95	1.95	1.93	1.97
	Latencia de fallo del HMC	122	78	69	69
Caché (x)RAM	Tasa de aciertos etiquetas (%)	96.54	96.26	96.46	96.18
	Latencia de la (x)RAM	89	51	43	44
NVRAM	Lectura / Escritura (%)	49.90 / 50.10	49.93 / 50.07	49.91 / 50.09	49.93 / 50.07
	Latencia de la NVRAM	254	242	242	246

Tabla VII: Estadísticas de Memtier para cada tecnología.

Estructura de memoria	Memtier / NV-S-D	commDRAM	DRAM	eDRAM	MRAM
Caché L2 (LLC)	Latencia de fallo de L2	85	68	63	63
Caché de Sectores	Tasa de aciertos del HMC	40.74	41.24	41.01	40.98
	HMC MPKI	3.39	3.40	3.45	3.40
	Latencia de fallo del HMC	108	80	71	71
Caché (x)RAM	Tasa de aciertos etiquetas (%)	82.77	82.90	83.27	82.91
	Latencia de la (x)RAM	81	53	45	46
NVRAM	Lectura / Escritura (%)	49.63 / 50.37	49.55 / 50.45	49.64 / 50.36	49.61 / 50.39
	Latencia de la NVRAM	258	252	255	255

a la memoria). Para ello, se han calculado las latencias de cada componente de la jerarquía de memoria: las cachés L1 y L2, la caché de sectores, la caché (x)RAM y la memoria principal NVRAM, se acumulan para todos los accesos a memoria y, a continuación, se dividen por el número total de instrucciones de memoria emitidas por el procesador (es decir, no sólo las que acceden a la HMC). A continuación, se traza una barra para los sistemas estudiados. La figura 4 presenta los resultados. Se trazan dos barras para cada benchmark, una para el sistema NV-S-D y otra para el NV-S con fines comparativos.

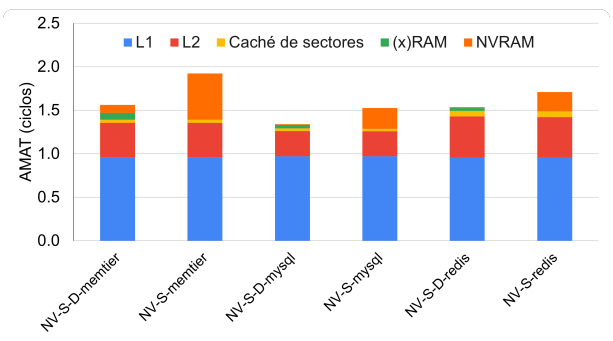


Fig. 4: Contribución por componente al tiempo medio de acceso a memoria (AMAT) en el sistema NV-S y NV-S-D para la caché *in-house* DRAM.

Como era de esperar, la caché L1 (parte azul) presenta aproximadamente la misma contribución para todas las aplicaciones, y la caché L2 (parte roja) y la caché HMC (parte amarilla) presentan la misma contribución para la misma aplicación independientemente del modelo de memoria. Se puede observar que la contribución del HMC es bastante baja, lo que significa que no se accede a ella frecuentemente por cada instrucción de memoria emitida. Una mayor longitud de la barra amarilla significa que esa aplicación ha tenido un número notable de accesos al HMC con una alta tasa de aciertos HMC. De los resultados de las tablas anteriores se desprende que

Redis es la aplicación con la mayor tasa de aciertos en el HMC (superior al 75 %).

Para averiguar las diferencias en las prestaciones entre el modelo de memoria NV-S-D y el modelo de memoria NV-S, se comparara la barra naranja del sistema NV-S con el conjunto de las barras naranja y verde del sistema NV-S-D, ya que las demás barras no presentan variaciones. Las mayores ganancias de prestaciones se observan en Memtier, ya que el componente NVRAM (barra naranja) es casi 3 veces mayor que en las demás aplicaciones. Los resultados muestran que la propuesta funciona eficazmente en todas las aplicaciones, ya que la barra naranja se reduce significativamente.

Los resultados presentados en la parte superior de este gráfico corroboran los resultados de mejora de la latencia presentados en la figura 3. Sin embargo, las prestaciones globales dependen de cómo se reduzca la longitud total de la barra (en %). Esto significa que los mecanismos aplicados a la HMC para reducir la latencia dependerán en gran medida de los componentes comunes del AMAT. Dicho de otro modo la reducción total del AMAT está limitada por la longitud de las barras amarilla, roja y azul.

Las ventajas generales en las prestaciones se ven limitadas por la reducción (en %) del AMAT. Desde un punto de vista analítico la pregunta es: ¿cómo podríamos mejorar el AMAT? Más concretamente, ¿cómo podríamos reducir la diferencia (en %) entre NV-S y NV-S-D? Hay dos opciones principales: i) reducir el impacto de la L2 y L1 o ii) reducir el aporte de la caché de sectores y la (x)RAM. La primera puede lograrse actuando en el lado del procesador, la segunda en el lado del HMC, como se discute a lo largo de este documento.

Mejoras en las prestaciones del sistema

Una vez analizados los principales componentes y su contribución al AMAT, estudiamos en qué medida las mejoras de latencia a lo largo de la jerarquía de caché y memoria se traducen en mejoras finales de las prestaciones del sistema. En otras palabras, analizamos el aumento de velocidad que introduce el controlador HMC ideado.

La figura 5 presenta los resultados de cada tecnología sobre el modelo de memoria NV-S. El aumento de las prestaciones del sistema oscila entre el 12 % y el 24 %. La DRAM convencional ofrece unas prestaciones muy inferiores a las de las demás tecnologías, con la única excepción de Redis, donde se observa unas prestaciones similares (1 % de diferencia). Como cabía esperar de los estudios anteriores, la eDRAM y la MRAM son las que ofrecen mejores resultados y alcanzan unas prestaciones similares, tanto como un 15 % en MySQL y un 24 % en MRAM. Recordemos que estas ganancias en las prestaciones se consiguen añadiendo la tecnología (x)RAM, es decir, la caché de segundo nivel, ya que el sistema NV-S ya incluye la caché HMC. Por lo tanto, podemos concluir que se pueden conseguir importantes ganancias en las prestaciones con las tecnologías eDRAM y MRAM.

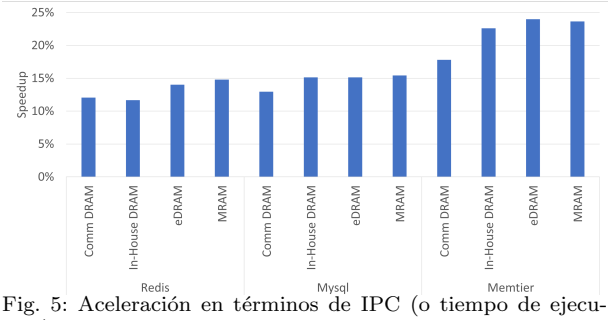


Fig. 5: Aceleración en términos de IPC (o tiempo de ejecución) del uso de tecnologías de memoria alternativas respecto al sistema NV-S.

VII. Conclusiones

Este trabajo se ha centrado en la mejora de las prestaciones de los sistemas de memoria para aplicaciones de big data, centrándose en el controlador de memoria y en las tecnologías de memoria alternativas. A diferencia de los enfoques existentes, la solución ideada propone una jerarquía de caché de dos niveles que combina distintas tecnologías en el controlador de memoria. La idea clave es atacar el cuello de botella en las prestaciones que introduce la lentitud de la NVRAM, mitigando la congestión del bus y mejorando las prestaciones.

Se han analizado las tecnologías de memoria *in-house* DRAM, eDRAM y MRAM actuando como caché (x)RAM. Los resultados muestran que eDRAM y MRAM son las tecnologías con mejores prestaciones. En comparación con el sistema NV-S, la eDRAM y la MRAM son capaces de mejorar la penalización por fallo del HMC en casi un 78 % en MySQL y por encima del 50 % en el resto de aplicaciones. Estas mejoras se reducen al 40, 33 y 28 % cuando se mejora la latencia de fallo de la caché L2 en MySQL, Memtier y Redis. Las prestaciones globales del sistema mejoran en un 15 % y un 23 % dependiendo de la carga de trabajo estudiada con respecto al uso de DRAM convencional.

Este nuevo diseño puede aportar importantes mejoras en las prestaciones en futuros procesadores de servidor y sistemas en la nube que ejecuten aplicaciones de big data y otras aplicaciones que requieran enormes cantidades de datos.

Como cualquier diseño inicial, necesita evolucionar para mejorar aún más las prestaciones globales del sistema. En este sentido pretendemos continuar investigando en mecanismos destinados a ocultar las grandes latencias de la NVRAM y procesadores más agresivos que soporten un mayor paralelismo a nivel de memoria.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación español y el FEDER europeo mediante las subvenciones PID2021-123627OB-C51 y TED2021-130233B-C32. Manel Lurbe cuenta con el apoyo de Ayudas para Contratos predoctorales UPV - subprograma 1 (PAID-01-20).

Referencias

[1] Rajeev Balasubramonian and Norman P Jouppi, *Multi-core cache hierarchies*, Springer Nature, 2022.

[2] Alan Jay Smith, "Cache memories," *ACM Comput. Surv.*, p. 473–530, sep 1982.

[3] Q.G. Samdani and M.A. Thornton, "Cache resident data locality analysis," *Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No.PR00728)*, pp. 539–546, 2000.

[4] Andreea Anghel, Gero Dittmann, Rik Jongerius, and R Luijten, "Spatio-temporal locality characterization," *1st Workshop on Near-Data Processing (WoNDP)*, pp. 1–5, 2013.

[5] Wm. A. Wulf and Sally A. McKee, "Hitting the memory wall: Implications of the obvious," *SIGARCH Comput. Archit. News*, p. 20–24, mar 1995.

[6] Xian-He Sun, "Remove the memory wall: from performance modeling to architecture optimization," *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pp. 2–2, 2006.

[7] Hosein Mohammadi Makrani, Setareh Rafatirad, Amir Houmansadr, and Houman Homayoun, "Main-memory requirements of big data applications on commodity server platform," *18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 653–660, 2018.

[8] Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, Rio de Janeiro, Brazil*, pp. 415–426, June 2015.

[9] O. Mutlu, "Rethinking memory system design," *Mobile System Technologies Workshop (MST)*, pp. 1–3, 2016.

[10] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, "Overview of candidate device technologies for storage-class memory," *IBM Journal of Research and Development*, pp. 449–464, 2008.

[11] Katsuhiko Hoya, Kosuke Hatsuda, Kenji Tsuchida, Yohii Watanabe, Yusuke Shirota, and Tatsunori Kanai, "A perspective on nvram technology for future computing system," *International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp. 1–2, 2019.

[12] Steven Pelley, Peter M. Chen, and Thomas F. Wenisch, "Memory persistency," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 265–276.

[13] Pengfei Zuo, Yu Hua, Ming Zhao, Wen Zhou, and Yuncheng Guo, "Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 442–454.

[14] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu, "Improving 3d NAND flash memory lifetime by tolerating early retention loss and process variation," *Abstracts of the ACM International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS, Irvine, CA, USA*, p. 106, June 2018.

[15] Saeed Kargar and Faisal Nawab, "Challenges and future directions for energy, latency, and lifetime improvements in nvms," *Distributed and Parallel Databases*, pp. 442–454, September 2022.

[16] Gaurav Dhiman, Raid Ayoub, and Tajana Rosing, "Pdram: A hybrid pram and dram main memory system," *46th ACM/IEEE Design Automation Conference*, pp. 664–669, 2009.

[17] Chenxi Wang, Huimin Cui, Ting Cao, John Zigman, Haris Volos, Onur Mutlu, Fang Lv, Xiaobing Feng, and Guoqing Harry Xu, "Panthera: Holistic memory management for big data processing over hybrid memories," *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 347–362, 2019.

[18] Reza Salkhordeh, Onur Mutlu, and Hossein Asadi, "An analytical model for performance and lifetime estimation of hybrid dram-nvm main memories," *IEEE Transactions on Computers*, pp. 1114–1130, 2019.

[19] Luiz E. Ramos, Eugene Gorbatov, and Ricardo Bianchini, "Page placement in hybrid memory systems," *Proceedings of the 25th International Conference on Supercomputing, Tucson, AZ, USA*, pp. 85–95, May-June 2011.

[20] Santiago Bock, Bruce R. Childers, Rami G. Melhem, and Daniel Mossé, "Concurrent page migration for mobile systems with os-managed hybrid memory," *Computing Frontiers Conference, CF'14, Cagliari, Italy*, pp. 31:1–31:10, May 2014.

[21] Santiago Bock, Bruce R. Childers, Rami G. Melhem, and Daniel Mossé, "Concurrent migration of multiple pages in software-managed hybrid main memory," *34th IEEE International Conference on Computer Design, ICCD, Scottsdale, AZ, USA*, pp. 420–423, October 2016.

[22] Ahmad Hassan, Hans Vandierendonck, and Dimitrios S. Nikolopoulos, "Energy-efficient hybrid dram/nvm main memory," *International Conference on Parallel Architecture and Compilation (PACT)*, pp. 492–493, 2015.

[23] Jeffrey C. Mogul, Eduardo Argollo, Mehul A. Shah, and Paolo Faraboschi, "Operating system support for NVM+DRAM hybrid main memory," *Proceedings of HotOS: 12th Workshop on Hot Topics in Operating Systems, Monte Verità, Switzerland*, May 2009.

[24] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang, "A durable and energy efficient main memory using phase change memory technology," *Proceedings - International Symposium on Computer Architecture*, pp. 14–23, June 2009.

[25] Yang Li, Saugata Ghose, Jongmoo Choi, Jin Sun, Hui Wang, and Onur Mutlu, "Utility-based hybrid memory management," *IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 152–165, 2017.

[26] HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael A. Harding, and Onur Mutlu, "Row buffer locality aware caching policies for hybrid memories," *IEEE 30th International Conference on Computer Design (ICCD)*, pp. 337–344, 2012.

[27] Justin Meza, Jichuan Chang, HanBin Yoon, Onur Mutlu, and Parthasarathy Ranganathan, "Enabling efficient and scalable hybrid memories using fine-granularity dram cache management," *IEEE Computer Architecture Letters*, pp. 61–64, 2012.

[28] Dong Li, Jeffrey S. Vetter, Gabriel Marin, Collin McCurdy, Cristian Cira, Zhuo Liu, and Weikuan Yu, "Identifying opportunities for byte-addressable non-volatile memory in extreme-scale scientific applications," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, 2012, pp. 945–956.

[29] E. Chen, D. Apalkov, Z. Diao, A. Driskill-Smith, D. Druist, D. Lottis, V. Nikitin, X. Tang, S. Watts, S. Wang, S. A. Wolf, A. W. Ghosh, J. W. Lu, S. J. Poon, M. Stan, W. H. Butler, S. Gupta, C. K. A. Mewes, Tim Mewes, and P. B. Visscher, "Advances and future prospects of spin-transfer torque random access memory," *IEEE Transactions on Magnetics*, pp. 1873–1878, 2010.

[30] Dmytro Apalkov, Bernard Dieny, and J. M. Slaughter, "Magnetoresistive random access memory," *Proceedings of the IEEE*, vol. 104, no. 10, pp. 1796–1830, 2016.

[31] Richard E. Matick and Stanley E. Schuster, "Logic-based edram: Origins and rationale for use," *IBM Journal of Research and Development*, p. 145–165, Jan 2005.

[32] P.W. Diodato, "Embedded dram: more than just a memory," *IEEE Communications Magazine*, vol. 38, no. 7, pp. 118–126, 2000.

[33] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, p. 28, 2009.

[34] Yu-Der Chih, Yi-Chun Shih, Chia-Fu Lee, Yen-An Chang, Po-Hao Lee, Hon-Jarn Lin, Yu-Lin Chen, Chieh-Pu Lo, Meng-Chun Shih, Kuei-Hung Shen, Harry Chuang, and Tsung-Yung Jonathan Chang, "13.3 a 22nm 32mb embedded stt-mram with 10ns read speed, 1m cycle write endurance, 10 years retention at 150°c and high immunity to magnetic field interference," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, pp. 222–224.

Modelo de predicción de direcciones basado en búfferes dinámicos y SVM

Pablo Sánchez-Cuevas[1], Fernando Díaz-del-Río[1], Antonio Ríos-Navarro[1] y Daniel Casanueva-Morato[1]

Resumen— **En los últimos años, se ha intensificado el cómputo masivo de datos, desde todo tipo de plataformas de propósito general y específico, con objeto de suplir la creciente demanda de sistemas inteligentes. A pesar de las diversas alternativas propuestas en el campo de la Arquitectura de Computadores, aún permanece un crucial cuello de botella: la eficiencia y velocidad del acceso a la memoria. En este trabajo, se explora una de las técnicas actualmente claves en los sistemas de memoria: la predicción de accesos (también llamada predicción de direcciones); y se discuten primero tanto las propuestas más clásicas, como las noveles. Como consecuencia, se propone el modelo predictor "BufferSVM", el cual mejora (en su versión ideal) el rendimiento de los predictores de la literatura actual (también, en su versión ideal). En una comparación directa y preliminar con el predictor DFCM, nuestra propuesta supera su rendimiento en más de un 5 %, ahorrando un coste en memoria mayor a un orden de magnitud, validando así las diferentes decisiones que hemos tomado en el diseño del predictor BufferSVM.**

Palabras clave— **Address Prediction, Support Vector Machines, Memory Access, Superscalar Processors, Machine Learning.**

I. Introducción

Nuestro contexto actual viene marcado, ya no sólo por un incremento significativo en la demanda de cómputo por la sociedad, sino también en la complejidad de la misma. Ya no sólo se busca expandir el enorme conjunto de servicios proporcionado por grandes infraestructuras como Internet, sino también la integración de funcionalidades de Inteligencia Artificial: optimización de procesos y recursos, visión por computador, reconocimiento de patrones, generación inteligente de contenidos multimedia... Además, el objetivo es aplicar estos requisitos en todo tipo de escenarios, proporcionados, no sólo por computadores de propósito general, sino también por sistemas de propósito específico que presentan notables restricciones.

Sin duda, los arquitectos de computadores se enfrentan a grandes retos (y grandes oportunidades) en este contexto. Como respuesta a la demanda de cómputo, han sido propuestas un conjunto de arquitecturas que explotan todos los niveles de paralelismo y mejoran la eficiencia.Algunos ejemplos en los últimos años han sido los SoCs (System on Chip), las TPUs (Tensor Processing Units) o las redes IoT (Internet of Things), entre otros.

Por otra parte, otro frente de evolución ha sido la microarquitectura. Entre otros elementos, el acceso a la memoria sigue siendo uno de los mayores cuellos de botella ([1]). Históricamente las jerarquías de memoria en general y los sistemas de caché en particular han aliviado este problema. Aun así, técnicas como el prefetching, que consiste en precargar los datos más probables a usar por el procesador a posteriori, han sido integradas para optimizar estos sistemas ([2]). No obstante, dada la demanda de cómputo masivo de datos, este cuello de botella requiere de optimizaciones adicionales.

Siguiendo el éxito de los predictores de salto (como el TAGE, por ejemplo [3]), una optimización del uso de memoria son los predictores de valores, los cuales permiten predecir el valor final de una instrucción de lectura de memoria [4]. Gracias a esto, se puede ejecutar de manera especulativa todo un conjunto de instrucciones evitando el tiempo de espera de la instrucción de acceso a memoria. Una variante de dichos predictores de valores es la predicción de accesos, que tienen por objetivo predecir la dirección de memoria usada por una instrucción de lectura o escritura en memoria.

Esta última técnica es ampliamente usada en la actualidad para permitir un prefetching mucho más efectivo y preciso. En concreto, se busca tener siempre en el sistema de caché todos los datos que va a usar el procesador. Además, esto permitiría introducir (de manera especulativa) directamente los datos precargados en el motor de ejecución del procesador, pudiendo aplicar así especulación de valores ([5]).

Dada la importancia de la predicción de accesos, en este trabajo proponemos un modelo de predicción novel llamado BufferSVM, que resulta una mejora sustancial en rendimiento en comparación con el predictor clásico DFCM. En la Sección II se exploran las características de los predictores tanto clásicos como noveles. En la Sección III se discuten mejoras de diseño con respecto a predictores de la literatura, desembocando en el diseño propuesto en la Sección IV. En la Sección V se especifica el conjunto de pruebas para evaluar el rendimiento del BufferSVM con respecto al Differential Finite Context Method (DFCM). Finalmente, las conclusiones e ideas de trabajo futuro se resumen en la Sección VI.

II. Background

Los métodos más *clásicos* realizan la predicción utilizando tablas, y entre ellos podemos encontrar ejemplos como el Last Value ([6]), el Differential Value TAgged GEometric ([7]) o el Differential Finite Context Method (DFCM, [8]). A modo de ejemplo de predictor de este tipo, el DFCM presenta dos tablas separadas: (1) una primera a la que se accede

[1]Dpto. de Arquitectura y Tecnología de Computadores, Universidad de Sevilla, email: {psanchez4,fdiaz,arios,dcasanueva}@us.es

mediante el Program Counter (PC) de la instrucción de lectura/escritura de memoria y devuelve un valor hash y la dirección del acceso anterior de la misma instrucción, y (2) una segunda a la que se accede por el valor hash obtenido de la primera tabla y devuelve el stride o delta que se suma con la dirección del acceso anterior dando como resultado la dirección predicha.

Estos predictores tienen la ventaja de ser sencillos y de bajo coste hardware, incurriendo en una latencia igual al tiempo necesario para obtener los resultados intermedios o finales de las tablas. Sin embargo, este mapeo basado en memoria conlleva un rendimiento que depende en gran medida de que la mayoría de contextos de acceso posibles se almacenen en una tabla de capacidad limitada. Por lo tanto, el rendimiento de los predictores clásicos depende en gran medida del coste de memoria y de la complejidad y disparidad de los contextos de acceso de un programa dado ([9]).

Los nuevos métodos basados en Machine Learning intentan superar estas restricciones implementando la función de mapeo utilizando un modelo de Machine Learning, como en [9], [10], [11]. Un modelo complejo como una Red Neuronal Recurrente (RNN) permite al predictor aprender todo tipo de patrones de acceso, minimizando el posible impacto de enfrentarse a contextos complejos, en contraste con la contrapartida del predictor basado en tablas. Aunque esto tiene el precio de tener un coste de hardware considerable en la matriz de pesos que suelen incluir estos modelos, y en el conjunto de operaciones lineales y no lineales computadas para procesar las características del contexto y cualquier resultado intermedio. Además, mientras que la aplicación de un aprendizaje offline en este tipo de modelos permite la implementación de un predictor preajustado, es necesario un aprendizaje online adicional para adaptar el predictor al contexto local dado por una determinada aplicación de ejecución y sus datos de entrada. Aquí aparece un inconveniente: cuanto más complejo es un modelo de Machine Learning (como es el caso de los modelos de Deep Learning), más difícil resulta ajustarlo aplicando aprendizaje online[12].

III. Definiendo un nuevo modelo

El nuevo predictor propuesto en este trabajo parte de un conjunto de decisiones de diseño que pretenden superar algunos de los inconvenientes de otros predictores. La primera de estas decisiones fue sustituir la función de indexación basada en tablas del DFCM por un modelo de Machine Learning. Los métodos de Aprendizaje Profundo de [13], [10] y [11] suponen un gran coste de memoria ya sólo para la matriz de pesos de la Red Neuronal Recurrente (RNN), y además, tienen una latencia considerable tanto en la inferencia como en el reentrenamiento. Por ello, aquí hemos optado por una solución basada en Máquina de Vectores de Soporte (SVM, [14]) lineal, ya que presenta un coste mucho menor tanto en memoria como en latencia.

Un punto importante en los métodos basados en Deep Learning referenciados anteriormente era la predicción de deltas (diferencias entre direcciones de acceso consecutivos) mediante la clasificación con un diccionario ajustado estáticamente, donde cada clase/palabra está asociada a un delta. Aunque permite un modelo RNN mucho más eficiente gracias a la reducción del espacio de salida de cualquier posible valor delta a solo unos cientos de clases/palabras, hace que el predictor sea menos dinámico y esté más limitado por las cargas de trabajo seleccionadas para el entrenamiento estático del modelo. Debido a esto, también se incorpora un diccionario en nuestra propuesta, pero con la diferencia de ser 100 % dinámico: todas las clases/palabras se almacenan durante la ejecución, siguiendo una política de reemplazo LFU (Least Frequently Used). Además, y como ventaja destacable, este enfoque permite un diccionario de tamaño mucho menor, centrándose sólo en los pocos deltas posibles que pueden aparecer en la ejecución en un instante determinado. Obsérvese que el rol nuestro diccionario puede considerarse similar al de la segunda tabla del predictor DFCM, pero de menor tamaño.

Otra elección importante de nuestra propuesta es realizar las predicciones, no sobre un historial global de accesos, que es la secuencia de los últimos accesos que han tenido lugar para cualquier instrucción de acceso, sino sobre un historial local de accesos asociado al PC (Program Counter) de cada instrucción de acceso. Como principal ventaja, este enfoque no mezcla accesos de diferentes instrucciones, lo que significa que el predictor aprenderá secuencias de patrones asociados a cada instrucción por separado, obteniendo así una predicción más precisa. Sin embargo, por otro lado, se necesita una memoria para almacenar el historial (es decir, la secuencia de accesos) y la última dirección de acceso para cada instrucción de lectura/escritura en memoria: el búfer de entrada. Este componente puede relacionarse por similitud con la primera tabla del predictor DFCM, ya que también se indexa por el PC de la instrucción con el fin de emitir una característica para la posterior predicción de direcciones.

Para aliviar el coste de memoria del búfer de entrada y, lo que es más importante, para mejorar la precisión del predictor, se opta por una última decisión: en lugar de tener secuencias de acceso como deltas entre direcciones consecutivas, tenemos secuencias de las clases (extraídas del diccionario) que corresponden a cada uno de los deltas. En consecuencia, el modelo SVM predecirá la clase del delta correcto a partir de la secuencia histórica de clases de una instrucción dada. Obsérvese que el espacio de entrada del modelo SVM se reduce significativamente en comparación con los de las propuestas RNN referenciadas.

IV. Modelo propuesto: BufferSVM

En este trabajo proponemos un modelo predictor que puede considerarse como un híbrido entre el DFCM y una RNN: el predictor Buffer-SVM. En

concreto, a modo de resumen, supone una continuación del DFCM, pues la indexación de delta/strides mediante códigos hash es sustituida por una máquina de vectores de soporte (SVM). Sus componentes se enumeran a continuación:

1. **Buffer de entrada**. Una tabla de memoria que, para cada instrucción de lectura/escritura de memoria, almacena una entrada con (1) la última dirección de memoria a la que se accedió a_p y (2) la secuencia q_p de clases/palabras de acceso de los últimos n_q accesos de la instrucción. La tabla está indexada por el PC p de la instrucción de acceso a memoria dada.
2. **SVM**. Una máquina de vectores de soporte lineal que toma como entrada una secuencia q de n_q clases y devuelve como salida una clase predicha $\hat{c_p}$ entre las posibles n_c clases consideradas. Una posible implementación presentará tantos hiperplanos como n_c clases de salida se tengan, siguiendo una clasificación *one-to-all*. La función de predicción para cada hiperplano se expresa como:

$$f(q,c) = w_c \cdot q - b_c \tag{1}$$

$$pred(q,c) = \begin{cases} +1 & \text{si} \\ -1 & \text{en caso contrario} \end{cases} \tag{2}$$

donde w_c y b_c son el vector normal y el término independiente del c-ésimo hiperplano respectivamente. Para ajustar cada hiperplano, dada una muestra de q' como entrada y y'_c (que es igual a -1 si q' es de la clase c o $+1$ en caso contrario) como salida, se puede implementar un ajuste por descenso del gradiente mediante las siguientes expresiones:

$$d(q', y'_c, c) = 1 - y'_c \cdot f(q', c) \tag{3}$$

$$\nabla w_c = \begin{cases} 0 & \text{si } d(q', y'_c, c) \leq 0, \\ -y'_c \cdot q' & \text{en caso contrario} \end{cases} \tag{4}$$

$$\nabla b_c = \begin{cases} 0 & \text{si} \\ y'_c & \text{en caso contrario} \end{cases} \tag{5}$$

$$w'_c = w_c + \eta \cdot \nabla w_c \tag{6}$$

$$b'_c = b_c + \eta \cdot \nabla b_c \tag{7}$$

donde ∇w_c y ∇b_c son los gradientes del vector normal y del término independiente del c-ésimo hiperplano respectivamente, que se multiplican por la tasa de aprendizaje η para obtener el nuevo vector normal y término independiente w'_c y b'_c.
3. **Diccionario**. Otra tabla de memoria que almacena n_c entradas. Estas entradas se corresponden con las clases (también conocidas como palabras) que se utilizan en el predictor. Cada entrada contiene (1) un delta d, que es una zancada o diferencia entre dos direcciones de acceso a memoria consecutivas, y (2) un valor de confianza k. Se accede a una entrada, bien indexándola directamente con una clase dada, o bien consultando la entrada (y su clase correspondiente) que almacena el delta dado. Se aplica una política de reemplazo de uso menos frecuente (LFU) haciendo uso de la confianza k de cada entrada, que se incrementa al acceder a una entrada y se decrementa al acceder otra entrada distinta.

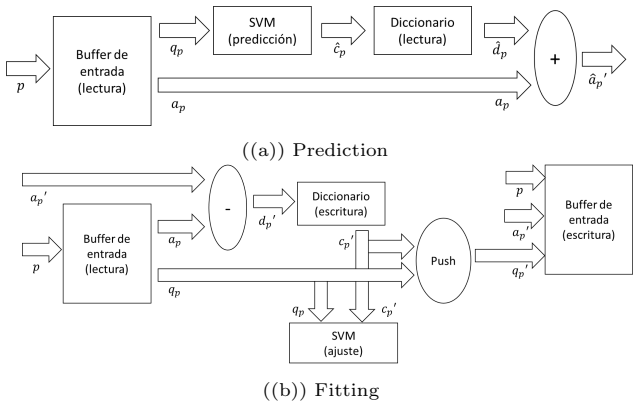


Fig. 1: Fases de predicción (a) y aprendizaje o ajuste (b) ejecutadas por el predictor BufferSVM. Se muestran también las operaciones realizadas por cada componente y sus entradas y salidas.

El funcionamiento preciso del predictor y cómo los componentes se comunican entre sí se explica en dos fases: la tubería ("pipeline") de predicción y la tubería de ajuste, como se representa en la Fig. 1. Sus dos partes se describen a continuación:

(a) La fase de predicción recibe un nuevo valor de PC p de una instrucción de acceso a memoria como entrada del predictor. Dicho PC se emplea como índice para el buffer de entrada, del que se leen tanto una dirección base a_p (de hecho, la última dirección de memoria accedida) como la secuencia de clases $q_p = [q_p(0), ..., q_p(n_q - 1)]$. La secuencia se introduce en la SVM lineal produciendo una predicción de clase $\hat{c_p}$. A continuación, dicha clase se utiliza para indexar el diccionario, del que se lee el delta resultante $\hat{d_p}$. La dirección de memoria predicha final es igual a la suma de la dirección base y el delta: $\hat{a_p}' = a_p + \hat{d_p}$. Si se ha producido un fallo durante la obtención de la entrada de la memoria intermedia, no se realiza ninguna predicción.

(b) La fase de ajuste responde a una situación opuesta: la dirección de memoria actual y realmente accedida $a_p{}'$ es obtenida por el procesador, además de su correspondiente PC p. Por lo tanto, se accede al búfer de entrada utilizando p como índice. Aparecen dos casos. (1) Si la entrada se almacena en la memoria intermedia de entrada (acierto), se calcula un nuevo delta como la diferencia entre la dirección de memoria actual y la última almacenada en esta entrada: $d_p{}' = a_p{}' - a_p$. (2) Si la entrada no se puede obtener de la memoria intermedia de entrada (fallo), el nuevo delta será $d_p{}' = 0$.
Así, para cualquiera de los dos casos, el nuevo

delta $d_p{}'$ se utiliza para indexar el diccionario, a partir de lo cual aparecen otros dos casos. (1) Si se produce un acierto en el diccionario, siendo su entrada la asociada a la clase $c_p{}'$, su valor de confianza se suma por un valor de salto de confianza (como llamamos a tal recompensa): $k(c_p{}') \leftarrow k(c_p{}') + J$. (2) Si se produce un fallo en el diccionario, se escribe el nuevo delta $d_p{}'$ en un entrada nueva cuya clase es $c_p{}'$, y un valor de confianza inicial igual a la mitad del máximo, $\frac{1}{2}k_{max}$. Las confianzas del resto de entradas ven su valor decrementado en uno.
Una vez que la clase $c_p{}'$ ha sido asociada al nuevo delta $d_p{}'$, se puede formar una nueva secuencia de clases y almacenarla en el buffer de entrada. En concreto, en el caso de que hubiera un acierto previamente al acceder al buffer de entrada, la nueva clase se empuja en la secuencia de clases anterior, que se almacenaba en la entrada correspondiente del buffer de entrada, siendo la nueva secuencia igual a $q_p{}' = [q_p(1), ..., q_p(n_q - 1), c_p{}']$. Por el contrario, si hubo un fallo en el búfer de entrada, es necesario asignar una nueva entrada donde la secuencia se establece como $q_p{}' = [n_c, ..., n_c, c_p{}']$. Además, tanto si se acierta como si se falla en el búfer de entrada, se escribe la dirección actual $a_p{}'$ como la última dirección de acceso de la entrada.
Por último, si la predicción para el PC p anterior a esta fase de ajuste no acertó, es decir, calculó una dirección $\hat{a_p}{}'$ distinta de la verdadera $a_p{}'$, se realiza un ajuste en la SVM. El modelo se ajusta con una muestra en la que la entrada es igual a la antigua secuencia de clases q_p y la salida es la nueva clase del diccionario $c_p{}'$.

En resumen, nuestra propuesta puede verse como una mezcla entre el citado DFCM y los predictores basados en Machine Learning, incluyendo todo un conjunto de optimizaciones con el objetivo de minimizar los costes y maximizar la precisión. De hecho, estas modificaciones pretenden centrar el BufferSVM en ajustes de contexto local y alta tasa de aprendizaje, en los que el predictor aprenderá vorazmente patrones que serán rápidamente reemplazados por otros patrones nuevos, lo que está en línea con la naturaleza dinámica de los patrones de acceso a memoria de un programa. De hecho, se comprueba en los resultados discutidos en la Sección V que el enfoque propuesto es exitoso.

V. Experimentación

En esta sección se presentan y discuten los resultados experimentales, validando la eficacia del modelo BufferSVM para la predicción de direcciones de memoria accedidas en aplicaciones reales del benchmark SPEC ([15]), y mostrando comparaciones detalladas entre los predictores de referencia y el propuesto (en sus versiones ideales).

A. Configuración del experimento

El proceso de experimentación consta de dos pasos: en primer lugar, se registran las direcciones de acceso a memoria instrumentando un grupo de aplicaciones, y en segundo lugar, se utilizan tales accesos registrados como conjunto de datos para los modelos de predictor.

El primer paso se realiza utilizando Intel Pin ([16]) para instrumentar la ejecución de los benchmarks SPEC 2017. Para ello, se registra el PC y la dirección de memoria de cada instrucción de acceso a memoria ejecutada en una CPU real. El resultado de este proceso dependerá del mapeo de memoria virtual asignado por el sistema operativo. En nuestro caso, este paso se lleva a cabo en un portátil Intel Core i7-10750H (2,60 GHz, 6 núcleos, cachés de datos de 6x32 kB, caché de nivel 2 de 6x256 kB, caché de nivel 3 de 12 MB) utilizando Ubuntu 18.04.

El segundo paso está programado y encapsulado como un proyecto C++. Este proyecto incluye todos los modelos de predictores y las tareas de simulación que emulan el comportamiento de los predictores en sus versiones ideales. La salida de estas tareas no depende del host, por lo que la información de entorno no es relevante en este paso.

B. Método de construcción del dataset

Un análisis más detallado del primer paso del proceso de experimentación revela que esta tarea no es trivial desde el punto de vista computacional, debido a la enorme cantidad de instrucciones de acceso a memoria que se ejecutan durante la ejecución de una aplicación SPEC.La tabla I muestra las aplicaciones seleccionadas en este trabajo para la experimentación, su tipo SPEC (entero, INT, o coma flotante, FP), y su correspondiente número de accesos a memoria. Este último número oscila entre $5,40 \times 10^9$ y $1,974 \times 10^{12}$ instrucciones para los diversos benchmarks, cubriendo una cantidad total de $7,72 \times 10^{12}$ accesos.

Tabla I: Aplicaciones de SPEC empleadas

Aplicación SPEC	**Tipo**	**Nº instr. acceso (10^9 instr.)**
perlbench	INT	300,2
gcc	INT	229,6
mcf	INT	623,9
lbm	FP	1974,0
omnetpp	INT	451,5
xalancbmk	INT	490,7
x264	INT	482,5
deepsjeng	INT	732,6
exchange2	INT	1393,3
leela	INT	623,4
roms	FP	5,4
cactuBSSN	FP	387,5

Construir un conjunto de datos de este tipo con un total de $7,7 \times 10^{12}$ de accesos no es factible, ni en almacenamiento, ni en el tiempo de procesamiento posteriormente requerido. Para hacer frente a esta

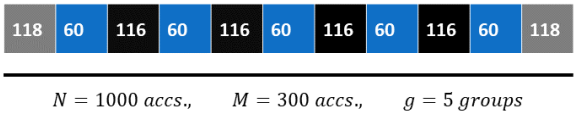


Fig. 2: Ejemplo de distribución de accesos para un programa de $N = 1000$ accesos en total. Los grupos de accesos, la separación entre ellos y los rellenos inicial y final aparecen como bloques azules, negros y grises respectivamente.

importante restricción, hemos implementado un modelo de selección de la carga de trabajo sencilla pero equitativa, con el fin de reducir significativamente el número de accesos y cubrir uniformemente la enorme carga de trabajo original.

Para cada aplicación SPEC, se extrae una traza de un total de $M = 10^9$ accesos, y se almacenan: (1) el PC de la instrucción, (2) un bit que indica si el acceso ha sido lectura o escritura, y (3) la dirección de memoria virtual. Tal cantidad de muestras da como resultado un archivo de trazas de casi 30 GB.

Para representar uniformemente los patrones de acceso globales de cada aplicación objetivo, se ha elegido una distribución uniforme, es decir, se han registrado un total de $g = 1000$ grupos, cada uno de $m = 10^6$ accesos, que están separados entre sí por una distancia de D accesos. Además, tanto al principio como al final de la traza completa, se deja sin registrar una secuencia de P accesos, considerada como relleno. Para un programa con una secuencia total de N accesos, estas dos últimas distancias pueden calcularse mediante las siguientes ecuaciones:

$$D = \left\lfloor \frac{N - M}{g + 1} \right\rfloor, \tag{8}$$

$$P = \left\lfloor \frac{(N - M) - D \times (g - 1)}{2} \right\rfloor \tag{9}$$

Un ejemplo sencillo de distribución siguiendo este modelo para $N = 1000$, $M = 300$, $g = 5$ se muestra en la Figura V-B.

Un interesante aspecto de recoger accesos en grupos separados de trazas es que se puede reproducir aproximadamente el comportamiento del cambio de contexto de procesos de una CPU. Es decir, al utilizar esta carga de trabajo, cada secuencia de m accesos puede percibirse como los accesos a memoria que tienen lugar tras asignar un proceso a la CPU y hasta su posterior desalojo (que será llevado a cabo por el planificador del sistema operativo). Este aspecto será importante en la siguiente subsección ya que se asumirá que este comportamiento está ocurriendo, y, por tanto, el predictor y sus parámetros se reiniciarán en cada cambio de grupo de trazas.

Aunque este método de registro de conjuntos de datos permite una aproximación mucho menos costosa para la extracción de trazas tanto en tiempo como en costes de almacenamiento, hay que aclarar que todo el proceso sigue requiriendo muchas horas de ejecución y devuelve cientos de gigabytes de datos. Mientras que nuestro método es totalmente justo y, además, emula la conmutación de procesos, existen otros métodos alternativos de construcción de conjuntos de datos en la literatura actual ([17], [18]) que también alivian el coste. Por ejemplo, cada traza de aplicación SPEC puede muestrearse siguiendo puntos de control previamente ajustados ([19]). A continuación, para cada punto de control, se puede proporcionar un peso como métrica de importancia: cuanto mayor sea su valor, más importancia se da a la secuencia de accesos que están presentes en el punto de control. Debido a los altos costes de experimentación, se deja para futuros trabajos una comparación completa entre nuestro método y el previamente mencionado.

C. Resultados

Se han seleccionado dos modelos de referencia para evaluar el rendimiento del predictor propuesto: un DFCM ideal y un Buffer-SVM ideal.

La elección de tales predictores se debe a tres motivos. En primer lugar, no es objeto de este trabajo el desarrollo de una propuesta con una implementación final, sino una evaluación preliminar. Segundo, el modelo DFCM es uno de los modelos de referencia en toda la literatura de predicción de valores. Aunque el VTAGE (o el DVTAGE) es otro predictor también de relevancia, no puede usarse en nuestro marco de trabajo ya que no extraemos los eventos de saltos tomados y no-tomados de los programas del SPEC (véase la Sección V-B). Tercero, los autores de los predictores propuestos en [9], [10], [11] no proporcionan ningún repositorio ni recurso para poder usar aquí su predictor.

Se implementaron versiones ideales de estos dos predictores y se probaron para los conjuntos de datos de acceso a memoria previamente almacenados. Como se ha indicado anteriormente, cada predictor evaluado restablece sus parámetros en cada cambio de grupo (véase la sección V-B).

El aspecto clave que confiere a ambos modelos de referencia su comportamiento ideal es su memoria infinita: a las tablas de los dos predictores de referencia se les ha dotado de una capacidad ilimitada, de modo que sólo podrían producirse fallos de caché forzosos (*compulsory*). De hecho, la precisión de DFCM y BufferSVM depende en gran medida de la tasa de aciertos alcanzada por sus tablas (cachés). Por tanto, tener memoria infinita implica que el predictor DFCM sólo fallará cuando aparezca un nuevo PC o un nuevo valor hash (véase la Sección II). Se han evaluado dos variaciones de este predictor, descritas en detalle en [20] y [21], y que se etiquetan aquí como:

- *HashOnHash*: El nuevo valor hash se calcula como el hash entre el valor hash anterior y el delta resultante (la diferencia entre la dirección actual y la anterior).
- *Orden-8*: El delta resultante se introduce en una secuencia de ocho deltas almacenados en la entrada accedida de la primera tabla. A continuación, el valor hash se calcula como el hash de los ocho deltas almacenados en esta entrada.

BufferSVM sigue un patrón similar con dos tablas:

el búfer de entrada y el diccionario, afectando el acceso al primero al acceso del segundo. Sin embargo, en el BufferSVM sólo se implementa memoria infinita en el buffer de entrada, ya que el diccionario debe ser de tamaño fijo, alineado con la salida de la SVM. En concreto, su entrada es una secuencia de clases de longitud $n_q = 8$ (almacenada en una entrada del búfer de entrada) y su salida es una clasificación entre $n_c = 8$ clases (véase la Sección IV). El diccionario está configurado para un valor de confianza máximo de $k_{max} = 255$, lo que permite saltos/tramos de confianza de $J = 32$. Por lo tanto, los costes en capacidad de memoria del modelo SVM y del diccionario son de 288 y 64 bytes respectivamente (véase la Sección IV).

Tanto los resultados en precisión como la capacidad de memoria dinámica media necesaria para cada predictor y cada aplicación se muestran en la Fig. 3. Sólo se representa la capacidad de memoria total para el BufferSVM ideal porque es idéntico a la capacidad del buffer de entrada. Para las versiones ideales y comparando los valores medios, se muestra que BufferSVM supera a las dos implementaciones DFCM *HashOnHash* y *orden-8* en el porcentaje de aciertos de predicción en más de un 5 %, requeriendo una capacidad total de memoria necesaria un orden de magnitud menor. Concretamente, la tasa de aciertos nuestra propuesta en su versión ideal fue de $81,5 \pm 14,3$ %, mientras que en los casos de *HashOnHash* y *orden-8* en sus versiones ideales se obtuvo $75,8 \pm 15,9$ % y $71,9 \pm 13,9$ %, respectivamente.

La razón de esta mejora es intrínseca a la construcción del BufferSVM: mientras que el buffer de entrada es la única tabla de tamaño considerable que necesita BufferSVM, DFCM requiere, no sólo una primera tabla similar al buffer de entrada del BufferSVM, sino también una segunda que necesita una importante cantidad de memoria para indexar todos los posibles contextos de acceso, codificados como valores hash. Desde otra perspectiva, se deduce que el BufferSVM consigue comprimir los patrones de contexto de acceso mucho mejor que su homólogo DFCM, gracias a una SVM que mapea todos los patrones posibles en sólo ocho posibilidades (tantas como clases del diccionario).

Además, las bajísimas tasas de fallo del búfer de entrada que consigue BufferSVM en sus tablas permiten unas muy buenas tasas de acierto en la predicción. De hecho, la tasa de fallos del diccionario sólo supera el 20 % en dos pruebas (*mcf* y *omnetpp*), cayendo por tanto la mayor parte del peso de la tasa de aciertos del predictor casi exclusivamente sobre el acierto del SVM.Se puede inferir de la Fig. 3 que el rendimiento del SVM por sí solo, es superior a la tasa de acierto de la segunda tabla del DFCM, ofreciendo así un mapeo más preciso entre un contexto local y su delta a predecir.

Un análisis más detallado del modelo SVM revela que sólo un conjunto de operaciones lineales, relativamente fáciles de implementar en hardware (sólo se necesitan operaciones de multiplicación y suma vectorial/escalar), es capaz de lograr las altas tasas de precisión descritas anteriormente. Esto valida plenamente los modelos de predicción propuestos basados en clases dinámicas o palabras del diccionario linealmente separables en el espacio de 8 dimensiones, que maneja la SVM. Una explicación plausible de esta separabilidad sería que las secuencias de clases que se introducen en la SVM tienden a concentrarse en clústeres, que, de hecho, están situados en posiciones bien separadas en el espacio de entrada. Por lo tanto, la SVM estaría calculando las posiciones y orientaciones de los 8 hiperplanos de acuerdo con estas ubicaciones de los clústeres, lo que resultaría en una clara relación entre cada entrada y su salida. Aunque este comportamiento particular no se puede demostrar usando la implementación actual del simulador, en [9] ya se probó que tal clustering de secuencias de clases es muy efectivo.

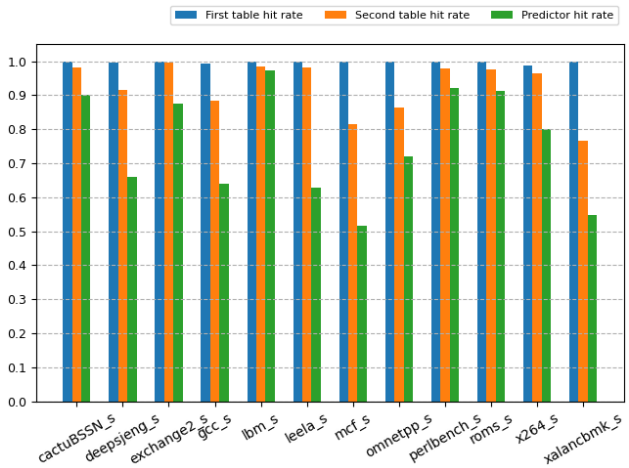
VI. Conclusiones y trabajo futuro

En este trabajo, se ha descrito detalladamente un nuevo modelo de predictor, BufferSVM, el cual ha sido evaluado en comparación con el predictor de referencia de la literatura DFCM. Los resultados han sido muy favorables para el BufferSVM, pues obtiene cerca de más de un 5 % de tasa de acierto más que el DFCM (usando versiones ideales de ambos) usando un orden de magnitud menos en capacidad de memoria requerida. Estos buenos resultados provienen, sin duda, de haber aprovechado exitósamente el paradigma de predicción a corto plazo mediante el uso del SVM y los búfferes dinámicos, lo cual se diferencia de otras propuestas de la literatura.

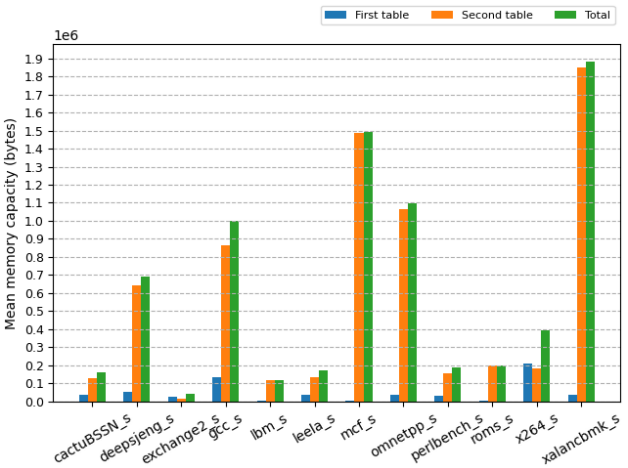
A pesar de que se han implementado tablas (cachés) infinitas que cumplían un rol similar para ambos predictores, y la comparación es, por ello, equitativa, queda pendiente como trabajo futuro una implementación realista restringida por las limitaciones reales de estos modelos. Cabe destacar que las pruebas presentadas dan ventaja al DFCM, pues su buena precisión se basa en disponer de tablas de memoria de capacidad suficiente. Por tanto, una comparación entre DFCM y BufferSVM con memoria limitada sería justa y podría mostrar una ventaja aún mayor a favor nuestra propuesta. Además, sería interesante una exploración de todo el espacio de configuraciones del predictor BufferSVM, de cara a compararlo incluso con la implementación ideal del DFCM aquí utilizada.

Más allá de las implementaciones emuladas en software, otro objetivo de peso sería una implementación hardware del predictor BufferSVM en una FPGA, con objeto de probar las prestaciones reales de un prefetcher que incorpore el predictor propuesto.

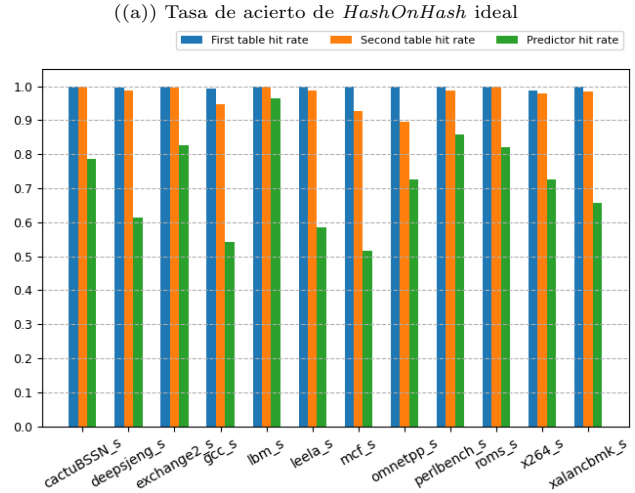
Adicionalmente, el diseño del BufferSVM supone un marco en el cual caben un conjunto de modificaciones y nuevas propuestas que permitirían mejorar los resultados de predicción o de cualquier otra métrica, como pueden ser el coste en memoria, la latencia en ciclos de reloj, el consumo energético, etc.



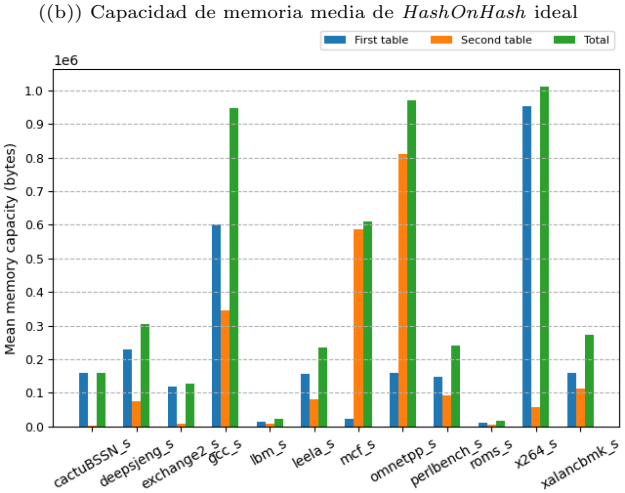
((a)) Tasa de acierto de *HashOnHash* ideal



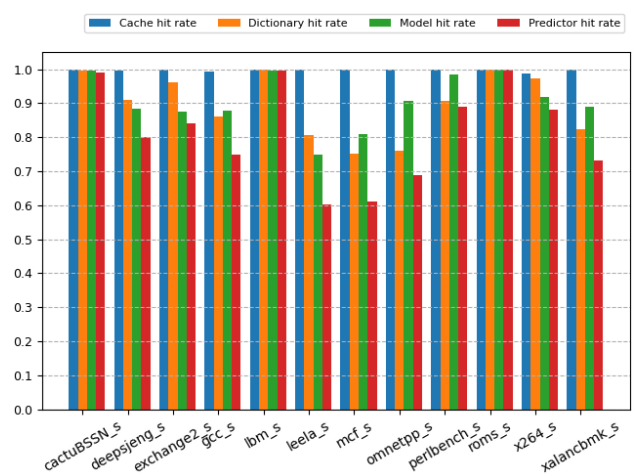
((b)) Capacidad de memoria media de *HashOnHash* ideal



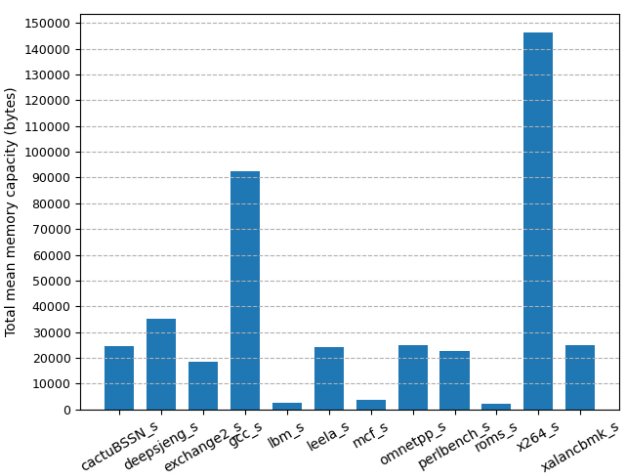
((c)) Tasa de acierto de *orden-8* ideal



((d)) Capacidad de memoria media de *orden-8* ideal



((e)) Tasa de acierto de Buffer-SVM ideal



((f)) Capacidad de memoria media de Buffer-SVM ideal

Fig. 3: Tasa de acierto y memoria total de los predictores de referencia ideales para los accesos de las aplicaciones de la Tabla I.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto nacional MINDROB (PID2019-105556GB-C33/AEI/10.13039/501100011033) y los proyectos PID2019-110455GB-I00 (AEI/FEDER,UE) y US-1381077 (JJAA/FEDER, UE).

Referencias

[1] John L. Hennessy and David A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 6th edition, 2017.

[2] Agustín Navarro-Torres, Biswabandan Panda, Jesús Alastruey-Benedé, Pablo Ibáñez, Víctor Viñals-Yúfera, and Alberto Ros, "Berti: an accurate local-delta data prefetcher," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 975–991.

[3] André Seznec and Pierre Michaud, "A case for (partially) tagged geometric history length branch prediction," *Journal of Instruction-level Parallelism - JILP*, vol. 8, 02 2006.

[4] Amir Roth, Ronny Ronen, and Avi Mendelson, "Dynamic techniques for load and load-use scheduling," *Proceedings of the IEEE*, vol. 89, pp. 1621 – 1637, 2001.

[5] Lois Orosa, Rodolfo Azevedo, and Onur Mutlu, "Avpp: Address-first value-next predictor with value prefetching for improving the efficiency of load value prediction," *ACM Transactions on Architecture and Code Optimization*, vol. 15, pp. 1–30, 2018.

[6] Freddy Gabbay and Avi Mendelson, "Using value prediction to increase the power of speculative execution hardware," *ACM Transactions on Computer Systems*, vol. 16, 1998.

[7] Arthur Perais and André Seznec, "Bebop: A cost effective predictor infrastructure for superscalar value prediction," *2015 IEEE 21st International Symposium on High Performance Computer Architecture, HPCA 2015*, 02 2015.

[8] Bart Goeman, Hans Vandierendonck, and Koen De Bosschere, "Differential fcm: increasing value prediction accuracy by improvingtable usage efficiency," 02 2001, pp. 207–216.

[9] Milad Hashemi, Kevin Swersky, Jamie A. Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan, "Learning memory access patterns," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, Jennifer G. Dy and Andreas Krause, Eds. 2018, vol. 80 of *Proceedings of Machine Learning Research*, pp. 1924–1933, PMLR.

[10] Ajitesh Srivastava, Aggelos Lazaris, Benjamin Brooks, Rajgopal Kannan, and Viktor Prasanna, "Predicting memory accesses: the road to compact ml-driven prefetcher," 2019, pp. 461–470.

[11] Pengmiao Zhang, Ajitesh Srivastava, Benjamin Brooks, Rajgopal Kannan, and Viktor K. Prasanna, "Raop: Recurrent neural network augmented offset prefetcher," in *The International Symposium on Memory Systems*, New York, NY, USA, 2021, MEMSYS 2020, p. 352–362, Association for Computing Machinery.

[12] Sangwook Kim, Swathi Kavuri, and Minho Lee, "Deep network with support vector machines," in *Neural Information Processing: 20th International Conference, ICONIP 2013, Daegu, Korea, November 3-7, 2013. Proceedings, Part I 20*. Springer, 2013, pp. 458–465.

[13] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan, "Learning memory access patterns," 2018.

[14] Matthew Farrens, Benjamin Culpepper, and Mark Gondree, "Svms for improved branch prediction," 01 2004.

[15] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski, "Spec cpu2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA, 2018, ICPE '18, p. 41–42, Association for Computing Machinery.

[16] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," *SIGPLAN Not.*, vol. 40, no. 6, pp. 190–200, jun 2005.

[17] E. Perelman, Greg Hamerly, and B. Calder, "Picking statistically valid and early simulation points," 10 2003, pp. 244– 255.

[18] Qinzhe Wu, Steven Flolid, Shuang Song, Junyong Deng, and Lizy K. John, "Invited paper for the hot workloads special session hot regions in spec cpu2017," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018, pp. 71–77.

[19] Qinzhe Wu, Steven Flolid, Shuang Song, Junyong Deng, and Lizy K. John, "Hot regions in spec cpu2017," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2018.

[20] B. Goeman, H. Vandierendonck, and K. de Bosschere, "Differential fcm: increasing value prediction accuracy by improving table usage efficiency," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, 2001, pp. 207–216.

[21] Yiannakis Sazeides and James E. Smith, "Implementations of context based value predictors," 1997.

Redes y Comunicaciones

Optimización de la Transmisión de Vídeo en Redes Vehiculares

P. Pablo Garrido Abenza, Pablo Piñol Peral, Manuel P. Malumbres, O. López Granado [1]

Resumen— **El funcionamiento del mecanismo multicanal del estándar IEEE 1609.4 utilizado en redes vehiculares puede tener un gran impacto en el rendimiento de la red si no se tiene en cuenta la alternancia entre el canal de control (CCH) y el canal de servicio (SCH). Los paquetes enviados desde la capa de aplicación durante el intervalo del canal de control tienen que esperar en la correspondiente cola de la capa MAC, y no serán enviados hasta que comience el intervalo del canal de servicio. Como consecuencia se produce una acumulación de paquetes, que se intenta transmitir al comienzo de dicho canal de servicio. Puesto que todos los nodos de la red están sincronizados del mismo modo, todos los nodos intentan transmitir simultáneamente, produciéndose altas tasas de colisiones debido a la transmisión de estas ráfagas de paquetes. Este problema es aun más notorio cuando se utiliza calidad de servicio (QoS), es decir, cuando se priorizan ciertos paquetes (audio o vídeo) según la recomendación del estándar IEEE 802.11p. En este trabajo se propone un mecanismo denominado SkipCCH que tiene en cuenta la alternancia de los canales de control y servicio a la hora de la planificación del envío de paquetes. A la vista de los resultados de los experimentos realizados, con este mecanismo se reducen las colisiones que se producen al comienzo del canal de servicio y se obtiene un mejor aprovechamiento del canal inalámbrico, mejorando el rendimiento de la red. El mecanismo mejora aún más el rendimiento de la red cuando se utiliza calidad de servicio, incrementando la calidad del vídeo recibido, sin perjudicar al resto de tráfico menos prioritario.**

Palabras clave— **MAC, Vídeo, Redes Vehiculares**

I. Introducción

LOS sistemas de transporte inteligente, o ITS (Intelligent Transportation Systems), ofrecen diversas las aplicaciones y servicios. Algunas de ellas requieren la transmisión de vídeo, tales como videovigilancia, información turística, entretenimiento (*infotainment*), o seguridad en carretera. Sin embargo, la transmisión de vídeo sobre redes inalámbricas tiene diversos problemas, debido al uso del canal inalámbrico compartido con un ancho de banda limitado, así como la existencia de ciertos fenómenos como la atenuación de la señal, la presencia de obstáculos, etc. Además, en el caso de las redes vehiculares se suman los continuos cambios en la topología debido a la alta movilidad de los vehículos. Aunque el uso de compresión de vídeo permite reducir la cantidad de datos a transmitir, la calidad del vídeo percibida por el usuario puede verse muy afectada por los paquetes perdidos durante su transmisión.

En la pila de protocolos WAVE (Wireless Access in Vehicular Environment) utilizada en redes vehiculares, o VANETs (Vehicular Ad-Hoc Networks), la capa MAC (Medium Access Control) está compuesta por los protocolos IEEE 802.11p y IEEE 1609.4.

El primero de ellos implementa calidad de servicio (QoS) mediante EDCA (Enhanced Distributed Channel Access), que permite la diferenciación de tráfico en cuatro tipos de tráfico, o Access Categories (ACs), que de mayor a menor prioridad son: AC_VO (voz), AC_VI (vídeo), AC_BE (*best effort*), y AC_BK (*background*). Cada una de estas categorías tiene una serie de parámetros (Tabla I), como la ventana de contención, o Contention Window (CW), y el Arbitration Interface Space Number (AIFSN), y organizan los paquetes en su propia cola antes de su envío.

Tabla I: Parámetros EDCA del estándar IEEE 802.11p.

AC	$CW_{min..max}$	AIFSN	TXOP limit
AC_BK	15..1023	9	0 ms
AC_BE	15..1023	6	0 ms
AC_VI	7..15	3	0 ms
AC_VO	3..7	2	0 ms

Por otro lado, el protocolo IEEE 1609.4 del estándar WAVE se encarga del funcionamiento multicanal, alternando entre un canal de control (CCH), reservado para el envío de mensajes de seguridad y *beacons*, y un canal de servicio (SCH), utilizado para la transmisión de datos de aplicaciones de usuario. Aunque ambos canales utilizan frecuencias distintas, no transmiten simultáneamente, sino que se alternan en intervalos de tiempo (*slots*) de 50 ms. Puesto que todos los nodos de la red están sincronizados, al comienzo de cada *slot* se reserva un intervalo de guarda de 4 ms para garantizar que todos los nodos hayan realizado el cambio de canal. Este mecanismo se ilustra en la parte superior de la figura 1, donde se puede observar claramente la alternancia entre los canales CCH y SCH.

La transmisión de vídeo con una mayor prioridad (AC_VI) permite la protección de los paquetes de datos de vídeo, que pueden ser críticos en ciertas aplicaciones, tales como seguridad en carretera (adelantamientos asistidos) o aplicaciones de emergencia (vídeo llamadas automáticas en caso de accidente). En experimentos realizados en este contexto se observó que en ciertos casos, al contrario de lo que se esperaba, con la asignación de una mayor prioridad a los paquetes de vídeo se obtenía una mayor tasa de pérdidas que para el resto de tráfico menos prioritario como AC_BE o AC_BK. Asimismo, se observó que la mayor parte de estas pérdidas se producían al comienzo de cada *slot* SCH, puesto que se intentaban enviar todos los paquetes acumulados en la cola de la capa MAC durante un *slot* CCH al mismo tiempo, al principio del siguiente *slot* de servicio (SCH), y esto ocurre en todos los nodos de la red. Por tanto,

[1]Dpto. de Ingeniería de Computadores, Universidad Miguel Hernández, Elche (Alicante), e-mail: {pgarrido, pablop, manuel.perez, otoniel}@umh.es.

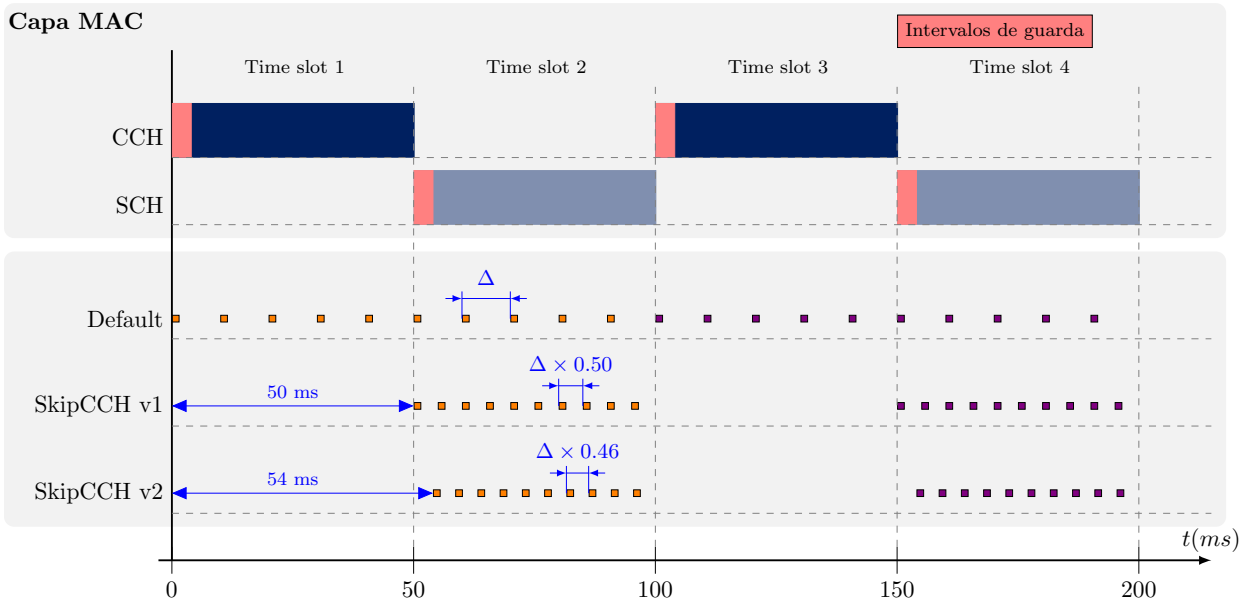


Fig. 1: Funcionamiento del mecanismo multicanal de la arquitectura WAVE (MAC layer), y planificación de paquetes según el mecanismo predeterminado (*Default*) y dos versiones del mecanismo propuesto (*v1* y *v2*).

la alternancia de canal tiene un efecto negativo de sincronización entre los nodos, que produce una tormenta de envíos de mensajes al principio de cada *slot* SCH. Este efecto provoca un incremento significativo de colisiones en el canal inalámbrico que incrementa sensiblemente la tasa de paquetes perdidos. Además, en el caso particular de la transmisión de vídeo, las pérdidas de paquetes debidas a estas ráfagas tiene un mayor impacto en el vídeo reconstruido. Debido a las características del vídeo comprimido, la resiliencia contra pérdidas de paquetes se reduce, es decir, a una misma tasa de pérdidas, si se trata de paquetes aislados tiene un menor impacto que si se trata de una ráfaga de paquetes.

Con objeto de mitigar este problema, se desarrolló un mecanismo denominado SkipCCH, el cual se presenta en este trabajo. Se analiza su efecto en la transmisión de vídeo codificado mediante *High Efficiency Video Coding* (HEVC) [1] sobre redes vehiculares.

El artículo está organizado de la siguiente forma. En la sección I, se introduce el problema y los estándares utilizados. En la sección II se presentan algunos trabajos relacionados. La sección III describe en detalle el mecanismo SkipCCH propuesto, con dos ligeras variantes. Después, se describen los experimentos realizados en la sección IV, y los resultados obtenidos en la sección V. Por último, en la sección VI se concluye el trabajo.

II. Trabajos relacionados

El problema descrito en la sección anterior también ha sido estudiado en trabajos como [2], aunque no para aplicaciones de transmisión de vídeo, las cuales tienen un mayor requerimiento de ancho de banda.

En [3] se utiliza calidad de servicio (QoS), y se propone un esquema para adaptar dinámicamente los valores de AIFSN a las condiciones de la red, con objeto de reducir las colisiones y mejorar el rendimiento de la red para la transmisión de tráficos de voz y vídeo. Sin embargo, se utiliza el estándar IEEE 802.11e, es decir, no está destinado a escenarios de redes vehiculares.

En estudios como [4] sí que se analizó el rendimiento del estándar IEEE 802.11p utilizado en la pila de protocolos WAVE, llegando a la conclusión de que el rendimiento de la red era menor y el retardo mayor en los paquetes a los que se les había asignado una mayor prioridad. Dicho trabajo investigó analíticamente y mediante simulación la probabilidad de colisión según la prioridad asignada, mostrando que las categorías de mayor prioridad (AC3 o AC_VO) tenían una mayor probabilidad de colisión que las de menor prioridad (AC0 o AC_BK), debido a su menor ventana de contención (ver Tabla I). Como consecuencia, se produce una mayor tasa de pérdidas y se reduce el rendimiento de la red, afectando especialmente a las aplicaciones que requieren la transmisión de vídeo.

En la literatura también se ha encontrado alguna propuesta, como en [5] y [6], en donde los autores analizan el caso para la transmisión de vídeo, y proporcionan un mecanismo denominado VoV, el cual evita la transmisión de paquetes de vídeo mientras el canal CCH esté activo. Sin embargo, no se tiene en cuenta que la tasa de envío de tramas debe ser diferente al utilizar únicamente uno de los *slots* (SCH) en lugar de ambos canales (*sync interval*), necesario para mantener la calidad necesaria en el vídeo recibido. Además, tampoco se considera el caso de utilizar calidad de servicio (QoS), donde el efecto de la sincronización provocaba un mayor número de pérdida de paquetes, a pesar de asignar una mayor prioridad a los paquetes de vídeo que al resto de tráfico.

En la siguiente sección se describe el mecanismo propuesto, que tiene en cuenta, tanto el canal activo en la capa MAC como la tasa de envío equivalente, necesaria para aplicaciones de transmisión de vídeo.

III. Mecanismo propuesto

En este trabajo se propone un mecanismo denominado SkipCCH para mitigar el problema descrito, que consiste en calcular los instantes de planificación de envío de paquetes teniendo en cuenta la alternancia de canales que sucede en la capa MAC, para enviarlos únicamente durante el *slot* correspondiente al SCH, distribuyéndolos en el tiempo disponible. A continuación, primero se describe el mecanismo por defecto (*Default*), es decir, sin tener en cuenta el canal activo. Después, se describe el mecanismo propuesto con dos ligeras variaciones, sin tener en cuenta los intervalos de guarda (*v1*) y teniéndolos en cuenta (*v2*).

A. Mecanismo por defecto: Default

En el mecanismo por defecto (*Default*), es decir, en caso de no tener en cuenta la alternancia de canal de la capa MAC, el funcionamiento sería el siguiente. Para la transmisión de una secuencia de vídeo a una determinada tasa de *bits* por segundo (*bitrate*) se requiere calcular el tiempo entre paquetes Δ (*inter-packet time*). Para calcular este tiempo tendríamos dos opciones: (a) En caso de vídeo almacenado (VoD), dividimos la duración de la secuencia por el número total de paquetes; (b) para el vídeo en vivo (*Life Video*) se podría calcular basándose en tasa de *bits* (*bitrate*) objetivo y el tamaño máximo de un paquete. Si el instante inicial del envío del vídeo es t_0, el primer paquete se planificará en el instante $t_1 = t_0$, y los siguientes paquetes incrementando el valor Δ ($t_2 = t_1 + \Delta$, $t_3 = t_2 + \Delta$, ..., en general, $t_i = t_{i-1} + \Delta$). La representación gráfica de este mecanismo de planificación se muestra en la parte inferior la figura 1, etiquetada como '*Default*'. Su implementación se muestra en el diagrama de flujo de la figura 2, siguiendo la rama 'SkipCCH? > Off'. En este mecanismo, si un paquete es planificado en un instante t_i que corresponde con un *slot* SCH (*slots* 2, 4, 6, 8, ...), el paquete podría ser transmitido en ese instante. Sin embargo, si el instante t_i se corresponde con un *slot* CCH (*slots* 1, 3, 5, 7, ...), el paquete tendrá que esperar en la cola correspondiente de la capa MAC hasta que se inicie el siguiente *slot* SCH, y se transmitirá en ráfaga junto con otros paquetes almacenados en la cola, produciéndose un elevado número de colisiones y paquetes perdidos.

B. Mecanismo propuesto: SkipCCH v1

Para evitar el problema mencionado, en este trabajo se propone un mecanismo, denominado SkipCCH, que consiste en la planificación de paquetes de forma sincronizada con el *slot* SCH dentro de la capa MAC, de tal manera que se planifica el envío de los paquetes de vídeo únicamente a lo largo de los *slots* SCH. Puesto que ahora se dispone de la mitad del tiempo para el envío del mismo número de paquetes, con el fin de mantener la misma tasa de envío de *frames* por segundo, el intervalo entre paquetes debe ser la mitad, es decir, $\Delta \times 0{,}50$. Por tanto, para calcular el instante de planificación de un paquete respecto el anterior enviado sería, en general, $t_i = t_{i-1} + (\Delta \times 0{,}50)$. Además, si el instante t_i calculado para el envío de un paquete corresponde a un *slot* CCH, éste se incrementará 50 ms, con objeto de que se planifique en el siguiente *slot* SCH.

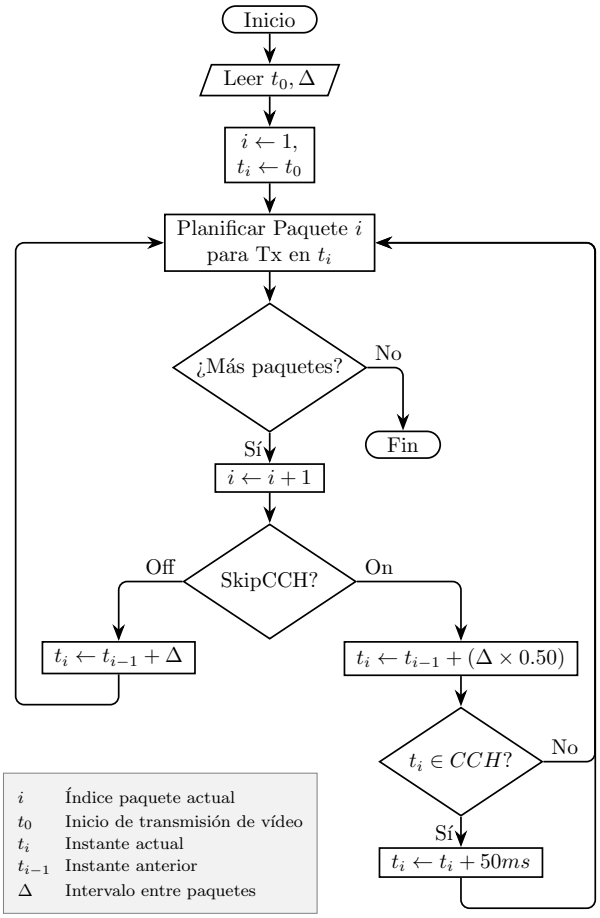


Fig. 2: Diagrama de flujo del mecanismo predeterminado y el mecanismo propuesto (*v1*).

En la figura 1 puede verse gráficamente los instantes en los que se planificaría el envío de una serie de paquetes con los distintos mecanismos. Esta primera versión del mecanismo se visualiza en el diagrama de flujo de la figura 2, siguiendo la rama 'SkipCCH? > On'.

C. Mecanismo propuesto: SkipCCH v2

El mecanismo propuesto en la sección anterior no tiene en cuenta los intervalos de guarda (4 ms), que no se utilizan para la transmisión de paquetes, quedando efectivos 46 ms en los *slots* en lugar de 50 ms. Por tanto, se modificó ligeramente el mecanismo para tener en cuenta los 4 ms del intervalo de guarda. En concreto, se modificó lo siguiente: (1) el intervalo entre paquetes debe ser algo menor, $t_i = t_{i-1} + (\Delta \times 0{,}46)$, y (2) el incremento para evitar los *slots* CCH debe ser 54 ms (en lugar de 50 ms).

Aunque esta versión no se muestra en el diagrama de flujo de la figura 2, sería el mismo que la versión *v1*, cambiando únicamente esos dos valores. En los resultados obtenidos se realiza una comparación del mecanismo predeterminado con las dos versiones del mecanismo propuesto.

IV. Entorno de simulación

Con objeto de evaluar el mecanismo propuesto, se utilizaron los siguientes elementos: SUMO v1.8.0 (Simulation of Urban MObility) [7] como simulador de tráfico, y OMNeT++ v5.6.2 [8] junto con Veins v5.1 (VEhicles In Network Simulation) [9] como simulador de red. El escenario consistió en un área rectangular de 2000 × 2000 m. de la ciudad de Las Vegas (EE.UU.), descargado desde OpenStreetMap [10] (figura 3). El escenario incluye un *Road-Side Unit* (RSU) en el centro (`rsu[0]`) que actúa como un elemento fijo que transmite una secuencia de vídeo de forma cíclica. Circulando por la zona se definen varios vehículos, uno que recibe la secuencia de vídeo (`node[0]`), otros 10 vehículos (`node[1..10]`) que generan tráfico de ruido de fondo (*background*) a distintas tasas (0, 12, 25, 50, 75 y 100 pps), y un número de vehículos (`node[11..60]`) que simplemente circulan por la zona. Todos los vehículos se mueven a una velocidad máxima de 14 m/s (50 km/h). El tiempo total de simulación se estableció en 200 segundos.

Para la codificación de vídeo se utilizó el estandar *High Efficiency Video Coding* (HEVC) [1], y como codificador y decodificador de vídeo, el software de referencia HM (*HEVC Test Model*) [11] v9.0. La secuencia de vídeo utilizada fue 'BasketBallDrill', que pertenece a la colección de secuencias de la *Common Test Conditions* del estándar HEVC [11]. Esta secuencia tiene una resolución de 832×480 píxeles, una tasa de 25 fps (*frames per second*), y una longitud de 250 *frames*, es decir, una duración de 10 segundos. Se codificó con el modo All Intra (AI), con un factor de cuantificación, o *Quantization Parameter* (QP), de 31 para conseguir un *bitstream* de buena calidad de vídeo, con un valor medio de PSNR (Peak Signal-to-Noise Ratio) de ≈36 dB, y un *bitrate* de 3,4 Mbps.

Todo el proceso fue gestionado mediante el entorno *Video Delivery Simulation Framework over Vehicular Networks* (VDSF-VN) [12], que consta de dos aplicaciones gráficas: *GatcomSUMO*, para la generación de los escenarios y movilidad de los vehículos, y *GatcomVideo*, para la codificación de vídeo, ejecución de simulaciones, y generación de gráficos con las estadísticas de red y la evaluación de la calidad de las secuencias de vídeo recibidas.

V. Resultados

El uso de QoS es altamente recomendable para transmisión de vídeo, sobre todo para aplicaciones críticas y seguridad en carretera. En trabajos previos [13] se estudió la transmisión de vídeo en VANETs aplicando técnicas de QoS para priorizar los paquetes de la secuencia de vídeo codificada, mostrando una mejora significativa en el rendimiento de la red, reduciendo la tasa de pérdidas de paquetes y mejorando la calidad final del vídeo reconstruido.

En este trabajo se cuantifica la ganancia de rendimiento cuando activamos la QoS para la transmisión de vídeo. Además verificamos el impacto que tiene el empleo del mecanismo propuesto, SkipCCH, tanto si usamos QoS como si no la utilizamos. A continuación se presentan y se discuten los resultados obtenidos de esta comparativa, y para distintas tasas de tráfico de fondo (desde 0 hasta 100 pps). Se da respuesta a las siguientes preguntas: (1) ¿Cómo afecta activar QoS a los paquetes de vídeo?, (2) ¿Qué impacto tiene utilizar QoS en el tráfico de fondo?, (3) Cuando activamos QoS, ¿el mecanismo propuesto SkipCCH incrementa el rendimiento de la transmisión de vídeo? Para ello se analizan varias estadísticas, entre las que se incluye: el número de entradas al proceso de *backoff* en el MAC, el número de paquetes recibidos de tráfico de fondo y de vídeo, el retardo (*End-to-End delay*), la variación del retardo (*jitter*), el porcentaje de paquetes perdidos, el porcentaje de *frames* de vídeo perdidos, así como métricas de calidad del vídeo reconstruido. Todas las estadísticas mostradas se corresponden con una de las secuencias de vídeo recibidas por el cliente (`node[0]`), en concreto, la recibida entre t=60s y t=70s.

La figura 4 muestra el número de veces que el servidor (`rsu[0]`) entra al proceso de *backoff* al encontrar el medio inalámbrico ocupado, aplicando la ventana de contención (CW). Podemos observar que cuando no se utiliza calidad de servicio (QoS=off), las tres curvas (SkipCCH=off, on (v1), on (v2)) tienen valores muy similares para todos los niveles de tráfico de fondo, aunque cuando el mecanismo SkipSCH está activado hay un ligero aumento cuando la carga de la red es media. Por otro lado, cuando sí se utiliza calidad de servicio (QoS=on) hay un incremento mucho más notable en el número de veces que se entra en *backoff*, mayor cuanto mayor es la carga de la red. Este resultado podría hacer pensar que al utilizar QoS se produce un menor aprovechamiento del canal, puesto que al detectar que está ocupado los paquetes deben esperar en la cola de la capa MAC antes de su transmisión. Sin embargo, es al contrario, puesto que, tal como se comentó anteriormente, la ventana de contención (CW) utilizada por las categorías de mayor prioridad (AC_VI y AC_VO) es mucho menor (Tabla I), por lo que se reintentará la transmisión mucho antes y los paquetes tendrán que esperar menos tiempo en total. Por tanto, se produce una optimización en el acceso al canal inalámbrico.

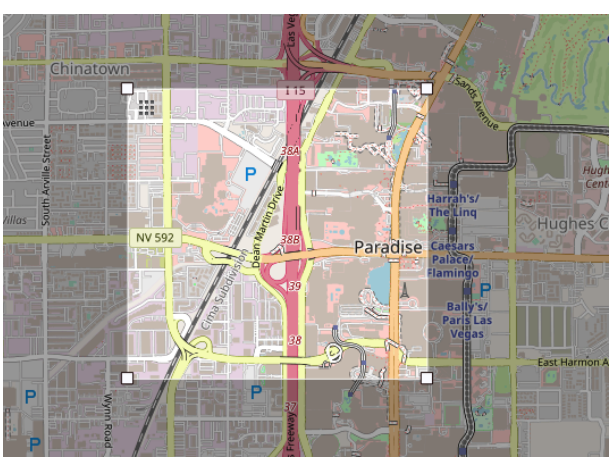


Fig. 3: Area seleccionada de la ciudad de Las Vegas (EE.UU.) en OpenStreetMap.

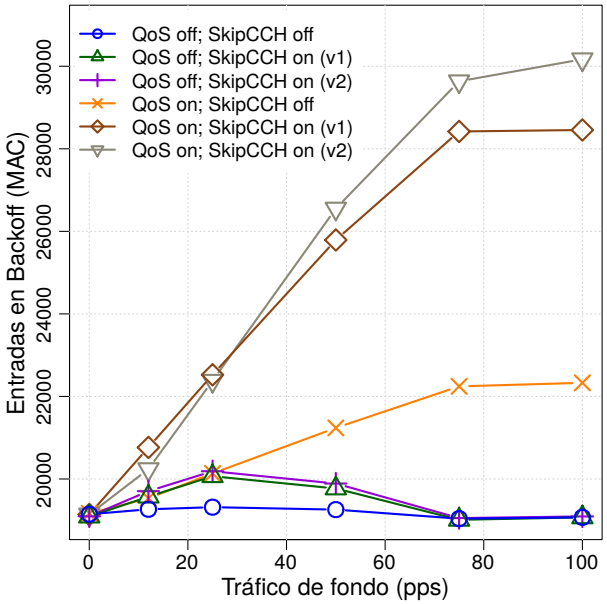
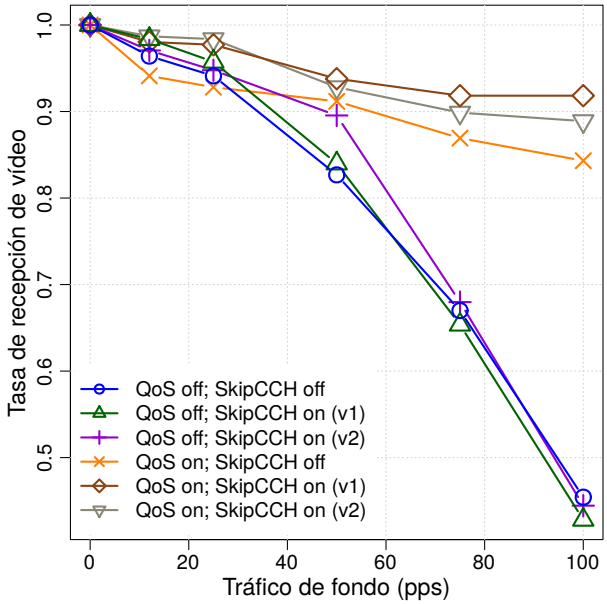
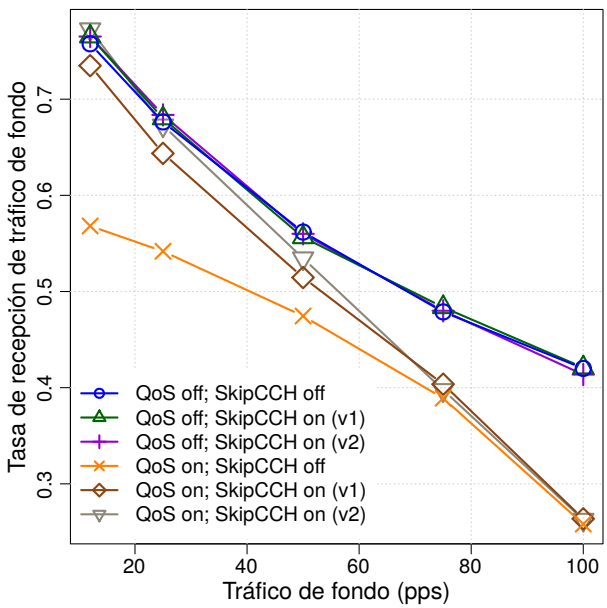


Fig. 4: Número de entradas al proceso de *backoff* (`rsu[0]`).



(a) Recepción de paquetes de vídeo



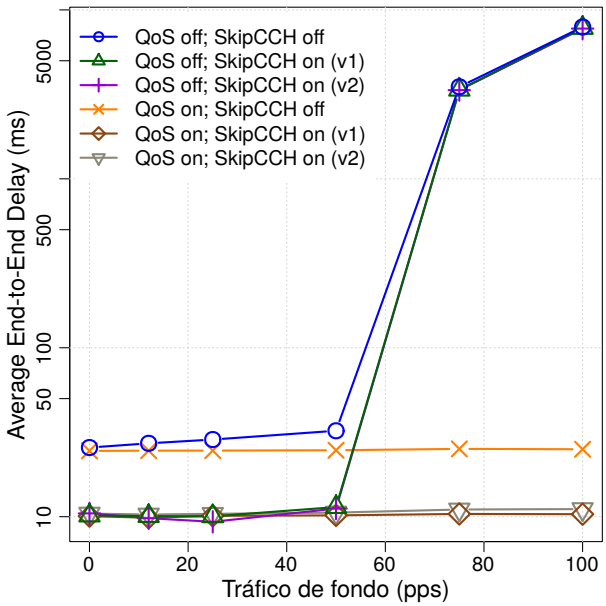
(b) Recepción de paquetes de tráfico de fondo

Fig. 5: Tasas de recepción de vídeo y tráfico de fondo (`node[0]`).

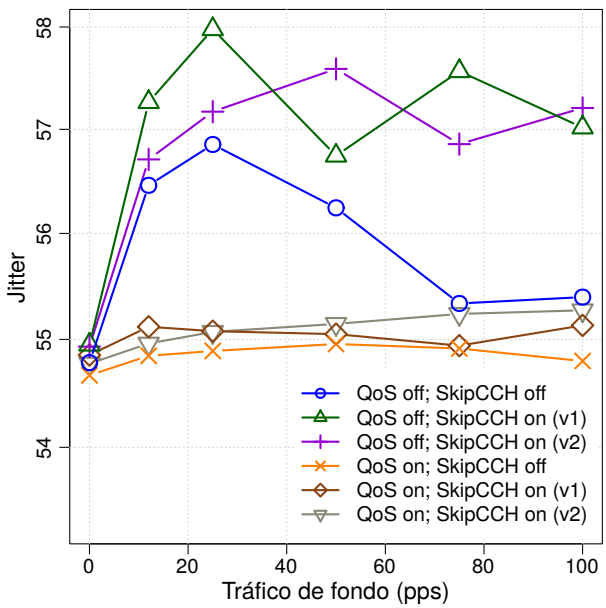
La figura 5 muestra las tasas de recepción de tráfico de vídeo y de tráfico de fondo (*background*) del nodo cliente (`node[0]`). Se puede observar que, al no utilizar calidad de servicio (QoS=off), tanto el tráfico de vídeo como el de *background* se ven muy afectados conforme la red se va saturando. Por otro lado, al utilizar calidad de servicio (QoS=on), el tráfico de vídeo tiene una mayor resistencia, muy notable para cargas medias y altas, a costa del tráfico de fondo. Sin embargo, se puede apreciar que el uso del mecanismo SkipCCH proporciona una mejora en la tasa de recepción de vídeo respecto al mecanismo por defecto, al tiempo que tiene un impacto mucho menor en el tráfico de fondo, es decir, se obtiene un mejor aprovechamiento del canal inalámbrico.

La figura 6a muestra el retardo (*End-to-End Delay)* del tráfico de vídeo transmitido por `rsu[0]`. Lo primero que se aprecia es que, cuando no se utiliza QoS, el retardo aumenta mucho cuando la red está bastante saturada, independientemente de si se utiliza el mecanismo SkipCCH o no, mientras que al utilizar QoS, el retardo se mantiene constante, puesto que los paquetes de vídeo tienen mayor oportunidad de transmisión que el tráfico de fondo. También se puede observar que cuando no se utiliza el mecanismo SkipCCH, el retardo es mayor (unos 15 ms por encima), tanto si se utiliza QoS como si no. No obstante, esto no quiere decir que el vídeo tarde más en transmitirse que cuando sí se utiliza SkipCCH, ya que la estadística considera el instante desde que el paquete se envía hacia la capa MAC para su transmisión; como ya se ha comentado, cuando el mecanismo SkipCCH está activo, los paquetes evitan el *slot* CCH, enviándose durante el siguiente *slot* SCH, por lo que ese tiempo no se contabiliza. Se utilice o no el mecanismo SkipCCH, en el peor de los casos, un paquete de vídeo tendrá que esperar más de 54 ms por la duración del *slot* CCH (50 ms) más el intervalo de guarda del siguiente *slot* SCH (4 ms). Por tanto, si una aplicación requiere un retardo menor de ese valor, no podrá utilizar el estándar WAVE [14]. Respecto al *jitter*, en la figura 6b se observa claramente valores menores y constantes (para cualquier condición de la red) con QoS activado respecto a cuando no, con curvas muy similares en los tres casos.

La figura 7 muestra los paquetes de vídeo perdidos debidos a errores de transmisión, así como los *frames* que no se han podido decodificar por la pérdida de alguno de sus paquetes. La pérdida de un único paquete puede provocar que un *frame* de vídeo ya no se pueda decodificar, siendo el mismo resultado si se hubieran perdido más paquetes de un mismo *frame*. Por ello, ambos gráficos muestran una tendencia similar, aunque la pérdida de muchos paquetes, sobre todo si estas pérdidas se producen en ráfaga, no afecta tanto a la pérdida de *frames*.

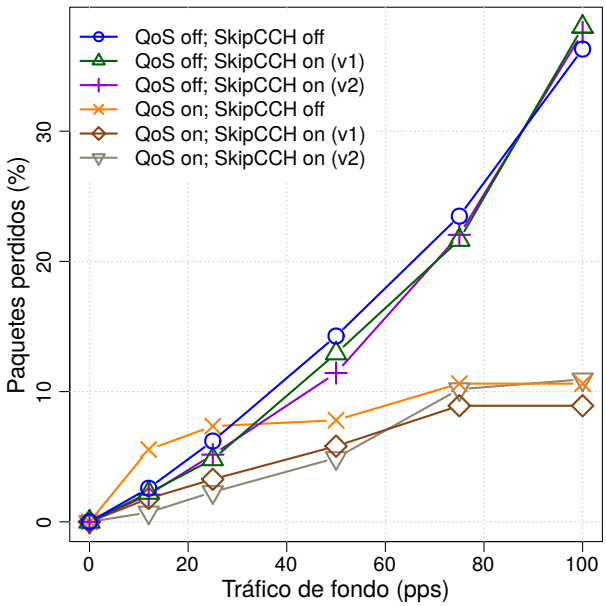


(a) End-to-End Delay

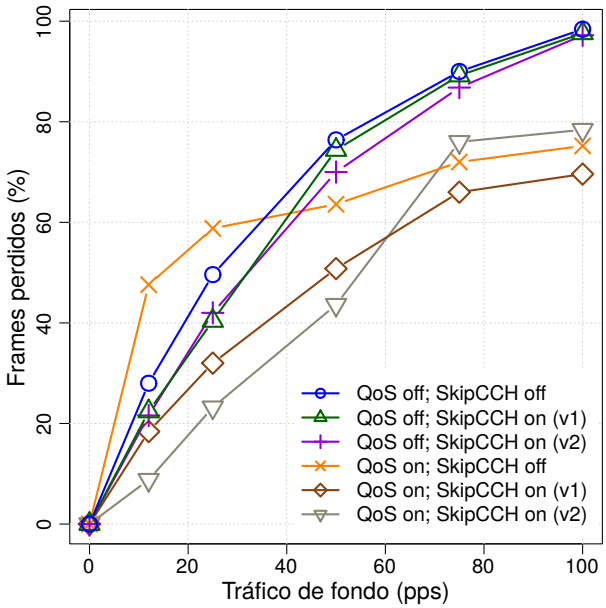


(b) Jitter

Fig. 6: End-to-End Delay y Jitter.



(a) Paquetes perdidos



(b) Frames perdidos

Fig. 7: Paquetes de vídeo perdidos y *frames* perdidos.

Tal como se puede observar en estas gráficas, así como en las Tablas II y III, cuando no se utiliza QoS, en el peor de los casos se produce una pérdida de paquetes de entre el 36 % y 38 % para los tres casos, llevando a una pérdida de *frames* de hasta el 98 %. Por otro lado, cuando se utiliza QoS con su mecanismo predeterminado (*Default*) se aprecia el efecto positivo sólo cuando la red comienza a estar saturada (50 o más pps de tráfico de fondo), es decir, se pierden menos paquetes y *frames* que cuando no se utiliza QoS. Sin embargo, al contrario de lo que se podría esperar, cuando la red está ligeramente saturada (12 o 25 pps), tanto la pérdida de paquetes como de *frames* es mayor cuando se utiliza QoS. Esto era debido al efecto de sincronización que se produce por la alternancia de canales del estándar IEEE 1609.4, y la pérdida de paquetes por la transmisión en ráfagas al comienzo de los *slots* SCH.

Con el mecanismo propuesto, SkipCCH, se puede apreciar que se reduce la pérdida de paquetes y *frames*, y, sobre todo, que las pérdidas siempre son menores que cuando no se utiliza QoS, eliminando el efecto de la sincronización. Por ejemplo, si comparamos la pérdida de *frames* para esos casos (12 y 25 pps), con QoS desactivado se pierden el 28 % y casi 50 % de los frames, respectivamente, y con QoS activado, con el mecanismo predeterminado se pierden más del 47 % y 58 % (mayor pérdida), mientras que con el mecanismo SkipCCH (v2), los porcentajes bajan al 8 % y 23 % (menor pérdida). Si comparamos las dos versiones del mecanismo, la segunda implementación (*v2*) funciona mejor que la primera (*v1*), tal y como se esperaba, puesto que en la *v2* ya se considera los intervalos de guarda (4 ms); sin embargo, cuando la red está muy saturada, la primera versión muestra un mejor resultado.

Tabla II: Paquetes de vídeo perdidos (%).

		Tráfico de fondo (pps)					
QoS	SkipCCH	0	12	25	50	75	100
off	off	0,0	2,6	6,2	14,3	23,5	36,3
	on (v1)	0,0	2,2	4,8	12,9	21,6	38,0
	on (v2)	0,0	2,1	5,2	11,4	22,0	37,6
on	off	0,0	5,6	7,3	7,8	10,6	10,6
	on (v1)	0,0	1,8	3,3	5,8	8,9	8,9
	on (v2)	0,0	0,7	2,3	4,9	10,2	10,9

Tabla III: *Frames* perdidos (%).

		Tráfico de fondo (pps)					
QoS	SkipCCH	0	12	25	50	75	100
off	off	0,0	28,0	49,6	76,4	90,0	98,4
	on (v1)	0,0	22,4	40,4	74,4	89,2	97,6
	on (v2)	0,0	21,6	42,0	70,0	86,8	97,2
on	off	0,0	47,6	58,8	63,6	72,0	75,2
	on (v1)	0,0	18,4	32,0	50,8	66,0	69,6
	on (v2)	0,0	8,8	23,2	43,6	76,0	78,4

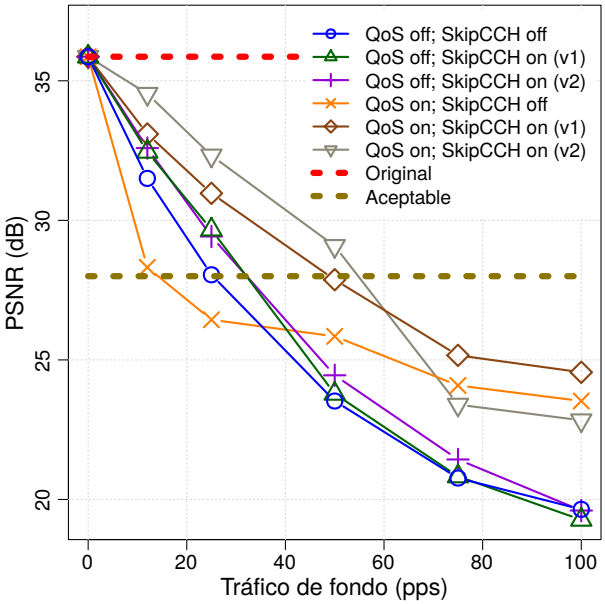
Tabla IV: PSNR del vídeo reconstruido (dB).

		Tráfico de fondo (pps)					
QoS	SkipCCH	0	12	25	50	75	100
off	off	35,9	31,5	28,1	23,5	20,8	19,7
	on (v1)	35,9	32,5	29,7	23,8	20,8	19,3
	on (v2)	35,9	32,6	29,4	24,5	21,4	19,6
on	off	35,9	28,3	26,4	25,9	24,1	23,5
	on (v1)	35,9	33,1	31,0	27,9	25,2	24,6
	on (v2)	35,9	34,5	32,3	29,1	23,4	22,8

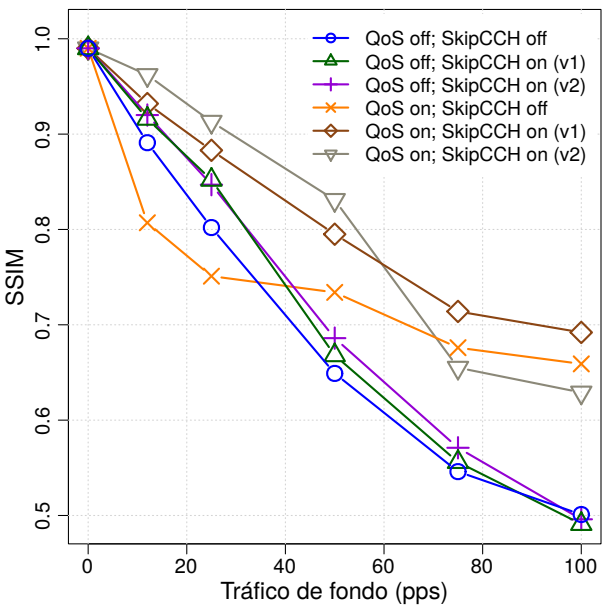
Por último, para comprobar el efecto de los *frames* perdidos sobre la calidad del vídeo percibida, se analizan dos métricas de calidad, el PSNR (Peak Signal-to-Noise Ratio) y la SSIM (Structural Similarity Index Measure). Ambas métricas muestran una tendencia muy similar (figura 8), aproximadamente inversas al gráfico de pérdida de *frames* (figura 7b). Al utilizar el modo AI, la pérdida de cada *frame* influye de igual manera en el PSNR final. La Tabla IV muestra los valores numéricos, resaltando con colores 4 rangos: verde si el valor de PSNR es mayor de 32 dB (valor muy bueno), amarillo si es mayor de 28 dB (valor aceptable), naranja si está por debajo de ese valor y hasta 24 dB (deficiente), y rojo para valores inferiores (muy deficiente). Se puede observar que utilizar QoS con el mecanismo predeterminado no aportaría nada a la calidad percibida, ya que, para baja carga de la red, el valor del PSNR es menor que cuando no se utiliza QoS (problema de la sincronización), y para una mayor carga, aunque el PSNR sí es mayor, no llega al umbral para considerar el valor como aceptable (28 dB). Sin embargo, utilizando el mecanismo propuesto (con cualquiera de las dos versiones), siempre se obtiene un PSNR mayor, y se alcanza el umbral aceptable hasta una carga media de la red.

VI. Conclusiones y Trabajo futuro

En este trabajo, se ha propuesto y evaluado un mecanismo denominado SkipCCH como solución al problema de la alta tasa de pérdida de paquetes que se produce debido a la alternancia entre los canales CCH y SCH del estándar IEEE 1609.4 de la pila de protocolos WAVE, sobre todo al utilizar QoS. Esta alternancia provoca una sincronización entre los nodos de una VANET que hace que se intenten transmitir ráfagas de paquetes al comienzo de los *slots* SCH, produciéndose un gran número de colisiones y la consecuente pérdida de paquetes, especialmente en aplicaciones que requieren la transmisión de vídeo, al requerir un mayor ancho de banda.



(a) Peak Signal-to-Noise Ratio (PSNR)



(b) Structural Similarity Index Measure (SSIM)

Fig. 8: Métricas de calidad perceptual.

El mecanismo SkipCCH planifica la transmisión de los paquetes de vídeo teniendo en cuenta el *slot* activo de la capa MAC, evitando los *slots* CCH y distribuyendo los paquetes consecuentemente a lo largo de los *slots* SCH para mantener la misma tasa de envío de *frames* por segundo. Como resultado, se eliminan las ráfagas, y se reducen las colisiones y las pérdidas de paquetes. A la vista de los resultados, se mejora el rendimiento de la red, y la calidad perceptual del vídeo reconstruido. Además, el mecanismo SkipCCH reduce la penalización que el mecanismo predeterminado provoca en el resto de tráfico de red

menos prioritario al utilizar QoS.

A pesar de los beneficios, tanto de la calidad de servicio como del mecanismo propuesto para la transmisión de vídeo, la calidad de vídeo sigue siendo inaceptable a partir de cargas de red moderadas. Por ello, en trabajos futuros, se propondrán técnicas de protección de vídeo tanto en la codificación HEVC (modos LP, LB y RA), en la aplicación de la QoS (diferentes prioridades para los distintos tipos de *frames*), como en el canal con mecanismos Forward Error Correction (FEC), que combinados adecuadamente permitan aumentar la robustez del vídeo ante las pérdidas de paquetes, mejorando sensiblemente la calidad de vídeo recibido para un rango amplio de cargas de red.

Agradecimientos

Esta publicación es parte del proyecto de I+D+i PID2021-123627OB-C55, financiado por MCIN/AEI/10.13039/501100011033/FEDER, UE.

Referencias

[1] "High Efficiency Video Coding (HEVC)," ITU-T Recommendation H.265, 2013.

[2] David Eckhoff, Christoph Sommer, and Falko Dressler, "On the necessity of accurate ieee 802.11p models for ivc protocol simulation," in *2012 IEEE 75th Vehicular Technology Conference (VTC Spring)*, 2012, pp. 1–5.

[3] Estefanía Coronado, José Villalón, and Antonio Garrido, "Dynamic aifsn tuning for improving the qos over ieee 802.11 wlans," in *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2015, pp. 73–78.

[4] Stephan Eichler, "Performance evaluation of the ieee 802.11p wave communication standard," in *2007 IEEE 66th Vehicular Technology Conference*, 2007, pp. 2199–2203.

[5] Guilherme Maia, Cristiano G. Rezende, Leandro A. Villas, Azzedine Boukerche, Aline Carneiro Viana, André L. L. de Aquino, and Antonio Alfredo Ferreira Loureiro, "Traffic aware video dissemination over vehicular ad hoc networks," in *16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM '13, Barcelona, Spain, November 3-8, 2013*, Björn Landfeldt, Mónica Aguilar-Igartua, Ravi Prakash, and Cheng Li, Eds. 2013, pp. 419–426, ACM.

[6] Erick Donato, Guilherme Maia, Edmundo Madeira, and Leandro Villas, "Impact of 802.11p channel hopping on VANET communication protocols," *Latin America Transactions, IEEE (Revista IEEE America Latina)*, vol. 13, pp. 315–320, 01 2015.

[7] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker, "Recent Development and Applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, 2012.

[8] András Varga and Rudolf Hornig, "An Overview of the OMNeT++ Simulation Environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, 2008, Simutools '08, pp. 60:1–60:10.

[9] Christoph Sommer, Reinhard German, and Falko Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, 2011.

[10] Mordechai (Muki) Haklay and Patrick Weber, "OpenStreetMap: User-Generated Street Maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[11] Joint Collaborative Team on Video Coding (JCT-VC), "HEVC reference software HM (HEVC Test Model) and Common Test Conditions," Retrieved August 20, 2018 from `https://hevc.hhi.fraunhofer.de`.

[12] Pedro Pablo Garrido Abenza, Manuel P. Malumbres, Pablo Piñol, and Otoniel López Granado, "A simulation tool for evaluating video streaming architectures in vehicular network scenarios," *Electronics*, vol. 9, no. 11, 2020.

[13] P. P. Garrido Abenza, M. P. Malumbres, P. Pinol Peral, and O. Lopez-Granado, "Evaluating the use of QoS for video delivery in vehicular networks," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, 2020, pp. 1–9.

[14] David Eckhoff, Nikoletta Sofra, and Reinhard German, "A performance study of cooperative awareness in etsi its g5 and ieee wave," in *2013 10th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, 2013, pp. 196–200.

Monitorización de redes InfiniBand para reaccionar eficazmente a la congestión

Gabriel Gomez-Lopez[1], Alberto Cascajo[2], Jesús Escudero-Sahuquillo[1], Pedro J. García[1], David E. Singh[2], Francisco Alfaro-Cortés[1], Francisco J. Quiles[1]y Jesus Carretero[2].

Resumen— **Las redes de interconexión de altas prestaciones para sistemas de altas prestaciones y de centros de datos incorporan mecanismos para evitar que la congestión reduzca el rendimiento de la red. En concreto, la popular especificación InfiniBand define un mecanismo para reducir la tasa de inyección de los flujos de tráfico que contribuyen a la congestión. Sin embargo, la eficacia de este mecanismo depende de los valores configurados para determinados parámetros, que pueden ser adecuados para algunas situaciones de congestión pero no para otras. Por ello, pensamos que estos parámetros deberían reconfigurarse dinámicamente, basándose en información precisa y actualizada sobre el estado real de la congestión. Para ello, hemos combinado una herramienta ligera de monitorización (LIMITLESS) con el software de control de InfiniBand (OpenSM), de forma que la primera proporciona al segundo información sobre la congestión para reconfigurar adecuadamente los parámetros que definen el comportamiento del mecanismo de control de la congestión. Los experimentos realizados en un clúster real basado en InfiniBand confirman que este enfoque reduce significativamente el número de veces que el mecanismo de control de congestión reacciona erróneamente.**

Palabras clave— **Redes de interconexión, clúster, control de congestión, monitor de tráfico, InfiniBand**

I. Estado de la cuestión y motivación

LAS redes de interconexión de alta velocidad y baja latencia son necesarias para que los modernos sistemas de computación de altas prestaciones (*High-Performance Computing*, HPC) y los centros de datos puedan hacer frente a los requisitos de comunicación de las exigentes aplicaciones y servicios que soportan estos sistemas; de lo contrario, la red se convertiría en el cuello de botella de todo el sistema. La Arquitectura InfiniBand (IBA) es una de las tecnologías de interconexión más populares debido al alto rendimiento que ofrece. IBA está ampliamente implantada en muchos sistemas incluidos en la lista Top500.[1].

Los componentes de red basados en la especificación IBA [2] (Switches, enlaces e interfaces de red (*host channel adapted*, HCA)) permiten construir diferentes topologías, como Fat-Trees, Dragonflies u otras redes de bajo diámetro, muy utilizadas en sistemas HPC y centros de datos. Tanto el descubrimiento de la topología de la red como la configuración de un encaminamiento adecuado son responsabilidad del*subnet manager* (SM), una entidad software, que no sólo controla estas funcionalidades sino varias otras. OpenSM [3] es la implementación más popular del SM, este está incluido en el software OpenFabrics (OFS) [4].

Más en detalle, al descubrir la topología de la red, el SM asigna direcciones de 16 bits, denominadas identificadores locales (LID) a los dispositivos de red (es decir, HCA y conmutadores), y rellena las tablas de encaminamiento en los conmutadores, basándose en el algoritmo de encaminamiento seleccionado. Aunque las redes basadas en IBA admiten encaminamiento no conscience (oblivious) y adaptativo, en este trabajo nos centramos en el encaminamiento determinista. Cada dispositivo de red tiene también un GUID (Global Unique Identifier), un identificador de 64 bits asignado a cada dispositivo dentro de una subred. Los GUID son independientes de los LID. El SM también reconfigura la red en tiempo de ejecución ante eventuales cambios, fallos o problemas.

Sin embargo, aunque el SM configure un encaminamiento eficiente para la topología específica del sistema, éste puede verse perjudicado por situaciones de congestión que surgen cuando las aplicaciones del sistema generen tráfico intenso que colapse la red (o partes de la red) al ser inyectado desde los HCA. Como es probable que estas situaciones acaben degradando el rendimiento de la red (y, por tanto, del sistema), la especificación IBA define un mecanismo de control de la congestión (CC) [5] para reducir la inyección de tráfico cuando se detecta que contribuyen a una situación de congestión (es decir, como *flujos de congestión*). Este mecanismo sigue un enfoque de bucle cerrado que puede resumirse en los siguientes pasos:

1. Se controla la ocupación de los búferes en los puertos del switch: si esta ocupación supera un umbral determinado, el puerto de salida correspondiente entra en estado de congestión.
2. Cualquier paquete que cruce un puerto en estado de congestión puede marcarse como congestionante (en función de una tasa de marcado configurada previamente) activando el bit FECN (*Forward Explicit Congestion Notification*) en la cabecera del paquete.
3. Cuando una HCA recibe un paquete marcado con FECN, devuelve un paquete de notificación con el bit BECN (*Backward Explicit Congestion Notification*) activado, dirigido a la HCA de origen del paquete marcado con FECN.
4. Las HCAs de origen que reciban BECNs reducirán la tasa de inyección de los flujos de tráfico

[1]Dpto. de Informática, Universidad de Castilla-La Mancha, e-mail: {gabriel.gomez,jesus.escudero, pedrojavier.garcia, fco.alfaro, francisco.quiles}@uclm.es

[2]Department de Informática e Ingeniería, Universidad Carlos III de Madrid, e-mail: {acascajo, dexposit, jcarrete}@inf.uc3m.es

dirigidos a las HCAs que envíen esos BECNs.

En concreto, cada HCA de origen aplica un retardo de la tasa de inyección (*injection rate delay*, IRD) específico para separar la inyección de paquetes dirigidos a una HCA de destino determinado. Cada HCA dispone de una tabla de control de congestión (CCT), que contiene entradas para almacenar 128 valores de IRD preconfigurados, ordenados de forma ascendente. Los valores iniciales de esta lista de IRDs pueden ser definidos por el usuario en la configuración de OpenSM. El IRD específico aplicado en una HCA de origen para inyectar paquetes dirigidos a un destino determinado se define mediante el *CCTI_Index*, que apunta a una entrada en el CCT. En cada recepción de BECN en una HCA de origen, el *CCTI_Index* para la HCA de destino que envió el BECN se incrementa para apuntar a una entrada en el CCT con un valor más alto del IRD. De este modo, se limita (es decir, se reduce) el caudal de inyección de ese flujo. El aumento específico aplicado al *CCTI_Index* en cada recepción de BECN se define mediante el parámetro *CCTI_Increase*. Por el contrario, una HCA fuente disminuye el IRD para un destino dado disminuyendo el *CCTI_Index* cada vez que expira un *Timer* y no se han recibido BECNs desde ese destino (lo que indica que la congestión está desapareciendo en el camino hacia ese destino, por lo que la tasa de inyección hacia ese destino debería aumentar en consecuencia). Cuando el *CCTI_Index* es 0 en una HCA de origen para un destino dado, no se aplica ningún IRD a la inyección de paquetes hacia ese destino, ya que el mecanismo considera que la congestión ha desaparecido en el camino hacia ese destino. El valor típico para *CCTI_Increase* es de 1, y este valor nunca cambia durante el tiempo de ejecución, por lo que el *CCTI_Index* aumenta y disminuye de forma individual, y así el IRD para un destino dado varía lentamente.

Desafortunadamente, el rendimiento del mecanismo de CC de IBA depende en gran medida del ajuste de los parámetros descritos anteriormente, lo que supone un verdadero reto [5], [6]. Además, la congestión es por naturaleza muy dinámica [7], ya que puede aparecer en algún punto de la red, trasladarse posteriormente a otro diferente, y seguir así indefinidamente. De ahí que sea muy complejo identificar con precisión y en todo momento los flujos que contribuyen a una situación de congestión. Incluso los flujos que no contribuyen a la congestión pueden ser identificados erróneamente como congestionados y, por tanto, su inyección se reduce innecesariamente (*"flujos víctimas"*). Todo ello provoca inestabilidades en el rendimiento de las redes que utilizan el mecanismo de CC de IBA.

Por estas razones, creemos que configurar y ajustar dinámicamente los parámetros del mecanismo de CC de IBA, basándose en la información proporcionada por ciertos contadores de rendimiento de IBA, mejoraría la eficiencia del mecanismo de CC. En concreto, en este trabajo nos centramos en los contadores de rendimiento de IBA utilizados en las HCAs para medir la gravedad de la congestión, concretamente *PortXmitWait*, que mide el número de ciclos de hardware gastados por una HCA esperando para transmitir paquetes debido a la insuficiencia de créditos en el buffer del siguiente switch, y *PortXmitData*, que mide el número de bytes inyectados en la red desde una determinada HCA.

Para recopilar la información de estos contadores de rendimiento en tiempo de ejecución, se puede utilizar una herramienta de monitorización de grano fino, y luego enviar la información recopilada a OpenSM para actualizar los parámetros del mecanismo de CC de IBA. En concreto, proponemos combinar el funcionamiento de OpenSM con una herramienta de monitorización ligera ya existente llamada LIMITLESS [8], [9], un monitor que puede trabajar bajo tasas de muestreo cercanas al segundo y que proporciona los valores actuales de los contadores de rendimiento del IBA. Básicamente, la idea de nuestra propuesta es que LIMITLESS recoja métricas sobre los volúmenes de comunicación en la red a través de los contadores de rendimiento IBA en los dispositivos de red. A continuación, LIMITLESS proporciona a OpenSM una imagen precisa de dónde se origina la congestión en la red y su grado de gravedad. Por último, OpenSM configura las HCA para ajustar de forma dinámica y precisa sus tasas de inyección dependiendo de si están contribuyendo a la congestión de la red o no.

Hay que tener en cuenta que la monitorización de sistemas para plataformas HPC a gran escala es una tarea compleja, que se convierte en un reto cada vez mayor a medida que la escala y la complejidad de la infraestructura crecen. De hecho, algunos monitores de sistemas como Ganglia [10], Nagios [11], entre otros, presentan diferentes problemas cuando se utilizan en sistemas HPC, y además, no están adaptados específicamente para monitorizar redes InfiniBand. En cambio, en [12] se presentó un trabajo genérico para herramientas de monitorización de sistemas a gran escala, incluyendo aplicaciones para InfiniBand. Más recientemente, INAM [13] y [14] permiten a los usuarios analizar y visualizar el estado de las comunicaciones en una red InfiniBand de 2000 nodos con una sobrecarga muy baja y utilizando OpenSM. LIMITLESS, en comparación con otras herramientas de monitorización HPC, presenta una baja sobrecarga de monitorización, es altamente escalable, recopila información global del sistema e implementa de forma nativa la monitorización InfiniBand. De este modo, toda esta información recopilada puede explotarse utilizando OpenSM.

En este trabajo, hemos implementado nuestra propuesta en OpenSM y la hemos evaluado en un cluster HPC real utilizando una red basada en IBA, concretamente el cluster CELLIA, cuyos detalles se pueden encontrar en la Sección III-A. Hemos modificado el código fuente de OpenSM para que pueda comunicarse con LIMITLESS y recibir información sobre la congestión, utilizando los contadores de rendimiento IBA de los dispositivos de red. Para los experimentos, utilizamos los benchmarks Netgauge [15] y GPCNeT

[16], activando el CC tradicional de IBA o nuestra propuesta. Basándonos en los resultados, podemos concluir que nuestra propuesta supera a la IBA CC cuando se ejecutan Netgauge y GPCNeT.

El resto del documento se organiza como sigue. La sección II describe nuestra propuesta de combinar un sistema de monitorización con el control de congestión. En la sección III se describen los experimentos llevados a cabo en el clúster real basado en IBA mencionado anteriormente y se analizan los resultados obtenidos. Finalmente, se extraen algunas conclusiones en la Sección IV.

II. Combinando el CC y la monitorización de grano fino

A. Arquitectura de la monitorización

En este trabajo, LIMITLESS se utiliza para monitorizar constantemente el clúster, y para proporcionar información sobre el estado actual de la red a OpenSM. El sistema de monitorización consta de tres componentes diferentes que se organizan de forma jerárquica para transmitir la información de los contadores de rendimiento de IBA desde los nodos y HCAs al servidor principal y OpenSM. La arquitectura se basa en una red superpuesta basada en árboles (TBON) y ofrece simplicidad, escalabilidad y tolerancia a fallos. La etapa final de este proceso consiste en almacenar las métricas en una base de datos que permite analizar y visualizar el estado del sistema.

Los principales componentes de LIMITLESS son: *Daemon Monitor* (LDM), *Daemon Aggregator* (LDA) y *Daemon Server* (LDS), que pueden verse en la Figura 1. El más relevante para este trabajo, es el demonio LDM, ya que recoge periódicamente la información sobre los contadores de rendimiento de IBA (es decir, *PortXmitData*, *PortXmitWait*, etc.), El LDM se ejecuta en cada nodo servidor y consulta los contadores de rendimiento de IBA HCA del nodo. Cada LDA (nótese que el número de LDAs depende del despliegue del monitor) recibe y transforma los datos aplicando compresión (para reducir el tráfico de red) y retransmite la información al siguiente componente de monitorización. El LDS recibe, procesa y almacena las métricas recopiladas de los HCA y las comparte con OpenSM. Según los datos recibidos, OpenSM reconfigurará los parámetros de CC.

Mediante LIMITLESS, es posible recoger las métricas de rendimiento y reconfigurar los parámetros de CC de todo el clúster cada 100ms. Para medir la sobrecarga producida por los LDM (nótese que son los componentes de monitorización ejecutados en los nodos de computación), hemos recogido el tiempo que cada LDM ha estado en ejecución y la carga media que se alcanza. Por cada hora de ejecución, cada LDM consume 19,85s con una carga media del 0,5 % de CPU, lo que representa una sobrecarga de monitorización reducida.

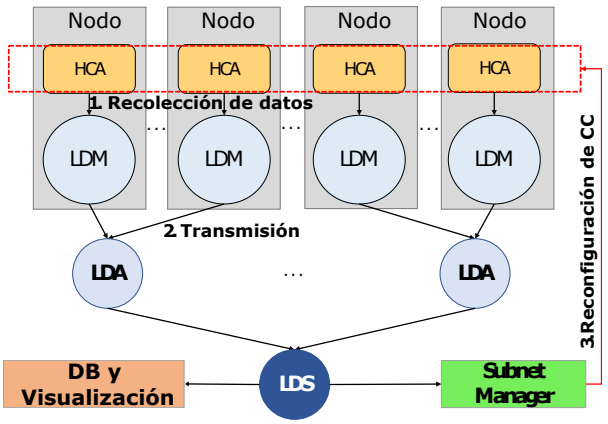


Fig. 1: El monitor LIMITLESS captura la información de los contadores de rendimiento de IBA en los nodos y la envía a nuestra implementación de OpenSM. OpenSM decide qué nodos deben reconfigurar sus parámetros de CC y, a continuación, el Subnet Manager ejecuta esas reconfiguraciones.

B. Reconfiguración dinámica de los parámetros de CC de IBA

Como se ha mencionado anteriormente, el monitor del sistema recopila y almacena los contadores de rendimiento de las HCA utilizando el proceso LDS que se comunica con OpenSM para proporcionar los datos recopilados, que se utilizarán para reaccionar cuando aparezca congestión.

Nuestra propuesta se centra en modificar la configuración de los parámetros de CC de IBA en aquellas HCAs que generan flujos de tráfico que contribuyen a la congestión, en base a la información proporcionada por el monitor, y de forma dinámica. La reconfiguración consiste en aplicar un retardo de tasa de inyección (IRD) más agresivo a los flujos congestionados en esas HCAs, lo que permite reaccionar de forma eficiente a la congestión de la red y evitar sus efectos negativos. Para detectar si una HCA está contribuyendo a la congestión, nos fijamos en el valor *PortXmitWait* proporcionado por LIMITLESS a OpenSM para cada HCA. Este valor cuenta el número de ciclos de reloj de procesador que una HCA específica tiene que esperar para inyectar tráfico en la red. Por lo tanto, los valores grandes indican congestión. Si este valor supera el valor *limite_superior* predefinido (explicado más adelante), entonces la HCA está esperando demasiado, lo que constituye un indicador temprano para identificar que esta HCA podría estar contribuyendo a la congestión.

Cuando una determinada HCA con un alto valor de *PortXmitWait* esté realmente contribuyendo a la congestión, los paquetes generados por este HCA serán marcados, y tarde o temprano se recibirán notificaciones BECN. A medida que OpenSM (en nuestra propuesta) actualice el *CCTI_Increase* para esa HCA, el *CCTI_Index* aumentará, de modo que se aplicará la reducción de la inyección en esa HCA. Sin embargo, pueden observarse valores altos de *PortXmitWait* en HCAs que generan flujos víctimas, en las

que no queremos que se reduzca la tasa de inyección. Obsérvese que estas HCA no tendrán en cuenta las notificaciones BECN recibidas, por lo que no se activará la reducción de inyección, lo que supone una ventaja para nuestra propuesta.

Si el valor de *PortXmitWait* cae por debajo de un *limite_inferior*, esto es una indicación de que el mecanismo de CC está reaccionando y/o la congestión está desapareciendo. En ese momento, ponemos el parámetro *CCTI_Increase* a cero, con lo que la reducción de la inyección se desactivará en esa HCA porque *CCTI_Index* no se incrementará, y disminuirá a cero a medida que el *Timer* expire repetidamente. Este algoritmo de toma de decisiones puede verse en el Algoritmo 1, que muestra específicamente cómo se realiza la reconfiguración.

OpenSM configura el mecanismo de CC utilizando los valores recomendados para los parámetros [5], pero con el parámetro *CCTI_Increase* igual a cero. Esto significa que los HCA descartan las BECN recibidas y no se aplica reducción de la inyección. De hecho, el CC no se activa en los HCA hasta que LIMITLESS informa a OpenSM de que el *PortXmitWait* supera el *limite_superior*. Por lo tanto, OpenSM necesita comprobar el valor de *PortXmitWait* para todos las HCA de la red, al mismo ritmo que estos valores son proporcionados por LIMITLESS. Obsérvese que cualquier nueva configuración del parámetro *CCTI_Increase* de una determinada HCA (ya sea para activar o desactivar dicho parámetro) es enviada por OpenSM a dicha HCA, a través de datagramas de gestión (MADs). Nótese que, debido a la necesidad de acelerar esta parte del algoritmo, los procesos utilizan archivos de memoria mapeados compartidos, lo que permite a los procesos leer y escribir el archivo compartido más rápidamente.

Además, como otra optimización para reducir posibles reconfiguraciones innecesarias, OpenSM sabe si esa HCA ya ha sido actualizada, por lo que es posible evitar enviar continuamente MADs con la configuración de parámetros *CCTI_Increase* a las HCAs.

Algorithm 1 *Actualizar_CC_HCAs* decide cuando actualizar los parámetros de CC en las HCAs.

Parámetros de entrada: limite_superior, limite_inferior, nuevo_valor_ccti

1: **for all** *HCAs en la red* **do**
2: **if** (!$cc\_modificado$ **and** $port\_xmit\_wait > limite\_superior$) **then**
3: $cc\_modificado \leftarrow true$
4: $ccti\_increase \leftarrow nuevo\_valor\_ccti$
5: $mandar\_nueva\_CCconf\_hca()$
6: **end if**
7: **if** ($cc\_modificado$ **and** $port\_xmit\_wait < limite\_inferior$) **then**
8: $cc\_modificado \leftarrow false$
9: $ccti\_increase \leftarrow 0$
10: $mandar\_inicial\_CCconf\_hca()$
11: **end if**
12: **end for**

C. Implementación en OpenSM

Esta sección describe las modificaciones realizadas en OpenSM v3.3.19 para aplicar nuestra propuesta.

Hemos configurado el encaminamiento de la red según el conocido algoritmo de encaminamiento determinista D-mod-K, que distribuye uniformemente las rutas entre todos los enlaces que conectan dos etapas cualesquiera del fat-tree, con lo que se consigue un rendimiento similar (o incluso mejor) al encaminamiento adaptativo en varios escenarios de tráfico, a la vez que se reduce la complejidad de implementación. Hemos evaluado nuestra propuesta en un clúster real utilizando una red basada en IBA, donde hemos construido una topología Slimmed Fat-Tree(SFT). Hemos incluido el código fuente de nuestra propuesta en el encaminamiento `ftree` disponible en OpenSM. En concreto, el encaminamiento `ftree` intercambia MADs con los dispositivos IBA (es decir, las HCA) mediante varias funciones proporcionadas por el gestor de control de congestión de OpenSM (*CCMgr*). Las dos funciones principales son `osm_ftree_send_ca_cong_setting` y `osm_ftree_mad_post`. El primero encapsula los nuevos valores de los parámetros CC en un MAD. El segundo se utiliza para enviar los MAD a la HCA correspondiente.

El monitor proporciona a OpenSM la información de los contadores de rendimiento en un archivo compartido mapeado en memoria, actualizado periódicamente (aproximadamente cada intervalo de muestreo). Este fichero tiene una línea por nodo de la red. Cada línea contiene el GUID, y el valor de los contadores de rendimiento IBA del HCA correspondiente. OpenSM necesita leer y procesar este archivo más rápido de lo que LIMITLESS recoge la información correspondiente. Para leer esta información, hemos incluido en el encaminamiento `ftree` una función llamada `ftree_read_monitor`, que se ejecuta periódicamente. Tenga en cuenta que esta función no se ejecuta cada vez que OpenSM realiza un *"sweep"*, sino que se ejecuta según el intervalo de muestreo del monitor (definido por el usuario como parámetro en OpenSM). Esta función obtiene el valor *PortXmitWait* para todos las HCA de la red, y suministra esta información a otra función, llamada `ftree_cc_update_hca` que implementa la funcionalidad descrita en el Algoritmo 1. Esta función decide si es necesario actualizar los parámetros de CC en una HCA determinada.

III. Evaluación de Resultados

En esta sección, describimos la prueba de concepto utilizada para analizar el comportamiento de nuestra propuesta en un banco de pruebas realista. Nuestro principal objetivo es verificar que la información proporcionada por el monitor LIMITLESS puede ser procesada y que podemos configurar la red para reaccionar mejor a la congestión. El banco de pruebas se ha construido en un clúster HPC utilizando una red de interconexión basada en InfiniBand. En este sistema, hemos generado escenarios de tráfico realis-

tas en los que aparece congestión (utilizando benchmarks usados en la evaluación de la red en los sistemas HPC), y hemos analizado el comportamiento de nuestra propuesta (implementada en OpenSM) en comparación con la configuración de CC de referencia. A continuación, detallamos el banco de pruebas, los experimentos realizados y los resultados obtenidos.

A. *Configuración Hardware para los test*

Para implementar nuestra prueba de concepto hemos utilizado el clúster CELLIA (*Cluster for the Evaluation of Low-Latency Interconnection Architectures*), el cual permite construir topologías de red utilizando hasta 50 nodos HP Proliant DL120 Gen9, con un procesador Intel Xeon E5-2630v3 de 8 núcleos a 1,80 GHz y 16 GB de RAM. La red de interconexión se construye con hardware basado en IBA[1]. CELLIA dispone de conmutadores Mellanox IS5022 de 8 puertos con tecnología QDR, cada puerto ofrece 40 Gbps, que pueden utilizarse para construir diferentes topologías de red. Concretamente,utilizamos cables de cobre QSFP con enlaces 4xQDR. Cada nodo tiene una HCA Mellanox ConnectX3 MCX353A-QCBT de doble puerto de 40 Gbps.

Hemos construido en CELLIA una topología *Slimmed Fat-Tree* (SFT)[17] de 36 nodos. La SFT es una topología muy conocida, debido a su popularidad y amplio uso en sistemas HPC. La Figura 2 muestra la topología SFT construida en CELLIA.

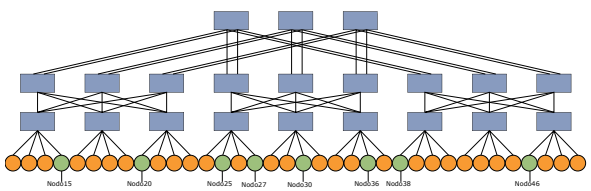


Fig. 2: Topología Slimmed Fat-Tree(SFT) de 36 nodos y 3 etapas construida en CELLIA. Los nodos verdes se utilizan en la prueba GPCNeT para medir el rendimiento, mientras que los nodos naranjas se utilizan para generar congestión.

Con el fin de proporcionar una comparación justa entre el mecanismo original de CC de IBA y nuestra propuesta, hemos diseñado un conjunto de pruebas utilizando tres configuraciones CC: (1) SinCC (desactivado), (2) IBA CC estándar (es decir, *estático*), y (3) LIMITLESS+CC (es decir, nuestra propuesta), que sintoniza los parámetros CC dinámicamente basándose en la información de monitorización. La Tabla I muestra la configuración de los parámetros de CC de IBA y de nuestra propuesta.

Tabla I: Configuración parámetros de CC y de nuestra propuesta.

	Parámetro	Valor
Parámetros Comunes	*Threshold*	0x7
	Packet_size	1
	Marking_Rate	0x000f
	CCTI_Timer	150
	CCTI_Increase	{1,2,5,10,20}
Parámetros de nuestra propuesta	*Limite_superior (ver Algoritmo 1)*	0.850 ms
	Limite_inferior (ver Algoritmo 1)	0.055 ms

Obsérvese que hemos utilizado distintos valores de *CCTI_Increase* para evaluar cómo afecta este parámetro a la congestión de los nodos; un valor alto de *CCTI_Increase* reduce la inyección de forma más agresiva. El resto de parámetros se han configurado según lo recomendado en otros estudios [5].

B. *Generación de tráfico en la red*

Para medir el rendimiento de la red en escenarios congestionados, hemos seleccionado e instalado en CELLIA dos conocidos benchmarks: Netgauge [15] y GPCNeT [16]. Estos benchmarks generan en la red patrones de tráfico comunes que nos permitirán validar la prueba de concepto de nuestra propuesta. *Netgauge* es una herramienta de alta precisión que mide el rendimiento de las red de los sistemas de HPC. Soporta muchos protocolos de red y patrones de comunicación diferentes, como el ancho de banda de bisección eficaz (*efficient bisection bandwidth, (ebb)*) o muchos-a-uno (*Nto1*). En concreto, el patrón de comunicación *ebb* mide el ancho de banda de bisección eficaz, tal y como se define en [18], generando varios patrones de tráfico aleatorios y calculando el ancho de banda de bisección en dichos patrones. Para el patrón de tráfico *Nto1*, $N-1$ procesos (donde $N = 36$ en nuestro cluster) transmiten datos a un único destino elegido (hot-spot), y éste les envía datos de vuelta. Hemos ejecutado Netgauge con 36 tareas MPI (es decir, una tarea MPI por nodo final en el clúster). *GPCNeT* se utiliza generalmente para medir el impacto de la congestión en sistemas HPC. Se ha probado en sistemas como Cray y Summit. GPCNeT elige aleatoriamente el 20 % de los nodos (es decir, los nodos de pruebas) para calcular las métricas de rendimiento y el resto de los nodos para generar tráfico congestionado (es decir, los nodos congestionantes). Hemos retocado GPCNeT para que los nodos de prueba elegidos aleatoriamente sean los mismos en el SFT entre las distintas configuraciones para el parámetro *CCTI_Increase*, y para medir con precisión el valor de los valores *PortXmitWait* y *PortXmitData* en función del tiempo (como describimos más adelante).

La Figura 2 muestra los nodos pruebas (en verde) y los nodos congestionantes (en naranja). Hemos ejecutado GPCNeT utilizando 8 tareas MPI por nodo para ambas topologías. GPCNeT ofrece métricas de rendimiento ejecutando las siguientes pruebas:

[1]Nótese que la red de CELLIA se basa en la tecnología InfiniBand QDR, es decir, no es la más reciente. No obstante, esta red permite comprobar la viabilidad y eficacia de la implementación de nuestra prueba de concepto. Además, nuestra propuesta es compatible con hardware InfiniBand más moderno, que, como se supone en nuestro enfoque, sigue utilizando un mecanismo de control de la congestión basado en la reducción de la inyección (véase: `https://support.mellanox.com/s/article/Nvidia-Congestion-Control`) y contadores de rendimiento para recopilar información sobre el estado de la red (véase `https://support.mellanox.com/s/article/understanding-mlx5-linux-counters-and-status-parameters`. Obsérvese, no obstante, que los resultados específicos de nuestro planteamiento para las distintas tecnologías InfiniBand podrían variar cuantitativamente

- *Random Ring Point-to-point Latency (P2P Lat).* Las tareas MPI de los nodos de pruebas se ordenan en una estructura de anillo aleatorio, cada tarea envía mensajes de 8 bytes a un vecino del anillo y recibe mensajes de 8 bytes del otro vecino del anillo. Nótese que, en esta prueba, la latencia de los paquetes se mide en microsegundos (μs).
- *Random Ring Point-to-point Bandwidth with Synchronization (P2PBW+Sync).* En la misma estructura mencionada anteriormente, se envían y reciben ocho mensajes de 128 KB a los vecinos izquierdo y derecho (16 mensajes en total), que suelen ser suficientes mensajes para alcanzar el ancho de banda máximo [16]. Esta prueba es sensible a la contención del ancho de banda y a la latencia de la sincronización.
- *AllReduce.* Se ejecutan pequeñas operaciones MPI_AllReduce (8 bytes). Esta prueba se utiliza para medir el impacto de la congestión en la comunicación colectiva sensible a la latencia.

En el caso de GPCNeT, para cada prueba se ejecutan los algoritmos descritos anteriormente, primero sin congestión (los nodos congestionantes no generan tráfico). Seguidamente, esos nodos congestionantes estresan la red y los nodos de pruebas vuelven a ejecutar de nuevo los mismas pruebas. Comparando los resultados obtenidos en ambos escenarios se puede obtener cuál es el impacto de la congestión en la red. Hemos analizado los contadores de rendimiento IBA proporcionados por LIMITLESS para cada prueba de esta evaluación.

C. Análisis de resultados de Netgauge

Las Figuras 3, 4, 5 y 6 muestran los resultados de los patrones de tráfico *Nto1* and *ebb* del benchmark Netgauge usando tres configuraciones: sin control de congestión (SinCC), estandard CC de InfiniBand (CC) y nuestra propuesta (denominada LIMITLESS+CC en las gráficas). Téngase en cuenta que para CC y LIMITLESS+CC en las Figuras 3 y 5 se muestran varias series de resultados, correspondientes a diferentes valores de *CCTI_Increase* (ver Tabla I). La Figura 3 muestra el ancho de banda obtenido (medido en Mbit/s) cuando el patrón *Nto1* de Netgauge se ejecuta con diferentes tamaños de mensaje MPI. Aunque no hay variaciones significativas entre las distintas configuraciones, LIMITLESS+CC consigue un rendimiento similar independientemente del valor de *CCTI_Increase* para mensajes menores de 4K (donde *CCTI_Increase* 1 consigue una ligera mejora), y el mejor rendimiento en tamaños grandes de mensajes para la mayoría de valores de *CCTI_Increase*', en comparación con las configuraciones SinCC y CC. Obsérvese también que el patrón de tráfico *Nto1* no genera flujos víctimas (es decir, todos los flujos de tráfico están congestionados).

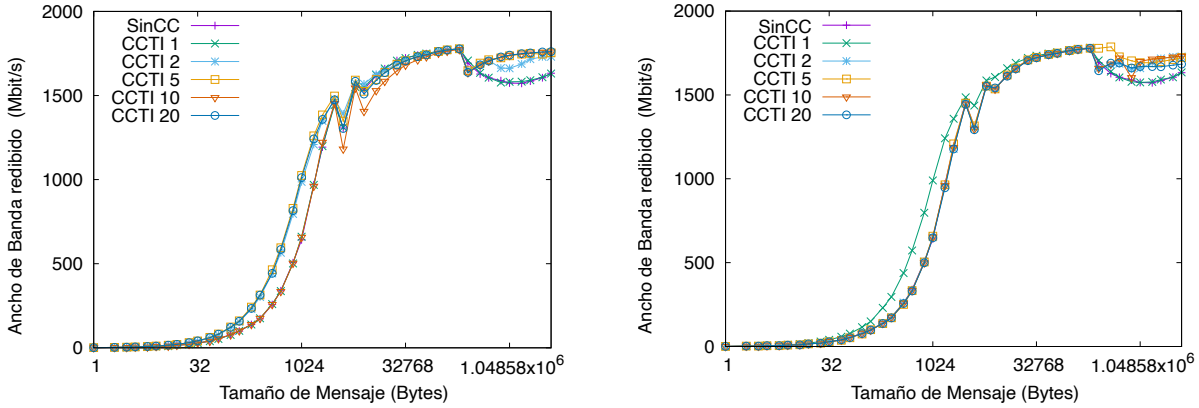
La Figura 4 muestra los valores de los contadores de rendimiento en el nodo 30 frente al tiempo durante el tiempo de ejecución de la prueba *Nto1* (alrededor de 22 segundos). Hemos establecido el parámetro *CCTI_Increase* en 5, que es un valor intermedio que hace que la reducción de la inyección sea más sensible al estado del tráfico de red. En concreto, los valores de *PortXmitData* (es decir, la tasa de inyección) son similares para las tres configuraciones de CC, ya que el patrón de tráfico *Nto1* no genera flujos de víctimas. Los resultados de *PortXmitWait*, medidos en ms, muestran que, cuando utilizamos LIMITLESS+CC, los paquetes esperan a ser inyectados un periodo de tiempo significativamente más corto, lo que significa que la reducción de la inyección (es decir, el valor de IRD) se ajusta con mayor precisión cuando aparece congestión.

Las Figuras 5 y 6 muestran los resultados obtenidos para el patrón de comunicación *ebb* de Netgauge. Obsérvese que *ebb* genera flujos de tráfico cuyo destino se elige aleatoriamente siguiendo una distribución uniforme. Por tanto, la prueba *ebb* no genera situaciones de congestión de larga duración. Como podemos ver en la Figura 5, se produce una degradación significativa del rendimiento cuando CC se configura con valores *CCTI_Increase* más altos. En cambio, LIMITLESS+CC no muestra variaciones de rendimiento independientemente del valor del parámetro *CCTI_Increase*. Por tanto, LIMITLESS+CC amplía la horquilla de valores adecuados para el parámetro CCTI_Increase.

La Figura 6 muestra un caso típico de oscilaciones de la tasa de inyección en los nodos. Obsérvese que el número de milisegundos de espera para enviar datos (es decir, los valores *PortXmitWait*) apenas aumenta por encima de 1ms, lo que significa que la congestión no está empeorando realmente el rendimiento de la red en este escenario. Por lo tanto, no es deseable habilitar el mecanismo CC en una situación de congestión ligera sin ningún impacto en el rendimiento de la red. Sin embargo, hay que tener en cuenta que, dado que no es una elección del administrador de red, habilitar nuestra propuesta gestiona mejor la congestión ligera en comparación con el mecanismo CC tradicional. Si se utiliza el CC estándar, se activa inmediatamente la reducción de la inyección de los flujos de tráfico que generan la congestión ligera, aunque esta congestión no sea peligrosa. Aunque existen oscilaciones en los resultados de LIMITLESS+CC, su rendimiento en este escenario es similar al de SinCC y CC, ya que la congestión ligera no empeora el rendimiento de la red.

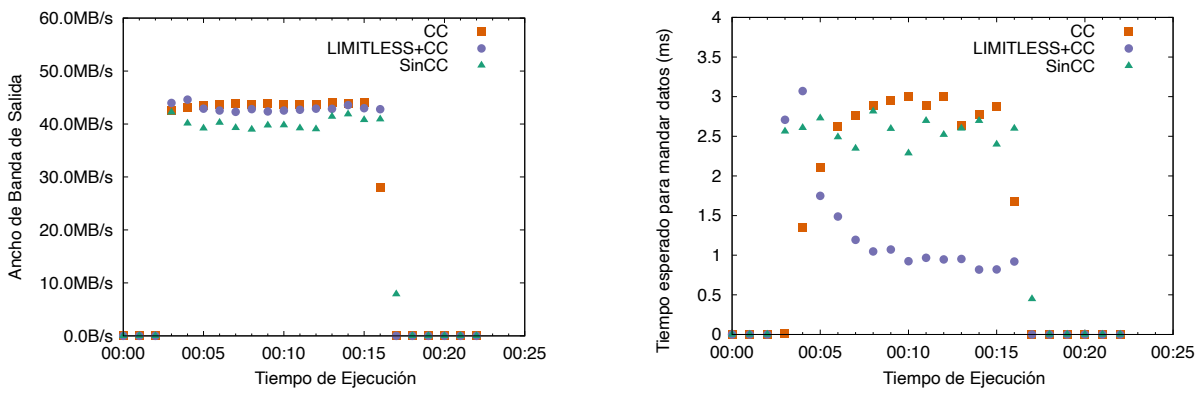
D. Análisis de resultados de GPCNeT

La Tabla II muestra los resultados de la prueba GPCNeT P2PBW+Sync. Nos hemos centrado en esta prueba porque nos interesan los resultados del ancho de banda. Hay que tener en cuenta que el clúster CELLIA es pequeño comparado con los superordenadores más grandes del mundo (donde GPCNeT se ha ejecutado y sus resultados se han publicado en el pasado). Vale la pena mencionar que la implementación LIMITLESS+CC propuesta en este trabajo es una prueba de concepto para demostrar que nuestra propuesta es capaz de aprovechar la información



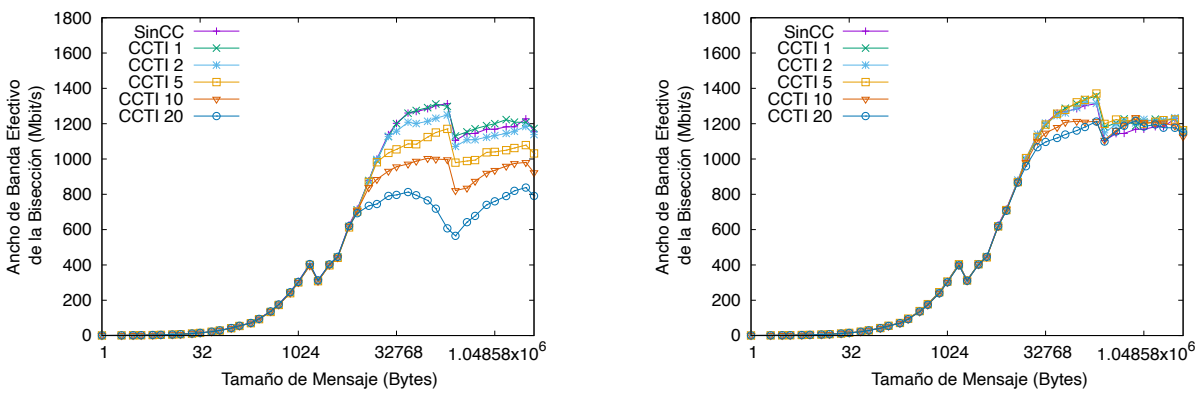
(a) *Nto1* CC. (b) *Nto1* LIMITLESS+CC.

Fig. 3: Ancho de banda para patrón de tráfico *Nto1* de Netgauge (MBit/s) respecto el Tamaño de mensaje (Bytes) en escala logarítmica. *CCTI_Increase* se establece en 1, 2, 5, 10 y 20. Los resultados de SinCC son el mismo en ambas gráficas.



(a) *PortXmitData*. (b) *PortXmitWait*.

Fig. 4: Resultados para los contadores de rendimiento de IBA *PortXmitData* y *PortXmitWait* en el nodo 30 con el patrón de tráfico *Nto1* respecto el tiempo (*CCTI_Increase* = 5).

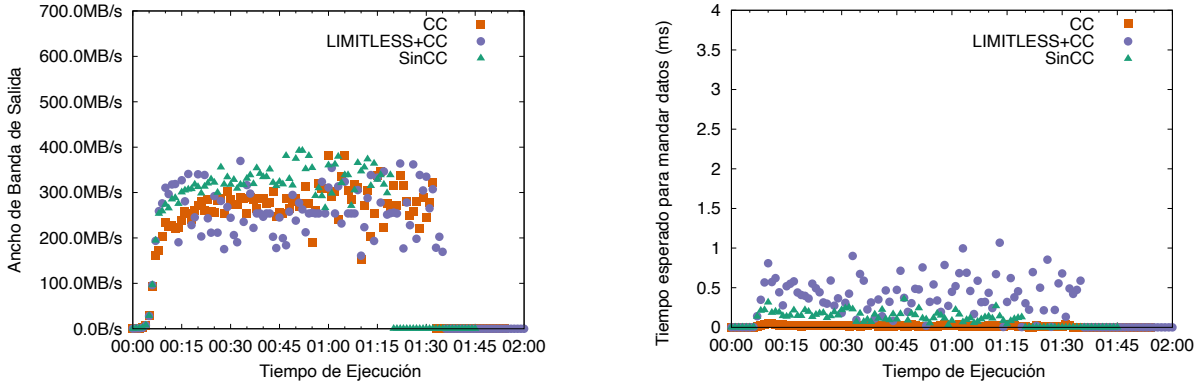


(a) *ebb* CC. (b) *ebb* LIMITLESS+CC.

Fig. 5: Ancho de banda para patrón de tráfico *ebb* de Netgauge (MBit/s) respecto al tamaño de mensaje (Bytes) en escala logarítmica. *CCTI_Increase* se establece en 1, 2, 5, 10 y 20. Los resultados de SinCC son el mismo en ambas gráficas.

de monitorización, y evitar la reacción exagerada del mecanismo tradicional de CC de IBA cuando la congestión es ligera o la inexactitud cuando la congestión es fuerte.

La Tabla II muestra en verde las celdas de la tabla en las que LIMITLESS+CC supera a CC. Nuestra propuesta supera varias veces a CC, independientemente de si hay poca congestión (filas de la tabla "Sin cong.") o mucha congestión (filas de la tabla "Con cong."). Más concretamente, LIMITLESS+CC supera a IBA CC 65 % si comparamos las celdas verdes de la Tabla II. Cuando hay congestión en la red, nuestra propuesta supera a las configuraciones CC estándar y SinCC para valores de *CCTI_Increase* de 1 y 5, tanto en la media como en el del percentil 99. Obsérvese que el mecanismo CC también supera al



(a) *PortXmitData*. (b) *PortXmitWait*.

Fig. 6: Resultados para los contadores de rendimiento de IBA *PortXmitData* y *PortXmitWait* en el nodo 30 con el patrón de tráfico *ebb* respecto el tiempo (*CCTI_Increase* = 5).

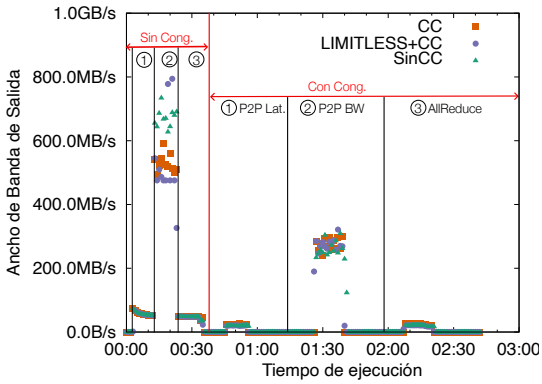
Tabla II: Resultados de GPCNet P2PBW+Sync (MiB/s/task) medios (Avg) y del percentil 99 (99th), utilizando CC de IBA y LIMITLESS+CC (LIM+CC) y el porcentaje de mejora comparado con SinCC (entre paréntesis).

			Valores de CCTI Increase					Sin CC
			1	2	5	10	20	
GPCNeT Con Cong. (P2P BW)	Avg.	CC	76,2 (-0,26 %)	82,6 (8,12 %)	77,5 (1,44 %)	85,7 (12,17 %)	82,3 (7,72 %)	76,4
		LIM+CC	80,5 (5,37 %)	75,6 (-1,05 %)	79,2 (3,66 %)	76,2 (-0,26 %)	78,9 (3,27 %)	
	99th	CC	63,2 (0 %)	63,9 (1,11 %)	62,2 (-1,58 %)	68,1 (7,75 %)	62,9 (-0,47 %)	63,2
		LIM+CC	65,8 (4,11 %)	62,7 (-0,79 %)	64,6 (2,22 %)	59,6 (-5,7 %)	63,2 (0 %)	
GPCNeT Sin Cong. (P2P BW)	Avg.	CC	200,2 (-0,15 %)	146,6 (-26,88 %)	183,4 (-8,53 %)	137,3 (-31,52 %)	159,3 (-20,55 %)	200,5
		LIM+CC	170,4 (-15,01 %)	177,2 (-11,62 %)	181,3 (-9,58 %)	173 (-13,72 %)	170 (-15,21 %)	
	99th	CC	167,4 (1,33 %)	122,2 (-26,03 %)	153,9 (-6,84 %)	118,9 (-28,03 %)	130,5 (-21 %)	165,2
		LIM+CC	131,1 (-20,64 %)	142 (-14,04 %)	123,9 (-25 %)	137,3 (-16,89 %)	133,1 (-19,43 %)	

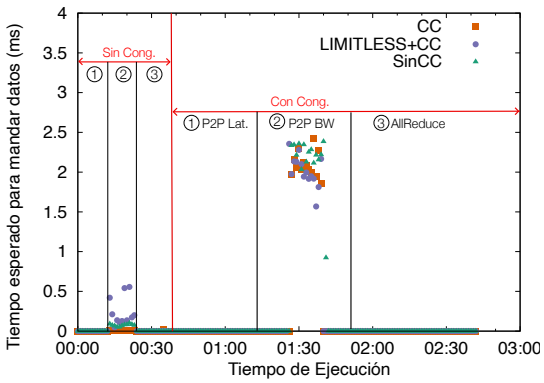
caso de referencia SinCC en la media, mientras que para el caso del percentil 99 el aumento de rendimiento disminuye. Esto significa que CC no se ocupa de los flujos congestionados de larga duración. Los resultados de LIMITLESS+CC siguen siendo coherentes (es decir, con pequeñas variaciones) independientemente de los valores de *CCTI_Increase*. Como hemos mencionado anteriormente, LIMITLESS+CC reduce la reacción exagerada del mecanismo CC estándar de IBA cuando la congestión es ligera.

La Figura 7 muestra los resultados de los valores de los contadores de rendimiento *PortXmitData* (en GB/s) y *PortXmitWait* (en milisegundos esperados de media para enviar datos), obtenidos al ejecutar el benchmark GPCNeT completo. Durante la ejecución de GPCNeT, primero se ejecuta la prueba P2P Lat, después la prueba P2PBW+Sync y, por último, la prueba AllReduce. Estas pruebas duran unos 40 segundos, y sólo generan tráfico en los nodos de pruebas (es decir, los coloreados en verde en la Figura 2). Tras este periodo de tiempo, los nodos congestionantes (es decir, los coloreados en naranja en la Figura 2) generan una fuerte congestión. Los resultados de *PortXmitData* muestran que, cuando no hay congestión (es decir, antes de 40 segundos), la configuración LIMITLESS+CC alcanza un rendimiento máximo de 800 MB/s superando tanto a la configuración CC como a la SinCC. Obsérvese que los valores de *PortXmitWait* oscilan entre 0 y 0,5 ms en este escenario.

Además, cuando la congestión comienza después de 40 segundos, la cantidad de tráfico generada en ese nodo de pruebas (nodo 30) se reduce a 300 MB/s, como muestran los valores de *PortXmitData*. Obsérvese que los nodos congestionantes (80 % según la definición de GPCNet) generan una gran cantidad de tráfico, por lo que la congestión repercute en el rendimiento general de la red y hace que las pruebas de GPCNeT duren más tiempo. Por otro lado, los valores de *PortXmitWait* indican que el tiempo de espera para enviar datos para las configuraciones LIMITLESS+CC, CC y SinCC aumenta significativamente (hasta 2,5ms) cuando aparece congestión, pero también se inyecta más tráfico en la red en comparación con el mecanismo CC estándar, ya que este último mecanismo siempre reacciona demasiado tarde ante la congestión, permitiendo la inyección de tráfico de fuentes congestionadas cuando debería reducir su tasa de inyección a la red.



(a) *PortXmitData*.



(b) *PortXmitWait*.

Fig. 7: Resultados de GPCNeT frente al tiempo para los contadores de rendimiento *PortXmitData* y *PortXmitData* en el nodo 30. La congestión comienza a los 40 segundos ($CCTI\_Increase = 5$).

IV. Conclusiones

En este trabajo hemos ampliado el software de control OpenSM de las redes IBA para que utilice información de monitorización de grano fino y configurado el control de congestión (CC) de IBA para que funcione con mayor precisión. Nuestra propuesta permite a OpenSM configurar dinámicamente los parámetros de CC de IBA en tiempo de ejecución, de forma que los nodos servidores no reaccionen de forma exagerada reduciendo la tasa de inyección de los flujos de tráfico innecesariamente. Hemos evaluado nuestra propuesta en un clúster real ejecutando patrones de tráfico ampliamente utilizados para estresar la red, como *Nto1* y *ebb* de NetGauge y GPCNet, utilizado para medir el impacto de la congestión en las interconexiones de alta velocidad. Los resultados muestran que nuestra propuesta reduce la sobrerreacción del mecanismo CC independientemente de la gravedad de la congestión, al tiempo que facilita la configuración de los parámetros correspondientes. Como trabajo futuro, tenemos previsto ampliar nuestra propuesta con el procesamiento distribuido de la información recogida directamente en las HCAs, dejando la intervención de OpenSM para aspectos más generales y acelerando la velocidad de reacción de nuestra propuesta.

Agradecimientos

El presente trabajo ha sido parcialmente financiado por el Ministerio de Ciencia Español MCIN/AEI/10.13039/501100011033, la Universidad de Castilla-La Mancha mediante el proyecto 2023-GRIN-34056, la Junta de Comunidades de Castilla-La Mancha mediante el proyecto SBPLY/17/180501/000498, la Unión Europea (European High-Performance Computing Joint Undertaking) mediante el projecto "Adaptive multi-tier intelligent data manager for Exascale" con identificación No 956748 - ADMIRE - H2020-JTI - EuroHPC-2019-1, y la agencia Española de Investigación con identificación PCI2021-121966.

Referencias

[1] Top500.org, "Top 500 List," 2023.

[2] InfiniBand Trade Association., "InfiniBandTM Architecture Specification Volume 1 - Release 1.3," Mar. 2015.

[3] OpenSM, "Opensm - infiniband subnet manager," 2019.

[4] The OpenFabrics Software, ," Web Page at: `https://www.openfabrics.org/\index.php/openfabrics-software.html`, 2016.

[5] E.G. Gran, M. Eimot, Sven-Arne Reinemo, T. Skeie, O. Lysne, L.P. Huse, and G. Shainer, "First experiences with congestion control in InfiniBand hardware," in *In Proc. of IPDPS Symposium*, 2010, pp. 1–12.

[6] Jesús Escudero-Sahuquillo, Ernst Gunnar Gran, Pedro Javier García, Jose Flich, Tor Skeie, Olav Lysne, Francisco J. Quiles, and José Duato, "Efficient and cost-effective hybrid congestion control for HPC interconnection networks," *IEEE Trans. Parallel Distributed Syst.*, vol. 26, no. 1, pp. 107–119, 2015.

[7] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Dynamic Evolution of Congestion Trees: Analysis and Impact on Switch Architecture," in *High Performance Embedded Architectures and Compilers, First International Conference, HiPEAC 2005, Barcelona, Spain*, Nov. 2005, vol. 3793 of *Springer Lecture Notes in Computer Science*, pp. 266–285.

[8] Alberto Cascajo, David E Singh, and Jesus Carretero, "Performance-aware scheduling of parallel applications on non-dedicated clusters," *Electronics*, vol. 8, no. 9, pp. 982, 2019.

[9] Cascajo, Alberto and Singh, David E. and Carretero, Jesus, "LIMITLESS- LIght-weight MonItoring Tool for LargE Scale Systems," *Microprocessors and Microsystems*, vol. 93, pp. 104586, sep 2022.

[10] Matthew L Massie, Brent N Chun, and David E Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

[11] "Nagios - The Industry Standard In IT Infrastructure Monitoring," 2018.

[12] Ross Miller, Jason Hill, David A Dillow, Raghul Gunasekaran, Galen M Shipman, and Don Maxwell, "Monitoring tools for large scale systems," in *Proceedings of Cray User Group Conference (CUG 2010)*, 2010.

[13] P. Kousha, S. D. Kamal Raj, H. Subramoni, D. K. Panda, H. Na, T. Dockendorf, and K. Tomko, "Accelerated Real-time Network Monitoring and Profiling at Scale using OSU INAM," *ACM International Conference Proceeding Series*, pp. 215–223, jul 2020.

[14] Nishanth Dandapanthula, *InfiniBand Network Analysis and Monitoring using OpenSM*, Ph.D. thesis, The Ohio State University, 2011.

[15] Torsten Hoefler, Torsten Mehlan, Andrew Lumsdaine, and Wolfgang Rehm, "Netgauge: A network performance measurement framework," in *High Performance Computing and Communications*, Ronald Perrott, Barbara M. Chapman, Jaspal Subhlok, Rodrigo Fernandes de Mello, and Laurence T. Yang, Eds., Berlin, Heidelberg, 2007, pp. 659–671, Springer Berlin Heidelberg.

[16] Sudheer Chunduri et al, "GPCNeT: designing a benchmark suite for inducing and measuring contention in hpc networks," in *Proceedings of the SC '19*, New York, NY, USA, 2019, SC '19, ACM.

[17] Xin Yuan, Santosh Mahapatra, Michael Lang, and Scott

Pakin, "Static load-balanced routing for slimmed fat-trees," *Journal of Parallel and Distributed Computing*, vol. 74, no. 5, pp. 2423–2432, 2014.

[18] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine, "Multistage switches are not crossbars: Effects of static routing in high-performance networks," in *IEEE Int. Conf. on Cluster Computing*, 2008, pp. 116–125.

Mejora de las prestaciones en las comunicaciones usando MPI mediante Redes Definidas por Software

Pablo Gomariz Martínez[1], Francisco M. Delicado Martínez[2] y Enrique Arias Antúnez[2]

Resumen— **Las aplicaciones que hacen uso de los centros de datos generan enormes cantidades de datos que necesitan ser distribuidos entre los computadores a través de la red de interconexión. El tiempo empleado en la transmisión de datos, junto con el tiempo de cómputo, influye en el tiempo total de la ejecución de este tipo de aplicaciones. Debido a esto, minimizar dicho tiempo de transmisión disminuiría el tiempo total de ejecución.**

Las redes definidas por software (*Software Defined Networks, SDN*) ofrecen un entorno sobre el que poder desarrollar soluciones a la hora de abordar la optimización del tiempo de transmisión de datos en aplicaciones de cómputo distribuido.

En este trabajo se expone una forma de integrar el software OpenMPI con una plataforma SDN con el objetivo de disminuir el tiempo de transmisión de datos entre procesos, presentando resultados experimentales que corroboran la viabilidad de la propuesta.

Palabras clave— **SDN, HPC, MPI, balanceo de carga.**

I. Introducción

La computación de altas prestaciones es vital a la hora de realizar tareas que requieren una gran capacidad de cómputo, como la secuenciación del genoma humano del proyecto *Human Genome Project* [1], el modelado del clima para la predicción de fenómenos meteorológicos [2], de tsunamis y terremotos [3], y el mismo análisis de todos los datos generados en estas tareas basadas en la computación paralela [4]. Gran cantidad de aplicaciones paralelas ejecutadas en centros de datos utilizan el paso de mensajes como método de comunicación entre procesos. El modelo de programación paralela mediante paso de mensajes utiliza tanto comunicaciones punto a punto como colectivas, y estas comunicaciones pueden hacer un uso intensivo de la red. Por ello, el tiempo empleado en la transmisión de datos entre procesos es un factor importante a la hora de calcular el tiempo total necesario para la obtención del resultado final. De esta forma, mecanismos que minimicen este tiempo de transmisión de datos entre procesos, pueden ser muy útiles para reducir el tiempo total de ejecución.

En este sentido las Redes Definidas por Software (SDN, *Software Defined Networks*) pueden ser una herramienta muy útil para establecer dichos mecanismos de optimización.

[1]Instituto de Investigación en Informática de Albacete, Universidad de Castilla - La Mancha, e-mail: pablo.gomariz@uclm.es.

[2]Departamento de Sistemas Informáticos, Universidad de Castilla - La Mancha, e-mail: {francisco.delicado,enrique.arias}@uclm.es.

Las SDN presentan un paradigma diferente en las redes de computadores, separando los planos de control y de reenvío de datos, permitiendo programar la lógica de los dispositivos de red y cómo estos procesan los paquetes de datos [5]. Una de las características de las SDN es el conocimiento de la topología de la red, que permite, entre otras cosas, establecer rutas previas al comienzo de un flujo de datos, pudiendo crear rutas específicas para cada uno de ellos. Dado que el tráfico de red de una aplicación de HPC en un centro de datos suele tener patrones concretos (*Broadcast*, *AllReduce*...), podemos beneficiarnos del conocimiento de la red para crear rutas adaptadas a cada aplicación de HPC que se ejecute.

Optimizar el tiempo de comunicación de los datos es el objetivo principal de este trabajo, que se puede dividir en los siguientes objetivos parciales:

- Programación de un sistema de notificación de procesos MPI al controlador SDN.
- Programación del controlador SDN para el establecimiento de rutas.
- Realización de pruebas y análisis de resultados.

En este documento se expone el procedimiento mediante el cual, los procesos de una aplicación basada en OpenMPI se inicializan al comenzar la ejecución y comparten su información de direccionamiento para poder comunicarse. Aprovechando este intercambio de información entre procesos durante la inicialización de la aplicación, se propone una modificación de OpenMPI para la integración con la red SDN, que permite notificar esta información de direccionamiento al controlador de la red para establecer las rutas necesarias. Así mismo, se propone un balanceador de tráfico basado en el *peso* de los enlaces, según el número de flujos que utiliza cada enlace en la red, con el objetivo de separar los tráficos entre distintos procesos en rutas diferentes para aprovechar la redundancia de enlaces y proporcionar balanceo de carga a las aplicaciones. Además, se presentan una serie de pruebas realizadas que muestran el funcionamiento del balanceo de carga en un entorno de prueba basado en dispositivos Raspberry Pi y mediante redes virtuales en Mininet [6].

El contenido del presente trabajo se estructura de la siguiente manera. En la Sección II se introduce el estado del arte sobre la interfaz de paso de mensajes (MPI) y las redes definidas por software; en la Sección III se plantea la metodología de trabajo y el desarrollo de la propuesta; en la Sección IV se

realiza una evaluación de la misma; por último, en la Sección V concluimos con algunas reflexiones sobre el trabajo realizado, los resultados obtenidos y el trabajo futuro.

II. Estado del Arte

A. Message Passing Interface

El paso de mensajes es un modelo de programación paralela utilizado en computadores con memoria distribuida. Cada proceso posee un conjunto de variables privadas independientes del resto y utiliza un espacio de direcciones propio. El paso de mensajes entre procesos permite copiar los datos del espacio de direcciones de un proceso al espacio de direcciones de otro. Este modelo de programación es ampliamente utilizado en arquitecturas *Multiple Instruction Multiple Data*, donde múltiples unidades de cómputo independientes ejecutan diferentes instrucciones sobre distintos datos. Los conjuntos de datos pueden encontrarse en espacios de memoria compartida o distribuida. [4].

MPI (*Message Passing Interface*) [7] establece el estándar de una interfaz de paso de mensajes que facilita el uso de este modelo de programación a los desarrolladores a modo de API (*Application Programming Interface*). En este trabajo utilizaremos la librería OpenMPI [8], una implementación de MPI de código abierto para los lenguajes C y Fortran. Estas librerías ofrecen al programador la abstracción de la gestión de procesos, la red de interconexión, el uso de *sockets*, el uso de memoria compartida y todo aquello ajeno al programa que se desea implementar.

B. Software Defined Networking

Las Redes Definidas por Software se caracterizan por la separación del plano de control del plano de datos de la red. Esto significa que la gestión y configuración de la red está totalmente separada del reenvío de mensajes, y son realizadas por agentes independientes.

En el plano de control se determinan las rutas que deben seguir los paquetes según la lógica deseada, mientras que en el plano de datos se redirigen los paquetes según las rutas generadas por el plano de control. Estas rutas se traducen en reglas que se almacenan en tablas [5]. El reenvío de paquetes en el plano de datos se realiza de manera mucho más rápida, pues cada dispositivo solo tiene que examinar determinados campos de los mensajes entrantes para determinar a que flujo de datos pertenece, y determinar la acción a realizar sobre él según las reglas programadas en las tablas.

Las reglas son programadas en los dispositivos de red por el controlador, situado en un servidor o máquina dedicada. Él es el encargado de analizar el estado de la red, y de calcular de rutas para nuevos flujos de datos siguiendo criterios de QoS, energía, etc. Dichas reglas son instaladas en las "tablas de reenvío"de cada uno de los dispositivos de red [5].

Para estandarizar la comunicación entre el plano de control y de datos, en 2008 se propuso el estándar abierto OpenFlow [9]. Este se basa en "*FlowRules*" o reglas de flujo, para especificar el comportamiento del reenvío.

Las *FlowRules* son una abstracción de las tablas de reenvío, que consisten en un par *Match-Action*. En la sección *Match* se especifica una regla en la que uno o varios campos de un paquete cumplan con unos requisitos, por ejemplo, que un paquete contenga una dirección IP destino concreta. Si se produce el *Match* se realizará la acción correspondiente. Las acciones actúan sobre el paquete, reenviándolo por uno o varios puertos, descartando el paquete, modificando alguno de sus campos, etc.

B.1 El plano de control centralizado

Para la implementación del controlador se introduce el concepto de *Network Operating System* (NOS) [5]. Un NOS es un sistema que proporciona al usuario una visión del estado de la red, abstrayéndolo del proceso de toma de estadísticas de funcionamiento de los dispositivos de red y proporcionando también una API para el establecimiento de "*flow rules*" en dichos componentes. La idea de un NOS es que los administradores de red desarrollen aplicaciones de gestión y configuración de esta basándose en las funcionalidades que les proporciona el NOS y abstrayéndose de los dispositivos de red en sí, y de la comunicación con ellos. Actualmente existen varios NOS, entre los que cabe destacar Ryu [10] y POX[11] implementados en Python, OpenDaylight [12], FloodLight [13] y ONOS[14] basados en Java, o NOX [15] basado en C.

B.2 El plano de datos programable

Un *switch* SDN con soporte OpenFlow se caracteriza por utilizar una o más tablas de flujo o *flow tables*, y uno o más canales de comunicación con el controlador [16]. El controlador se comunicará con los *switches* mediante mensajes del protocolo OpenFlow para añadir, modificar o eliminar *flow entries o FlowRules* en las tablas de flujo. Estas se utilizan para reenviar, eliminar y/o modificar los mensajes recibidos. Existen implementaciones software de *switches* SDN utilizadas para emular redes virtuales. Entre las más conocidas hay que destacar Open vSwitch [17].

C. Trabajos Relacionados

En relación al uso de técnicas SDN para optimizar el paso de información entre los procesos de un programa desarrollado usando MPI, [18] propone la mejora de operaciones colectivas como *Broadcast* y *AllReduce* cambiando la dirección MAC destino del mensaje MPI, por información sobre la aplicación MPI ejecutada, el tipo de operación colectiva que se realiza, el comunicador y los procesos origen y destino del mensaje.

Esta información codificada en el campo de MAC destino, es utilizada por el controlador para establecer las reglas en cada "*flow rule*". Estas reglas son calculadas según los algoritmos de encaminamiento

propuestos en [19] y [20]. Un inconveniente de estas propuestas es que se basan en realizar las modificaciones de los campos Ethernet a nivel de "*kernel*" del sistema operativo en cada nodo de cómputo, con el consiguiente aumento de carga computacional.

III. Metodología y desarrollo de la implementación propuesta

Mientras que [18] proponía una solución de marcado de paquetes mediante la modificación de los mismos a través del kernel del sistema operativo, nuestra propuesta evita la modificación de los paquetes. En nuestro caso, optamos por modificar la librería OpenMPI, ya que no es raro que en los centros de cálculo, las herramientas que se utilizan sean adaptadas y compiladas específicamente para las máquinas que ejecutarán las aplicaciones. Además, al contrario que los trabajos citados anteriormente, no buscamos mejorar algunas llamadas concretas de la librería como *AllGather* o *Broadcast*, si no optimizar rutas de manera general para cualquier llamada y tarea de OpenMPI.

Nuestro trabajo se basa en notificar la dirección y puerto de cada proceso de OpenMPI cuando se ponen a la escucha durante la inicialización del entorno, antes de comenzar la ejecución del programa. Notificar esta información al controlador de la red permite establecer rutas de manera proactiva, evitando perder tiempo durante los establecimientos de conexión.

En el caso de este trabajo, utilizamos una red en la que las conexiones entre origen y destino se basan en el protocolo TCP. Tras analizar el funcionamiento del envío de datos de OpenMPI sobre TCP, observamos que todos los tipos de envíos de datos implementados, tanto punto a punto como colectivos, se realizan en última instancia mediante conexiones cliente-servidor entre cada proceso, por lo que nos centraremos en identificar estas conexiones y crear rutas para cada una de ellas, sin conocer el tipo de llamada (colectiva o punto a punto) al que pertenece el tráfico.

A. *Notificación de procesos durante la inicialización de OpenMPI*

Durante la inicialización de OpenMPI, cuando los procesos se ponen a la escucha en los puertos disponibles en cada nodo, cada proceso envía su información (dirección IP, número de puerto, y otra información relativa a la ejecución de OpenMPI) al resto de nodos a través de SSH, y otro tipo de conexiones conocidas como *Out of Band* (OOB). Nuestra propuesta aprovecha que los procesos van a compartir esta información para crear una notificación que podrá ser capturada por los dispositivos SDN, y reenviada al controlador, como podemos observar en el diagrama resumen de MPI_Init en la *Figura 1*, donde observamos en color verde la aportación de código que permite realizar esta notificación, y en color azul la secuencia de acciones para inicializar el componente TCP de OpenMPI de manera simplificada.

La notificación enviada irá encapsulada en un mensaje UDP, ya que, al no establecerse conexión a nivel de transporte, nos permite realizar envíos de información de manera rápida. El mensaje UDP va dirigido a una IP y puerto específicos, de tal modo que, en los *switches* se instalará una regla para dicha IP y puerto, que reenviará dichos mensajes al controlador, donde se analizará su contenido. El mensaje UDP contiene, encapsulados en el *payload*, la dirección IP y el puerto de escucha de cada proceso OpenMPI.

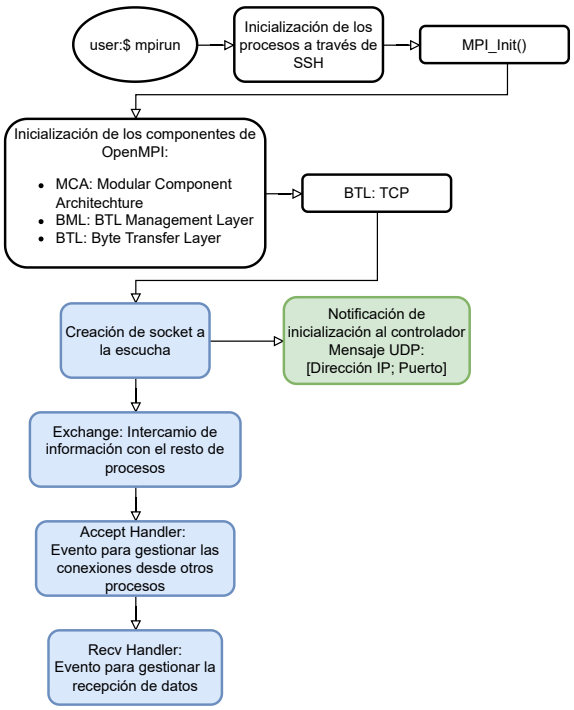


Figura 1: Diagrama de flujo simplificado de la inicialización en MPI_Init

B. *Integración del controlador SDN y asignación de rutas*

El controlador SDN se encargará de analizar los paquetes que lleguen desde los dispositivos de red y tomar decisiones en función de estos. Además gestionará las rutas activas en la red, instalando las FlowRules necesarias en los *switches*, y llevará un control de los procesos de OpenMPI activos, que le permitirá seleccionar rutas balanceando la carga según el uso de los enlaces.

Mientras el controlador almacena los procesos notificados con su información de direccionamiento, establecerá las rutas entre los mismos, seleccionando el camino más corto teniendo en cuenta el número de saltos y la suma del uso de los enlaces que componen la ruta. Cada una de estas rutas resultará en dos reglas (de origen a destino y viceversa) en cada *switch* que la compone, donde cada mensaje que haga *match* con el destino especificado (uno de los procesos notificados), será redirigido por un puerto del *switch*. La interacción entre las notificaciones implementadas en OpenMPI y la SDN puede verse en el diagrama de secuencia de la *Figura 2*. La lógica de la aplicación de control ejecutada en el controlador puede verse en el diagrama de bloques de la *Figura 3*, que se describe a continuación:

1. Al iniciar la aplicación de control de red desarrollada, se instalan un conjunto de reglas en todos los dispositivos de red que permiten identificar los paquetes entrantes que corresponden a los protocolos IPv4, ARP y LLDP. El protocolo LLDP permite identificar los *hosts* conectados a la red.
2. Cuando un paquete llega a un *switch*, existen 2 opciones:
   a) Si no existe ninguna FlowRule en el *switch* (por ejemplo, cuando acaba de iniciarse la aplicación de control) o no hace *match* con ninguna de las reglas existentes (es un paquete generado por un tráfico nuevo en la red que no existía previamente), el paquete se envía desde el *switch* al controlador para ser analizado.
   b) Si existe alguna FlowRule que hace *match* con las características del paquete (protocolo, origen, destino...), se reenvía según indica la FlowRule.
3. Al recibir un paquete en el controlador, habrá que diferenciar si es de tipo ARP o IPv4. Si es de tipo ARP, se obtendrá el destino y se encaminará directamente, para evitar inundar la red con tráfico ARP. Si es de tipo IPv4, habrá que diferenciar entre los protocolos TCP o UDP.
   a) Si el paquete recibido en el controlador es TCP, se analizará el origen y el destino del paquete. Si el origen y el destino están bajo el mismo *switch*, se instalarán las reglas necesarias para encaminar el tráfico entre los dos puertos adecuados en ese *switch*. Si el origen y el destino están en *switches* distintos, se calcularán las rutas origen→destino y destino→origen. En este caso, se utilizará el balanceo de carga implementado, aunque podemos dar prioridad al tráfico de los procesos de OpenMPI. Se instalarán las reglas adecuadas en todos los *switches* implicados en las rutas obtenidas.
   b) Si el paquete es UDP, se analizará si es una notificación de un proceso de OpenMPI. Si no lo es, el comportamiento es el mismo al del protocolo TCP mencionado en el punto anterior. Si es una notificación de un proceso de OpenMPI, la información de direccionamiento recibida en el mensaje UDP se tomará como un posible destino de tráfico de OpenMPI. Después, se calcularán las rutas posibles entre cada nuevo proceso de OpenMPI y el resto notificados previamente. Las rutas se calcularán con el algoritmo de balanceo de carga desarrollado.

El balanceo de carga desarrollado consiste en una asignación de pesos a cada uno de los enlaces existentes en la red, según la cantidad de FlowRules que utilizan los enlaces para encaminar los paquetes. El funcionamiento de la asignación de pesos se basa en las siguientes reglas:

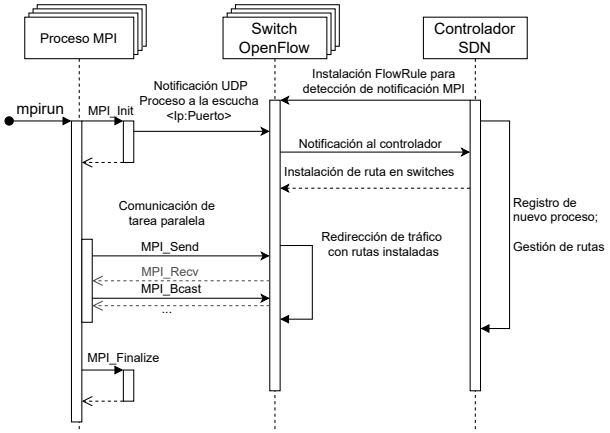


Figura 2: Diagrama de secuencia de la interacción entre los procesos de MPI y el controlador de la red

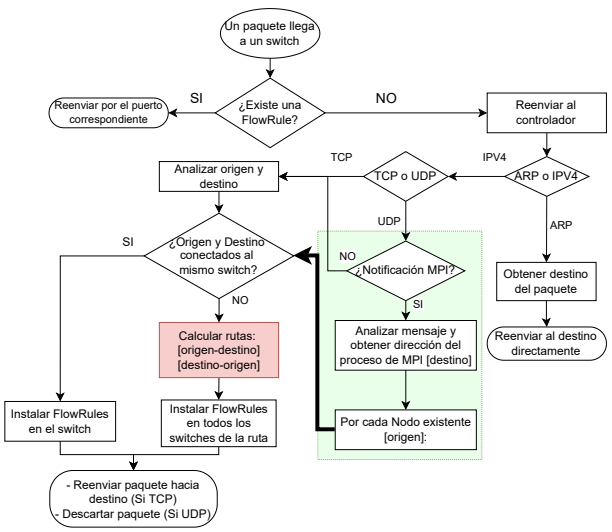


Figura 3: Diagrama de flujo de la lógica del controlador implementado

- Se utiliza una estructura de datos de tipo diccionario para almacenar los enlaces utilizados (ID de cada *link* de la topología) y el número de usos de cada uno.
- Otra estructura del mismo tipo que la anterior se utiliza para almacenar cada ID de las FlowRules instaladas en la red, y el ID del *link* que utiliza cada FlowRule para reenviar los paquetes.
- Cuando se calcula una nueva ruta entre origen y destino, se obtienen todas las rutas posibles. Cada ruta contiene una lista de *links*. Obtenemos el uso actual de cada *link*, y seleccionamos la ruta con los *links* menos utilizados.
- Una vez seleccionada una ruta, se instalan las FlowRules en la red. Por cada FlowRule se almacenará el *link* utilizado y se aumentará el uso del mismo en una unidad en la estructura de datos mencionada en primer lugar.
- Cuando el tráfico finaliza, las FlowRules instaladas se eliminan de los dispositivos de red (pueden ser eliminadas con una cuenta atrás desde el momento en el que deja de haber tráfico, o manualmente). Se decrementará el uso de los enlaces utilizados por las FlowRules eliminadas.

De este modo, siempre que se vaya a calcular una nueva ruta, se utilizará la que tenga los *links* menos utilizados. Si existen rutas con *links* con el mismo uso, se escogerá la primera ruta obtenida por defecto, creando así una ruta con mayor uso, y obligando a seleccionar otra alternativa de menor uso para la siguiente ruta a definir. Un resumen de la lógica del balanceo de carga anteriormente descrita se detalla en el pseudocódigo de la *Figura 4*.

IV. Evaluación del sistema

A. Entorno de trabajo

Para desarrollar el controlador de red y realizar las pruebas, el entorno de trabajo ideal constaría de una red con una topología de enlaces redundantes entre los dispositivos de interconexión, p.e. *Fat Tree*. Sin embargo, dado que no disponíamos de los recursos hardware necesarios para la experimentación en este entorno, se ha optado por implementar la topología de red utilizando dos escenarios. En el primero se utilizaron dispositivos Raspberry Pi con una implementación de OpenVSwitch, como *switches*, utilizando cada puerto USB con un adaptador USB-Ethernet. Algunas de las ventajas de utilizar estos dispositivos son: el bajo coste en comparación con el hardware dedicado, el bajo consumo energético, la flexibilidad que otorga un dispositivo de estas características a la hora de utilizarlo como un nodo de cómputo y como un *switch*, y la gran variedad de software open-source que podemos utilizar, además de la amplia comunidad de desarrollo que lo rodea. Sin embargo, encontraremos limitaciones hardware como la baja memoria disponible, un procesador modesto, y, en nuestro caso, la particularidad del bus USB único, que comparten todos los puertos del dispositivo, y que limitará el ancho de banda disponible que podrá ser usado a la hora de actuar como *switch*. En el segundo escenario se utilizó el simulador Mininet.

En ambos casos la topología de red utilizada se puede observar en la *Figura 5*. A pesar de no ser un *Fat Tree*, contiene enlaces redundantes entre *switches* que permiten desarrollar y probar la funcionalidad principal del controlador de red, que consiste en asignar rutas diferentes para distintos tráficos, y por tanto aplicar balanceo de carga para no saturar los mismos enlaces, como ocurre con los *switches* de capa 2.

El software utilizado para la ejecución del controlador de red ha sido ONOS, y la aplicación de control ha sido desarrollada en Java. El controlador se ha ejecutado en una máquina con Ubuntu 20.04, Ryzen 7 4800H y 16GB de memoria RAM. OpenMPI se ejecutó en las Raspberry Pi utilizadas como nodos de cómputo en la topología. El modelo de Raspberry Pi utilizado en este trabajo es la *Raspberry Pi 4 Model B*, de 1GB de RAM, a excepción del nodo maestro, que es una *Raspberry Pi 3 Model B+*, como se puede observar en la *Figura 5*.

B. Especificación de las pruebas realizadas

Las pruebas ejecutadas para la obtención de resultados se basan en diversos métodos de transmisión de datos en OpenMPI utilizando la configuración de *switches* de capa 2, y el controlador desarrollado. Con ello se pretende observar las diferencias en el funcionamiento de la transmisión de los datos mediante la utilización de los enlaces disponibles en la red, el uso del ancho de banda de cada uno de los enlaces por cada conexión existente y las limitaciones características de los dos sistemas utilizados para ejecutar las pruebas, tanto el hardware físico, como el simulado.

El detalle de las pruebas realizadas es el siguiente:

- Producto de matrices con OpenMPI utilizando comunicación punto a punto: Se ejecutará un producto de matrices de tamaño 1024x1024 con 4 procesos distribuidos en la topología de la *Figura 5*, usando las primitivas de comunicación punto a punto `MPI_Send()` y `MPI_Recv()`. El producto de dos matrices A y B consistirá en el reparto de A entre el número de procesos que ejecuten el programa paralelo, y el envío completo de la matriz B a estos mismos procesos. Después, los fragmentos de la matriz resultado C se recogen en el proceso "maestro". Se ejecutará utilizando una configuración de *switches* de capa 2, y la configuración del controlador desarrollado para observar el comportamiento de las rutas utilizadas en cada conexión. Esta prueba se ejecutará solamente en los dispositivos Raspberry Pi.
- Producto de matrices con OpenMPI utilizando comunicación colectiva: Se ejecutará el producto de matrices de tamaño 1024x1024 con 4 procesos distribuidos en la topología mencionada anteriormente, usando las primitivas de comunicación colectivas `MPI_Bcast()`, `MPI_Scatter()` y `MPI_Gather()`. Se ejecutará utilizando las configuraciones de dispositivos de red mencionadas anteriormente para observar el comportamiento de las rutas utilizadas en cada conexión. Esta prueba se ejecutará solamente en los dispositivos Raspberry Pi.
- Envío de mensajes de 1 GB con OpenMPI: Se realizará el envío de mensajes de 1 GB de datos entre dos pares de nodos de manera simultánea. Permitirá apreciar mejor el tiempo empleado en las comunicaciones, utilizando la configuración de *switches* de capa 2 y el controlador desarrollado. Esta prueba se lanzará sólo en Mininet, debido a las limitaciones de reserva de memoria de los dispositivos Raspberry Pi.

C. Evaluación de los experimentos

A continuación se muestran los experimentos realizados sobre la topología de la *Figura 5*. Los datos mostrados son los correspondientes al *switch* SW1 en las dos primeras pruebas sobre el producto de matrices, y sobre SW2 en el envío de mensajes de 1GB

```
global ActiveLinks<LinkId, Weight>
global ActiveFlowRules<FlowRuleId, LinkId>
global Topology

func installBalancedPath(Source, Destination):
    list[link] balanced_path
    list[path] possible_paths <- Topology.getPath(Source, Destination)

    balanced_path <- selectBalancedPaths(possible_paths)

    if exist balanced_path:
        for each link in balanced_path:
            //source-destination route
            FlowId <- install_flowrule(link.src.switch, Match: [srcHost, dstHost], Action: to port X)
            ActiveFlowRules.add(FlowId, link)
            ActiveLinks.merge(link, sum:1.0)
            //reverse route
            FlowId <- install_flowrule(link.dst.switch, Match: [dstHost, srcHost], Action: to port Y)
            ActiveFlowRules.add(FlowId, link)
            ActiveLinks.merge(link, sum:1.0)

func selectBalancedPaths(paths):
    var pathScore <- Double.MAX_VALUE
    var bestPath
    for each path in paths:
        var auxPathScore <- 0.0
        for each link in path:
            auxPathScore <- auxPathScore + ActiveLinks.get(link)
            if auxPathScore < pathScore:
                pathScore <- auxPathScore
                bestPath <- path

    return bestPath
```

Figura 4: Pseudocódigo del algoritmo de balanceo de carga para el controlador desarrollado

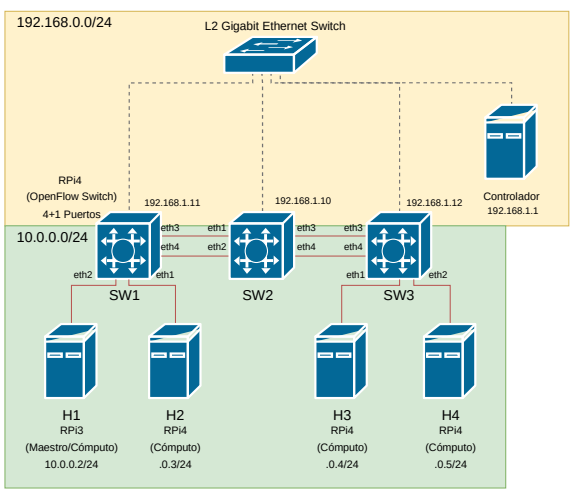


Figura 5: Topología final implementada con Raspberry Pi.

entre pares de nodos.

C.1 Producto de matrices utilizando comunicaciones punto a punto

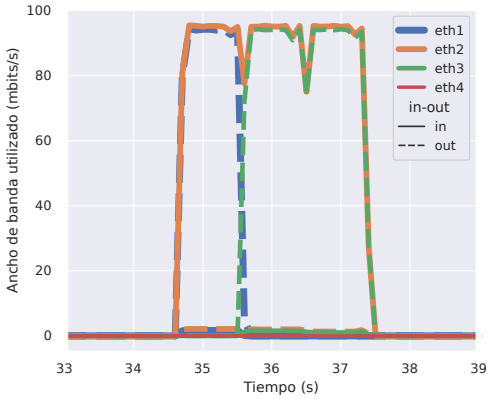
En primer lugar, observaremos los datos obtenidos del *switch* SW1 con la configuración de capa 2 de la topología de red. Este gráfico indica el tráfico de entrada y salida en el *switch*, ya sea entre SW1 y SW2 o los *hosts* H1 y H2, ejecutando el producto de matrices con OpenMPI y las primitivas de comunicación punto a punto. En la *Figura 6* observamos los tráficos generados por el producto de matrices en dos instantes determinados. En primer lugar, el tráfico generado al enviar los datos a todos los procesos participantes en el producto de matrices (*Figura 6a*), donde observamos un tráfico entrante al *switch*, perteneciente a H1 conectado a *eth2* (línea naranja), tráfico saliente hacia H2 (línea azul discontinua) conectado a *eth1*, y hacia el siguiente *switch* por *eth3* (línea verde discontinua), con destino a H3 y H4. Podemos ver diferenciados los tres mensajes enviados a H2, H3 y H4, que contienen la matriz A completa y el fragmento de B que le corresponde a cada proceso.

También en la *Figura 6b* observamos la recepción de los resultados parciales desde cada proceso. Existen dos flujos de datos de entrada, siendo el azul el mensaje entrante desde H2, y la línea verde los dos mensajes entrantes desde H3 y H4. La línea naranja discontinua es el tráfico de salida conjunto hacia H1. Con estos dos gráficos se observa la comunicación completa del producto de matrices con comunicaciones punto a punto, donde observamos que no se realiza un balanceo de carga hacia el siguiente *switch*. Con un correcto balanceo del tráfico se podrían utilizar *eth3* y *eth4* para cada conexión de H1 con H3 y H4, pero sólo se está utilizando el enlace por el puerto *eth3*.

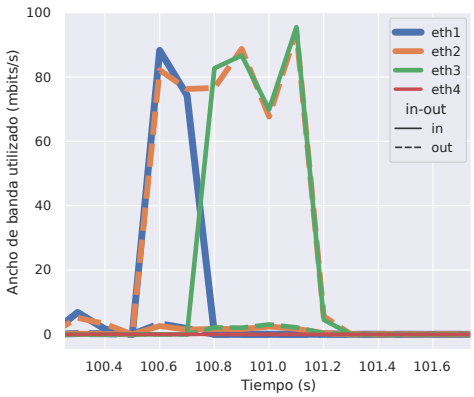
En la *Figura 7a* observamos de nuevo el envío de los datos desde H1 al resto de nodos. En este caso, al enviar los datos a H3 y H4 se utilizan las dos conexiones existentes hacia SW2 (*eth3* y *eth4*), por lo que se aprecia el correcto funcionamiento del balanceo de carga por cada proceso existente. Además, este balanceo del tráfico entre los enlaces existentes se mantiene en la recogida de los resultados desde H3 y H4, donde también se utilizan estos mismos puertos desde SW2 (*Figura 7b*).

Estas pruebas no reflejan una mejora en el tiempo de comunicación del programa ya que se realizan con un tamaño de problema pequeño, adecuado al hard-

Producto de matrices Punto a Punto en SW1 Switch L2



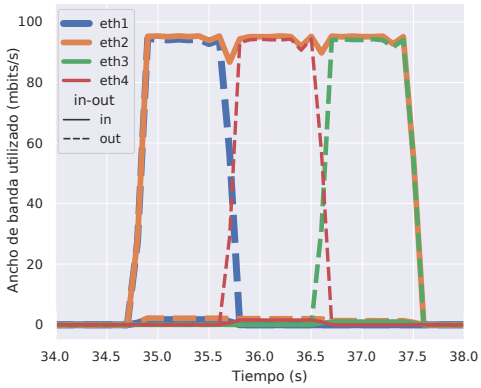
(a) Envío de datos a los nodos (MPI_Send)



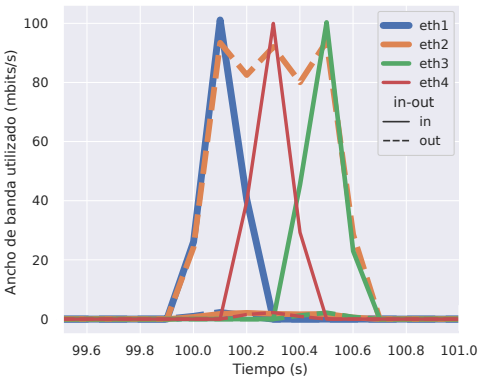
(b) Recepción de resultados (MPI_Recv)

Figura 6: Tráfico del producto de matrices en SW1. Comunicaciones punto a punto con Raspberry Pi. Configuración de *switches* de capa 2.

Producto de matrices Punto a Punto en SW1 MPI Controller



(a) Envío de datos a los nodos (MPI_Send)



(b) Recepción de resultados (MPI_Recv)

Figura 7: Tráfico del producto de matrices en SW1. Comunicaciones punto a punto con Raspberry Pi. Configuración del controlador desarrollado.

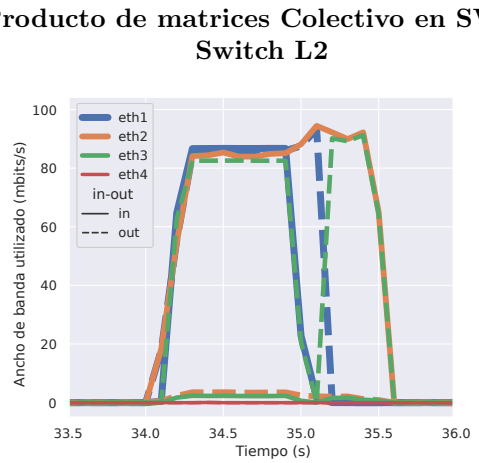
ware de las Raspberry Pi, y buscan apreciar el correcto funcionamiento del balanceo de carga utilizando las comunicaciones punto a punto de OpenMPI.

C.2 Producto de matrices utilizando comunicaciones colectivas

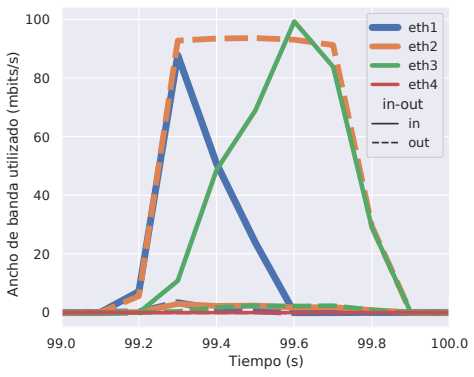
En la *Figura 8* observamos tanto el reparto de datos a los procesos para realizar el producto de matrices utilizando primero un *broadcast* seguido de *scatter* (*Figura 8a*), como la recogida de resultados utilizando *gather* (*Figura 8b*). Durante el reparto de datos, el primer bloque de tráfico que se observa representa el *broadcast*, entrante por *eth2* desde H1, y saliente por *eth1* hacia H2. Este envío de datos en *broadcast* funciona a modo de "cadena" (aunque existen otros patrones como la comunicación de procesos en árbol), por lo que H2 enviará los datos a H3, y después serán enviados a H4. Los dos picos siguientes representan el reparto de datos con *scatter*, el primero por *eth1*, y el segundo por *eth3* hacia H3 y H4. En este caso vemos como no se están utilizando los dos enlaces existentes hacia SW2 para enviar datos entre los pares H1-H3 y H1-H4. Del mismo modo sucede en la recogida de datos, donde solo recibimos datos desde SW2 por el puerto *eth3*, lo que nos indica que no se está realizando el balanceo de carga ya que estamos utilizando la configuración de *switches* de capa 2.

En la *Figura 9* observamos el uso de todos los enlaces en SW1, utilizando tanto *eth3* como *eth4* para enviar los datos de *scatter* hacia SW2 con destino a H3 y H4, además del primer bloque de *broadcast*. En este caso puede apreciarse mejor el comportamiento de envío de datos en "cadena". H1 empieza enviando sus datos en *broadcast* hacia H2, el tráfico entrante por *eth2* sale por *eth1*, y se reenvía por *eth3*. Después comienza cada *scatter* por separado, y cada uno se realiza por un enlace diferente. Del mismo modo sucede en la recepción de resultados, donde se aprecia tráfico de entrada desde tres puertos diferentes, hacia H1. Esto indica que el balanceo de carga con las llamadas de comunicación colectiva funciona correctamente, aprovechando todos los enlaces disponibles en el *switch* por todas las conexiones existentes.

Estas pruebas no reflejan una mejora en el tiempo de comunicación del programa ya que se realizan con un tamaño de problema pequeño, adecuado al

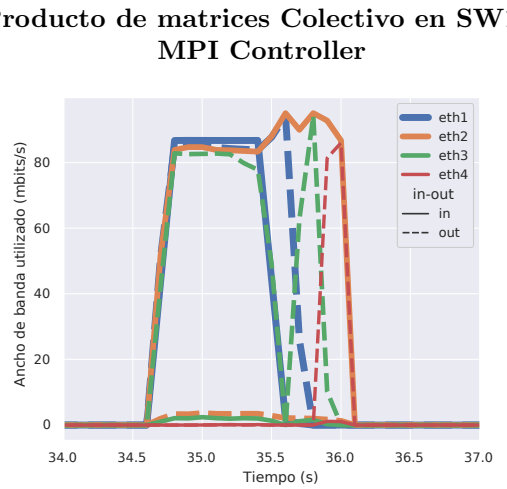


(a) Envío de datos a los nodos (MPI_Send)

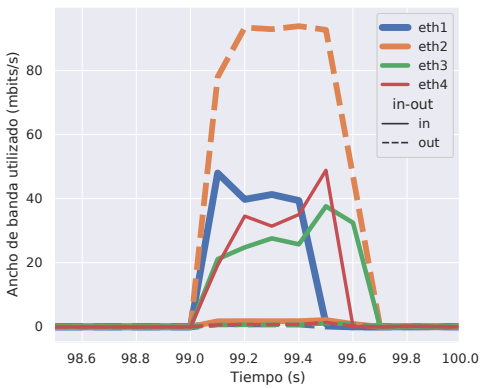


(b) Recepción de resultados (MPI_Recv)

Figura 8: Tráfico del producto de matrices en SW1. Comunicaciones colectivas con Raspberry Pi. Configuración de *switches* de capa 2.



(a) Envío de datos a los nodos (MPI_Send)



(b) Recepción de resultados (MPI_Recv)

Figura 9: Tráfico del producto de matrices en SW1. Comunicaciones colectivas con Raspberry Pi. Configuración del controlador desarrollado.

hardware de las Rasbperry Pi, y buscan apreciar el correcto funcionamiento del balanceo de carga utilizando las comunicaciones colectivas de OpenMPI.

C.3 Envío simultáneo de mensajes de 1GB entre pares de nodos

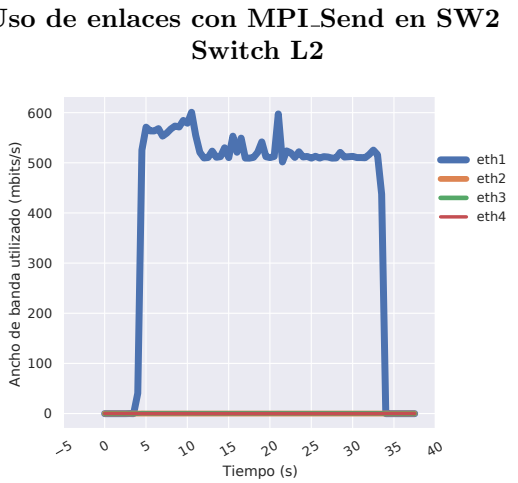
Esta última prueba no se ha podido realizar en los dispositivos Raspberry Pi debido a la falta de memoria RAM para poder reservar el espacio necesario y poder enviar los datos entre los nodos. Se ha realizado en el emulador Mininet, donde podemos comprobar el funcionamiento del balanceo de carga y el aprovechamiento total de los enlaces. La prueba consiste en realizar, mediante un programa con MPI, un envío de dos mensajes de 1 GB entre dos pares de nodos de la topología. Así, el nodo H1 enviará datos al nodo H3, y H2 enviará datos a H4, según el esquema de la *Figura 5*.

En la *Figura 10* podemos observar el uso de los enlaces del *switch* raíz (SW2) utilizando la configuración de *switches* de capa 2, donde tenemos el enlace *eth1* recibiendo tráfico de entrada desde el *switch* donde están conectados los nodos que envían el tráfico (H1 y H2). A su vez, se está reenviando el tráfico de salida hacia los nodos destino, por el enlace *eth3*. Como podemos observar, está utilizando solamente un enlace de entrada y uno de salida para dos envíos de datos simultáneos, por lo que no están utilizando todos los enlaces disponibles para cada una de las conexiones.

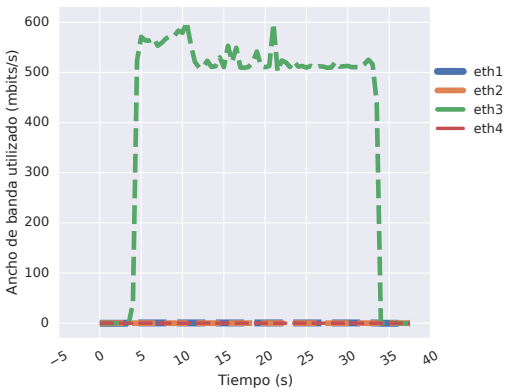
Sin embargo, cuando utilizamos el controlador desarrollado, se puede observar en la *Figura 11* cómo se utilizan dos enlaces de entrada (azul y naranja) y de salida (verde y rojo) para ambas conexiones entre nodos, realizando un correcto balanceo de carga. Ambas conexiones utilizan el máximo ancho de banda permitido, y además se reduce el tiempo de comunicación de los procesos, pasando de los 30 segundos aproximadamente en el caso anterior, a unos 15 segundos aproximadamente utilizando el controlador desarrollado.

V. Conclusiones

La metodología propuesta para adoptar las SDN en un entorno de ejecución de aplicaciones HPC permite, como hemos visto tras el análisis de las pruebas realizadas, mejorar el aprovechamiento del ancho de

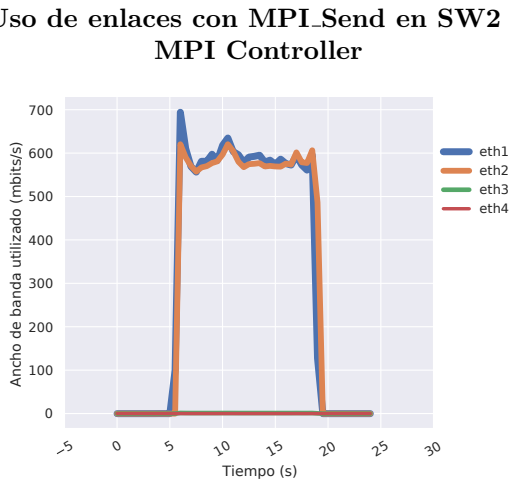


(a) Tráfico de entrada

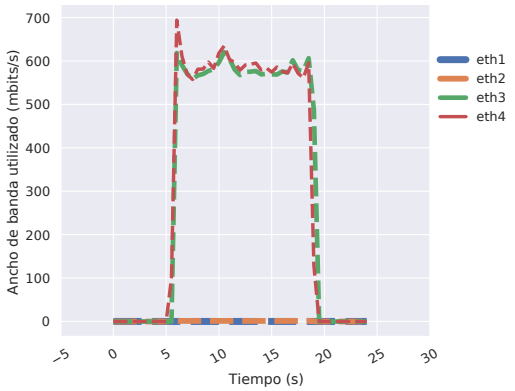


(b) Tráfico de salida

Figura 10: Tráfico de MPI_Send() enviando 1GB de datos entre dos pares de nodos en el *switch* raíz (SW2) con Mininet. Configuración de *switches* de capa 2.



(a) Tráfico de entrada



(b) Tráfico de salida

Figura 11: Tráfico de MPI_Send() enviando 1GB de datos entre dos pares de nodos en SW2 con Mininet. Configuración del controlador desarrollado.

banda disponible en una topología de red con enlaces redundantes. Además, es posible complementar esta funcionalidad con las aplicaciones que hacen uso de las librerías de paso de mensajes, permitiendo separar conexiones entre nodos a través de rutas diferentes. La programación de un algoritmo adecuado de balanceo de tráfico permitirá sacar partido de esta funcionalidad, dando mayor prioridad a flujos de datos concretos, o estableciendo niveles de QoS deseados, según el estado de la red.

El paso lógico tras del desarrollo de este trabajo será realizar pruebas en un entorno real, utilizando *switches* con mayor capacidad y nodos de cómputo más potentes sobre los que ejecutar una aplicación paralela basada en el paso de mensajes, y analizar los resultados obtenidos para verificar la mejora de rendimiento del sistema propuesto frente a la utilización de *switches* de capa 2.

Agradecimientos

Este trabajo ha sido desarrollado dentro del marco del proyecto PID2021-123627OB-C52, financiado por la Agencia Estatal de Investigación del Ministerio de Ciencia e Innovación y la Unión Europea MCIN/AEI/10.13039/501100011033/FEDER, UE; "Otra forma de hacer Europa", el Fondo Europeo de Desarrollo Regional y Junta de Comunidades de Castilla La Mancha (proyecto SBPLY/21/180501/000195), y los proyectos 2020-GRIN-28846 y 2020-GRIN-28708. Agradecemos a los doctores Enrique S. Quintana y a Manuel Francisco Dolz por arrojar un rayo de luz al inicio de este trabajo que nos iluminó lo suficiente para continuar con el mismo.

Referencias

[1] KA Wetterstrand, *DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP).*

[2] Atos SE, *Atos boosts supercomputing power by 10 for Spanish State Meteorological Agency AEMET*, 2021, Accessed: Jul, 2022. [Online]. Available: `https://atos.net/en/2021/press-release_2021_02_23/atos-boosts-supercomputing-power-by-10-for-spanish-state-meteorological-agency-aemet` .

[3] International Research Institute of Disaster Science, Tohoku University, Earthquake Research Institute, The University of Tokyo, Fujitsu Laboratories Ltd., "Fujitsu Leverages World's Fastest Supercomputer 'Fugaku' and AI to Deliver Real-Time Tsunami Prediction in Joint Project," 2021, Accessed: Jun. 3, 2021.

[4] A. Grama, V. Kumar, A. Gupta, and G. Karypis, *Introduction to Parallel Computing*, Addison-Wesley, 2003.

[5] Larry Peterson, Carmelo Cascone, Bruce Davie, Brian O´Connor, and Thomas Vachuska, *Software-Defined*

Networks: A Systems Aproach, Systems Approach LLC, 2022.
[6] Mininet Project, "Mininet," `http://mininet.org/`, Accessed: May 26, 2023.
[7] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 2.1*, chapter 1, 2008.
[8] The Open MPI Project, "OpenMPI," `https://www.open-mpi.org/`, Accessed: May 26, 2023.
[9] Open Networking Foundation, *OpenFlow Switch Specification Version 1.5.1*, chapter 7, 2015.
[10] Nippon Telegraph and Telephone Corporation, "Ryu," `https://ryu.readthedocs.io/`, 2011-2014, Accessed: May 26, 2023.
[11] McCauley et al., "Installing POX - POX Manual," `https://noxrepo.github.io/pox-doc/html/`, 2015, Accessed: May 26, 2023.
[12] OpenDaylight Project, "Opendaylight," `https://www.opendaylight.org/`, 2016-2023, Accessed: May 26, 2023.
[13] Project Floodlight, "Floodlight controller," `https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview`, 2018, Accessed: May 26, 2023.
[14] Open Networking Foundation, "Onos," `https://wiki.onosproject.org/`, Accessed: May 26, 2023.
[15] "Nox repo," `https://github.com/noxrepo/nox/`.
[16] Open Networking Foundation, *OpenFlow Switch Specification Version 1.5.1*, chapter 2, 2015.
[17] The Linux Foundation, "Open vSwitch," `https://www.openvswitch.org/`, 2022, Accessed: May 26, 2023.
[18] K. Takahashi, S. Date, D. Khureltulga, Y. Kido, H. Yamanaka, E. Kawai, and S. Shimojo, "UnisonFlow: A Software-Defined Coordination Mechanism for Message-Passing Communication and Computation," *IEEE Access*, vol. 6, pp. 23372–23382, 2018.
[19] Hiroaki Morimoto, Khureltulga Dashdavaa, Keichi Takahashi, Yoshiyuki Kido, Susumu Date, and Shinji Shimojo, "Design and Implementation of SDN-enhanced MPI Broadcast Targeting a Fat-Tree Interconnect," in *2017 International Conference on High Performance Computing Simulation (HPCS)*, 2017, pp. 252–258.
[20] Pisit Makpaisit, Kohei Ichikawa, Putchong Uthayopas, Susumu Date, Keichi Takahashi, and Dashdavaa Khureltulga, "MPI_Reduce algorithm for OpenFlow-enabled network," in *2015 15th International Symposium on Communications and Information Technologies (ISCIT)*, 2015, pp. 261–264.
[21] Brian W Barrett, *Open MPI Data Transfer*, 12 2012.
[22] George Bosilca, Galen Shipman, and Tim Woodall, *Open MPI P2P Architecture*, 4 2006.

Análisis de métodos de redistribución de datos para aplicaciones MPI maleables

Iker Martín-Álvarez[1], José I. Aliaga[1], Maribel Castillo[1], Sergio Iserte[2]

Resumen— **La maleabilidad de procesos puede definirse como la capacidad de un trabajo paralelo MPI distribuido para modificar el número de procesos sin detener su ejecución, reasignando los recursos computacionales inicialmente asignados al trabajo tantas veces como sea necesario. En general, la maleabilidad se compone de cuatro etapas: reasignación de recursos, gestión de procesos, redistribución de datos y reanudación de la ejecución. Entre ellas, la tercera etapa es la que más tiempo consume y domina el tiempo total de reconfiguración. En este artículo se comparan diferentes implementaciones de esta etapa utilizando operaciones MPI punto a punto y colectivas, incluyendo sus versiones no bloqueantes, tanto para Ethernet 10G como para Infiniband EDR. Estas estrategias de redistribución de datos se combinan con diferentes métodos para expandir/reducir trabajos utilizando una aplicación que solo realiza la segunda y tercera etapa de la maleabilidad, lo que permite evaluar el coste de las diferentes metodologías de modo aislado. Los resultados muestran que la versión punto a punto no bloqueante junto al método de creación de procesos *Merge*, es la alternativa que más reduce el tiempo de redistribución.**

Palabras clave— **MPI, Maleabilidad, Redistribución de datos, Reconfiguración, Paralelismo**

I. Introducción

La maleabilidad es la técnica que permite modificar en tiempo de ejecución los recursos computacionales asignados a una aplicación. La puesta en práctica de esta técnica en un sistema puede ser muy diversa y puede abordar la asignación de diferentes tipos de recursos. En este trabajo, la maleabilidad se refiere a la capacidad de un trabajo paralelo y distribuido de redimensionar su tamaño, en términos de procesos MPI[1], modificando los recursos computacionales que tenía asignados en cualquier punto de la ejecución, y tantas veces como sea necesario. En los grandes sistemas, es habitual encontrar trabajos de este tipo que utilizan la interfaz de paso de mensajes MPI, como herramienta de desarrollo estándar de facto. Estos trabajos se pueden modificar para que en un momento dado reciban una comunicación del gestor de recursos del sistema (RMS) para modificar su asignación de recursos, con el objetivo de maximizar las prestaciones del sistema de acuerdo a algún criterio definido en su política de gestión.

El uso de la maleabilidad en las aplicaciones se justifica por los beneficios que produce, y puede analizarse de dos formas diferentes. Desde el punto de vista de una aplicación, el beneficio se suele conseguir con la mejora de su rendimiento, que habitualmente se alcanza al expandir el trabajo, es decir, al aumentar el número de recursos (procesadores) asignados para la ejecución. Desde el punto de vista del sistema, este beneficio se obtiene al aumentar el rendimiento global, medido en términos de productividad y trabajos finalizados por segundo [2], [3], [4], [5], [6].

La inclusión de la maleabilidad en un trabajo paralelo suele aprovechar la existencia de puntos de control específicos a lo largo de la ejecución, en los que los procesos se sincronizan, para su activación. En aplicaciones iterativas, estos puntos de control suelen aparecer al inicio de cada iteración, mientras que en aplicaciones no iterativas los puntos se definen al inicio de cada fase.

Una fase de maleabilidad conlleva la realización de varias etapas:

1. *Etapa* 1*: Reasignación de recursos.* Preguntar al RMS si la aplicación tiene que ser reconfigurada, bien porque necesita más recursos o bien porque puede liberar alguno. Esta decisión la tomará el RMS en función de los recursos disponibles y de las necesidades que tengan los trabajos que están en ejecución y en espera en ese momento.
2. *Etapa* 2*: Gestión de procesos.* Crear/finalizar procesos MPI en función de la decisión de reconfiguración tomada por el RMS. A partir de este momento, los procesos originales de un trabajo redimensionado serán considerados como los procesos padre, mientras que los procesos en ejecución tras la finalización de la maleabilidad se identificarán como procesos hijo.
3. *Etapa* 3*: Redistribución de datos.* Comunicar datos entre los procesos padre (NP) e hijo (NH), para que la ejecución pueda continuar correctamente en el mismo punto donde se concluyó el redimensionado, pero utilizando los procesos hijo.
4. *Etapa* 4*: Continuación de la ejecución.* Continuar la ejecución de la aplicación con los hijos.

La primera fase es la que inicia la maleabilidad, por lo que las tres últimas etapas sólo se llevan a cabo si el RMS toma la decisión de reconfigurar la aplicación.

La realización de las etapas 2 y 3, suele suponer un importante sobrecoste que puede afectar negativamente a las prestaciones, tanto de la aplicación como del sistema en su conjunto, por lo que una implementación eficiente resulta fundamental. Además, ambas etapas están muy relacionadas por lo que no se pueden implementar y analizar de forma separada. En [7] se propusieron diferentes métodos y estrategias para llevar a cabo la etapa 2, analizando las pres-

[1]Dpto. de Ingeniería y Ciencia de los Computadores, Universitat Jaume I de Castelló, e-mails: martini@uji.es, aliaga@uji.es, castillo@uji.es

[2]Computer Science Department, Barcelona Supercomputing Center, e-mail: sergio.iserte@bsc.es

taciones y el sobrecoste que cada alternativa suponía tanto para la aplicación como para el sistema en su totalidad. El objetivo del presente trabajo es analizar en detalle las operaciones necesarias para completar la redistribución de datos (etapa 3 de maleabilidad) y evaluar que alternativas son más adecuadas en un escenario aislado. Esto nos llevará a seleccionar que método es más adecuado y, en función del tipo de gestión de procesos que se utilice para llevar a cabo la etapa 2, implementar de forma eficiente las etapas 2 y 3 de la maleabilidad que son fundamentales en esta operación.

Las comunicaciones requeridas para completar una distribución de datos se puede realizar utilizando diferentes operaciones MPI: operaciones punto a punto (P2P), que se realizan entre dos procesos concretos, u operaciones colectivas (COL), en las que están involucrados todos los procesos de un comunicador. Además, cada una de estas operaciones tiene su variante bloqueante y no bloqueante. Estas últimas se utilizan para solapar cálculo y comunicaciones.

En el campo de la maleabilidad, éstas se pueden utilizar para solapar la redistribución de datos con la ejecución de la aplicación maleable, dando lugar a dos alternativas para completar la etapa 3: síncrona o asíncrona. En este trabajo se considera que una operación de comunicación es síncrona cuando el usuario tiene que esperar a que termine. En cambio será asíncrona si el usuario puede realizar otras tareas al mismo tiempo. Por tanto las comunicaciones síncronas están basadas en operaciones bloqueantes, mientras que las asíncronas en no bloqueantes.

En la redistribución síncrona, los procesos padre detienen su ejecución mientras se lleva a cabo esta operación. Mientras que en la redistribución asíncrona, los datos se comunican desde los procesos padre a los hijos como una tarea en segundo plano, sin detener la ejecución de los padres.

También es importante comentar que el rendimiento de las diferentes operaciones de comunicación está muy determinada por la red que se utilice, que influye directamente sobre las diferentes alternativas definidas para completar la redistribución. En este trabajo se compara el funcionamiento de las diferentes alternativas en una red Ethernet 10G y en una red Infiniband EDR, con el fin de mostrar cual de ellas realiza la redistribución en el menor tiempo posible de forma aislada.

El resto del artículo se organiza como sigue. La Sección II describe las principales operaciones MPI requeridas para llevar a cabo una redistribución de datos y explica brevemente diferentes alternativas que se han implementado para realizar la etapa 2 de la maleabilidad. La Sección III describe los métodos utilizados para completar la etapa 3. En la Sección IV se muestran los resultados experimentales obtenidos de aplicar conjuntamente las etapas 2 y 3 de maleabilidad sobre las dos redes de comunicación comentadas sobren un clúster con 8 nodos. Finalmente, la Sección V presenta las conclusiones de este estudio.

II. Operaciones MPI para maleabilidad

La tecnología más utilizada para la implementación de aplicaciones maleables es la Interfaz de Paso de Mensajes (MPI) [1]. Esta define una especificación para el paso de mensajes entre procesos implementada a través de una librería para diversos lenguajes y constituye una tecnología ampliamente utilizada para la paralelización de códigos.

MPI cuenta con una serie de operaciones con las que implementar cualquier comunicación entre procesos, que se pueden dividir en dos tipos: operaciones *punto a punto* y operaciones *colectivas*. Las primeras definen comunicaciones entre dos procesos concretos, mientras que las segundas involucran a todos los procesos de un comunicador. Todo comunicador está definido por un grupo de procesos, en los que cada uno de sus miembros tiene un identificador (*rank*). Si una comunicación se realiza entre procesos de un grupo, el comunicador utilizado se caracteriza como *intra-comunicador*. En cambio, cuando se requiere comunicar entre procesos de dos grupos distintos, es necesario crear un *inter-comunicador*, en los que aparecen procesos de ambos grupos.

A continuación se introducen las operaciones MPI que serán utilizadas para completar las etapas 2 y 3 de la maleabilidad. En primer lugar se introducen las alternativas descritas en [7] para modificar el número de procesos de una aplicación (etapa 2), y posteriormente se describen las operaciones MPI utilizadas para realizar la redistribución de datos (etapa 3).

A. Modificación del número de procesos

En [7], se presentaron diferentes métodos y estrategias con el objetivo de analizar y comparar el rendimiento para expandir o reducir el número de procesos de una aplicación en ejecución, sin desviarse del estándar MPI [1]. A continuación, se describen, de forma muy resumida, los más representativos para que la descripción posterior de los métodos de redistribución de datos que se proponen sean fácilmente comprensibles. Como ya se ha comentado, los métodos aplicados en las etapas 2 y 3 de maleabilidad no pueden analizarse de forma totalmente independiente, ya que en la etapa 2 se generan un número de procesos y un tipo de comunicador que serán utilizados en la etapa 3.

La creación de procesos en MPI se basa en la función `MPI_Comm_spawn` que es una operación colectiva sobre un determinado comunicador. En el Listado 1 se muestra la definición de este rutina, tal y como aparece en el estándar [1].

Listado 1: Definición de la función de MPI para la creación de procesos dinámicos.

```
1 int MPI_Comm_spawn(const char *command,
2   char *argv[], int maxprocs, MPI_Info info,
3   int root, MPI_Comm comm, MPI_Comm *intercomm,
4   int array_of_errcodes[])
```

La alternativa más sencilla para completar la Etapa 2 de la maleabilidad es el uso de la rutina `MPI_Comm_spawn` para generar los NH procesos necesarios para continuar la ejecución. Este método se ha

denominado *Baseline* en [7] y aparece gráficamente en la Figura 1a. En este ejemplo, inicialmente había $NP = 2$ procesos (padres) ejecutando la aplicación, tras completar la llamada a esta función MPI, se crean $NH = 4$ procesos nuevos (hijos). Tras esta llamada, los hijos deben continuar la ejecución de la aplicación en el mismo punto donde la dejaron los padres, y estos deben finalizar su ejecución. El uso del método *Baseline* involucra 3 comunicadores diferentes:

- Intra-comunicador `MPI_COMM_WORLD` definido por todos los padres.
- Intra-comunicador `MPI_COMM_WORLD` definido por todos los hijos.
- Inter-comunicador que conecta los padres con los hijos.

De estos, únicamente el último es necesario para completar la Etapa 3 de la maleabilidad.

El principal problema del método *Baseline* es la suscripción en exceso (*oversubscription*), es decir, que un procesador deba manejar un número de procesos mayor que el número de núcleos que incluya, lo que puede afectar a las prestaciones del sistema. Una posible alternativa es reutilizar los procesos padres existentes como hijos y generar únicamente el resto de procesos necesarios para completar la reconfiguración hasta NH hijos. Este método, denominado *Merge* en [7] se puede ver gráficamente en las Figuras 1d y 1e y las funciones colectivas adicionales en el Listado 2. En el primer caso, reconfiguración desde $NP = 2$ a $NH = 4$, se crean únicamente 2 nuevos procesos, siendo necesario utilizar la rutina `MPI_Intercomm_merge` para generar el intra-comunicador que agrupa a todos los procesos que continuarán la ejecución [8]. Se trata de la primera función del Listado 2, una operación colectiva en la que participan todos los padres e hijos, donde el parámetro `high` fija como se numerarán los procesos en la agrupación final.

Por su parte, en la Figura 1e (reconfiguración desde $NP = 4$ a $NH = 2$), se muestra que cuando $(NP > NH)$ no hace falta crear nuevos procesos, sino que la alternativa es detener la ejecución de $(NP - NH)$ procesos padre. Para ello, se divide el intra-comunicador de los padres en dos comunicadores, utilizando la segunda función del Listado 2, `MPI_Comm_split`. El parámetro `color` se utiliza para obtener varios comunicadores, aunque en este caso solo hace falta dividir el comunicador original en dos partes. Posteriormente, se debe suspender la ejecución de los procesos que no son hijos y continuar la ejecución de la aplicación.

Listado 2: Definición de la funciones de MPI adicionales para la creación de procesos dinámicos con el método *Merge*.

```
1 int MPI_Intercomm_merge(MPI_Comm intercomm,
2   int high, MPI_Comm *newintracomm);
3
4 int MPI_Comm_split(MPI_Comm comm, int color,
5   int key, MPI_Comm *newcomm);
```

Las Figuras 1a-1d-1e muestran un modo de funcionamiento síncrono, en el que los padres detienen la ejecución de la aplicación antes de iniciar la reconfiguración, y la ejecución continua en los hijos al completarse esta. Pero existe un modo de funcionamiento alternativo, denominado asíncrono, en el que la ejecución de los padres continúa mientras la reconfiguración se completa. Las Figuras 1b-1f-1c muestran estas versiones muy similares a las anteriores y donde se aprecia esta particularidad. En estas alternativas, puede ocurrir que los hijos, una vez generados, tengan que esperar hasta que los padres comprueben que ha finalizado la tarea de reconfiguración, así estos pueden acabar la ejecución y los hijos puedan continuarla. En las Figuras 1b-1f-1c, estas esperas están representadas por bloques rayados, mientras que los padres realizan la comprobación en los puntos de control. En estas figuras, también se observa que el modo asíncrono se implementa creando una hebra auxiliar por parte de cada padre, que será la encargada de realizar la creación de los procesos hijos.

B. Comunicaciones MPI para la maleabilidad

En la etapa de redistribución de datos es posible utilizar comunicaciones *P2P* o *COL*. Las primeras se basan en el uso de las operaciones `MPI_Send` y `MPI_Recv` (ver Listado 3), que se pueden utilizar para comunicar cualquier información entre procesos. En este caso, su uso será enviar información de padres a hijos.

Por su parte, las principales operaciones de comunicación colectiva utilizadas para la redistribución de los datos en la maleabilidad son `MPI_Bcast` y `MPI_Alltoallv` (ver Listado 4). Con la operación `MPI_Bcast`, un proceso envía una información al resto de procesos. En la redistribución de datos, esta operación se puede utilizar para enviar el tamaño de los datos o los datos replicados.

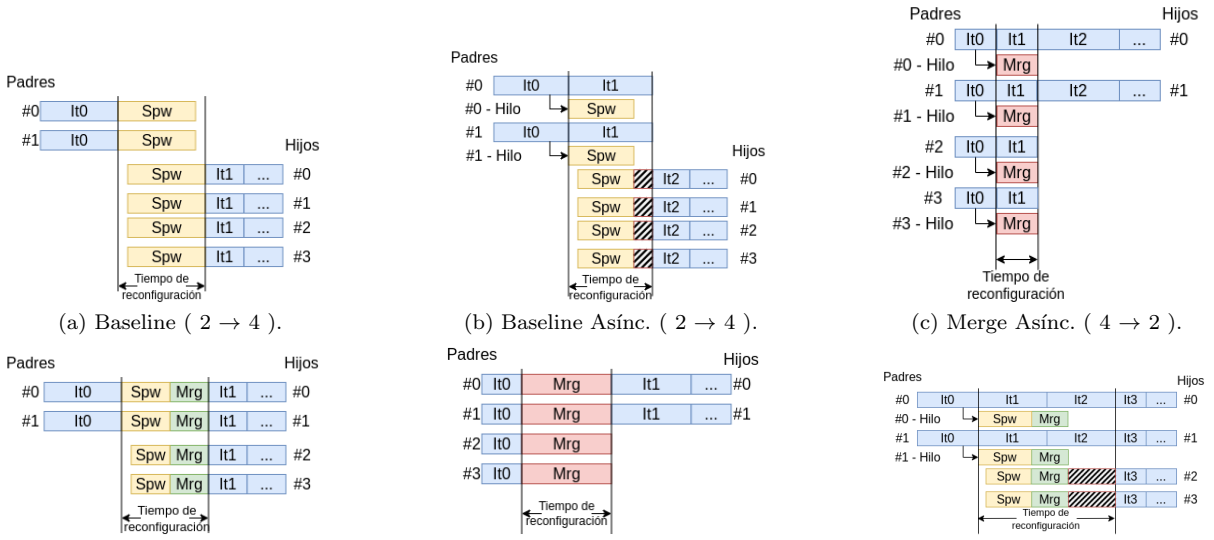
En cambio, la operación `MPI_Alltoallv` es la de mayor coste para la redistribución de datos. Esta función permite que cada proceso envíe/reciba una cantidad de información diferente. Es la operación más utilizada en la etapa 3 de maleabilidad.

Listado 3: Definición de la funciones P2P de MPI para la redistribución de datos. Versiones bloqueantes y no bloqueantes.

```
1 int MPI_Send(const void *buf, int count,
    MPI_Datatype datatype, int dest, int tag,
    MPI_Comm comm);
2 int MPI_Recv(void *buf, int count, MPI_Datatype
    datatype, int source, int tag, MPI_Comm
    comm, MPI_Status *status);
3 int MPI_Isend(const void *buf, int count,
    MPI_Datatype datatype, int dest, int tag,
    MPI_Comm comm, MPI_Request *request);
4 int MPI_Irecv(void *buf, int count, MPI_Datatype
     datatype, int source, int tag, MPI_Comm
    comm, MPI_Request *request)
```

Listado 4: Definición de la funciones COL de MPI para la redistribución de datos. Versiones bloqueantes y no bloqueantes.

```
1 int MPI_Bcast(void *buffer, int count,
    MPI_Datatype datatype, int root, MPI_Comm
    comm);
2 int MPI_Alltoallv(const void *sendbuf, const int
     sendcounts[], const int sdispls[],
    MPI_Datatype sendtype, void *recvbuf, const
     int recvcounts[], const int rdispls[],
    MPI_Datatype recvtype, MPI_Comm comm);
3 int MPI_Ibcast(void *buffer, int count,
    MPI_Datatype datatype, int root, MPI_Comm
    comm, MPI_Request *request);
```



(a) Baseline (2 → 4). (b) Baseline Asínc. (2 → 4). (c) Merge Asínc. (4 → 2).

(d) Merge (2 → 4). (e) Merge (4 → 2). (f) Merge Asínc. (2 → 4).

Fig. 1: Esquema métodos de reconfiguración. Los bloques horizontales son procesos, donde *ItX* es la iteración en ejecución, *Spw* la creación de procesos y *Mrg* fusión de comunicadores. Los bloques rayados indica el tiempo de espera.

```
4 int MPI_Ialltoallv(const void *sendbuf, const
      int sendcounts[], const int sdispls[],
      MPI_Datatype sendtype, void *recvbuf, const
       int recvcounts[], const int rdispls[],
      MPI_Datatype recvtype, MPI_Comm comm,
      MPI_Request *request);
```

El comportamiento de las comunicaciones *P2P* y *COL* varia según el tipo de comunicador que se usa. A continuación se describen dichas diferencias.

B.1 Comunicaciones con intra-comunicadores

Un intra-comunicador se define a partir de un grupo de procesos con un contexto y un nombre de comunicador, como es el caso de *MPI_COMM_WORLD*. Todo proceso del grupo tiene un identificador en el grupo, que se utiliza para definir el origen y el destino de una operación *P2P*.

Existen dos tipos de operaciones *COL*, las que definen un proceso raíz (es decir origen/destino) de la comunicación, y las que consideran que todos los procesos trabajan de igual modo y se especifica que se envía y recibe de cada proceso del comunicador. Entre las primeras destacamos la operación `MPI_Bcast`, donde todos deben indicar al mismo proceso raíz, mientras que `MPI_Alltoallv` es del segundo tipo.

B.2 Comunicaciones con inter-comunicadores

Los inter-comunicadores se componen de dos grupos de procesos con sus contextos y un nombre de comunicador. Las comunicaciones siempre deben realizarse desde uno o más procesos de un grupo hacia uno o más procesos del otro grupo. Es por ello que el origen/destino de una comunicación *P2P* debe ser un identificador de proceso válido en el otro grupo.

Para las operaciones *COL* que definen un proceso raíz, el proceso que envía/recibe la información debe utilizar la constante `MPI_ROOT`, mientras que el resto de procesos de su grupo deben usar `MPI_PROC_NULL`. Por su parte, todos los procesos del otro grupo deben utilizar el identificador del proceso raíz en su grupo.

Para el resto de operaciones *COL*, si los parámetros de la operación definen un patrón de comunicación en el que hay comunicación efectiva en los dos sentidos, esta se debe completar en dos fases. Una primera enviando información del grupo A al B, y a continuación comunicando en sentido inverso.

C. Comunicaciones bloqueantes/no bloqueantes

El estándar define dos tipos de operaciones MPI, *bloqueantes* y *no bloqueantes*, cuyas características pueden ser de interés en la redistribución de datos. A continuación se introducen ambos tipos:

- *Bloqueante*: Aquellas que requieren de una respuesta para poder continuar la ejecución. En el caso de `MPI_Recv`, la respuesta es la recepción del mensaje, pero para `MPI_Send` depende del modo de ejecución. En el modo *synchronous*, la operación finaliza cuando se completa la transferencia, mientras que en el modo *buffered* finaliza tras hacer una copia de la información a enviar. El funcionamiento estándar de `MPI_Send` intenta explotar el modo *buffered*, utilizando los tampones internos de la librería MPI, y si no fuese posible, utiliza el modo *synchronous*.
- *No bloqueante*: Aquellas que no bloquean el flujo de ejecución del programa ya que retornan inmediatamente después de su llamada, sin tener en cuenta si la operación de comunicación se ha completado, por lo que permiten que el proceso realice otras tareas mientras esperan su finalización. El nombre de estas operaciones siguen el formato `MPI_Ixxx`, donde *xxx* representa el nombre de la operación correspondiente.

En los Listados 3 y 4 también se muestran las comunicaciones *no bloqueantes* para *P2P* y *COL*, respectivamente. En estas comunicaciones aparece un argumento adicional respecto a su versión bloqueante, que se asocia con un manejador `MPI_Request` utilizado para consultar el estado de la comunicación. Este manejador se puede usar en la operación `MPI_Wait` para esperar a que finalice la comunicación, o en la operación `MPI_Test`, devolviendo un va-

Tabla I: Algoritmos de MPICH para la función `MPI_Alltoallv`.

Tipo comunicador	Tipo función	
	Bloqueante	**No bloqueante**
Intra-comunicador	Scattered	Schedule Blocked
Inter-comunicador	PairWise Exchange	Schedule PairWise Exchange

lor lógico que indica si la operación se ha completado. Ambas operaciones incluyen una variante con el sufijo *All*, que realizan la espera/validación de varias comunicaciones no bloqueantes.

D. Algoritmos de comunicación colectiva para Alltoallv en MPICH

Buena parte de los datos en las aplicaciones paralelas se encuentran distribuidos entre todos los procesos, por lo que para su redistribución se requerirá el uso de la operación `MPI_Alltoallv`. A continuación se realiza un análisis más detallado de como se lleva a cabo esta operación en el estándar de MPICH [9].

La Tabla I clasifica los diferentes algoritmos utilizados para implementar `MPI_Alltoallv`, según el tipo de comunicador utilizado y si la operación se usa en modo *bloqueante* o *no bloqueante*. A continuación se describen sus principales características:

- *Scattered*: Algoritmo bloqueante para intra-comunicadores basado en `Irecv` + `Isend`. El total de comunicaciones de un proceso se divide en conjuntos. Para cada conjunto se hacen hasta 32[1] comunicaciones y una llamada a `MPI_Waitall`. Los procesos solo realizan comunicaciones con aquellos a los que tienen que comunicar datos. En el Algoritmo 1 se muestra una simplificación del mismo.
- *Schedule Blocked*: Variante no bloqueante del algoritmo *Scattered* para intra-comunicadores. El algoritmo es el mismo pero no se utiliza `MPI_Waitall` para que el proceso pueda continuar con la realización de otras tareas. Por tanto, la llamada termina para un proceso cuando crea sus solicitudes de comunicación.
- *PairWise Exchange*: Algoritmo bloqueante para inter-comunicadores basado en `MPI_Sendrecv`. Cada proceso del grupo A realiza tantas comunicaciones como procesos haya en el grupo B y viceversa (ver Algoritmo 2). El algoritmo no comprueba si hay datos que enviar a ese proceso antes de realizar la llamada a `MPI_Sendrecv`.
- *Schedule PairWise Exchange*: Versión no bloqueante del algoritmo *Schedule Blocked* pero para inter-comunicadores. La única diferencia es que no se hace por conjuntos ni tampoco comprueba si hay bytes a comunicar antes de realizar una solicitud de comunicación.

III. Redistribución de datos

En esta sección, se describen los diferentes algoritmos y estrategias utilizados para redistribuir la información desde los procesos padre hacia los hijos para que estos continúen con la ejecución (Etapa 3 de la maleabilidad).

[1]Por defecto el valor por conjunto en el algoritmo *Scattered* es de 32, pero es posible modificarlo.

Algoritmo 1 Simplificación del algoritmo *Scattered*.

```
// Validaciones previas
b = 32 // Valor por defecto del bloque
c_size = MPI_Comm_size
for ( ii = 0; ii < c_size; ii+=b ) do
    ss = c_size − ii < b ? c_size − ii : b
    for ( i = 0; i < ss; i++ ) do
        dst = (myId + i + ii) %c_size;
        MPI_Irecv: dst → myId
        dst = (myId − i − ii + c_size) %c_size;
        MPI_Isend: myId → dst
    end for
    MPI_Waitall // Solo en algoritmo bloqueante
end for
```

Algoritmo 2 Simplificación del algoritmo *PairWise Exchange*.

```
// Validaciones previas
l_size = MPI_Comm_size
r_size = MPI_Comm_remote_size
m_size = max: r_size, l_size
for ( i = 0; i < m_size; i++ ) do
    dst = (myId + i) %m_size
    src = (myId − i + m_size) %m_size
    if ( dst ≥ r_size ) then dst = MPI_PROC_NULL
    end if
    if ( src ≥ r_size ) then src = MPI_PROC_NULL
    end if
    MPI_Sendrecv: myId → dst, src → myId
end for
```

Esta sección esta estructurada en dos partes. En primer lugar, se analizan algunas cuestiones básicas relacionadas con la redistribución de datos. A continuación, se presentan los distintos métodos implementados para completar esta etapa. El conjunto de métodos se basan en el tipo de comunicación, síncrona o asíncrona que a su vez utilizarán operaciones del tipo *P2P* o *COL*.

A. Conceptos previos

La redistribución de los datos se realiza junto con el redimensionado del número de procesos. Primero se generan/liberan los procesos y después se realiza la redistribución. Es por ello que en las imágenes que aparecen en la Figura 1, la redistribución debería aparecer a continuación de la Etapa 2 y siendo una parte del tiempo de reconfiguración de la aplicación.

En la redistribución de datos hay varios aspectos a considerar. Uno de ellos es la estructura interna de la información a comunicar, que suelen ser vectores, matrices densas y matrices dispersas. Otro es el modo en el que esta información aparece distribuida entre los procesos. El uso de un reparto estático por bloques facilita esta labor en los dos primeros casos, pero en nada ayuda en el último caso, ya que el número de elementos no nulos que se debe comunicar entre dos procesos puede variar mucho. Esta situación es todavía más acusada en repartos dinámicos y/o con estructuras de datos más complejas.

Es por ello, que generalizar el patrón de comunicación entre procesos no es una tarea fácil. En muchos casos, la comunicación se divide en dos fases: en la primera, cada padre envía a los hijos el tamaño de la información a comunicar, para que estos puedan crear las estructuras necesarias, mientras que en la segunda fase, se realiza la comunicación de la infor-

Algoritmo 3 Estructura básica de redistribución de datos.

```
// Padres envían tamaños a hijos
if ( myId ≥ frstChd && myId ≤ lstChld ) then
    Los hijos crean estructuras internas
end if
// Padres envían información a hijos
```

Algoritmo 4 Inicialización vectores de comunicación.

```
r_size = Malleability_Other_group
counts = calloc: → r_size
displs = calloc: → r_size
for ( i = 0; i < r_rsize; i++ ) do
    r_ini, r_end = Block_id: → i
    if (ini ≥ r_end||end ≤ r_ini) then continue
    end if
    big_ini = ini > r_ini?ini : r_ini
    small_end = end < r_end?end : r_end
    counts[i] = small_end − big_ini
    displs[i + 1] = displs[i] + counts[i]
end for
```

mación completa. El Algoritmo 3 muestra esquemáticamente la estructura básica de la redistribución.

B. Cálculos relacionados con la redistribución

Como tarea previa a la comunicación de una estructura distribuida, cada padre/hijo debe calcular dos vectores en el que se indique cuantos elementos va a enviar/recibir a/de cada padre/hijo. Aquellos procesos que tengan ambos roles deberán realizar este calculo dos veces, uno por cada rol.

Los vectores a calcular son *cuentas* (*counts*) y *desplazamientos* (*displs*). El tamaño de los vectores para los padres es igual al número de hijos, y viceversa para los hijos. El primer vector indica el número de elementos a comunicar a/desde cada proceso destino/origen. Por su parte, el segundo vector marca la posición del primer elemento a comunicar al citado proceso.

En el Algoritmo 4 se muestra como calcular ambos vectores para un proceso padre/hijo. Asumiendo una distribución por bloques, los elementos en un proceso son los comprendidos en el intervalo $[ini, end]$. Para calcular los elementos a comunicar, cada proceso padre/hijo debe calcular la intersección con la de los correspondientes hijos/padres, que indican el número de elementos a comunicar a/de cada proceso. La función `Malleability_Other_group` obtiene el número de procesos padres/hijos, mientras que la función `Block_id` obtiene los valores *ini* y *end* de un proceso remoto, padre/hijo.

C. Métodos síncronos

Estos métodos se pueden implementar utilizando operaciones *P2P* o *COL*.

Los métodos síncronos basados en *P2P* utilizan las funciones `MPI_Send` y `MPI_Recv`. Aunque ambas sean operaciones bloqueantes, su uso junto al método *Baseline* imposibilita la aparición de cualquier tipo de interbloqueo, ya que la intersección entre padres e hijos es vacía. Pero el uso de estas funciones junto al método *Merge* puede ser más problemático, ya que la intersección no es vacía; aún así la utilización del modo *buffered* de `MPI_Send` resolvería muchos de los posibles interbloqueos. En cualquier caso, la versión más segura combinaría el uso de la

Algoritmo 5 Redistribución utilizando operaciones *P2P*.

```
if (myId ≥ frsPrnt && myId ≤ lstPrnt) then
    // Padres envían información
    for ( i = frstChd; i ≤ lstChld; i++ ) do
        if ( i == myId ) then
            Copia local usando memcpy
        else
            MPI_Isend: myId → i, tag = 77
        end if
    end for
end if
if ( myId ≥ frstChd && myId ≤ lstChld ) then
    // Hijos reciben información
    for ( i = frsPrnt; i ≤ lstPrnt; i++ ) do
        if ( i ≠ myId ) then
            MPI_Recv: MPI_ANY_SOURCE , tag = 77
        end if
    end for
end if
if (myId ≥ frsPrnt && myId ≤ lstPrnt) then
    // Padres esperan la finalización
    MPI_Waitall: tag = 77
end if
```

Algoritmo 6 Redistribución utilizando operaciones *COL*.

```
// Procesos envían/reciben información
MPI_Alltoallv: parents → children
```

función `MPI_Isend` con `MPI_Waitall`/`MPI_Testall`. El Algoritmo 5 muestra como podría implementarse esta comunicación, donde los identificadores de los procesos padre e hijo se definen, respectivamente, en dos intervalos $[frsPrnt, lstPrnt]$ y $[frsChld, lstChld]$, respectivamente. La utilización adecuada de `MPI_ANY_SOURCE` acelera la recepción de los mensajes por parte de los hijos. Además, el uso de valores diferentes de etiqueta (`tag`) permitiría realizar consecutivamente varias comunicaciones con un único control de finalización al final del algoritmo.

Los métodos síncronos basados en operaciones *COL*, por su propio diseño, no provocan interbloqueos. Este se basa en la utilización de las funciones `MPI_Alltoall` o `MPI_Alltoallv` en su modo *bloqueante*. Esta alternativa simplifica el código necesario para la redistribución de datos desde los procesos padre a los hijos, puesto que no hay que comprobar la intersección entre padres e hijos y no hay que hacer copias locales. El Algoritmo 6 muestra esta implementación.

D. Métodos asíncronos

Para este tipo de comunicación, una opción es utilizar *hebras auxiliares* que son las responsables de llevar a cabo la redistribución de datos, liberando a la hebra principal de esta tarea. Estas hebras utilizarán alguna de las opciones descritas con anterioridad en el Algoritmos 5 o 6. Otra alternativa es el uso de las versiones *no bloqueantes* de las operaciones de comunicación. En el caso del Algoritmo 5, la solución es separar la fase de comunicación de la fase de finalización, que se debería realizar en los puntos de sincronización elegidos por la aplicación. Además habría que sustituir `MPI_Recv` por `MPI_Irecv` y `MPI_Waitall` por `MPI_Testall`. Hay que tener en cuenta, que el primer cambio es obligatorio cuando los procesos han sido generados con el método *Merge*, ya que un padre puede participar como hijo al mismo tiempo.

Por su parte, en el Algoritmo 6 se debería utilizar `MPI_Ialltoallv` para iniciar la comunicación, y `MPI_Testall` en cada punto de sincronización para comprobar la finalización de las mismas.

IV. Resultados experimentales

En esta sección se presentan los experimentos y análisis realizados para comparar los métodos descritos en la Sección III. Las conclusiones de este estudio serán fundamentales para implementar una versión optimizada de las etapas 2 y 3 de la reconfiguración de una aplicación maleable.

A. Hardware y Software utilizados

Los experimentos se han realizado en un clúster de ocho nodos con dos procesadores Intel Xeon 4210 de 10 núcleos cada uno, por tanto con un total de 160 núcleos. Los nodos están interconectados con una red Infiniband EDR de 100GB/s y con una Ethernet de 10GB/s, utilizándose una versión diferente de MPI para trabajar con cada red. Por un lado, se utiliza la versión MPICH 4.0.3 [9] compilada con CH4:OFI netmod (Infiniband) y, por otro, la versión MPICH 3.4.1 se ha compilado con CH3:Nemesis netmod (Ethernet).

En el estudio se han realizado 10 ejecuciones de una aplicación que solo realiza dos tareas: creación de procesos y redistribución de datos. En la primera tarea se utilizan los métodos descritos en el apartado II-A, mientras que en la segunda tarea se redistribuyen 5Gb de información desde NP padres a NH hijos, utilizando las diferentes estrategias de redistribución de datos descritas en la Sección III. El estudio consta de una reconfiguración por experimento, que se inician con 2, 20, 40, 80, 120 y 160 procesos padre y se reconfiguran a los mismos números de procesos hijo, dando lugar a 30 (NP, NH) pares. El número de nodos del clúster ocupados en cada ejecución se calcula como $\lceil N/20 \rceil$, donde N es el máximo entre NP y NH, lo que permite minimizar los recursos asignados por el RMS.

Para completar los experimentos, se han utilizado las operaciones *P2P* y *COL* en sus variantes *síncrona* (comunicación bloqueante) y *asíncrona* (comunicación no bloqueante). En este último caso, aunque la comunicación sea del tipo asíncrono, no se solapa con ningún otro cómputo, porque el objetivo es analizar el comportamiento de la comunicación entre padres e hijos, y descubrir qué alternativa es la más eficiente cuando únicamente se realiza la redistribución. Por su parte, el método de creación de procesos determina el tipo de comunicador a utilizar. Por defecto, *Baseline* utiliza inter-comunicadores y *Merge* maneja intra-comunicadores, aunque también se ha implementado una variante de *Baseline* que utiliza intra-comunicadores, *BaselineIntra*. Finalmente, el análisis se ha realizado para los dos tipos de redes que dispone el sistema, Ethernet e Infiniband.

B. Tiempos de redistribución

En este apartado se realiza un estudio sobre el tiempo de redistribución de $5Gb$ de datos en las dos redes objeto de análisis, Infiniband y Ethernet, considerando diferentes alternativas de comunicación y operación. Estas alternativas se han obtenido al combinar las diferentes estrategias implementadas para llevar a cabo las etapas 2 y 3 de la maleabilidad con distintos valores de pares (NP, NH).

Las Figuras 2 y 3 muestran el tiempo de redistribución sobre Ethernet e Infiniband, respectivamente. En ambas figuras, se han separado los pares asociados al aumento del número de procesos (expansión), que aparecen en la gráfica superior, y los pares asociados a la reducción del número de procesos (reducción), que aparecen en la gráfica inferior. Además, el eje X muestra los diferentes pares (NP, NH) considerados, mientras que el eje Y indica el coste de la comunicación en segundos. Por su parte, las leyendas indican el método de reconfiguración utilizado, (*Baseline*, *BaselineIntra* y *Merge*), el tipo de operación (*P2P* o *COL*) y si se ha utilizado una variante *Síncrona* o *Asíncrona* (marcado por el sufijo S o A, respectivamente).

Al analizar las figuras asociadas a la **expansión**, se observa que todas las variantes tienen un comportamiento muy similar independientemente del número de procesos, excepto en la variante *Baseline-COLS*, cuya diferencia de tiempos es grande y oscila entre $1s$ y $8,5s$. Este comportamiento se justifica por la implementación de la operación `MPI_Alltoallv` sobre inter-comunicadores en el compilador MPICH, que utiliza el algoritmo `PairWise Exchange`, basado en comunicación de tipo S, en el que se requiere que cada proceso termine una comunicación para realizar la siguiente. Por su parte, el resto de algoritmos *P2P* o *COL* están basados, parcial o completamente, en comunicaciones *no bloqueantes* que pueden solapar comunicaciones.

También resulta destacable una ligera mejora entre $0,2s$ y $0,3s$ en las variantes *Merge* sobre las *Baseline* en la red Infiniband, independientemente del tipo de comunicador, debido, principalmente, a la aparición de la suscripción en exceso en estos últimos. En los experimentos sobre la red Ethernet solo se observa este efecto para valores grandes de NP (80 ó 120).

El comportamiento de la **reducción** cuando $NH \geq 20$ es similar al descrito en la expansión, observándose que la variante *Baseline-COLS* necesita un tiempo mayor que el resto de variantes, con una diferencia que oscila desde $3s$ a $8s$, justificado por el uso del algoritmo *PairWise Exchange* en la implementación de la operación. Además, las variantes *Merge* en ambas redes, obtienen tiempos ligeramente mejores por una diferencia de $0,1s$ y desde $0,1s$ a $0,4s$ en Ethernet e Infiniband, respectivamente, debido a la aparición de la suscripción en exceso en las variantes *Baseline*.

Sin embargo, el comportamiento cuando $NH = 2$ es irregular en todas las variantes, siendo las más rápidas en la red Ethernet *Merge-COLA* y *Merge-*

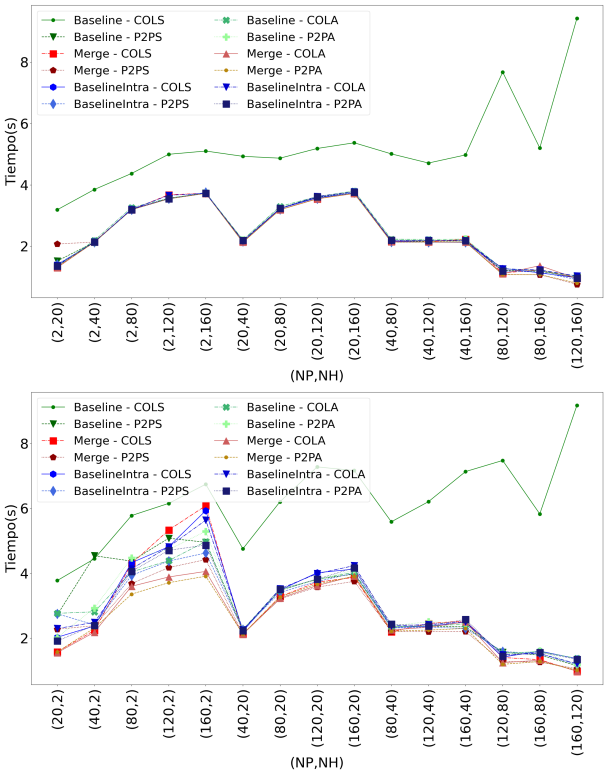


Fig. 2: Tiempo de redistribución de 5Gb de datos en Ethernet desde NP a NH. Expansión (arriba), Reducción (abajo).

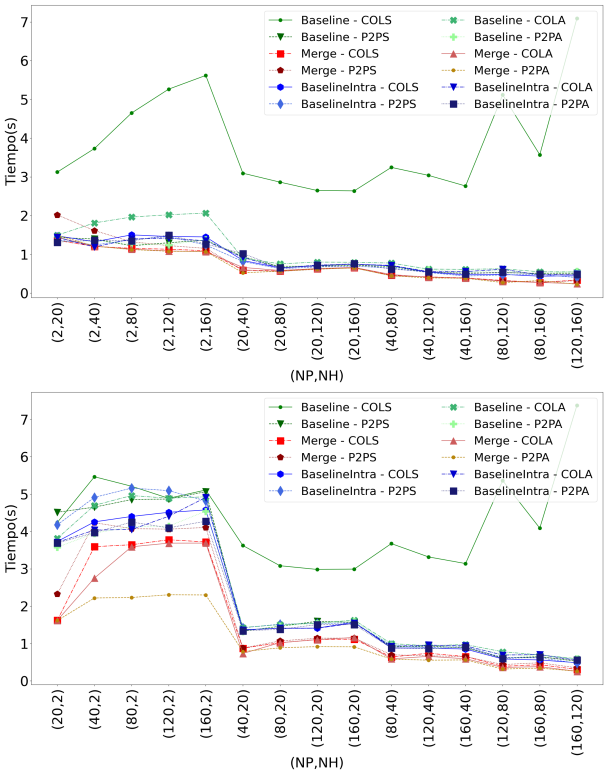


Fig. 3: Tiempo de redistribución de 5Gb de datos en Infiniband desde NP a NH. Expansión (arriba), Reducción (abajo).

P2PA, con una diferencia máxima de $0,4s$ respecto a la variante *Merge-P2PS*. En cambio, la variante más rápida en la red Infiniband es *Merge-P2PA*, con una diferencia máxima de $1,4s$ respecto a la variante *Merge-COLA*. Este comportamiento se puede justificar por la congestión de mensajes en los hijos, que deben recibir mensajes desde muchos padres.

La comparación de las variantes *Baseline* y *BaselineIntra* permite analizar el impacto del uso de **intra-** e **inter-comunicadores**. En este análisis, solo se detectaron diferencias significativas en la variante *Baseline-COLS*, ya que es la única que utiliza el algoritmo *PairWise Exchange*, basado en comunicaciones bloqueantes, que no genera solapamiento de comunicaciones. El resto de variantes utilizan algoritmos basados en comunicaciones asíncronas: *Scattered* y *Schedule Blocked* para intra-comunicadores, mientras que *Schedule PairWise Exchange* utiliza la variante *no bloqueante* sobre inter-comunicadores.

C. Sobrecoste de variantes asíncronas

En este apartado se realiza un estudio del sobrecoste que tienen las variantes asíncronas (A) respecto a las variantes síncronas (S). Este sobrecoste, que se refleja en el valor de α, se ha calculado como el cociente de cada una de las variantes A con su respectiva variante S. Así, valores superiores a 1 indican que la variante A es más costosa que la variante S asociada, y valores menores que 1 es justo lo contrario.

Las Figuras 4 y 5 muestran los valores de α con la red Ethernet e Infiniband, respectivamente. La parte superior de ambas figuras es para el caso de expansión de procesos, y la parte inferior para reducción. La interpretación de las leyendas y los valores del eje X también son idénticas al descrito en el estudio anterior. En este caso, únicamente varía el significado del eje Y que indica el valor de α.

En el caso de la **expansión** para la red Ethernet, los valores de α siempre tienen una diferencia menor a $\pm 5\,\%$ respecto al valor 1. Solo en la variante *Baseline-COLA* existe una reducción del tiempo de redistribución entre el $30\,\%$ y $70\,\%$ debido al mayor coste de la variante *Baseline-COLS*.

Se observa un comportamiento más irregular para la red Infiniband, donde la diferencia llega hasta $\pm 10\,\%$ respecto al valor 1, siendo en 4 de 75 casos mayor a $\pm 10\,\%$, exceptuando la variante *Baseline-COLA*. Pero a partir de 20 procesos padre los valores se estabilizan alrededor de 1.

El análisis de la **reducción** muestra dos situaciones diferentes. La primera se produce cuando ($NH = 2$), donde aparece un comportamiento irregular de todas las variantes para ambas redes, debido a la congestión de mensajes ya mencionada con anterioridad.

Por su parte, la segunda situación se observa cuando ($NH > 2$) y es dependiente de la red. En la red Ethernet, los valores de α siempre tienen una diferencia menor a $\pm 10\,\%$ respecto al valor 1. En cambio, en 11 de las 20 combinaciones posibles de las variantes *Merge* asíncronas sobre la red Infiniband, el valor de α tiene una diferencia mayor a $\pm 10\,\%$ respecto a 1. Además, 9 de estos 11 casos son de la variante *Merge-P2PA*, que tiene un menor coste, ya que utiliza comunicaciones no bloqueantes.

Finalmente, comentar que la variante *Baseline-COL* siempre obtiene valores de α menores a 1, independientemente de la red. Por tanto, resulta aconsejable utilizar *Baseline-COLA* antes que *Baseline-*

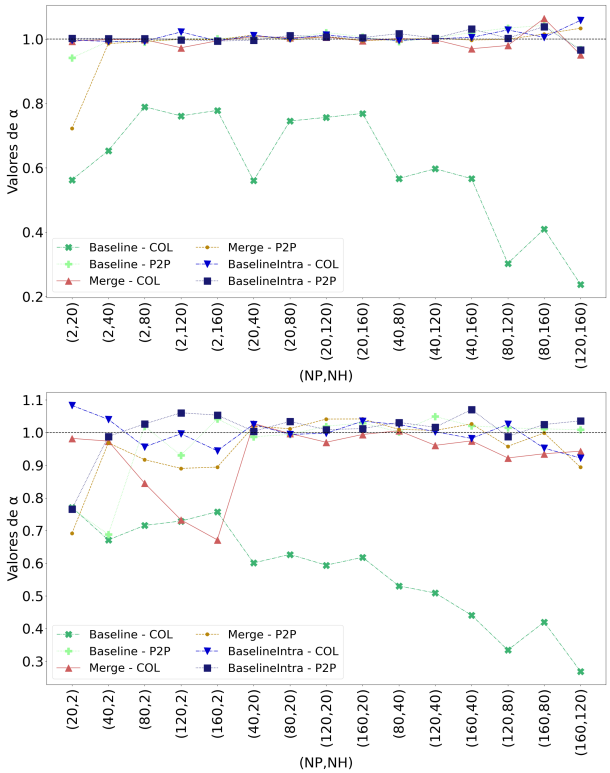


Fig. 4: Valores de α para la redistribución de $5Gb$ de NP a NH en Ethernet. Expansión (arriba), Reducción (abajo).

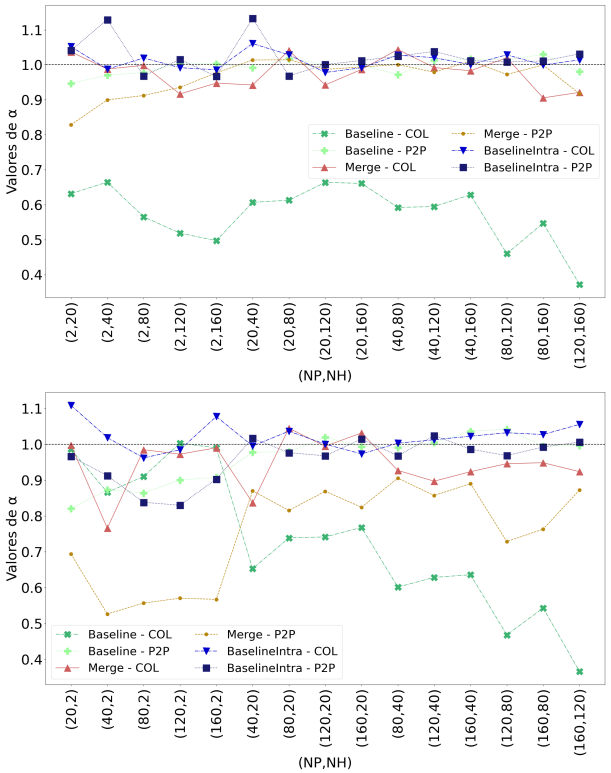


Fig. 5: Valores de α para la redistribución de $5Gb$ de NP a NH en Infiniband. Expansión (arriba), Reducción (abajo).

COLS, aún cuando se produzca en el primero cierto retraso en el inicio de la ejecución de los hijos, puesto que los padres tienen que alcanzar el punto de control y comprobar el final de la comunicación.

D. Análisis de las variantes de redistribución

En este apartado se ha realizado un estudio estadístico para determinar que variante es la más rápida para realizar una redistribución de datos desde NP padres a NH hijos.

Para este estudio se han utilizado las pruebas de Shapiro-Wilk [10], Kruskal-Wallis [11] y Post hoc Connover-Iman [12]. Al realizar la primera se concluye que todas las variantes rechazan la hipótesis nula (H0), en la que cada variante provendría de una distribución normal, y por tanto son necesarias pruebas no paramétricas para comparar las variantes.

Con la prueba de Kruskal-Wallis se comprueba para cada par (NP, NH), si todos las variantes provienen de la misma población (es decir, tienen la misma mediana, H0), o al menos una de ellas proviene de una distribución diferente (H1). Para los casos en los que se rechaza H0 se realiza la prueba de Post hoc Connover-Iman para descubrir qué variantes son diferentes para ese mismo par (NP, NH).

La Figura 6 muestra en una cuadrícula la variante óptima para realizar una distribución de datos para cada par (NP, NH), tanto para Ethernet (Izquierda) como para Infiniband (Derecha). El triangulo superior se asocia con las expansiones de la aplicación, mientras que el triangulo inferior lo hace con las reducciones. El número y color de cada celda muestran la variante más rápida para realizar la redistribución según las pruebas estadísticas. En caso de igualdad en una celda, se selecciona aquella variante que haya sido más seleccionada en el resto de celdas.

Una primera observación de la figura permite concluir que la variante *Merge-P2PA* es la más rápida en cualquiera de las dos redes, excepto tres excepciones en el caso de Ethernet y una para Infiniband.

La elección de las variantes *Merge*, respecto de las variantes *Baseline* y *BaselineIntra*, se justifica por la suscripción en exceso que generan estos últimos y que hace que aumente su tiempos de operación.

Por su parte, la elección de las comunicaciones *P2P* en lugar de las *COL*, se justifica por una diferencia significativa en sus medianas. La implementación de la variante *Merge-P2P* es muy similar al algoritmo *Scattered* de la función `MPI_Alltoallv` de MPICH utilizado en las variantes *Merge-COL*. Es por ello que las validaciones que se realizan al inicio del algoritmo *Scattered* son la principal razón que puede justificar esta diferencia.

Las tres excepciones de la red Ethernet se producen al reducir de $NP = 80$ o más procesos padre a $NH = 20$. En estos casos la variante más eficiente es la *Merge-P2PS*. En cambio, la excepción de la red Infiniband aparece al expandir de $NP = 2$ a $NH = 20$ procesos, siendo la variante más rápida la *Merge-COLS*. Estas excepciones ocurren porque el tiempo máximo de la variante *Merge-P2PA* ha sido superior al de las excepciones con subidas desde $0,2s$ a $0,6s$, pero en todos los casos la cota inferior de *Merge-P2PA* es mejor por $0,1s$.

V. Conclusiones

En este artículo se han analizado 12 variantes diferentes para realizar la etapa de redistribución de datos durante la reconfiguración de una aplicación MPI maleable. Primero se presentan 2 variantes, una ba-

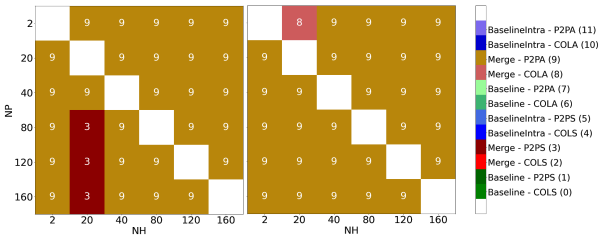


Fig. 6: Variantes óptimas para redistribuir datos en Ethernet (Izquierda) e Infiniband (Derecha).

sada en comunicaciones punto a punto (*P2P*) y otra en comunicaciones colectivas (*COL*). A continuación se indica como utilizar dichas variantes utilizando la versión *síncrona* o *asíncrona* de estas operaciones, creando dos alternativas adicionales.

Además, las variantes pueden tener comportamientos diferentes según el método utilizado en la creación de procesos: *Baseline* o *Merge*.Pero es posible seleccionar si el método *Baseline* usa intercomunicadores o intra-comunicadores, mientras que con el método *Merge* solo es posible utilizar intracomunicadores, dando lugar a tres nuevas alternativas.

También hay que destacar que los métodos de redistribución basados en comunicaciones *COL* solo requieren realizar llamadas a funciones colectivas de MPI, mientras que para las comunicaciones *P2P*, se han tenido que diseñar dos algoritmos basados en operaciones no bloqueantes para evitar interbloqueos.

El estudio experimental de las variantes realiza la redistribución de $5Gb$ de datos desde los procesos padre a los hijos, en una aplicación que solo lleva a cabo las etapas 2 y 3 de la maleabilidad, analizando el comportamiento en dos redes existentes en el sistema, Ethernet 10G e Infiniband EDR. El análisis de los tiempos permite concluir que la variante *Baseline-COLS* presenta un coste más elevado que el resto, debido al algoritmo que utiliza MPICH para implementar `MPI_Alltoallv` sobre inter-comunicadores que se basa en operaciones bloqueantes. El resto de variantes convergen a tiempos similares con diferencias máximas de $0,6s$, salvo los casos de reducción de procesos con ($NH = 2$), en los que aparece un comportamiento irregular debido a la congestión que sufren los dos procesos hijo.

Finalmente se han comparado estadísticamente todos las variantes modificando los valores de NP y NH, obteniendo que la variante *Merge-P2PA* es la óptima en 56 de 60 casos. Hay tres razones que justifican esta elección: (i) evitar el estado de suscripción en exceso del método *Baseline*; (ii) realizar las comunicaciones sin un orden predeterminado, gracias al uso de comunicaciones no bloqueantes; y (iii) evitar las validaciones internas de la operación `MPI_Ialltoallv`, que no se realizan al utilizar operaciones *P2P*.

El trabajo futuro se centrará en analizar el efecto de solapar la realización de las etapas 2 y 3 de la maleabilidad con el procesamiento de una aplicación científica, y su impacto sobre las prestaciones. También se diseñarán nuevos métodos de redistribución de datos basadas en comunicaciones RMA (*Remote Memory Access*) de MPI.

Agradecimientos

El presente trabajo ha sido subvencionado por el proyecto PID2020-113656RB-C21 financiado por MCIN/AEI/10.13039/501100011033. El trabajo del investigador I. Martín-Álvarez fue subvencionado por la ayuda predoctoral ACIF/2021/260, financiada por el Gobierno Autonómico Valenciano y por la European Social Funds.

Referencias

[1] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 4.0*, June 2021.

[2] Jonas Posner and Claudia Fohry, "Transparent resource elasticity for task-based cluster environments with work stealing; transparent resource elasticity for task-based cluster environments with work stealing," *50th International Conference on Parallel Processing Workshop*, 2021.

[3] Sergio Iserte, Héctor Martínez, Sergio Barrachina, Maribel Castillo, Rafael Mayo, and Antonio J Peña, "Dynamic Reconfiguration of Noniterative Scientific Applications," *The International Journal of High Performance Computing Applications*, p. 109434201880234, sep 2018.

[4] Sergio Iserte, Rafael Mayo, Enrique S. Quintana-Ortí, Vicenç Beltran, and Antonio J. Peña, "DMR API: Improving Cluster Productivity by Turning Applications into Malleable," *Parallel Computing*, vol. 78, pp. 54–66, oct 2018.

[5] Gonzalo Martín, David E. Singh, Maria-Cristina Marinescu, and Jesús Carretero, "Enhancing the Performance of Malleable MPI Applications by Using Performance-aware Dynamic Reconfiguration," *Parallel Computing*, vol. 46, pp. 60–77, jul 2015.

[6] Chao Huang, Orion Lawlor, and L. V. Kalé, "Adaptive MPI," in *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003), LNCS 2958*, College Station, Texas, October 2003, pp. 306–322.

[7] Iker Martín-Álvarez, José I Aliaga, Maribel Castillo, Sergio Iserte, and Rafael Mayo, "Dynamic spawning of mpi processes applied to malleability," *The International Journal of High Performance Computing Applications*, vol. 0, no. 0, pp. 10943420231176527, 0.

[8] Nicholas Radcliffe, Layne Watson, and Masha Sosonkina, "A comparison of alternatives for communicating with spawned processes," in *Proceedings of the 49th Annual Southeast Regional Conference*, New York, NY, USA, 2011, ACM-SE '11, p. 132–137, Association for Computing Machinery.

[9] MPICH Development team, "Mpich website," `https://www.mpich.org/`.

[10] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

[11] W.H. Kruskal and W.A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[12] W J Conover and R L Iman, "Multiple-comparisons procedures. informal report," 2 1979.

Modelado y análisis de topologías de red jerárquicas para redes de interconexión de altas prestaciones en supercomputadores y centros de datos

Carlos Medrano Navalón[1], Jesús Escudero-Sahuquillo[2], Pedro Javier García[2], Francisco J. Quiles[2]
{Carlos.Medrano, Jesus.Escudero, PedroJavier.Garcia, Francisco.Quiles}@uclm.es

Resumen— **En los últimos años, en la era del Big-Data, la demanda de sistemas multiprocesador de alto rendimiento y de centros de datos a gran escala ha aumentado considerablemente. Además, se prevé que esta demanda aumente en los próximos años debido a los rápidos avances tecnológicos y a las enormes cantidades de datos que es necesario procesar. En estos sistemas, la red de interconexión es un elemento clave. Una de las principales características de una red de interconexión es su topología, que es el patrón de interconexión que siguen sus elementos.**

En este artículo se realiza un estudio de una topología jerárquica propuesta recientemente, la Megafly (o Dragonfly+). Además de analizar sus propiedades, se ha modelado la topología en una herramienta de simulación, realizando primero una modificación en su arquitectura para facilitar la implementación de más topologías jerárquicas. Posteriormente, este modelo se ha ejecutado con varias configuraciones para obtener métricas de rendimiento y comparar sus resultados con otras topologías existentes.

Palabras clave— **Redes de interconexión, Dragonfly, Megafly, encaminamiento, simulación.**

I. Introducción

En los últimos años, el desarrollo tecnológico y la digitalización de la sociedad han provocado un gran aumento de las necesidades de potencia de cálculo. Se necesitan sistemas informáticos grandes y potentes para ejecutar determinadas aplicaciones de software, como modelos complejos de ingeniería y ciencia. Además, el rápido incremento de la cantidad de datos generados diariamente implica la necesidad de utilizar centros de datos. Los centros de datos son infraestructuras que ofrecen múltiples servicios que requieren baja latencia y alta potencia de cálculo, como la computación en la nube, los modelos de inteligencia artificial, el almacenamiento masivo en la nube o las aplicaciones de Big-Data. Se trata de sistemas con miles de nodos necesarios para recoger y procesar de forma eficiente las enormes cantidades de datos que se generan continuamente.

En estos sistemas, la red de interconexión es un componente clave, ya que es el elemento que permite una comunicación eficiente entre todos los nodos que deben trabajar conjuntamente. Por ello, el diseño de la red de interconexión debe ajustarse a los requisitos del sistema (debe ser escalable, tener un ancho de banda equilibrado, mecanismos de control de flujo adecuados, ajustarse al presupuesto, etc.), ya que juega un papel muy importante en su rendimiento: si no se diseña adecuadamente, puede convertirse en el cuello de botella y afectar negativamente a su rendimiento.

Para diseñar correctamente la red de interconexión hay que tener en cuenta múltiples parámetros: la topología, el algoritmo de encaminamiento, la técnica de control de flujo, la arquitectura de los switches, etc. En particular, este artículo se centra en las topologías y los algoritmos de encaminamiento. Las topologías pueden clasificarse en múltiples categorías en función de sus características. A lo largo de los años, se han propuesto nuevas topologías para resolver los problemas que presentaban las existentes en determinadas circunstancias: eficiencia, escalabilidad, coste de implementación, complejidad de encaminamiento, dificultad para implementarlas en la vida real (sobre todo debido a las restricciones de longitud de los cables), etc. En consecuencia, existe una gran variedad de topologías que pueden ser elegidas para un sistema de computación de altas prestaciones.

Partiendo de esta base, este artículo se centra en el estudio y modelado de una topología jerárquica propuesta recientemente: Megafly, o Dragonfly+ [1]. Para ello, se utilizarán varias herramientas de simulación, con objetivo de modelarla y evaluar su rendimiento en comparación con otras topologías ya existentes.

El resto del artículo está organizado de la siguiente forma. En la Sección II, se revisan los antecedentes de las redes de interconexión de altas prestaciones, la topología jerárquica Megafly y las herramientas de simulación utilizadas para el desarrollo del trabajo. La Sección III describe el procedimiento realizado para facilitar la implementación de topologías jerárquicas en la biblioteca TopGen, modelar la topología Megafly y su algoritmo de encaminamiento determinista. En la Sección IV, se describen los experimentos que hemos llevado a cabo, y se analizan los resultados de rendimiento de la topología Megafly respecto a otras topologías implementadas previamente. Por último, se extraen algunas conclusiones en la Sección

[1]Instituto de Investigación en Informática de Albacete, Universidad de Castilla-La Mancha
[2]Dpto. Sistemas Informáticos, Universidad de Castilla-La Mancha

V.

II. Antecedentes

A. Redes de interconexión

En este apartado, se detallarán los conceptos fundamentales sobre redes de interconexión de altas prestaciones, necesarios para comprender el resto del artículo.

La red de interconexión desempeña un papel crítico en el comportamiento global de un sistema, ya que su mal funcionamiento puede degradar significativamente la productividad del mismo. De hecho, en la mayoría de los sistemas actuales, la red actúa como cuello de botella, lo que la convierte en un factor clave para lograr un rendimiento óptimo. Sin embargo, diseñar una red para un sistema complejo puede convertirse en un gran reto, ya que se trata de un problema complicado que debe tener en cuenta diversas restricciones como el coste, el suministro eléctrico y la distribución física de los elementos [2]. Además, no existe una solución universal para todos los casos, ya que el diseño de la red debe ser adecuado para satisfacer los requisitos únicos de cada caso de uso. En consecuencia, deben tenerse en cuenta detalles de diseño específicos para lograr los resultados de rendimiento deseados y, al mismo tiempo, satisfacer unas especificaciones determinadas.

Uno de los parámetros clave en las redes de interconexión es la topología, que constituye la forma en la que se interconectan los elementos de la red [3]. Estos elementos pueden ser nodos de cómputo, switches, o enlaces entre ellos. En función de cómo se organicen estos elementos, las topologías se pueden clasificar en directas, indirectas, híbridas o jerárquicas.

Las redes directas son aquellas en las que cada nodo de la red actúa como terminal (o nodo final) y como switch. Por lo tanto, los nodos de la red están conectados directamente entre sí, sin necesidad de utilizar switches intermedios. Las topologías directas más comunes son las mallas, toros e hipercubos. Dada su regularidad, es muy sencillo implementar algoritmos de encaminamiento para redes directas. No obstante, este tipo de topologías no son adecuadas para redes de gran escala, ya que la distancia media entre los nodos aumenta, lo cual se traduce en una mayor latencia.

Las redes indirectas proporcionan conectividad entre los nodos finales mediante switches, en lugar de conectar directamente los nodos entre sí como hacen las topologías directas [4]. Así, los nodos no incluyen un switch en su interior, sino que se conectan a los switches a través de adaptadores de red. Cada switch tiene un número fijo de puertos, denominado "radix". Estos puertos pueden utilizarse para conectar nodos finales u otros switches. Las topologías indirectas permiten crear redes más baratas con características similares a las de las directas. En el caso ideal, en lugar de crear una red directa donde todos los nodos se conectan con todos, todos los nodos podrían conectarse a un mismo switch, reduciendo significativamente el coste de la red. Esta topología se conoce como crossbar. Otras topologías indirectas comunes son las MINs[1] y los Fat-Trees [5].

Las redes híbridas son aquellas que no cumplen completamente las características para ser directas o indirectas, sino que mezclan características de ambas. Las topologías híbridas pretenden ser escalables y rentables, ofreciendo baja latencia, gran ancho de banda y diversidad de rutas [6]. Mientras que las topologías directas ofrecen un coste de implementación reducido pero presentan problemas de implementación para más de 3 dimensiones y limitan el rendimiento de la red para supercomputadores con un elevado número de nodos, y las topologías indirectas proporcionan un alto rendimiento para grandes sistemas pero implican un coste de hardware muy elevado, las topologías híbridas son una alternativa intermedia, ya que presentan un buen rendimiento con un menor coste [7]. Aunque su rendimiento no es tan alto como el de las redes indirectas y su coste es superior al de las topologías directas, ofrecen una gran relación rendimiento-coste. Un ejemplo típico de topología híbrida es la KNS.

Las redes jerárquicas establecen múltiples niveles de jerarquía por los que puede viajar la información. Es habitual que estos niveles jerárquicos utilicen diferentes tecnologías, por lo que es más fácil ajustar el coste de la red a sus necesidades. Algunas de las topologías jerárquicas más relevantes en los últimos años han sido las Dragonfly canónicas [8] y sus variaciones. Una de las variaciones recientemente propuestas de la Dragonfly, es la topología Dragonfly+ [1], también denominada Megafly [2]. La topología Megafly, que es el centro de este trabajo, será descrita en la Sección II-B.

B. La topología Megafly

La topología Megafly (o Dragonfly+) es una variante de la Dragonfly canónica. Una Dragonfly canónica está estructurada en tres niveles de jerarquía: switch, grupo y sistema. El sistema está formado por una serie de grupos conectados con enlaces globales. Cada grupo tiene una topología interna que conecta los nodos finales con enlaces locales. En una Dragonfly canónica, cada switch tiene p procesadores conectados directamente a él, $a - 1$ enlaces con otros switches del mismo grupo (a es el número de switches por grupo) y h enlaces globales (enlaces con otros grupos). Por lo general, los switches de un grupo tienen una topología totalmente conectada, y cada grupo tiene (al menos) una conexión con los demás. En la Figura 1 se muestra una topología Dragonfly canónica de 72 nodos.

La diferencia principal de una Megafly con respecto a una Dragonfly típica es que los grupos están formados por una red tipo Clos (véase la Figura 2), en lugar de conectarse todos con todos. La topología interna a un grupo tiene 2 etapas de switches: los switches globales (*spine*), que conectan el grupo con el resto de grupos; y los switches de nodo (*leaf*), que conectan los nodos de cómputo al grupo. Normal-

[1] redes multietapa

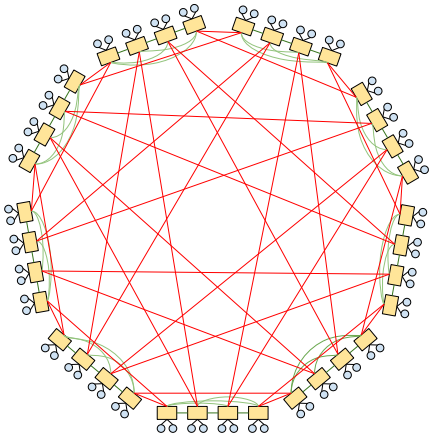


Fig. 1: Topología Dragonfly de 72 nodos, donde $a = 4$ y $p = h = 2$.

mente, cada switch de nodo conecta con todos los switches globales, y viceversa.

Al igual que en las Dragonfly canónicas, estos parámetros se pueden expresar en forma de variables: un switch de nodo tiene p nodos y s switches globales conectados a él, y un switch global tiene l switches de nodo y h enlaces globales a otros grupos. Según estas variables, la restricción para que una Megafly esté equilibrada sería: $p = l = s = h$, donde $p = radix/2$. Aunque no es obligatorio, se recomienda equilibrar los parámetros de la red, al igual que en las Dragonflies, para evitar problemas con el ancho de banda. Estos cambios respecto a las Dragonfly clásicas proporcionan las siguientes ventajas [2]:

- **Mayor escalabilidad**: permite conectar más nodos utilizando switches con el mismo número de puertos.
- **Alta flexibilidad**: permite ajustar con más precisión el coste de la red, al poder realizar cambios en la estructura interna de los grupos.
- **Asegura la evitación de interbloqueos** (*deadlocks*) utilizando un único canal virtual y encaminamiento determinista.
- **Mayor rendimiento**.

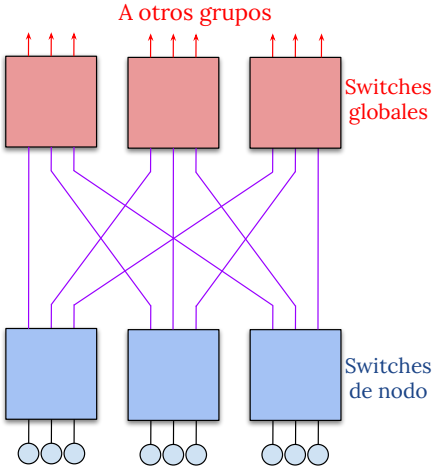


Fig. 2: Ejemplo de un grupo de Megafly, donde $p = l = s = h = 3$

C. Herramientas de simulación

La demanda de una mayor capacidad de procesamiento proviene principalmente de aplicaciones específicas que requieren una gran potencia de cálculo. En particular, hay muchos problemas científicos y de ingeniería que son demasiado complejos o costosos para evaluar en la vida real, por lo que se abordan mediante el uso de herramientas de simulación software. Implementar redes de interconexión de altas prestaciones es un proceso costoso y complejo debido a la tecnología específica necesaria, las limitaciones de espacio físico, el consumo de energía, la disipación de calor, la necesidad de ingenieros especializados, cableado, etc. En consecuencia, el diseño e implementación de una red de interconexión es uno de estos desafíos que se pueden resolver con herramientas de simulación, para permitir a los diseñadores experimentar con parámetros de red y estimar cuál es la mejor solución para sus requisitos en la vida real.

El grupo RAAP del Instituto de Investigación de Informática de Albacete desarrolla diversas herramientas de simulación de redes de interconexión con este objetivo. Dos de ellas son la biblioteca TopGen y el simulador INASim, que han sido utilizados para este trabajo. TopGen [9] es una biblioteca de generación de topologías que puede ser utilizada como submódulo con cualquier simulador de redes de interconexión. Es de código abierto[2] y está desarrollada en C++. Además, TopGen ofrece una variedad de topologías como un conjunto de abstracciones de componentes de red (switches, enlaces y nodos), así como información de encaminamiento. Por otro lado, INASim (Interconnection Networks Agile Simulator) [10] es un simulador de redes de interconexión de altas prestaciones. Desarrollado en C++ utilizando un paradigma de programación orientado a objetos, ofrece modularidad para facilitar la implementación de nuevas funciones. INASim puede simular diferentes arquitecturas de red y generar archivos de salida con estadísticas de rendimiento. Estas estadísticas pueden utilizarse para crear gráficos personalizados o analizarse con otras herramientas de software. Además, INASim permite ejecutar simulaciones en *clusters* de supercomputación. El simulador funciona de forma dirigida por eventos, ejecutando secuencialmente los eventos en función de sus dependencias y generando nuevos eventos durante la ejecución.

III. Descripción del modelado

A. Facilitando la implementación de topologías jerárquicas

Antes del inicio de este trabajo, la biblioteca TopGen se implementó para comportarse como un módulo que permite obtener información de una topología. Ofrece un compendio de topologías conocidas que se implementan por separado, de modo que, aunque muchas de ellas tienen una estructura similar, el desarrollo de una nueva topología no depende de las

[2]El código de TopGen se puede acceder desde el siguiente repositorio: `https://gitraap.i3a.info/jesus.escudero/libtopgen/-/tree/Development?ref_type=heads`

demás, y los desarrolladores tienen libertad para implementar nuevas topologías como deseen mientras satisfagan la interfaz genérica de topología.

Este diseño tiene la ventaja de proporcionar aislamiento entre topologías distintas y dar libertad a los programadores para implementar la lógica detrás de cada una, pero puede mejorarse permitiendo la reutilización de código en topologías que tengan algunas propiedades similares (en este caso, las topologías jerárquicas). Por lo tanto, en lugar de simplemente añadir la topología Megafly como una más, como se había hecho hasta el momento, se modificará la estructura de clases para crear nuevas clases e interfaces de topologías jerárquicas. Estas nuevas clases generalizarán algunas de las características comunes a las mismas, principalmente a las variantes de Dragonfly (dado que era la única topología jerárquica ya existente en la biblioteca), de modo que futuros experimentos con este tipo de redes requieran menos esfuerzo de desarrollo.

Por ejemplo, la nueva implementación permitirá el uso de otras topologías existentes como grupos dentro de topologías jerárquicas. Los grupos Megafly están compuestos por redes tipo Clos, que pueden modelarse en TopGen utilizando la clase `CXGFT`[3]. Los XGFTs proporcionan una amplia definición de los fat-trees, por lo que, con esta nueva implementación, podríamos modificar la estructura interna de cada grupo de manera muy sencilla, manteniendo el patrón de interconexión entre grupos y sin tener que programar una nueva topología adicional (lo cual implicaría duplicar código, facilitaría la introducción de errores y complicaría innecesariamente la biblioteca).

Sin embargo, no cualquier topología existente puede constituir un grupo, ya que necesita cumplir una serie de restricciones (como tener puertos libres para realizar la interconexión global). Para garantizar esto, se incluirá la interfaz `IGroupableTopology`, que tendrán que implementar obligatoriamente las topologías que vayan a constituir grupos jerárquicos. Esta interfaz no modificará el comportamiento actual de las topologías que la implementen, por lo que se podrán seguir simulando por sí solas como hasta ahora. Este modelo puede ser muy útil en el futuro para experimentar con poco esfuerzo con variantes de Dragonfly que contengan toros, mallas, hipercubos u otras topologías dentro de sus grupos, ya que la jerarquía global será la misma y no será necesario reprogramarla.

Conforme a estos cambios, se propone una nueva estructura de clases. A continuación, se detallan los cambios más significativos presentes en la nueva estructura:

- `CHierarchicalTopology`. La clase de topología jerárquica es el núcleo de la nueva estructura de implementación de topologías jerárquicas. Su propósito es ser la clase base para todas las topologías jerárquicas existentes en la biblioteca, definiendo sus características comunes. Así, cada nueva topología jerárquica derivará de esta clase, heredando sus atributos y métodos (aunque se pueden sobrescribir en caso de que la nueva topología tenga alguna característica especial).
- `CHierarchicalGroup`. La clase de grupo jerárquico es la clase genérica para cualquier grupo que forme una topología jerárquica. Toda topología jerárquica estará formada por la interconexión de varios grupos entre sí. Por tanto, las clases específicas de cada grupo derivarán de `CHierarchicalGroup` de forma análoga a como las topologías jerárquicas derivarán de `CHierarchicalTopology`. Además, los grupos jerárquicos implementan la interfaz `IGroupableTopology`, que será la topología real dentro del grupo que contiene la información sobre la disposición de los elementos de la red.
- `IGroupableTopology`. Es la interfaz ya mencionada que implementarán las topologías que se pueden utilizar dentro de un grupo. Por el momento, la interfaz sólo tiene un constructor `protected` por defecto (para que sólo las clases derivadas puedan utilizar este constructor para construirse a sí mismas) y la cabecera para el método `GetFreePorts()`, que debe ser implementado por todas las topologías derivadas. Este método devuelve una lista de tuplas del tipo `<switch,port>` con la lista de puertos libres que pueden utilizarse para establecer la interconexión global entre grupos.
- `CMFly y CMFlyGroup`. Estas son las clases principales que modelan la topología Megafly, que es el objetivo principal de este trabajo. Es la primera topología jerárquica modelada siguiendo esta estructura, aparte de Dragonfly, que se programó originalmente y luego se adaptó a la nueva estructura.

B. Modelado de la topología Megafly

Para implementar la Megafly en TopGen, se han seleccionado las partes del código existente de Dragonfly (que es la única topología jerárquica implementada previamente) que pueden ser comunes al resto de topologías jerárquicas. Este código se incluirá en `CHierarchicalTopology` y `CHierarchicalGroup`. Los principales parámetros que se pueden generalizar son:

- **Parámetros generales de la topología** (como número de nodos, número de grupos, identificador del grupo, etc).
- ***Array* de grupos**, que pasará de ser de objetos `CDFlyGroup` a `CHierarchicalGroup` (dado que los grupos de Dragonfly ahora heredarán de los grupos jerárquicos).
- **El patrón de interconexión global** de los grupos. Es un patrón de conexión todos con todos que puede ser utilizado en el resto

[3]Las redes Clos se modelan específicamente con la clase `CRLFT`, pero los RLFT no dejan puertos libres en los switches de la etapa superior, necesarios para interconectar el grupo con otros grupos.

de topologías. Todas las conexiones de Dragonfly estaban implementadas en el método `BuildInterconnection()`, por lo que la parte interna al grupo (que es diferente en Dragonflies y Megaflies) se ha extraído a un nuevo método `BuildLocalInterconnection()`. Además, la parte global se ha movido al mismo método `BuildInterconnection()`, pero en la clase base. En caso de que los desarrolladores quisieran desarrollar una topología con un patrón de interconexión global diferente, el método podría ser sobrescrito en la clase de topología específica.

A continuación, se ha implementado el resto de la lógica detrás de la topología, así como un algoritmo de encaminamiento determinista. Una consideración de diseño importante que se ha tomado es que (para este primer modelo) las Megaflies proporcionadas por TopGen estarán equilibradas, es decir, cumplirán obligatoriamente la condición $p = l = s = h$ establecida en [1]. Esto restringirá la flexibilidad de la topología, pero facilitará la implementación y garantizará que las topologías estén equilibradas para aprovechar al máximo el ancho de banda de bisección. Además, como ocurre en la implementación de la Dragonfly, sólo habrá un enlace entre cada par de grupos.

Para crear la topología, el *array* de grupos contendrá los objetos de cada grupo inicializados, que encapsularán XGFTs en su interior. Estos grupos se conectarán mediante los enlaces usados por los XGFT, por lo que no es necesario desarrollar una nueva clase "`CMflyChannel`". Los canales globales se almacenarán en la lista `m_pGlobalChannelList`, dado que los enlaces locales ya están generados y almacenados en cada objeto de tipo `CXGFT`. Además, la topología tendrá atributos para los parámetros necesarios (p,l,s,h,a) y un puntero al objeto del algoritmo de encaminamiento.

Esta estructura genera un nuevo problema, que es la numeración de los elementos y el hecho de que todos los elementos de la topología deben ser almacenados en la lista del objeto `CMFly`, en lugar de estar dentro de cada objeto `CXGFT`. La solución a este problema pasa por crear unos métodos que transformen todos los identificadores locales de la topología de cada grupo en identificadores globales, e insertar todos estos elementos (enlaces, switches y adaptadores de red) en las listas de la topología jerárquica.

C. *Implementación del algoritmo de encaminamiento*

El encaminamiento en TopGen se implementa mediante el método `GetOutputPort(int identificador, int destino)`, que forma parte de la interfaz y puede invocarse con el identificador de un switch y el identificador de un nodo para obtener el puerto de salida correspondiente para un paquete en el switch dado. Por lo tanto, cada switch en un simulador puede tener una LFT (*Linear Forwarding Table*) con una entrada por cada nodo de destino, donde el contenido es el puerto de salida que debe ser utilizado en ese switch para alcanzar ese nodo según lo especificado por el algoritmo de encaminamiento determinista.

En el caso de la Megafly, se ha implementado un encaminamiento determinista mínimo mediante el método `DeterministicRouting` declarado en la clase `CMFlyRoutingAlgorithm`. Así, `GetOutputPort` ejecutará `DeterministicRouting` si el encaminamiento especificado es mínimo, y lanzará un error y detendrá la ejecución en caso contrario. El encaminamiento determinista mínimo desarrollado se basa en el algoritmo de encaminamiento Dragonfly existente y en el encaminamiento D-mod-K (que se utilizará dentro de los grupos). Aunque los XGFT estaban ya implementados en TopGen, aún no disponían de funciones de encaminamiento, por lo que ha sido necesario obtener la implementación de D-mod-K para los RLFT y adaptarla para que funcionara en los XGFT. Es importante mencionar que la implementación de este algoritmo puede no funcionar para cualquier configuración posible de XGFT, ya que hay muchas configuraciones posibles de árboles que se pueden representar como un XGFT, y pueden tener características diferentes. Sin embargo, es válido para redes tipo Clos, por lo que funciona para los árboles que se instanciarán dentro de las Megaflies.

Una vez que se dispone del encaminamiento interno de los grupos, el algoritmo mínimo debe invocar el encaminamiento D-mod-K para los paquetes que se dirigen a nodos del mismo grupo. Si el paquete debe salir del grupo, se enviará al switch global que conecta con el grupo de destino (sólo hay un enlace que conecta cada par de grupos). Siguiendo este algoritmo, es imposible que un switch global reciba un paquete que no esté dirigido a ese grupo, por lo que los switches globales proporcionarán un valor `UINT_MAX` fijo si eso ocurriera debido a un error. En la Figura 3 se representa gráficamente el algoritmo descrito.

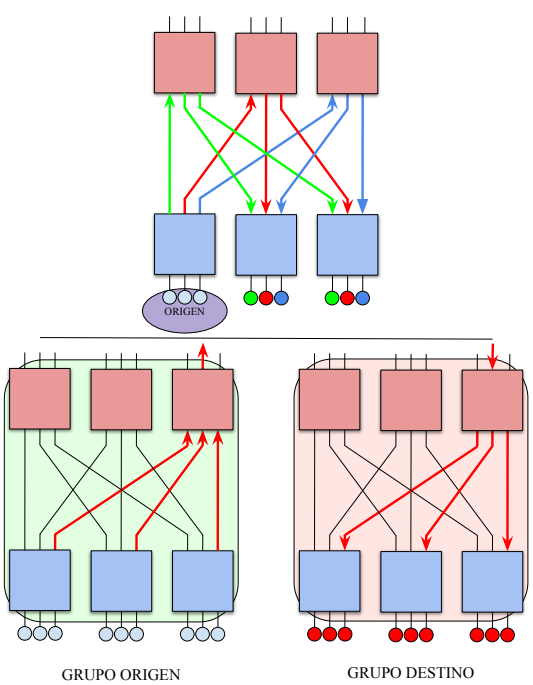


Fig. 3: Encaminamiento determinista mínimo en Megafly.

IV. Resultados experimentales

A. Configuración de los experimentos

Para evaluar el rendimiento de la topología Megafly, se han ejecutado varias configuraciones de esta topología con el simulador INASim. A continuación, se han simulado otras topologías similares (Dragonfly y Real Life Fat-Trees) con parámetros análogos y se compararán los resultados.

Los experimentos con las topologías comparadas se han ejecutado utilizando 2 patrones de tráfico diferentes: aleatorio y tornado. El tráfico aleatorio genera un 100 % de tráfico aleatorio en cada nodo, por lo que cada nodo envía tráfico a un destino aleatorio. Sin embargo, el tráfico aleatorio *all-to-all* no es la mejor forma de evaluar el rendimiento de la red, ya que equilibra de forma natural la carga entre canales, y las aplicaciones reales no suelen seguir este patrón de tráfico en sus comunicaciones. Por otro lado, el patrón tornado genera paquetes de forma que el nodo final i envía mensajes al nodo $i + ((N-1)/2) \, mod \, N$, donde N es el número de nodos de la red. Este patrón de tráfico es adversario, por lo que no equilibra la carga. En consecuencia, tornado es un patrón más "realista" para evaluar el rendimiento de la red en las aplicaciones reales. Según [2], el encaminamiento determinista proporciona una eficiencia baja en redes jerárquicas bajo tráfico tornado. En INASim se implementan tanto patrones tornado como aleatorios, así como otros patrones relevantes.

Para el resto de parámetros de la red, se fijan en los siguientes valores:

- Se utiliza 1 VL[4] (la evitación de *deadlocks* está garantizada con encaminamiento mínimo determinista en Megafly, como se indica en [2]).
- Control de flujo PFC.
- Red con pérdidas.
- Tamaño de paquete de 1024 B.
- Switches con arquitectura de memoria compartida.
- El algoritmo de encaminamiento utilizado es determinista mínimo.
- Los experimentos simulan la inicialización de la red y su estabilización al cabo de cierto tiempo.

A.1 Configuraciones de Megafly

Se han simulado múltiples redes con distintos tamaños, y todas las configuraciones de Megafly están equilibradas. En concreto, mostraremos los resultados de una topología Megafly pequeña, de 90 nodos ($radix = 6$), y otra más grande, de 1332 ($radix = 12$).

A.2 Configuraciones de Dragonfly

La primera topología que hemos comparado con Megafly es la Dragonfly canónica. En este caso, hay un problema: la topología Dragonfly no garantiza el encaminamiento libre de *deadlocks* con sólo 1 VL y encaminamiento determinista, sino que necesita utilizar al menos 2 VLs según [2]. El problema con esto es que el simulador INASim no soporta actualmente el uso de más de un VL para topologías Dragonfly, por lo que las redes Dragonfly que simulemos pueden generar situaciones de interbloqueo. De todos modos, se han probado algunas configuraciones de Dragonfly con un VL y los mismos patrones de tráfico que en Megafly para comprobar si se produce un *deadlock*, como se indica en la bibliografía. Para comparar con Megafly, mostraremos los resultados de una Dragonfly pequeña, de 72 nodos ($radix = 7$), y otra más grande, de 1332 ($radix = 16$).

A.3 Configuraciones de Real Life Fat-Trees

La otra topología que se comparará en este estudio son los RLFT. Los RLFT son topologías indirectas que pueden interpretarse como MIN. Además, también pueden considerarse jerárquicas en el sentido de que tienen múltiples etapas que conectan los nodos entre sí. Los RLFT se han utilizado ampliamente en sistemas de alto rendimiento en las últimas décadas, por lo que son una buena topología para comparar en términos de eficiencia y escalabilidad. También se les atribuye la capacidad de ofrecer un alto rendimiento bajo patrones de tráfico tornado.

Las configuraciones de RLFT se han elegido para acercarse a los tamaños de red de la configuración Megafly en términos de nodos finales. Así, se pueden evaluar el rendimiento y el coste de construir sistemas con un número similar de nodos. Es importante señalar que el *radix* de los switches necesarios para construir un RLFT de 2 etapas con el mismo número de nodos finales que una Megafly o una Dragonfly canónica es significativamente mayor. Por otra parte, las Dragonflies y (especialmente) las Megaflies, requieren un mayor número de switches para ser construidas.

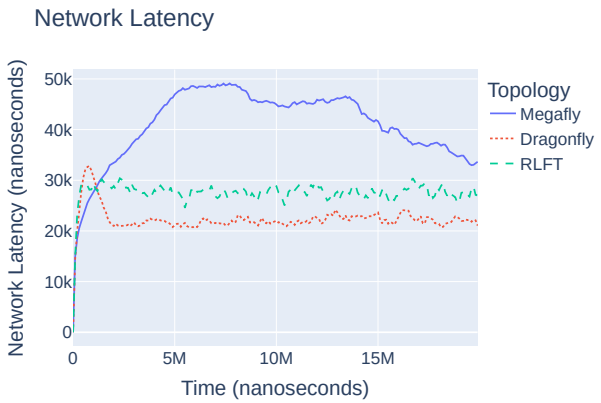
B. Resultados

Los resultados de latencia de red y rendimiento normalizado de las simulaciones se pueden observar en las figuras 4, 5, 6 y 7. Estas figuras muestran los resultados para las simulaciones con tráfico aleatorio (redes del orden de 100 nodos y redes del orden de 1300 nodos) y con tráfico tornado (también para las redes de ambos tamaños), respectivamente.

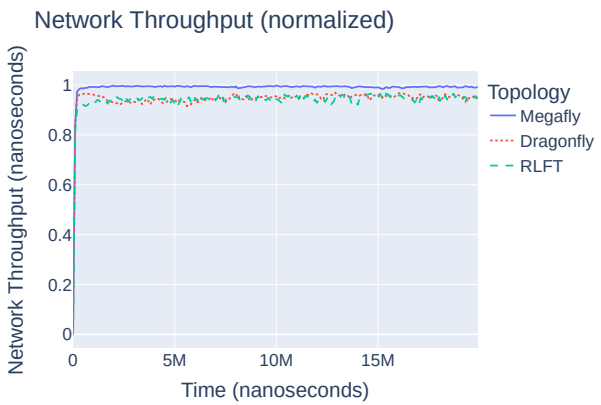
En cuanto a las simulaciones con tráfico aleatorio, la Figura 4a muestra los resultados de latencia para redes del orden de 100 nodos en Megafly, RLFT y Dragonfly, con un patrón de tráfico aleatorio. Puede observarse cómo el RLFT de 98 nodos y 2 etapas ofrece unos resultados de latencia más bajos y estables que la Megafly de 98 nodos. La red Dragonfly con 72 nodos consigue evitar los *deadlocks* en este caso (puede ocurrir, ocasionalmente, que las Dragonflies con 1-VL y encaminamiento determinista no produzcan *deadlocks*, especialmente con tamaños de red pequeños) y, de hecho, ofrece los resultados de latencia más bajos.

La Figura 4b presenta los resultados de rendimiento normalizado para las tres topologías. Se puede ob-

[4]Virtual Lane

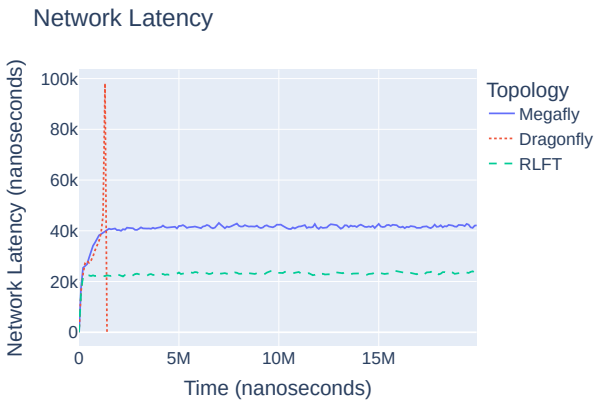


(a) Latencia de red

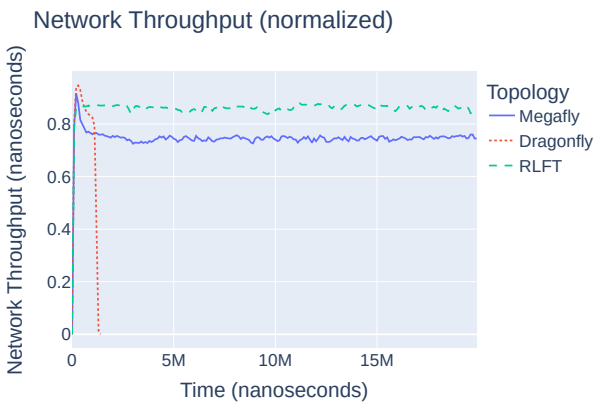


(b) Rendimiento normalizado

Fig. 4: Simulaciones con tráfico aleatorio y topologías de orden de 90 nodos.



(a) Latencia de red



(b) Rendimiento normalizado

Fig. 5: Simulaciones con tráfico aleatorio y topologías de orden de 1300 nodos.

servar cómo la Megafly rinde muy eficientemente con un tamaño de red pequeño, consiguiendo resultados ligeramente mejores que el RLFT. La red Dragonfly también muestra muy buenos resultados de rendimiento, cercanos a los conseguidos por la Megafly.

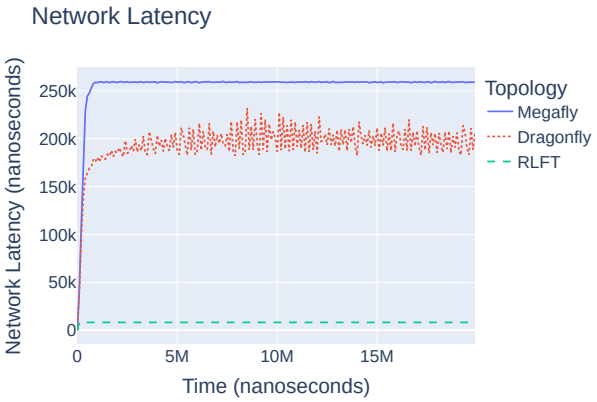
En la Figura 5 se muestran los mismos resultados para las redes de, aproximadamente, 1300 nodos. Podemos comprobar como, al escalar el tamaño de la red, la Dragonfly sí que sufre un *deadlock*, mientras que la Megafly es capaz de evitarlo (así como el RLFT). Aunque el RLFT sigue obteniendo una latencia menor y una eficiencia más alta, la Megafly ofrece unos resultados con variaciones menores que en su versión de 90 nodos.

Ahora, analizaremos los resultados bajo el patrón de tráfico tornado. Según la bibliografía disponible, la topología Megafly ofrece un rendimiento pobre bajo el patrón de tráfico tornado cuando se utiliza encaminamiento determinista, mientras que los Fat-Trees son muy eficientes bajo este patrón. Los resultados obtenidos en el modelo desarrollado muestran el mismo comportamiento. La Figura 6 muestra los resultados de latencia y rendimiento para las redes de alrededor de 100 nodos cuando se ejecuta el patrón de tráfico tornado. Como puede observarse (Figura 6a), la red con topología Megafly ofrece latencias realmente elevadas en comparación con las obtenidas por el RLFT. La Dragonfly rinde mejor, pero sigue siendo superada por el RLFT (cabe recordar que sigue tratándose de una red inválida al garantizar la evitación de *deadlocks*). Respecto al rendimiento (Figura 6b), vemos que los RLFT obtienen un valor del 100 % durante toda la ejecución (es decir, aprovechan todo el ancho de banda disponible), mientras que las redes jerárquicas se encuentran alrededor del 20 %.

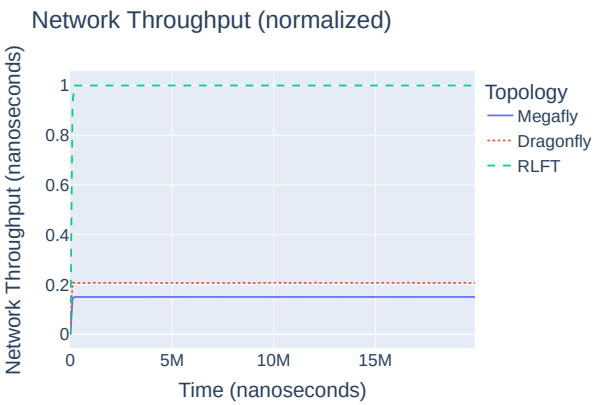
Finalmente, en la Figura 7 se muestran los mismos resultados para las redes "grandes". Estos resultados también muestran que las RLFT son buenas topologías para ejecutar tráfico tornado, ya que ofrecen niveles de eficiencia muy cercanos al 100 %. Mientras tanto, Megafly proporciona latencias muy altas (7a) y eficiencias extremadamente bajas (Figura 7b).

V. Conclusiones

La necesidad de optimizar el rendimiento de las redes de interconexión de superordenadores y centros de datos surge de la demanda de nuevas aplicaciones con elevados requisitos de cómputo. En particular, las topologías tienen un gran impacto en el rendimiento de la red. En este trabajo se ha modelado la topología Megafly en herramientas de simulación de redes y se ha evaluado mediante un estudio basado en un conjunto de experimentos, variando topologías, tamaños de red, y patrones de tráfico. Además, se ha presentado la modificación realizada en la biblioteca TopGen para facilitar la implementación de topologías jerárquicas, dado que resultan de gran interés dada su escalabilidad y relación entre coste y rendi-

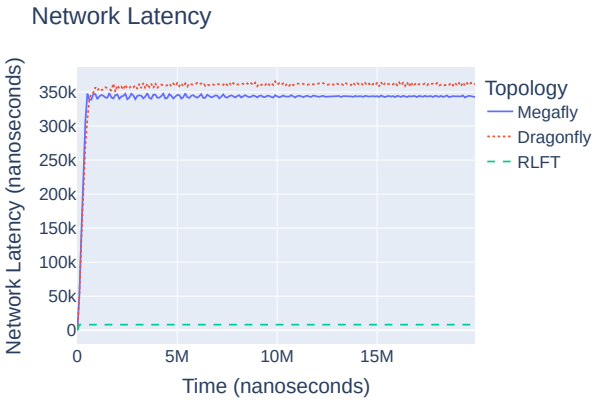


(a) Latencia de red

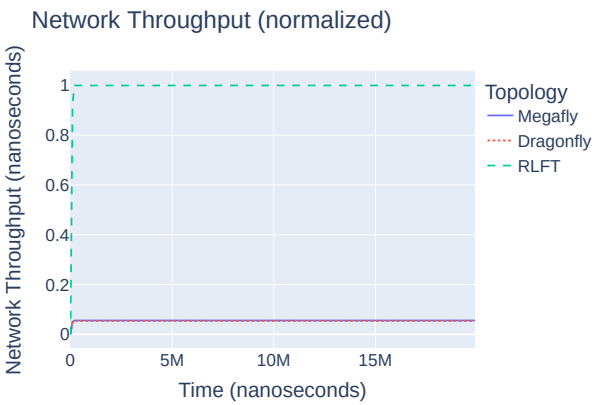


(b) Rendimiento normalizado

Fig. 6: Simulaciones con tráfico tornado y topologías de orden de 90 nodos.



(a) Latencia de red



(b) Rendimiento normalizado

Fig. 7: Simulaciones con tráfico tornado y topologías de orden de 1300 nodos.

miento.

Las principales conclusiones que pueden extraerse sobre esta topología a partir de los resultados obtenidos son que:

1. La topología Megafly mejora a la Dragonfly canónica en términos de requisitos cuando se utiliza encaminamiento determinista, dado que garantiza la evitación de deadlocks con un solo VL, mientras que la Dragonfly canónica necesita más de uno.
2. El rendimiento de las redes Megafly se ve fuertemente perjudicado bajo patrones de tráfico adversos, como el tornado, cuando se utiliza encaminamiento determinista. Por tanto, esta topología no es adecuada para sistemas diseñados para ejecutar aplicaciones que generan patrones de tráfico adversos.
3. Las redes Megafly necesitan switches con *radix* mucho más pequeños que los RLFT de 2 etapas para conectar el mismo número de terminales. Esta es una ventaja de las topologías jerárquicas, dado que el coste de los switches aumenta mucho en función de su número de puertos.
4. Por otro lado, las Megaflies equilibradas necesitan un número elevado de switches en comparación con las Dragonflies canónicas y los RLFT, dado que su relación nodos:switches sea 1:1.
5. Los RLFT muestran un mejor rendimiento general que las Megaflies en los experimentos realizados mediante simulación.

Finalmente, podemos concluir que la topología Megafly puede ser adecuada cuando se desea sacrificar parte del rendimiento para obtener una mayor escalabilidad con un coste menor, al permitir construir topologías grandes mediante switches con un menor número de puertos.

Agradecimientos

Este trabajo ha sido financiado por la Junta de Comunidades de Castilla-La Mancha bajo el proyecto Tetra2 (SBPLY/21/180225/000103), y por la Universidad de Castilla-La Mancha bajo el proyecto 2023-GRIN-34056.

Referencias

[1] Alexander Shpiner, Zachy Haramaty, Saar Eliad, Vladimir Zdornov, Barak Gafni, and Eitan Zahavi, "Dragonfly+: Low cost topology for scaling datacenters," *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, pp. 1–8, 2017.

[2] Mario Flajslik, Eric Borch, and Mike A. Parker, "Megafly: A topology for exascale systems," *LNTCS*, vol. 10876, pp. 289–310, 2018.

[3] William James Dally and Brian Towles, *Principles and Practices of Interconnection Networks*, Morgan Kauffman, 2004.

[4] J. Duato, Sudhakar Yalamanchili, and Lionel Ni, *Interconnection Networks: An Engineering Approach*, Mor-

gan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[5] Charles E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, oct 1985.

[6] Pedro Yebenes, Jose Rocher-Gonzalez, Jesus Escudero-Sahuquillo, Pedro Javier Garcia, Francisco J. Alfaro, Francisco J. Quiles, Crispín Gómez, and Jose Duato, "Combining source-adaptive and oblivious routing with congestion control in high-performance interconnects using hybrid and direct topologies," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 2, apr 2019.

[7] Roberto Peñaranda, Crispín Gómez Requena, María Engracia Gómez, Pedro López, and José Duato, "A new family of hybrid topologies for large-scale interconnection networks," in *11th IEEE International Symposium on Network Computing and Applications, NCA 2012, Cambridge, MA, USA, August 23-25, 2012*. 2012, pp. 220–227, IEEE Computer Society.

[8] John Kim, William J. Dally, Steve Scott, and Dennis Abts, "Technology-driven, highly-scalable dragonfly topology," *2008 International Symposium on Computer Architecture*, pp. 77–88, 2008.

[9] Juan Antonio Villar, German Maglione Mathey, Jesús Escudero-Sahuquillo, Pedro Javier García, Francisco J. Alfaro, José Luis Sánchez Garcia, and Francisco J. Quiles, "Topgen: A library to provide simulation tools with the modeling of interconnection network topologies," in *2018 International Conference on High Performance Computing & Simulation, HPCS 2018, Orleans, France, July 16-20, 2018*. 2018, pp. 452–459, IEEE.

[10] Jesus Escudero-Sahuquillo, Pedro J Garcia, and Francisco J Quiles, "Inasim: Una herramienta para la evaluacion de redes de interconexion de altas prestaciones," .

Diseñando nodos basados en IoT y Redes LPWAN para un proyecto de seguridad medioambiental en entornos rurales

María Ángeles Amador[1], Javier Martínez[1], Celia Garrido-Hidalgo[1], Luis Roda-Sanchez[2], Teresa Olivares[3], Francisco M. Delicado[3], Ismael García-Varea[3] y Jesús Martínez-Gómez[3]

Resumen— **Las tecnologías de IoT son cruciales para la sociedad, permitiendo la interconexión de dispositivos y sistemas para mejorar la eficiencia y seguridad en diversos sectores. En la detección de delitos medioambientales, el uso de sensores y sistemas inteligentes basados en IoT surge como una oportunidad para la detección temprana y una respuesta más efectiva. A pesar de los desafíos, es esencial desarrollar soluciones innovadoras, sostenibles y de bajo costo para proteger nuestros recursos naturales. En este trabajo se presenta el diseño de un nodo IoT desarrollado dentro del marco del proyecto CDTI CAMTIMA, que busca implementar soluciones basadas en IoT y en los nuevos estándares de red LPWAN, como NB-IoT, para la detección temprana y gestión eficiente de delitos medioambientales en entornos rurales de regiones como Castilla-La Mancha.**

Palabras clave— **IoT, LPWAN, NB-IoT, proyecto CAMTIMA, delitos medioambientales, integración de sensores.**

I. Introducción

En el contexto actual, las tecnologías de Internet de las cosas (IoT, Internet of Things) desempeñan un papel crucial en nuestra sociedad, permitiendo la interconexión de dispositivos y sistemas. Esto ha llevado a un crecimiento y transformación significativos en diversas áreas, como la industria, la salud, la agricultura y el transporte. Además, IoT facilita la recopilación y el análisis de datos en tiempo real, lo que proporciona información valiosa para la toma de decisiones estratégicas y la mejora de la calidad de vida.

En el ámbito de la prevención de incendios forestales y los delitos medioambientales, las tecnologías IoT basadas en redes inalámbricas de sensores para la monitorización de variables (WSN, Wireless Sensor Networks) y las redes de comunicación de bajo costo y largo alcance para enviar los datos recogidos en entornos rurales (LPWAN, Low Power Wide Area Networks) desempeñan un papel especialmente relevante. Estos desastres naturales causan daños significativos en el medio ambiente y representan riesgos para la vida humana. Mediante la implementación de sensores y sistemas inteligentes en áreas vulnerables, es posible monitorizar en tiempo real factores como la temperatura, humedad, velocidad y dirección del viento, y la presencia de gases inflamables. Esto permite una detección temprana de incendios y una respuesta más rápida y efectiva por parte de los equipos de emergencia [1].

Sin embargo, implementar soluciones integrales y efectivas para combatir los incendios forestales representa un desafío considerable. Se requiere una tecnología resistente y robusta que pueda funcionar de manera continua, incluso en condiciones meteorológicas adversas. Además, es fundamental garantizar la precisión y confiabilidad de los datos recopilados, así como la eficiencia en la transmisión de los mismos [2]. A pesar de estas dificultades, es necesario desarrollar y adoptar tecnologías IoT innovadoras para abordar la problemática de los incendios forestales y proteger nuestros valiosos recursos naturales. En este contexto, el proyecto CAMTIMA (Control de Activos, Multitudes, Tráfico Ilícito y Medio Ambiente) [3] busca implementar soluciones innovadoras basadas en el IoT para la detección temprana y gestión eficiente de incendios forestales en la región de Castilla-La Mancha (CLM), entre otras cosas. El objetivo principal de este proyecto es optimizar la utilización de los recursos humanos de las Fuerzas de Seguridad, proporcionándoles herramientas automatizadas para combatir los delitos en estas áreas específicas.

En este artículo se propone la creación de un nodo IoT equipado con sensores de bajo costo para la detección de incendios forestales y la monitorización ambiental. Se ha desarrollado y evaluado el nodo, centrándose en el diseño y pruebas del protocolo de comunicaciones NarrowBand IoT (NB-IoT) [4] para transmitir los datos recogidos en tiempo real. El propósito de este artículo es detallar el proceso de diseño e implementación de una solución NB-IoT de prevención de incendios y aportar una serie de lecciones aprendidas en base a experimentos realizados con hardware real.

II. Fundamentos de NB-IoT

Este trabajo surge como respuesta a la creciente adopción de tecnologías LPWAN en el ámbito de IoT, y se enfoca específicamente en su aplicación en un entorno rural para la detección de incendios forestales. En este contexto, la tecnología NB-IoT ha surgido como un estándar inalámbrico para redes LPWAN [5]. Fue definida por el *3rd Generation Partnership Project* (3GPP) en junio de 2016 como una alternativa a las redes LPWAN basadas en el es-

[1]Instituto de Investigación en Informática de Albacete, Universidad de Castilla-La Mancha, e-mail: {mangeles.amador,javier.martinez53}@alu.uclm.es, celia.garrido@uclm.es

[2]NEC Ibérica S.L., e-mail: luis.roda@emea.nec.com

[3]Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, e-mail: {teresa.olivares, celia.garrido, francisco.delicado, ismael.garcia, jesus.martinez}@uclm.es.

pectro no licenciado de radiofrecuencias. NB-IoT se desarrolló sobre la tecnología LTE (Long Term Evolution) y aprovecha el espectro licenciado de frecuencias utilizado por los operadores de telefonía móvil. La elección de NB-IoT en este estudio se basa en su amplia cobertura extendida [6], lo cual resulta fundamental para establecer comunicaciones confiables en áreas rurales y facilitar la detección oportuna de incendios forestales.

El estándar NB-IoT presenta una serie de características técnicas que lo hacen altamente adecuado para diversas aplicaciones de IoT [7]. Estas características abarcan un ancho de banda de 180 kHz, una velocidad de aproximadamente 100 kbps y una impresionante vida útil de la batería de hasta 10 años incluso en situaciones de movilidad constante. Además, en áreas suburbanas, su alcance puede extenderse hasta los 10 km. Con una notable sensibilidad de recepción de -142 dBm, NB-IoT permite una conexión bidireccional y tiene la capacidad de soportar hasta 100,000 conexiones por celda. Es importante destacar que NB-IoT no admite la transmisión de voz y posee una velocidad máxima de descarga y subida de 250 kbps [7].

La arquitectura de red de NB-IoT se basa en la de LTE, pero ha sido optimizada para admitir la conexión masiva de dispositivos IoT [6]. En cuanto a las capas del protocolo, NB-IoT utiliza el protocolo UDP (User Datagram Protocol) en la capa de transporte y el protocolo CoAP (Constrained Application Protocol) en la capa de aplicación. El uso de UDP permite una transmisión más rápida y un menor consumo de energía, mientras que CoAP es un protocolo web diseñado específicamente para dispositivos con recursos limitados. NB-IoT utiliza diversas bandas de frecuencia, con un total de 25 bandas definidas en las versiones 13, 14 y 15 de la tecnología [8].

La tecnología NB-IoT se adapta a diversos escenarios y optimiza los recursos de transmisión gracias a sus diferentes modos de operación [6]:

- Operación autónoma (*in-band operation*). En este modo, se pueden utilizar frecuencias *Global System For Mobile Comunication* (GSM) ya en uso, dado que su ancho de banda permite un intervalo de guarda de 10 kHz a cada lado del espectro. Esto significa que NB-IoT puede operar de manera independiente en frecuencias GSM sin interferir con las comunicaciones existentes.
- Operación de banda de guarda (*guard band operation*). En este modo, se utilizan los bloques de recursos no utilizados dentro de la banda de guarda de un portador LTE. NB-IoT aprovecha estos bloques de recursos no utilizados para transmitir datos de IoT sin afectar la capacidad y calidad de las comunicaciones LTE.
- Operación en banda (*stand alone operation*). En este caso, se emplean bloques de recursos dentro de un portador LTE. A diferencia de la operación de banda de guarda, donde se utilizan los bloques no utilizados, en la operación en banda se asignan bloques de recursos específicos dentro de la capacidad del portador LTE. Esto permite la coexistencia de NB-IoT y LTE en la misma banda de frecuencia [9].

En el nivel físico, NB-IoT utiliza un canal descendente (downlink) con tres canales físicos utilizados para transmitir diferentes tipos de información, junto con las señales de sincronización primaria y secundaria de banda estrecha (NPSS y NSSS). Por otro lado, el canal ascendente (uplink) utiliza dos canales físicos y una señal de referencia de demodulación (DMRS) [6].

En cuanto a la cobertura en España, NB-IoT tiene una amplia cobertura que abarca la mayoría del territorio, aunque algunas áreas montañosas pueden presentar falta de cobertura [4]. Para el objetivo de este artículo, que se centra en implementar un sistema de monitoreo en una zona rural concreta de CLM, la cobertura proporcionada por NB-IoT es adecuada. NB-IoT es un estándar propietario de Vodafone, que es uno de los socios del proyecto donde se enmarca este trabajo.

III. Estado del arte

A continuación, se presentan tres ejemplos de casos de éxito recientes que demuestran el impacto de la tecnología IoT en diferentes sectores [10] y tres estudios científicos relacionados con el trabajo presentado en este artículo.

- Producción agrícola inteligente: Se trata de un laboratorio de agricultura inteligente en Hungría, donde se utilizan sensores IoT para recopilar datos sobre el suelo, el clima y otros parámetros relacionados con los cultivos de maíz y trigo. Estos datos se procesan mediante técnicas de inteligencia artificial y modelos de cultivo para optimizar la producción agrícola y mejorar la calidad de los alimentos.
- Proyecto *Ríos Ciudadanos*: Es una iniciativa para concienciar y cuidar los ríos en Aragón, donde se involucra a los ciudadanos en el monitoreo de la calidad del agua de los ríos utilizando kits de evaluación de bajo costo. Esta iniciativa fomenta la conciencia ambiental y promueve el cuidado del medio ambiente.
- Agricultura de precisión y riego automático en cultivos orgánicos: Se trata de un proyecto realizado en Cerdeña, donde se utilizan sensores IoT para monitorear y controlar el riego automático y otros aspectos clave de la plantación. Esta tecnología ayuda a minimizar el impacto ambiental y mejorar la calidad de los cultivos. Estos casos ejemplifican cómo el IoT puede contribuir a la eficiencia, la sostenibilidad y la mejora de la calidad en diferentes industrias.

Por otro lado, el estudio presentado en el artículo [11] se enfoca en el diseño e implementación de un sistema de monitoreo de calidad del agua para acuicultura basado en tecnología de NB-IoT. El sistema utiliza energía solar y almacenamiento de batería como fuente principal de energía y recopila datos de

calidad del agua como oxígeno disuelto, pH, temperatura, turbidez y salinidad. Los resultados experimentales muestran que el sistema funciona de manera continua y estable sin perder el suministro de energía y logra una comunicación confiable con una tasa de pérdida de paquetes del 0,89 %. Este estudio destaca la importancia de utilizar tecnologías IoT licenciadas como NB-IoT para el monitoreo de calidad del agua.

En [12], se describe una prueba de campo agrícola utilizando tecnología de comunicación NB-IoT. El objetivo era desarrollar un sistema de automatización de fertilización e irrigación basado en la detección de parámetros ambientales para optimizar el crecimiento de los cultivos. Los resultados muestran que NB-IoT proporciona una cobertura extendida y es adecuado para aplicaciones con poca sensibilidad a la latencia. El estudio utiliza Raspberry Pi y módulos NB-IoT para la comunicación y se evalúa el rendimiento de la señal y la tasa de pérdida de paquetes en diferentes escenarios de prueba.

Además, en [13] se revisan los protocolos de comunicación inalámbrica utilizados en aplicaciones de agricultura inteligente. Se comparan las propiedades y aplicaciones de los protocolos ZigBee, Wi-Fi, Sigfox, NB-IoT y LoRaWAN. Se destacan los requisitos específicos de las aplicaciones agrícolas inteligentes y se discuten los problemas relacionados con el costo, la fiabilidad y el hardware de los protocolos. Tales como Wi-Fi 6 que ofrece alto rendimiento y baja latencia. ZigBee que destaca por su bajo consumo de energía y alta velocidad de transferencia de datos y para las LPWAN, Sigfox se enfoca principalmente en minimizar el consumo energético, NB-IoT es una buena opción de tecnología de comunicación de largo alcance y LoRaWAN que es ideal para aplicaciones con baja tasa de transferencia de datos.

IV. Especificaciones del nodo

En esta sección se proporcionan las especificaciones del nodo que ha sido diseñado como parte de este trabajo, que consta de una variedad de sensores y componentes hardware, así como de software específico que permite el monitoreo en tiempo real del entorno y la generación de alertas en caso de situaciones anómalas. A continuación, se presentan de manera concisa los detalles tanto del hardware como del software desarrollados para el nodo.

A. Hardware

Este nodo se compone de varios sensores y componentes que desempeñan un papel fundamental en su funcionamiento y que se muestran en la Figura 1. A continuación, se muestra una breve descripción de los sensores y componentes hardware adicionales en la Tabla I.

En la Figura 2, se presenta el prototipo resultante del nodo desplegado en un entorno rural de CLM. La utilización de este tipo de hardware experimental agregó cierta complejidad ya que fue necesario realizar modificaciones para abordar problemas identificados y mejorar su funcionamiento siguiendo una metodología de mejora continua.

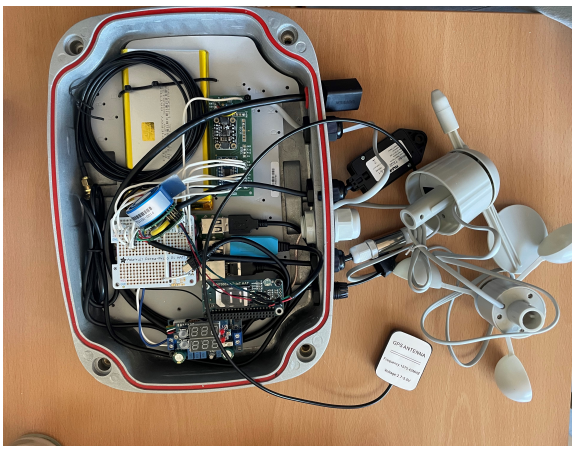


Fig. 1: Componentes del nodo.

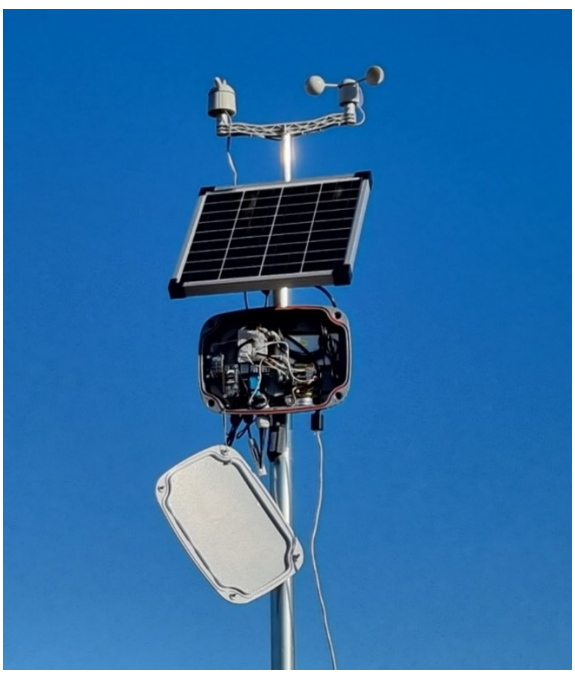


Fig. 2: Disposición final del nodo y despliegue en el entorno rural.

B. Software

Se implementó una API REST (Representational State Transfer), que se basa en una serie de restricciones arquitectónicas en lugar de ser un protocolo o estándar en sí mismo. La comunicación se realiza a través del protocolo HTTP y los datos se transmiten en formato *JavaScript Object Notation* (JSON). Las API REST son fáciles de utilizar, más rápidas y ligeras, y son ideales para el IoT.

Se utilizó Flask [27], un framework de Python considerado ”micro”debido a que proporciona las herramientas básicas para el desarrollo de aplicaciones web funcionales. Flask ofrece ventajas como proporcionar una estructura de proyecto consistente, facilitar la colaboración en equipo, ser compatible con Python3 y Web Server Gateway Interface (WSGI), ofrecer un manejo eficiente de rutas y ser versátil en

Tabla I: Resumen de los componentes hardware.

Componente	Descripción	Especificaciones
Raspberry Pi Model B+ [14]	Single Board Computer (SBC)	1.4 GHz y 1GB de RAM
SIM7000E-NB-IoT-HAT [15]	Radio con soporte para NB-IoT y GNSS	Interfaz UART
PiJuice [16]	Gestor de batería	Bajo consumo de energía
Solarmodul y PiJuice Battery [17], [18]	Panel solar y batería	Interfaz I_2C
Telaire T3022 Series [19]	Sensor de concentración de CO_2	Interfaz I_2C
SHT20 [20]	Sensor de temperatura y humedad del aire	Interfaz I_2C
ZE-11 [21]	Sensor de bencenos	Interfaz UART
MQ2 [22]	Sensor de humo y gases ajustables	Interfaz I_2C
Estación meteorológica [23]	Veleta y anemómetro	Adaptador RJ11
TXS0108E [24]	Conversor de nivel de voltaje	Convierte niveles de 3.3V a 5V
Adafruit ADS1015 [25]	Convertidor ADC	Interfaz I_2C
PL2303TA [26]	Módulo adaptador USB a puerto serie	Interfaz UART

tipos de aplicaciones. Flask es un proyecto de código abierto respaldado por una licencia *Berkeley Software Distribution* (BSD) y una comunidad activa de desarrolladores.

Cada nodo cuenta con su propio repositorio que incluye archivos de código o scripts desarrollados en Python3.9.2 para leer los datos de los sensores, configurar la batería y enviar los datos a un servidor. Estos scripts se configuran como servicios del sistema utilizando *Systemd* para asegurar una transmisión continua de datos. Esto significa que se definen como unidades de servicio y se integran en el sistema operativo de la SBC Raspberry Pi 3+, permitiendo su gestión y control de forma automatizada. *Systemd* se encarga de iniciar, detener y supervisar estos servicios, asegurando su correcto funcionamiento.

Se utilizó *Systemd* para ejecutar los scripts de Python de todos los sensores como servicios en segundo plano en el sistema Linux. Se configuró cada servicio en un archivo de servicio *.service* con opciones como el tipo de proceso, el usuario, el comando de inicio, la configuración de reinicio y el tiempo máximo de inicio [28]. A continuación, se presenta un ejemplo de la configuración de un servicio en *Systemd* para su ejecución en un sistema Raspbian, que es una distribución de Linux optimizada para Raspberry Pi.

```
[Unit]
Description=FlaskService

[Install]
WantedBy=multi-user.target

[Service]
Type=simple
User=camtima
PermissionsStartOnly=true
ExecStart=/usr/bin/python3 /home/camtima/API_REST/
    src/app.py
Restart=on-failure
TimeoutSec=600
```

Todos los demás servicios serían similares en términos de funcionamiento, con la única diferencia de que los demás scripts deben iniciarse después de que Flask haya comenzado a ejecutarse para poder enviarle los datos. Por lo tanto, es necesario incluir la opción `Requires=flask.service`.

Node-RED, una herramienta de programación con una interfaz visual, se utilizó para visualizar el flujo de información y la representación gráfica de los nodos [29]. Se configuró el modo *siempre encendido* en Node-RED para asegurar que los nodos estén siempre disponibles y recopilando datos. Asimismo, se configuró una opción para apagar el nodo cuando sea necesario. Los datos recibidos se visualizan en el dashboard de Node-RED.

El diagrama de flujo representado en la Figura 3 ilustra el proceso que el nodo debe seguir para leer los datos. Es importante mencionar que la Plataforma Multimodal (PM) utilizada en este sistema es propiedad de Vodafone y se encuentra fuera del alcance de esta investigación.

Para comenzar, se realiza una inicialización del dispositivo. A continuación, se procede a la lectura de los datos provenientes de los sensores, seguido de su procesamiento. Una vez que los datos son procesados, se almacenan en la memoria interna del nodo y se envían a la PM si es necesario, teniendo en cuenta consideraciones de consumo de energía. Es importante destacar que la transmisión y recepción de datos constituyen la principal fuente de consumo de energía en una WSN.

El nodo también puede recibir mensajes de la PM para configurar y ajustar su comportamiento. Con el objetivo de optimizar el consumo energético, el nodo puede entrar en un estado de inactividad prolongada y posteriormente despertar después de un período de tiempo preestablecido.

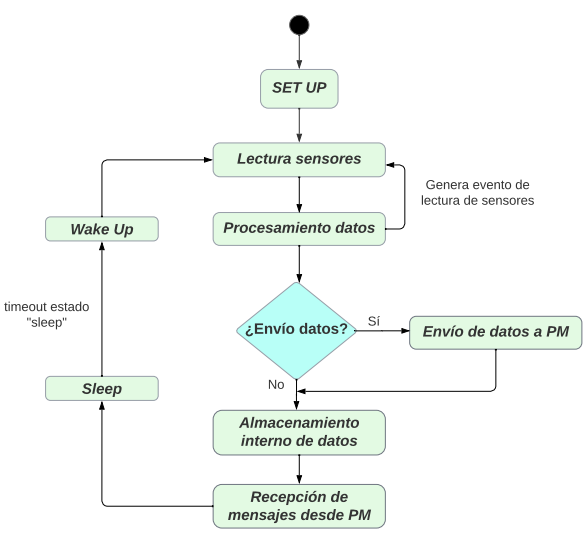


Fig. 3: Diagrama de flujo del funcionamiento del nodo.

V. Resultados experimentales

En la Figura 4 se presenta el esquema de la arquitectura del sistema completo, el cual se encarga de la recolección y envío de datos utilizando el módulo NB-IoT. El enfoque principal de este trabajo se ha centrado en garantizar el adecuado funcionamiento de esta tecnología y asegurar la transmisión confiable de los datos recopilados.

Una vez que los datos fueron enviados exitosamente a través del chip de radio NB-IoT instalado en el nodo, se alcanzó la plataforma de gestión (PM) a través de Internet. Es importante destacar que en esta etapa no hubo intervención directa en la gestión y operación de la PM.

El enfoque de este trabajo se centró en la recolección de datos y en asegurar su correcto envío mediante el módulo NB-IoT. Esta labor fue de gran importancia, ya que permitió verificar y evaluar el rendimiento del módulo NB-IoT al comprobar que los datos llegaban adecuadamente a la PM.

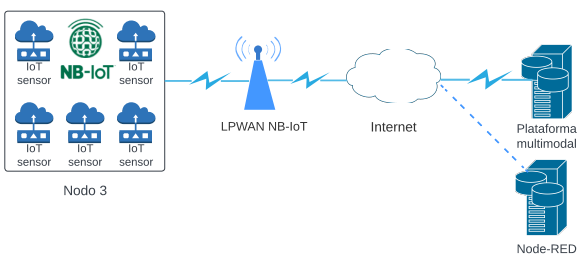


Fig. 4: Esquema general de sensórica medioambiental.

Las pruebas unitarias son fundamentales en la evaluación y validación de los sensores utilizados en el sistema (ver Tabla I). En este contexto, se realizaron pruebas exhaustivas para verificar el funcionamiento de cada componente. A continuación, se describen estas pruebas unitarias:

- Sensor de benceno: Para el sensor de benceno (ZE-11), se realizaron pruebas conectándolo a una placa de prototipado o protoboard y programándolo para recibir y enviar datos a través de una comunicación *Universal Asynchronous Receiver-Transmitter* (UART). Se verificó su funcionamiento acercando un recipiente con gasoil y observando el aumento de los valores de las mediciones.
- Sensor de CO_2: Para el sensor de CO_2 (T3022), se llevaron a cabo pruebas de conexión y comunicación utilizando el protocolo *Inter-Integrated Circuit* (I_2C). Se compararon las mediciones obtenidas por el sensor con las de un medidor de CO_2 comercial [30] previamente calibrado.
- Estación meteorológica: En este caso, se empleó el componente ADS1015 para convertir las señales analógicas de la veleta en valores digitales interpretables por la Raspberry Pi. Asimismo, se utilizó el componente TXS0108E para adaptar el nivel de voltaje entre el conversor analógico/-digital (ADC) y la Raspberry Pi. Por otro lado, el anemómetro utilizó un sensor electromecánico que cerraba un contacto magnético, generando pulsos que representaban la velocidad del viento. Cada ciclo de cierre en un segundo se registraba como una velocidad de 2,4 Km/h. Dado que el sensor actuaba como un interruptor y generaba pulsos, no fue necesario utilizar un ADC para conectarlo directamente a la Raspberry Pi.
- Sensor de humo: Durante las pruebas unitarias del sensor de humo (MQ2), se conectó al ADC (ADS1015) y se estableció una conexión con el TXS0108E para adaptar los niveles de voltaje de las señales de comunicación. Al utilizar una vela encendida, se comprobó que los valores de las mediciones aumentaban al acercar el sensor al humo que desprendía la vela y disminuían al alejarlo, confirmando la generación de eventos ante cambios en el nivel de humo en el ambiente.
- Sensor de temperatura y humedad: Durante las pruebas del sensor de temperatura y humedad (SHT20), se establecieron conexiones con la Raspberry Pi. El sensor fue alimentado con una tensión de 3.3 V proveniente de la Raspberry Pi para garantizar su correcto funcionamiento. Además, utilizando el protocolo I_2C, la Raspberry Pi pudo enviar señales de control al sensor y recibir lecturas de temperatura y humedad. Se llevó a cabo una prueba consistente en tapar el sensor con la mano, lo cual resultó en un incremento en las lecturas. Posteriormente, al destaparlo, las lecturas regresaron a su estado original, lo que evidenció el correcto funcionamiento del sensor en cuanto a su capacidad de detectar cambios en su entorno.

Después de completar la realización de las pruebas unitarias de cada sensor de forma individual, se procedió a realizar las pruebas de integración, donde todos los sensores fueron probados simultáneamente en la placa de prototipado. El objetivo de estas pruebas fue simular el ambiente en el que operará el sistema completo y verificar el correcto funcionamiento de los sensores en conjunto.

Durante el proceso de desarrollo, se implementó gradualmente la incorporación de los componentes al sistema. En primer lugar, se realizaron pruebas exitosas con la veleta y el anemómetro, los cuales operaron de manera conjunta sin presentar inconvenientes. Posteriormente, se incorporaron el sensor de CO_2 y el sensor de humedad y temperatura, los cuales utilizan el protocolo I_2C para la comunicación. En un principio, se intentó conectar estos sensores en el mismo bus de comunicaciones que la veleta, pero surgieron conflictos que afectaron su funcionamiento. Como solución, fue necesario utilizar buses separados para cada sensor, lo que aseguró un correcto funcionamiento aislado y evitó interferencias.

Además, se agregó el sensor de humo, el cual utiliza la librería `board` para detectar automáticamente el hardware de la Raspberry Pi y se comunica a través del bus 1. También se incorporó el sensor de benceno, que se conectó a la Raspberry Pi mediante un puerto USB. Durante la conexión de este sensor, se encontraron dificultades para identificar el puer-

to *tty* correspondiente, pero se lograron resolver y se verificó el correcto funcionamiento del conjunto. De manera similar, se realizó la conexión del módulo NB-IoT mediante otro de los puertos USB de la Raspberry Pi.

Por último, se procedió a verificar el funcionamiento conjunto del gestor de batería PiJuice y el sensor de humo en el bus 1. Afortunadamente, no se encontraron problemas de compatibilidad ni conflictos al compartir este bus, lo que permitió que ambos componentes operaran de manera óptima.

Estas pruebas de integración permitieron evaluar el rendimiento general del sistema y detectar posibles problemas de interoperabilidad entre los sensores. La verificación de las conexiones y la alimentación de los componentes fue fundamental para garantizar el correcto funcionamiento del sistema en su conjunto. La detección temprana de cualquier error o conflicto durante las pruebas de integración contribuyó a mejorar la calidad y la fiabilidad del sistema en general, lo que resulta crucial en un contexto científico donde se busca obtener mediciones precisas y confiables.

En la fase de pruebas de funcionamiento, se llevaron a cabo pruebas exhaustivas para verificar el correcto funcionamiento de los sensores en el nodo, utilizando el montaje final del sistema. Se llevaron a cabo 4 pruebas en el exterior del I3A después de haber realizado también pruebas al aire libre con el nodo en la terraza de este edificio.

Antes de realizar las pruebas 1 y 2 en el exterior del I3A, se registró una temperatura ambiente de 15ºC y una humedad del 65 %. El sensor de CO_2 mostró una lectura inicial (una vez transcurrido el período de calentamiento y estabilización) de entre 400 y 500 partes por millón (ppm), lo cual indica la concentración de dióxido de carbono en el ambiente. Por otro lado, el sensor de humo y el sensor de benceno registraron una lectura inicial de 0 mV y 0 ppm respectivamente, lo que significa que no se detectó presencia de humo ni de bencenos peligrosos en ese momento. En cuanto al sensor de humedad, su primera medición se correspondió con la humedad ambiente y el sensor de temperatura indicó una lectura de 15ºC, reflejando la temperatura ambiente en ese momento. Estos valores iniciales se utilizaron como referencia antes de llevar a cabo los experimentos.

A. *Prueba 1: Recogida de datos de sensores sin acelerantes*

Para verificar el funcionamiento de los sensores en conjunto, se realizó un experimento real en el que se acercó el nodo a un cubo de metal que contenía madera en llamas (ver Figura 5). Durante esta prueba, se pudo observar que el sensor de temperatura y humedad del nodo empezaba a registrar valores altos de temperatura y bajos de humedad, tal y como se esperaba en presencia de fuego cercano. Además, el sensor de CO_2 registró un aumento en sus ppm, indicando la presencia de bencenos emitidos por la combustión. El sensor de humo también monitorizó valores más altos.

Durante los experimentos realizados, se recopilaron numerosas mediciones de CO_2, humedad, temperatura, bencenos y humo. Sin embargo, para simplificar la presentación de los resultados y resaltar los puntos clave, se han seleccionado solo algunas muestras representativas de la totalidad de los datos recopilados (ver Figuras 6 y 7).

Fig. 5: Experimento llevado a cabo sin acelerantes (izquierda) y con acelerantes (derecha).

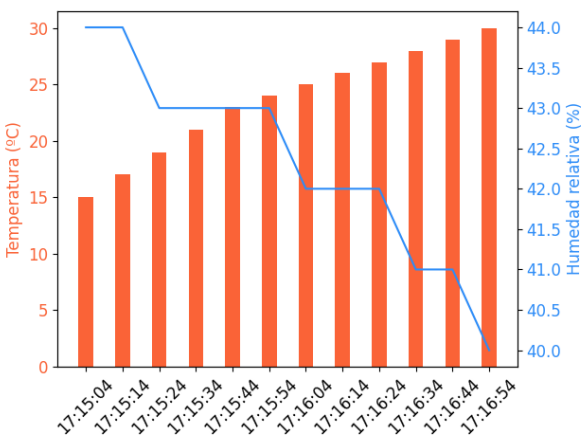


Fig. 6: Temperatura y humedad relativas en la prueba 1.

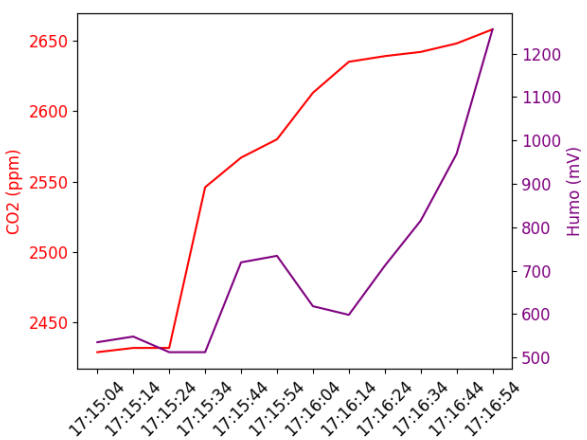


Fig. 7: Valores de CO_2 y humo en la prueba 1.

Sin embargo, durante esta primera prueba, el sensor de benceno del nodo no mostró cambios significativos en sus valores y devolvió un valor de 0 ppm.

B. *Prueba 2: Recogida de datos de sensores con acelerantes*

Para validar el correcto funcionamiento del sensor de benceno, se llevó a cabo una segunda prueba en la que se agregó un trapo impregnado con gasoil (que es un acelerante) al cubo de metal con maderas en llamas (ver Figura 5).

Durante esta segunda prueba, los sensores de temperatura, humedad, CO_2 y humo volvieron a mostrar los mismos valores que en la primera prueba. Sin embargo, esta vez el sensor de benceno del nodo devolvió valores significativos, lo que validó el funcionamiento apropiado del sensor para detección de acelerantes en incendios (ver Figura 8).

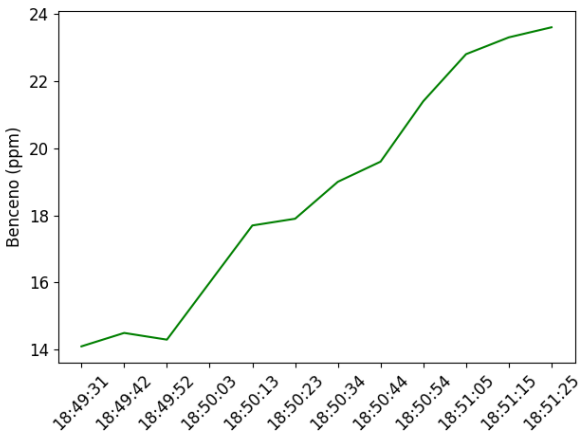


Fig. 8: Valores de benceno en la Prueba 2.

La realización de estos experimentos permitió no solo comprobar el correcto funcionamiento de los sensores de los nodos para la detección de incendios, sino también definir umbrales para la generación de alarmas asociados a cada sensor específico. Después de realizar las pruebas correspondientes, se establecieron los siguientes umbrales para garantizar una detección temprana y precisa de incendios: el sensor de benceno se configuró con un umbral de 15 ppm, el sensor de humedad se fijó con un umbral del 20 %, y el sensor de CO_2 se ajustó con un umbral de 3000 ppm. Estos valores límite fueron determinados en base a los resultados obtenidos durante las pruebas y permiten activar las alarmas en caso de detectarse un incendio. Es importante destacar que, para los umbrales asociados a humo y otros gases, se requiere un estudio más exhaustivo debido a la falta de estabilidad en los datos proporcionados por los sensores, a fin de establecer límites confiables y precisos. Estos umbrales permiten garantizar una detección temprana y precisa de incendios, lo cual desempeña un papel fundamental en la prevención de posibles daños significativos y la salvaguarda de vidas humanas.

C. *Pruebas de comunicación con NB-IoT*

El objetivo de este experimento era la validación del funcionamiento del módulo NB-IoT (SIM7000E-NB-IoT-HAT) que se muestra en la Figura 9 para la transmisión de datos en bruto de los sensores en caso de generarse alarmas asociadas a incendios.

Se realizaron los siguientes pasos:

Fig. 9: Módulo SIM7000E NB-IoT HAT.

1. Se procedió a insertar la tarjeta SIM en la ranura correspondiente y se estableció la conexión de la antena *Global System For Mobile Comunication* (GSM).
2. A continuación, se conectó la interfaz USB del SIM7000E NB-IoT HAT a un ordenador mediante un cable micro USB. Tras realizar la conexión, se observó que el indicador luminoso comenzó a emitir luz. Inicialmente, el indicador parpadeaba rápidamente (una vez por segundo), indicando que el módulo aún no se había registrado en la red. Sin embargo, una vez que el registro se completó, el indicador comenzó a parpadear de forma más lenta, a una frecuencia de una vez cada tres segundos.
3. Se confirmó la correcta conexión de la antena y se verificó el funcionamiento adecuado de la tarjeta SIM, asegurándose de que estuviera correctamente insertada y operativa.
4. Luego, se instaló el controlador SIM7000 y se identificó el número de puerto COM correspondiente al módulo SIM7000E.
5. Para verificar el correcto funcionamiento del módulo NB-IoT, se llevaron a cabo pruebas utilizando comandos AT (Attention). Los comandos AT son un conjunto de comandos estándar utilizados para controlar y comunicarse con dispositivos a través de una interfaz serie. Estos comandos permiten realizar diversas operaciones, como establecer conexiones, enviar y recibir datos, entre otras acciones. El manual suministrado por el fabricante proporciona una amplia variedad de comandos AT que se pueden utilizar para configurar y realizar diversas operaciones en el módulo [31]. Es importante destacar que los comandos AT no son exclusivos de NB-IoT, sino que se utilizan en diferentes dispositivos.
   Por ejemplo, se utilizó el comando `AT+CSQ` para medir el nivel de señal. Utilizando una serie de comandos AT similares, se configuró el módulo NB-IoT de la siguiente manera:
   `AT+CREG=1`: Activa el registro en la red.
   `AT+CREG?`: Verifica el estado del registro en la red (registrado / no registrado).
   `AT+COPS=?`: Obtiene la lista de operadores disponibles.
   `AT+COPS=`: Selecciona un operador.
   `AT+COPS?`: Verifica la red actual.
   `AT+CGATT=1`: Realiza el registro mediante GPRS.
   `AT+CGATT?`: Obtiene el estado actual de la conexión GPRS.

`AT+CGDCONT=1,IP,em`: Define el contexto Packet Data Protocol (PDP).
`AT+CSQ`: Mide la calidad de la señal.
`AT+CGACT=1`: Activa el contexto PDP.
`AT+CGDCONT?`: Obtiene el estado actual de la sesión de datos.

De esta manera, se logró conectar el módulo NB-IoT a la red. Durante la fase de despliegue del nodo en el campo, se utilizó un script de Python para enviar estos comandos de forma programada en lugar de realizarlos manualmente, como se hizo durante las pruebas para verificar el funcionamiento del módulo.

Los desarrollos llevados a cabo en cuanto a la conexión con el módulo NB-IoT se basaron en un proyecto existente en GitHub para simplificar la conexión con el módulo NB-IoT. El proyecto seleccionado se encuentra en el repositorio de GitHub llamado `maxis-nbiot-hackathon` y ha sido desarrollado por Cytron Technologies [32]. Una vez instalado, fue necesario verificar que la ruta por defecto (0.0.0.0 0.0.0.0) utilizara la interfaz ppp0. Si al realizar una prueba de conectividad con Internet se recibían respuestas de los paquetes enviados, esto indicó que el módulo SIM7000E estaba funcionando correctamente como un módem y que se pudo establecer una conexión exitosa con Internet. En este escenario, el sistema operativo se encargó de la gestión del módem, liberándonos de esa responsabilidad. Cabe destacar que este proyecto también ofreció funcionalidad para gestionar el GPS.

D. Pruebas de los nodos desplegados en la finca

Se realizaron pruebas de campo en un entorno real en la finca *El Palomar*, ubicada en Alcaraz (Albacete, España). Aunque los resultados detallados de estas pruebas no se abordan en este trabajo, se presentan en la subsección V-E algunos de los desafíos encontrados durante dichas pruebas.

Para permitir el despliegue de los nodos en el campo, se requirió configurar direcciones IP estáticas, ya que no había un enrutador disponible que pudiera asignar direcciones IP dinámicamente. La configuración de la dirección IP estática en Raspbian se realizó mediante la edición del archivo `/etc/dhcpcd.conf` utilizando el comando `sudo nano /etc/dhcpcd.conf`. Para establecer una dirección IP estática, se realizaron las modificaciones mostradas en el siguiente fragmento de código:

```
Configuracion de IP estatica:
interface eth0

static ip_address=192.168.1.100/24

#static ip6_address=fd51:42f8:caae:d92e::ff/64

#static routers=192.168.0.1

#static domain_name_servers=192.168.0.1 8.8.8.8
     fd51:42f8:caae:d92e::1
```

A continuación se detallan los parámetros a tener en cuenta:

- `interface`: Nombre de la interfaz que se desea configurar, en este caso, eth0.
- `static ip_address`: Dirección IP fija que se desea asignar (manteniendo la máscara al final).
- `static ip6_address`: Este parámetro hace referencia a la configuración de una dirección IPv6 estática. Sin embargo, en el presente caso, donde el uso de IPv6 no está involucrado, se concluye que este parámetro carece de relevancia.
- `static routers`: Esta opción se utiliza para especificar la dirección del enrutador o gateway. En el contexto específico en el que nos encontramos, este parámetro no es necesario.
- `static domain_name_servers`: Dirección del servidor DNS (por ejemplo, la del enrutador o servidores externos como 8.8.8.8 de Google). Si se desea utilizar varios servidores DNS, se pueden añadir separados por espacios [33]. Sin embargo, en este escenario de campo, no hay un enrutador disponible, por lo que este parámetro no es relevante.
- Una vez realizados los cambios, se guardó el archivo y se reinició la Raspberry Pi. Para verificar la correcta aplicación de la dirección IP configurada, se ejecutó el comando `ifconfig eth0`.

El propósito de configurar una dirección IP estática y establecer una conexión directa con la Raspberry Pi a través de un cable Ethernet es permitir la comunicación con la Raspberry Pi en un entorno de campo, ya que los datos recolectados por los sensores se envían a través de la interfaz ppp0 que crea el módulo NB-IoT. Además, también es necesario cambiar el adaptador de red del ordenador que se va a conectar al nodo.

E. Incidencias y lecciones aprendidas

A continuación, se exponen las principales dificultades encontradas durante el desarrollo del nodo, las cuales, a partir de la validación final realizada en un entorno rural, han permitido extraer valiosas lecciones aprendidas para la comunidad investigadora.

- Problemas con la disponibilidad del hardware necesario en el mercado debido a los problemas de comercio internacional: La escasez de material condujo a la necesidad de cambiar a una plataforma de hardware con un mayor consumo energético, lo que resultó en una notable reducción del tiempo de operación y la autonomía de los nodos.
- Conectividad al Bus I_2C: Algunos sensores requerían un bus I_2C exclusivo, lo que generó conflictos cuando múltiples dispositivos I_2C necesitaban comunicarse en el mismo bus. Esto causó colisiones de datos y respuestas incorrectas.
- Problema con las rutas en los servicios: La configuración de servicios en la ruta `/etc/systemd/system` requería rutas absolutas en lugar de rutas relativas, lo que resultaba en errores al acceder a los archivos necesarios.
- Pérdida de conexión: Se experimentó pérdida de conexión con el sensor de temperatura y humedad (STH20) después de un periodo prolongado

de lectura. Fue necesario implementar un manejo de excepciones para controlar esta situación.
- Velocidades de lectura variables: Cada sensor opera a una velocidad de comunicación (baudrate) distinta, lo que implica la necesidad de incorporar pausas (sleep) adecuadas entre las lecturas para evitar inconvenientes al intentar acceder a valores no disponibles.
- Alimentación del sensor de CO_2: Se encontraron dificultades con el sensor de CO_2 TELAIRE T3022 SERIES al ser alimentado con 5V utilizando una Raspberry Pi. Solo se obtuvo un funcionamiento correcto al proporcionar una tensión de entrada de 3,3V, lo cual contrasta con las indicaciones del datasheet del fabricante.
- Relojes internos: La discrepancia de tiempo entre el sistema operativo de la Raspberry Pi y el reloj RTC de Pijuice causaba problemas en aplicaciones que dependían del tiempo. Por tanto, fue necesario sincronizar la hora del sistema operativo con el reloj RTC de Pijuice.
- Inconvenientes relacionados con la cobertura del estándar NB-IoT: Se identificaron dificultades de cobertura NB-IoT durante la implementación en campo, lo cual tenía un impacto en el tiempo de notificación y la autonomía de los nodos.
- Sensor de humo: El sensor de humo MQ2 no cumplió con las especificaciones mostradas en la hoja de características del fabricante, lo que dificultó su calibración bajo un criterio unificado.

Las lecciones aprendidas a partir de las incidencias mencionadas son las siguientes:

- Planificación de adquisición de material: Se recomienda realizar los pedidos con suficiente antelación para evitar posibles retrasos debido a la escasez de suministros.
- Resolución del conflicto en el bus I_2C: Se implementó la creación de nuevos buses I_2C de software para conectar cada sensor de forma independiente, evitando conflictos de comunicación.
- Uso de rutas absolutas en la configuración de servicios: Es esencial especificar rutas absolutas en la configuración de los servicios para garantizar un acceso adecuado a los archivos necesarios.
- Sincronización precisa de los relojes internos: Se ha llevado a cabo una sincronización precisa entre el reloj del sistema operativo y el reloj RTC de Pijuice para evitar discrepancias de tiempo.
- Calibración personalizada de los sensores: Cada tipo de sensor requiere una calibración específica, realizada en entornos similares a la ubicación final de los nodos, con el fin de obtener mediciones precisas y confiables.
- Mejora de los requisitos iniciales: Se ha llevado a cabo una optimización de los requisitos iniciales al aumentar el tamaño de la batería y el panel solar, con el fin de garantizar una mayor autonomía de los nodos y asegurar la recopilación continua de datos.

VI. Conclusiones y trabajos futuros

La implementación de nodos de IoT basados en la tecnología NB-IoT para la detección y prevención de incendios forestales brinda una serie de beneficios significativos. Estos incluyen la capacidad de proporcionar conectividad confiable en áreas remotas, la eficiencia energética que prolonga la duración de la batería de los dispositivos, la comunicación bidireccional para la transmisión de alertas y comandos remotos, así como la optimización de recursos y la reducción de costos.

En el presente trabajo se ha llevado a cabo el desarrollo de un nodo IoT equipado con sensores de bajo costo para la detección de incendios forestales y el monitoreo medioambiental. El enfoque principal se ha centrado en el diseño, integración y puesta en marcha de los sensores, así como en las pruebas de comunicación con NB-IoT para la transmisión en tiempo real de los datos recogidos. Después del desarrollo y evaluación del nodo, se han compartido las lecciones aprendidas a partir de los experimentos realizados con hardware real.

Estas tecnologías permiten el monitoreo ambiental en tiempo real y la generación de alertas en caso de condiciones propicias para incendios forestales. Al utilizar redes celulares existentes, la tecnología NB-IoT asegura una cobertura amplia y estable, incluso en áreas rurales y de difícil acceso. Esto permite la implementación estratégica de nodos de IoT en ubicaciones clave para detectar y prevenir incendios forestales de manera oportuna.

Como trabajo futuro se propone ampliar las investigaciones experimentales llevadas a cabo en el proyecto que sirvió como marco de referencia para realizar este trabajo y continuar trabajando en la mejora de estándares de red para aplicaciones medioambientales. Más concretamente, en los nuevos estándares mMTC (massive Machine-Type Communication) enmarcados en el desarrollo de las redes más allá del 5G (Beyond 5G) que supondrán toda una revolución para el envío de información continua en áreas del entorno rural.

Agradecimientos

Al Instituto de Investigación en Informática de Albacete (I3A). A la ayuda 2023-GRIN-34056 financiada por la Universidad de Castilla-La Mancha. Al Proyecto PID2021-123627OB-C52, financiado por MCIN/AEI/10.13039/501100011033, FEDER (UE). A los Contratos de I+D 220062UCTR Tecnologías de Sensórica para el Medio Ambiente y 220066UCTR Tecnologías de sensórica para el controlo cinegético, del proyecto CDTI CAMTIMA.

Referencias

[1] P Kanakaraja, P Syam Sundar, N Vaishnavi, S Gopal Krishna Reddy, and G Sai Manikanta, "IoT enabled advanced forest fire detecting and monitoring on Ubidots platform, journal = Materials Today: Proceedings," vol. 46, pp. 3907–3914, 2021, International Conference on Materials, Manufacturing and Mechanical Engineering for Sustainable Developments-2020 (ICMSD 2020).

[2] Fengmei Cui, "Deployment and integration of smart sensors with IoT devices detecting fire disasters in huge forest environment," *Computer Communications*, vol. 150, pp. 818–827, 2020, Accedido el 7/06/2023.

[3] Grupo TRC, "Desarrollo de Software IDi CAMTIMA," Disponible en: `https://www.grupotrc.com/TRCProductos-82-Desarrollo-de-software-IDi-Camtima.aspx`, 2023, Accedido el 6/04/2023.

[4] Vodafone, "Vodafone: Narrowband IoT," Disponible en: `https://www.vodafone.es/c/empresas/es/narrowband-iot/`, 2023, Accedido el 29/03/2023.

[5] David Sánchez Rosado, "Análisis práctico de las soluciones de Telefónica y Vodafone en tecnologías celulares Narrow-band NB-IoT," M.S. thesis, Universidad Complutense de Madrid, Septiembre 2019.

[6] Maria Luisa Machado González, "Estudio de NB-IoT y comparativa con otras tecnologías LPWAN," M.S. thesis, Universidad Oberta de Cataluña, Enero 2019.

[7] Silvia Hernández Caballero, "Estudio en detalle de LoRaWAN. Comparación con otras tecnologías LPWAN considerando diferentes patrones de tráfico," M.S. thesis, Universidad Oberta de Cataluña, Enero 2020.

[8] Javier Saiz Miranda, "Estudio en detalle de NB-IoT. Comparación con otras tecnologías LPWAN considerando diferentes patrones de tráfico," M.S. thesis, Universidad Oberta de Cataluña, Junio 2019.

[9] Rohde and Schwarz, "Internet de las cosas en banda estrecha," Disponible en: `https://www.rohde-schwarz.com/es/aplicaciones/internet-de-las-cosas-en-banda-estrecha-white-paper_230854-314242.html`, 2016, Accedido el 2/04/2023.

[10] Libelium, "Libelium Website," Disponible en: `https://www.libelium.com/`, 2023, Accedido el 4/06/2023.

[11] Chaowanan Jamroen, Nontanan Yonsiri, Thitiworada Odthon, Natthakun Wisitthiwong, and Sutawas Janreung, "A standalone photovoltaic/battery energy-powered water quality monitoring system based on narrowband internet of things for aquaculture: Design and implementation," *Smart Agricultural Technology*, vol. 3, pp. 100072, 2023.

[12] Giovanni Valecce, Pierpaolo Petruzzi, Sergio Strazzella, and Luigi Alfredo Grieco, "NB-IoT for Smart Agriculture: Experiments from the Field," in *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2020, vol. 1, pp. 71–75, Accedido el 7/06/2023.

[13] Ercan Avşar and Md Najmul Mowla, "Wireless communication protocols in smart agriculture: A review on applications, challenges and future trends," *Sensors and Actuators A: Physical*, vol. 341, pp. 111079, 2022, Accedido el 7/06/2023.

[14] Raspberry Pi Foundation, "Raspberry Pi 3 Model B+ Product Brief," 2023.

[15] WaveShare, "SIM7000E NB-IoT HAT," 2023.

[16] Ultra Lab, "PiJuice HAT: Plataforma de Alimentación Portátil para Raspberry Pi," 2023.

[17] Solarv.de, "Monokristalline Solarmodul 20W," 2023.

[18] Pi Supply, "PiJuice 12000mAh Battery," 2023.

[19] Amphenol Sensors, "CO2 Transmitters," 2023.

[20] DFRobot, "SHT20 I2C Temperature and Humidity Sensor (Waterproof Probe)," 2023.

[21] Winsen Electronics Technology Co., Ltd., "ZE11 Toxic Gas Sensor," 2023.

[22] Seeedstudio, "Grove - Gas Sensor (MQ2)," 2023.

[23] Argent Data Systems, "Argent Data Systems - APRS TinyTracker," 2023.

[24] Texas Instruments, "TXS0108E 8-Bit Bi-directional Level Shifter," Diciembre 2007.

[25] Texas Instruments, *ADS1015 12-Bit ADC with Programmable Gain Amplifier*, Texas Instruments Incorporated, Mayo 2009.

[26] Amazon, "Módulo adaptador PL2303TA USB a TTL RS232," 2020.

[27] Pallets Projects, "Flask Documentation," Disponible en: `https://flask.palletsprojects.com/en/2.2.x/`, 2023, Accedido el 12/04/2023.

[28] WasiUllah Khan, "Setup a Python Script as a Service through Systemctl (systemd)," Aug 2020, Accedido el 12/04/2023.

[29] Grupo Sinelec, "¿Qué es Node-RED y para qué sirve?," Disponible en: `https://blog.gruposinelec.com/actualidad/que-es-node-red-y-para-que-sirve/`, 2023, Accedido el 15/04/2023.

[30] Ahmad Mohammadshirazi, Vahid Ahmadi Kalkhorani, Joseph Humes, Benjamin Speno, Juliette Rike, Rajiv Ramnath, and Jordan D Clark, "Predicting airborne pollutant concentrations and events in a commercial building using low-cost pollutant sensors and machine learning: A case study," *Building and Environment*, vol. 213, pp. 108833, 2022.

[31] Waveshare, "SIM7000 Series AT Command Manual," Disponible en: `https://www.waveshare.com/w/upload/6/6a/SIM7000_Series_AT_Command_Manual_V1.03.pdf`, 2023, Accedido el 17/05/2023.

[32] Cytron Technologies, "maxis-NB-IoT-hackathon," Disponible en: `https://github.com/CytronTechnologies/maxis-nbiot-hackathon`, 2023, Accedido el 22/05/2023.

[33] Luis Llamas, "Raspberry Pi: Cómo configurar una IP estática," Disponible en: `https://www.luisllamas.es/raspberry-pi-ip-estatica/`, 2018, Accedido el 20/05/2023.

Proyecto RED-SEA: Resultados Intermedios

José Duro[1], Adrián Castelló[1], María E. Gómez[1], Julio Sahuquillo[1], Enrique Quintana[1], Gabriel Gómez[2], Miguel Sánchez[2], Jesús Escudero-Sahuquillo[2], Pedro J. García[2], Francisco J. Alfaro[2], José L. Sánchez[2], Francisco J. Quiles[2]

Resumen— **El objetivo general de RED-SEA es diseñar una nueva generación de red de interconexión europea, que posibilite la computación exascala en Europa, mediante una interconexión económicamente viable y tecnológicamente eficiente, aprovechando tecnología de interconexión europea (BXI) junto a la tecnología estándar y madura (Ethernet), iniciativas anteriores financiadas por la UE, como ExaNeSt, EuroEXA, ECOSCALE, Mont-Blanc, los proyectos DEEP y el proyecto de procesador europeo (EPI), así como estándares abiertos y API compatibles.**

Para alcanzar este objetivo global, el proyecto RED-SEA se desarrolla en torno a cuatro pilares fundamentales: i) arquitectura y codiseño - con el objetivo de optimizar el ajuste con los otros proyectos EuroHPC y con los procesadores EPI; ii) desarrollo de un bridge de altas prestaciones, baja latencia y sin fisuras con Ethernet iii) gestión de recursos de red, incluyendo congestión y calidad de servicio; y iv) funciones de extremo a extremo implementadas en la red.

Este artículo presenta los principales logros alcanzados a mitad del proyecto por los 2 socios españoles que participan en el proyecto, es decir, la Universitat Politècnica de València (UPV) y la Universidad de Castilla La-Mancha (UCLM), contribuyentes a los pilares 1 y 3. En este sentido, cabe destacar i) la definición de los requisitos de la red y la arquitectura de la red, una lista inicial de aplicaciones y el modelado de la arquitectura BXI3 para poder evaluar las prestaciones de las propuestas del proyecto; ii) la caracterización de la congestión de las aplicaciones y las propuestas para reducir esta congestión mediante la optimización de las primitivas de comunicación colectiva.

Palabras clave— **Redes de interconexión, HPC, congestión, datacenter, primitivas de comunicación colectiva, baja latencia.**

I. Introducción

Las redes de interconexión de próxima generación deben escalar para soportar sistemas de procesamiento masivamente paralelo (cientos de miles de nodos y millones de núcleos) y proporcionar un conjunto de características que permitan a las aplicaciones HPC, HPDA e IA alcanzar la computación exascala, beneficiándose al mismo tiempo de las nuevas tendencias de hardware y software. En este sentido, el sistema debe permitir que las aplicaciones escalen eficientemente, estar preparado para aceleradores y unidades de computación de bajo consumo, y soportar aplicaciones emergentes y generalizadas centradas en datos y relacionadas con la IA.

El consorcio RED-SEA reúne a los mejores centros académicos con las principales fuerzas industriales europeas en este ámbito. El consorcio RED-SEA persigue el objetivo mencionado, aprovechando las competencias y los antecedentes europeos clave, incluida BXI, que es una tecnología de red de interconexión europea en producción, así como los resultados de una serie de proyectos financiados por la UE sobre interconexiones y sistemas HPC.

RED-SEA está apoyando e impulsando la computación exascala y las tecnologías basadas en datos dentro de Europa mediante la ampliación y optimización de la tecnología de red BXI Exascale para anticiparse a los requisitos de los sistemas en el horizonte temporal 2022-2025. BXI versión 2 está actualmente en producción y aparece en los sistemas Top 500. El proyecto RED-SEA está poniendo en marcha la tercera generación de la interconexión BXI (conocidad como BXI3), contribuyendo a su hoja de ruta mediante: (i) la definición del proyecto de arquitectura y los modelos de simulación correspondientes; (ii) el diseño de los nuevos bloques de construcción (IP) necesarios para abordar los nuevos retos de los superordenadores modulares; (iii) la realización de una prueba de concepto inicial de sus componentes críticos en aplicaciones de la vida real; y (iv) el desarrollo del ecosistema y la creación de una comunidad más amplia de usuarios y desarrolladores que combine equipos de investigación e industriales.

El resto del artículo se organiza de la siguiente manera. La sección II resume el trabajo previo relacionado con el proyecto. En la sección III se presentan los socios que participan en RED-SEA. En la sección IV se discuten los principales retos que debemos afrontar para llevar a cabo el proyecto. En la sección V se describe el enfoque de RED-SEA para abordar estos retos. En la sección VI se describen las metodologías empleadas. En la Sección VII se presentan los paquetes de trabajo implementados en el proyecto para alcanzar su objetivo final, y en la Sección VIII los logros alcanzados hasta el momento en el proyecto por parte de los dos socios españoles. Finalmente, en el apartado IX se exponen las principales conclusiones del trabajo.

II. Estado del arte

En los últimos años se han hecho muchos intentos de implementar redes de alta velocidad, orientadas a la computación HPC, en sistemas basados en FPGA. En el marco del proyecto ExaNeSt [1] se diseñó, verificó y desplegó una interconexión de altas prestaciones y baja latencia basada en FPGA en un banco de pruebas de 128 SoCs Zynq Ultrascale+. ExaSNet es una interconexión de red jerárquica caracterizada por una topología todo a todo a nivel de nodo (formada por 4 FPGA interconectadas) y una red Torus 3D escalable para la interconexión entre nodos a nivel de bastidor. ExaNeSt integra una interfaz de red

[1]Universitat Politècnica de València, e-mail: e-mail: megomez@disca.upv.es

[2]Universidad de Castilla La-Mancha, e-mail: Jesus.Escudero@uclm.es

optimizada y embebida orientada a ARM, que ofrece múltiples canales de hardware para transferencias RDMA y packetizer-mailbox fiables a nivel de usuario [2], [**?**], y un bloque switch/router de alto rendimiento procedente de APEnet con 6 enlaces bidireccionales 3D Torus de hasta 32 Gb/s, en una única FPGA. En el mismo proyecto se ha implementado un rumtime MPI especial, y se han portado y ejecutado aplicaciones HPC en un banco de pruebas de 8 palas con 128 FPGAs (en 128 SoCs) o 512 núcleos ARMv8. EuroEXA [3] aprovecha ExaNeSt para impulsar el concepto de ïnterconexión escalable de topología híbrida"(*TriFeCta*) a escala extrema. EuroEXA diseñó una versión mejorada de la arquitectura ExaNet que proporciona diferentes topologías y características en los distintos niveles de la jerarquía de red. EuroEXA diseñó un innovador çonmutador personalizado"basado en una única FPGA Virtex Ultrascale+ que implementa una topología todo a todo de 2 saltos a nivel de placa, una red Torus 3D a nivel de bastidor y un puente ExaNet-Ethernet 100/200G para la conectividad entre bastidores.

III. Consorcio

El consorcio reúne a grupos de investigación bien establecidos en toda Europa, con una larga experiencia en redes de interconexión, incluido el diseño, despliegue y evaluación de redes. La UPV y la UCLM (España) han desarrollado técnicas punteras en casi todos los aspectos de las interconexiones, incluyendo topologías eficientes, estrategias de encaminamiento adaptativo sin bloqueo, control de flujo, provisión de QoS, técnicas de gestión de la congestión y estrategias de encaminamiento tolerantes a fallos. Además de estos socios españoles firmantes de este trabajo, participan otros socios que describimos a continuación. FORTH (Grecia) ha sido pionera en la gestión de la congestión y la regulación por flujo. ETH Zúrich (Suiza) aporta investigación puntera sobre redes escalables de altas prestaciones, programación paralela (destacando en MPI) y computación programable en red. INFN (Italia), el desarrollador de APEnet [4], tiene una larga experiencia en la creación de prototipos de sistemas y en códigos HPC eficientes. EXTOLL (Alemania), una empresa con sede en la UE, spin-off de la Universidad de Heidelberg, que ofrece IPs de baja latencia, de última generación y altas prestaciones para interconexiones HPC; ParTec (Alemania) desarrolla runtimes MPI para interconexiones, ExactLab optimiza códigos HPC y ExaPSYS es una joven startup con buenos conocimientos sobre simulaciones y sistemas heterogéneos. CEA (Francia), con actividades que abarcan desde el funcionamiento de grandes despliegues de HPC hasta nuevas tecnologías de hardware y software para sistemas e interconexiones. FZJ (Alemania) y CEA albergan dos de los mayores centros de supercomputación de Europa, y llevan a cabo investigación científica sobre códigos de computación de alto rendimiento. Atos (Francia) y CEA colaboran estrechamente en el programa nacional francés de exaescala. Por último, pero no por ello menos importante, el coordinador del proyecto, Atos/Bull, único fabricante europeo de ordenadores, cuenta con equipos de investigación y desarrollo experimentados que han producido una serie de interconexiones y sistemas comerciales on-chip y off-chip.

IV. Retos

A continuación se enumeran los principales retos para lograr las prestaciones y la compatibilidad deseados en RED-SEA:

- *Escalabilidad, fiabilidad*: Demostrar formas de escalar las interconexiones más allá de 100 K nodos, cumpliendo al mismo tiempo los objetivos clave de prestaciones y fiabilidad, y satisfaciendo diversos requisitos, desde bibliotecas de comunicación (MPI) e IA hasta aplicaciones centradas en datos.
- *Convergencia HPC/datacenter*: desarrollar y demostrar métodos a nivel de producto que integren de forma óptima IP y el tráfico Ethernet y RoCE (RDMA sobre Ethernet convergente) en una interconexión HPC, consiguiendo una baja latencia y altas velocidades de mensajes.
- *Throughput, ancho de banda*: Multiplicar por 4 el ancho de banda y la tasa de mensajes disponibles para cada punto final de la red duplicando la frecuencia del enlace (hasta 200 Gbps) y duplicando el número de interfaz de red para cada proceso (multi-rail).
- *Calidad del servicio*: Desarrollar nuevos algoritmos de control de la congestión y mecanismos de provisión de QoS adecuados para entornos HPC ágiles centrados en datos, evaluarlos en plataformas y modelos de simulación escalables, y esbozar su camino hacia el producto de interconexión.
- *Programabilidad, latencia*: Desarrollar formas de configurar mediante programación el motor de descarga de red, permitiendo también el cómputo en red y consiguiendo una mejor latencia/eficiencia energética.
- *Nuevos procesadores, relación con EPI*: Demostrar la interoperabilidad de la interconexión diseñada con componentes de la Iniciativa Europea de Procesadores, como procesadores y aceleradores Arm y RISC-V, y definir arquitecturas de red alternativas para los sistemas europeos Exascale.
- *Protección, compartición*: Demostrar métodos para la partición de un sistema HPC existente, clúster en múltiples nubes (privadas), manteniendo la protección, la seguridad y el aislamiento.
- *Ir al mercado / impacto*: Definir una trayectoria de salida al mercado y optimizar nuestras posibilidades de tener una parte importante de estas IP y resultados europeos utilizados en los principales sistemas europeos en el horizonte 2022-23, al tiempo que reforzamos nuestras posiciones actuales en el segmento de mercado de la interconexión.

V. La aproximación RED-SEA

Los sistemas HPC y los orientados a datos de próxima generación serán heterogéneos en cuanto a los dispositivos que utilizarán, incluidos procesadores ARM y RISC-V de bajo consumo, CPU de gama alta, unidades de aceleración vectorial y GPU adecuadas para cargas de trabajo masivas de una sola instrucción y múltiples datos (SIMD), así como diseños FPGA y ASIC adaptados para códigos personalizados de consumo extremadamente eficiente [5]. Las modernas unidades de procesamiento paralelo de datos, como las GPU y los aceleradores vectoriales, pueden procesar datos a velocidades asombrosas (decenas de TFLOPS). En este panorama, la red se está convirtiendo en el próximo gran cuello de botella. El consorcio RED-SEA está abordando estos retos mediante:

- Ethernet de altas prestaciones como red de federación con semántica de comunicación RDMA de baja latencia de última generación.
- BXI como estructura HPC, que consta de dos componentes discretos: una NIC BXI, un conmutador BXI y el gestor de estructura BXI. La tercera generación de BXI añade nuevas funciones y aumentará sus prestaciones para alcanzar los objetivos enumerados.

A. RED-SEA Capa Física

El continuo aumento del ancho de banda conduce a enlaces serie cada vez más sofisticados. El proyecto se centra en enlaces de 200 Gb/s por dirección, formados por cuatro carriles diferenciales independientes que funcionan a 56 Gb/s. El proyecto se centra en el desarrollo de IP modulares MAC y PCS que puedan reutilizarse tanto para enlaces Ethernet como para futuros enlaces BXI.

B. RED-SEA Capa de Transporte

Instalar sistemas de producción que cuenten con más de 100 K nodos ya es un reto, pero escalar las prestaciones es el verdadero requisito. Las prestaciones globales dependen directamente del comportamiento de la red. Los requisitos de fiabilidad siguen a la explosión del número nodos y el mecanismo de fiabilidad extremo a extremo debe diseñarse para soportar simultáneamente hasta 100 K nodos y mantener las prestaciones de 200 Gb/s para cada enlace. El proyecto diseña un IP de fiabilidad E2E que proporcione un mecanismo de recuperación para fallos transitorios y permanentes que garantice la integridad de los mensajes, el orden de los mensajes y la entrega de los mensajes.

C. Interfaz de host RED-SEA

El proyecto tiene como objetivo una reducción muy agresiva de la latencia entre los procesadores de host y la red. Este objetivo se lleva a cabo de dos formas principales. El cambio más disruptivo consiste en eliminar la interfaz PCIe estándar y disponer de un acceso directo a los núcleos de procesador de bajo consumo a través de una interfaz coherente para reducir la latencia y simplificar la interfaz de software.

D. Entorno de software de RED-SEA

El proyecto tiene como objetivo desarrollar la pila de software y las bibliotecas para aprovechar las capacidades de descarga de BXI, como las operaciones colectivas de altas prestaciones. La red BXI se basa en la API Portals 4. Portals 4 [6] es un estándar desarrollado en colaboración por Sandia National Labs y por la Universidad de Nuevo México. Se eligió porque es la única interfaz disponible que soporta tanto MPI como PGAS, a la vez que proporciona bloques de construcción apropiados para las comunicaciones de software del sistema (E/S paralela, lanzamiento de trabajos).

E. RED-SEA Congestión y QoS

Abordar la congestión de la red es clave para proporcionar una gestión eficiente de los recursos de red. RED-SEA propone, implementa y evalúa nuevos mecanismos de gestión de la congestión para la tecnología de red BXI3, como se muestra en la Figura 1. En este punto es donde la UPV y la UCLM están haciendo las mayores aportaciones.

En primer lugar, en el contexto de este proyecto con componentes de computación y datos extremos, las aplicaciones paralelas generán una gran congestión debido a las operaciones colectivas como presentaremos en este trabajo posteriormente. En el proyecto abordamos la optimización de primitivas de comunicación colectiva clave.

Otras técnicas previstas para hacer frente a la congestión son: encaminamiento adaptativo de grano fino y medio, gestión de la congestión inteligente y con capacidad de respuesta. Además, el proyecto aborda una QoS altamente flexible. Las decisiones clave deben tomarse lo antes posible en el hardware para aumentar la eficacia de las técnicas propuestas, diseñando e implementando mecanismos inmediatamente en la tarjeta de interfaz de red o en los conmutadores de red. Además, los conmutadores deben ofrecer una visión global de la red. La solución de gestión global de la congestión incluye innovaciones desarrolladas por los socios a varios niveles: i) la definición del protocolo y la especificación de las sondas de hardware para supervisar el estado de la red; ii) los algoritmos para tomar las mejores decisiones de encaminamiento adaptativo y regulación de la inyección; y iii) el soporte para la gestión de la congestión adaptado a las operaciones colectivas.

Las IP desarrolladas abarcan desde los módulos de hardware en los puertos BXI hasta los módulos de firmware que se ejecutan en los componentes y en el software de gestión global del red.

F. Puente RED-SEA Ethernet

Los sistemas de exaescala ya no son sistemas monolíticos y cerrados. Son híbridos, compuestos de particiones especializadas y deben estar abiertos al

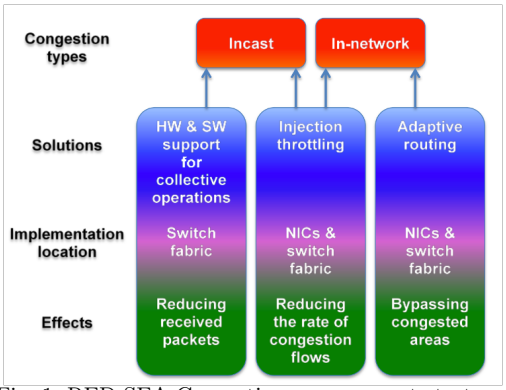


Fig. 1: RED-SEA Congestion management strategy.

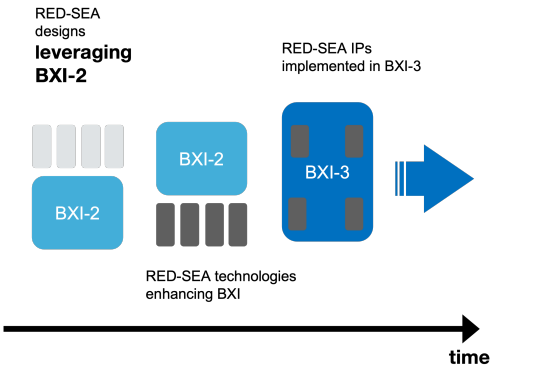


Fig. 2: In the RED-SEA project, we will leverage the existing BXI interconnect to develop, test and evaluate new technologies, planning to integrate many of them to advance BXI..

mundo exterior. Se comunican con otros superordenadores, nubes híbridas y servidores Edge para participar en un flujo de trabajo global. Esto constituye un enfoque de computación continua. Se trata de un cambio disruptivo en la naturaleza de los superordenadores. Como consecuencia, ya no es posible comprometerse en cuestiones de seguridad, y una segunda consecuencia es que las pasarelas a las redes externas están bajo presión. Uno de los demostradores del proyecto es un prototipo de pasarela Ethernet que conecta el la red HPC a la red de almacenamiento Ethernet. La pasarela se implementa como una FPGA de gama alta ajustada a los puertos de los conmutadores BXI que interactúan con los conmutadores Ethernet.

VI. Metodologías

Este proyecto hace uso de varias plataformas de evaluación, incluidos modelos y cálculos analíticos, modelos de simulación por ordenador existentes o nuevos, así como plataformas de emulación de hardware que combinan FPGA, servidores y equipos comerciales (incluido BXI).

En las siguientes subsecciones resumimos las metodologías del proyecto. Se puede encontrar una descripción más extensa en [7].

A. Conexión entre la investigación y BXI

Como se muestra en la Figura 2, el proyecto RED-SEA está creando un bucle de retroalimentación positiva entre la investigación y la industria. El proyecto aprovecha la tecnología BXI2, que se utiliza en sistemas HPC operativos y ofrece altas velocidades de enlace (100-200 Gb/s), conmutadores de alta radix (48 puertos), control de flujo por VC, retransmisiones de extremo a extremo y salto a salto, y descarga de red avanzada. Muchas de las actividades de RED-SEA utilizan BXI2 y sus actuales capacidades de alto rendimiento para probar y validar nuevas IP. Esto proporciona a los equipos de investigación de RED-SEA una ventaja competitiva para avanzar en el estado del arte de las interconexiones industriales, que pueden servir como punto de partida para el desarrollo de ASICs BXI3.

B. Codiseño

En RED-SEA, el codiseño se entiende como un proceso colectivo e iterativo en el que (i) las aplicaciones, cargas de trabajo y ecosistemas relevantes (requisitos); (ii) los diseñadores de interconexión (nuevas soluciones/IPs); y (iii) la empresa Atos (consideraciones de mercado) se reúnen para dar forma, evaluar y promover la IP de interconexión desarrollada en RED-SEA. La interconexión BXI y su evolución son fundamentales para el éxito general de este esfuerzo. Los propietarios de aplicaciones, los proveedores de plataformas HPC y los diseñadores de interconexiones experimentados han trabajado en la confección de una lista ampliada de aplicaciones y cargas de trabajo propias o ampliamente conocidas. Como resultado de este proceso, el consorcio ha confeccionado una serie de cargas de trabajo previstas, asociadas a objetivos de rendimiento relacionados. Todas las soluciones propuestas por los diseñadores de redes dentro de RED-SEA se están evaluando en función de las cargas de trabajo y los objetivos de rendimiento seleccionados por el codiseño de RED-SEA. En RED-SEA, utilizaremos una combinación de plataformas para desarrollar, probar y verificar las soluciones propuestas. Las plataformas de evaluación previstas incluyen un conjunto de modelos de simulación y bancos de pruebas de hardware para ajustarse a los requisitos particulares de cada tarea y potenciar la productividad.

C. Plataformas y modelos de simulación

En RED-SEA, utilizamos modelos de simulación de red con diferentes niveles de abstracción, para evaluar las soluciones previstas. En general, se aceptan ciertas simplificaciones para mejorar el tiempo hasta la solución, en lo que respecta a las evaluaciones de simulación. Cuando se estudia la gestión de la congestión, por ejemplo, las simulaciones ayudan a evaluar diferentes soluciones en diferentes escenarios de congestión dentro de la red, y por lo tanto el modelo no tiene que capturar los detalles del procesador. Cuando se estudia la interfaz de red, pueden ser necesarios modelos detallados de la ruta que conecta el procesador con la interfaz de red para capturar con precisión la latencia y la tasa de mensajes.

Los modelos SystemC y RTL, herramientas que utilizan los equipos de diseño de hardware, también se emplean con frecuencia para examinar en detalle el comportamiento y la corrección de las IP bajo diversas cargas de trabajo. Entre las herramientas de simulación que utilizará el consorcio figuran: OMNeT++ [8], NS3 [9], SystemC, Verilog/SystemVerilog, Gem5 [10], Custom frameworks [11].

En concreto, como se describe más adelante, la UPV y la UCLM están utilizando el simulador de red SAURON [12], para modelar la arquitectura de la tecnología de red BXI, y simular cientos de miles de nodos. Así mismo, para generar tráfico realista que alimente las redes modeladas, se hace uso del framework VEF Traces [13], que también se ha utilizado para capturar los patrones de comunicación de las aplicaciones HPC que se han definido como casos de uso del proyecto: NEST, LAMMPS, LinkTest, etc. Finalment, en cuanto al modelado de estas aplicaciones, la UCLM y la UPV, han impulsado en RED-SEA la colaboración con otros proyectos financiados con fondos europeos, como el DEEP-SEA, IO-SEA y MAELSTROM, con el objetivo de capturar trazas de tráfico en la red adicionales en el formato VEF, conforme a las aplicaciones y casos de uso de estos proyectos. Se han obtenido trazas de aplicaciones como GROMACS, PATMOS o LinkPack Todas estas trazas están disponibles en un repositorio público[1].

D. Estrategia de emulación de hardware

Uno de los objetivos del proyecto RED-SEA es desarrollar nuevos IPs hardware que puedan ser utilizados para mejorar la interconexión BXI. Se utilizan simulaciones a nivel funcional para evaluar el impacto en el rendimiento de estas IPs bajo patrones de tráfico sintéticos o trazas de mini-aplicaciones. RED- SEA está probando además el rendimiento de estas IPs en despliegues reales utilizando una plataforma de emulación de hardware, que mezcla placas comerciales/verificadas (por ejemplo, conmutador BXI) programables FPGAs programables con modelos modificables de componentes BXI (por ejemplo, conmutadores BXI y NICs), servidores informáticos y placas FPGA.

Testbeds de pequeña escala. El consorcio está utilizando bancos de pruebas a pequeña escala para fines de emulación que incluyan placas FPGA acopladas a componentes existentes, como conmutadores BXI, tarjetas de interfaz de red (NIC) BXI, conmutadores y adaptadores Ethernet, con el fin de validar la funcionalidad de las IP y medir su rendimiento.

Testbeds de alta escala. El consorcio aprovecha la plataforma multi-FPGA del proyecto ExaNeSt [1] para desarrollar y probar protocolos y puntos finales heterogéneos, como procesadores RISC-V y aceleradores FPGA.

Dibona: Arm-based HPC Cluster. El cluster Dibona es una máquina diseñada en el proyecto Mont-Blanc 3. Dibona se está reutilizando para análisis y optimizaciones de prestaciones de BXI. Dibona cuenta con procesadores ARMv8 ThunderX2 conectados mediante interconexión InfiniBand o BXI en una topología fat-tree. Dibona se compone de 3 placas (nodos de cálculo), cada uno de ellas optimizado para integrar 2 zócalos (CPU) y 16 canales de memoria. En el contexto de este proyecto, las placas base se actualizarán con soporte BXI en el mezzanine de interconexión.

E. Cargas HPC y Datacentre Relevantes

La lista de aplicaciones y puntos de referencia de red específicos seleccionados en el proyecto es:

NEST. NEST es una aplicación consolidada que abarca una gama mucho más amplia de modelos neuronales y de conectividad sináptica, que admite la ejecución paralela mediante un híbrido MPI y OpenMP, y que proporciona una interfaz Python para facilitar la configuración y la interoperabilidad con códigos para la manipulación algebraica y la investigación estadística de la red simulada y su dinámica.

LAMMPS. Large-scale Atomic/Molecular Massively Parallel Simulator es un motor clásico de dinámica molecular centrado en el modelado de materiales. Se utiliza ampliamente en varias ramas de la ciencia: física del estado sólido, química computacional, biofísica y muchas otras.

SOM. Los mapas autoorganizados (SOM) son redes neuronales artificiales que se utilizan en el contexto del aprendizaje automático no supervisado. En el contexto de RED SEA, se ha desarrollado una implementación paralela del algoritmo SOM. El paquete, llamado DIAPASOM, explota los enfoques de paralelización MPI y PGAS (a través de OPENSHMEM). El código se ha publicado bajo licencia BSD-4-Clause y está a disposición del público en https://github.com/exactlab/diapasom.

DAW. DAW (Datacentre-inspired adversarial workloads) es un conjunto de generadores de escenarios de banco de pruebas para reproducir cargas de trabajo interesantes de la plataforma a gran escala ExaNeSt que ponen a prueba las capacidades de interfaz de red a escala y las capacidades de calidad de servicio de la interconexión.

LinkTest. LinkTest es una herramienta para la evaluación comparativa escalable de API de comunicación. Las API y el hardware de comunicación asociado se evalúan mediante el envío de mensajes entre tareas alojadas en la misma o en diferentes CPU/GPU. Los mensajes pueden enviarse entre dos tareas en paralelo: una tarea envía su mensaje a la otra mientras ésta le devuelve el suyo. Alternativamente, los mensajes pueden enviarse uno tras otro. Además, se puede controlar la ubicación en la que se almacenan estos mensajes. Pueden residir en la RAM de la

[1] https://gitraap.i3a.info/jesus.escudero/vef-traces-repository

CPU o en la RAM de la GPU.

PCVS. PCVS (Parallel Computing Validation Suite) es un motor de validación diseñado para evaluar las capacidades de descarga de la red de alta velocidad ejecutando grandes bases de pruebas de forma escalable, aprovechando entornos altamente paralelos para reducir su tiempo de obtención de resultados, mejorando posteriormente la eficiencia del proyecto gracias a un proceso de validación más regular.

VII. Implementación

El proyecto RED-SEA se está llevando a cabo en torno a cuatro pilares clave y en torno a estos pilares, RED-SEA define 4 paquetes de trabajo técnicos que se muestran en la Figura 3.

El paquete de trabajo WP1, "Co-design & performance", tiene como objetivo recopilar los requisitos de las aplicaciones y ecosistemas objetivo. Además, el WP1 cubre el trabajo necesario para construir o adaptar las plataformas de evaluación existentes (bancos de pruebas de emulación y simulación) y evaluar las soluciones nuevas o existentes, cubriendo aspectos como la calidad del servicio y la fiabilidad a escala, la portabilidad de los códigos existentes, frente a las aplicaciones y las cargas de trabajo establecidas anteriormente.

El paquete de trabajo WP2, "High performance Ethernet"(Ethernet de alto rendimiento), persigue desarrollar IPs que permitan puentes de altas prestaciones y baja latencia entre BXI y Ethernet, para eliminar la necesidad de sockets Ethernet en los servidores y de switches/routers Ethernet consolidando el tráfico Ethernet.

El paquete de trabajo 3, "Gestión eficiente de los recursos de red", aprovecha los recursos disponibles en la arquitectura BXI para mejorar el rendimiento de la red, centrándose principalmente en la gestión de la congestión y la calidad del servicio, pero también en el enrutamiento adaptativo y la gestión de la energía. En cuanto a la gestión de la congestión, se estudian distintos enfoques (optimización colectiva de las comunicaciones primitivas, estrangulamiento de la inyección, esquemas de colas, etc.) para tratar de forma más eficiente los distintos tipos de congestión.

Por último, el paquete de trabajo 4, "Funciones de punto final y fiabilidad", persigue desarrollar protocolos de extremo a extremo mejorados para avanzar en la fiabilidad a la escala de sistema prevista, trabajar en optimizaciones de MPI y en modelos de programación para aceleradores de red, así como en la interoperabilidad de la interconexión con tecnologías emergentes de procesadores y aceleradores de bajo consumo, como las diseñadas en EPI.

VIII. Hitos Intermedios

Durante la primera mitad del proyecto RED-SEA, se han llevado a cabo acciones fructíferas en todos los paquetes de trabajo técnicos, que son los pilares clave para guiar y alcanzar los retos globales del proyecto. A continuación, resumimos los principales logros para los paquetes de trabajo 1 y 3, en los que UPV y UCLM están más activos.

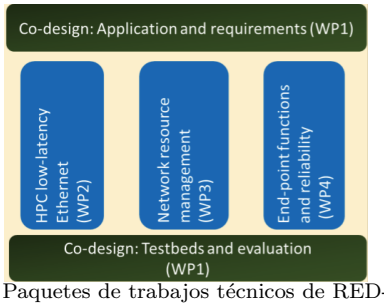


Fig. 3: Paquetes de trabajos técnicos de RED-SEA.

A. WP1: Architecture, Co-design, and Performance

El diseño de hardware de la red de interconexión y el diseño de software de middleware necesitan del conocimiento de los patrones de comunicación de la red, en un entorno de codiseño de hardware y software.

Partiendo de una descripción detallada del ámbito y campo de aplicación de cada una de las aplicaciones, hemos esbozado los requisitos de hardware y software de cada uno de ellos. VEF Traces framework [13], desarrollada por la UCLM, se ha establecido como el marco de referencia en el proyecto para recopilar trazas y utilizarlas con fines de simulación. Además, el proyecto ha proporcionado apoyo a otros proyectos SEA para instalar y ejecutar el entorno. De hecho, el proyecto DEEP-SEA ha compartido con RED-SEA algunas trazas VEF recogidas de aplicaciones objetivo de DEEP-SEA, como GROMACS, PATMOS y LinPack. A continuación, mediante el análisis de las trazas de red, hemos proporcionado recomendaciones para el diseño del sistema de red en relación con la latencia, el ancho de banda en rangos específicos de tamaño de mensaje, el número de mensajes MPI y la velocidad de transmisión.

Con el fin de proporcionar recomendaciones para la arquitectura de red, hemos realizado un análisis ampliado de las dos principales aplicaciones disponibles en RED-SEA, LAMMPS y NEST, utilizando el conjunto de herramientas VEFtraces.

Por último mencionar que la UPV y la UCLM han invertido un esfuerzo importante en el simualdor Sauron **??** con el objetivo de modelar la arquitectura BXI3, y validarla contra el diseño de bajo nivel de ATOS (en SystemC). Este modelo se está ampliando para incluir y evaluar las propuestas realizadas en el WP3.

B. WP3: Efficient Network Resources Management

En el paquete de trabajo 3, un punto clave a atacar para gestionar eficientemente los recursos de red es la congestión. Por ello, uno de los primeros pasos de este paquete de trabajo ha sido realizar la caracterización de los escenarios de congestión de red derivados del tráfico generado por las aplicaciones objetivo del proyecto RED-SEA (LAMMPS y NEST) identificadas en el paquete de trabajo 1. Realizamos esta caracterización con el objetivo de adquirir un cono-

cimiento completo del comportamiento de la congestión causada por las aplicaciones objetivo, con el fin de diseñar los mecanismos de control de la congestión de este paquete de trabajo. Para realizar esta caracterización, se han utilizado trazas obtenidas mediante el framework VEF Trace [13] al ejecutar las dos aplicaciones mencionadas en el clúster Dibona.

Hemos observado que las prestaciones de la red están muy determinadas por las primitivas colectivas (tipo y frecuencia) que realiza cada carga de trabajo. Más concretamente, las prestaciones de la red vienen determinadas por los patrones de comunicación definidos por estas primitivas colectivas de las cargas de trabajo en ejecución. Por este motivo, como primer paso, analizamos el tipo, la frecuencia y los patrones de comunicación de las primitivas de comunicación colectiva de las aplicaciones en ejecución. Los resultados experimentales han mostrado que LAMMPS es una aplicación con una variedad de primitivas de comunicación colectiva (ver Figura 4a), mientras que NEST está dominada por la primitiva All2All (ver Figura 4b) y esto afecta a la distribución de mensajes y tráfico en la red. En el caso de NEST (ver Figura 5b), la distribución global de mensajes, considerando toda la traza, es uniforme entre pares de tareas, pero en el caso de LAMMPS la distribución no es uniforme (ver Figura 5a), ya que en LAMMPS existen primitivas colectivas donde los mensajes son enviados o recibidos por la tarea número cero (una sola tarea).

Otra observación interesante es el comportamiento a ráfagas del tráfico de red generado por ambas aplicaciones, que puede impedir que los paquetes avancen, generando una alta ocupación en las colas, aumentando así la latencia de los paquetes y reduciendo la productividad de la red.

Con el objetivo de reducir la congestión provocada por las primitivas de comunicación colectiva, estamos llevando a cabo una optimización software-hardware de algunas primitivas de comunicación colectiva cruciales teniendo en cuenta la topología de la red. En comparación con nuestro objetivo, las instancias actuales de MPI para la primitiva colectiva son agnósticas a la topología de la red. Al no tener en cuenta la topología, estas implementaciones malgastan el ancho de banda de la red, creando congestión. La optimización del hardware se basa en el enrutamiento multidifusión asistido por hardware. El objetivo es reducir el número de mensajes generados, así como el volumen de tráfico.

Las optimizaciones hardware se basan en tener soporte de los switches de la red para reducir el número de mensajes enviados por la red en la implementación de la primitivas de comunicación colectivas y de esta forma reducir la congestión.

Las optimizaciones software se basan en optimizar los algoritmos que implementan las primitivas de comunicación colectiva (CCP). En esta línea, distinguimos dos técnicas de optimización de software. En primer lugar, la aplicación de pipelining de comunicación mediante la segmentación del mensaje, con el fin de lograr un solapamiento más eficiente con el cómputo. En segundo lugar, la introducción de técnicas que tienen en cuenta la topología para lograr una comunicación más eficiente.

La figura 6 muestra el efecto sobre las prestaciones de la segmentación de mensajes (solapamiento de cálculo y comunicación) para dos CCP asíncronos de OpenMPI: MPI-Ibcast (gráfico de la izquierda), MPI-Ireduce-scatter, y (gráfico de la derecha), mostrando que es posible acelerar el rendimiento aplicando pipelining.

Además se han implementado dos algoritmos conscientes de la topología dragonfly y la primitiva broadcast: LLF (Local Level First) and GLF (Global Level First). En la Figura 7 puede verse una comparativa de los algoritmos tradicionales (N y log N) con las optimizaciones hardware y los dos algoritmos software optimizados (LLF and GLF). En la figura se muestra el speedup para los algoritmos con respecto al algoritmo tradicional N. Como se puede ver el algoritmo HW es el que mejora más las prestaciones seguido del LLF y además son capaces de mantener la mejora con el aumento del tamaño del mensaje.

Además, con el objetivo de abordar la congestión, en este paquete de trabajo se está también trabajando en propuestas de control de la congestión basadas en colas y encaminamiento adaptativo. Respecto al encaminamiento adaptativo se ha acordado con Atos el diseño de un algoritmo que requiere bajo coste e impedirá la propagación de la congestión.

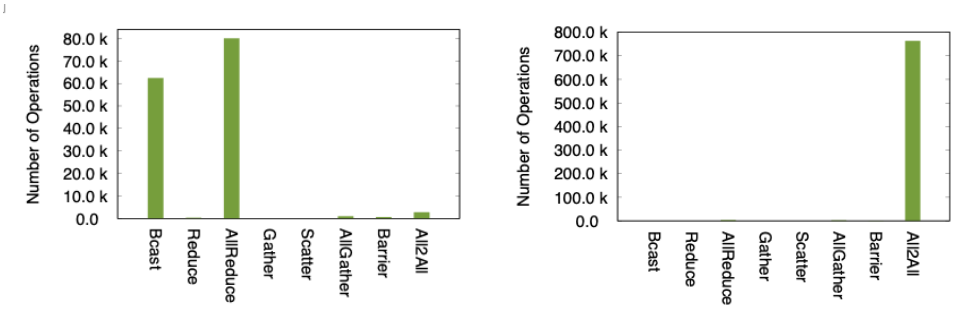
En este paquete de la trabajo la UPV y la UCLM también están colaborando junto con Atos en el diseño de técnicas de QoS y aislamiento de tráfico que sean compatibles con BXI y de técnicas para la reducción del consumo de energía en la red.

IX. Conclusiones

Este trabajo ha resumido los retos impuestos para desarrollar la próxima red europea para un sistema de escala extrema. Hemos presentado el enfoque de RED-SEA para abordar estos retos durante un plazo de 3 años. Estos ambiciosos objetivos son alcanzables, como demuestran las acciones llevadas a cabo con éxito a mitad del proyecto.

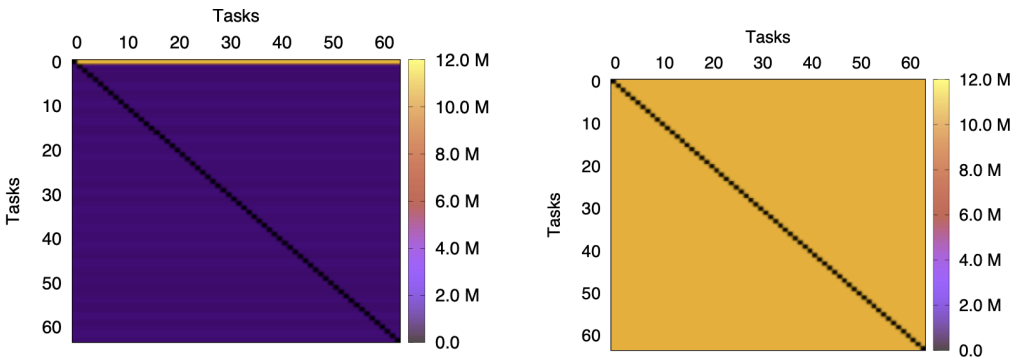
Entre los hitos más relevantes nos gustaría citar los siguientes:

- Se han definido los requisitos de la red, la arquitectura y una lista inicial de puntos de referencia y aplicaciones. En cuanto al tipo típico de comunicaciones MPI, las dos aplicaciones consideradas hacen uso de MPIAllToAll, MPIBroadcast y MPIAllReduce principalmente, por lo que el correspondiente tipo de patrón de comunicaciones requerirá especial atención durante el diseño de la red.
- Se ha caracterizado la congestión en el sistema objetivo y se ha dado cuenta del alto impacto de algunas primitivas de comunicación colectiva en las prestaciones y se han propuesto mecanismos para hacer frente a la congestión mostrando los beneficios de prestaciones que se pueden lograr.



(a) LAMMPS (b) NEST

Fig. 4: Número de llamadas a colectivas por tipo para Lammps y Nest.



(a) LAMMPS (b) NEST

Fig. 5: Bytes transferidos entre los pares de tareas.

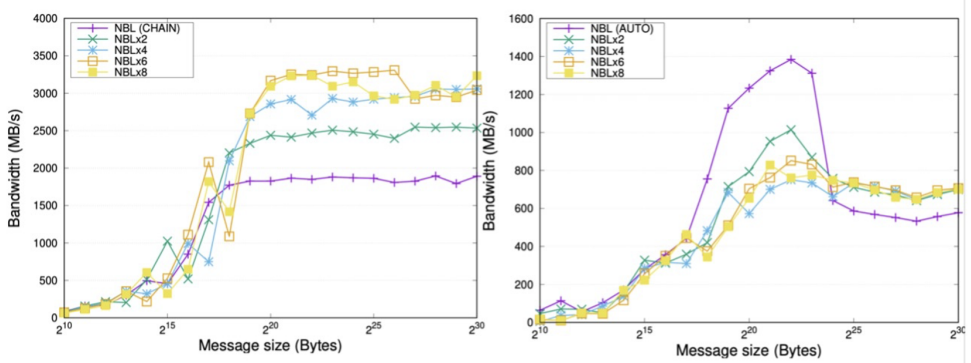


Fig. 6: Prestaciones de MPI-Ibcast y MPI-Ireduce-scatter OpenMPI con número fijo de segmentos. 8 nodos conectador por red EDR. NBL corresponde con la implementación inicial.

Fig. 7: Prestaciones de MPI-Ibcast y MPI-Ireduce-scatter OpenMPI con número fijo de segmentos. 8 nodos conectador por red EDR. NBL corresponde con la implementación inicial.

Agradecimientos

El presente trabajo ha sido financiado por la Comisión Europea (programa Horizon 2020, call H2020-JTI-EuroHPC-2019-1 - grant agreement ID: 955776) y Ministerio de Ciencia e Innovación de España (proyectos PCI2021-121934 y PCI2021-121976).

Referencias

[1] Manolis Katevenis et al., "Next generation of exascale-class systems: Exanest project and the status of its interconnect and storage development," *Microprocessors and Microsystems*, vol. 61, pp. 58–71, 2018.

[2] Manolis Ploumidis, Nikolaos D. Kallimanis, Marios Asiminakis, Nikos Chrysos, Pantelis Xirouchakis, Michalis Gianoudis, Leandros Tzanakis, Nikolaos Dimou, Antonis Psistakis, Panagiotis Peristerakis, Giorgos Kalokairinos, Vassilis Papaefstathiou, and Manolis Katevenis, "Software and hardware co-design for low-power hpc platforms," Berlin, Heidelberg, 2019, p. 88–100, Springer-Verlag.

[3] Biagioni, Andrea et al., "Euroexa custom switch: an innovative fpga-based system for extreme scale computing in europe," *EPJ Web Conf.*, vol. 245, pp. 09004, 2020.

[4] R Ammendola, A Biagioni, O Frezza, F Lo Cicero, A Lonardo, P S Paolucci, D Rossetti, F Simula, L Tosoratto, and P Vicini, "APEnet+: a 3D torus network optimized for GPU-based HPC systems," *Journal of Physics: Conference Series*, vol. 396, no. 4, pp. 042059, dec 2012.

[5] Jeffrey S Vetter, Ron Brightwell, Maya Gokhale, Pat McCormick, Rob Ross, John Shalf, Katie Antypas, David Donofrio, Travis Humble, Catherine Schuman, et al., "Extreme heterogeneity 2018-productive computational science in the era of extreme heterogeneity: Report for doe ascr workshop on extreme heterogeneity," 2022.

[6] Ken Raffenetti, Antonio J Pena, and Pavan Balaji, "Toward implementing robust support for portals 4 networks in mpich," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2015, pp. 1173–1176.

[7] Andrea Biagioni et al., "RED-SEA: network solution for exascale architectures," in *25th Euromicro Conference on Digital System Design, DSD 2022, Maspalomas, Spain, August 31 - Sept. 2, 2022*. 2022, pp. 712–719, IEEE.

[8] András Varga, "Omnet++," in *Modeling and Tools for Network Simulation*, Klaus Wehrle, Mesut Günes, and James Gross, Eds., pp. 35–59. Springer, 2010.

[9] George F. Riley and Thomas R. Henderson, *The ns-3 Network Simulator*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[10] Nathan Binkert et al., "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, aug 2011.

[11] Nikolaos Tampouratzis, Ioannis Papaefstathiou, Antonios Nikitakis, Andreas Brokalakis, Stamatis Andrianakis, Apostolos Dollas, Marco Marcon, and Emanuele Plebani, "A novel, highly integrated simulator for parallel and distributed systems," *ACM Trans. Archit. Code Optim.*, vol. 17, no. 1, mar 2020.

[12] P. Yebenes, J. Escudero-Sahuquillo, P. J. Garcia, and F. J. Quiles, "Networks of exascale systems with omnet++.," in *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2013, pp. 203–207.

[13] F. J. Andújar, J. A. Villar, F.J. Alfaro, and et al., "An open-source family of tools to reproduce mpi-based workloads in interconnection network simulators.," *Journal of Supercomputing*, , no. 72, pp. 042059, 2016.

[14] Daniele De Sensi, Salvatore Di Girolamo, Kim H. McMahon, Duncan Roweth, and Torsten Hoefler, "An in-depth analysis of the slingshot interconnect," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–14.

[15] Mark S. Birrittella, Mark Debbage, Ram Huggahalli, James Kunz, Tom Lovett, Todd Rimmer, Keith D. Underwood, and Robert C. Zak, "Intel® omni-path architecture: Enabling scalable, high performance fabrics," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, 2015, pp. 1–9.

[16] Saïd Derradji, Thibaut Palfer-Sollier, Jean-Pierre Panziera, Axel Poudes, and François Wellenreiter Atos, "The bxi interconnect architecture," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. IEEE, 2015, pp. 18–25.

[17] Roberto Ammendola, Massimo Bernaschi, Andrea Biagioni, Mauro Bisson, Massimiliano Fatica, Ottorino Frezza, Francesca Lo Cicero, Alessandro Lonardo, Enrico Mastrostefano, Pier Stanislao Paolucci, Davide Rossetti, Francesco Simula, Laura Tosoratto, and Piero Vicini, "Gpu peer-to-peer techniques applied to a cluster interconnect," in *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, 2013, pp. 806–815.

[18] Adrià Armejach, Bine Brank, Jordi Cortina, François Dolique, Timothy Hayes, Nam Ho, Pierre-Axel Lagadec, Romain Lemaire, Guillem López-Paradís, Laurent Marliac, Miquel Moretó, Pedro Marcuello, Dirk Pleiter, Xubin Tan, and Said Derradji, "Mont-blanc 2020: Towards scalable and power efficient european hpc processors," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 136–141.

[19] Norbert Eicker, Thomas Lippert, Thomas Moschny, Estela Suarez, and for the DEEP project, "The deep project an alternative approach to heterogeneous cluster-computing in the many-core era," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 8, pp. 2394–2411, 2016.

[20] Theodoropoulos et al., "The AXIOM project (Agile, eXtensible, fast I/O Module)," in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2015, pp. 262–269.

[21] "Epi: European processor iniciative," .

[22] Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, Luca Benini, and Torsten Hoefler, "A risc-v in-network accelerator for flexible high-performance low-power packet processing," in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, 2021, ISCA '21.

[23] Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler, "Flare: Flexible in-network allreduce," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2021, SC '21, Association for Computing Machinery.

[24] Nikolaos Chrysos and Manolis Katevenis, "Scheduling in non-blocking buffered three-stage switching fabrics.," in *INFOCOM*, 2006, vol. 6, pp. 1–13.

[25] Antonis Psistakis, Nikos Chrysos, Fabien Chaix, Marios Asiminakis, Michalis Gianioudis, Pantelis Xirouchakis, Vassilis Papaefstathiou, and Manolis Katevenis, "Optimized page fault handling during rdma," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2022.

Análisis de la influencia de la orografía en despliegues IoT basados en LoRaWAN

Vicente Torres-Sanz†, Julio A. Sanguesa†, Felix Serna†,
Francisco J. Martinez†, Piedad Garrido†, Carlos T. Calafate⋆
†Depto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza
⋆Depto. de Informática de Sistemas y Computadores, Universitat Politècnica de València
Email: {vtorres, jsanguesa, fserna, f.martinez, piedad}@unizar.es, calafate@disca.upv.es

Resumen— **Long Range Wide Area Network (LoRaWAN) es un protocolo de red diseñado específicamente para aprovechar las ventajas de la tecnología Long Range (LoRa). LoRaWAN se utiliza para la conexión de dispositivos IoT a través de una infraestructura de red de largo alcance, proporcionando una capa de comunicación segura y eficiente que permite la conectividad de una cantidad masiva de dispositivos en una red LoRa. El objetivo de este trabajo es evaluar el rendimiento de LoRaWAN en entornos con distinta orografía. En concreto, se comparan dos escenarios (uno llano y otro montañoso), analizando la tasa de éxito en la recepción de paquetes de datos. El propósito es analizar cómo afecta la orografía a la comunicación entre los dispositivos y gateways LoRa, así como explorar alternativas para superar los desafíos en entornos con terreno accidentado. Por ello, en el entorno montañoso, además se analiza si el uso de drones puede ser beneficioso para mejorar las comunicaciones. Los resultados de este estudio revelan que la orografía del terreno tiene un impacto significativo en las comunicaciones LoRa, obteniendo una disminución del 58,63 % en el número de paquetes recibidos con éxito en comparación con un entorno llano. Además, un tercio de los nodos no lograron establecer comunicación con el gateway terrestre, lográndose una mejora parcial en la conectividad, mediante el uso de gateways desplegados en drones.**

Palabras clave— **IoT, LoRaWAN, LoRa, orografía, drones.**

I. Introducción

El llamado Internet de las Cosas (IoT, por sus siglas en inglés) proporciona a cada objeto la capacidad de estar conectado a Internet. Al hacer esto, podemos monitorizar parámetros relevantes y acceder a ellos en cualquier momento y lugar. Se espera que el número de dispositivos IoT conectados en todo el mundo crezca un 12 % anual en promedio, hasta alcanzar los 125 mil millones en 2030 [1].

En este contexto, el IoT se utiliza cada vez más, y han surgido tecnologías de comunicaciones especialmente diseñadas para ese ámbito, como es el caso de Long Range Wide Area Network (LoRaWAN), un protocolo de comunicación de largo alcance y bajo consumo, basado en la tecnología de modulación LoRa, el cual permite la transmisión inalámbrica de datos a larga distancia entre dispositivos.

El funcionamiento de LoRaWAN se basa en una arquitectura de red donde los dispositivos finales (nodos) se comunican con una o varias estaciones base (gateways), que actúan como intermediarios entre los dispositivos y el servidor central de red. Las estaciones base reciben los mensajes de los nodos y los retransmiten al servidor, y viceversa.

Una de las principales características de LoRa es su amplio alcance. En teoría, puede proporcionar comunicación bidireccional a varios kilómetros, incluso en áreas urbanas densamente pobladas o en entornos rurales, aunque esto puede variar dependiendo de varios factores, como la potencia de transmisión utilizada y la orografía, entre otros. En condiciones óptimas, pueden alcanzar varios kilómetros de distancia. Por ejemplo, Arratia et al. [2] consiguieron comunicación entre el nodo emisor y un gateway a una distancia máxima de 11,69 km en el mar, un escenario totalmente llano y sin obstáculos.

Otra característica clave de esta tecnología, es que los dispositivos LoRa son capaces de operar durante largos períodos con baterías de tamaño reducido, o incluso con fuentes de energía alternativas, lo que los hace ideales para su uso en aplicaciones IoT, donde la eficiencia energética es fundamental.

Además de su largo alcance y bajo consumo de energía, LoRaWAN también ofrece una capacidad de conexión masiva, ya que puede admitir miles de dispositivos conectados simultáneamente; esto permite una implementación escalable de soluciones IoT en una amplia variedad de casos de uso.

Otra ventaja importante radica en la capacidad de recopilar datos en tiempo real desde dispositivos LoRa mediante el uso de un dron. El empleo de un gateway integrado en un dron puede representar una alternativa más rentable y eficiente en términos de recursos, en comparación con la instalación de infraestructuras de comunicación terrestre, especialmente en áreas rurales con orografías montañosas y con escasa conectividad.

Por todo lo anterior, la tecnología LoRa tiene un amplio potencial, y su flexibilidad y eficiencia energética, la convierten en una opción atractiva para una amplia gama de aplicaciones de IoT. Algunos ejemplos de uso son:

- Monitorización y gestión de infraestructuras inteligentes, como las redes eléctricas [3], los sistemas de agua [4] y gas [5], así como la gestión de residuos en las smart cities [6].
- Agricultura de precisión, para la monitorización de cultivos, la gestión inteligente del riego [7], el seguimiento del ganado [8], y la sensorización de las condiciones ambientales en tiempo real [9].

- Ciudades inteligentes, para habilitar soluciones de iluminación [10], aparcamiento inteligente [11], seguimiento de flotas [12], o el despliegue de sensores para el control de los niveles de contaminación [13].
- Salud y bienestar, como la monitorización en remoto de pacientes y sistemas de alerta médica [14].
- Seguridad y vigilancia, aplicada a los sistemas de alarma y detección de incendios [15], la seguridad en edificios y el control de perímetros [16].

En las comunicaciones inalámbricas, la zona de Fresnel [17] es una región elíptica alrededor de la línea de visión directa entre los dispositivos que se comunican, en la que la señal se propaga con una mínima atenuación y distorsión. Por ello esta zona se muestra como un factor clave a la hora de lograr comunicaciones con éxito en largas distancias. Si existen obstáculos entre los dispositivos (como colinas o edificios), puede producirse una difracción o incluso el completo bloqueo de la señal, lo que impediría la comunicación. Esto hace que sea importante la correcta ubicación de los dispositivos y las antenas, para evitar obstáculos dentro de la zona de Fresnel. Por lo tanto, resulta crucial tener en cuenta la topografía del escenario y garantizar que exista suficiente espacio libre en dicha región (al menos un 60 %) para asegurar una correcta comunicación.

En el presente artículo se presenta una comparación del funcionamiento y la conectividad de dispositivos LoRa en dos escenarios diferentes: (i) un escenario con perfil llano y (ii) otro con perfil montañoso, con el objetivo de comprobar el impacto de la orografía en las comunicaciones basadas en esta tecnología. Por último, se pretende analizar los beneficios de utilizar drones como una opción para establecer conexiones con los nodos LoRa en entornos montañosos. De esta manera, se busca comprender hasta qué punto la orografía influye en el desempeño de las redes LoRaWAN y explorar nuevas alternativas para superar los desafíos que presenta la comunicación en entornos con relieve accidentado.

El resto del documento se organiza del siguiente modo: en la Sección II, se presentan los escenarios utilizados para comparar el desempeño y la cobertura de dispositivos LoRa en entornos con topografía llana (Sección II-A) y montañosa (Sección II-B). A continuación, en la Sección III se analizan los resultados de las pruebas realizadas para evaluar el rendimiento de los dispositivos LoRa en ambos escenarios, y se presenta una comparación de los resultados obtenidos. Por último, en la Sección IV, se presentan las conclusiones derivadas del estudio realizado.

II. Entorno de pruebas

Con el propósito de llevar a cabo las pruebas, se establecieron dos escenarios para evaluar el funcionamiento de LoRaWAN. El primero en un entorno llano, y el segundo con una orografía montañosa. Estos escenarios se encuentran en los municipios de Aguilar del Alfambra y Aliaga, respectivamente, ambos situados en la provincia de Teruel.

La metodología consistió en el despliegue de 6 nodos LoRa, manteniendo las mismas distancias respecto al gateway en ambos entornos, y que estuvieron transmitiendo paquetes de datos cada 30 segundos, examinando a posteriori el porcentaje de paquetes que el gateway recibió con éxito. El objetivo principal era establecer comunicación y transmitir paquetes de datos hacia el gateway, con el fin de evaluar y comparar los resultados obtenidos en los dos escenarios.

Adicionalmente, en el escenario montañoso se incorporó un dron equipado con otro gateway, con el propósito de realizar una comparación de rendimiento frente al gateway terrestre, y así explorar la viabilidad de utilizar drones como una alternativa favorable para establecer conexiones con los nodos LoRa. Esta estrategia permite evaluar el rendimiento y las posibles ventajas del uso de este tipo de vehículos, con el fin de mejorar la comunicación y superar los desafíos ocasionados por la orografía montañosa.

A. Escenario llano

El primer experimento se ha realizado con el propósito de evaluar el rendimiento de la comunicación de dispositivos en un escenario prácticamente plano, que ofrece un escenario favorable en la comunicación de los dispositivos, evaluando la tasa de éxito de llegada al gateway de los paquetes enviados por los dispositivos. Con esta finalidad, se ha instalado un gateway en la proximidad de una ermita, mientras que se han desplegado seis nodos que se encargan de enviar paquetes de datos al gateway central (ver Figura 1).

Como se puede observar en la Tabla I, tanto el gateway como los nodos en las posiciones 1, 2, 3 y 4 se encuentran a una altitud de 1275 metros. Sus distancias respecto al gateway fueron de 250, 190, 500 y 620 metros, respectivamente. Por otro lado, el nodo en la posición 5 se encontraba a una altitud de 1305 metros y a una distancia de 3,2 kilómetros. En cuanto al nodo situado en la posición 6, presentaba una altitud de 1305 metros y una distancia de 2,2 kilómetros. Por consiguiente, la máxima diferencia de altitud entre un nodo y el gateway alcanza los 30 metros (nodos P5 y P6).

El Spreading Factor (SF) es un parámetro utilizado en los sistemas de comunicación como los basados en LoRa, que determina la velocidad de transmisión de datos, así como la resistencia a interferencias y la capacidad de penetración de la señal. Cuanto ma-

Tabla I: Altitud de cada nodo y distancia al gateway, en el escenario llano.

Nodo	**Altitud (m.)**	**Distancia al gateway (m.)**
P1	1275	250
P2	1275	190
P3	1275	500
P4	1275	620
P5	1305	3200
P6	1305	2200

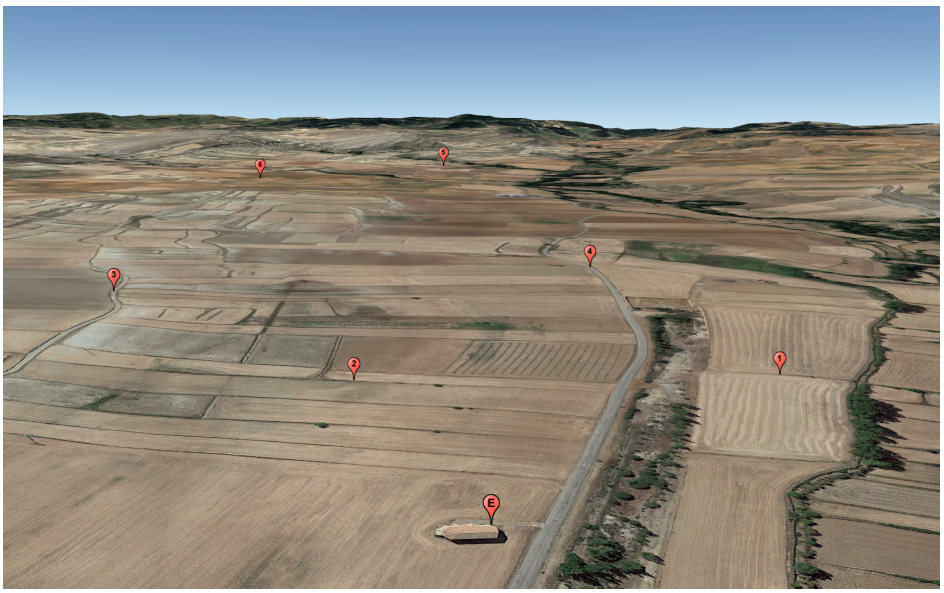


Fig. 1: Ubicación del gateway (E) y de los nodos (1-6), en el escenario llano (Aguilar del Alfambra, Teruel).

Tabla II: Altitud de cada nodo y distancia hasta el gateway y el dron situado a diferentes alturas, en el escenario montañoso.

Nodo	Altitud (m.)	Distancia al Gateway terrestre (m.)	Distancia al dron 50 m. (m.)	Distancia al dron 120 m. (m.)	Distancia al dron 200 m. (m.)
P1	1310	250	254	277	173
P2	1310	190	196	224	592
P3	1285	500	502	514	611
P4	1350	620	622	630	616
P5	1300	3200	3200	3200	2982
P6	1385	2200	2200	2202	2053

yor sea el Spreading Factor, menor será la velocidad de transmisión de datos, pero se logrará un mayor alcance de la señal y una mejor penetración en entornos con obstáculos. En LoRa, el SF varía de 7 a 12.

En nuestro experimento realizamos tres repeticiones, variando el SF (con valores de 7, 10 y 12, respectivamente). El objetivo principal es determinar si la modificación del SF permite mejorar la capacidad de comunicación de los dispositivos LoRa en este escenario.

En definitiva, el análisis de los resultados obtenidos en este experimento permitirá conocer el rendimiento de la comunicación de dispositivos LoRa en un entorno llano y favorable. Además nos permitirá conocer si el SF es un parámetro determinante a la hora de realizar comunicaciones con dispositivos LoRa en un escenario con orografía llana. Los datos recopilados proporcionarán información valiosa sobre la efectividad de la tecnología en escenarios sin interferencias topográficas, con el objetivo de poder compararlo con un escenario montañoso, a priori, menos favorable.

B. Escenario montañoso

Se ha realizado un segundo experimento, con el objetivo de analizar el rendimiento de la comunicación de dispositivos LoRa en un entorno que presenta una orografía montañosa (ver Figura 2).

Para ello, se ha desplegado un gateway en los alrededores de una masía y se han desplegado seis nodos LoRa a la misma distancia del gateway que en el escenario llano. El objetivo principal de este experimento es evaluar la conectividad de los dispositivos con el gateway en un entorno donde la orografía montañosa puede tener un impacto negativo en las comunicaciones. Se busca obtener el porcentaje de paquetes recibidos correctamente en este entorno adverso.

En este escenario, el gateway fue posicionado a una altitud de 1.295 metros, mientras que los nodos, tal y como se refleja en la Tabla II, se ubicaron a altitudes de 1310, 1310, 1285, 1350, 1300 y 1385 metros respectivamente. El nodo situado en la posición 6 presenta la mayor diferencia de altitud con respecto al gateway, con una diferencia de 90 metros. Las distancias entre los nodos y el gateway fueron las mismas que en el escenario previo. Se observa que las diferencias de altitud entre el gateway y los nodos no difieren mucho de las utilizadas en el escenario con topografía llana, sin embargo, en este caso, el escenario presenta una orografía montañosa.

Por otra parte, la integración de un gateway en un dron puede presentar diversas ventajas significativas en el contexto de las comunicaciones IoT. En primer lugar, al acoplar el gateway al dron, se amplía considerablemente el alcance de la red, permitiendo acortar las distancias y establecer conexiones inalámbricas en áreas remotas o de difícil acceso. Además, la movilidad del dron brinda flexibilidad para desplegar rápidamente la infraestructura de comunicación en diversas ubicaciones, lo que resulta especialmente beneficioso en situaciones de emergencia en entornos que requieren respuestas ágiles.

Para analizar el impacto del dron en las comuni-

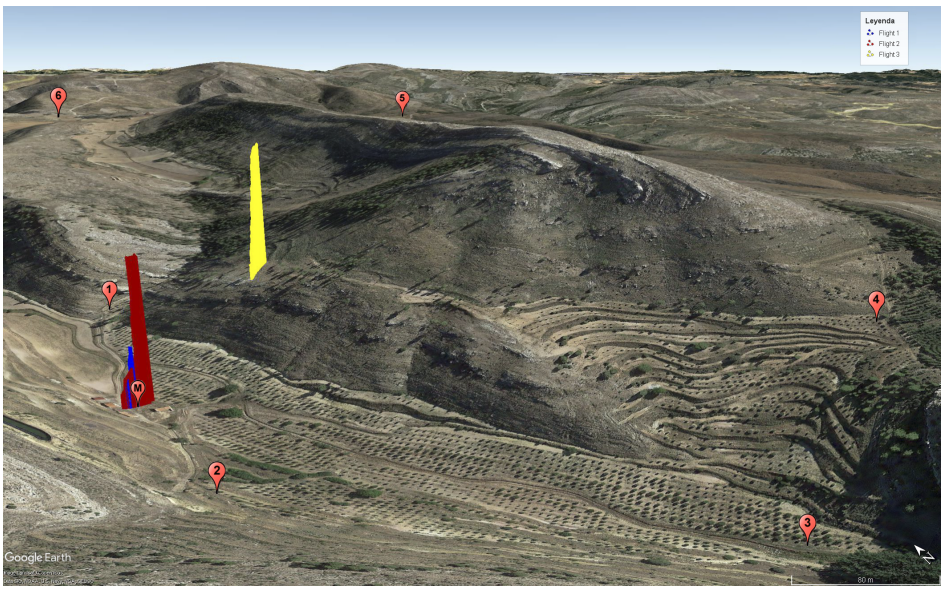


Fig. 2: Ubicación del gateway terrestre (M), del dron (en azul, rojo y amarillo), así como de los nodos (1-6), en el escenario montañoso (Aliaga, Teruel).

caciones, se realizaron tres pruebas: (i) volar el dron a una altura de 50 metros (representado en azul en la Figura 2), (ii) elevar el dron a una altura de 120 metros (representado en rojo) e (iii) incrementar la altura del dron a 200 metros (representado en amarillo).

De esta manera, se recopilaron datos utilizando el gateway terrestre, y se examinó si el vuelo del dron a diferentes alturas podría resultar beneficioso para lograr un aumento en el porcentaje de paquetes recibidos exitosamente. En este caso, se empleó un SF de 12, el cual proporciona una mayor cobertura y facilita la comunicación entre los dispositivos y el gateway.

Cabe destacar que a medida que se aumenta la altura del dron, produce un incremento en la distancia entre este y los nodos. Como resultado, las distancias son mayores cuando este se eleva a 50 metros, y se incrementan aún más cuando se eleva a 120 metros, en comparación con las distancias existentes frente al gateway terrestre.

Para elevar el dron a una altura de 200 metros, fue necesario desplazarse a una pequeña elevación, lo que implica que las distancias sean ligeramente diferentes. Un aspecto de interés será determinar si el incremento de altura del dron afectará al porcentaje de paquetes recibidos.

En definitiva, el análisis de los resultados obtenidos en este experimento proporcionará una mayor comprensión sobre el rendimiento de la comunicación de dispositivos LoRa y el gateway en un entorno montañoso, a priori desfavorable. Esto posibilitará obtener información que permita comprender los desafíos y limitaciones que impone la orografía montañosa en una red LoRaWAN, y ayudará a identificar estrategias y soluciones para mejorar la cobertura y la calidad de la comunicación en estos entornos adversos.

Asimismo, se busca investigar si la incorporación de drones puede aportar ventajas sustanciales en términos de mejora de las comunicaciones en este tipo de redes en un entorno montañoso, a priori, desfavorable.

III. Resultados

En esta sección, se muestran los resultados experimentales obtenidos en nuestras pruebas. Se presentan los porcentajes de paquetes recibidos por el gateway provenientes de los nodos desplegados en un entorno de orografía llana, seguidos de los resultados obtenidos en un entorno de orografía montañosa, incluyendo el uso de un dron para la mejora de las comunicaciones. Finalmente, se realizará una comparación entre los resultados de ambos escenarios, exponiendo los aspectos más relevantes.

A. Escenario llano

La Figura 3 muestra la comunicación de los dispositivos LoRa en el escenario con orografía llana. Como se observa, al utilizar un SF de 7, el más bajo, se logra un porcentaje del 100 % de paquetes recibidos con éxito por el gateway para todos los nodos, excepto para el nodo ubicado en el punto 6, donde se obtuvo un 82 % de éxito.

Al cambiar a un spreading factor de 10, se observa que el porcentaje de paquetes recibidos con éxito es del 100 % para todos los nodos, excepto para el nodo situado en la posición 1, donde se registró un 95 %. Dado que este punto es el segundo más cercano al gateway, se deduce que pudo haber alguna interferencia o problema que impidió alcanzar el 100 % de éxito en la comunicación en un momento puntual.

Finalmente, al utilizar un spreading factor de 12, se logra un porcentaje del 100 % de éxito en el envío de paquetes en todos los dispositivos.

Se observa también que, en la prueba realizada en un escenario con orografía llana, no se aprecian diferencias significativas en el porcentaje de paquetes comunicados con éxito por los nodos. En las tres pruebas llevadas a cabo con diferentes spreading factors (SF), la comunicación se ha logrado exitosamente.

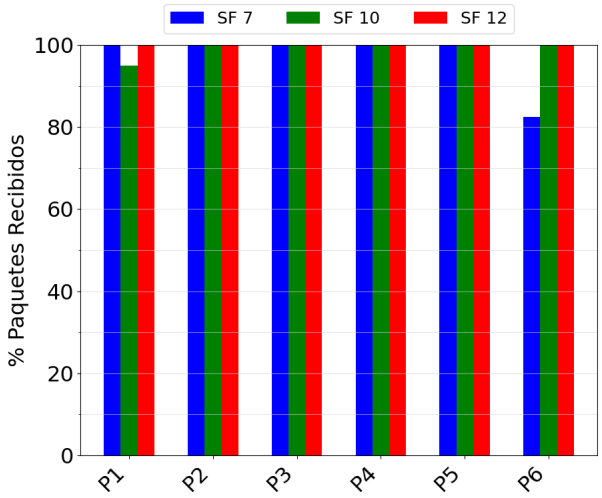


Fig. 3: Porcentaje de paquetes recibidos en el escenario llano.

La única diferencia notable se observa en el nodo ubicado en el punto 6, que es el segundo más alejado del gateway, donde una pequeña elevación ha dificultado la comunicación al utilizar un SF de 7. Sin embargo, al utilizar un SF de 10 y 12, este obstáculo se superó con éxito. Por lo tanto, cuando se presentan dificultades orográficas, parece que utilizar un SF más alto proporciona una mayor fiabilidad en la comunicación, mientras que en terrenos llanos, con las distancias utilizadas, no se aprecian diferencias al modificar el SF.

Un aspecto de gran relevancia en el análisis de los resultados obtenidos consistió en determinar las razones por las cuales el dispositivo ubicado en el punto 5 presentó mejores resultados en comparación con el dispositivo situado en el punto 6, a pesar de que el primero se encontraba a una distancia mayor. Para ello, es fundamental considerar el perfil topográfico del terreno entre el gateway y el punto 6, que se muestra en la Figura 4.

En dicha figura, se puede observar que el nodo situado en el punto 6 se encuentra a una distancia aproximada de 2,2 km del gateway y a una altura de 30 metros con respecto al mismo. Además, se evidencia la existencia de una pequeña elevación que impide una línea de visión directa con el gateway. Esta situación ha representado una dificultad en la comunicación con el gateway, dificultad que ha sido superada aumentando el SF, a pesar de que no se respeta la zona de Fresnel, requisito indispensable según la teoría. En el caso del nodo P5, no existe ningún obstáculo entre el gateway y el nodo, por lo que se reciben correctamente todos los paquetes, a pesar de que se encuentra a distancia bastante mayor del gateway.

B. Escenario montañoso

La Figura 5 muestra la comunicación de los dispositivos LoRa en el escenario con orografía montañosa. Como se observa, en este escenario hubo dos nodos que no lograron establecer comunicación con el gateway terrestre, específicamente los nodos situados en las posiciones 5 y 6. Por lo que en este experimento, un tercio de los nodos no pudieron establecer comunicación con el gateway situado en tierra. La incapacidad de que esos nodos establecieran comunicación con el gateway se atribuye a la ausencia de una línea de visión directa entre los mismos y el gateway. Esta falta de línea de visión impide la formación de la zona de Fresnel necesaria para una comunicación efectiva y, como resultado, se ha visto obstaculizada la transmisión de datos entre ambos dispositivos.

Por otro lado, el uso del dron no fue completamente satisfactorio en el estudio. Al elevar el dron a una altura de 50 metros, se obtuvo una mejora en el número de paquetes recibidos con éxito en un tercio de los nodos, logrando que el número de paquetes recibidos fuera un 7 % mayor para el nodo P1, y un 4 % mayor para el nodo P4, en cuanto a número de paquetes recibidos, en comparación con el gateway terrestre. Sin embargo, esta mejora no se tradujo en una mejora en los dos tercios restantes de los nodos, ya que los nodos P2, P3 y P4 obtuvieron los mismos porcentajes de paquetes recibidos que con el gateway terrestre. Además, no se logró establecer comunicación con los nodos ubicados en las posiciones 5 y 6.

Al elevar el dron a una altura de 120 metros, solo se observó una mejoría en los resultados en uno de los seis nodos desplegados, específicamente en el nodo P1, que es el segundo más cercano en distancia al gateway. Se logró un número paquetes recibidos con éxito un 10 % mayor en comparación con el gateway terrestre, y un 3 % mayor en comparación con los resultados obtenidos al elevar el dron a 50 metros. Sin embargo, no se observó una mejoría en los resultados obtenidos en los nodos P2, P3 y P4. Además, elevar el dron a esa altura tampoco permitió establecer comunicación con los nodos ubicados en las posiciones 5 y 6.

Por último, al elevar el dron a una altura de 200 metros, se obtuvo un aumento de los paquetes recibidos en el nodo P1 de un 10 % en comparación con los resultados obtenidos por el gateway situado en tierra, un 3 % en comparación con los resultados obtenidos al elevar el dron a 50 metros, pero solo mejoró en un 0.3 % los resultados obtenidos al elevar el dron a 120 metros de altura. Lo que destaca es que, por primera vez, se logró establecer comunicación con el nodo P6, obteniendo un 34 % de paquetes recibidos con éxito. Esto es debido a que, al aumentar la altura del dron, se logra superar los obstáculos orográficos y, por lo tanto, permite respetar la zona de Fresnel necesaria para la comunicación

Cabe destacar que el nodo ubicado en la posición 5 no logró establecer comunicación en ningún caso, ni con el gateway terrestre ni con el dron a diferentes alturas. En este caso, había una colina que impedía encontrar una zona de Fresnel que permitiera la comunicación (Ver Figura 6).

C. Comparación de escenarios

La Figura 7 muestra la comparación de los resultados obtenidos en el primer escenario de orografía llana con los obtenidos en el segundo experimento

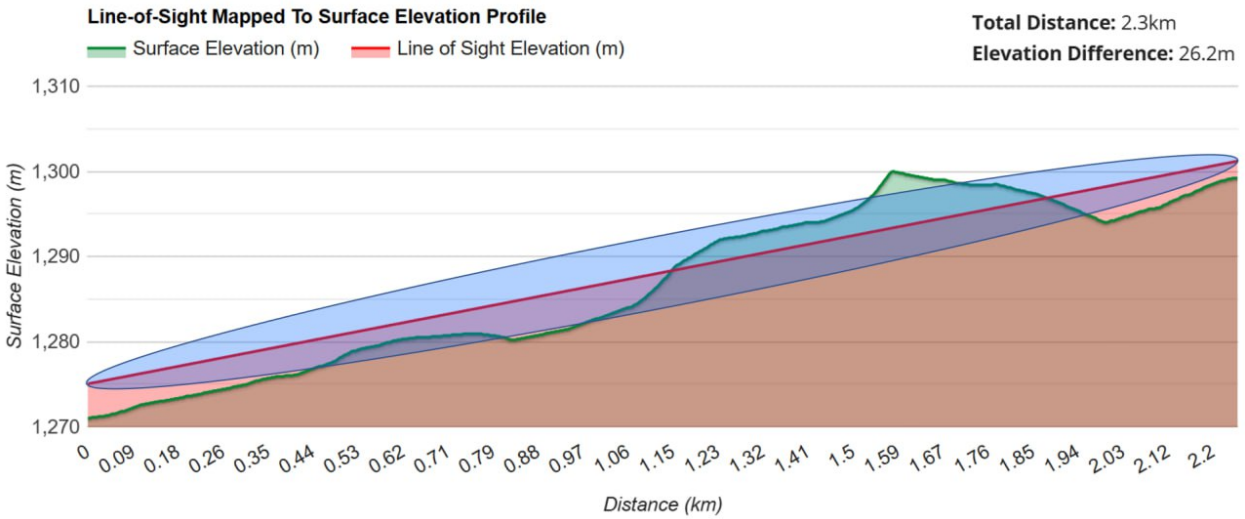


Fig. 4: Perfil topográfico de la distancia entre el gateway y el nodo P6 (escenario llano) [18].

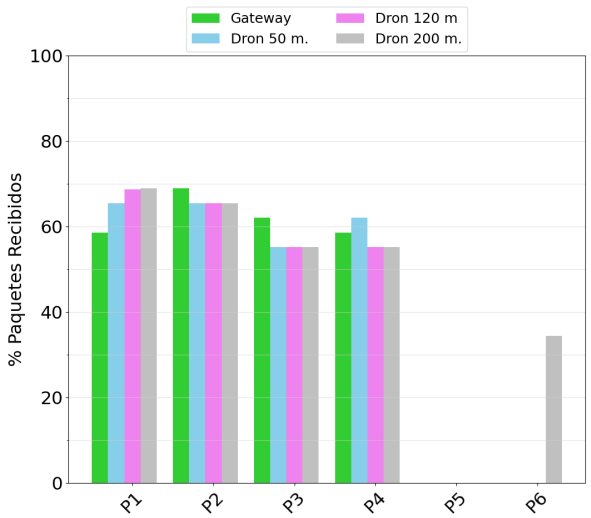


Fig. 5: Porcentaje de paquetes recibidos en el escenario montañoso.

realizado en un entorno montañoso, ya que las distancias para los mismos puntos son equivalentes. Se pueden apreciar diferencias significativas en cuanto al porcentaje de paquetes recibidos por el gateway provenientes de los nodos, según el entorno.

Esto indica que la orografía tiene un efecto significativo en las redes LoRaWAN, observándose una reducción del 58,63 % en la tasa de éxito de recepción de paquetes, en comparación con los datos obtenidos en un entorno plano.

En el escenario llano, la distancia no parece representar un problema al utilizar LoRa, ya que los nodos ubicados a mayor distancia lograron comunicar el 100 % de los paquetes utilizando un SF de 12. Sin embargo, en el escenario montañoso, se observa una reducción considerable en el número de paquetes recibidos. Además, los nodos ubicados a mayor distancia no lograron establecer comunicación con el gateway ubicado en tierra ni con el dron a diferentes alturas.

IV. Conclusiones

Los resultados de este estudio pueden ser beneficiosos a la hora de realizar un despliegue de dispositivos LoRa en entornos reales, ya que proporcionan información sobre la efectividad de esta tecnología en diferentes escenarios, y permite comprender mejor su rendimiento y limitaciones.

En el escenario llano, con las distancias utilizadas, los nodos han podido comunicarse correctamente y no se observa que modificar el SF haya supuesto una diferencia significativa en el porcentaje de paquetes recibidos por el gateway. Sin embargo, la orografía es un factor mucho más determinante que la distancia entre los nodos y el gateway. La presencia de obstáculos orográficos ha reducido el número de paquetes que los nodos han logrado transmitir correctamente en un 58,63 % en comparación con los resultados obtenidos en el escenario llano. También hay que destacar que un tercio de los nodos no ha podido transmitir datos al gateway en este escenario.

Además, la incorporación de un dron en el escenario montañoso ha demostrado ser beneficiosa, al permitir superar los desafíos orográficos que limitaban la comunicación utilizando un gateway terrestre. Al elevar el dron a una altura de 200 metros, se ha logrado establecer comunicación con el nodo P6, logrando una tasa de éxito del 34 % en la transmisión de mensajes. Sin embargo, no se ha obtenido una mejora significativa en las posiciones 2, 3 y 4, e incluso se ha experimentado una ligera disminución en el número de paquetes recibidos con éxito en algunos casos. Esto puede atribuirse al ligero aumento de la distancia al incrementar la altura del dron, así como a posibles interferencias.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Gobierno de Aragón y el Fondo Social Europeo “Construyendo Europa desde Aragón”(Grupo de Investigación T40 23D), por el proyecto de I+D PID2021-122580NB-I00, financiado por MCIN/AEI/10.13039/501100011033 y FEDER “Una ma-

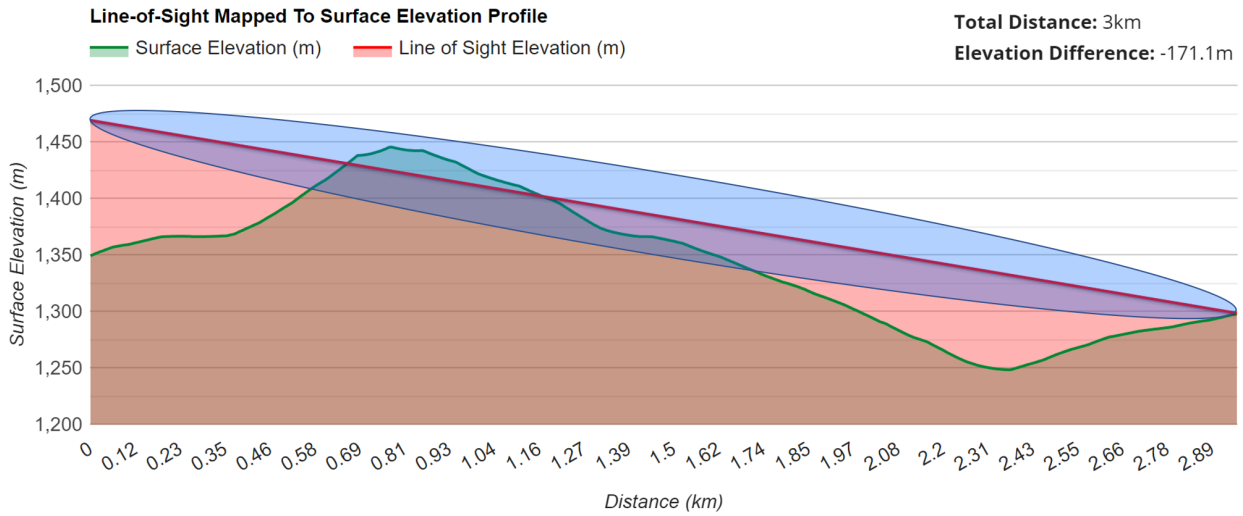


Fig. 6: Perfil topográfico de la distancia entre el gateway y el Nodo P5 (escenario montañoso) [18].

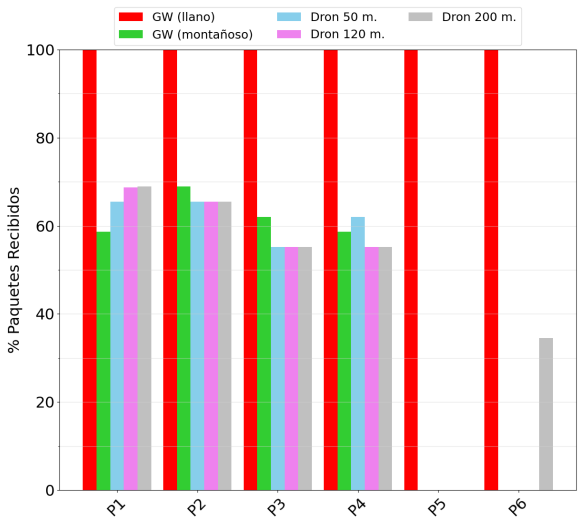


Fig. 7: Comparación del porcentaje de paquetes recibidos en ambos escenarios (SF 12).

nera de hacer Europa", así como por las Ayudas para el desarrollo de Proyectos de investigación 2023 de la Fundación Antonio Gargallo (2022/B003).

Referencias

[1] IHS Markit, *The Internet of Things: a movement, not a market.* Penguin Classics, 2017. [Online]. Available: https://cdn.ihs.com/www/pdf/IoT_ebook.pdf

[2] B. Arratia, P. García-Guillamón, C. T. Calafate, J.-C. Cano, J. M. Cecilia, and P. Manzoni, "A modular and mesh-capable LoRa based content transfer protocol for environmental sensing," in *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, 2023, pp. 378–383.

[3] V. Mateev and I. Marinova, "Distributed Internet of Things system for wireless monitoring of electrical grids," in *2018 20th International Symposium on Electrical Apparatus and Technologies (SIELA)*. IEEE, 2018, pp. 1–3.

[4] A. M. Manoharan and V. Rathinasabapathy, "Smart water quality monitoring and metering using LoRa for smart villages," in *2018 2nd International Conference on Smart Grid and Smart Cities (ICSGSC)*. IEEE, 2018, pp. 57–61.

[5] Y. Cheng, H. Saputra, L. M. Goh, and Y. Wu, "Secure smart metering based on LoRa technology," in *2018 IEEE 4th International Conference on Identity, Security, and Behavior Analysis (ISBA)*. IEEE, 2018, pp. 1–8.

[6] R. O. Andrade and S. G. Yoo, "A comprehensive study of the use of LoRa in the development of smart cities," *Applied Sciences*, vol. 9, no. 22, p. 4753, 2019.

[7] M. Ji, J. Yoon, J. Choo, M. Jang, and A. Smith, "LoRa-based visual monitoring scheme for agriculture IoT," in *2019 IEEE sensors applications symposium (SAS)*. IEEE, 2019, pp. 1–6.

[8] S. Benaissa, D. Plets, E. Tanghe, J. Trogh, L. Martens, L. Vandaele, L. Verloock, F. A. Tuyttens, B. Sonck, and W. Joseph, "Internet of animals: characterisation of LoRa sub-ghz off-body wireless channel in dairy barns," *Electronics Letters*, vol. 53, no. 18, pp. 1281–1283, 2017.

[9] S. W. Prakosa, M. Faisal, Y. Adhitya, J.-S. Leu, M. Köppen, and C. Avian, "Design and implementation of LoRa based IoT scheme for indonesian rural area," *Electronics*, vol. 10, no. 1, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/1/77

[10] (2022) Urbiotica will deploy more than 2.000 U-Spot M2M sensors in the next 6 months. Parking Sensor. [Online]. Available: https://urbiotica.com/en/urbiotica-will-deploy-more-than-2-000-u-spot-m2m\-sensors-in-the-next-6-months/

[11] R. K. Kodali, K. Y. Borra, S. S. GN, and H. J. Domma, "An IoT based smart parking system using LoRa," in *2018 International conference on cyber-enabled distributed computing and knowledge discovery (CyberC)*. IEEE, 2018, pp. 151–1513.

[12] N. Ramli, M. Mun'im Zabidi, A. Ahmad, and I. A. Musliman, "An open source LoRa based vehicle tracking system," *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, vol. 7, no. 2, pp. 221–228, 2019.

[13] S. Sendra, J. L. Garcia-Navas, P. Romero-Diaz, and J. Lloret, "Collaborative LoRa-based sensor network for pollution monitoring in smart cities," in *Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, 2019, pp. 318–323.

[14] A. Sherif, S. Sherif, C. P. Ooi, and W. H. Tan, "A -driven home security system for a residential community in a retirement township," *International Journal of Technology*, vol. 10, no. 7, pp. 1297–1306, 2019.

[15] R. Vega-Rodríguez, S. Sendra, J. Lloret, P. Romero-Díaz, and J. L. Garcia-Navas, "Low cost LoRa based network for forest fire detection," in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE, 2019, pp. 177–184.

[16] S. Venkatraman, R. Varshaa, and P. Vigneshwary, "IoT based door open or close monitoring for home security with emergency notification system using LoRa technology," in *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1. IEEE, 2021, pp. 173–178.

[17] M. Bruhl, G. J. Vermeer, and M. Kiehn, "Fresnel zones for broadband data," *Geophysics*, vol. 61, no. 2, pp. 600–604, 1996.

[18] Solwise Ltd., "Surface Elevation Tool," https://www.solwise.co.uk/wireless-elevationtool.html.

Docencia en Arquitectura y Tecnología de Computadores

Clúpiter: un mini-supercomputador de Raspberry Pis con fines divulgativos

Alonso Rodríguez-Iglesias, María J. Martín y Juan Touriño[1]

Resumen— **El principal objetivo de este trabajo es acercar la supercomputación y el procesamiento paralelo a públicos no especializados mediante la construcción de un clúster formado por Raspberry Pis, al que hemos llamado Clúpiter, que emula el funcionamiento de un supercomputador. Está formado por ocho nodos, ocho Raspberry Pis interconectadas entre sí para que puedan ejecutar trabajos en paralelo. Para que sea más sencillo mostrar cómo funciona, se ha desarrollado una aplicación web desde la que se pueden lanzar aplicaciones paralelas y acceder a un sistema de monitorización que permite ver el uso de los recursos cuando se están ejecutando. Desde dicha aplicación web también se puede acceder a un par de vídeos educativos que tratan de manera muy divulgativa los conceptos de supercomputación y programación paralela.**

Palabras clave— **Supercomputador, programación paralela, MPI, Raspberry Pi**

I. Introducción

Los supercomputadores son equipos informáticos que están compuestos por cientos o miles de procesadores, junto con cantidades ingentes de memoria, para ofrecer una elevada velocidad y capacidad de cálculo y de procesamiento de datos. Sin embargo, estas máquinas y sus mecanismos y procesos suelen quedar muchas veces fuera de la comprensión del público general. Por eso, el objetivo de este trabajo es acercar este ámbito de la informática a personas ajenas al mismo.

Para ello, se ha construido un pequeño clúster con Raspberry Pis con el nombre de Clúpiter (mezcla de las palabras Clúster y Pi) que pretende ser una réplica a pequeña escala de un supercomputador.

La Raspberry Pi es un computador de placa reducida de bajo coste y consumo usado muy habitualmente en entornos educativos [1], [2]. En concreto, para la construcción de Clúpiter se ha utilizado la Raspberry Pi 4B, la versión más nueva del formato *Standard* que estaba disponible en el momento de la realización de este trabajo.

Las placas de Raspberry Pis se han conectado y configurado para que puedan trabajar de forma colaborativa como si se tratase de un único equipo, se ha evaluado su rendimiento y se ha desarrollado una aplicación web específica que ayuda a demostrar cómo se pueden utilizar los supercomputadores para acelerar la ejecución de aplicaciones computacionalmente costosas.

El trabajo está estructurado de la siguiente manera. En la sección II se hace un breve resumen de los trabajos relacionados. En la sección III se expone el análisis de requisitos realizado, así como las decisiones de diseño hardware y software y el proceso de configuración de la infraestructura resultante. En la sección IV se lleva a cabo una evaluación de rendimiento, utilizando para ello los NAS Parallel Benchmarks (NPBs) [3]. La sección V describe la aplicación web desarrollada para ayudar a explicar el funcionamiento de Clúpiter. El artículo finaliza exponiendo las conclusiones.

[1]Universidade da Coruña, CITIC, Grupo de Arquitectura de Computadores, e-mail: {alonso.rodriguez,mariam,juan}@udc.es.

II. Trabajo relacionado

Las placas de Raspberry Pis incluyen todos los circuitos esenciales, tales como CPUs (*Central Processing Unit*), GPUs (*Graphical Processing Unit*) y circuitos de entrada y salida, lo que las hacen muy apropiadas para ser utilizadas en proyectos educativos relacionados con la informática. Se han utilizado, por ejemplo, para enseñar tópicos como el tratamiento de imágenes [4], el procesamiento de señal [5], algoritmos de control en tiempo real [6] o ciberseguridad [7].

Más vinculado con la temática de este artículo, existen en la bibliografía experiencias previas en las que se han utilizado varias Raspberry Pis para construir clústeres de bajo coste y consumo que ayuden a entender los conceptos relacionados con la computación de altas prestaciones. Algunos ejemplos son el clúster *Iridis-pi* [8], que consiste en 64 Raspberry Pis conectadas a través de Ethernet y alojadas en un chasis construido con bloques Lego; la infraestructura *Wee Archie* (`https://www.archer2.ac.uk/community/outreach/materials/wee_archie`), construida con 18 Raspberry Pis empaquetadas en una caja transparente y que pretende ser la versión reducida del supercomputador del EPPC (Edinburgh Parallel Computer Center) ARCHER2; o el proyecto *cluster coffer* [9], 16 Raspberry Pis ubicadas en un maletín metálico portátil e interconectadas por una red Gigabit Ethernet.

Las propuestas anteriores se diferencian fundamentalmente en el aspecto final del clúster y la información de monitorización que se puede obtener durante la ejecución de aplicaciones paralelas. En este sentido, los puntos fuertes de Clúpiter son dos: su organización, ya que su hardware ha sido ensamblado para emular la estructura de un supercomputador real; y su aplicación web, que permite la ejecución y monitorización de forma sencilla y visual de todas las aplicaciones de los NAS Parallel Benchmarks (NPBs) [3].

Fuera del ámbito educativo, los clústeres de Raspberry Pis han sido utilizados para un gran número de

Tabla I: Coste total de Clúpiter

Uds.	Material	Coste (€)
8	Raspberry Pi 4B	392,00
1	Switch Gigabit	33,60
1	USB Ethernet	23,00
2	Torres RPI	54,00
1	Ventilador	18,00
1	Fuente alimentación	27,00
8	MicroSD 32GB	104,00
10	Cables USB-C magnéticos	26,16
1	Step-up MT3608	1,79
Total		**679,55**

aplicaciones diferentes. Por mencionar algunos ejemplos, en [10] se utilizan para ejecutar algoritmos de minería de datos, en [11] para acelerar la ejecución de un algoritmo de marca de agua paralelo, o en [12] para monitorizar terrenos agrícolas.

III. Diseño de Clúpiter

Antes de decidir los componentes hardware y software del clúster se lleva a cabo un análisis de requisitos llegando a las siguientes conclusiones:

- Debe ser **pequeño** y **manejable**, descartando estructuras donde los nodos queden "libres" y "desperdigados".
- Debe ser **visualmente agradable** y **comprensible**, con partes fácilmente identificables, lo más aisladas y señalables posible.
- Todos los nodos deben estar conectados entre sí en una **topología N a N**, es decir, la típica de un switch Ethernet.
- Debe ser capaz de ejecutar **aplicaciones MPI** [13] genéricas.
- Debe ser sensato, empleando una calidad y cantidad de materiales adecuada a las expectativas del mismo y utilizando componentes actuales con una **buena relación calidad/precio**.

A continuación se describen los componentes hardware y software que se han seleccionado para cumplir con esos requerimientos y la configuración que se ha llevado a cabo de los mismos.

A. Componentes hardware

La Tabla I resume los componentes hardware empleados y su coste (sin IVA).

Se ha elegido como nodos del clúster ocho Raspberry Pis 4B. La CPU de este modelo es la Broadcom BCM2711, un procesador con arquitectura ARMv8-A y 4 núcleos Cortex-A72, que funcionan a 1,5 GHz y cuentan con 2 GB de memoria. La elección de esta placa se ha basado en su reducido formato y su bajo consumo y coste, lo que lo hace una solución idónea para este trabajo, que no requiere de hardware muy potente, sino más bien de mucho hardware poco potente que permita simular la estructura de un supercomputador. Para cada Raspberry Pi se ha adquirido una tarjeta MicroSD, necesaria para almacenar el sistema operativo.

Los componentes del clúster reciben la energía necesaria para su funcionamiento de una fuente de alimentación de 5V y 30A, resultando en una potencia de 150W, que triplica los requisitos energéticos mínimos, y duplica los recomendados.

Para conectar los nodos se ha utilizado un switch Gigabit Ethernet de 8 puertos que funciona a 5V, lo que permite conectarlo directamente a la fuente de alimentación. Hay que tener en cuenta que el clúster debe poder ser accedido desde el exterior, por lo que en realidad se necesitan 9 puertos. Para solucionar este inconveniente se añade una adaptador USB 3.0 a Gigabit Ethernet que se conecta al nodo maestro de Clúpiter y es puenteado con la interfaz interna, creando así un switch virtual de 9 puertos (ver Figura 1).

Las Raspberry Pis se apilan verticalmente en dos torres de 4 placas cada una para poder ser cableadas, accedidas y refrigeradas de forma sencilla. Entre las dos torres se ubica un ventilador de 120 mm, que si bien no es imprescindible a efectos de disipación de calor, es interesante para realizar analogías con la importancia de la refrigeración en los supercomputadores reales. El ventilador elegido funciona a 12V, sin embargo, alimentarlo a esa tensión haría que funcionase a máxima potencia constantemente. Para evitar esto, se emplea un *step-up* variable con el que se mantiene el ventilador a una velocidad fija y lo más baja posible para rebajar el nivel de ruido. La conexión eléctrica de estos componentes se puede observar en la Figura 2.

El resultado final del ensamblaje de todos estos módulos hardware se puede observar en el *render* de la Figura 3. Sobre esa imagen se pueden identificar las múltiples zonas del chasis. Comenzando por la zona inferior tenemos la fuente de alimentación y las conexiones de los cables de corriente continua y alterna. En la zona superior se encuentra el switch, que interconecta los ocho dispositivos entre sí a 1 Gbps en modo *full duplex*. La zona intermedia está ocupada las dos torres de Raspberry Pis, orientadas con las interfaces de entrada/salida hacia fuera, y que son refrigeradas por el ventilador que las atraviesa. La Figura 4 muestra una foto real de Clúpiter y sus conexiones.

B. Componentes software

Sobre el hardware descrito en la sección anterior se instaló el sistema operativo Arch Linux, una distribución de Linux ligera y flexible que dispone de una amplia documentación en la Arch Wiki (`https://wiki.archlinux.org`).

Para poder ejecutar códigos paralelos sobre Clúpiter utilizaremos MPI (Message Passing Interface) [13], el estándar de facto para la programación de los sistemas paralelos de memoria distribuida. En este trabajo se emplea una implementación libre de MPI, OpenMPI, ya que es la librería MPI que ofrece Arch Linux en sus repositorios oficiales.

Adicionalmente, se desarrolló una aplicación web para monitorizar el estado e histórico del clúster en

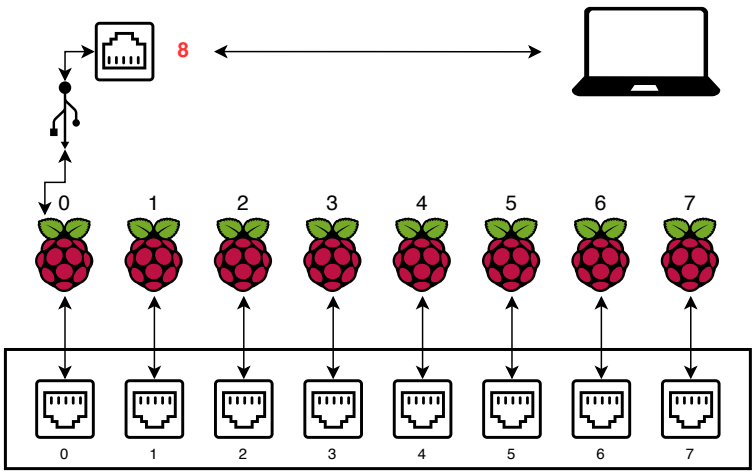


Fig. 1: Esquema físico de red de Clúpiter

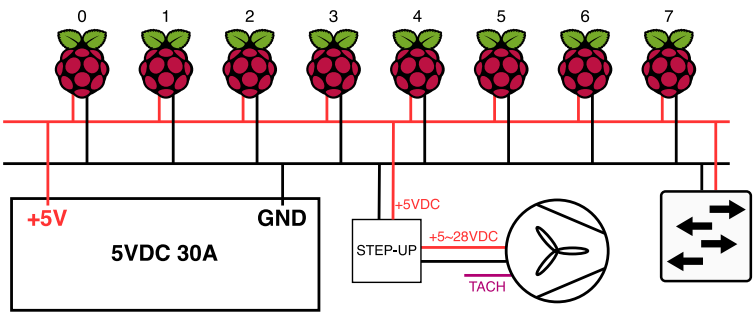


Fig. 2: Diagrama eléctrico de Clúpiter

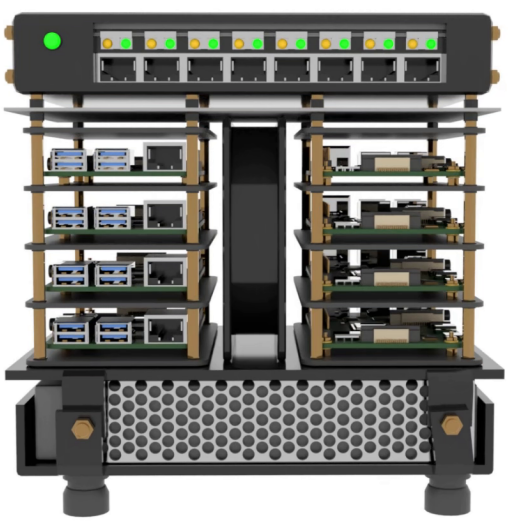


Fig. 3: Infografía de Clúpiter

Fig. 4: Foto de Clúpiter y sus conexiones

tiempo real, y poder realizar explicaciones interactivas acerca del funcionamiento de los programas MPI. Esta aplicación web se explicará en detalle en la Sección V.

C. Configuración del clúster

Para poder administrar el clúster, es necesario que a cada nodo se le asigne una dirección IP. Debido a las características del mismo, y especialmente a que la conexión desde el exterior va a ser variable y no siempre va a estar disponible, se configuran direcciones IP estáticas en cada uno de los nodos, así como una dirección de *gateway* predeterminada, que algunas veces estará activa y otras no.

Para poder lanzar aplicaciones MPI se crea una cuenta de usuario denominada `mpiuser`. Además, se configura la autenticación no interactiva para SSH mediante el uso del par de claves pública y privada. La autenticación no interactiva por clave privada es necesaria para que el usuario `mpiuser` pueda establecer conexiones SSH de forma desatendida con el resto

de nodos durante la ejecución de programas MPI.

Finalmente, debido a que las llamadas al comando `mpirun` para la ejecución de trabajos MPI en múltiples nodos ejecutan el mismo comando en todos ellos, debe existir algún tipo de almacenamiento compartido montado con el mismo nombre en todos los nodos. Para satisfacer esta necesidad de almacenamiento compartido elegimos NFS (*Network File System*). El servidor será el nodo maestro y el resto serán clientes.

En [14] se encuentra documentado en mayor detalle todo el proceso de configuración, especificando todos los comandos Linux utilizados para ello.

IV. Evaluación del rendimiento

Una vez puesto en marcha el clúster, se comprueba su capacidad para ejecutar aplicaciones MPI y su rendimiento. Si bien el rendimiento no es una prioridad, es conveniente realizar estas pruebas, especialmente para poder observar el impacto que tiene la red de comunicaciones entre los núcleos de una sola CPU (recordemos que cada una tiene 4 núcleos), o entre múltiples CPUs y memorias. Para ello se utilizan los NAS Parallel Benchmarks (NPBs) [3], un conjunto de tests de cálculo numérico diseñados por la División de Supercomputación de la NASA para la medida del rendimiento de supercomputadores. Las Figuras 5, 6 y 7 muestran los resultados obtenidos, expresados en millones de operaciones por segundo (MOPS), para un subconjunto representativo de las aplicaciones. Los resultados obtenidos con las otras aplicaciones de los NPBs se pueden consultar en [14].

En concreto, se muestran los resultados para:

- CG (Conjugate Gradient): Resuelve sistemas de ecuaciones lineales de matrices simétricas y definidas positivas utilizado el método iterativo del gradiente conjugado.
- EP (Embarrassingly Parallel): Contiene un *kernel* masivamente paralelo que sirve para proporcionar una estimación de los límites del rendimiento en punto flotante.
- IS (Integer Sort): Ordena números enteros. Es útil para comprobar tanto la velocidad de cómputo con enteros como el rendimiennto de las comunicaciones.

Las tres aplicaciones se han ejecutado con clase C, se han realizado 5 ejecuciones de cada una y se ha calculado la media. La línea vertical discontinua de las gráficas señala la transición de ejecución en un solo nodo (utilizando los 4 núcleos) a ejecución en múltiples nodos. Las comunicaciones intranodo se realizarán vía memoria compartida, mientras que las comunicaciones internodo utilizarán la red Gigabit Ethernet.

Como se puede apreciar, los resultados son excelentes para EP que, por su naturaleza, permite obtener un muy buen rendimiento en punto flotante. Por otro lado, se nota un fuerte impacto al ejecutar benchmarks que hacen un uso intensivo de las comunicaciones entre nodos. Esto se puede apreciar en

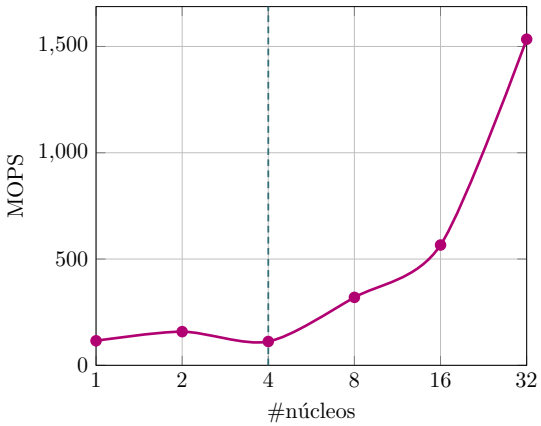


Fig. 5: Rendimiento para el kernel CG

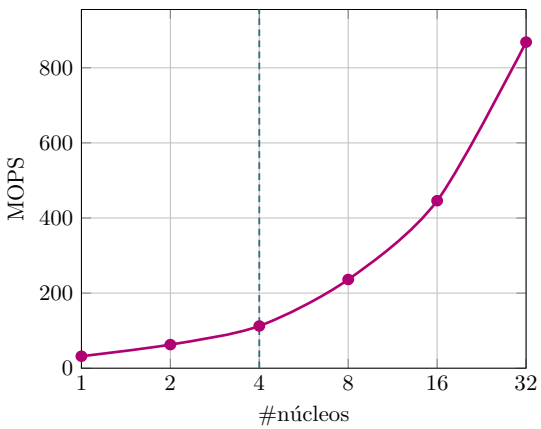


Fig. 6: Rendimiento para el kernel EP

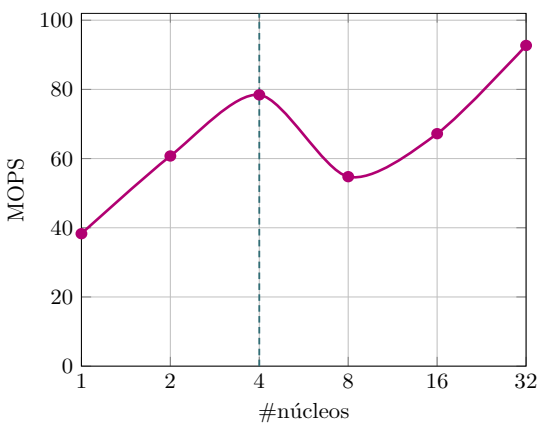


Fig. 7: Rendimiento para el kernel IS

la gráfica de IS, donde el rendimiento cae de forma pronunciada en la ejecución con dos nodos (es decir, 8 núcleos). En cuanto a CG, se puede ver que no solo no se obtiene un mayor rendimiento pasando de dos a cuatro núcleos, sino que se pierde. Este efecto es debido al pequeño ancho de banda que ofrece el único chip de memoria de cada Raspberry Pi. A pesar de este comportamiento en las ejecuciones en un único nodo, cuando se ejecutan los kernels en memoria distribuida, el rendimiento sí que aumenta. En algunos casos, incluso, más que duplicando el rendimiento anterior, como por ejemplo ocurre al pasar de 16 a 32 núcleos en CG y EP.

V. Aplicación web

El objetivo final de este trabajo es construir una réplica a pequeña escala de un supercomputador que sirva para explicar el funcionamiento de un sistema paralelo, así como realizar demostraciones prácticas, tanto en directo como mediante vídeos, de su funcionamiento. Para ello se ha desarrollado una aplicación web desde la cual se podrán ejecutar y monitorizar los benchmarks discutidos en la sección anterior, así como visualizar vídeos divulgativos sobre la computación paralela.

Se ha elegido implementar esta funcionalidad como una aplicación web porque proporciona universidalidad (nos podremos conectar desde cualquier operativo) y sencillez. Para ejecutarla solamente será necesario conectarse mediante un navegador a la dirección del nodo maestro, que es el que aloja la aplicación.

La tecnología sobre la que se programa el *backend* de la aplicación web es *nodejs*, en concreto *expressjs* (`https://expressjs.com/es/`), y se hace uso de las APIs de *Netdata* y *socket.io*. La elección de *socket.io* se realiza debido a la sencillez y elegancia que aporta al código, ya que desde la propia aplicación web se pueden hacer llamadas al *backend* con el mismo estilo de programación que en Android, resultando familiar y sencillo. Por otro lado, Netdata [15] es un software de monitorización gratuito y de código abierto, de fácil configuración e integración, que permite obtener información de los nodos de Clúpiter en tiempo real. Para emplear los datos que proporciona este software, primero debe instalarse y activarse en cada uno de los nodos del clúster.

La aplicación web dispone de 3 pestañas: "Inicio", "Monitorización" y "Acerca de". Desde la pestaña de "Inicio" se puede acceder a dos vídeos didácticos desarrollados específicamente para este proyecto. El primero de ellos dura unos 6 minutos e introduce, de manera muy divulgativa, el concepto de supercomputador, aportando contexto histórico y cifras acerca de la capacidad de estas infraestructuras, así como de sus utilidades y logros. Tras ello, se realiza una descripción de Clúpiter y sus partes fundamentales, y se relacionan con sus análogos en un supercomputador real. El segundo vídeo, ya más técnico, dura unos 7 minutos y en él se introduce, mediante animaciones ilustradas a mano, el funcionamiento interno de un supercomputador, su necesidad en el día a día, y las restricciones que tienen que superar los programas paralelos, utilizando ejemplos sencillos y evitando los tecnicismos. Tras ello, se proporciona una breve introducción a las operaciones más básicas de MPI. Ambos vídeos se han alojado y subtitulado en YouTube y pueden reproducirse desde las direcciones: `https://youtu.be/o76-VP6WFCo` y `https://youtu.be/if0MWI_9xzM`.

La pestaña de "Monitorización" da acceso a una página desde la que se pueden ejecutar cada uno de los benchmarks de los NPBs (clase `C`), y observar su salida por terminal, así como ver en tiempo real el impacto que la ejecución de los mismos tiene sobre la carga en los nodos y el tráfico de red. Esta página de monitorización se muestra en la Figura 8. En la parte superior se encuentran los medidores de uso de CPU y el tráfico de red de cada una de las Raspberry Pis. En la parte inferior se encuentran los botones para ejecutar los diferentes programas de prueba y poder observar en tiempo real el impacto sobre los diferentes nodos. En concreto, la figura está mostrando la salida durante la ejecución del kernel FT (Fast Fourier Transform). Gracias a esta monitorización se puede observar que durante la ejecución de esta aplicación todos los nodos trabajan en paralelo y se van alternando las fases de comunicación y computación.

Finalmente, en la pestaña "Acerca de" se incluye, entre otra información, una breve descripción de Clúpiter.

VI. Conclusiones

En este trabajo se ha presentado Clúpiter, un supercomputador en miniatura con todas sus secciones bien diferenciadas y extrapolables a un supercomputador real.

A pesar de que el rendimiento y la escalabilidad no son buenos (las Raspberry Pis no están diseñadas para ser un computador especialmente eficaz en este sentido), Clúpiter permite ejecutar programas paralelos y comprobar cómo son capaces de cooperar los diferentes componentes hardware del sistema para acelerar la ejecución de aplicaciones científicas y de ingeniería. Al fin y al cabo, esto es lo que se perseguía como objetivo final de este proyecto.

Clúpiter está actualmente disponible en la "Sala Demostrador Tecnológico"del Centro de Investigación en Tecnologías de la Información y las Comunicaciones (CITIC, `https://citic.udc.es/demostrador-tecnologico/`) de la Universidade da Coruña como herramienta divulgativa, y se utiliza en las visitas al centro de, entre otro público, estudiantes de secundaria.

Todos los recursos relacionados con este trabajo, incluido el código de la aplicación web, se encuentran disponibles en la dirección `https://github.com/forcegk/GEI_TFG` bajo licencia MIT.

Agradecimientos

Clúpiter ha sido financiado por el Grupo de investigación en Arquitectura de Computadores de la Universidade da Coruña (`https://gac.udc.es`).

Referencias

[1] Branko Balon and Milenko Simić, "Using Raspberry Pi computers in education," in *42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2019, pp. 671–676.

[2] Narasimha Saii Yamanoor and Srihari Yamanoor, "High quality, low cost education with the Raspberry Pi," in *2017 IEEE Global Humanitarian Technology Conference (GHTC)*, 2017, pp. 1–5.

[3] NASA, "NAS Parallel Benchmarks," `https://www.nas.nasa.gov/software/npb.html`, Last updated: January 27, 2022.

[4] Julien Marot and Salah Bourennane, "Raspberry Pi for image processing education," in *2017 25th European Signal Processing Conference (EUSIPCO)*, 2017, pp. 2364–2366.

Fig. 8: Página de monitorización de Clúpiter

[5] Gianni Pasolini, Alessandro Bazzi, and Flavio Zabini, "A Raspberry Pi-based platform for signal processing education [SP Education]," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 151–158, 2017.

[6] Jaroslav Sobota, Roman PiŜl, Pavel Balda, and MiloŜ Schlegel, "Raspberry Pi and Arduino boards in control education," *IFAC Proceedings Volumes*, vol. 46, no. 17, pp. 7–12, 2013.

[7] Phil Legg, Alan Mills, and Ian Johnson, "Teaching offensive and defensive cyber security in schools using a Raspberry Pi cyber range," *Journal of The Colloquium for Information Systems Security Education*, vol. 10, no. 1, 2023, 9 pages.

[8] Simon J Cox, James T Cox, Richard P Boardman, Steven J Johnston, Mark Scott, and Neil S O'Brien, "Iridis-pi: a low-cost, compact demonstration cluster," *Cluster Computing*, vol. 17, pp. 349–358, 2014.

[9] Philipp Gschwandtner, Alexander Hirsch, Peter Thoman, Peter Zangerl, Herbert Jordan, and Thomas Fahringer, "The cluster coffer: teaching HPC on the road," *Journal of Parallel and Distributed Computing*, vol. 155, pp. 50–62, 2021.

[10] João Saffran, Gabriel Garcia, Matheus A Souza, Pedro H Penna, Márcio Castro, Luís FW Góes, and Henrique C Freitas, "A low-cost energy-efficient Raspberry Pi cluster for data mining algorithms," in *Euro-Par 2016: Parallel Processing Workshops*. Springer, 2017, pp. 788–799.

[11] Khalid M Hosny, Amal Magdi, Nabil A Lashin, Osama El-Komy, and Ahmad Salah, "Robust color image watermarking using multi-core Raspberry Pi cluster," *Multimedia Tools and Applications*, vol. 81, no. 12, pp. 17185–17204, 2022.

[12] Agraj Aher, Janhavi Kasar, Palasha Ahuja, and Varsha Jadhav, "Smart agriculture using clustering and IOT," *International Research Journal of Engineering and Technology*, vol. 5, no. 03, pp. 4065–4068, 2018.

[13] Message Passing Interface Forum, "MPI: A message-passing interface standard version 4.0," `https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf`, June 2021.

[14] Alonso Rodríguez-Iglesias, "Diseño e implementación hardware y software de un mini-supercomputador con Raspberry Pi," `https://github.com/forcegk/GEI_TFG/releases/download/memoria/MemoriaTFG.pdf`, 2021, Trabajo Fin de Grado, Universidade da Coruña.

[15] Netdata, Inc., "Monitoring everything in real time for free," `https://www.netdata.cloud`.

Propuesta docente: Evaluación de arquitecturas de VoIP

José Luis Ávila Jiménez[1] y Manuel Agustín Ortíz López[1]

Resumen— **En este trabajo se presenta una propuesta didáctica con un doble objetivo: aumentar el conocimiento del alumnado sobre el servicio VoIP y enseñarles una metodología de evaluación de servicios en red. Se describen el servicio y las herramientas utilizadas, las métricas de evaluación y benchmarking a las que se han obtenido de un servidor VoIP de pruebas. Finalmente, se proporciona un análisis de los resultados obtenidos y se presentan las conclusiones.**

Palabras clave— **VoIP, propuesta didáctica, evaluación, benchmark**

I. Introducción

El grado en ingeniería informática se imparte en la Escuela Politécnica Superior de Córdoba de la Universidad de Córdoba desde el año 2011. La actual configuración del plan de estudios se publicó en BOE en el año 2015 [1]. La asignatura de Configuración y Evaluación de Sistemas Informáticos se imparte en el tercer curso del Grado de Ingeniería Informática y las competencias que en ella se desarrollan se enfocan en aumentar la capacidad del estudiante para reunir e interpretar datos relevantes, comunicar información y soluciones a públicos especializados y no especializados, tener conocimiento profundo de la estructura y funcionamiento de sistemas informáticos y aplicar este conocimiento para solucionar problemas, analizar y evaluar arquitecturas de computadoras, y crear nuevos desarrollos tecnológicos en el campo de la informática [2].

En la configuración y evaluación de un sistema informático, las tareas de monitorización y referenciación son fundamentales. La monitorización implica la recopilación y análisis de datos en tiempo real sobre el rendimiento del sistema, mientras que la referenciación implica la comparación de datos de rendimiento con los de sistemas de referencia o benchmarks. Los índices clásicos de comparación de rendimiento y los benchmarks son herramientas comunes utilizadas en la referenciación y comparación de medidas de rendimiento [3]. Todo esto es esencial para evaluar el rendimiento de sistemas informáticos y optimizar su funcionamiento.

Se ha observado que, a menudo, las competencias mencionadas suelen impartirse de manera aislada. Esta fragmentación puede dificultar la comprensión de los estudiantes sobre cómo se interrelacionan los conceptos, las tecnologías y herramientas en un sistema real. Este trabajo surge del deseo de proporcionar a los estudiantes una experiencia de aprendizaje más integrada y práctica. Se espera que los estudiantes no sólo adquieran un conocimiento más profundo de cada una de estas áreas, sino que también sean capaces de aplicar de manera más efectiva lo que han aprendido en situaciones del mundo real.

En este contexto se presenta esta propuesta docente que busca que el alumno adquiera las competencias anteriormente descritas a la vez que profundiza en las técnicas de monitorización y referenciación mediante el estudio de un servicio muy empleado a día de hoy como es el de Voz sobre IP.

VoIP (*Voice over Internet Protocol*, Voz sobre Protocolo de Internet) es un servicio que permite la transmisión de voz a través de una red basada en IP. Esto implica que los usuarios pueden utilizar su propia red privada para gestionar servicios de comunicación y telefonía. En otras palabras, VoIP permite a los usuarios un mayor control y flexibilidad en la gestión de sus servicios de comunicación [4].

Para estudiar en detalle el servicio de VoIP, se han identificado los componentes básicos de la carga y se han elegido parámetros característicos para medir el rendimiento de los servidores de VoIP. Los componentes elegidos son la RAM, la CPU y la red, y los parámetros específicos son las llamadas, los paquetes perdidos y la fluctuación temporal. Estos parámetros se utilizan para evaluar la calidad del servicio. Se ha realizado una comparación entre dos equipos Intel de dos generaciones distintas para evaluar su rendimiento, y también se ha probado el comportamiento de una Raspberry para dar servicio de llamadas VoIP.

El trabajo se divide en los siguientes apartados: en primer lugar, en la sección II, se introduce el concepto de Servidores de VoIP. A continuación, en la sección III, se definen las principales métricas de rendimiento utilizadas en este tipo de servicios. Posteriormente se presentan las opciones de Benchmarking en la sección IV. Las arquitecturas utilizadas para la comparación y las pruebas realizadas se definen en la sección V. Finalmente se presentan los resultados en la sección VI. Una vez planteada toda la propuesta, en la sección VII se detallan algunas algunas directrices sobre temas como temporización, evaluación y objetivos mínimos que ayuden al profesorado que quiera implantar la propuesta, para acabar el análisis con las conclusiones obtenidas en la sección VIII.

II. VoIP

Un servicio de VoIP es una tecnología que permite la transmisión de voz sobre una red basada en IP. Esto significa que en lugar de utilizar líneas telefónicas tradicionales, se utilizan conexiones de internet para realizar y recibir llamadas telefónicas [4].

Una de las principales ventajas de utilizar VoIP es

[1]Dept. Ing. Electrónica y de Computadores, Universidad de Córdoba, e-mail: {jlavila,el1rolom}@uco.es.

la reducción de costes en comparación con los servicios de telefonía tradicional. Al utilizar la infraestructura de internet existente, se eliminan los costos de mantenimiento y operación de líneas telefónicas, lo que puede resultar en ahorros significativos para las organizaciones.

Además, VoIP ofrece una mayor escalabilidad, lo que significa que se puede aumentar o disminuir la cantidad de líneas telefónicas según sea necesario, sin la necesidad de hacer cambios físicos en la infraestructura de telecomunicaciones.

Otra ventaja de VoIP es que permite un mejor aprovechamiento del ancho de banda disponible, ya que se pueden utilizar las mismas conexiones de internet para transmitir voz, vídeo y datos. Esto significa que se pueden realizar videollamadas y conferencias en línea, así como también compartir archivos y otros tipos de información en tiempo real [4].

Es cierto que los servicios de VoIP también presentan algunas desventajas y limitaciones. Una de las principales desventajas es que la calidad de la llamada puede verse afectada si la red no está bien dimensionada o si hay problemas en la conexión. La transmisión de voz sobre IP se basa en la división de la información en paquetes que se envían a través de la red, y si algunos de estos paquetes se pierden o llegan con retraso, la calidad de la llamada puede verse afectada.

Sin embargo, estas desventajas pueden ser mitigadas con la implementación de una red adecuada y bien dimensionada. También existen soluciones y tecnologías que pueden ayudar a mejorar la calidad de la voz sobre IP, como la priorización de paquetes y la gestión de ancho de banda [5]. La tabla I resume las principales ventajas e inconvenientes del uso de VoIP.

Tabla I: Ventajas e inconvenientes del uso de de VoIP.

Ventajas	*Inconvenientes*
-Reducción de costes	-Calidad de llamada
-Mayor escalabilidad	
-Aprovechamiento de ancho de banda	

En cuanto a los pilares fundamentales del desarrollo de la voz sobre IP, el protocolo SIP (Session Initiation Protocol) es una tecnología clave que se utiliza para establecer, modificar y terminar sesiones de comunicación en una red IP [6]. Las aplicaciones de software libre como *Asterisk*, son herramientas muy utilizadas en la implementación de soluciones de telefonía IP basadas en PBX (Private Branch Exchange), permitiendo a las empresas tener su propia centralita telefónica y gestionar sus comunicaciones internas y externas de forma eficiente y económica [7].

A. Conceptos básicos en VoIP

Para poner en contexto este tipo de servicios de VoIP y entender algunos de los aspectos de configuración que se presentan en apartados posteriores, es necesario presentar algunos conceptos:

Un PBX (Private Branch Exchange) es un dispositivo que se utiliza para conectar los terminales telefónicos de una empresa o una organización de forma independiente a los proveedores de telefonía. Con el desarrollo de las tecnologías de VoIP, surgen las centralitas IPBX (Intranet PBX), que permiten la transmisión de la voz sobre redes de datos en lugar de las redes telefónicas analógicas. Las centralitas IPBX ofrecen muchas ventajas, como una mayor flexibilidad, una mayor escalabilidad y la posibilidad de integrar servicios de voz y datos en una misma red. Además, las centralitas IPBX pueden ser administradas de forma remota a través de una interfaz web, lo que facilita su gestión y configuración. Estas centralitas pueden ser físicas o virtuales, y pueden ser implementadas en una empresa o en la nube [4].

Para la transmisión de VoIP, se pueden identificar tres elementos fundamentales en su estructura [7].

- *Terminales:* Son los puntos finales de comunicación y pueden encontrarse de dos tipos, Hardware (teléfono con soporte de VoIP nativo) y Software, también conocidos como softphone (aplicación ejecutable desde pc o dispositivo móvil que se comunica con las PBX a través de la LAN).
- *Servidor:* Permite el manejo y funciones administrativas para soportar el enrutamiento de llamadas a través de la red. Este servidor puede adoptar diferentes nombres dependiendo del protocolo de señalización utilizado. Encontramos servidores basados en el protocolo H.323 y tenemos un servidor tipo GateKeeper o en sistema SIP que se conocen servidores SIP.
- *Puerta de enlace (gateway):* Enlace de la red VoIP con la red telefónica analógica o RDSI. Adapta las señales de estas redes a VoIP y viceversa de forma totalmente transparente al usuario. Es habitual encontrar juntos al Servidor y puerta de enlace.

En VoIP, existen dos tipos de protocolos principales: los protocolos de señalización y los protocolos de transporte [6].

Los protocolos de señalización son los encargados de establecer, modificar y finalizar las llamadas. El protocolo de señalización más utilizado es el Session Initiation Protocol (SIP), que permite la comunicación entre dos o más dispositivos y define cómo se establecen y finalizan las llamadas [8]. La tabla II muestra los métodos básicos definidos en la RFC 254. Otro protocolo de señalización común es el H.323, que también permite la comunicación de voz y vídeo entre dispositivos.

Por otro lado, los protocolos de transporte se encargan de la transmisión de los datos de voz en tiempo real. El protocolo de transporte más utilizado es el User Datagram Protocol (UDP), que proporciona una conexión no fiable y sin confirmación de entrega de los datos. También se utiliza el Transmission Control Protocol (TCP), que proporciona una conexión

Tabla II: Métodos básicos SIP (RFC254).

INVITE	Permite invitar a un usuario o servicio a participar en una sesión o a modificar parámetros en una sesión existente.
ACK	Confirma el establecimiento de una sesión.
OPTION	Solicita información sobre las capacidades de un servidor.
BYE	Indica la terminación de una sesión.
CANCEL	Cancela una petición pendiente.
REGISTER	Registrar al User Agent.

fiable y confirma la entrega de los datos.

Los codecs se utilizan para comprimir la señal de audio y reducir el tamaño del archivo para que pueda ser transmitido más eficientemente por la red. Sin embargo, la compresión también puede afectar la calidad del sonido, por lo que es importante encontrar un equilibrio entre la eficiencia y la calidad de la voz[10].

Existen varios modelos de codecs de audio utilizados en VoIP, cada uno con sus propias características y rendimiento, como G.711, G.726, GSM, MP3, entre otros. El algoritmo G.711 es el más comúnmente utilizado en VoIP y es conocido por su simplicidad y bajo consumo de recursos del sistema. Además, G.711 ofrece una calidad de voz de alta fidelidad, aunque su tasa de compresión es relativamente baja [9].

Otros codecs, como G.726 y GSM, utilizan técnicas de compresión más avanzadas que ofrecen una mayor eficiencia en la transmisión de la señal de audio, pero a costa de una calidad de voz ligeramente inferior. Por otro lado, los codecs MP3 se utilizan principalmente para la transmisión de música y archivos de audio de alta calidad, pero no se recomiendan para la transmisión de voz en tiempo real en VoIP debido a su alta tasa de compresión.

B. Servidor Asterisk

Asterisk es una aplicación de software libre con licencia GPL que permite simular las funciones de una centralita telefónica IPBX en un servidor informático [4]. Es capaz de manejar llamadas de voz, videoconferencias, mensajería instantánea, correo de voz y otros servicios de comunicaciones en tiempo real. Además, gracias a su arquitectura modular, permite integrar múltiples tecnologías y protocolos de comunicación como SIP, H.323, IAX, PSTN, GSM, entre otros [5].

En cuanto a su compatibilidad con diferentes sistemas operativos, *Asterisk* fue inicialmente desarrollado para Linux pero también puede ser utilizado en otros sistemas como MacOS, Solaris y FreeBSD, aunque la versión para Windows aún se encuentra en desarrollo [4].

III. Métricas de rendimiento

Cuando se evalúa el rendimiento de un sistema VoIP es importante medir tanto la calidad del servicio como el rendimiento general del servidor de VoIP para asegurarse de que el sistema está funcionando correctamente y proporcionando una experiencia de usuario óptima. Por ello la medición del rendimiento de un servidor de VoIP puede abordarse desde dos perspectivas diferentes, pero complementarias.

Por un lado, la calidad del servicio (QoS) es una métrica crítica en VoIP, ya que se trata de un servicio de comunicaciones en tiempo real que requiere una transmisión de voz clara y sin interrupciones. Para medir la calidad del servicio se utilizan una serie de métricas específicas, como:

- *Retardo o latencia:* mide el tiempo que tarda un paquete de voz en viajar desde el origen hasta el destino. Un retardo excesivo puede afectar la calidad del servicio al provocar cortes y pérdida de calidad de voz.
- *Pérdida de paquetes:* mide el número de paquetes que se pierden durante la transmisión. La pérdida de paquetes puede causar interrupciones en la comunicación y afectar negativamente la calidad de voz.
- *fluctuación temporal o Jitter(J):* mide la variación en el retardo de los paquetes que llegan al destino. Una fluctuación elevada puede causar problemas de sincronización y afectar la calidad de voz.
- *Ancho de banda (AB):* mide la cantidad de datos que se transmiten por segundo. Un ancho de banda insuficiente puede causar problemas de calidad de voz y retrasos en la transmisión. Para conocer el ancho de banda de una llamada VoIP se calcula mediante la ecuación 1 [6]. En esta ecuación, se observa que el ancho de banda es proporcional al tamaño del paquete por un factor PPS que indica la eficiencia del codec.

$$AB = TAM \times PPS \tag{1}$$

$$TAM = (ENC_{IP}) + (ENC_{CPU}) + (Voz)$$

$$PPS = \frac{Vel_{Codec}}{Voz}$$

- *Llamadas Perdidas(N_f):* Como ya se ha comentado, la congestión de la red o una mala configuración de la misma puede conllevar una perdida en la calidad de las llamadas, pero también es capaz de causar el fallo de estas. Esto podemos medirlo para estimar a qué nivel de saturación puede llegar nuestro sistema sin que ninguna llamada se pierda.

Por otro lado, también es importante medir el rendimiento general del servidor de VoIP y sus componentes para asegurarse de que el sistema está funcionando correctamente. Para ello, se pueden considerar métricas generales de rendimiento, como:

- *Tiempo de respuesta:* mide el tiempo que tarda el servidor en procesar una solicitud. Un tiempo de respuesta elevado puede indicar problemas de rendimiento.
- *Tasa de transferencia:* mide la cantidad de datos que se transfieren por segundo. Una tasa de transferencia insuficiente puede afectar negati-

vamente el rendimiento del sistema.

- *Carga del procesador y RAM:* mide la cantidad de recursos utilizados por el servidor. Una carga elevada puede afectar negativamente el rendimiento del sistema.

IV. Herramientas de monitorización y benchmark

Para monitorizar la carga del sistema se han utilizado varias herramientas, como *Sar*, *s-tuy* y el monitor del sistema de Ubuntu. *Sar* es una herramienta de línea de comandos que permite monitorizar distintos aspectos del sistema, incluyendo el uso de CPU, memoria, disco y red. Ofrece información detallada sobre la actividad del sistema en intervalos de tiempo regulares y permite generar informes en distintos formatos. Es recomendable utilizar varias herramientas para obtener una visión más completa del rendimiento del sistema y asegurar la veracidad de las mediciones. Para medir la temperatura de la CPU se puede utilizar *s-tui*, y para medir el uso de RAM se pueden utilizar *vmstat* y *sar*.

La red se ha monitorizado principalmente mediante dos herramientas, *Sngep* y *Wireshark* [11].

Sngrep es una herramienta de línea de comandos que permite visualizar y analizar el tráfico de llamadas de VoIP en tiempo real. Es una herramienta muy útil para los administradores de sistemas que gestionan servidores de telefonía IP, ya que les permite inspeccionar el tráfico de llamadas, detectar problemas y realizar diagnósticos precisos. Entre las funciones que ofrece *sngrep* se encuentra la posibilidad de personalizar la información mostrada en la pantalla, lo que permite al usuario adaptar la herramienta a sus necesidades específicas. Además, *sngrep* ofrece acceso a información detallada de cada llamada, incluyendo la hora de inicio, la duración, el origen y destino, el tipo de mensaje de señalización SIP utilizado, entre otros.

Wireshark es una herramienta de análisis de paquetes de red de código abierto y multiplataforma muy versátil que permite capturar, analizar y filtrar el tráfico de red en tiempo real, incluyendo el tráfico de voz y vídeo que se produce en las llamadas de VoIP. Para analizar llamadas de VoIP con *Wireshark*, es necesario elegir la interfaz de red adecuada para la comunicación de VoIP y seleccionar los protocolos que se quieren analizar, como SIP y RTP. *Wireshark* permite filtrar el tráfico de red para mostrar sólo la información relevante para el análisis de las llamadas de VoIP, lo que facilita la detección de problemas y el diagnóstico de errores en el sistema.

Para poder realizar una comparación en el rendimiento del servidor VoIP de *Asterisk*, se ha utilizado un modelo de carga que permitiera simular el servicio de llamadas. SIPp es una herramienta muy útil para realizar pruebas de carga y stress en aplicaciones y sistemas que utilizan el protocolo SIP. Con SIPp se pueden crear escenarios de prueba personalizados y simular el tráfico de llamadas SIP con distintos patrones de comportamiento, como por ejemplo el establecimiento y finalización de llamadas, transferencias, conferencias, etc. Además, SIPp también permite enviar tráfico de audio (RTP) y así analizar la calidad del audio y detectar posibles problemas de degradación de la señal. Esto es muy útil para asegurarse de que el servidor VoIP está funcionando correctamente y no presenta problemas en la transmisión de audio [12].

V. Arquitecturas

En esta sección se detallan las características técnicas de los tres equipos que se han utilizado para la monitorización y benchmark. Dos equipos portátiles con Ubuntu y un procesador i5 e i7 y una RaspBerry Pi como servidor. La tabla III resume las características hardware de dichos equipos.

Tabla III: Arquitecturas de prueba.

Modelo: Raspberry PI 3 CPU + GPU: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit Velocidad procesador: 1.4GHz. Memoria RAM: 1GB LPDDR2 SDRAM Wifi + Bluetooth: 2.4GHz y 5GHz IEEE 802.11 Ethernet: Gigabit Ethernet sobre USB 2.0 (300 Mbps) GPIO: 40 pines HDMI 4 puertos USB 2.0
Modelo: Leonovo IdeaPad5 Velocidad del procesador: 1.8GHz/4.3GHz Memoria RAM: 16GB Tarjeta Gráfica: AMD Radeon Graphics Procesador: AMD Ryzen 7 5700U 8C/16T, 1.8 GHz/4.3GHz, 8 MB
Modelo: ThinkiPad T450S Velocidad del procesador: 2.3GHz Memoria RAM: 8GB Tarjeta Gráfica: Intel HD Graphics 550 Procesador:Intel Core i5-5300U 2C/4T

VI. Resultados

Los resultados obtenidos en las pruebas experimentales llevadas a cabo se detallan en la tabla IV.

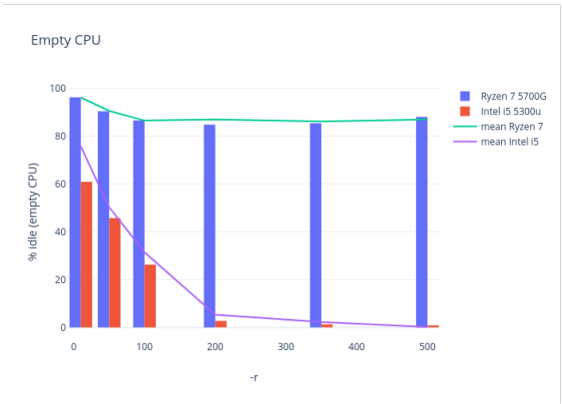
Una representación gráfica de los resultados de las métricas asociadas al servidor puede observarse en en la Figura 1. Sobre el uso de la CPU (Figura 1a), se ha observado que el Ryzen experimenta una disminución leve en la memoria disponible que se estabiliza a partir de las 100 llamadas por segundo. En contraste, al aumentar la frecuencia de llamadas por segundo en el equipo Intel, se ha observado una reducción simultánea en el porcentaje de CPU disponible.

En cuanto al uso de la RAM, se ha observado que su porcentaje de utilización se mantiene bastante estable en relación al número de llamadas por segundo. Se produce un pequeño aumento hasta las 200 llamadas por segundo, pero después apenas hay variaciones.

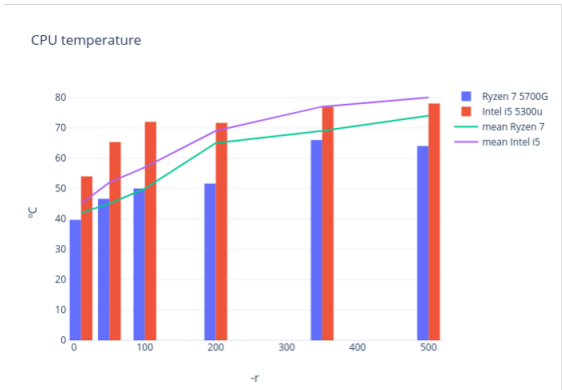
Las métricas del propio servicio muestran gráficamente en la Figura 2 y las de la red en la Figura . En cuanto al número total de llamadas, se ha observado que el equipo Ryzen 7 no ha sido capaz de procesar todas las llamadas programadas después de alcanzar las 200 llamadas por segundo durante la prueba. Por

Tabla IV: Resultados experimentales.
N/s: llamadas por segundo.N: llamadas totales. N_c:llamadas concluidas. N_f: llamadas fallidas. AB: Ancho de banda

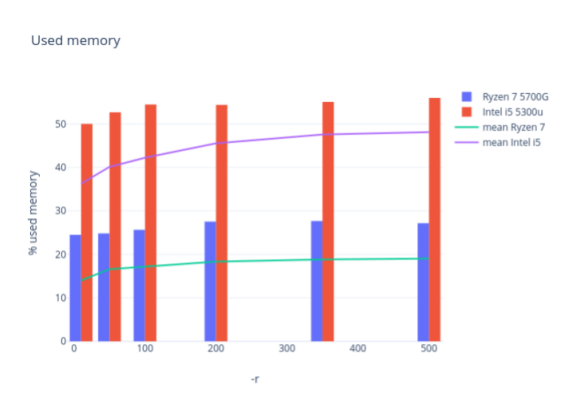
	N/s	N	N_c	N_f	% fallo	% CPU Libre	Temp CPU	%Uso mem.	Fluctuación	AB
Ryzen 7	10	299	299	0	0	96.11	42.5	14	0.0885	15.69
Intel i5	10	299	299	0	0	75.69	45.2	36.26	0.0825	20.33
Ryzen 7	50	1500	1498	2	0.13	90.53	45.7	16.59	0.092	60.1
Intel i5	50	1499	1499	0	0	50.23	52.3	40.1	0.4344	65.54
Ryzen 7	100	2999	2999	0	0	86.49	50.6	17.18	0.033	160.58
Intel i5	100	3000	2992	8	0.27	31.42	57.3	42.2	0.2503	174.54
Ryzen 7	200	5977	4999	978	16.36	86.94	65.2	18.32	0.026	318.19
Intel i5	200	4000	3554	446	11.15	5.32	69.7	45.5	0.1519	385.67
Ryzen 7	350	7495	4089	3406	45.44	86.15	69.3	18.8	0.024	332.16
Intel i5	350	2012	1612	400	19.88	2.21	77.7	47.54	0.0379	455.97
Ryzen 7	500	8718	4914	3804	43.63	86.97	74.4	19	0.0205	360.45
Intel i5	500	2312	1846	466	20.16	0.13	80.1	48.1	>0.01	425.81



(a) Uso de CPU



(b) Temperatura de CPU



(c) Uso de la memoria
Fig. 1: Métricas del Servidor

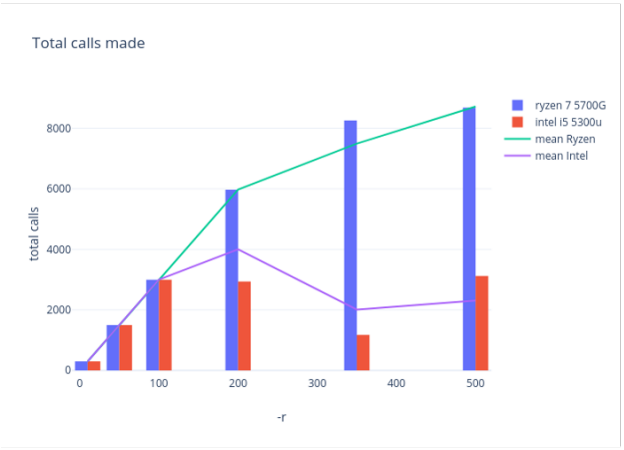
ejemplo, para 350 llamadas programadas, solo se han procesado 8.000 llamadas, lo que representa más del 50 % de fallas en la ejecución. Lo mismo ocurre con el Intel i5, pero a las 200 llamadas por segundo, ya que no se han procesado ni siquiera la mitad de las llamadas programadas, es decir, solo se han procesado 6.000 llamadas.

En cuanto a las llamadas fallidas, se ha observado que el Ryzen 7 tiene un porcentaje mayor que el Intel i5 después de alcanzar las 200 llamadas por segundo. Esto se explica en parte porque el número de llamadas en cola es mayor en el caso del Ryzen 7 que en el del Intel i5.

Es importante destacar que el equipo Intel i5 muestra un mayor consumo de ancho de banda a partir de las 100 llamadas, lo que coincide con una reducción significativa en el número total de llamadas realizadas (Figura 3a). Es posible que el servidor *Asterisk* haya intentado establecer la comunicación previamente a la llamada fallida, enviando paquetes adicionales a través del protocolo de señalización SIP. Cuando estos paquetes no son recibidos en un tiempo determinado, el intento de comunicación se cancela.

En el análisis de la fluctuación temporal, se observó una evolución diferente a la esperada en relación a la pérdida de paquetes y la falta de completado de llamadas (Figura 3b). Por lo tanto, para entender este parámetro es importante profundizar en qué medida es la más conveniente, qué ocurre realmente en los escenarios y determinar si hay pérdida de paquetes.

La tabla V muestra los resultados del análisis estadístico utilizando el test de la t de Student. Se observaron diferencias estadísticamente significativas con una confianza del 95 % en el uso de la CPU y de la RAM. Estos hallazgos coinciden con estudios previos [10], lo que refuerza la validez de dichos resultados. Sin embargo, no se puede determinar con certeza si estas diferencias en el uso de RAM se deben al servidor o a que no se tuvo en cuenta el nivel de uso de RAM antes de realizar las pruebas. Existen indicios de que las dos arquitecturas estaban utilizando la RAM de manera diferente antes de la ejecución de la prueba. En cuanto al resto de métricas, no se encontraron diferencias estadísticamente significativas con una confianza del 95 %.



(a) Total de llamadas

(b) Llamadas fallidas
Fig. 2: Métricas del Servicio

Tabla V: Test estadístico T de Student.

Métrica	p_value	$H_0 \alpha = 0{,}05$	$H_0 \alpha = 0{,}10$
Ancho banda	0.0606	Acepta	Rechaza
Fluctuación	0.0996	Acepta	Rechaza
Llamadas_fallidas	0.1375	Acepta	Acepta
Llamadas totales	0.1113	Acepta	Acepta
Uso RAM	0	Rechaza	Rechaza
CPU	0.0027	Rechaza	Rechaza

(a) Ancho de banda

(b) Fluctuación temporal
Fig. 3: Métricas de la red

VII. Implementación de la Propuesta

En esta sección se presentan las directrices prácticas para llevar a cabo la propuesta de enseñanza descrita anteriormente. Se abordarán aspectos como la temporización, la evaluación, los objetivos mínimos, el material didáctico a proporcionar a los alumnos y las actividades de búsqueda independiente que se les podría encomendar.

A. Temporización

La duración de esta propuesta docente puede variar en función de la disponibilidad de tiempo y el nivel de los estudiantes. Sin embargo, como guía general, se podría dividir el contenido de la siguiente manera en 4 semanas con 2 horas prácticas a la semana:

- **Semana 1:** Introducción a VoIP y sus componentes básicos y los parámetros de rendimiento de servidores.
- **Semana 2:** Introducción a las técnicas de monitorización y referenciación.
- **Semana 3:** Aplicación práctica de estas técnicas a los servidores VoIP.
- **Semana 4:** Evaluación y comparación de diferentes sistemas

B. Evaluación

La evaluación de los estudiantes podría basarse en una combinación de evaluación continua y un proyecto final. La evaluación continua podría incluir cuestionarios semanales sobre el contenido cubierto, mientras que el proyecto final podría involucrar la implementación y evaluación de un servidor VoIP utilizando las técnicas aprendidas, entregando y defendiendo una memoria de prácticas sobre el trabajo desarrollado.

C. Objetivos mínimos

Los objetivos mínimos que los estudiantes deben alcanzar tras aplicar esta propuesta docente son los siguientes:

- Comprender los conceptos básicos de VoIP y sus componentes.
- Ser capaz de identificar y medir los paráme-

tros de rendimiento relevantes para los servidores VoIP.
- Adquirir habilidades en técnicas de monitorización y referenciación.
- Ser capaz de aplicar estas técnicas para evaluar y optimizar el rendimiento de un servidor VoIP.

D. Material didáctico

Para facilitar el aprendizaje, se proporcionarán a los alumnos guías de estudio que cubran los conceptos teóricos, así como guías prácticas que les guíen a través de las técnicas de monitorización y referenciación.

Además del material proporcionado, se animará a los estudiantes a realizar investigaciones independientes para complementar su aprendizaje. Esto podría incluir la búsqueda de literatura relevante, la exploración de diferentes herramientas de monitorización y referenciación, y la experimentación con diferentes configuraciones de servidores VoIP.

VIII. Conclusiones

En este trabajo se ha abordado el estudio de un servicio de red con un doble objetivo: mostrar cómo proceder en este tipo de estudios y permitir al alumnado profundizar en el conocimiento del servicio de VoIP. Para ello, se han utilizado diversas herramientas de monitoreo que han permitido una monitorización completa del servicio, abarcando métricas como el ancho de banda y el uso de Wireshark para escuchar las llamadas.

Se puede observar de los resultados obtenidos que un alumno que lleve a cabo un trabajo como el que se presenta para análisis de un servicio de voz IP aprendería el uso de nuevas herramientas a ser crítico con la métrica empleada para medir el rendimiento de un servicio, ser precisos en la medición de los recursos utilizados en en un sistema informático, y por último emplear métodos estadísticos para llevar a cabo una comparación de cuál opción de arquitecturas sería más adecuada utilizar entre diferentes posibilidades que puedan tener disponibles.

Referencias

[1] Resolución de 17 de marzo de 2015, de la Universidad de Córdoba, por la que se publica la modificación del plan de estudios de Graduado en Ingeniería en Informática. Boletín Oficial del Estado, 79, de 2 de abril de 2015.

[2] Rodríguez Lozano, Francisco Javier, Ávila Jiménez, José Luis. Guía docente Configuración y Evaluación de Sistemas Informáticos. Curso 2022-2023. https://www.uco.es/eguiado/guias/2022-23/101403es_2022-23.pdf. Consultado: 11 Marzo 2023

[3] David J. Lilja. Measuring computer performance: a practitioner's guide. Cambridge University Press, 2000

[4] Gómez López, J., Gil Montoya, F. . VoIP y Asterisk, redescubriendo la telefonía. Ra-Ma, 2014.

[5] Egea López, E. . Proyecto Fin de Master. Instalación de un sistema VoIP corporativo basado en Asterisk. Escuela Técnica Superior de Ingeniería de Telecomunicación. Universidad Politécnica de Cartagena. 2008.

[6] Miroslav Voznak, J. R. . SIP Infrastructure Performance Testing.9th WSEAS International Conference on Telecommunications and Informatics. 1969.

[7] Rahman, M. M., Islam, N. S. VoIP Implementation Using Asterisk PBX. IOSR Journal of Business and Management, 15(6), 47–53. 2014.

[8] J. Rosenberg, H. Schulzrinne et al. SIP: Session Initiation Protocol Junio 2002. (Network Working Group RFC; 3261). ftp://ds.internic.net/rfc/rfc3261.txt. Consultado: 11 Marzo 2023

[9] Salcedo, Octavio; López, Danilo and Hernández, Cesar. Estudio comparativo de la utilización de ancho de banda con los protocolos SIP e IAX. Tecnura . 2012, vol.16, n.34, pp.171-187. ISSN 0123-921X.

[10] Mohiuddin Ahmed and Abdul Malik Mansor. 2008. CPU dimensioning on performance of Asterisk VoIP PBX. In Proceedings of the 11th communications and networking simulation symposium (CNS '08). Association for Computing Machinery, New York, NY, USA, 139–146.

[11] Rouse, M., 2011. What is Wireshark? - Definition from WhatIs.com. [online] WhatIs.com. Available at: http://whatis.techtarget.com/definition/Wireshark Consultado: 7 Agosto 2016.

[12] Tian, Lu , Dailly, Nicolas , Qiao, Qiao , Lu, Jihua , Zhang, Jiannan , Guo, Jing , Zhang, Ji'ao. (2011). Study of SIP protocol through VoIP solution of "Asterisk". 2011 Global Mobile Congress, GMC 2011. 10.1109/GMC.2011.6103925.

Integración del simulador CREATOR con hardware RISC-V: caso de estudio con microcontrolador ESP32

Diego Camarmas-Alonso, Félix García-Carballeira, Alejandro Calderón-Mateos y Elías Del-Pozo-Puñal[1]

Resumen— **Actualmente existen multitud de simuladores de lenguaje ensamblador que permiten a los estudiantes ver y comprender cómo ejecutan los programas ensamblador. Sin embargo, estos simuladores ocultan a los estudiantes todas las implicaciones en términos de rendimiento, consumo de memoria o consumo de energía que conlleva ejecutar estos programas sobre un *hardware* real.**

Por ello, en este trabajo se presenta una nueva funcionalidad desarrollada en el simulador CREATOR que permite cargar y ejecutar el código implementado en el propio simulador en un microcontrolador. Permitiendo a los estudiantes comprender cómo ejecutan los programas ensamblador en un *hardware* real, así como ser una primera toma de contacto en la programación de microcontroladores.

Para este trabajo, se ha utilizado como caso de uso, el lenguaje ensamblador RISC-V, integrándolo y probándolo inicialmente con el microcontrolador Espressif ESP32-C3, lo que permite a los estudiantes ver cómo funcionan sus programas sobre un dispositivo.

Palabras clave— **RISC-V, ESP32-RISCV, simulador, ensamblador, Arquitectura de Computadores.**

I. Introducción

El trabajo presentado describe el diseño y desarrollo de una nueva funcionalidad en el simulador educativo CREATOR [1] [2]. Este simulador es de código abierto y su código fuente se puede encontrar en su repositorio de GitHub [3]. Esta nueva funcionalidad permite que el código ensamblador escrito, compilado y depurado en el simulador pueda ser ejecutado en un dispositivo *hardware* real. Concretamente, para este trabajo, se va a utilizar como caso de uso un microcontrolador ESP32-C3.

Cabe destacar que CREATOR se ha diseñado para ser independiente del *hardware* utilizado, es decir, permite simular el código ensamblador sobre la propia herramienta sin la necesidad de que haya un microcontrolador conectado.

A pesar de la existencia de múltiples simuladores de lenguaje ensamblador, estos no permiten a los estudiantes ver cómo sus programas pueden ser llevados y ejecutados sobre un *hardware* real. Esto provoca que en muchas ocasiones los estudiantes no sean conscientes de las implicaciones en términos de rendimiento, control de gasto de energía y consumo de memoria, entre otros, que su código tiene cuando ejecuta en un *hardware* real.

[1]Departamento de Informática, Universidad Carlos III de Madrid (UC3M), e-mail: {dcamarma,fgcarbal,acaldero,edelpozo}@inf.uc3m.es

[2]https://creatorsim.github.io/

[3]https://github.com/creatorsim/creator

Por otra parte, los entornos de desarrollo que trabajan con *hardware* real como un microcontrolador, habitualmente son entornos profesionales complejos de utilizar o bien son entornos para aprendizaje basados en Arduino con UIFlow [2] o Python. Hasta donde los autores conocen, no existe un entorno intermedio que permita aprender de forma sencilla ensamblador y que, al mismo tiempo, permita ejecutar el código desarrollado sobre *hardware* real, combinando sencillez y profesionalidad a la vez.

Por todo ello, el objetivo de este trabajo es que los estudiantes dispongan de una integración entre el simulador y el *hardware* asequible, fácil y familiar en la que puedan ejecutar sus ejercicios de ensamblador RISC-V y comprender el impacto de su ejecución sobre un dispositivo.

Además, cabe señalar, que el uso de RISC-V [3] permite trabajar con un *hardware* abierto y emergente. Su similitud en ciertos aspectos a MIPS [4] (uno de los ensambladores más usados en docencia) permite una fácil transición a un nuevo procesador diseñado teniendo en cuenta los requisitos actuales y futuros de la arquitectura de computadores.

Asimismo, dado que se ha elegido para el caso de uso presentado en este trabajo como *hardware* un microcontrolador, el precio es más asequible respecto al de las placas SBC (*Single Board Computer*) parecida a la Raspberry Pi, cuyos precios actualmente son muy elevados. Además, al estar basado en un microcontrolador, muchos estudiantes estarán familiarizados con el uso de estos dispositivos y su entorno de trabajo, al haber utilizado anteriormente los microcontroladores basados en Arduino.

El resto del documento se estructura de la siguiente forma: la Sección II describe el Estado del Arte; la Sección III presenta las adaptaciones realizadas en CREATOR para permitir su integración con el *hardware*. En la Sección IV se presenta un caso de uso con un microcontrolador ESP32-C3. Por último, la Sección V presenta las principales Conclusiones y Trabajos Futuros.

II. Estado del arte

Actualmente, como se puede ver en [5], existen diferentes herramientas didácticas basadas en el ensamblador RISC-V. Las más conocidas son Jupiter [6], RARS [7] y Venus [8], pero también existen otras herramientas que están especialmente diseñadas para simular la ejecución de ensamblador con un *pipeline* de 5 etapas como son Ripes [9] y WebRISC-V [10].

Por último, cabe destacar RVfpga [11], que es un curso de RISC-V que hace uso de *hardware*. Los usuarios al finalizar estos cursos, según los autores, tendrán un sistema RISC-V en funcionamiento y experiencia práctica en la exploración y el uso de RISC-V SoC y la cadena de herramientas RISC-V, incluidos compiladores y simuladores.

Jupiter [6], según sus autores, es un simulador multiplataforma (Linux, macOS y Windows) educativo de RISC-V (RV32IMF) implementado en Java. Este simulador permite ejecutar programas ensamblador escritos en varios ficheros, permitiendo una mayor modularidad en el código. Además, este simulador dispone de dos modos de operación: interfaz gráfica y línea de órdenes. Sin embargo, este simulador al estar basado en Java dificulta su uso en los dispositivos y no puede ser ejecutado en dispositivos móviles como *tablets*, que cada día son más usadas ya que son fáciles de transportar.

Otro ejemplo es el simulador de RISC-V llamado RARS [7] que está desarrollado utilizando como base el simulador de MIPS-32 MARS, pero no integrado con él. Por lo que tiene funcionalidades y limitaciones muy similares a las de MARS. Entre estas limitaciones está, como ocurre con el simulador anterior, la dificultad para desplegar el simulador al estar basado en Java en vez de en HTML5, el juego de instrucciones disponible no es editable y su accesibilidad es limitada, entre otras.

Venus [8] es un simulador educativo de RISC-V que dispone de una versión web y de una versión implementada en Java. Este simulador permite simular y depurar los programas ensamblador implementados en el editor del propio simulador, así como visualizar el estado de la memoria principal. Sin embargo, como ocurre con los simuladores mencionados anteriormente solo implementa un subconjunto limitado de la ISA de RISC-V (RV32IM) y no puede ser editado.

Además de los simuladores descritos anteriormente existen otros simuladores que permiten simular el programa ensamblador con un *pipeline*. Entre estos se encuentra la herramienta Ripes [9] que es un simulador basado en un *pipeline* de 5 etapas del ensamblador RISC-V que comenzó a desarrollarse en 2016. Cabe destacar que esta herramienta también permite simular la memoria caché. Sin embargo, este simulador está desarrollado en C++ y su interfaz gráfica se basa en la biblioteca Qt. Lo que imposibilita su uso en dispositivos móviles (*smartphones* y *tablets*) ya que solo puede ser usado en los sistemas operativos Linux, macOS y Windows.

Por último, existe WebRISC-V [10] que es un simulador web que ejecuta programas escritos en RISC-V de 32 bits. Al igual que el simulador Ripes, este simulador ejecuta los programas ensamblador con segmentación (*pipeline*) de 5 etapas. Sin embargo, este simulador implementa un subconjunto muy limitado del juego de instrucciones de RISC-V y tampoco permite editar este subconjunto para añadir nuevas instrucciones.

Tras estudiar los simuladores anteriores CREATOR facilita el aprendizaje porque reúne las características recomendadas:

- Es multiplataforma y no precisa de instalación (solo precisa de un navegador web).
- Dispone del juego de instrucciones RISC-V y MIPS32, pero el juego de instrucciones es personalizable.
- Dispone de interfaz gráfica, así como ejecución en línea de comandos (permitiendo la ejecución de correctores automáticos).
- Muestra errores de compilación, errores de ejecución y errores en el uso del convenio de paso de parámetros.

Todas estas características permiten a los estudiantes aprender lenguaje ensamblador de forma iterativa en un único simulador, permitiendo una fácil transición entre diferentes lenguajes ensamblador.

Por otro lado, en cuanto al *hardware* que se puede utilizar para ejecutar los programas RISC-V desarrollados en CREATOR, encontramos dos posibilidades principales: bien usar un SBC o bien un microcontrolador.

En cuanto a las SBC, es muy conocida la Raspberry Pi como plataforma para aprendizaje usando el procesador ARM. En el caso de RISC-V, la empresa SiFive ofrece una solución muy similar basada en procesador RISC-V llamada HiFive1 Rev B [12]. Dicha placa permite ejecutar un sistema operativo Linux completo, de forma que es posible, por ejemplo, además de ejecutar programas RISC-V, arrancar un servicio web que permita interactuar con el simulador CREATOR. Además, SiFive dispone también de placas como la Vision Five 2 [13] con GPU integrada. Sin embargo, las SBC tienen algunos inconvenientes en cuanto a su uso en una clase de laboratorio en docencia. El coste de una placa puede representar un problema para los estudiantes, la crisis de microchips que se ha acentuado debido a la pandemia de COVID-19, hace que la compra de estos dispositivos sea todavía más cara, tarde bastante en poder servirse o incluso no existir *stock*. Además, estas placas necesitan una fuente de alimentación para poder funcionar, hay que tener una tarjeta MicroSD con el sistema operativo, conexión a red, etc. lo que obliga a que la clase de laboratorio esté acondicionada.

Otra alternativa son los microcontroladores. Los microcontroladores más conocidos son los usados junto con la plataforma Arduino. Estos permiten la programación tanto en UIFlow [2] como en Python, permitiendo transferir el código binario al microcontrolador para su ejecución en el Arduino. Existen diversos fabricantes de microcontroladores basados en RISC-V, destacando Espressif [14] que dispone de muchos ejemplos y documentación en la plataforma GitHub y en su propia página Web. Los microcontroladores respecto a las SBC son más económicos, facilitando su acceso a los estudiantes, los tiempos de espera del envío generalmente es menor, no precisan de alimentación adicional a la del USB y, además, la

gran mayoría de los estudiantes han utilizado alguna vez Arduino, por lo que están más familiarizados con este *hardware*.

III. Integración CREATOR-*hardware*

En esta sección se va a describir cómo se ha llevado a cabo la integración del simulador CREATOR para permitir el cargado y ejecución de los programas ensamblador implementados en el simulador sobre un microcontrolador, así como los pasos que hay que realizar para su puesta en marcha (prerrequisitos).

A. Diseño de la ejecución en hardware real

Para poder llevar a cabo la carga del programa ensamblador desarrollado en CREATOR sobre un dispositivo *hardware* se han tenido que realizar pequeños cambios en el simulador respecto a la versión inicial [1]. Estos cambios afectan especialmente a la interfaz gráfica y a la implementación de un nuevo servicio web. Este servicio web está implementado en Python 3, y será el encargado de comunicarse con los *drivers* de la placa a utilizar. Para ello, se ha utilizado el *framework* Flask de Python 3 que facilita la implementación de servicios web.

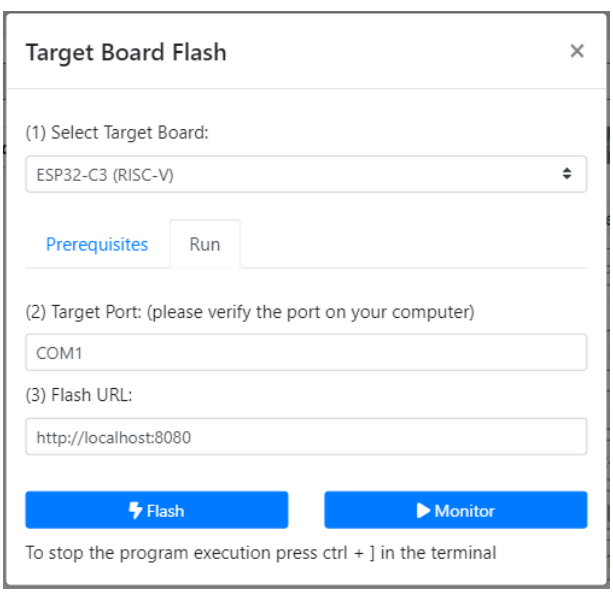


Fig. 1: Formulario de cargado y ejecución de un programa en el microcontrolador.

Respecto a la interfaz de usuario, en CREATOR se dispone de un cuadro de diálogo (ver Figura 1) que permite a los usuarios definir tres parámetros imprescindibles para el correcto cargado del programa en el microcontrolador:

- Modelo del microcontrolador: indica el modelo exacto del *hardware* que se va a utilizar para ejecutar el programa. Por defecto, el modelo será ESP32-C3 de RISC-V.
- Puerto de conexión: define el puerto al que se ha conectado el *hardware*. Como este puerto depende del sistema operativo, por defecto, CREATOR es capaz de detectar el sistema operativo en el que se ejecuta el simulador y poner el puerto que se utiliza por defecto en este.
- URL del servicio web: indica la URL en la que está ejecutando el servicio web, que, por defecto, será: `http://localhost:8080`.

Como se puede ver en la Figura 1 se pueden llevar a cabo dos acciones: *flash* (cargado del programa en el microcontrolador) y *monitor* (visualización de la ejecución del programa cargado en el microcontrolador). Cuando se pulsa alguno de estos botones CREATOR se conectará con el servicio web desarrollado realizando las acciones descritas en la Figura 2.

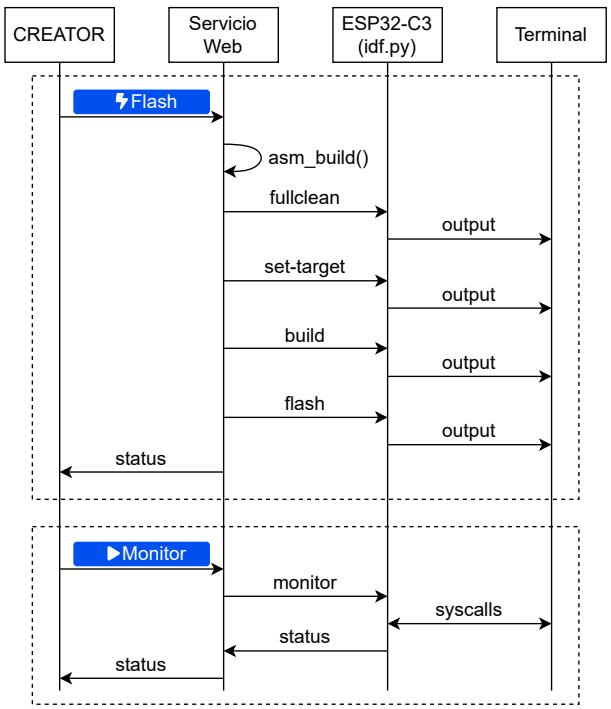


Fig. 2: Intercambio de operaciones entre CREATOR, el servicio web y el *driver* del microcontrolador ESP32-C3.

Por un lado, está el método *flash* que envía al servicio web el modelo del microcontrolador, el puerto al que se ha conectado este y el programa ensamblador que va a ser compilado y cargado en el microcontrolador.

Para ello, en primer lugar, el servicio web guarda el programa ensamblador en un archivo temporal y realiza un preprocesado del código ensamblador para poder emular las llamadas al sistema implementadas en CREATOR en el propio microcontrolador. La Tabla I muestra las llamadas al sistemas en CREATOR, donde la columna *Id.* representa el código de la llamada (que se pasa en el registro a7), la columna *Argumentos* representa el resto de valores a pasar en los registros además del código de llamada, y la columna *Resultado* representa los registros y valores que las llamadas pueden devolver. Este preprocesado es muy importante, ya que permite que el resultado de ejecutar estas llamadas al sistema sea el mismo en el simulador y en el microcontrolador, ya que estas pueden diferir puesto que estos dispositivos no siguen estrictamente los convenios de los lenguajes ensam-

blador y ejecutan en modo protegido.

Tabla I: Llamadas al sistema implementadas en CREATOR.

Llamada al sistema	Id.	Argumentos	Resultado
Print integer	1	a0 = integer	
Print float	2	fa0 = float	
Print double	3	fa0 = double	
Print string	4	a0 = dir. string	
Read integer	5		integer en a0
Read float	6		float en fa0
Read double	7		double en fa0
Read string	8	a0 = dir. string a1 = longitud	
Sbrk	9	a0 = longitud	dir. en a0
Exit	10		
Print char	11	a0 = char ASCII	
Read char	12		char en a0

Tras realizar este preprocesado, el servicio web se encarga de interactuar con los *drivers* específicos del microcontrolador, realizando el proceso de compilación y cargado de datos en el chip de memoria del microcontrolador. A continuación, se devuelve a CREATOR el resultado (*status*) de realizar todos estos pasos para que el simulador pueda devolver un mensaje de error en caso de que algún paso haya fallado.

Por otro lado, el método *monitor* se encarga de visualizar la ejecución del último programa ensamblador cargado en el microcontrolador con el método *flash*. Además, también, permitirá interactuar con el terminal del ordenador que ejecuta el servicio web para poder realizar las diferentes llamadas al sistema, como imprimir valores por pantalla o leer valores desde teclado.

Como ocurre con el método *flash* el resultado global de la ejecución (*status*) se envía de vuelta a CREATOR para mostrar retroalimentación al usuario del resultado de la ejecución.

B. Prerrequisitos de la ejecución en hardware real

Para poder utilizar el soporte de *hardware* disponible en CREATOR es necesario seguir una serie de pasos. Estos pasos se pueden dividir en dos bloques principales: por un lado, la instalación del *software* necesario para trabajar con el microcontrolador y, por otro lado, la ejecución de este *software*.

La instalación del *software* necesario dependerá del Sistema Operativo utilizado en el ordenador donde se va a instalar y ejecutar el *software*, que podrá ser Linux, macOS o Windows, así como del modelo de microcontrolador que queremos utilizar.

Partiendo de la premisa de que está Python 3 instalado en el sistema operativo hay que instalar los paquetes de Python 3 necesarios para ejecutar el servicio web, así como el *software* de desarrollo del microcontrolador para dicho sistema operativo.

Una vez hemos seleccionado el *hardware* a utilizar en CREATOR, los pasos que hay que realizar serán mostrados en el propio simulador, como se puede ver en la Figura 3.

La primera vez que trabajemos con este microcontrolador tendremos que realizar los siguientes pasos:

1. Instalar el *software* de desarrollo del microcontrolador.
2. Instalar los paquetes `flask` y `flask_cors` de Python 3.
3. Descargar desde CREATOR el archivo zip con el *driver* asociado al microcontrolador. Este archivo comprimido contiene un proyecto para el *hardware* seleccionado, así como el servicio web comentado al principio de la sección.
4. Descomprimir dicho archivo zip.

Cada vez que ejecutemos el servicio web, los pasos que hay que realizar son:

1. Cargar las variables de entorno asociadas al microcontrolador. Por ejemplo, en ESP32:
   `. $HOME/esp/esp-idf/export.sh`
2. Cambiar el directorio de trabajo al directorio donde están los archivos descomprimidos del *driver*. Por ejemplo, en ESP32:
   `cd $HOME/creator/esp32c3/`
3. Ejecutar el servicio web. Por ejemplo, en ESP32:
   `python3 gateway.py`

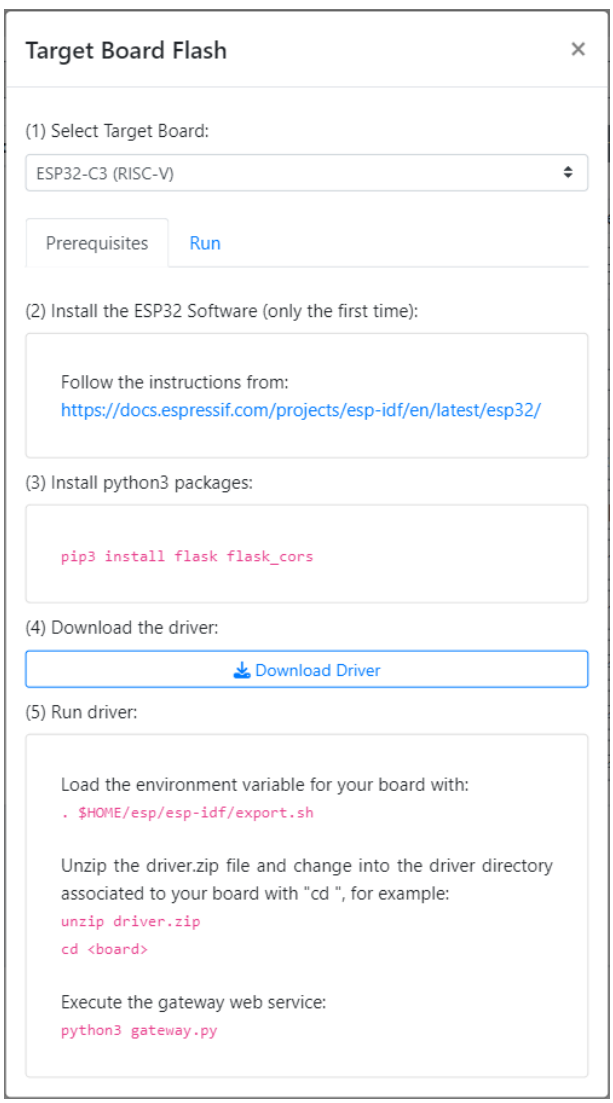


Fig. 3: Prerrequisitos para utilizar el soporte de *hardware* de CREATOR.

```
.text
factorial:
   # crear marco de pila
   addi sp, sp, -12
   sw   ra, 8(sp)
   sw   fp, 4(sp)
   addi fp, sp, 4

   li   x5, 2
   bge  a0, t0, b_else

   # if (a0 < 2):
   #   a0 = 1
   #   return a0
   li   a0, 1
   beq  x0, x0, b_efs

   # else:
   #   a0=a0*factorial(a0-1)
   #   return a0
 b_else:
   sw   a0, -4(fp)
   addi a0, a0, -1
   jal  x1, factorial
   lw   t1, -4(fp)
   mul  a0, a0, t1

 b_efs:
   # finalizar marco de pila
   lw   ra, 8(sp)
   lw   fp, 4(sp)
   addi sp, sp, 12

   # return a0
   jr ra
```

```
main:
    # crear marco de pila
    addi sp, sp, -20
    sw ra, 16(sp)
    sw s0, 12(sp)
    sw s1,  8(sp)
    sw s2,  4(sp)
    sw s3,  0(sp)

    # s=0
    li s1, 0
    li s0, 3
    # while (s1 < 3) {
w1: bge  s1, s0, w1_end

    # s2 = get_cycles()
    rdcycle s2
    # factorial(10)
    li  a0, 10
    jal x1, factorial
    # s3 = get_cycles()
    rdcycle s3

    # print(s3-s2)
    sub a0, s3, s2
    li a7, 1
    ecall

    # s1++
    addi s1, s1, 1
    beq  zero, zero, w1
    # }

w1_end:
    # restaurar pila
    lw s3,  0(sp)
    lw s2,  4(sp)
    lw s1,  8(sp)
    lw s0, 12(sp)
    lw ra, 16(sp)
    addi sp, sp, 20
    jr ra
```

Fig. 4: Programa RISC-V que mide de forma aproximada el número de ciclos para calcular el factorial de 10 con recursividad e imprime dicho número de ciclos.

Una vez se han seguido los pasos anteriores tendremos ejecutando el servicio web y CREATOR se podrá conectar con él para cargar y ejecutar el programa ensamblador deseado.

IV. Caso de uso: Espressif ESP32-C3

Para probar la nueva funcionalidad de CREATOR presentada en este trabajo, se ha usado como caso de uso el microcontrolador ESP32-C3 (como el mostrado en la Figura 5).

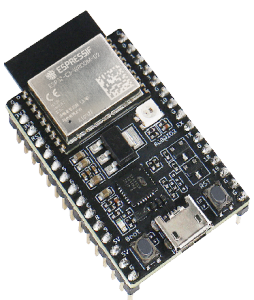


Fig. 5: Microcontrolador ESP32-C3-DevKitC-02 basado en RISC-V [14].

Este microcontrolador tiene una CPU RISC-V de 32 bits con un único core de hasta 160 MHz, 400 KB de SRAM, 384 KB de ROM y una memoria caché asociativa de 16KB con un tamaño de bloque de 32 bytes [15]. Cabe destacar que este microcontrolador no tiene soporte de operaciones en coma flotante.

Para llevar a cabo este caso de uso, se han utilizado el programa RISC-V mostrado en la Figura 4. Este programa mide de forma aproximada el número de ciclos de reloj que se necesitan para calcular el factorial de 10 (la subrutina factorial está implementada de forma recursiva). Una vez realizada la medición se imprime el número de ciclos de reloj. El cálculo de ciclos de ejecución se lleva a cabo 3 veces, esto permite ver a los estudiantes como la memoria caché del microcontrolador permite reducir el número de ciclos necesarios para la ejecución de un mismo código al no tener que realizar accesos a memoria principal la segunda y tercera vez que se ejecuta.

En primer lugar, una vez se ha implementado el programa ensamblador usando el editor de código de CREATOR, hay que compilarlo para verificar que no existen errores en el código. Si la compilación no devuelve ningún error, el siguiente paso será ejecu-

tar este programa en el simulador para verificar qué funciona correctamente antes de llevarlo al dispositivo *hardware*.

Una vez hemos verificado que la ejecución es correcta utilizando CREATOR, procederemos a cargar el programa en el microcontrolador haciendo uso del formulario mostrado en la sección anterior (ver Figura 1). Concretamente, ejecutaremos la acción *flash*.

Cuando se ejecuta la acción de *flash* en el servicio web, como se explicó en la sección anterior, se realiza un preprocesado sobre el código ensamblador implementado para que las llamadas al sistema funcionen de igual forma que en CREATOR y el resultado sea el mismo. Una vez se ha adaptado el código ensamblador, el servicio web carga el programa ensamblador en el microcontrolador mostrándose por pantalla el proceso de cargado, como se puede ver en la Figura 6.

```
Flash will be erased from 0x00000000 to 0x00005fff...
Flash will be erased from 0x00010000 to 0x00048fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Compressed 20656 bytes to 12717...
Writing at 0x00000000... (100 %)
Wrote 20656 bytes (12717 compressed) at 0x00000000 in 0.8 seconds
(effective 204.1 kbit/s)...
Hash of data verified.
Compressed 231584 bytes to 114382...
Writing at 0x00010000... (14 %)
Writing at 0x0001eefe... (28 %)
Writing at 0x0002587c... (42 %)
Writing at 0x0002d138... (57 %)
Writing at 0x000345bf... (71 %)
Writing at 0x0003a701... (85 %)
Writing at 0x0004156f... (100 %)
Wrote 231584 bytes (114382 compressed) at 0x00010000 in 3.6 second
s (effective 516.4 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.1 seconds (ef
fective 279.6 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
Done
```

Fig. 6: Proceso de cargado del programa en el microcontrolador.

Por último, una vez se ha cargado correctamente el programa ensamblador en el microcontrolador, podremos visualizar su ejecución en el dispositivo. Para ello, se utilizará la acción *monitor* del cuadro de diálogo mostrado en la Figura 1. Durante la ejecución de esta acción, en la terminal donde se ejecuta el servicio web se irá mostrando la salida del programa. Y, también, será posible introducir valores por teclado si el programa realiza alguna llamada al sistema de lectura de teclado.

En el Listado 1 se puede ver el resultado de la ejecución del programa ensamblador presentado en la Figura 4. Se puede destacar que la primera vez que se calcula el factorial de 10 se necesitan 1382 ciclos de reloj, mientras que en los cálculos posteriores se necesitan solamente 220 ciclos. Esto se debe a que el código a ejecutar se encuentra en memoria caché y no es necesario leerlo de memoria principal, reduciendo el número de ciclos que necesita la subrutina `factorial` para ejecutarse.

Listado 1: Ciclos de las 3 ejecuciones del factorial de 10.

```
Started program...
-------------------
>1382
>220
>220
Finished program: 53638 cycles
-------------------
```

V. Conclusiones

En este artículo se ha presentado el diseño y desarrollo realizado en el simulador CREATOR para permitir a los estudiantes ejecutar sus programas ensamblador en un microcontrolador real desde CREATOR, así como un caso de uso utilizando el microcontrolador ESP32-C3. Cabe destacar, que el diseño modular de la interfaz de CREATOR ha permitido añadir esta nueva funcionalidad de forma rápida y segura.

Como principales conclusiones de los primeros resultados, podemos destacar que esta nueva funcionalidad añadida en CREATOR permite a los estudiantes probar su código ensamblador usando *hardware* real, ayudándoles a ser conscientes del impacto de la ejecución de su código en términos de rendimiento y gasto energético, como se ha podido ver en el caso de uso. La elección de un microcontrolador para desarrollar esta integración permite no excluir a personas por factores económicos al ser más accesibles que las SBC. Además, su semejanza con Arduino favorece el acercamiento a un entorno de trabajo conocido por estudiantes que hayan trabajado previamente con este microcontrolador.

Como principales líneas de trabajos futuros se plantea la integración con más modelos de microcontroladores compatibles con RISC-V así como MIPS-32. También se quiere simplificar la puesta en marcha del entorno en diferentes plataformas, usando una imagen de contenedor *docker* por ejemplo. Otra línea de trabajo futuro consiste en simular en CREATOR los diferentes GPIO del microcontrolador de forma que se puedan simular operaciones de E/S antes de ejecutarlas directamente sobre el microcontrolador. También, se pretende realizar casos de uso en los que se analice el consumo de energía, así como de memoria.

Agradecimientos

Este trabajo ha sido parcialmente apoyado por la 20ª Convocatoria de Apoyo a Experiencias de Innovación Docente del curso 2022-2023 de la Universidad Carlos III de Madrid.

Referencias

[1] *CREATOR: Simulador didáctico y genérico para la programación en ensamblador*. Zenodo, July 2021.

[2] Puput Dani Prasetyo Adi and Akio Kitagawa, "A review of the blockly programming on m5stack board and mqtt based for programming education," in *2019 IEEE 11th International Conference on Engineering Education (ICEED)*, Nov 2019, pp. 102–107.

[3] David A. Patterson and John L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2017.

[4] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, Amsterdam, 5 edition, 2012.
[5] RISC-V.org, "RISC-V.org Exchange," https://riscv.org/exchange/?_sft_exchange_category=software. Accedido el 19-05-2023. [Online], 2023.
[6] Andrés Castellanos, "Jupiter," https://github.com/andrescv/Jupiter. Accedido el 19-05-2023. [Online], 2023.
[7] Benjamin Landers, "Rars," https://github.com/TheThirdOne/rars. Accedido el 14-05-2023. [Online], 2021.
[8] Keyhan Vakil Stephan Kaminsky, "Venus," https://github.com/ThaumicMekanism/venus. Accedido el 19-05-2023. [Online], 2023.
[9] Morten B Petersen, "Ripes: A visual computer architecture simulator," in *2021 ACM/IEEE Workshop on Computer Architecture Education (WCAE)*. IEEE, 2021, pp. 1–8.
[10] Roberto Giorgi and Gianfranco Mariotti, "WebRISC-V: a web-based education-oriented risc-v pipeline simulation environment," in *ACM Workshop on Computer Architecture Education (WCAE-19)*, Phoenix, AX, (USA), jun 2019, pp. 1–6.
[11] Sarah L. Harris, Daniel Chaver, Luis Piñuel, J.I. Gomez-Perez, M. Hamza Liaqat, Zubair L. Kakakhel, Olof Kindgren, and Robert Owen, "Rvfpga: Using a risc-v core targeted to an fpga in computer architecture education," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 145–150.
[12] SiFive, "SiFive HiFive1 rev-b," https://www.sifive.com/boards/hifive1-rev-b. Accedido el 19-05-2023. [Online], 2023.
[13] starfivetech, "StarFive VisionFive 2," https://www.starfivetech.com/en/site/boards. Accedido el 19-05-2023. [Online], 2023.
[14] Esspressif, "Esspressif ESP32," https://www.espressif.com. Accedido el 19-05-2023. [Online], 2023.
[15] Espressif, "Espressif ESP32-C3 Datasheet," https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf. Accedido el 19-05-2023. [Online], 2023.

Comprobación automática del funcionamiento de programas en lenguaje ensamblador del RISC-V

Miguel Turrión[1], Lidia Sánchez-González[1] y Virginia Riego[1]

Resumen— **Este artículo presenta una herramienta de web scraping para poder comprobar automáticamente el funcionamiento de programas. En concreto, los programas están realizados en lenguaje ensamblador del procesador RISC-V utilizando el simulador CREATOR, desarrollado por el grupo ARCOS de la Universidad Carlos III de Madrid. Empleando GitHub Classroom, los ejercicios desarrollados por los estudiantes se almacenan en repositorios y utilizando esta herramienta se permite comprobar su funcionamiento para así tener información de cuántos de los ejercicios propuestos están programados adecuadamente y cómo es la evolución de los estudiantes a lo largo del semestre. Consideramos que es una herramienta útil para poder realizar un seguimiento apropiado cuando el número de matriculados es elevado.**

Palabras clave— **RISC-V, Simulador CREATOR, Docencia en Arquitectura de Computadores, Corrección automática.**

I. Introducción

La materia de Arquitectura de Computadores conlleva comprender cómo funciona un procesador y, para ello, en muchas Universidades se incluye la programación en ensamblador como destreza a adquirir en la titulación de Grado en Ingeniería Informática. De esta manera, se permite entender cómo se almacenan las instrucciones de los programas en memoria, cómo se gestiona la memoria tanto para datos estáticos como otras estructuras tipo la pila.

Puesto que las asignaturas que desarrollan esta materia suelen estar en los primeros cursos, la programación en lenguaje ensamblador suele resultar tediosa para el estudiantado ya que sus destrezas y entendimiento de conceptos como funciones, punteros, paso de parámetros, acceso a memoria, entre otros, son muy incipientes. Existen varios simuladores para distintas arquitecturas que pretenden proporcionar un entorno de desarrollo amigable y en muchos casos educativo, poniendo a disponibilidad del usuario por ejemplo, el repertorio de instrucciones del procesador o visualizando distintas partes del mismo, como son los valores que toman los registros o el estado de la memoria durante la ejecución de los programas.

En este artículo, se muestra la experiencia realizada en la asignatura de Estructura de Computadores de la Universidad de León para la comprobación de la adquisición de competencias de programación en lenguaje ensamblador.

En el apartado II se dan los detalles de la asignatura donde se ha realizado la experiencia. El apartado III explica el programa realizado. Finalmente, se presentan las conclusiones alcanzadas en el apartado V.

[1]Dpto. de Ingenierías Mecánica, Informática y Aeroespacial, Universidad de León, e-mail: lidia.sanchez@unileon.es

II. Contexto de la experiencia realizada

A. Asignatura

En la asignatura de Estructura de Computadores, de 1º de Grado en Ingeniería Informática de la Universidad de León, impartida el segundo semestre, se trata, entre otras materias, la programación en ensamblador. Aunque durante muchos años se estudiaba el procesador MIPS [1], de gran popularidad en entornos educativos como así lo demuestran las guías docentes de asignaturas similares, desde el curso 2021/2022 se escogió el procesador RISC-V [2], [3], debido a las características que presenta. Dichos motivos se fundamentan principalmente en la apuesta que están haciendo tanto la Unión Europea como grandes compañías en dicho procesador [4], [5].

La asignatura de Estructura de Computadores tiene una media de unos 125 estudiantes cada año. En la parte de teoría se estudian los fundamentos de los computadores, cómo se representa la información y las distintas partes que los componen, es decir, la unidad central de proceso, incidiendo en la ruta de datos del procesador y su unidad de control, la unidad de memoria y la unidad de Entrada/Salida. Las clases prácticas complementan lo visto en teoría y, aunque en las primeras sesiones se imparte programación en lenguaje C, en las últimas sesiones se aborda el lenguaje ensamblador. El motivo de tratar la programación en lenguaje C viene impuesto por los requisitos definidos en asignaturas de segundo curso, que necesitan que tengan esos conocimientos. Aunque en un primer momento no parece la mejor solución a adoptar, en la práctica el lenguaje C se complementa muy bien con el lenguaje ensamblador puesto que se entiende mejor, entre otros, la llamada a funciones, el paso de parámetros, la gestión de memoria dinámica vista en el lenguaje de alto nivel y trabajado posteriormente con el lenguaje ensamblador. Puesto que los conocimientos en programación que tienen los estudiantes cuando comienza el segundo semestre son únicamente en el lenguaje Java, les sirve para repasar las estructuras de control utilizadas y comprender con mayor profundidad cómo se realizan las llamadas a las funciones, cómo se almacenan los datos en memoria, cómo se pasan los argumentos, etc.

B. Simulador utilizado

En las prácticas de lenguaje ensamblador, se programa empleando el lenguaje ensamblador del procesador RISC-V y aunque en los dos primeros años se empleó el simulador RARS [6], este curso 2022/2023 se ha empezado a utilizar el simulador CREATOR [7], [8], [9], desarrollado por el grupo ARCOS de la Universidad Carlos III de Madrid.

Este simulador solo necesita un navegador web y permite visualizar las distintas partes del procesador, facilitando la comprensión por parte del usuario de qué ocurre cuando se van ejecutando las instrucciones. Es muy intuitivo y permite la representación en distintos formatos, lo que facilita el seguimiento viendo cómo cambia el estado de los registros, el segmento de datos, el segmento de texto y la pila mientras se ejecutan los programas.

La figura 1 muestra cómo es la pantalla principal del simulador CREATOR, donde se puede ver cómo se cargan las instrucciones en memoria, los valores de los registros, las distintas opciones de los menús para visualizar el estado de la memoria, tanto el segmento de texto -con las instrucciones codificadas en lenguaje máquina- como el segmento de datos con los datos declarados en el programa o, en el caso de que se vaya ejecutando paso a paso, incluso cómo se va actualizando la pila.

Además, proporciona un conjunto de funcionalidades muy interesantes, ya que permite añadir instrucciones o incluso crear una nueva arquitectura, aunque estas destrezas no las trabajamos en la asignatura.

C. Organización de la docencia práctica

Puesto que la asignatura se organiza en dos sesiones prácticas semanalas de hora y media de duración, se presentan poco a poco los contenidos a trabajar en la sesión, bien sea el segmento de datos y su acceso al mismo, las distintas llamadas al sistema que proporciona el simulador, el uso de funciones, la gestión de la pila y su uso con distintos ejemplos.

Para favorecer la adquisición de las competencias que se trabajan en la asignatura, se proponen diversos ejercicios prácticos y se mantiene un ritmo de corrección semanal, proporcionando a los estudiantes realimentación de los ejercicios realizados.

Una práctica suele constar de una breve explicación del concepto que se trabaja en la práctica y 3 ó 4 ejercicios propuestos de menor a mayor dificultad que permiten trabajar dicha materia. A medida que se avanza en la asignatura, se van incluyendo ejercicios más complejos que combinan todo lo tratado en clase.

D. Herramienta de control de versiones

Al igual que hacen en las práctica de programación en C, los estudiantes han de utilizar una herramienta de control de versiones ya que de esta forma su desarrollo software se realiza de forma profesional, como muy probablemente harán cuando comience su vida laboral. Estas herramientas hacen posible que se puedan monitorizar los cambios en el código fuente, siendo posible gestionar el desarrollo cooperativo y simular un entorno de desarrollo profesional. El objetivo es también que no pierdan la costumbre de usarlo, ya que en ocasiones herramientas vistas en alguna asignatura, si no se utilizan con asiduidad durante la titulación, se olvidan y falta destreza en el uso de las mismas cuando finalizan sus estudios.

En el Grupo de Innovación Docente "Robótica, *Learning Analytics* y TICS aplicadas a los procesos de enseñanza/aprendizaje" se ha realizado en los últimos años la coordinación en el uso de herramientas en varias asignaturas para conseguir que los egresados posean competencias avanzadas en el uso de las mismas. En este contexto se ha escogido como sistema de control de versiones Git, junto con GitHub como aplicación web, ampliamente conocida en el ámbito profesional. Además, se utiliza la aplicación web GitHub Classroom ya que facilita la gestión de repositorios relacionados con las prácticas de una asignatura [10].

Entre las funcionalidades que proporciona GitHub Classroom se encuentran las siguientes: es posible automatizar la creación de repositorios, gestionando su acceso y monitorizando la actividad de los estudiantes. Además, en el grupo se han realizado distintas experiencias extrayendo datos de los repositorios de los estudiantes para realizar análisis de los mismos y estimar distintos parámetros relacionados con el rendimiento académico [11]. En la Figura 2 se muestra una captura de esta aplicación.

Mediante la supervisión de la evolución de los repositorios de los estudiantes, se puede realizar una evaluación continua y analizar cómo está siendo la adquisición de las competencias, la dedicación de los estudiantes, su progresión a lo largo de las semanas e incluso detectar los aspectos que les resultan más complejos. La Figura 3 es una muestra de cómo se listan los repositorios de los estudiantes para cada práctica, pudiendo ver si ha realizado la entrega, si ha cumplido el plazo o lo ha hecho con retraso y con un enlace para acceder directamente al repositorio en cuestión.

III. Método propuesto

En el contexto anteriormente descrito, de adquisición de competencias en programación en lenguaje ensamblador del procesador RISC-V, uso del simulador CREATOR y de la herramienta de control de versiones GitHub Classroom, se propone un programa para la navegación automática por los repositorios de los estudiantes y así comprobar el correcto funcionamiento de las prácticas subidas.

Empleando las herramientas conocidas de *web scraping* se ha desarrollado un pequeño programa que permite visitar los repositorios de los estudiantes de la asignatura. Las herramientas utilizadas han sido Node.js [12], el navegador Mozilla Firefox junto con el driver de Selenium [13] y Cheerio como analizador [14].

El algoritmo propuesto ejecuta las siguientes ta-

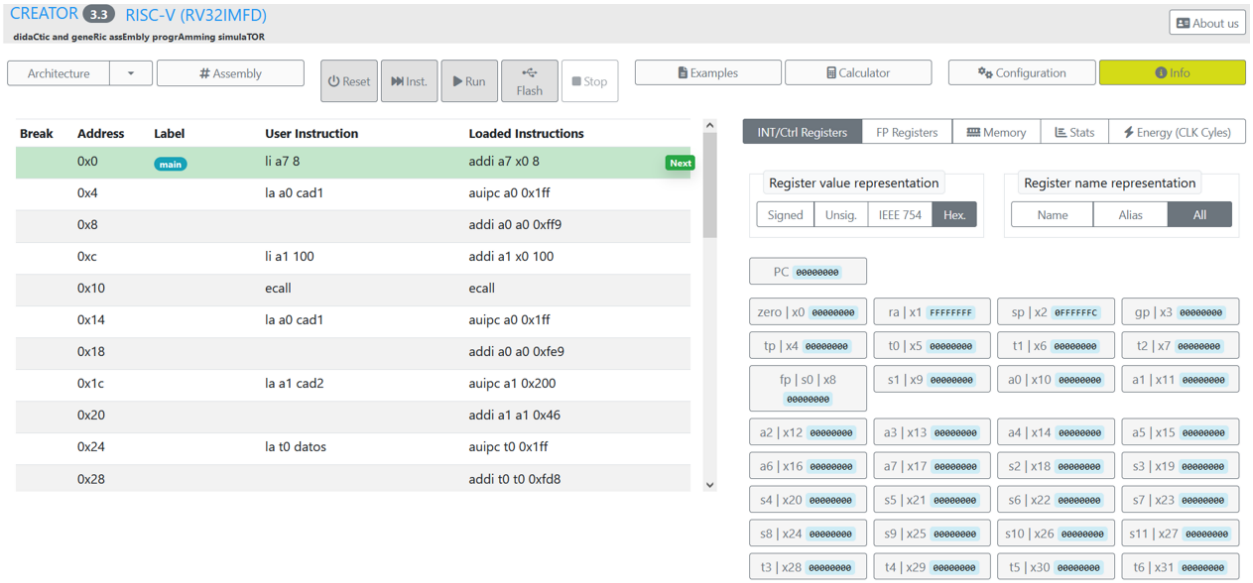


Fig. 1: Captura del simulador CREATOR.

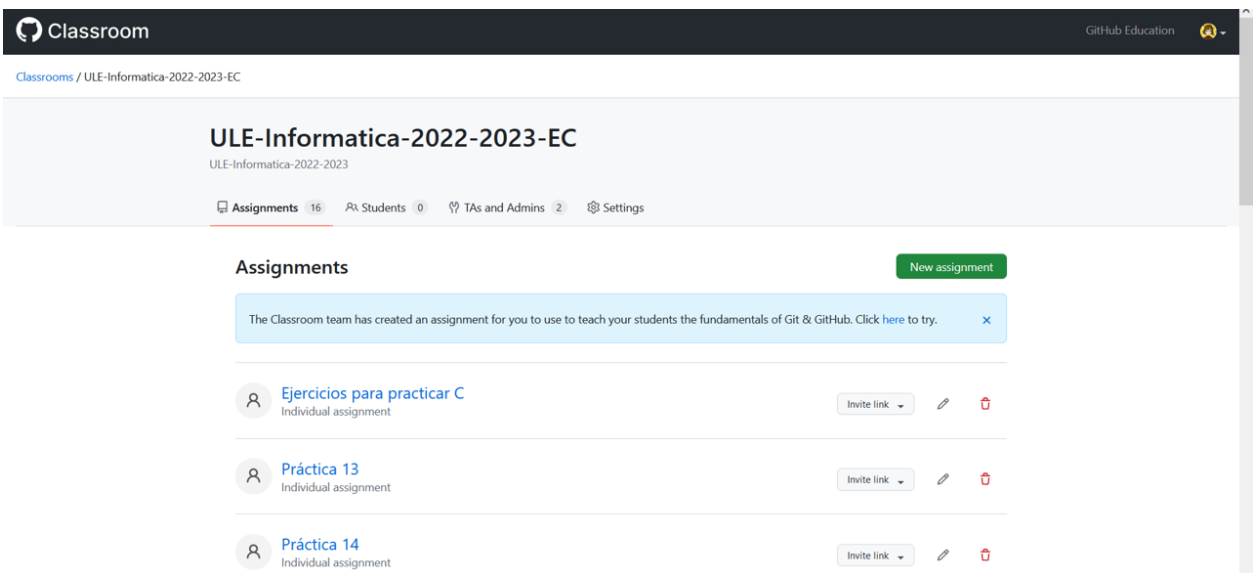


Fig. 2: Captura de la herramienta GitHub Classroom.

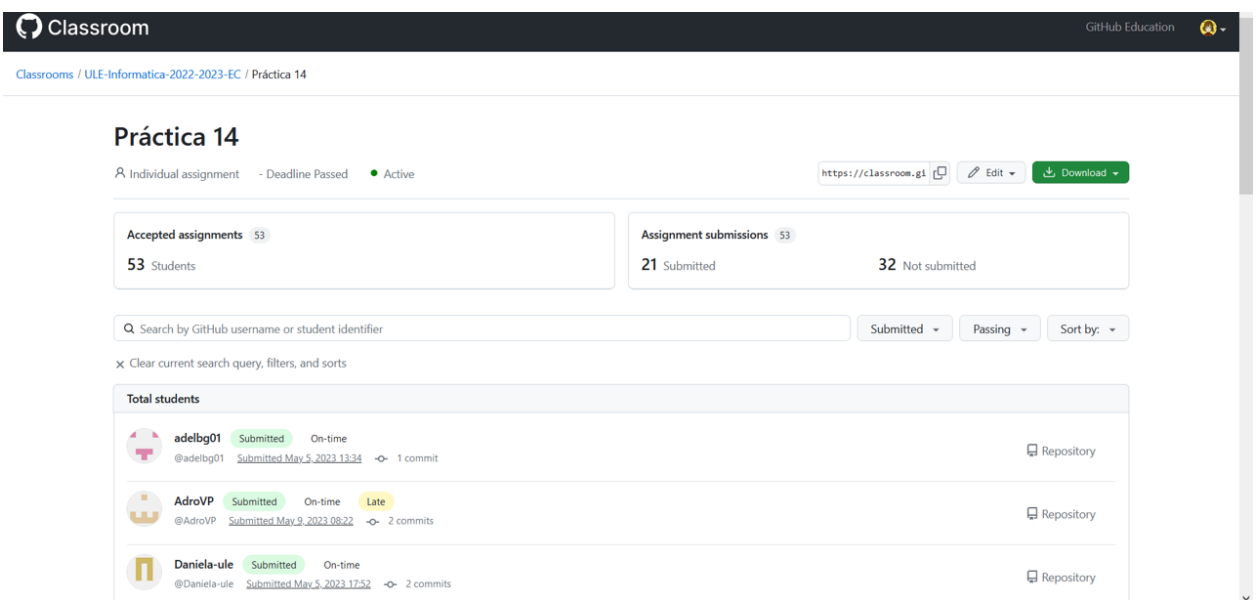


Fig. 3: Ejemplo de una práctica y cómo se listan los repositorios de los estudiantes (se omite su nombre de usuario).

reas, combinando las funcionalidades proporcionadas por Cheerio y Selenium para la navegación automática:

1. Accede a la URI especificada en el archivo de texto plano.
2. Espera que se cargue el archivo.
3. Lee el modal que se abre para saber si ha habido error al compilar o no.
4. Si no ha compilado correctamente acaba y devuelve el código del error.
5. Si ha compilado correctamente, ejecuta el programa.
   a) Repite tantas veces como entradas tenga el programa:
      - Busca el componente para insertar el texto y escribe la entrada deseada, haciendo clic en el botón `Enter` a continuación.
   b) Captura la salida del programa del componente correspondiente, comparándola con la salida deseada.

Por añadir más detalle a este algoritmo, se muestra el código con el que se comprueba si ha compilado correctamente:

```
try {
    const driver = await new Builder().
        forBrowser(Browser.FIREFOX).build();
    await driver.get(url);
    await waitModalError(driver);
    const element = await driver.findElement(By.
        css('body'));
    const element_text = await element.
        getAttribute('innerHTML');
    const $ = cheerio.load(element_text);
    const compileError = $('#
        modalAssemblyError___BV_modal_outer_');

    if(compileError.length) {
        console.log('COMPILE ERROR:');
        const errorCode = $(compileError).find
            ('.errorAssembly').find('.row');
        const len = errorCode.length;
        for(var i = 0; i < len; i++) {
            console.log(i+ ' : ' +$(errorCode[i
                ]).text());
        }
        await driver.quit();
    } else {
        console.log('COMPILE SUCCESS');
        runProgram(driver, input, output);
    }
} catch(e) {
    console.log('ERROR COMPILE:')
    console.log(e.message);
}
```

Para comprobar si la salida es la apropiada, hay que ejecutar el programa, pasarle unos datos de entrada y comprobar si la salida es correcta. Eso se realiza con la siguiente función:

```
async function runProgram(driver, input, output)
    {
    try {
        await driver.executeScript("document.
            getElementById('playExecution').
            click();");
        if(input !== '') {
            const inputElement = await driver.
                findElement(By.id('
                textarea_keyboard'));
            await driver.executeScript("
                arguments[0].scrollIntoView(
                true);", inputElement);
            await driver.actions().sendKeys(
                inputElement, input).perform();

            const inputButton = await driver.
                findElement(By.id('
                enter_keyboard'));
            await inputButton.click();

            await new Promise((resolve) => {
                setTimeout(() => { resolve(true)
                    ; }, 500);
            });
            await inputButton.click();
        }
        await driver.executeScript("window.
            scrollTo(0, 0);");
        await new Promise((resolve) => {
            setTimeout(() => {
            resolve(true); }, 1000); });
        await driver.wait(until.elementLocated(
            By.className('toast-body')), 5000);
        const toastFinish = await driver.
            findElements(By.className('toast-
            body'));
        const len = toastFinish.length;
        for(var i = len - 1; i >= 0; i--) {
            console.log(await toastFinish[i].
                getAttribute('innerHTML'));
        }
        if(output !== '') {
            const outputElement = await driver.
                findElement(By.id('
                textarea_display'));
            await driver.executeScript("
                arguments[0].scrollIntoView(
                true);", outputElement);
            const outputText = await
                outputElement.getAttribute('
                value');
            console.log('OUTPUT: (' +output+ '
                == ' +outputText+ ') =' +(
                output==outputText));
        }
    } catch(e) {
        console.log('ERROR EXECUTE:')
        console.log(e.message);
    } finally {
        await driver.quit();
    }
}
```

IV. Resultados experimentales

La evaluación del método se ha realizado con los repositorios creados para 10 prácticas de la asignatura de Estructura de Computadores donde se trabajaba la competencia de programación en lenguaje ensamblador. Salvo las sesiones donde hay algún festivo, es común que se realicen dos prácticas semanales de hora y media de duración, en las que se aborda una parte específica del lenguaje ensamblador y se proponen una serie de ejercicios. Puesto que el número de matriculados es elevado, en ocasiones hay que identificar qué ejercicios presentan más dificultad y hay que repasar en futuras sesiones, permitiendo una adaptación semanal a las necesidades de los estudiantes.

En las indicaciones de los enunciados de las prácticas, se les pide que además de subir el archivo con el código fuente, se refleje en un archivo de texto plano, la dirección URI del ejercicio escrito en lenguaje ensamblador, funcionalidad proporcionada por el simulador.

Mediante el programa desarrollado, se accede a dicho archivo de texto plano, se lee la dirección URI y se accede a ella a través del navegador, automatizando la ejecución del programa desarrollado en lenguaje ensamblador, ya que incluye la escritura de las entradas correspondientes y la comparación de la salida obtenida con la esperada, pudiendo evaluar el correcto funcionamiento de la misma.

En la Figura 4 se muestra cómo se navega automáticamente en este caso para un programa que calcula la longitud de la cadena, pudiendo introducir una cadena determinada "Esto es una prueba22" y comprobando que la longitud es 20 (Fig. 5).

De esta forma, se consigue comprobar el funcionamiento de las prácticas subidas por los estudiantes e identificar aquellos ejercicios que no funcionan correctamente para así determinar en qué conceptos hay que incidir en sesiones futuras.

V. Conclusiones

En este trabajo se presenta una prueba de concepto para la comprobación del funcionamiento de las prácticas en lenguaje ensamblador del procesador RISC-V realizadas por los estudiantes de la asignatura de Estructura de Computadores. Combinando distintas tecnologías como GitHub Classroom, el simulador de RISC-V CREATOR y un programa de *web scraping*, se visitan los repositorios de los estudiantes y se ejecuta el código que han desarrollado, pudiendo conocer el estado de sus prácticas de forma automática. Esto facilita el seguimiento diario cuando la asignatura tiene un elevado número de alumnos y hay dos prácticas semanales, con sus correspondientes ejercicios. Una mejora a implementar sería analizar el código desarrollado, no solo su funcionamiento, permitiendo comprobar que el uso de registros y de instrucciones es apropiado, si los programas está bien organizados, etc. Como experiencia inicial en este ámbito se considera de interés para los docentes puesto que proporciona información sobre el proceso de enseñanza/aprendizaje.

Agradecimientos

Este proyecto ha sido posible gracias al apoyo del Grupo de Robótica de la Universidad de León, al Grupo de Innovación Docente Robótica, *Learning Analytics* y TICS aplicadas a los procesos de enseñanza/aprendizaje. El presente trabajo ha sido financiado mediante el Plan de Apoyo al a Innovación Docente de la Universidad de León.

Referencias

[1] D.A. Patterson and J.L. Hennessy, *Computer Organization and Design MIPS Edition: The Hardware/Software Interface*, ISSN. Elsevier Science, 2020.

[2] David A. Patterson and John L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2017.

[3] S. Harris and D. Harris, *Digital Design and Computer Architecture, RISC-V Edition*, Elsevier Science, 2021.

[4] Consortium with the European Commission, "European processor initiative (EPI)," `https://www.european-processor-initiative.eu/`, 2022, [Online; accessed 20-May-2023].

[5] Aimee Kalnoskas, "RISC-V: The background, the benefits, and the future," `https://www.microcontrollertips.com/risc-v-background-benefits-and-future-faq/`, 2022, [Online; accessed 20-May-2023].

[6] TheThirdOne, "RARS – RISC-V assembler and runtime simulator," `https://github.com/TheThirdOne/rars`, 2023, [Online; accessed 20-May-2023].

[7] Grupo ARCOS, "didaCtic and geneRic assEmbly progrAmming simulaTOR," `https://creatorsim.github.io/`, 2023, [Online; accessed 20-May-2023].

[8] *CREATOR: Simulador didáctico y genérico para la programación en ensamblador*. Zenodo, July 2021.

[9] Diego Camarmas-Alonso, Félix García-Carballeira, Elías Del-Pozo-Puñal, and Alejandro Calderón Mateos, "A new generic simulator for the teaching of assembly programming," in *2021 XLVII Latin American Computing Conference (CLEI)*, 2021, pp. 1–9.

[10] Inc. GitHub, "Enseña con github classroom. aprende cómo configurar tu aula y tus tareas," `https://docs.github.com/es/education/manage-coursework-with-github-classroom/teach-with-github-classroom`, 2023, [Online; accessed 12-April-2023].

[11] Ángel Manuel Guerrero-Higueras, Camino Fernández Llamas, Lidia Sánchez González, Alexis Gutierrez Fernández, Gonzalo Esteban Costales, and Miguel Ángel Conde González, "Academic success assessment through version control systems," *Applied Sciences*, vol. 10, no. 4, 2020.

[12] The OpenJS Foundation, "Node.js," `https://nodejs.org/es`, 2023, [Online; accessed 12-April-2023].

[13] Software Freedom Conservancy, "Selenium automates browsers. that's it!," `https://www.selenium.dev/`, 2023, [Online; accessed 12-April-2023].

[14] The Cheerio contributors, "Cheerio the fast, flexible & elegant library for parsing and manipulating html and xml," `https://cheerio.js.org/`, 2023, [Online; accessed 12-April-2023].

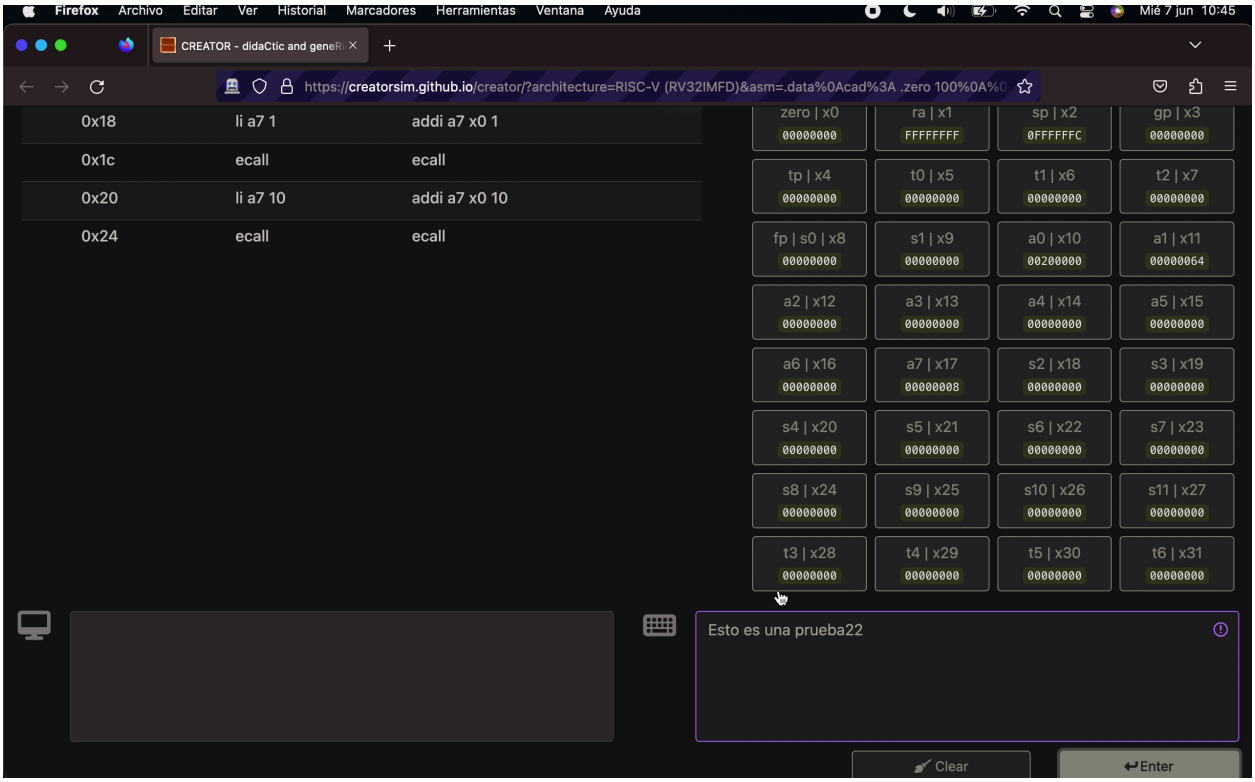


Fig. 4: Entrada automática de datos.

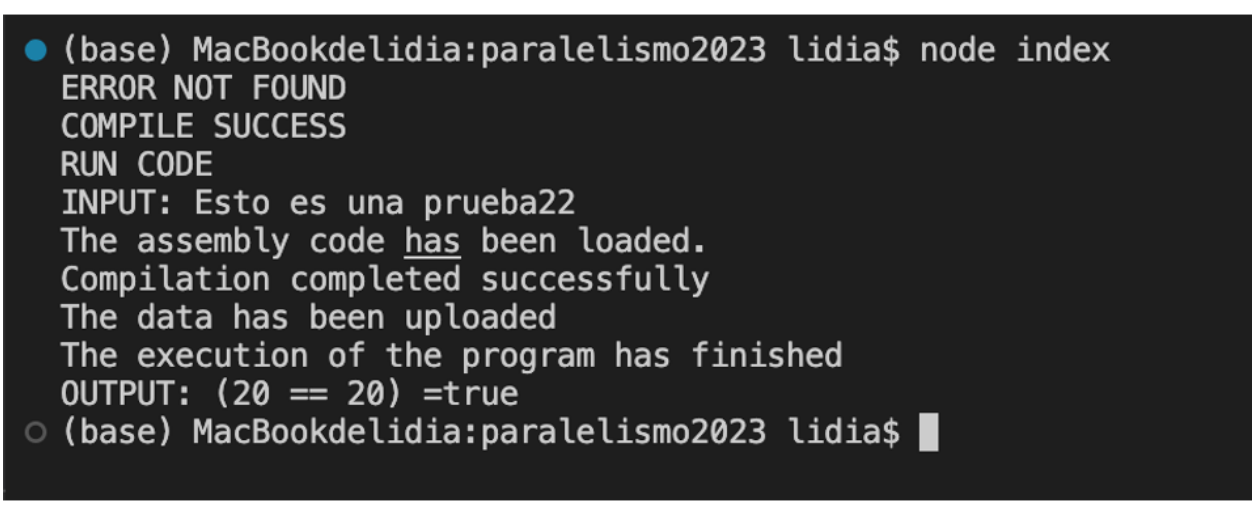


Fig. 5: Comprobación de que la salida es 20, la longitud esperada.

Tecnologías clúster, plataformas distribuidas, Big Data y Deep Learning

Heterogeneous Data Centre Task Scheduling with Deep Reinforcement Learning

Jaime Fomperosa, Mario Ibañez, Esteban Stafford, Jose Luis Bosque [1]

Abstract— **This article advocates for the leveraging of machine learning to develop a workload manager that will improve the efficiency of modern data centres. The proposals stem from an existing tool that allows training deep reinforcement agents for this purpose. However, it incorporates several major improvements. It confers the ability to model heterogeneous data centres and then it proposes a novel learning agent that can not only choose the most adequate job for scheduling, but also determines the best compute resources for its execution. The evaluation experiments compare the performance of this learning agent against well known heuristic algorithms, revealing that the former is capable of improving the scheduling.**

Keywords— **Deep Reinforcement Learning, Task scheduling, Heterogeneous data centres, Machine Learning.**

I. Introduction

Modern Information Technology (IT) relies heavily on *data centres* which host massive amounts of interconnected computers. A subset of these data centres support the scientific and engineering communities with high performance computing services. The computers that integrate these combine their processing capabilities to accelerate the execution of complex problems [1].

To harness the power of computer clusters, data centres rely on a Workload Manager. It is in charge of job scheduling, or choosing jobs awaiting execution and assigning them to computing resources of the data centre. But this is an NP-Complete problem that cannot be solved in polynomial time. This is exacerbated by the huge growth of data centres [2], the wide variety and heterogeneity of architectures and configurations they host [3], [4]. This means that the decision space of the workload manager has increased substantially, and consequently so has the difficulty of finding optimal solutions to the problem. It is possible to find *near-optimal* solutions using *approximation* methods [5] or *heuristic* [6] algorithms. The latter are commonly found at the core of modern resource managers, like Slurm [7]. They are characterised by sacrificing optimality for speed, which is a necessary compromise.

These heuristic algorithms are fairly simple. Three algorithms are usually implemented nowadays: First In, First Out (FIFO), Shortest Job First (SJF) [8] and BackFill [9]. There are more complex algorithms that consider several attributes of each job in order to compute a score, which is then used to sort and prioritize them, such as WFP3 or UNICEP [10] or F1 [11]. However, these have difficulties in adapting to changes in the resources, the type of job or the objectives. Recently, machine learning has shown its adaptability to different scenarios, contrasting with the static approach of heuristic algorithms [12], [13].

Reinforcement Learning (RL) is a branch of machine learning that can autonomously improve its behaviour through trial and error. A key advantage of this approach is that it can consider many more parameters than heuristic algorithms and learn which are the most important. In this context *IRMaSim* [14], emerges as a tool to develop and test reinforcement learning algorithms on a simulator of heterogeneous data centres. A further development of this idea is *RLScheduler* [15]. Its results are fairly good despite its coarse simulator, where only homogeneous data centers with identical compute devices can be modeled.

The main *hypothesis* of this article is that the Deep Reinforcement Learning (DRL) techniques used in [15] can be adapted to schedule jobs in a heterogeneous data centre and with better performance than state-of-the-art heuristic algorithms. The pursuit of this hypothesis requires the completion of three steps. First, the definition of an environment that adequately represents heterogeneous data centres. To this end the cores of the cluster are grouped into nodes with possibly differing properties. Second, is the development of the agent itself, deciding its internal structure, how is the information from the environment fed to it, and how is the action selected. And third, an evaluation procedure must be devised where the performance of the agent is compared to that of well known heuristic algorithms.

The experimental results presented in the evaluation section show two important conclusions. First, that heterogeneity poses new challenges to the scheduling problem, even for classic algorithms that are optimal in homogeneous systems. Secondly, the proposed agent is able to obtain better results in all the studied objectives than heuristic algorithms, which confirms that machine learning-based scheduling is an important new field of study.

The remainder of this article is organised as follows. Section II gives an overview of reinforcement learning resource managers. Section III describes the main proposals of the article. Section IV presents the evaluation methodology and discusses its results. Finally, a summary with the most important conclusions of the article is in Section V.

[1]Dpto. de Ingeniería Informática y Electrónica, Universidad de Cantabria, Spain e-mail: {jaime.fomperosa, mario.ibanez, esteban.stafford, joseluis.bosque}@unican.es

II. Background

Reinforcement learning systems usually revolve around the concept of an *agent* that must drive the behaviour of the *environment* in order to reach a given *objective*. The agent is in charge of making decisions that affect the environment in some manner, and its aim is to learn how to satisfy the objective. Internally the agent is implemented with a *Deep Neural Network (DNN)* that, before going into production, must be trained. This is done by exposing the environment to stimuli, the agent considers the consequences of the *actions* it takes and it progressively learns which ones are better than others.

The training process is divided in *epochs*, or iterations of sets of stimuli. In turn, epochs consist of a series of *steps*, representing the processing of a given stimulus [16]. In each step the agent performs an *action* that has an impact in the environment. This is measured through *observations* and qualified by a *reward* value that indicates whether the impact was positive or negative. At the end of each epoch, the agent evaluates these and encourages those actions that helped in reaching the objective. After experiencing a number of epochs, the agent converges to using a particular set of actions. These are the ones that maximise the rewards it obtains, and therefore, satisfies the objective.

In the context of resource managers, the environment represents the compute resources of a data centre and the set of jobs, or workload, to be executed. The agent must observe the incoming jobs and the state of the data centre, and decide which job is allocated to which resource in order to achieve an optimization objective, e.g. slowdown or average waiting time. The jobs are usually stored in a *workload queue*, which can potentially be very long and become unmanageable. Modern resource managers, use an *eligible job queue*, which is a fixed length queue that holds the oldest jobs pending execution. The scheduler only considers jobs in this queue for execution, and when one gets chosen, it vacates the queue leaving space for another job from the workload queue.

RLScheduler combines a reinforcement learning resource manager with a simplistic data centre simulator to accelerate the training process [15]. The simulated environment defines a number of computational resources, all with the same characteristics. Then it is only necessary to keep a number of free resources to represent the status of the data centre. And knowing to which processors in particular the job is assigned does not really matter. In RLScheduler, an observation represents the state of the environment by means of a vector that contains the attributes of all the eligible jobs.

The simplicity of RLScheduler is also its major drawback, as it considers the resources to be identical and unrelated. On contrast, modern data centres are heterogeneous and structured, as they host compute nodes with different number of processors or cores. These can have different architectures and compute capacities, which can have a great impact on scheduling. In addition, some applications must execute on processors belonging to the same node. Rising to these challenges is the main objective of this paper. Thus, a redesign of RLScheduler is proposed that will allow the modeling of heterogeneous systems. As a consequence, it will train agents to decide on which job to schedule and to which resource it will be assigned.

III. DRL for Scheduling in Heterogeneous Data Centres

This section details the improvements made to RLScheduler allowing its use in heterogeneous and structured data centres, and also proposes a scheduler agent that is able to select the best possible combinations of job and node. In order to adequately model these systems, the simulated environment must keep track of the jobs assigned to each node and their attributes to properly predict the execution time of the jobs. This also increases the amount of information that must be taken into account by the agent to make the best possible scheduling decisions. As a consequence, the observation and actions spaces must be redefined.

A. Observation and Action Spaces

The observation space must be able to represent the state of the environment that the agent will use to decide its next action. Similarly, the action space contains all the possible actions an agent can take over the environment. As mentioned in the previous section, RLScheduler considers all the computational resources of the data centre to have the same properties, making it unnecessary to identify which resources are allocated to each job. However, in heterogeneous data centres it is imperative that the compute resources are represented as a set of nodes with different number of processors. This information must be included in the observation space. Therefore, it is divided in two sets of attributes, the *Node Observation* representing the state of the data centre nodes, and the *Job Observation* containing the job information.

The proposed representation of the computational resources is based on the concept of *node*. Each one can have a different *size* and *speed*, regarding the number of processors it contains and its clock frequency. As for the jobs, they are considered memory-sharing embarrassingly parallel applications requesting a number of *processors*. Meaning that a job cannot be assigned to more than one node, that the node must have enough free processors to host the complete job, and that there is no communication overhead. The reason behind this decision is to streamline the simulator model. The proposed set of attributes for the observation space is shown in Table I. The number of attributes is lower than in real resource managers, but since the model is expandable, it is fairly simple to add new attributes for the agent to consider.

Field Name	Notation	Description
Job Observation Space		
Requested Processors	n_j	Number of processors requested for the job
Requested Time	r_j	Amount of time requested for the job
Wait Time	w_j	Amount of time spent by the job in the job queue
Node Observation Space		
Total Processors	tp_n	Number of processors in the node
Free Processors	fp_n	Free processors in the node
Frequency	f_n	CPU clock rate of the node processors

Table I: List of attributes of the Job and Node Observations.

The action space has also been improved to accommodate the node concept. The agent must not only decide which is the next job to be scheduled, but also to which resource it is allocated. This translates into a new bidimensional action space, where one dimension covers the jobs in the eligible job queue and the other represents the nodes in the data centre.

B. Agent architecture

The proposed agent adheres to the actor-critic architecture, which is common in DRL systems. It combines the use of two similar networks, the actor decides on the next action, while the critic evaluates the performance of the actor. This structure tends to improve training times. Compared to the agent in [15], there are changes in the observation and action spaces that have a significant impact on its design. A diagram describing the new agent, as well as its relationship with said spaces is shown in Figure 1.

Since the input to the DNN has to have a fixed size and the number of jobs in the queue varies over time, an eligible job queue is used with the first 128 pending jobs. This value is the same as in RLScheduler and is also common practice in workload managers such as Slurm. The agent considers the jobs in the eligible job queue and their three corresponding attributes, composing the Job Observation, an array of size 128×3 **(1a)**. Simultaneously it obtains information about the nodes through the Node Observation array. It has as many rows as nodes available in the data centre and three columns, one for each node attribute **(1b)**.

Next, the agent prepares the observation by merging the Job Observation and the Node Observation attributes of each combination of job and node, and adds an extra value called *Can Be Scheduled* **(2)**, which is defined as $c_{j,n} = n_j \leq fp_n$ resulting in true if the node n has room for the job j and false otherwise. This combined observation is a matrix with $128 \times NumNodes$ rows and $JobAttributes + NodeAttributes + 1$. Here, the rows represent all the possible pairings of jobs and nodes, and the columns are the total number of attributes that define each of these pairs, which in this instance it is equal to seven. The fact that the Node and Job Observations are combined serves the purposes of presenting the agent all the possible pairings of nodes and jobs, and allow it to make decisions with sufficient information.

The next step is to let the agent select from the observation the job-node pair to be scheduled. This decision is reached with the aid of the actor network of the agent **(3a)**, which has seven inputs, one for each attribute in the observation. Then it has three fully-connected hidden layers of 32, 16 and 8 neurons each, with ReLU as their activation function. Finally, the output layer is of size 1, as purpose of this network is to provide a single score value for each jobs-node pairs. The actor is fed the whole observation matrix, therefore, the output is also not a single score but a column vector of $128 \times NumNodes$ scores. Then, a mask is applied to the score vector to remove the values corresponding to padding jobs added to complete the observation, or those that request more processors than those free in the node **(4)**. This way, any job that the agent may choose is assured to be able to be scheduled without waiting for resources to get free.

Next, a softmax function is applied to the masked vector, transforming the scores into a probability distribution in which the sum of all elements is 1 **(5)**. With these probabilities, an action is selected that will indicate the job and node to be scheduled next, favouring those with higher score. In production this step is skipped and the job-node pair with the highest score is chosen. The job-node pair is an integer $a_i \in [0, 128 \times NumNodes - 1]$, the index of the job is calculated as $\lfloor \frac{i}{128} \rfloor$ and the node as $i \bmod NumNodes$ **(6)**.

This action is passed on to the simulator that executes the corresponding scheduling operation. After the simulator advances the time to the next event, which results in a new state of the environment, a reward **(7)** will be obtained based on the chosen objective, together with a new observation representing the new state of the environment. These are used by the critic network in the agent **(3b)** to evaluate the performance of the actor network. It guides the training process of both networks toward a state where the agent consistently schedules the jobs to the right computing resources, such that the objective is satisfied. The goal of the critic network is to predict the reward that a set of jobs would produce with the given objective.

Three reward functions have been implemented, each corresponding to a different scheduling metric to be minimised. If e_i and w_i are the execution and wait times of job i, the metrics are

- *Slowdown (SLD)*: is the average slowdown, defined as $\frac{w_i + e_i}{e_i}$, for all the jobs. This metric can

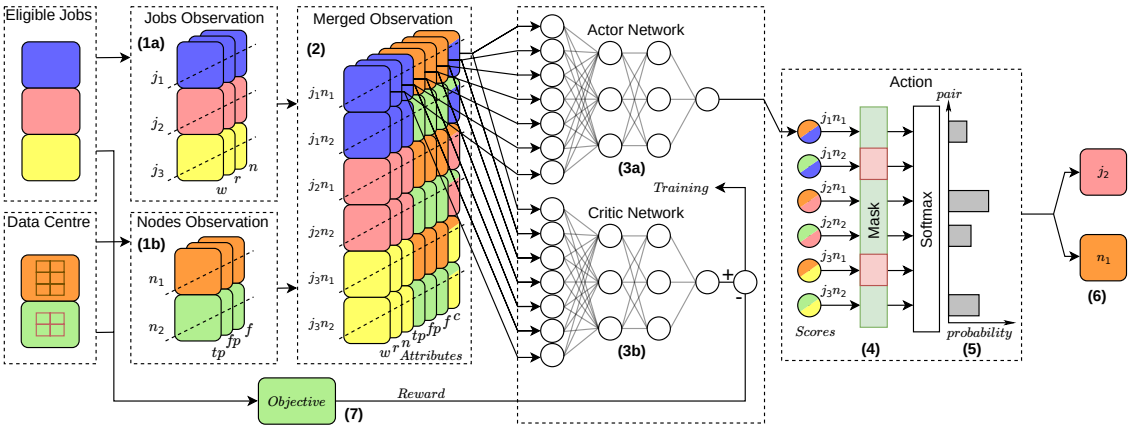


Fig. 1: Proposed agent with observation and actions spaces for three jobs and two nodes.

give very high values when jobs are short.

- *Average bounded slowdown (BSLD)*: variation of the slowdown that is less sensitive to very short execution times. The bounded slowdown of a job is defined as $max((w_i + e_i)/max(e_i, 10), 1)$.
- *Average waiting time (AVGW)*: simply averages w_i of all jobs.

It is worth noting that all the objectives are related, since they strive to reduce the delay in the execution of the jobs. However, the average the bounded slowdown metric is better suited to agent training than the other two. First, it conveys more information than the average waiting time because it includes the execution time of the jobs, and second, it is more stable than the average slowdown, because it avoids giving the very high slowdown values of jobs with execution times that are too short.

C. Size reduction through clustering

A drawback of this agent is the large size of the input. For instance, in a data centre with 16 nodes, the total number of elements of both observations would be $128 \times 16 \times 7 = 14336$. Doubling the number of nodes in the data centre results in 28672 elements, which has a clear impact in the performance and the scalability. Since the size of the queue is fixed to 128, reducing the size of the observation can only be done by limiting the size of the node observation. This section describes how this can be accomplished with clustering techniques.

In a data centre, it is common that many nodes have a similar situation, either due to their equivalent architectural properties or similar load. Then, it is not necessary to identify exactly which node is going to receive a job, and it suffices to indicate what kind of node is the target. Taking this into account, the n nodes of the data centre can be grouped in k *clusters* of similar attributes using the *k-means* algorithm [17]. The attributes of each cluster are calculated as their mean value for all the nodes in each one. This is applied to the node observation **(1b)** before it is merged to the job observation (Fig. 1), which now carries job-cluster pairs instead of job-node pairs. Like before an attribute c is calculated, indicating if the job fits in at least one node in the cluster.

The final step after the job-cluster selection has been made is to choose a specific node for scheduling the job, which is done by simply finding the first node of the cluster that can execute the job. This selection does not need any further considerations, as the assumption is that nodes in the same cluster are similar enough. By grouping the nodes in a fixed number of clusters, the size of the node observation becomes constant. Thus, it is possible to increase the number of nodes in the platform without complicating the agent.

IV. Evaluation

The proposed agent is evaluated through four instances. The *SqSLD agent*, *SqBSDL agent* and *SqAVGW agent* aim to minimise the slowdown, average bounded slowdown and waiting time, respectively. The *ClBSLD agent* uses clustering of the compute resources to minimize the average bounded slowdown, although any of the other two objectives could have been employed.

The target system is a heterogeneous data centre with 20 nodes. Each can have between 4 and 64 processors, running at 2, 2.5, 3 or 3.5 GHz. The workload used is generated from models defined in the Parallel Workloads Archive, *Lublin, 1999/2003*, commonly used in HPC [18], [19]. This workload is composed of 10 000 jobs with varying required processors and execution times.

Also, a set of heuristic algorithms were used for comparison. They are able to select jobs and the nodes to execute them. These result by combining two algorithms, one to choose the job and another for the resource. These are summarised in Table II, then algorithm xy combines job selection x with node selection y.

The hyper-parameters used to control the training process of the agents are mostly the same as in RLScheduler. The most relevant ones are the learning rate α, with values of 0.0003 and 0.001 for the actor and the critic networks, respectively, and gamma γ is equal to 0.99.

To explore the training phase, each instance of the agent is subjected to 100 epochs and the evolution of

Table II: Heuristic job and node selection algorithms.

Name	Symbol	Description
Job Selection		
Random	r	Random job from the job queue is selected
First	f	Job with lowest submit time is selected
Shortest	s	Job with lowest requested run time is selected
Smallest	l	Chooses job with lowest requested number of processors
Node Selection		
Random	r	Random node is selected
Biggest node	b	Node with highest number of processors is selected
Fastest node	f	Node with highest frequency is selected

the process is observed to ensure that it converges. To lighten this process, the workload trace is not used in its full length. One training epoch consists of 20 *trajectories*, which are sets of 256 consecutive jobs, taken at a random time from the original trace. The experimental results show that 100 epochs are more that enough because convergence was observed after 60 epochs, since the behaviour did not improve in the following epochs.

Once the training phase concludes, the inference stage is evaluated. The trained agents must schedule trajectories of 1024 jobs extracted from the same workload. Note that the presented results consider 20 repetitions to avoid obtaining wrong conclusions due to outliers. The scheduling results are evaluated by comparing the behaviour of the trained instances to that of the heuristic algorithms. These are shown in graphs where the horizontal axis represents the values of the metrics used, and the vertical axis shows the different schedulers, sorted by the median. To avoid clutter, only the results for the best heuristic algorithms are shown. The graphs combine box-and-whisker and violin representations of the results. The box shows the 25 and 75 percentile, the line in the box indicates the median, and the whiskers represent extreme values. The violin plots show result distribution, where fatter parts indicate a higher data density.

Figures 2, 3 and 4 show the promising results of the four agents. In general, the results prove that intelligent schedulers can perform better than the state-of-the-art algorithms in a heterogeneous data centre, at least for the objectives considered here.

Note how the algorithms that select the random or first jobs, the first six in the graphs, give very bad results. In some cases, more than tripling the results of the corresponding agent. The algorithms that choose the shortest or smallest jobs exhibit a better performance, especially with the random or fastest resource selection policies.

Considering only the heuristic algorithms, the graphs show the importance of choosing the right

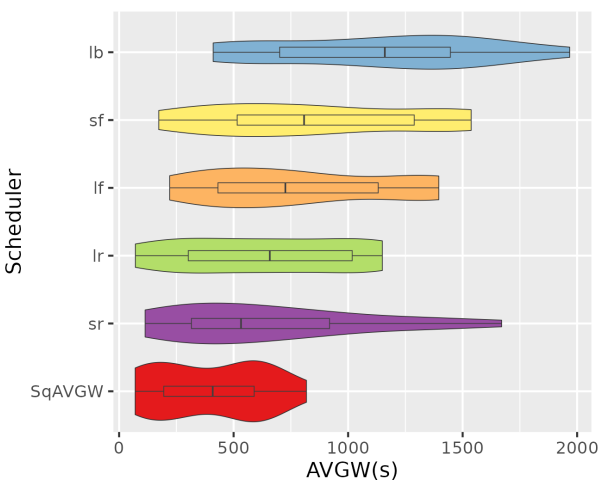


Fig. 3: Average waiting time results for heuristic algorithms and SqAVGW Agent.

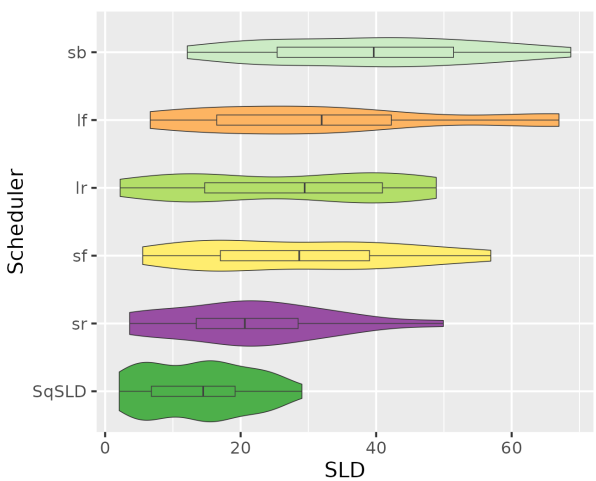


Fig. 2: Average slowdown results for heuristic algorithms and SqSLD Agent.

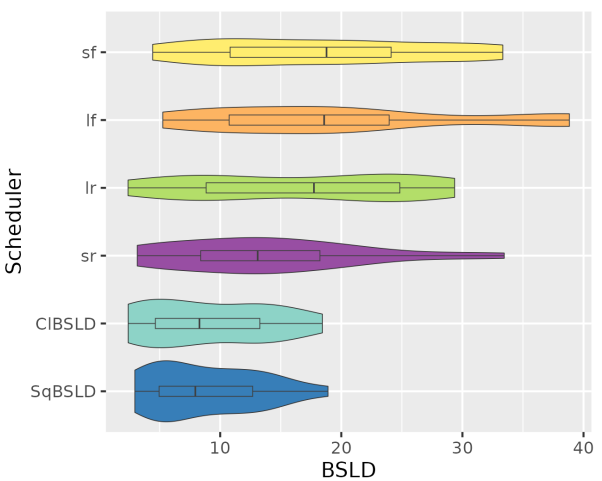


Fig. 4: Average bounded slowdown results for heuristic algorithms, SqBSLD and ClBSLD Agents.

node for a job in the context of a heterogeneous data centre. Note that only the algorithms that choose the shortest or smallest jobs first appear in the graphs. The rest had significantly worse results and were excluded to avoid clutter. It can be seen that *sr* has always the lowest median, followed by *sf* or *lr*, depending on the metric. As for node selection policy, the best results are obtained by either random or fastest. It is noticeable that *lr* presents the lowest values in all three metrics.

However, all these algorithms are always bested by the intelligent agents. Indeed, the graphs show that the median is always lower than that of the best heuristic algorithm *sr*, also the minimum values of the agents are similar to the *lr*. But in all cases they have lower variance than any of the heuristic algorithms, meaning that good results are given in a more consistent manner.

An improvement was proposed where the complexity of the agent was reduced by incorporating a clustering algorithm to group the nodes of the data centre. As this evaluation aims only to establish the cost-benefit relation of adding the clustering vs. reducing the complexity of the agent, only one objective has been tested, the average bounded slowdown. The 20 nodes of the system were grouped into 10 clusters, consequently, the complexity of the DNN was reduced in half.

The results of the ClBSLD Agent (Fig. 4) are comparable to those of the SqBSDL Agent. The minimum result given by the ClBSLD is smaller than the SqBSDL, but since it has higher variance, the median ends up being slightly higher. At any rate, the experiment proves that the clustering method can be applied in cases where the combined observation has become too large, and many of the nodes have the same or very similar characteristics.

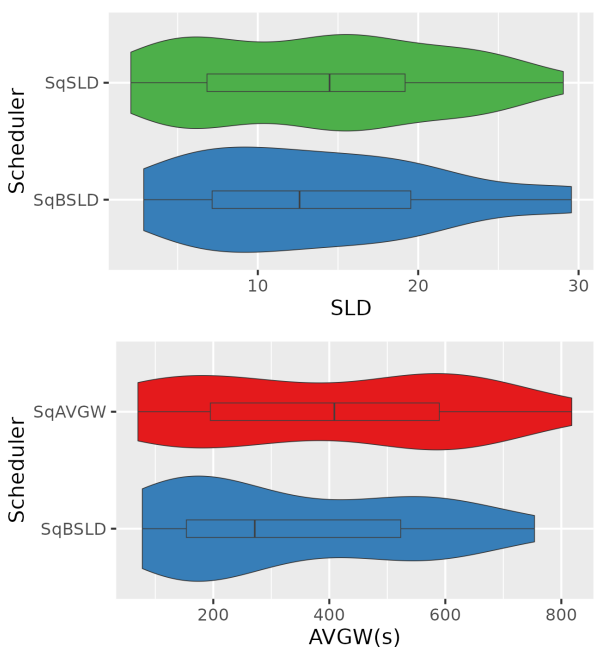


Fig. 5: Agent performance with metrics different of the one used for training.

It is interesting to observe the results of the agents with metrics different to the ones used for training. To this aim, the SqBSDL Agent instance, trained to minimise the bounded slowdown, was selected and tested with the other metrics, average slowdown and waiting time. The results are shown in Figure 5, compared to the results of the SqSLD and SqAVGW Agents. In both cases the results of the agents are roughly similar. Although the median values of SqBSDL are lower than those of the other two, the best case results are always obtained by the other instances. This is explained by the fact that bounded slowdown is better suited to agent training, and therefore, it is able to give a better scheduling.

The above evaluation proves that an intelligent agent is able to learn how to take scheduling actions and obtain better results than classic scheduling algorithms. And this can be done not for a single goal but for different ones, only constrained by the capabilities of the simulator in which it is working. All this suggests that using a machine learning agent to schedule a real data centre is an idea worth considering. Provided that it is possible to obtain a trace of the jobs typically executed in the system to perform the training of the agent.

V. Conclusions

The fact that data centres are more and more heterogeneous, combined with the variety of the applications and their requirements, complicates scheduling significantly. With homogeneous clusters, heuristic algorithms are used to schedule jobs, but in heterogeneous ones it is crucial to decide also to which compute resource they scheduled. This problem is no longer possible to solve with such algorithms and there has been advances in employing machine learning instead.

This article presents a first approach to solving the scheduling problem in heterogeneous clusters with deep reinforcement learning. To this aim, it was necessary to redefine the observation space of the agent, allowing it to perceive more data from the environment. As well as to broaden the action space to accommodate the fact that not only jobs but also nodes had to be selected.

Also, two different agents were developed capable of successfully processing the state of a small heterogeneous data centre and learning to choose adequate scheduling actions. The second agent is a refinement of the first that through the use of clustering techniques is capable of giving the similar performance using a fraction of the memory requirements. The successful training of the agents was possible thanks to the development of a simulation infrastructure with a simplistic model of a heterogeneous data centre, that can simulate nodes with a different number of processors and frequencies.

The evaluation included in this article suggests that it is possible to replace heuristic schedulers with ones that leverage machine learning techniques. The experiments show that the behaviour of the machine

learning agent gives very promising results, compared to well known heuristic algorithms.

Next developments could see larger clusters simulated with more detail, in which contention could be modeled, like that appearing in memory or network access. Furthermore, the set of objectives to optimise by the scheduler could be increased by considering energy related metrics.

Acknowledgment

This work has been supported by the Spanish Science and Technology Commission under contract PID2019-105660RB-C22 and the European HiPEAC Network of Excellence.

Bibliography

[1] Diego García-Saiz, Marta E. Zorrilla, and José Luis Bosque, "A clustering-based knowledge discovery process for data centre infrastructure management," *J. Supercomput.*, vol. 73, no. 1, pp. 215–226, 2017.

[2] José Luis Bosque and L. P. Perez, "Theoretical scalability analysis for heterogeneous clusters," in *4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2004), Chicago USA*. 2004, pp. 285–292, IEEE Computer Society.

[3] Esteban Stafford and José Luis Bosque, "Improving utilization of heterogeneous clusters," *Journal of Supercomputing*, vol. 76, no. 11, pp. 8787–8800, 2020.

[4] Esteban Stafford and José Luis Bosque, "Performance and energy task migration model for heterogeneous clusters," *J. Supercomput.*, vol. 77, no. 9, pp. 10053–10064, 2021.

[5] Vijay V Vazirani, *Approximation algorithms*, Springer Science & Business Media, 2013.

[6] Judea Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley Longman Publishing Co., Inc., USA, 1984.

[7] Andy B. Yoo, Morris A. Jette, and Mark Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, Eds. 2003, pp. 44–60, Springer.

[8] Michael Pinedo, *Scheduling*, vol. 29, Springer, 2012.

[9] Sergei Leonenkov and Sergey Zhumatiy, "Introducing new backfill-based scheduler for slurm resource manager," *Procedia Computer Science*, vol. 66, pp. 661–669, 2015, 4th International Young Scientist Conference on Computational Science.

[10] Wei Tang, Zhiling Lan, Narayan Desai, and Daniel Buettner, "Fault-aware, utility-based job scheduling on blue, gene/p systems," in *IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 1–10.

[11] Danilo Carastan-Santos and Raphael Y De Camargo, "Obtaining dynamic scheduling policies with simulation and machine learning," in *Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–13.

[12] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, 2016, pp. 50–56.

[13] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, SIGCOMM '19, p. 270–288.

[14] A Herrera, M Ibáñez, E Stafford, and JL Bosque, "A simulator for intelligent workload managers in heterogeneous clusters," in *2021 IEEE/ACM 21st Int. Sym. on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 196–205.

[15] D. Zhang, D. Dai, Y. He, Forrest Sheng Bao, and Bing Xie, "RLscheduler: an automated HPC batch job scheduler using reinforcement learning," in *SC20: Int. Conf. for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–15.

[16] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[17] John A Hartigan and Manchek A Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the royal statistical society. series c*, vol. 28, no. 1, pp. 100–108, 1979.

[18] Dror G Feitelson, Dan Tsafrir, and David Krakov, "Experience with using the parallel workloads archive," *Jouranl of Parallel and Distributed Comp.*, vol. 74, no. 10, pp. 2967–2982, 2014.

[19] Uri Lublin and Dror G Feitelson, "The workload on parallel supercomputers: modeling the characteristics of rigid jobs," *Journal of Parallel and Distributed Computing*, vol. 63, no. 11, pp. 1105–1122, 2003.

Quantum Machine Learning en emuladores: PoC con pronósticos de precipitación

Carmen Calvo-Olivera[1], Ángel Manuel Guerrero-Higueras[2], Jesús Lorenzana[3] y Eduardo García-Ortega[4]

Resumen— **Los fenómenos meteorológicos resultan de gran interés por su repercusión en las actividades cotidianas en sectores críticos como la agricultura, el transporte o el comercio. Las previsiones meteorológicas se calculan con modelos numéricos de predicción meteorológica. Sin embargo, a veces se obtienen resultados insatisfactorios debido a la configuración inadecuada del estado inicial. Este trabajo presenta un método para evaluar la incertidumbre de una predicción meteorológica obtenida mediante el modelo *Weather Research and Forecasting*. Se compara un modelo basado en aprendizaje automático clásico, con otro basado en aprendizaje automático cuántico; ambos entrenados con datos reales de la Confederación Hidrográfica del Ebro y resultados de predicción del WRF para calcular la incertidumbre. Los resultados experimentales muestran que la aplicación de técnicas de *Quantum Machine Learning* puede ser una alternativa a las soluciones clásicas.**

Palabras clave— **Machine learning, Quantum, HPC, emuladores cuánticos, predicción de precipitación, pronósticos meteorológicos.**

I. Introducción

La atmósfera tiene un papel fundamental en el desarrollo de la vida en nuestro planeta ya que convierte la Tierra en un planeta habitable. Es un sistema físico no lineal, caótico y dinámico, muy sensible a las condiciones iniciales, lo que implica que la predicción empeora en el tiempo.

La previsión meteorológica consiste en predecir las condiciones de la atmósfera para un lugar y un intervalo de tiempo determinados. Tradicionalmente y de forma simplificada, en esta previsión, la atmósfera se modela como un fluido mediante simulaciones físicas. Se toma una muestra del estado actual de la atmósfera y se calcula el estado futuro resolviendo numéricamente las ecuaciones de la dinámica de fluidos y la termodinámica.

Las previsiones perfectas son imposibles, y toda previsión meteorológica es, hasta cierto punto, incierta [1] y para muchas aplicaciones las previsiones sólo se consideran valiosas si se les puede asignar una estimación de la incertidumbre. En la actualidad, el mejor método para proporcionar una estimación de confianza para las previsiones es producir un conjunto de simulaciones meteorológicas numéricas, lo que es computacionalmente muy costoso [2].

[1]Supercomputación Castilla y León, e-mail: mcalo@unileon.es
[2]Dpto. Ing. Mecánica, Informática y Aeroespacial, Universidad de León, e-mail: amguerrero@unileon.es
[3]Supercomputación Castilla y León, e-mail: jesus.lorenzana@scayle.es
[4]Dpto. Química y Física Aplicadas, Universidad de León, e-mail: eduardo.garcia@unileon.es

Por un lado, el aprendizaje automático, *Machine Learning* (ML) en inglés, representa una alternativa relativamente robusta a las perturbaciones y no requiere una comprensión completa de los procesos físicos que gobiernan la atmósfera, ni tampoco una gran potencia de cálculo –con la excepción de la fase de entrenamiento–. Por tanto, el aprendizaje automático puede representar una alternativa viable para el pronóstico de precipitaciones de predicciones meteorológicas.

Por ejemplo, y centrándonos en la precipitación, la predicción es esencial, y numerosos trabajos abordan este tema, como [3], [4], [5], aplicando técnicas de aprendizaje automático para obtener predicciones de precipitaciones para lugares específicos. Otros trabajos se centran en introducir el ML y demostrar sus ventajas en el campo meteorológico como [6], [7], [8], [9], [2], [10], que proponen el uso de distintos algoritmos para la resolución del cálculo de la incertidumbre inherentemente asociada a una predicción meteorológica.

A menudo, los algoritmos utilizados en estas soluciones, especialmente dentro del campo de la meteorología, tienen una alta demanda de cómputo en la fase de aprendizaje principalmente por el elevado volumen de los datos de entrenamiento. Por tanto, habitualmente se precisa de los servicios de centros de supercomputación para obtener modelos optimizados que luego puedan usarse en entornos reales.

Por otro lado, la tecnología cuántica ha supuesto un cambio de paradigma en el campo de la computación. Apoyándose en las leyes de la mecánica cuántica se resuelven de manera más eficaz algunos problemas complejos que no pueden ser resueltos por los ordenadores tradicionales. Surge el denominado *Quantum Machine Learning* (QML), un campo en evolución. Al igual que el caso de los algoritmos clásicos, en la actualidad los computadores cuánticos reales no son accesibles y aquellos que lo son, presentan varias limitaciones. Esto obliga a llevar a cabo las ejecuciones sobre entornos de emulación. Hoy en día, se dispone de una amplia gama de *frameworks* de emulación cuánticas como Qiskit [11], Pennylane [12], Cirq [13], Qibo [14] o IntelQS [15]. Sin embargo, muchas de las herramientas mencionadas no pueden ser ejecutadas en equipos personales o pequeñas estaciones de trabajo. El mayor inconveniente que presenta es la elevada demanda de memoria y requiere de su instalación y configuración en entornos *High Performance Computing* (HPC). Google, en [16], describe los requerimientos a nivel máqui-

na de los experimentos realizados para alcanzar la supremacía cuántica, la cual representa la capacidad de resolver de forma cuántica algún problema hasta el momento irresoluble por las soluciones clásicas conocidas. El estado de n cúbits requiere de al menos 16 x 2^n bytes de memoria [17]. Son numerosos los trabajos que estudian las alternativas y aplicaciones de versiones cuánticas de los algoritmos más tradicionales como [18], [19] o [20] donde explican las soluciones clásicas y su versión cuántica.

El auge tecnológico en este campo puede ofrecer una mejora frente a las soluciones clásicas ya conocidas. Actualmente, y a nivel nacional, el proyecto QuantumSpain [21] pretende proveer acceso al ordenador cuántico real instalado en el Centro Nacional de Supercomputación en Barcelona (BSC-CNS) y a los simuladores HPC a través de los canales de la Red Española de Supercomputación, lo cuál ha hecho que los centros HPC proporcionen horas de cálculo para investigación y reciban solicitudes para llevar a cabo sus investigaciones.

Dada la incertidumbre de las predicciones meteorológicas, resulta necesario utilizar modelos fiables que nos permitan conocer si podemos considerar valiosa, o no, dicha predicción. El objetivo de este artículo es presentar un modelo basado en aprendizaje automático para evaluar su fiabilidad y comparar los resultados con un modelo basado en aprendizaje automático cuántico en un emulador sobre recursos hardware clásicos.

El resto del artículo se organiza como sigue: la sección II describe el experimento realizado así como los diferentes elementos (hardware y software) utilizados; la sección III enumera los resultados obtenidos; por último, en la sección IV se exponen las conclusiones alcanzadas con este trabajo, además de presentar las líneas de trabajo futuro.

II. Descripción experimental

Para demostrar si con la aplicación de técnicas de ML y QML a la predicción meteorológica obtenemos buenos resultados, se ha realizado un experimento con los siguientes elementos: *dataset*, se ha utilizado un conjunto de datos específicamente creado para el entrenamiento de modelos de inteligencia artificial; *Support Vector Machines*, como algoritmo de evaluación tanto en su versión clásica como en su versión cuántica y el supercomputador Caléndula, el superordenador operado por el Centro de Supercomputación de Castilla y León (SCAYLE), León (España), como plataforma hardware de ejecución.

A continuación se describen cada uno de estos elementos y se presenta la metodología y métricas utilizadas en el experimento.

A. Quantum Machine Learning

El aprendizaje automático cuántico es la integración de algoritmos cuánticos en programas de aprendizaje automático [22]. Dicha combinación puede darse de diversas maneras, obteniendo así cuatro áreas de trabajo 1.

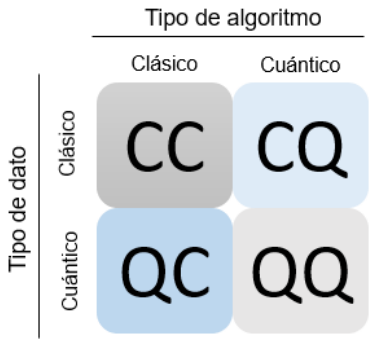


Fig. 1: Áreas de trabajo de QML.

CC Se refiere a procesar datos clásicos con ordenadores clásicos, haciendo uso de algoritmos inspirados en computación cuántica. En otras palabras, hace referencia al *machine learning* clásico que directamente no tiene una base cuántica, si no que toma ideas de la física cuántica.

CQ En este caso, se procesan datos clásicos utilizando algoritmos de *quantum machine learning*. En este área se pretende encontrar soluciones más eficaces para problemas típicamente solucionados con ML pero sobre ordenadores cuánticos.

QC Se trata de un área de investigación en el que se utilizan algoritmos clásicos para tratar datos cuánticos.

QQ Se podría decir que esta sería la aproximación más "pura", ya que se realiza un procesamiento de los datos cuánticos que se procesan haciendo uso del aprendizaje automático cuántico.

La parte cuántica de esta prueba de concepto encaja dentro del área que se ha denominado CQ ya que se lleva a cabo un procesamiento de los datos clásicos de nuestro *dataset* por un algoritmo con base cuántica. En el caso del área CC, su objetivo sería el tratamiento de datos clásicos por algoritmos también clásicos que presentan una base de la física cuántica. Las áreas de QC y QQ son las más difíciles de procesar ya que en el caso de ambas, el procesamiento de los datos de manera puramente cuántico requiere de recursos hasta la fecha limitados y, en el caso de QQ, también sería ideal el uso de un computador cuántico.

Dado el carácter novedoso que acompaña a la computación cuántica y la actualidad del tema, vamos a introducir de forma sencilla algunos conceptos básicos.

Bit cuántico o cúbit en inglés *quantum bit* o *qubit*. Es la unidad básica de la información cuántica. Es análogo al conocido bit clásico. Un cúbit general será de la forma $|u\rangle = a\,|0\rangle + b\,|1\rangle$ donde a y b son dos números complejos, también llamados amplitudes, es decir cuatro números reales. Esto permite que, al contrario que un bit, un cúbit puede existir en una superposición de los estados 0 y 1.

Superposición concepto de la computación cuánti-

ca por el que un cúbit es una combinación lineal de dos estados, $|0\rangle$ y $|1\rangle$, hasta que se mide. Mientras que un bit puede tener el valor 0 o 1, un cúbit puede tener un valor que sea 0, 1 o una superposición cuántica de 0 y 1. A diferencia de las partículas clásicas, si dos estados A y B son estados cuánticos válidos de una partícula cuántica, cualquier combinación lineal de los estados también es un estado cuántico válido: $qState = \alpha A + \beta B$. Esta combinación lineal de estados cuánticos A y B se denomina superposición. Aquí, α y β son las amplitudes de probabilidad de A y B, respectivamente, de modo que $|\alpha|^2 + |\beta|^2 = 1$ [23] [24].

Entrelazamiento Las partículas cuánticas, como los cúbits, pueden estar conectadas o entrelazadas de tal forma que no pueden describirse independientemente unas de otras. Los resultados de sus mediciones están correlacionados aunque estén separados a una distancia infinita. El entrelazamiento es esencial para medir el estado de un cúbit [23] [24].

Interferencia Otro de los conceptos clave. La probabilidad de que un cúbit colapse en una u otra forma está determinada por la interferencia cuántica. Esta interferencia cuántica afecta, por tanto, al estado de un cúbit influyendo de forma directa en la probabilidad de obtener un determinado resultado durante la medición [24].

B. Datos

Una de las partes más importante en la inteligencia artificial son los datos utilizados de entrenamiento para el modelo. En este caso particular, los datos utilizados pertenecen al Valle del Ebro, en el Noreste de España (ver Figura 2) siendo ésta una de las regiones de Europa con mayor número de tormentas convectivas estivales que provocan intensas y fuertes precipitaciones de lluvia y granizo [25]. Las 367 estaciones meteorológicas con las que cuenta el Sistema Automático de Información Hidrológica (SAIH) de la cuenca del Ebro recogen datos de fondo como temperatura y precipitación haciendo posible recabar de datos de verdad terreno.

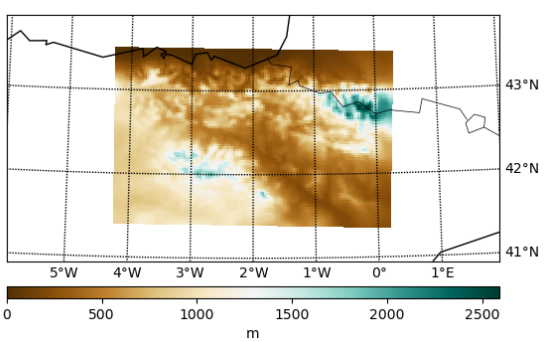


Fig. 2: Demarcación hidrográfica del Valle del río Ebro.

El conjunto de datos (*dataset* en inglés) utilizado para las pruebas se ha generado del postprocesamiento de las previsiones meteorológicas generadas por el modelo WRF, un modelo NWP de mesoescala no hidrostático para la previsión en tiempo real y el análisis atmosférico. Las predicciones WRF corresponden al periodo de tiempo comprendido desde enero de 2008 hasta diciembre de 2018 en la zona de estudio y su postproceso se han realizado en Caléndula (ver II-E). Tras filtrar la mayoría de variables obtenidas en dichas predicciones, ya que no todas tienen que ver con la formación de precipitaciones, obtenemos las variables de temperatura y razón de mezcla a diferentes niveles de presión (500 hPa, 700 hPa y 850 hPa). La lista completa se muestra en la Tabla I siendo estos los predictores. Además, las variables de temperatura y razón de mezcla no son acumulativas, por lo que las obtenemos con un intervalo de 3 horas de 0 a 21 h. Dado que consideramos siete variables en tres niveles de presión ocho veces al día, hacen un total de 168 variables para cada punto de cuadrícula de la zona de estudio representado por su latitud, longitud y altura sobre el mar. El *dataset* completo está disponible online [1]. Cada muestra del conjunto de datos lleva asociada una etiqueta –"rain" or "no rain"– las cuales se han asignado en función de los valores de precìtación obtenidos por el SAIH Ebro.

Dicho conjunto de datos permite ajustar el modelo de predicción que necesitamos para llevar a cabo los experimentos propuestos y comparar los resultados obtenidos por los algoritmos sujetos a estudio y contiene un total de 19.885.973 muestras correspondientes a puntos de la cuadrícula en una fecha específica.

Tabla I: Predictores a 500 hPa, 700 hPa, and 850 hPa cada uno.

Predictor	**Description**
T	Temperature
QVAPOR	Column water vapour content
QCLOUD	Column liquid water content
QRAIN	Column rain
QICE	Column ice water vapour content
QSNOW	Column snow
QGRAUP	Column graupel

C. Algoritmos

Dado que necesitamos predecir una clase –'lluvia" o "no lluvia", los algoritmos de clasificación son más adecuados que los de regresión o agrupación. Con el fin de comparar los resultados entre un algoritmo clásico y su análogo en versión cuántica, se ha decidico escoger los algoritmos SVM (*Support Vector Machines*) y QSVM (*Quantum Support Vector Machines*).

C.1 SVM

Las *Support Vector Machines* son un conjunto de modelos de aprendizaje supervisado que se puede aplicar tanto en procesos de clasificación, como regresión o detección de valores atípicos [26].

De forma resumida, el SVM pretende maximizar la distancia entre dos clases del conjunto de datos

[1] https://doi.org/10.5281/zenodo.6421268

mediante la construcción de un hiperplano óptimo con el fin de obtener una mejor predicción.

En otras palabras, busca separar lo máximo posible las categorías del *dataset*. El SVM también permite clasificar conjuntos de datos que contengan más de dos categorías o clases mediante la generación de un conjunto de hiperplanos. Obtenemos una buena separación mediante un hiperplano que tenga la mayor distancia al punto de datos de entrenamiento más cercano de cualquier categoría (el llamado margen funcional), ya que en general cuanto mayor es el margen, menor es el error de generalización del clasificador [27].

En nuestro caso, la implementación se ha llevado a cabo haciendo uso de scikit-learn y su función SVC() ya definida [28]

C.2 QSVM

El *Quantum Support Vector Machines* es una combinación de computación clásico con la computación cuántica. Esta técnica aprovecha la capacidad del SVC de trabajar con kernels que pueden ser calculados en computación cuántica y beneficiándose de la alta dimensionalidad en el espacio de Hilbert [18] [29] [30].

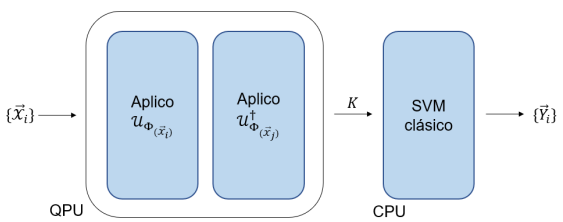


Fig. 3: Flujo esquemático de ejecución de QSVM.

El flujo de ejecución de las QSVM se recoge en la imagen 3. En la parte indicada como QPU, es donde tiene lugar la parte cuántica de la ejecución y donde se genera el kernel cuántico. Del mismo modo, en este punto se mapea para dato clásico de nuestro *dataset* a datos cuánticos para que sepan interpretarlos. El segundo circuito corresponde a la obtención de la matriz del kernel correlacionado con las características del conjunto de datos. Finalmente, y de forma clásica, se aplica el kernel cuántico al algoritmo clásico.

Para este caso, se ha hecho uso de la función de Qiskit que amplía el clasificador de scikit-learn sklearn.svm.SVC e introduce un parámetro adicional para añadir el kernel cuántico [31].

D. Software

Actualmente, existen a nuestra disposición diferentes plataformas que permiten llevar a cabo cálculos cuánticos como las plataformas de IBM o Google. En España, existe el proyecto QuantumSpain [21] cuyo principal objetivo es impulsar una infraestructura competitiva y completa de computación cuántica en España dotando al ecosistema cuántico nacional de las herramientas necesarias para desarrollar un sólido tejido científico y tecnológico en torno a la computación cuántica y sus aplicaciones en Inteligencia Artificial. SCAYLE, junto BSC-CNS (Centro de Supercomputación de Barcelona) y CESGA (Centro de Supercomputación de Galicia), albergará uno de los emuladores cuánticos que conformarán la infraestructura de emuladores del proyecto con la finalidad de simular el comportamiento de un ordenador cuántico mediante el despliegue de un software de emulación y beneficiarse de los recursos hardware clásicos de computación de altas prestaciones (HPC). Teniendo en cuenta que el objetivo de este trabajo es comprobar la viabilidad del uso de algoritmos cuánticos y compararlos con las soluciones clásicas en el ámbito de las predicciones meteorológicas, se decidió llevar a cabo todas las pruebas en el mismo entorno.

Los software utilizado para el desarrollo de los experimentos en el clúster han sido principalmente los siguientes:

- *Jupyter Notebooks [32]* es una aplicación web que permite crear y ejecutar código de forma dinámica y poder ir viendo los resultados. En el clúster de SCAYLE, existe la opción de trabajar con esta herramienta una vez reservados los recursos a utilizar y resulta útil a la hora de desarrollar pequeñas pruebas.
- *Scikit learn [33]* es una librería de código abierto basada en Python enfocada al aprendizaje automático. Incluye tanto algoritmos de clasificación como de regresión lo que la convierte en una herramienta versátil.
- *Qiskit [11]* es un software de código abierto para trabajar con ordenadores cuánticos a nivel de circuitos, pulsos y algoritmos que cuenta con varias API de aplicaciones específicas. Su objetivo es construir una pila de software (*software stack*) que facilite el uso de ordenadores cuánticos permitiendo diseñar fácilmente experimentos y aplicaciones y ejecutarlos en ordenadores cuánticos reales y/o simuladores clásicos.

E. Hardware

Todas las ejecuciones se han llevado a cabo en Caléndula, el clúster HPC del Centro de Supercomputación de Castilla y León (SCAYLE). Caléndula es uno de los 17 superordenadores que conforman la Red Española de Supercomputación (RES) y tiene una potencia de cálculo de 397 TFlops –en el momento de realizar este trabajo– distribuidos en 345 servidores interconectados por una red Infiniband de baja latencia que gestiona una comunicación bidireccional que permite compartir datos de entrada y salida.

Más concretamente, se han utilizado 8 equipos que, cada uno de ellos, cuenta con 2 procesadores Intel Xeon Platinum 8358 (codename Ice Lake), con 32 cores cada procesador (64 cores totales por equipo) y una frecuencia de funcionamiento de 2.6 GHz. Una de las características más destacables, junto con su GPU NVidia Tesla A100, es 1 TB de memoria RAM. Dada la demanda de memoria en cálculos cuánticos, era imprescindible contar con máquinas que tuvie-

ran lo máximo posible de memoria. También tienen entre ellos una conexión Infiniband HDR 100 Gbps. Estos equipos cuentan con Centos v7.9 como sistema operativo.

F. Metodologías y métricas

Con el fin de validar la premisa presentada en la introducción, se ha realizado una prueba de concepto evaluando, en primer lugar, el rendimiento de los modelos teniendo en cuenta su puntuación de exactitud, como se muestra en la ecuación 1), donde T_P es la tasa de verdaderos positivos, T_N es la tasa de verdaderos negativos, F_P es la tasa de falsos positivos y F_N es la tasa de falsos negativos.

$$accuracy = \frac{T_P + T_N}{T_P + F_P + T_N + F_N} \tag{1}$$

También se han considerando los siguientes indicadores de rendimiento obtenidos a través de la matriz de confusión: Precisión ($\mathcal{P}$), Recall ($\mathcal{R}$), y F_1-score (F_1). Las ecuaciones $\mathcal{P}$, $\mathcal{R}$, y F_1 [34] se calculan como se muestra en las ecuaciones 2, 3, y 4. La puntuación $\mathcal{P}$ muestra la relación entre el número de predicciones correctas positivas y el número total de predicciones positivas (tanto verdaderas como falsas) mostrando la capacidad del clasificador de no etiquetar como positiva una muestra negativa. La puntuación $\mathcal{R}$ muestra la tasa de casos positivos identificados correctamente por el algoritmo. La puntuación F_1 está relacionada tanto con $\mathcal{P}$ como con $\mathcal{R}$, ya que es su media armónica [35].

$$\mathcal{P} = \frac{\mathcal{T}_\mathcal{P}}{\mathcal{T}_\mathcal{P} + \mathcal{F}_\mathcal{P}} \tag{2}$$

$$\mathcal{R} = \frac{\mathcal{T}_\mathcal{P}}{\mathcal{T}_\mathcal{P} + \mathcal{F}_\mathcal{N}} \tag{3}$$

$$F_1 = 2\frac{\mathcal{P} \times \mathcal{R}}{\mathcal{P} + \mathcal{R}} \tag{4}$$

Respecto al proceso de entrenamiento, ha sido necesario reducir el número de predictores ya que el *backend* utilizado en la parte cuántica del experimento tiene, en el momento de realización de las pruebas, una limitación en cuanto al número de cúbits simulados siendo el máximo de 24. En Qiskit, un *backend* representa un simulador o un ordenador cuántico real y es responsable de ejecutar los circuitos cuánticos y devolver los resultados y para el entrenamiento de nuestro modelo cuántico hemos utilizado el *backend* de Qiskit denominado "statevector-simulator" ya que permite una mejor representación de los resultados. En este caso, cada cúbit representa una de las características del conjunto de datos. Teniendo en cuenta que el *dataset* utilizado cuenta con 168 variables en total, fue necesario seleccionar las características más relevantes para el entrenamiento de los modelos y poder comparar. De este modo, y haciendo uso de la función SelectKBest() de scikit-learn se obtienen los resultados recogidos en la gráfica 4.

Así mismo, también hemos reducido el tamaño de muestras con las que entrenar los modelos, siendo finalmente datos de una semana correspondiente al mes de enero de 2009 durante la que se recogieron precipitaciones.

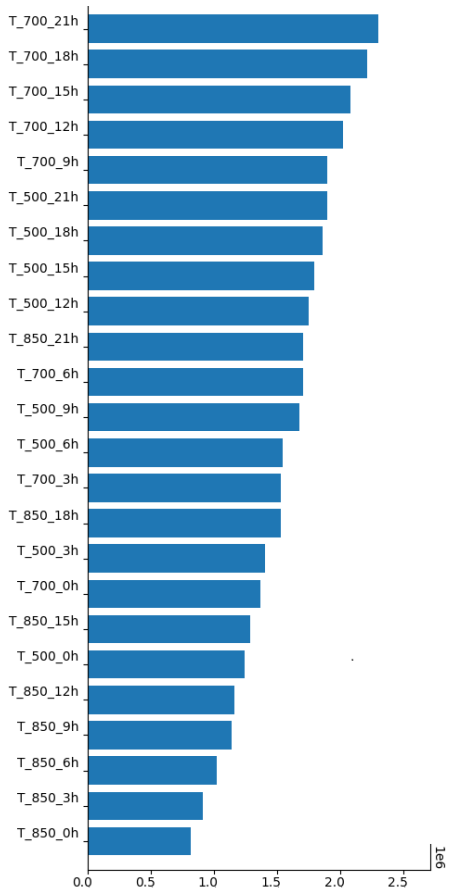


Fig. 4: Ranking con las 24 características más relevantes. El eje x corresponde al valor devuelto por la función SelectKBest(), *scores_*, el cual proporciona la puntuación de cada característica de acuerdo a su relevancia.

El detalle de los resultados se presenta en el siguiente apartado.

III. Resultados

La tabla II muestra los valores de precisión para el algorítmo SVM y para el QSVM para el conjunto de datos reducido.

Tabla II: Resultados de la exactitud para cada algoritmo.

Classifier	**Accuracy**
SVM	0.941
QSVM	0.823

La tabla III muestra los valores de precisión, recall y F_1 para el algoritmo SVM y para el QSVM para una semana de previsiones meteorológicas.

Tabla III: Índices de precisión ($\mathcal{P}$), recall ($\mathcal{R}$), y F_1.

Classifier	**Class**	$\mathcal{P}$	$\mathcal{R}$	F_1
SVM	No rain	0.917	0.960	0.943
	Rain	0.958	0.923	0.941
	Average	0.940	0.941	0.940
QSVM	No rain	0.837	0.917	0.879
	Rain	0.773	0.583	0.661
	Average	0.807	0.823	0.805

Para obtener un método que permita evaluar la veracidad de una predicción meteorológica obtenida por el modelo WRF, necesitamos datos de verdad sobre el terreno no disponibles en tiempo real. Nuestro objetivo es, por tanto, evaluar la viabilidad del uso de técnicas cuánticas emuladas sobre recursos tradicionales de hardware como disponibles en centros de supercomputación. Como necesitamos que este modelo sea fiable, hemos calculado $\mathcal{P}$, $\mathcal{R}$, F_1 para evaluar los modelos y verificar si podemos mantener o mejorar el rendimiento con soluciones cuánticas aplicadas a casos reales de prediciones meteorológicas.

Los resultados de la tabla II muestran las puntuaciones de precisión para cada modelo. Vemos que el SVM supera el 94 % mientras que el QSVM no alcanza el 85 %. Este resultado es significativo pero debemos tener en cuenta que el dataset utilizado ha sido una muestra del original correspondiente a una semana de precipitaciones. Vemos que las soluciones bien conocidas como el algoritmo SVM obtiene mejores resultados pero que la alternativa cuántica, en el futuro, puede resultar interesante a la hora de trabajar con este tipo de datos.

En la tabla III vemos los resultados para la precission, recall y F_1. En este caso se mantienen las diferencias que hemos visto previamente y la solución clásica obtiene mejores resultados para los tres indicadores.

IV. Conclusiones

Determinar la exactitud de una predicción meteorológica es esencial para las actividades cotidianas. El avance en aprendizaje automático ha permitido utilizar modelos que permitan facilitar esta tarea y el avance en los últimos años de la computación cuántica, y más concretamente del *Quantum Machine Learning*, llevan a buscar alternativas aún mas eficientes a estos modelos. Este artículo describe los resultados experimentales obtenidos al comparar un modelo de aprendizaje automático clásico con su análogo cuántico. Para ello se ha utilizado un conjunto de datos real y generado específicamente para el propósito de entrenar modelos lo más precisos posibles.

Para demostrar que las técnicas de aprendizaje automático, tanto clásicas como cuánticas, proporcionan un modelo que ayude a determinar el grado de precisión de una predicción, se han seleccionado previamente los algoritmos a comparar y se ha generado, en primer lugar, un modelo clásico con el algoritmo SVM y posteriormente, y siempre con el mismo conjunto de datos, se ha generado y evaluado el modelo cuántico basado en QSVM.

Los experimentos han demostrado que los algoritmos de aprendizaje automático cuánticos proporcionan modelos fiables para determinar la exactitud de las previsiones meteorológicas de forma eficaz aunque sin alcanzar los resultados de los algoritmos clásicos, que es el principal objetivo de este trabajo. Otra de las conclusiones a las que llegamos es la limitación actual que presentan los emuladores/computadores cuánticos inherentes a las limitaciones físicas y de hardware que tienen. Igualmente, el análisis propuesto, identifica que se siguen obteniendo mejores resultados para los modelos clásicos pero que sus versiones cuánticas pueden ser una buena alternativa conforme la tecnología vaya avanzando.

El trabajo expuesto aquí solamente es una primera aproximación del uso de Quantum Machine Learning en el campo de las predicciones meteorológicas en entornos HPC. Este estudio abre la puerta a una serie de trabajos futuros, como por ejemplo, la evaluación sistemática para otros algoritmos que permitan seleccionar el que mejor resultado obtenga para un problema determinado.

Agradecimientos

La investigación descrita en este artículo ha sido parcialmente financiada por el Centro de Supercomputación de Castilla y León (SCAYLE).

Los autores desean expresar su agradecimiento al SAIH Ebro por facilitar los datos de temperatura y precipitación de su red de estaciones meteorológicas.

Referencias

[1] Edward N Lorenz, "Deterministic nonperiodic flow," *Journal of atmospheric sciences*, vol. 20, no. 2, pp. 130–141, 1963.

[2] Sebastian Scher and Gabriele Messori, "Predicting weather forecast uncertainty with machine learning," *Quarterly Journal of the Royal Meteorological Society*, vol. 144, no. 717, pp. 2830–2841, 2018.

[3] Mahdie Afshari Nia, Fatemeh Panahi, and Mohammad Ehteram, "Convolutional neural network-ann-e (tanh): A new deep learning model for predicting rainfall," *Water Resources Management*, pp. 1–26, 2023.

[4] Romulus Costache, Alireza Arabameri, Iulia Costache, Anca Crăciun, and Binh Thai Pham, "New machine learning ensemble for flood susceptibility estimation," *Water Resources Management*, vol. 36, no. 12, pp. 4765–4783, 2022.

[5] Ali Mokhtar, Nadhir Al-Ansari, Wessam El-Ssawy, Renata Graf, Pouya Aghelpour, Hongming He, Salma M Hafez, and Mohamed Abuarab, "Prediction of irrigation water requirements for green beans-based machine learning algorithm models in arid region," *Water Resources Management*, pp. 1–24, 2023.

[6] Enrico Camporeale, "The challenge of machine learning in space weather: Nowcasting and forecasting," *Space Weather*, vol. 17, no. 8, pp. 1166–1207, 2019.

[7] Azam Moosavi, Vishwas Rao, and Adrian Sandu, "Machine learning based algorithms for uncertainty quantification in numerical weather prediction models," *Journal of Computational Science*, vol. 50, pp. 101295, 2021.

[8] MG Schultz, Clara Betancourt, Bing Gong, Felix Kleinert, Michael Langguth, LH Leufen, Amirpasha Mozaffari, and Scarlet Stadtler, "Can deep learning beat numerical weather prediction?," *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2194, pp. 20200097, 2021.

[9] Markus Reichstein, Gustau Camps-Valls, Bjorn Stevens, Martin Jung, Joachim Denzler, Nuno Carvalhais, et al., "Deep learning and process understanding for data-

driven earth system science," *Nature*, vol. 566, no. 7743, pp. 195–204, 2019.

[10] Peter Grönquist, Tal Ben-Nun, Nikoli Dryden, Peter Dueben, Luca Lavarini, Shigang Li, and Torsten Hoefler, "Predicting weather uncertainty with deep convnets," *arXiv preprint arXiv:1911.00630*, 2019.

[11] Robert Wille, Rod Van Meter, and Yehuda Naveh, "Ibm's qiskit tool chain: Working with and developing for real quantum computers," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1234–1240.

[12] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M Sohaib Alam, Guillermo Alonso-Linaje, B AkashNarayanan, Ali Asadi, et al., "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, 2018.

[13] Paramita Basak Upama, Md Jobair Hossain Faruk, Mohammad Nazim, Mohammad Masum, Hossain Shahriar, Gias Uddin, Shabir Barzanjeh, Sheikh Iqbal Ahamed, and Akond Rahman, "Evolution of quantum computing: A systematic survey on the use of quantum computing tools," in *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2022, pp. 520–529.

[14] Stavros Efthymiou, Sergi Ramos-Calderer, Carlos Bravo-Prieto, Adrián Pérez-Salinas, Diego García-Martín, Artur Garcia-Saez, José Ignacio Latorre, and Stefano Carrazza, "Qibo: a framework for quantum simulation with hardware acceleration," *Quantum Science and Technology*, vol. 7, no. 1, pp. 015018, 2021.

[15] Gian Giacomo Guerreschi, Justin Hogaboam, Fabio Baruffa, and Nicolas PD Sawaya, "Intel quantum simulator: A cloud-ready high-performance simulator of quantum circuits," *Quantum Science and Technology*, vol. 5, no. 3, pp. 034007, 2020.

[16] Igor L Markov, Aneeqa Fatima, Sergei V Isakov, and Sergio Boixo, "Quantum supremacy is both closer and farther than it appears," *arXiv preprint arXiv:1807.10749*, 2018.

[17] Jun Doi, Hitomi Takahashi, Rudy Raymond, Takashi Imamichi, and Hiroshi Horii, "Quantum computing simulator on a heterogenous hpc system," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, 2019, pp. 85–93.

[18] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[19] M Cerezo, Guillaume Verdon, Hsin-Yuan Huang, Lukasz Cincio, and Patrick J Coles, "Challenges and opportunities in quantum machine learning," *Nature Computational Science*, vol. 2, no. 9, pp. 567–576, 2022.

[20] Vedran Dunjko and Peter Wittek, "A non-review of quantum machine learning: trends and explorations," *Quantum Views*, vol. 4, pp. 32, 2020.

[26] William S Noble, "What is a support vector machine?," *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[21] Quantum Spain, "Quantum spain. creación de un ecosistema nacional de computación cuántica para la ia," 2023.

[22] Anna Dawid, Julian Arnold, Borja Requena, Alexander Gresch, Marcin Płodzień, Kaelan Donatella, Kim Nicoli, Paolo Stornati, Rouven Koch, Miriam Büttner, Robert Okuła, Gorka Muñoz-Gil, Rodrigo A. Vargas-Hernández, Alba Cervera-Lierta, Juan Carrasquilla, Vedran Dunjko, Marylou Gabrié, Patrick Huembeli, Evert van Nieuwenburg, Filippo Vicentini, Lei Wang, Sebastian J. Wetzel, Giuseppe Carleo, Eliška Greplová, Roman Krems, Florian Marquardt, Michał Tomza, Maciej Lewenstein, and Alexandre Dauphin, "Modern applications of machine learning in quantum sciences," 2022.

[23] Tony Hey, "Quantum computing: an introduction," *Computing & Control Engineering Journal*, vol. 10, no. 3, pp. 105–112, 1999.

[24] Microsoft, "Azure quantum," 2023.

[25] E García-Ortega, L Hermida, R Hierro, A Merino, E Gascón, S Fernández-González, JL Sánchez, and L López, "Anomalies, trends and variability in atmospheric fields related to hailstorms in north-eastern spain," *International journal of climatology*, vol. 34, no. 11, pp. 3251–3263, 2014.

[27] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman, *The elements of statistical learning: data mining, inference, and prediction*, vol. 2, Springer, 2009.

[28] Scikit-Learn, "sklearn.svm.svc," 2023.

[29] Maria Schuld and Nathan Killoran, "Quantum machine learning in feature hilbert spaces," *Physical review letters*, vol. 122, no. 4, pp. 040504, 2019.

[30] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, no. 7747, pp. 209–212, 2019.

[31] Qiskit, "algorithms.qsvc," 2023.

[32] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al., *Jupyter Notebooks-a publishing format for reproducible computational workflows.*, vol. 2016, 2016.

[33] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al., "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[34] Marina Sokolova and Guy Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[35] Mohammad Hossin and Md Nasir Sulaiman, "A review on evaluation metrics for data classification evaluations," *International journal of data mining & knowledge management process*, vol. 5, no. 2, pp. 1, 2015.

Applying Machine Learning for characterizing social networks Agent-based models

Haoyuan Li[1], Carla Viñas Templado[1], Eduardo César Galobardes[1], Anna Sikora[1]

Abstract— **Nowadays, social media network plays an important role in our life. It is not only a platform to express your opinion, but also for information dissemination and advertising. As it becomes more and more commercial, it becomes necessary to study the users behaviours and preferences.**

As we know, there are billions of users among all the social platforms and the social network keeps updating every millisecond. Besides, the network among all the users and platforms are complex. We need a model that can involve all the features of social media network that can perform well with such a huge and complex computing requirement.

Agent-based modeling is widely employed in previous works to study social network community. It can be used to explore how microscopic interaction of agents which represent the users can influence macroscopic phenomena. We can specify each agent's individual behavioral rules, to describe the circumstances in which the individuals reside and then to execute the rules to determine possible system-level evolution [1]. Moreover, significantly more powerful and scaled up data processing, networking and data storage capabilities are required to implement the advantages of agent-based models. Therefore, it could be a good option to use High Performance Computing (HPC) which has been widely used for computing complex, voluminous or iteration intensive calculations and analysis. However, there are significant challenges for validating the models and implementing efficient simulators.

In this work, we are going to use Machine Learning methods to characterize users by their attributes and post contents, and to develop a general user model for ABM systems simulation in social networks. We will mainly focus on characterizing the attributes of users of Twitter and analyzing their topics of tweets.

keywords— **social media network, Agent-based model, user attribute and behaviour analysis, machine learning**

I. Introduction

THE purpose of early social media networks was to provide an online channel for those who could not connect in real life. As the social networks continue to grow and explore ways to monetize, these platforms have begun to enrich their products. Nowadays they have become a very influential aspect of modern society, affecting how people communicate, interact, and form opinions. As a result, understanding the dynamics of social media networks has become an important area of research.

Agent-based modeling (ABM) allows researchers to simulate the behavior of individuals within a network and investigate how their interactions give rise to emergent properties of the network as a whole. ABM is a computational modeling technique that involves creating virtual agents that can represent individual entities within a social system, such as people or organizations. These agents can have complex behaviors and interact with each other according to predefined rules, allowing us to observe how the collective behavior of the system emerges from the interactions of its individual components. There are many related works that have been done aimed on solving different problems by using agent-based modeling, such as [2] and [3]. Each of these works usually propose different models for social network users which makes it difficult to validate, compare and extend their results.

The long term goal of our research is to define a general agent-based model for social media networks and investigate how they can be used to better understand the dynamics of these networks. Besides, we want to implement this agent-based model in a HPC platform that could realize the full potential of ABMS. Our general objective of creating simulators that will be executed in HPC platforms is because of the large scale of agents and their potential complexity, but before implementing a parallel social network simulator we have to define the model. Specifically, we aim to explore how agent-based modeling can be used to simulate the users behaviors and interactions, the spread of information and to analyze how these dynamics are influenced. Through our research, we hope to provide insights into the underlying mechanisms that govern social media networks.

The first step for reaching this goal consist in characterizing social network users. Machine learning techniques are especially appropriate for classifying subjects according to their features. Consequently, the goal of this work is to apply machine learning methods in characterizing social media users by their different attributes or posting topics.

After that, we will be able to create an agent-based model to study how the social media users behave and interact. This model would be used to analyze any kinds of users from our selected social network platform. Afterwards, when we get an accurate model and efficient simulation, we can choose a topic of the aspects we want to study, by adding different algorithms to make it work on specific problems. For example, we can use it to study emotion contagion in online social media, to detect fake news, to analyze users preferences and then advertise more accurate, or to analyze how the memes propagate among users of different ages etc.

This paper is structured as follows. Section II discusses the related work and current approaches. In Section III, different methods for characterizing users

[1]Universitat Autònoma de Barcelona, Cerdanyola, Spain, e-mail: {Haoyuan.Li, Carla.Vinas}@autonoma.cat, {Eduardo.Cesar, Anna.Sikora}@uab.cat

will are discussed and compared. Section IV shows a case study applying machine learning methods for characterizing users in Twitter. Finally, Section V presents the summary of this work.

II. Related Work

In [4] Tayfun Tuna et al. made a summary of analysis of user characterization of online social networks. This paper focused on the problem of characterizing users in online social networks, which involves identifying users with similar interests, behaviors, and social interactions. They provided an overview of the general ways of user analysis which includes attribute determination, behavior analysis, mental models, user categorization and entity resolution. Their work presented the studies from different perspectives such as finding hidden information from posted user data, determining features for characterization and user behavior analysis in an organized and concise manner. This paper provided a guide for us of exploring the attributes of social network users and methods of doing so: they discussed several applications of user characterization that we can take inspiration and explore new applications of user characterization in different domains. They presented several different algorithms and compared the results that can guide us to choose the appropriate one to apply in different situations.

In[3], Anna Gausen et al. used agent-based modelling to compare the impact of four different newsfeed curation algorithms on the spread of misinformation and polarization based on real Twitter data. The agents will sample a probability to determine if they are online at each timestep. If the agents are online, they will sample probabilities to determine their behaviors. At the end of each timestep, the agent reporter records a set of metrics which are tracked through the duration of the simulation. In this work they set each agent with the following attributes: followers, followees, time of most recent post, retweet number, beliefs, whether their account is verified and number of tweets they have posted about the story of interest. We will include more user's attributes to make the model more general for different platforms and that could provide better study of users.

In [5], Usman et al. created a model that is called Twitter Behavior Agent-Based Model (TBAM) to simulate Twitter pattern and behavior. They used Netlogo to create their agent-based model and had four types of agents: turtles, patches, links and observer. These four types of agents are connected by their different rules, for example they set the links as agents that connect two turtles, the observers observe the agents and their interactions. However, in real social media network the connections are not so simplified. Therefore, in our work we hope to create a general model that can represent the complex social network and to set all the agents more realistically to simulate the behaviors and interactions of the users.

In[6] Marcelo Maia et al. sought to systematically identify user behaviors in online social systems. They crawled data from YouTube and used K-means algorithm to group users that share similar behavioral pattern. Their methodology used in this work is not restricted to YouTube, it is applicable to online social networking systems in general. We can learn from their ideas of clustering users and setting feature vectors. But their work focused on investigating typical user behaviors instead of the attributes that may influence their behaviors, as well as the networks emerged from users interactions. Our work will involve these aspects and we could also exploit the user behaviors to define different classes to develop more accurate models

Generally, each of these works is focusing on a particular aspect of social network modeling, which makes it difficult to compare, validate and generalize their results.

III. Machine Learning Methods

To achieve analysis of users' attributes and behaviors, as well as their classification, we will apply machine learning methods that are commonly used in analyzing social media users. Many social media platforms provide an application programming interface that allows for accessing to user-generated text, images and videos, as well as to accompanying metadata, such as where and when the content was uploaded, and connections between users. Machine learning can be trained to filter these contents to identify relevant information to classify the sentiment and preferences of social media users [7]. Five methods are widely applied in analyzing users which are: decision trees, random forests, support vector machines, neural networks and K-means.

1. Decision trees

The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called "root" that has no incoming edges. All other nodes have exactly one incoming edge. In a decision tree, each internal node splits the instance space into two or more sub-spaces according to a certain discrete function of the input attributes values[8]. Decision trees are sequential models, which logically combine a sequence of simple tests; each test compares a numeric attribute against a threshold value or a nominal attribute against a set of possible values[9].

Decision trees can be used for both classification and regression problems. In our work, decision trees can be used by creating a model that represents the decision-making process for classifying users into different groups based on their attributes. We will decide the attributes we want to classify as the target variables, then the decision trees will be trained to classify them based on our input variables. The input variables are those relevant for classifying characteristics. After this, we need to use the input variables and target variables to train the decision tree algorithm. The algorithm will recursively partition the data into smaller and more homogeneous groups based on the input variables, and then assign the

target variable based on the majority class in each group.

Even though decision trees are very simple and fast, they can deal with noisy data, have significantly more complex representation for some concepts due to replication problem and have a problem of overfitting[10].

2. Random forests

The method of random forests is a generalization of the decision tree method, in which the data space is recursively partitioned (usually a binary split) according to the value of one of the predictor variables, such that the observations within a partition become more and more homogeneous. Random forests builds multiple decision trees using random samples of observations for each tree and at each split point random samples of predictors. The resulting forest of those trees provides fitted values, which are more accurate than those of any single tree. Moreover, the random forests method comes with a built-in protection against over fitting by using part of the data that each tree in the forest has not seen to calculate its goodness-of-fit[11].

The main steps of random forests to do the classification are almost the same that with decision trees, but this algorithm creates multiple decision trees by randomly selecting subsets of the input variables, and then aggregates the results of the individual trees to make a classification. Compared with decision trees, random forests have a improved accuracy, robustness to noise and over fitting, and better handling of high-dimensional data with many input variables. However, they can also be more computationally expensive and harder to interpret than a single decision tree[12].

3. Support vector machine

Support vector machine (SVM) is a machine learning method originally introduced for the problem of classification and later generalized to various other situations. It is based on principles of statistical learning theory and convex optimization, and is currently used in various domains of application. The aim of support vector classification is to find the optimal hyperplane that separates the data into different classes. The hyperplane is chosen to maximize the margin between the classes, and data points that lie on the margins are called support vectors[13]. SVM works well when data is non-linearly distributed in the classification.

Like the previous two methods, SVM also needs to define the target variables and select input variables. Then, we can use the target variables and input variables to train this model to make it find a hyperplane that maximally separates the different classes in the data while also minimizing the margin between the hyperplane and the closest data points. After that, we can evaluate it by assessing the accuracy and performance of the classification.

SVM is effective in high dimensional spaces. It is effective in cases where number of dimensions is greater than the number of samples[14]. But if the dataset is large, SVM does not perform well. It can also be sensitive to the choice of hyperparameters and kernel functions.

4. Neural networks

Neural networks are a type of machine learning algorithm inspired by the structure and function of biological neural networks. Neural networks can be used for both supervised and unsupervised learning problems. They work by creating a network of interconnected nodes (neurons) that process input data and produce output data. In general, the network consists of processing neurons and information flow channels between the neurons, usually called "interconnects". Each processing neuron calculates the weighted sum of all interconnected signals from the previous layer plus a bias term and then generates an output through its activation transfer function. The adjustment of the neural network function to experimental data (learning process or training) is based on a non-linear regression procedure. Training is done by assigning random weights to each neuron, evaluating the output of the network and calculating the error between the output of the network and the known results by means of an error or objective function. If the error is large, the weights are adjusted and the process goes back to evaluate the output of the network. This cycle is repeated till the error is small or a stop criterion is satisfied[15]. Neural networks are particularly useful for tasks where the data is highly non-linear and complex, and where there may be interactions between different input variables.

The first two steps of creating the neural network model are still define the target variables and select input variables. But the next step is to design the architecture of the neural network, which means to determine the number of layers and nodes. The neural network can set connections between the input variables and the target variables by adjusting the weights and biases of the nodes in the network. It is also necessary to evaluate it by assessing the accuracy and performance of the classification.

Neural networks can offer several advantages over other machine learning algorithms such as: it can implicitly detect complex nonlinear relationships between independent and dependent variable, requires less formal statistical training to develop. However, it requires greater computational resources, it can be difficult to interpret and may suffer from overfitting if not carefully designed and trained[16].

5. K-means

The K-means is a unsupervised clustering algorithm described in detail by Hartigan [17]. The aim of the K-means algorithm is to divide M points in N dimensions into K clusters so that the within-cluster sum of squares is minimized. K-means works by iteratively assigning data points to clusters based on their proximity to the cluster centroid, and then updating the centroid based on the newly assigned data points. The algorithm continues until the cluster assignments and centroids converge.

By using K-means, we need to prepare the data

and select the relevant attributes that will be used for clustering. Then the most important step is to choose the K value. The K value represents the number of clusters that we want to form. After choosing a proper K value we can do K-means clustering that will group similar users into clusters based on their attributes. In the end, we can interpret the clusters and assign labels to each group based on the characteristics of the users in the cluster.

K-means is simple and very fast, so in many practical applications, the method is proved to be a very effective way that can produce good clustering results. This method reduces the impact of isolated points and the "noise", so it enhances the efficiency of clustering. But compared with other methods, K-means requires that the number of clusters is specified in advance, which can be difficult to determine[18].

IV. Case Study

In this work, we want to apply machine learning methods in characterizing social media users with different attributes or posting different topics. We decided to focus on characterizing the attributes of users of Twitter and analyzing their topics of tweets. In this section there are two parts: the first part we talk about the steps to characterize users by their attributes, the second part is the analysis of the topics of their tweets.

1. Study of users' attributes

A. Generating data

In this work, we generate our own dataset by accessing the Twitter API (Tweepy[19]) to obtain all the attributes it offered. To carry out the acquisition of the data, we have used different starting seeds along with different API access credentials. This is due to the fact that the Tweepy API has several restrictions, one of them is the limit of calls. Twitter limits the number of calls that can be made to its API in a given period of time. If a developer exceeds the call limits, they are prevented from making API queries for a specified period of time. Therefore, different access credentials are used to obtain different seeded environments. In this way, graphs have been formed that refer to the closest circles of the seed nodes. The execution in each case has ended when all nodes have been fully explored (followers and followees). Once captured, in order to have all the data in a single graph, the compose function of the NetworkX[20] library has been used.

B. Deciding attributes

The resulting dataset contained 38004 samples with 13 different attributes that are: created_at, friends_count, followers_count, listed_count, location, favorites_count, statuses_count, verified, protected, geo_enabled, default_profile, default_profile_image, tweets. The description of these attributes is presented in Table I. Because of some users' information is incomplete there is necessary to clean the dataset. Finally, we get a dataset that contains 37991 samples. It is also necessary for all attributes to be numeric. This is because the Euclidean distance will be used later to measure the similarity between data points, and this distance can only be computed between points with numerical values. So all the Boolean variables have been modified to values 0 (false) or 1 (true). Besides, a pre-processing has been performed on users' tweets by translating the text into English, removing emojis, urls and mentions to standardize the input and reduce the noise.

Tabla I: Description of the attributes that make up the database.

Attribute	Description	Data type
created_at	Date the account was created (only consider the year of creation)	Integer
friends_count	Number of friends for the account	Integer
followers_count	Number of followers for the account	Integer
listed_count	Number of public lists of which the user is a member	Integer
favorites_count	Total number of tweets written by users	Integer
statuses_count	Total number of tweets posted by users	Integer
location	Location entered by the user	Boolean
verified	If the account is authenticated	Boolean
protected	If the account is protected	Boolean
geo_enabled	If the account is geographically enabled	Boolean
default_profile	If the user has a default account	Boolean
default_profile_image	If the user has an account image by default	Boolean
tweets	Last 5 tweets (if user has)	String List

C. Training a ML method to classify the users

Once a proper dataset and set of attributes has been produced, we can start doing the classification. In this work, we choose K-means method to characterize users' attributes. K-means is a popular unsupervised learning algorithm used for clustering which focuses on finding hidden patterns or structures in the data without the help of labels or previous answers. In this way, it is intended to look for patterns that allow users to be separated into different groups. The K-means clustering method is a non-hierarchical method for clustering objects that partitions the dataset into K distinct, non-overlapping clusters, meaning that no observation can belong to more than one cluster. The number of required clusters or subgroups has to be set at the start.

The number of clusters is defined by using the elbow method. This technique is used in clustering to determine the optimal number of clusters to use in a dataset. The elbow method calculates the squared difference of different K values of clusters. As the K value increases, the average distortion degree becomes smaller. The number of samples contained in each category decreases, and the samples are closer to the center of gravity. As the K value increases, the position where the improvement effect of the distortion degree decreases the most is the K value corresponding to the elbow[21]. The elbow method involves plotting the sum of intra-cluster squared distances (sum of squares within clusters or SSD) as a function of the number of clusters, and choosing the

number of clusters at the point where the decrease of variance slows down significantly, which often forms an elbow-shaped curve.

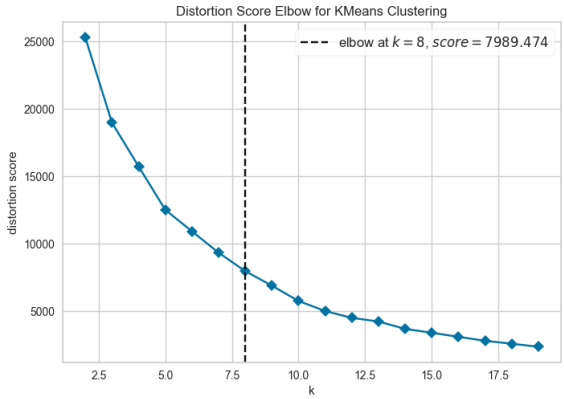


Fig. 1: Elbow method graph for K-means algorithm

From Fig 1, we could note that with the elbow method we obtain K=8. Besides, there are other parameters needed to be defined at the beginning of the training:

(1) The initialization method

This is the method used to select the initial centroids of each cluster.

(2) The maximum number of iterations

This is the maximum number of times the K-Means algorithm will be performed before convergence is declared. If the clusters do not converge after the maximum number of iterations, the algorithm stops.

(3) The convergence tolerance

This is the value of the difference of the inertia between iterations that indicates that convergence has been achieved. If the difference is less than the tolerance, the algorithm is considered to have converged and stops.

Once the number of groups going to be created was calculated, we could get the parameters and the K-means method would be trained with K=8, 5000 iterations, 0.000001 convergence tolerance and k-means++[1] as a method of cluster initialization.

The algorithm yielded the following distribution of users in each cluster.

Tabla II: Distribution of users per cluster obtained by the K-means algorithm.

cluster	distribution
0	12.02%
1	23.55%
2	12.84%
3	20.89%
4	10.87%
5	5.52%
6	7.01%
7	7.28%

From Fig. 2 and Table. II we could see cluster 1

[1]The k-means++ algorithm is used to initialize the center of mass of the cluster. Select a point away from the first center of mass to ensure that the centers of mass are correctly distributed. In this way, the algorithm will converge to the minimum global.

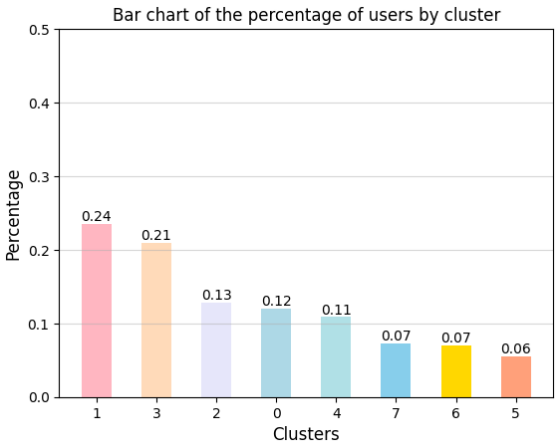


Fig. 2: Percentage of users per cluster obtained by K-means algorithm

and 3 account for almost 50% of network users, the remaining 6 clusters represent a smaller-percentage of the network users.

Fig. 3 and Fig. 4 are a violin plot (for numerical attributes) and a bar plot (for boolean attributes), respectively showing the distribution of numbers and boolean variables in each generated cluster. Fig. 3 accurately shows the distribution of each numerical variable for each generated group. Fig. 4 provides the bar charts for each of the boolean variables in the dataset. For each group, the original graph represents the percentage of users with attributes actived (orange) and the percentage of users without attributes actived (gray). Combining both figures with Table II enable us to analyze the formed groups:

Group 7 contains 7.28% of network users. Composed of accounts created between 2009 and 2011 that are fully authenticated, non protected, and with no default Twitter profile or image. In addition, they contain the largest number of followers and the largest number of followees. Those who are members of more public lists, and those who generate the most tweets. We call this group **influencers** because their descriptions match those of users who are generally considered "influencers" in social networks. That is to say, having a large number of fans and following the accounts of many users shows that they have an active and loyal audience. In addition, they are members of many public lists, indicating that their content is relevant to many users and widely shared. The fact that these accounts have been verified and active for several years also shows that they have certain authority and credibility on the platform.

Group 5 is the opposite to Group 7, it consists of unchecked accounts with images and profiles by default, which are newly created accounts between 2017 and 2022. Besides, they have very few fans and they write very few tweets. In addition, a small proportion belongs to public list. This contains 5.52% of the users which is the lowest percentage. We call this group **inactive** because their descriptions indicate that these accounts are not actively used. That said, they are recently created accounts with few fans, sug-

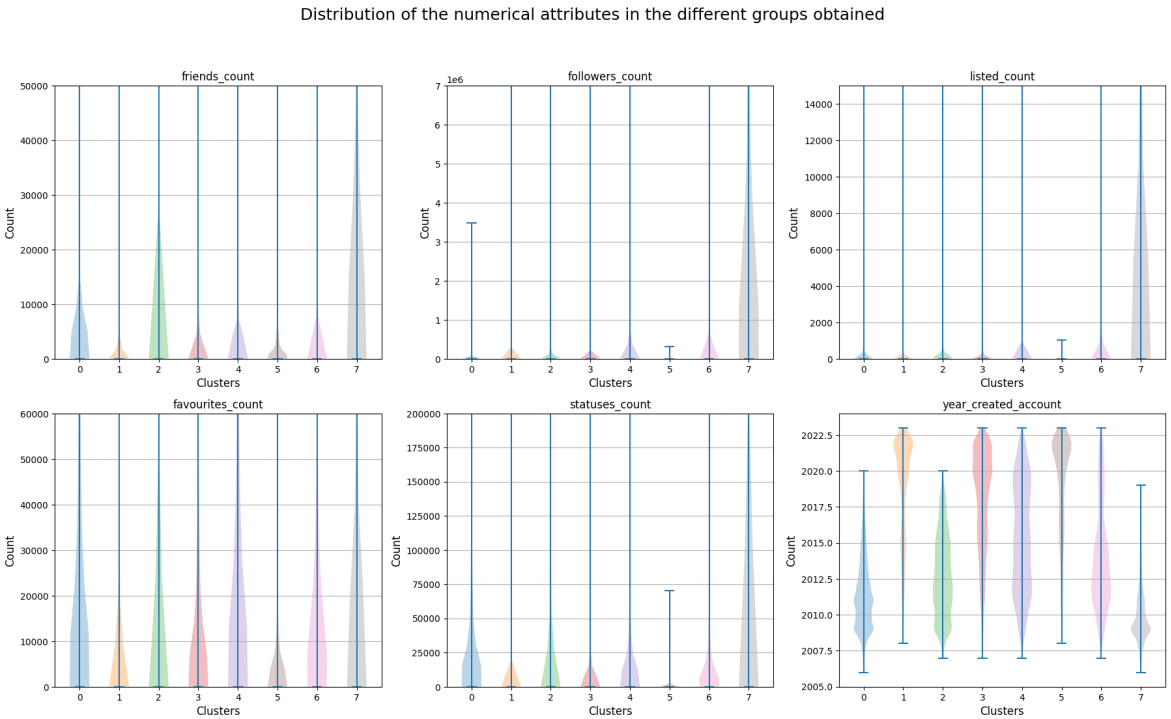


Fig. 3: Distribution of the numerical attributes in the different groups obtained without considering outliers

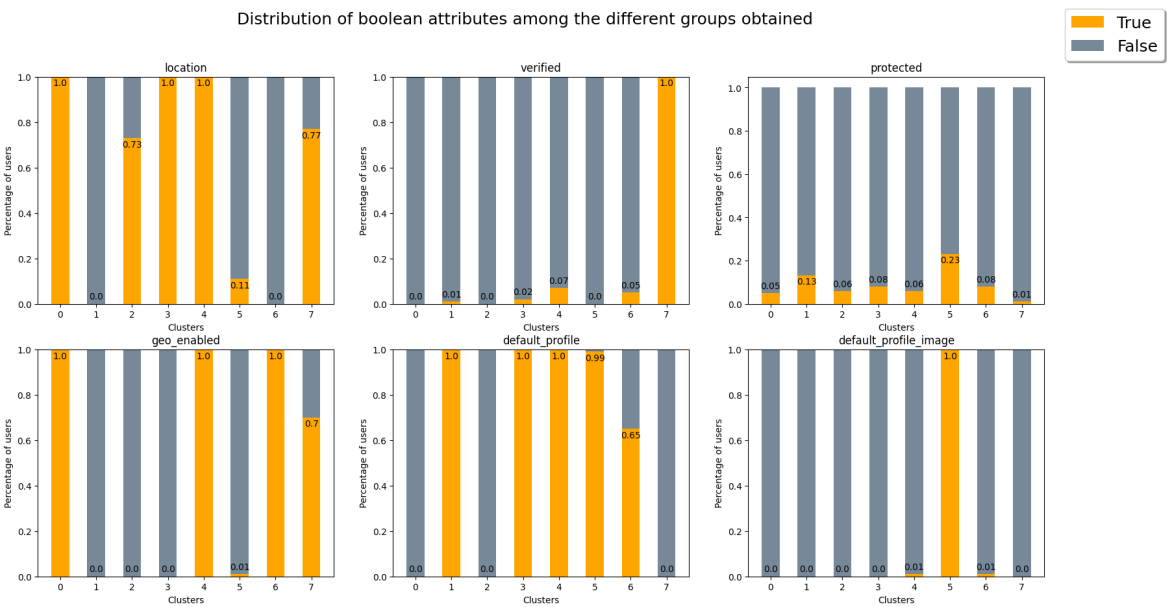


Fig. 4: Distribution of the boolean attributes in the different groups obtained

gesting they have failed to attract large audiences. In addition, they write few or no tweets and pay little attention to accounts, indicating that they are not actively interacting with other users on the platform.

Group 0 contains 12.02%, and it is defined by an unauthenticated account, there is no default profile or image. All of these enable geographic locations and record locations. Most of them are members of public lists and are very active users of tweets and likes. This group has been called **explorers** because that name indicates that they are users who are active on the platform and are exploring and discovering content. Enabling location services and providing location information, such as in tweets and link posts, indicates that users are interested in engaging with platform content positively.

Group 2 contains 12.84%, which was similar to explorer group but slightly different, is an unauthenticated account, has no activated geographic location and no default profile or predefined image. However, 73% of accounts have localization enabled. It is a user with many friends, but by contrast, there are few fans. The vast majority of users who write tweets are not members of any public list and have lower activity levels than other users on the platform, resembling passive users rather than active ones. This group is called **connected**. The name is based on the presence of quite a lot friends, but only has few fans indicates that these users are not actively seeking to increase their presence on the platform, but are

paying more attention to interact with close friends. The fact that most of these accounts are location-enabled also indicates that they are interested in sharing information about their location and possibly connecting with other users in their geographic area.

Group 1 contains 23.55% of network users. It consists of almost unauthenticated accounts, of which 13% are protected. All of these have default profiles, no registered locations, and disabled geographic locations. Most accounts were created between 2011 and 2015. It is a public list account, which occasionally generates tweets and does something they like, but it won't become very active. In both cases, the number of followers and followees is relatively small. Users who form part of this cluster are called **basic users**. They feature basic configuration files and configurations, without much activity or validation, but are still part of the network and contribute to its existence.

Group 3, Group 4 and Group 6 contains 20.89%, 10.87% and 7.01% respectively, which are almost 40% of internet users. These clusters are very similar to each other: they have very similar distribution in the number of followers and followees, likes and tweet generation. The difference is that Group 3 users are a slightly smaller part of public lists and usually contain a slightly smaller number of fans. Similarly, Group 3 enabled localization and disabled geographic location, Group 6 disables location and enables geographic location, and Group 4 enables both. In addition, all accounts in Groups 3 and 4 are default profile accounts, more than 60% accounts in Group 6 are as well . The vast majority of accounts in Group 3 were created between 2015-2022, Group 6 between 2008-2017 and Group 4 covers a wider range of years, created between 2009-2022. Therefore, we can call Groups 3, 4, and 6 **standard users** because they are not very different from each other.

Summarizing, six different dataset groups have been identified:

- Group 7 is **influencer**: authenticated accounts, no protection, lots of fans and ongoing activity.
- Group 5 is **inactive**: authenticated accounts, few followers, no activity, and no members of the public list.
- Group 0 is **explorer**: unauthenticated accounts, no default images or profiles, active geographic locations and continuous activity, and members of some public lists.
- Group 2 is **connected**: unauthenticated accounts, no activated geographic location, but most accounts have registered locations, many friends, and fairly stable activity. Most accounts are either members of a few public lists or they are not.
- Group 1 is **basic**: unauthenticated accounts with default profiles, no activated geographic location, fairly consistent activity, and few followers and members in the public list.
- Groups 3, 4 and 6 are **standard**: they are not as unique as other groups.

2. Analysis of the topics of users' tweets

In addition to considering the users' attributes, we are also going to analyze users by the content of their tweets.

For this part, Deep Learning(DL) has been used for the extraction of the content that exists in a tweet. In particular, Natural Language Inference(NLI) [22] model has been used for sequence classification.

NLI refers to the task of determining the logical relationship between two given sentences: a premise and a hypothesis. The goal is to determine whether the hypothesis can be inferred from the premise, typically expressed as a binary classification task. The main idea of NLI models is to take the sequence that we are interested in labeling as the "premise" and convert each candidate label into a "hypothesis". If the NLI model predicts that the premise implies the hypothesis, the label is taken to be true.

Based on these, it is intended to verify the type of content offered by each group of users found in the first characterization. Relating in this way both the information of the type of account and the content it offers.

The model we use is facebook/bart-large-mnli[23]. This is part of the family of BART models, which use the sequence-to-sequence denoising pre-training technique (Denoising AutoEncoder[2]) to improve comprehension and text generation. Therefore, the models that use BART are capable of performing text generation, translation and natural language processing tasks.

The facebook/bart-large-mnli model is a developed pre-trained natural language model using the framework of BART[24] work and trained with the dataset MultiNLI (MNLI)[25]. The MNLI dataset on the model, which is a large and diverse dataset containing more than 400k examples of various topics, allowing the model to have a good performance in the task of text classification for multiple topics. This set of data has been used mainly for the task of classifying text, in which an attempt is made to classify a fragment of text into one of three categories: neutral (neutrality), entailment (implication) or contradiction (contradiction). Therefore, the output of the model is a vector of the three values which correspond to the probabilities of the premise. Each value represents the estimated probability that belong to the premise of each corresponding category. The category with the highest probability is considered the prediction of the model.

In our case, a threshold[3] of 0.85 has been defined in the implication category to determine that a specific topic belongs to a particular tweet.

In addition, the model has the option of activating

[2]The Denoising Sequence-to-Sequence technique seeks to train natural language models that are capable of generating more coherent and cleaner text, eliminating noise and inconsistencies that may appear in the generated word sequences.

[3]Limit value that is used to perform a classification.

the mode multi-label. This allows for handling multiple output tags instead of a single tag as previously described. This means that instead of classifying a pair of sentences into a single category of semantic relationship, such as contradiction, neutral, or entailment, the model can classify the pair of sentences in function of multiple labeled categories and provide a confidence measure for each label.

In the context of MNLI, where the model is trained and evaluated across multiple topics, this is especially useful as more than one topic can appear in the same sentence.

When multi label mode is activated, the model also provides a measure of confidence for each output label. This means that the model not only tells in which topic categories the semantic relationship between the sentences fits, but it also indicates how confident the model is about each of those categories. Therefore, for each tweet, each topic that had an entailment with a value greater than or equal to 0.85 has been taken as valid.

Tabla III: Percentage of users showing tweets by group.

Group	The percentage of users with tweets	The percentage of users without tweets
Influencer	91.87%	8.13%
Inactive	11.2%	88.8%
Basic	74.13%	25.87%
Explorer	65.55%	34.45%
Connected	80.47%	19.53%
Standard	55.51%	44.49%

Table III shows the percentage of users in each group from which we could get the "last 5 tweets". Based on these data, the content generated by each group is analyzed.

The total number of topics used for analysis is 16. The following are some examples of topics that may be popular in each category:

News: The latest local or international news.

Travel: Travel destinations, travel tips, personal experiences, photos of interesting places, etc.

Food: Recipes, restaurant suggestions, food reviews, cooking trends, etc.

Politics: Political analysis, discussion on hot issues, views on policies and policies, etc.

Entertainment: Movies, TV programs, music, celebrities, entertainment activities, etc.

Sports: Game results, sports news, debates about teams and players, strategic analysis, etc.

Fashion: Fashion trends, fashion tips, brand and designer reviews, appearance of the day, etc.

Health: Health tips, health news, exercise programs, dietary advice, etc.

Technology: Latest technology developments, equipment reviews, platform and application discussions, productivity recommendations, etc.

Fig. 5 presents the 5 most related topics for each group. Analyzing these results we can see that the repeated topics in all the groups are News, Travel, Entertainment and Technology. Sports is repeated in 5 groups, Food only appears in the inactive group. This is the only difference between the inactive group and the other five groups.

So, as it can be seen, the distribution of all topics is very similar, and the topic they wrote more is about News. Therefore, we can say that there is no difference in content among different groups, and hence, all groups have very similar diversity.

V. Conclusions

The analysis of social network user attributes provides valuable insights into user behavior and interests. We are able to identify different groups of users with different characteristics by examining factors such as the number of followers, posting frequency or location data. Most users are active on the social network, what indicates that they are interested in participating in the platform's content.

In general, describing the attributes of social networks users helps to better understand user behavior, which can be used to inform marketing and advertising strategies, as well as platform design and improve user experience. By continuing to analyze user data and identify patterns, we can learn more about how users interact with social media platforms.

Therefore, our future work will based on this work and create an agent-based model that can be used to analyze social media users behavior and interactions. Then we can extend this model by adding different algorithms to make it possible to work on some specific situation, for instance, analyze users preferences and then advertise more accurate or analyze how the memes propagate among users of different ages.

Acknowledgement

This work has been supported by the Ministerio de Ciencia e Innovación MCIN AEI/10.13039/501100011033 under contract PID2020-113614RB-C21 and by the Catalan government under contract 2021 SGR 00574.

Referencias

[1] Nicholson Collier and Michael North, "Parallel agent-based simulation with repast for high performance computing," *SIMULATION*, vol. 89, pp. 1215–1235, 10 2012.

[2] Rui Fan, Ke Xu, and Jichang Zhao, "An agent-based model for emotion contagion and competition in online social media," *Physica A: Statistical Mechanics and its Applications*, vol. 495, pp. 245–259, 2018.

[3] Anna Gausen, Wayne Luk, and Ce Guo, "Using agent-based modelling to evaluate the impact of algorithmic curation on social media," *Journal of Data and Information Quality*, vol. 15, 07 2022.

[4] Tayfun Tuna, Esra Akbas, Ahmet Aksoy, M Abdullah Canbaz, Umit Karabiyik, Bilal Gonen, and Ramazan Aygun, "User characterization for online social networks," *Social Network Analysis and Mining*, vol. 6, pp. 104, 11 2016.

[5] Usman Anjum, Vladimir Zadorozhny, and Prashant Krishnamurthy, "Tbam: Towards an agent-based model to enrich twitter data," 2023.

[6] Marcelo Maia, Jussara Almeida, and Virgílio Almeida, "Identifying user behavior in online social networks," in *Proceedings of the 1st Workshop on Social Network Systems*, New York, NY, USA, 2008, SocialNets '08, p. 1–6, Association for Computing Machinery.

[7] Enrico Di Minin, Christoph Fink, Henrikki Tenkanen, and Tuomo Hiippala, "Machine learning for tracking illegal wildlife trade on social media," *Nature ecology & evolution*, vol. 2, no. 3, pp. 406–407, 2018.

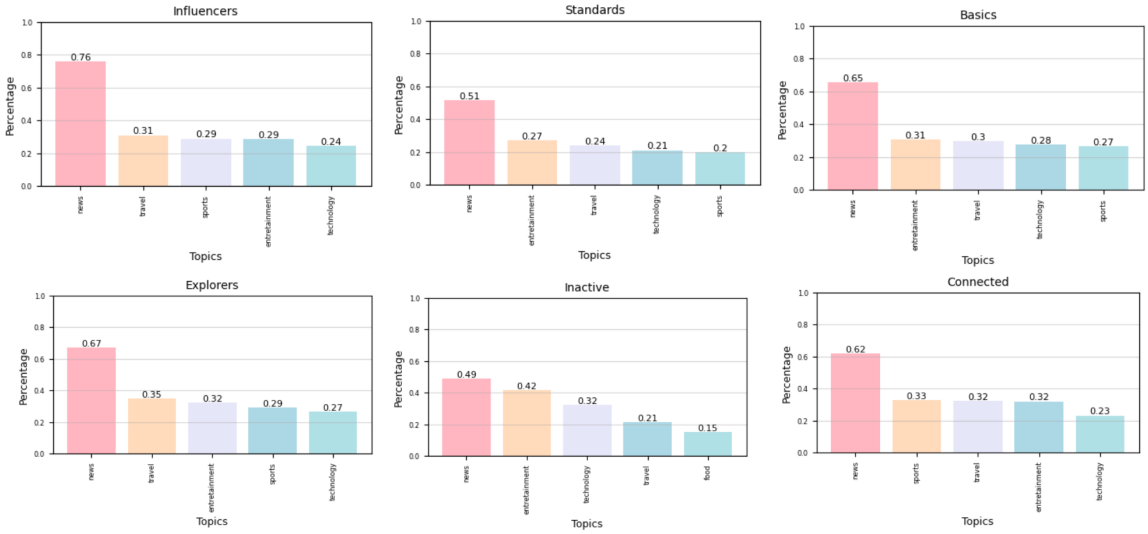


Fig. 5: Top 5 most written topics by the users of each group.

[8] Lior Rokach and Oded Maimon, *Decision Trees*, pp. 165–192, Springer US, Boston, MA, 2005.

[9] S. B. Kotsiantis, "Decision trees: a recent overview," 2013.

[10] Sayali D Jadhav and HP Channe, "Comparative study of k-nn, naive bayes and decision tree classification techniques," *International Journal of Science and Research (IJSR)*, vol. 5, no. 1, pp. 1842–1845, 2016.

[11] Kazunaga Matsuki, Victor Kuperman, and Julie A Van Dyke, "The random forests statistical technique: An examination of its value for the study of reading," *Scientific Studies of Reading*, vol. 20, no. 1, pp. 20–33, 2016.

[12] Grzegorz Dudek, "Short-term load forecasting using random forests," in *Intelligent Systems' 2014: Proceedings of the 7th IEEE International Conference Intelligent Systems IS'2014, September 24-26, 2014, Warsaw, Poland, Volume 2: Tools, Architectures, Systems, Applications*. Springer, 2015, pp. 821–828.

[13] Alessia Mammone, Marco Turchi, and Nello Cristianini, "Support vector machines," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 3, pp. 283–289, 2009.

[14] S Amarappa and SV Sathyanarayana, "Data classification using support vector machine (svm), a simplified approach," *Int. J. Electron. Comput. Sci. Eng*, vol. 3, pp. 435–445, 2014.

[15] FAN Fernandes and LMF Lona, "Neural network applications in polymerization processes," *Brazilian Journal of Chemical Engineering*, vol. 22, pp. 401–418, 2005.

[16] Jack V Tu, "Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes," *Journal of clinical epidemiology*, vol. 49, no. 11, pp. 1225–1231, 1996.

[17] John A Hartigan, Manchek A Wong, et al., "A k-means clustering algorithm," *Applied statistics*, vol. 28, no. 1, pp. 100–108, 1979.

[18] Shi Na, Liu Xumin, and Guan Yong, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," in *2010 Third International Symposium on intelligent information technology and security informatics*. Ieee, 2010, pp. 63–67.

[19] Joshua Roesslein, "tweepy documentation," *Online] http://tweepy. readthedocs. io/en/v3*, vol. 5, pp. 724, 2009.

[20] Aric Hagberg and Drew Conway, "Networkx: Network analysis with python," *URL: https://networkx. github. io*, 2020.

[21] Mengyao Cui et al., "Introduction to the k-means clustering algorithm based on the elbow method," *Accounting, Auditing and Finance*, vol. 1, no. 1, pp. 5–8, 2020.

[22] Bill MacCartney, *Natural language inference*, Stanford University, 2009.

[23] Kenneth Alperin, Emily Joback, Leslie Shing, and Gabe Elkin, "A framework for unsupervised classificiation and data mining of tweets about cyber vulnerabilities," *arXiv preprint arXiv:2104.11695*, 2021.

[24] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online, July 2020, pp. 7871–7880, Association for Computational Linguistics.

[25] Adina Williams, Nikita Nangia, and Samuel Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana, June 2018, pp. 1112–1122, Association for Computational Linguistics.

Sistema de Ficheros Distribuido para IoT basado en Expand

Elías Del-Pozo-Puñal[1], Félix García-Carballeira[1], Diego Camarmas-Alonso[1] y Alejandro Calderón-Mateos[1]

Resumen— **El continuo aumento de uso de sensores y dispositivos inteligentes para aplicaciones domésticas en los últimos años ha provocado la implantación de infraestructuras y entornos capaces de gestionar, procesar y almacenar toda la información generada en un corto periodo de tiempo. Estos entornos tienen que tratar con la problemática de un entorno cambiante (puede aumentar o disminuir el número de sensores y dispositivos inteligentes) y con posibles picos en la demanda de recursos (pudiendo llevar a la congestión de la red o la posible pérdidad de datos durante esos periodos de tiempo). Por estas razones, en el presente trabajo se muestra la adaptación de un sistema de ficheros, focalizado en entorno del Internet de las Cosas (IoT) para la captación, tratamiento y almacenamiento de los datos generados por sensores a lo largo del tiempo.**

Palabras clave— **Almacenamiento IoT, MQTT, Sistema de Ficheros Paralelo y Distribuido, Expand**

I. Introducción

LOS sistemas de ficheros paralelos y distribuidos actuales se han ido diseñando y desarrollando para las necesidades que han ido surgiendo en las pasadas décadas. Las nuevas necesidades en las nuevas aplicaciones, nuevos dispositivos de almacenamiento, etc. hacen que tengan que adaptarse.

A modo de ejemplo, las aplicaciones en las áreas de Inteligencia Artificial o *Big Data* están evolucionando rápidamente y demandando el tratamiento masivo de datos, por ejemplo a través de la ejecución de *workflows*. Estos datos pueden provenir de multitud de sensores que se recolectan y a través de una red de dispositivos IoT (*Internet of Things*) precisan de cierto tratamiento, envío y almacenamiento. En esta evolución de las aplicaciones hay nuevas necesidades de llevar parte del cómputo e incluso un almacenamiento parcial a los propios dispositivos.

Estas áreas de trabajo precisan de un entorno de computación de alto rendimiento, pero su evolución en los patrones de acceso a los diferentes datos, etc. supone una gran diferencia a las aplicaciones de computación de alto rendimiento (HPC) tradicionales [1] [2]. Es preciso que el sistema de almacenamiento utilizado para guardar y recuperar los datos esté diseñado para este tipo de nuevas aplicaciones y ofrezca unas prestaciones adecuadas a ellas.

Durante los últimos años ha aumentado el uso de este tipo de dispositivos y se espera que para los próximos años alcance la cantidad de cien mil millones de dispositivos conectados [3] [4].

No obstante, la existencia de una alta cantidad de dispositivos presenta problemas de congestión de tráfico y pérdida de información que, dependiendo del caso de uso, puede ser crítico. Por eso, existen diferentes mecanismos o protocolos capaces de evitar o, en su defecto, tratar tales problemas. Estos protocolos de comunicación están pensados para asegurar que la información llega a su destino o para que la mayoría de esta información no se pierda, como son protocolos sobre TCP/IP [5] o MQTT [6].

La elección entre protocolos como HTTP, SMTP, etc. o bien MQTT dependerá de las necesidades y restricciones específicas de cada aplicación o entorno. Si para un tipo de aplicaciones la confiabilidad, la entrega garantizada de datos, el control de congestión y la latencia son determinantes, HTTP/HTTPS puede ser una buena opción para el envío de datos entre extremos. No obstante, si lo que se busca es la eficiencia en el consumo de recursos, la escalabilidad, notificaciones asíncronas y, además, una menor sobrecarga de la red, MQTT puede ser una alternativa para ese tipo de infraestructuras. Ambos protocolos tienen sus ventajas e inconvenientes, por lo que es importante analizar y determinar con cautela los requisitos y limitaciones del sistema.

Se parte de una versión de Expand Ad-Hoc diseñada inicialmente para entornos HPC cuya descripción está en [7]. Este trabajo presenta un enfoque inicial donde usaremos el sistema de ficheros distribuido y Ad-Hoc Expand para que los sensores IoT almacenen todos los datos generados a lo largo del tiempo con llamadas del estándar POSIX. La parte cliente de Expand (que implementa la interfaz POSIX para los sensores IoT) enviará los datos a unos servidores utilizando un intercambio de mensajes basado en TCP/IP [8] (Expand TCP).

Partiendo de este enfoque inicial el siguiente paso consiste en adecuar el diseño a uno más adecuado para sistemas IoT utilizando el intercambio de mensajes basado en MQTT con la implementación Mosquitto (Expand MQTT). Con este enfoque se consigue que los propios sensores puedan utilizar servicios POSIX para almacenar los datos que generan en la capa Fog o Cloud. En las Figuras 1 y 2 se pueden ver las diferencias que existen entre dos posibles enfoques: un uso tradicional y un uso de MQTT con *brokers*.

Este trabajo está organizado de la siguiente forma: En la Sección II se describe el Estado del Arte, seguido de una explicación del paradigma en la Sección III. Después, se realiza una evaluación del acercamiento desarrollado en la Sección IV. Finalmente, se detallan una serie de líneas futuras a seguir y unas breves conclusiones en la Sección V.

[1]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: {edelpozo,fgcarbal,dcamarma,acaldero}@inf.uc3m.es

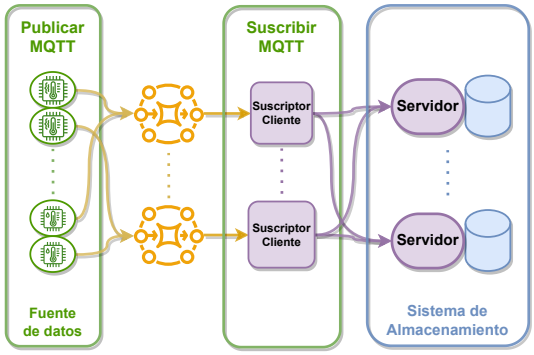


Fig. 1: Diagrama caso de uso: Enfoque tradicional de MQTT para el envío de la información

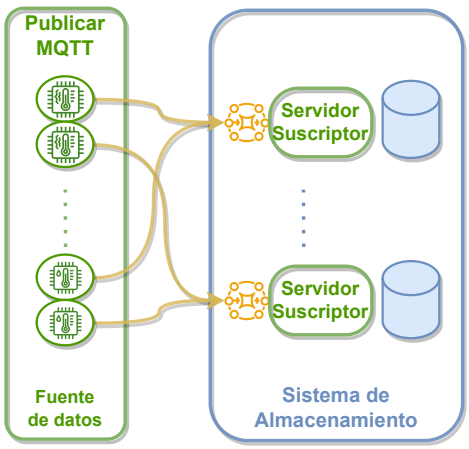


Fig. 2: Diagrama caso de uso: Enfoque donde los sensores usan las llamadas de POSIX para el almacenamiento de los datos

II. Estado del arte

En esta sección se analizará los sistemas de almacenamiento adaptados para Internet de las Cosas (IoT). Para ello se presentarán, por un lado, los aspectos más destacados del Internet de las Cosas (IoT). Por otro lado, se presentará el protocolo MQTT, revisando tanto las ventajas como las desventajas de MQTT en comparación con HTTP/etc., que es comúnmente utilizado para el envío y recepción de mensajes en entornos distribuidos y paralelos.

A. Internet of Things (IoT)

Internet of Things o Internet de las Cosas es un paradigma que ha ido evolucionando y teniendo mucho impacto en la vida de las personas durante los últimos años. Gracias al propio uso de Internet, ámbitos como la Educación, Ciencia, Tecnología, etc. han visto incrementada su productividad y uso dado que facilita y agiliza el uso, análisis y procesamiento de la información a pequeña y gran escala [9].

Es, además, una arquitectura que facilita la comunicación entre varios puntos de la red para la transmisión de diferentes tipos de datos, ofreciendo una mayor capacidad y velocidad de acceso a la información y así aumentar la productividad en diferentes servicios [10].

Gracias a que durante la última década han ido surgiendo nuevos dispositivos conectados como son por ejemplo los *Smartphones* o teléfonos inteligentes, las *tablets*, *Smartwatches*, etc. que influido en el crecimiento drástico de la cantidad de dispositivos conectados a Internet. Por ende, ha aumentado sustancialmente el tránsito de información entre los diferentes puntos de la red, obligando a que haya una mayor cantidad de infraestructuras capaces de gestionar en tiempo real todo este tránsito diariamente [11].

Otro acercamiento a los dipositivos IoT consiste en tratarlo como un superconjunto de dispositivos conectados que solo pueden ser identificados por técnicas de comunicación de campo cercano o *Near Field Communication* (NFC).

Un aspecto importante de los dispositivos IoT a tratar son los diferentes tipos de modelos de comunicación para poder determinar la mejor forma de conectarlos entre ellos [12] que se indican a continuación [3]:

- Comunicaciones dispositivo a dispositivo (D2D): Es la comunicación más básica en la que dos o más dispositivos se conectan y comunicam directamente entre ellos. Se pueden comunicar en varios tipos de redes, como redes IP o Internet, pero en su mayoría se suelen comunicar usando *Bluetooth*. Esto puede ser útil en entornos donde la latencia y la privacidad son importantes, como en el área de la salud.
- Comunicaciones dispositivo a nube (D2C): Esta comunicación implica que el dispositivo IoT se conecta directamente a un servicio en la nube. A menudo se usan conexiones Ethernet o WiFi pero también se suele usar redes 4G o 4.5G. La nube puede entonces enviar comandos de vuelta a los dispositivos, lo que permite el control remoto de los mismos. Este modelo es útil cuando se requiere un gran poder de procesamiento o almacenamiento de datos.
- Comunicaciones nube a dispositivo (C2D): En este caso, el servicio en la nube se conecta al propio dispositivo para controlarlo o monitorizarlo y modificar su comportamiento. Esto puede ser útil para el entorno del hogar y controlar diferentes componentes como puede ser termostatos o cerraduras.
- Comunicaciones nube a nube (C2C): Por último, este modelo de comunicación está focalizado en el caso en el que diferentes nubes de diferentes proveedores se comunican entre sí para compartir información y servicios entre ellos, lo que resulta útil para sistemas de integración de sistemas IoT.

Por último, a parte de los diferentes modelos de comunicación existen diferentes protocolos de comunicación bastante extendidos para el envío y recepción de datos entre dispositivos de IoT, como son MQTT o CoAP [13]:

- MQTT (*Message Queing Telemetry Transport*): Protocolo de comunicación basado en publicación-suscripción que permite la comunicación entre un gran número de dispositivos. Existe un servidor central llamado *broker* que recibirá los

mensajes de los dispositivos emisores y los distribuirá entre los receptores.
- CoAP (*Constrained Application Protocol*): Protocolo entre dispositivos de bajo consumo que usa el modelo REST de HTTP junto con soporte a UDP.

B. MQTT

Como se ha comentado anteriormente, MQTT [14] [15] (*Message Queing Telemetry Transport*) es un protocolo de red ligero sobre TCP/IP, basado en la publicación y suscripción, desarrollado por IBM en 1999, cuyo objetivo consistía en enviar datos con precisión en condiciones de gran retardo en la red y con poco ancho de banda. No obstante, en la actualidad se usa ampliamanete en aplicaciones de IoT y M2M (*Machine-to-Machine*).

La principal característica de este protocolo de transporte es la simplicidad y la fiabilidad en entornos o infraestructuras donde el ancho de banda está limitado y donde hay conexiones inestables. Otro aspecto a destacar es que, como se basa en el modelo de publicación/suscripción, los dispositivos publican los datos o mensajes en "temas" o "*topics*" y los dispositivos receptores que se suscriban a esos temas recibirán la información publicada de forma automática.

Los componentes más importantes que conforman el protocolo MQTT son los siguientes:

- *Broker* MQTT: Servidor central de la arquitectura de MQTT donde se encargará de recibir los mensajes publicados en un tema específico y reenviarlos a todos aquellos dispositivos que se hayan suscrito al mismo tema. Para poder hacer esta conexión, todos los dispositivos deben conectarse previamente.
- Publicador/Suscriptor MQTT: Dispositivo que se conecta previamente al *broker* y se publica/suscribe a uno o varios temas para poder enviar/recibir los datos que el *broker* deberá recibir/enviar [16].
- Temas: Cadena de texto que se comporta como el identificador para organizar los mensajes que le lleguen al *broker*. Los temas se estructuran en una jerarquía de niveles separados por barras ("/") y, además, admite dos valores comodín para que los dispositivos puedan suscribirse a más de un tema del mismo nivel a la vez. Estos son el valor "+" y el valor "#".
- Calidad de Servicio (QoS): MQTT describe varios niveles de Calidad de Servicio (QoS) para garantizar la entrega fiable de mensajes entre los extremos [17]. Cada nivel de QoS ofrece un equilibrio diferente entre la fiabilidad y el rendimiento, y es importante entender las diferencias entre ellos para seleccionar el nivel adecuado según las necesidades de la aplicación. Los diferentes niveles son [18]:
  1. `QoS 0`: El mensaje se envía como mucho una vez y no garantiza que se entregue el mensaje a los dispositivos suscritos.
  2. `QoS 1`: La información se envía como mínimo una vez hasta que se garantiza la entrega a los suscriptores. Si falla, el suscriptor puede recibir mensajes duplicados.
  3. `QoS 2`: Calidad que garantiza que cada mensaje se entrega al suscriptor y únicamente una vez.

En resumen, MQTT es un protocolo ideal para bajo consumo energético, soporta todos los mensajes de la red (publicar/suscribir y solicitud/respuesta), tiene una calidad de servicio ajustable, y tiene una baja sobrecarga de protocolo [19].

No obstante, resulta ser un protocolo complejo para escenarios de alta disponibilidad y escalabilidad, requiere conectividad con un *broker* de MQTT, lo que puede introducir fallos al haber un único punto de comunicación y, por último, aunque MQTT está muy utilizado, no existe una implementación única y estandarizada [6].

C. Aplicaciones de MQTT en entornos distribuidos

Con respecto a las posibles aplicaciones de MQTT en sistemas distribuidos, destaca un diseño en el que se presenta una extensión de la idea que plantea MQTT. En vez de centrarse en la comunicación de los dispositivos IoT con un *broker* principal que hace de unión entre los extremos, los autores plantean incorporar más *brokers* diferentes que cooperan entre ellos de forma distribuida [20].

Se propone una solución enfocada en el descubrimiento de los *brokers* disponibles en el sistema y de la recuperación ante fallos de la infraestructura. Además, también focalizan su atención en la creación de redes superpuestas y reenvío de mensajes entre los *brokers*. En este trabajo también se aborda el problema de posibles múltiples *topics* en el sistema, proponiendo así un esquema de enrutamiento basado en *topics* para los diferentes *brokers*. Los autores proponen una solución como una extensión al *broker* MQTT HiveMQ [21] y realizan experimentos para analizar su rendimiento para varias configuraciones de publicadores/suscrptores y con varios *topics* en el sistema.

D. Sistemas de Ficheros existentes para IoT

Por último, existen diferentes sistemas de ficheros creados específicamente para los dispositivos IoT y que pueden adaptarse a las necesidades y limitaciones de los dispositivos o de la aplicación a usar. Entre ellos destacan:

- IPFS (*InterPlanetary File System*) [22]: Sistema de ficheros distribuido focalizado en la tecnología *blockchain* donde, en vez de usar la estructura de directorios tradicional, usa una estructura basada en contenido, donde cada fichero se identifica con su *hash* criptográfico. Ofrece una idea innovadora para el almacenamiento y la distribución de archivos pero se debe tener en cuenta sus limitaciones para su uso a gran escala.
- *MQTT based file delivery* [23]: Funcionalidad

que proporciona AWS IoT Core, que es el servicio de IoT de Amazon Web Services. Permite la transferencia eficiente de ficheros mediante el protocolo MQTT. Con este sistema, se pueden enviar y recibir archivos entre los dispositivos IoT y la nube de AWS. Está diseñado para la transferencia de archivos de tamaño pequeño por limitaciones de ancho de banda y recursos de los IoT.

III. Expand Ad-Hoc para IoT

Expand inicialmente se diseñó [7] como un sistema de ficheros paralelo para *clusters* heterogéneos usando servidores NFS [24] existentes. El diseño original se ha cambiado para disponer de un sistema de almacenamiento Ad-Hoc para IoT. Los siguientes apartados detallan este diseño general, así como los aspectos especialmente adaptados para su uso con aplicaciones IoT.

A. Diseño general

La Figura 3 muestra la arquitectura resultante del cambio del diseño de Expand para transformarlo en un sistema de ficheros paralelo Ad-Hoc para IoT.

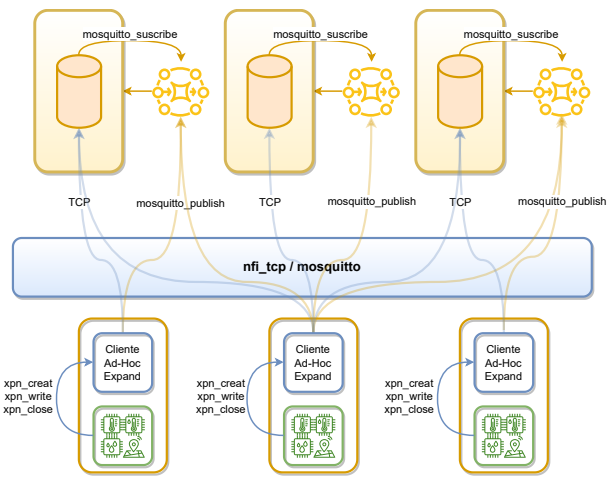


Fig. 3: Arquitectura de Expand Ad-Hoc for IoT.

Los distintos nodos IoT ejecutan las aplicaciones que usan una biblioteca con el cliente de Expand. Dicho cliente de Expand ofrece una interfaz similar a POSIX, de forma que las nuevas aplicaciones usen la interfaz nativa de Expand. No obstante, las aplicaciones existentes pueden beneficiarse también de una biblioteca de interceptación, la cual permite utilizar las llamadas a POSIX en las aplicaciones sin tener que modificar el código.

La biblioteca cliente de Expand se comunica con los distintos servidores de Expand mediante el uso de una implementación basada en sockets de TCP/IP o bien una implementación basada en MQTT. Los servidores de Expand se ejecutan en nodos de cómputo, de manera que cada servidor utiliza el espacio de almacenamiento local de su nodo (ya sea SSD, disco duro, etc.) para guardar datos. Los datos que envía el cliente de Expand se distribuyen entre un conjunto de servidores que forman una partición paralela, de forma que los accesos para guardar (o para leer) los datos se hace en paralelo (véase la Figura 4).

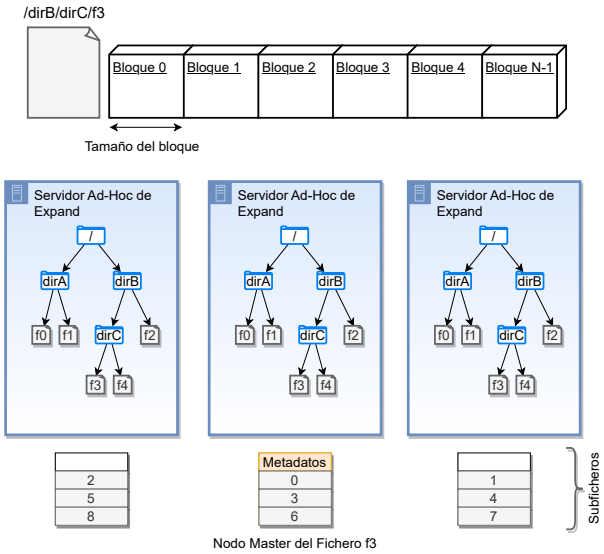


Fig. 4: Estructura de los ficheros y directorios en Expand.

Para facilitar la puesta en marcha de estos nodos de almacenamiento, los servidores de Expand son aplicaciones que pueden ejecutarse en espacio de usuario y que no precisan de una instalación particular (puede compilarse el código fuente de Expand para generar el binario del servidor de Expand, así como la librería cliente de Expand).

Este diseño facilita tanto que se usen nodos servidores de almacenamiento como también que haya nodos IoT especializados de almacenamiento. Incluso combinar ambas, tener unos servidores de almacenamiento en nodos IoT y copiar la información desde allí a nodos especializados de almacenamiento, por ejemplo, en la nube.

En la Figura 5 se puede ver el proceso de ejecución de una posible aplicación que use la implementación basada en el protocolo TCP/IP donde el sensor creará un fichero, enviará datos que se hayan generado para poder ser almacenados y procesados y, finalmente, se cerrará el fichero una vez termine de recoger datos. Como se puede ver, para cada operación que realice el sensor, tendrá que esperar confirmación del servidor para determinar si se ha realizado correctamente, lo que conlleva un tiempo de espera extra para realizar la siguiente operación.

B. Aspectos relevantes para IoT

Las aplicaciones IoT pueden que no dispongan de una conexión TCP estable durante todo el tiempo en el que están ejecutando. Para facilitar este escenario, las operaciones que solicita el cliente de Expand a los servidores de Expand se pueden realizar mediante conexión por operación, de manera que no se mantiene una conexión TCP durante toda la ejecución de la aplicación IoT sino que en cada operación se realiza una conexión.

El protocolo TCP permite la transmisión de datos asegurando orden (usando números de secuencia

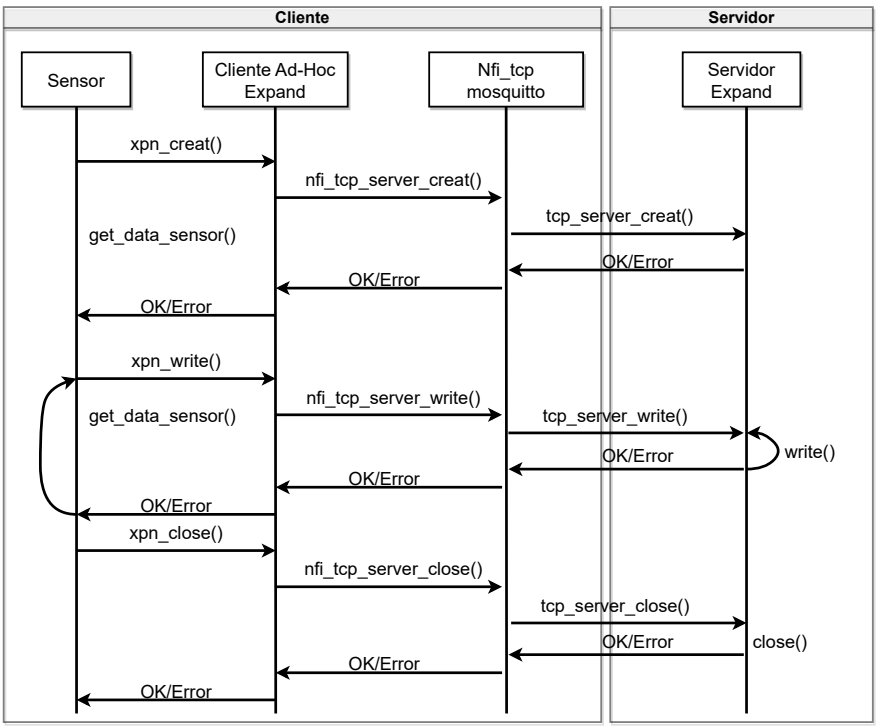


Fig. 5: Diagrama Expand TCP.

por ejemplo) y la llegada (usando retransmisión por ejemplo). No obstante, TCP solo asegura la entrega del mensaje mientras el receptor esté conectado. En algunos escenarios de aplicaciones IoT puede tolerarse un grado de pérdida de datos en el caso de que el subscritor no esté conectado, por lo que el coste podría reducirse bastante (especialmente en dispositivos sensibles al consumo energético). Expand permite el uso de MQTT para ayudar a simplificar el envío de datos (es un protocolo binario eficiente) y la entrega de datos (permitiendo decidir la entrega cuando el subscriptor vuelva a estar conectado).

No obstante, conviene explicar más en profundidad aquellos detalles más relevantes de cada protocolo. Ambos tienen características únicas que pueden ser beneficiosas según los escenarios.

Usando un diseño basado en el protocolo TCP/IP [25] [26] se obtiene:

1. Entrega de datos garantizada: Garantiza entrega ordenada y sin errores en los datos, lo que resulta importante en un sistema de ficheros, donde la integridad de los datos es crítica.
2. Control de congestión: TCP monitorea y controla el flujo de datos en función de la capacidad de la red, evitando la saturación y el bloqueo de recursos, lo que puede resultar útil en un sistema de ficheros con varios dispositivos enviando información a la vez.
3. Amplia compatibilidad: TCP es ampliamente usado e implementado en casi todos los dispositivos y sistemas operativos modernos, por lo que la probabilidad de encontrarse problemas de compatibilidad será menor.
4. Latencia: Ofrece una baja latencia y una alta velocidad, importante en un sistema de ficheros donde puede haber una gran cantidad de datos que transferir en tiempo real.

Por otro lado, si se eligiese un diseño basado en el protocolo MQTT [27] [15] se obtendrían las siguientes ventajas:

1. Ligero y eficiente: Es más eficiente en consumo de ancho de banda y uso de energía ya que está diseñado para dispositivos con recursos y capacidades de procesamiento y almacenamiento limitados.
2. Suscripción basada en *topics*: Los dispositivos pueden suscribirse a temas relevantes y recibir solo la información que desean, lo que resulta útil en sistemas donde diferentes dispositivos requieren varios conjuntos de datos.
3. Escalabilidad y notificaciones asíncronas: Es muy escalable y permite la comunicación asíncrona entre los dispositivos y el servidor, lo que puede ser beneficioso para notificar eventos y cambios en tiempo real.
4. Menor sobrecarga de red: MQTT usa una cabecera más pequeña y se usa un modelo de publicación/suscripción que reduce la sobrecarga de red, por lo que puede ser una ventaja frente a sistemas con un ancho de banda limitado.

C. MQTT en Expand

Como se ha dicho anteriormente, se quería analizar el impacto en el rendimiento entre usar o no un diseño basado en MQTT en un sistema de fichero distribuido y paralelo. Dado que el sistema de ficheros Expand posee ya una implementación basada en TCP, se procedió a modificar dicho diseño para permitir utilizar aplicaciones IoT con el protocolo MQTT.

En este ámbito, existen varias implementaciones,

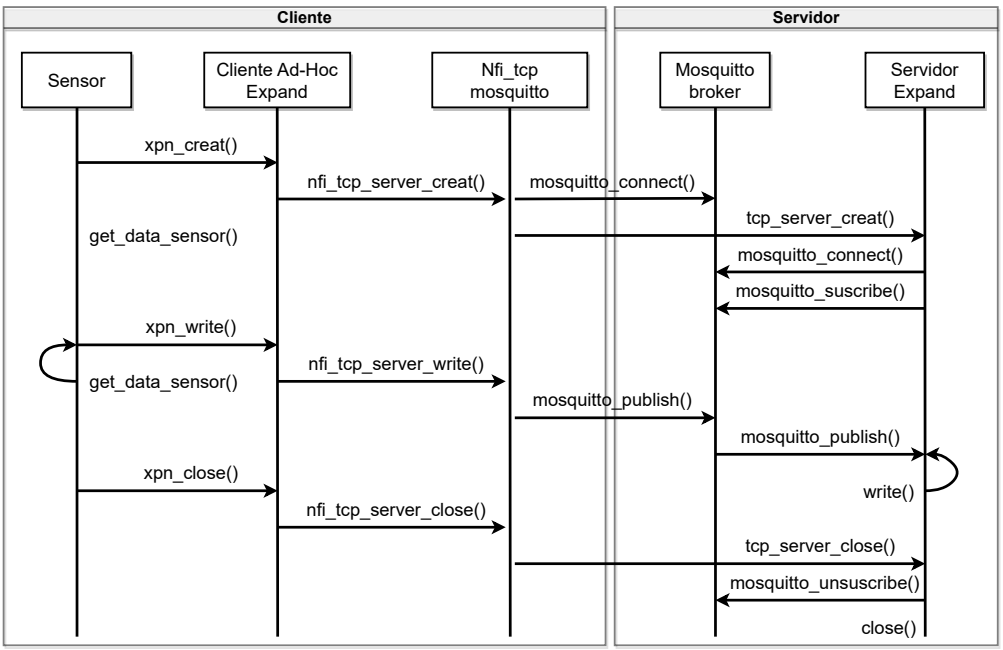


Fig. 6: Diagrama Expand MQTT.

que ofrecen soporte para utilizar el protocolo MQTT, como son Eclipse Mosquitto [28], HiveMQ [21] y ActiveMQ [29], entre otros.

Entre las posibles elecciones existentes [21] en este ámbito se ha decidido usar Mosquitto ya que destaca por su simplicidad, ligereza y facilidad de uso, lo que lo hace adecuado para implementaciones sencillas y entornos con recursos limitados. HiveMQ destaca por su escalabilidad y su gestión de sesiones y control de calidad de servicio, lo que lo hace una opción viable para escenarios de alta carga. Por último, ActiveMQ es una solución que permite el uso de diferentes protocolos y ofrece unas características que le ofrecen mayor flexibilidad y control de entrega de mensajes.

Una vez elegida la implementación de MQTT para este trabajo, es importante explicar el proceso de diseño de las capacidades que posee Mosquitto al sistema de ficheros Expand y las diferentes decisiones que se han llevado a cabo para su correcto funcionamiento.

El proceso de ejecución sigue el orden especificado en la Figura 6, donde se tiene el lado cliente, formado por el Sensor, el Cliente Ad-Hoc de Expand y el nfi (*Network File Interface*) de TCP/IP y Mosquitto, y, además, el lado servidor, formado por el *broker* de Mosquitto y el propio servidor de Expand. Lo que se pretende mostrar y explicar con la Figura es una ejecución habitual del sistema de ficheros donde se crea y abre un fichero, y se trabaja sobre él, para, finalmente cerrarlo pero haciendo uso de la implementación basada en el protocolo de MQTT. Los pasos para ello son:

1. Primero, los sensores que formen parte de la infraestructura arrancarán y piden al cliente de Expand que creen un fichero en los servidores asociado a su número de cliente y su identificador de sensor (p.ej.: test_1_1, test_idDispositivo_idSensor).
2. A su vez, el cliente de Expand creará una conexión con el *broker* de Mosquitto para publicar posteriormente los datos a almacenar en los servidores de Expand.
3. Al mismo tiempo, el servidor de Expand recibirá por TCP la creación de un fichero en sus nodos y creará una conexión con el *broker* para suscribirse al fichero creado y esperar la llegada de datos. Para que el servidor de Expand pueda atender todos los mensajes asociados a un fichero, tendrá que suscribirse usando el valor comodín "/#" (p.ej.: path_fichero/test_1_1/#).
4. Una vez creado el fichero en el servidor, los sensores recogerán datos cada cierto tiempo y publicarán esta información usando `mosquitto_publish` a través del cliente de Expand, indicando el fichero asociado, la posición desde la que escribir y la cantidad de datos a escribir. Toda esta información se enviará en el *topic* al que está escuchando el servidor de Expand (p.ej: path_fichero/test_1_1/*offset*/*position*/*bytes*).
5. El *broker* ira recibiendo cada paquete de datos asociado al fichero que hayan publicado los sensores y reenviará estos datos a aquellos nodos que se hayan suscrito a dicho fichero.
6. Los servidores de Expand ejecutarán una función *callback* que obtendrá los metadatos del *topic* y almacenarán el mensaje en el fichero creado.
7. Por último, cuando el sensor deje de enviar datos, se cerrará el fichero y el servidor se desuscribirá de dicho tema para dejar de recibir datos.

D. POSIX en Expand

Como se ha indicado anteriormente, tanto para la implementación basada en el protocolo TCP/IP como en el protocolo MQTT de Expand, se usará el

estándar POSIX para la creación y manipulación de ficheros entre los sensores y los servidores de Expand.

Para ello, cuando se vaya a desarrollar una aplicación dentro del ámbito de Expand, el usuario podrá utilizar las llamadas típicas del estándar como son `open, read, write` o `close`, entre otras, ya que Expand posee una biblioteca de interceptación o *bypass* que captará las llamadas a POSIX para enviar la información deseada a sus servidores. Por estas razones, en las Figuras 5 y 6 se hace uso de las llamadas `xpn_creat, xpn_write, xpn_close` como traducción a las llamadas a POSIX.

IV. Evaluación

En esta evaluación se analizará el rendimiento y escalabilidad mostrados por Expand cuando se usa una implementación basada en MQTT gracias a Mosquitto frente al rendimiento que muestra el sistema de ficheros si se usa una implementación basada en TCP/IP.

Se ha decidido usar Expand en un entorno HPC junto con las implementaciones basadas en TCP/IP y MQTT para obtener una evaluación inicial. Posteriormente, se evaluará en un entorno real o simulado de IoT.

Elegir un entorno HPC nos proporcionará una cota superior de lo que podría ser un dispositivo IoT aunque sea complejo de conseguir actualmente.

De esta forma, no solo se comprenderá si es una alternativa para sistemas de IoT de diferente tamaño si no también si se le pueden dotar a los propios IoT capacidades de almacenamiento y reducir la computación de los mismos.

Esta evaluación se ha realizado en el supercomputador Picasso de la Universidad de Málaga que posee 126 x SD530 nodos. Cada nodo dispone de 52 cores (Intel Xeon Gold 6230R @ 2.10GHz), 192 GB de RAM y 950 GB en disco para almacenamiento local.

A. Comportamiento de Expand MQTT vs. Expand TCP/IP

A continuación, se quiere estudiar la fiabilidad y la escalabilidad del Sistema de Ficheros Expand a un caso de uso real del ámbito de IoT.

La parte fundamental de esta comparativa consiste en que cada nodo cliente estará formado por un número determinado de procesos que actuarán como sensores los cuales irán enviando paquetes de datos que se escribirán en los servidores de Expand.

Se han llevado a cabo pruebas donde se tienen ocho nodos diferentes, en los que en cuatro de ellos estarán los procesos servidores junto con sus *brokers* de Mosquitto, mientras que los cuatro nodos restantes se comportarán como nodos cliente.

Cada proceso cliente tendrá 50, 100, 200, 250, 400 y 500 sensores donde cada uno de ellos enviará 500 paquetes de 128 bytes. En total, habrá un total de 200, 400, 800, 1000 y 2000 sensores que enviarán datos al broker.

Durante un tiempo determinado se iniciarán los sensores para establecer la conexión con los servidores. Una vez iniciada la conexión, cada sensor enviará los paquetes de información ya sea al *broker* (si se usa la implementación de MQTT) o la implementación TCP/IP.

De esta forma, se podrá determinar cómo se comporta el sistema a medida que aumenta la cantidad de datos que se envían usando la implementación basada en el protocolo de MQTT o de TCP/IP.

Más concretamente, se van a tener dos tipos de prueba diferentes, una donde cada mensaje mandado por cada sensor se envía cada cierto periodo de tiempo (concretamente con 500 microsegundos de *delay*) y otro donde los paquetes se envían uno a continuación de otro (sin *delay*), para probar la sobrecarga de la infraestructura. En el Listado 1 se muestra el pseudocódigo de la aplicación que deben ejecutar los sensores haciendo uso de las llamadas a POSIX mencionadas en líneas anteriores y que se traducirían al ser interceptadas por el *bypass* de Expand.

Listado 1: Aplicación de los sensores

```
#define BUFFER 128
int main(int argc, char *argv[])
{
    char buf [BUFFER];
    int ret = 0;
    int fd1 = creat("nombre_fichero", 0777);
    if ( fd1 < 0 ) return -1;

    do
    {
        recogida_datos_sensor(buf);

        ret = write(fd1, buf, BUFFER);
        if ( ret < 0 )
        {
            close (fd1);
            return -1;
        }
    }
    while ( fin recogida de datos );

    close (fd1);
    return 0;
}
```

Para ambos casos, se mostrará el tiempo medio que ha tardado cada sensor en escribir (enviar al servidor/*broker*) 500 paquetes de datos y, además, el porcentaje de datos que realmente se ha escrito en el servidor, ya sea usando directamente TCP o usando el *broker* como intermediario.

Para entender mejor el proceso de análisis de la tasa de información escrita se ha realizado el siguiente cálculo: Si, por ejemplo, se quiere saber cuánto deben escribir 50 sensores por cliente, se tiene que los servidores deberán recibir un total de: $50\,sensores/cliente \times 4\,clientes \times 128\,bytes/paquete \times 500\,paquetes = 12800000\,bytes$

En la Figura 7 se muestran los tiempos de escritura medio de los sensores cuando se envían los paquetes de datos cada 500 microsegundos. El tiempo de escritura se mide en milisegundo (ms) y se presentan en la Figura usando escala logarítmica (más tiempo es peor).

Como se puede comprobar en la Figura 7, para Expand MQTT, independientemente del QoS que se use, el tiempo medio no supera el segundo, mientras que con TCP va aumentando el tiempo de escritura

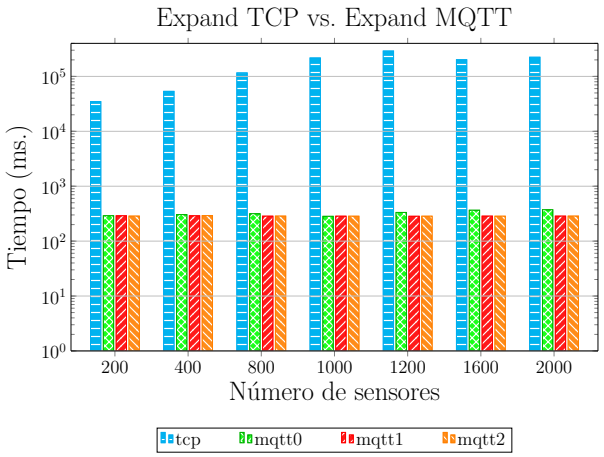


Fig. 7: Tiempos de escritura Expand TCP vs. Expand MQTT **con** *delay*

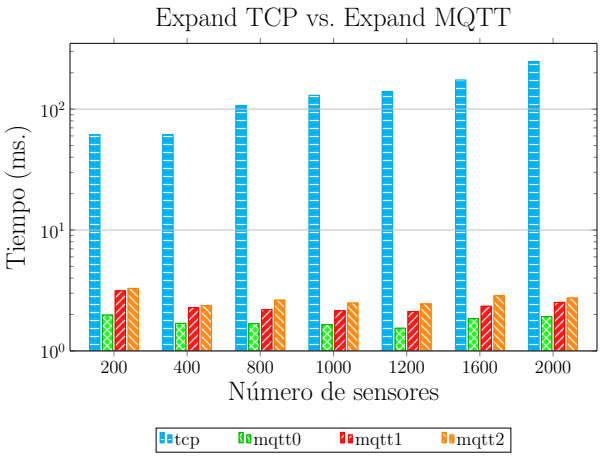


Fig. 8: Tiempos de escritura Expand TCP vs. Expand MQTT **sin** *delay*

conforme más sensores participan en la prueba y llega a tardar más de 200 segundos de media en enviar los paquetes de datos para cada sensor.

Tanto si se usa Expand MQTT con Mosquitto como si se usa Expand TCP/IP directamente, el sistema de ficheros es capaz de almacenar toda la información generada por los sensores tanto con MQTT como con TCP. Asimismo, el *broker* de Mosquitto también puede gestionar el intercambio de mensajes entre los sensores publicadores y los servidores suscriptores.

Por otro lado, si se desea saber qué es lo que ocurre si la información que envían los sensores no tuviese ningún tipo de retardo o *delay*, en la Figura 8 se muestra el tiempo medio que tardan los sensores en enviar los paquetes de información eliminando los 500 microsegundos por envío mencionados anteriormente.

En las pruebas realizadas, Expand TCP con 200 sensores tarda una media de 61 ms y, a medida que aumenta la cantidad, sube también el tiempo de envío hasta llegar a los 250 ms aproximadamente (con 2000 sensores) Esto se debe a que en el proceso de envío de datos de Expand TCP, los sensores deben esperar una respuesta por parte del servidor, mientras que usando Expand MQTT los sensores envían dicha información sin depender de una respuesta de dicho servidor.

Por último, si analizamos también qué porcentaje se ha escrito en los servidores de Expand tras eliminar el retardo de los sensores se observó también que los nodos son capaces de gestionar dicho tráfico y no perder los datos durante el proceso.

V. Conclusiones y trabajos futuros

Tras los datos obtenidos se puede considerar que el diseño e implementación realizado en el sistema de ficheros paralelo y distribuido Expand basado en MQTT para aplicaciones orientadas al Internet de las Cosas puede llegar a ser una alternativa para enviar los datos generados por los sensores en tiempo real. El uso de MQTT (la implementación Mosquitto en concreto) reduce en gran medida el tiempo que tarda en enviar los datos al *broker* que se encarga de reenviar la información a los nodos suscriptores sin comprometer la fiabilidad del sistema.

En los resultados mostrados en la evaluación se puede ver cómo MQTT con las diferentes calidad de servicio que ofrece es capaz de enviar los datos sin aumentar el tiempo y sin pérdida datos.

Como trabajos futuros se quiere estudiar el alcance de este tipo de infraestructuras con el sistema de ficheros Expand aumentando el número de sensores que actúan al mismo tiempo. También se quiere evaluar los resultados obtenidos usando una aplicación de cálculo de potencias para sistema ferroviario.

Además, se realizará una comparativa de rendimiento entre una infraestructura donde Mosquitto forma parte del sistema de ficheros distribuido con otra donde Mosquitto se encuentra fuera de esta infraestructura. Esto significa que, en el primer caso, el cliente del sistema de ficheros publicarán la información al *broker* de Mosquitto y los servidores actuarán de suscriptores del sistema, mientras que el segundo caso los suscriptores del sistema serán los propios clientes del sistema de ficheros a usar. De esta forma, se compara un entorno de dos niveles (el caso analizado en este artículo) con otro donde existen tres niveles que se asemeja más a una infraestructura tradicional de Mosquitto.

Agradecimientos

Este trabajo ha sido parcialmente apoyado por el proyecto del Ministerio de Ciencia e Innovación español *New Data Intensive Computing Methods for High-End and Edge Computing Platforms (DECIDE)* (Ref. PID2019-107858GB-I00) y por el proyecto “Expand - Sistema de almacenamiento de altas prestaciones para entornos HPC y Big Data” (Ref. TED2021-131798B-I00).

Los autores agradecen los recursos informáticos de Picasso y el apoyo técnico prestado por la Universidad de Málaga (IM-2023-1-0012).

Referencias

[1] Anthony JG Hey, Stewart Tansley, Kristin Michele Tolle, et al., *The fourth paradigm: data-intensive scientific discovery*, vol. 1, Microsoft research Redmond, WA, 2009.

[2] Robert B. Ross and Robert Latham, "PVFS - PVFS: a parallel file system," in *SC*. 2006, p. 34, ACM Press.

[3] Karen Rose, Scott Eldridge, and Lyman Chapin, "The internet of things: An overview," *The internet society (ISOC)*, vol. 80, pp. 1–50, 2015.

[4] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, Dusit Niyato, Octavia Dobre, and H Vincent Poor, "6G internet of things: A comprehensive survey," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 359–383, 2021.

[5] Alexander Afanasyev, Neil Tilley, Peter Reiher, and Leonard Kleinrock, "Host-to-host congestion control for TCP," *IEEE Communications surveys & tutorials*, vol. 12, no. 3, pp. 304–342, 2010.

[6] Dan Dinculeană and Xiaochun Cheng, "Vulnerabilities and limitations of MQTT protocol used between IoT devices," *Applied Sciences*, vol. 9, no. 5, pp. 848, 2019.

[7] F. Garcia-Carballeira, A. Calderon, J. Carretero, J. Fernandez, and J. M. Perez, "The design of the expand parallel file syste," *In The International Journal of High Performance Computing Applications*, vol. 17, no. 1, 2003.

[8] Adrián Ruiz Arroyo, "Desarrollo de un conector de un sistema de ficheros paralelo para el sistema HDFS: Evaluación de una prueba de concepto utilizando expand," B.S. thesis, UC3M, 2017.

[9] Dave Evans, "Internet de las cosas," *Cómo la próxima evolución de Internet lo cambia todo. Cisco Internet Bussiness Solutions Group-IBSG*, vol. 11, no. 1, pp. 4–11, 2011.

[10] Rolf H Weber, "Internet of things–new security and privacy challenges," *Computer law & security review*, vol. 26, no. 1, pp. 23–30, 2010.

[11] Jordi Salazar and Santiago Silvestre, "Internet de las cosas," *Techpedia. České vysoké učení technické v Praze Fakulta elektrotechnická*, 2016.

[12] Hannes Tschofenig, Jari Arkko, Dave Thaler, and D McPherson, "Architectural considerations in smart object networking," Tech. Rep., IETF, 2015.

[13] Nitin Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *2017 IEEE International Systems Engineering Symposium (ISSE)*, 2017, pp. 1–7.

[14] Dipa Soni and Ashwin Makwana, "A survey on MQTT: a protocol of internet of things (iot)," in *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*, 2017, vol. 20, pp. 173–177.

[15] Tim Pulver, *Hands-on internet of things with MQTT : build connected IoT devices with Arduino and MQ telemetry transport (MQTT)*, Packt Publishing, 1st edition. edition, 2019.

[16] Valerie Lampkin, Weng Tat Leong, Leonardo Olivera, Sweta Rawat, Nagesh Subrahmanyam, Rong Xiang, Gerald Kallas, Neeraj Krishna, Stefan Fassmann, Martin Keen, et al., *Building smarter planet solutions with mqtt and ibm websphere mq telemetry*, IBM Redbooks, 2012.

[17] Hyun Cheon Hwang, JiSu Park, and Jin Gon Shon, "Design and implementation of a reliable message transmission system based on mqtt protocol in iot," *Wireless Personal Communications*, vol. 91, pp. 1765–1777, 2016.

[18] Andrew Campbell, Geoff Coulson, and David Hutchison, "A quality of service architecture," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 2, pp. 6–27, 1994.

[19] Neven Nikolov, "Research of mqtt, coap, http and xmpp iot communication protocols for embedded systems," in *2020 XXIX international scientific conference electronics (ET)*. IEEE, 2020, pp. 1–4.

[20] Edoardo Longo and Alessandro E.C. Redondi, "Design and implementation of an advanced mqtt broker for distributed pub/sub scenarios," *Computer Networks*, vol. 224, pp. 109601, 2023.

[21] Filippo Antonielli, *Development and comparison of MQTT distributed algorithms for HiveMQ*, pp. 1–79, Politecnico di Milano, 2021.

[22] Shapna Muralidharan and Heedong Ko, "An interplanetary file system (ipfs) based iot framework," in *2019 IEEE international conference on consumer electronics (ICCE)*. IEEE, 2019, pp. 1–2.

[23] Souvik Pal, Vicente Garcia Diaz, and Dac-Nhuong Le, "Iot: Security and privacy paradigm," 2022.

[24] R. Sandberg, D. Golgberg, S. Kleiman, D. Walsh, and B. Lyon, *Design and Implementation of the Sun Network Filesystem*, p. 379–390, Artech House, Inc., USA, 1988.

[25] Libor Dostalek, *Understanding TCP/IP a clear and comprehensive guide to TCP/IP protocols*, From technologies to solutions. Packt Pub., Birmingham, U.K., 1st edition. edition, 2006.

[26] Charles M. Kozierok, *The TCP/IP guide: a comprehensive, illustrated Internet protocols reference*, No Starch Press, 1st ed. edition, 2005.

[27] Dinesh Thangavel, Xiaoping Ma, Alvin Valera, Hwee-Xian Tan, and Colin Keng-Yan Tan, "Performance evaluation of mqtt and coap via a common middleware," in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2014, pp. 1–6.

[28] Eclipse Mosquitto, "An open source mqtt broker," *Eclipse Mosquitto™[cit. 2018-04-23]. Dostupné z: Mosquitto. org*, 2018.

[29] Binildas Christudas, *ActiveMQ*, pp. 861–867, Apress, Berkeley, CA, 2019.

[30] D Shanmugapriya, Akshet Patel, Gautam Srivastava, and Jerry Chun-Wei Lin, "MQTT protocol use cases in the internet of things," in *Big Data Analytics: 9th International Conference, BDA 2021, Virtual Event, December 15-18, 2021, Proceedings 9*. Springer, 2021, pp. 146–162.

[31] Roger A Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, no. 13, pp. 265, 2017.

[32] Biswajeeban Mishra, "Performance evaluation of mqtt broker servers," in *Computational Science and Its Applications–ICCSA 2018: 18th International Conference, Melbourne, VIC, Australia, July 2–5, 2018, Proceedings, Part IV 18*. Springer, 2018, pp. 599–609.

Intercambio seguro y eficiente de datos confidenciales utilizando patrones arquitectónicos de software

Catherine A. Torres-Charles[1], J. L. González-Compeán[2], Miguel Morales-Sandoval[2] y Jesús Carretero[1]

Resumen— **Los sistemas seguros de intercambio de datos son herramientas clave para que las organizaciones eviten incidentes como accesos no autorizados, violaciones de privacidad y confidencialidad cuando los usuarios finales intercambian datos confidenciales en la nube. Este artículo presenta un framework para construir sistemas seguros de intercambio de datos eficientes y personalizables. Utiliza patrones genéricos de arquitectura de software para diseñar tuberías que integren herramientas de compartición, entrega/recuperación y almacenamiento con aplicaciones de seguridad en un único servicio de compartición. También, se incluye un modelo de procesamiento implícito basado en patrones paralelos para reducir la sobrecarga producida por la ejecución de operaciones de compartición segura. La evaluación experimental reveló que el framework era lo suficientemente flexible como para crear múltiples combinaciones de servicios de compartición segura, que mostraban un rendimiento significativamente más eficiente que otras soluciones similares dentro del estado del arte.**

Palabras clave— **Infraestructura en la nube, Arquitectura de software, Sistemas seguros, Política como código, Información sensible, Mejora de la eficiencia**

I. Introducción

La computación en la nube, el almacenamiento y los servicios de entrega de contenidos se están convirtiendo en piedras angulares para que las organizaciones y los usuarios finales participen flujos de trabajo organizativos en línea, en trabajo remoto o el trabajo desde casa [1]. En estos escenarios, los participantes crean estructuras lógicas denominadas *tuberías de datos*, en las que mueven contenidos en cada operación de compartición. Estos contenidos podrían ser sensibles para las organizaciones y para cada participante en el proceso de compartición de datos [2]. Esto da lugar a escenarios críticos en los que los participantes comparten información sensible a través de estas tuberías utilizando servicios externos de computación y almacenamiento en la nube, gestionados por proveedores honestos pero curiosos. En este contexto, incidentes como alteraciones de datos, violaciones de privacidad, confidencialidad y accesos no autorizados, podrían surgir en este tipo de infraestructuras [3]. Para mitigar estos riesgos y contribuir a reducir el impacto de estos incidentes en la experiencia de servicio de los usuarios finales, las organizaciones están adoptando políticas de seguridad de la información implementadas mediante el uso de aplicaciones de seguridad locales [4]. De esta manera, la entrega de información a los socios o su recuperación [5], está asegurada cuando esa información es sensible, tanto para operaciones de datos dentro o fuera de las organizaciones [6], [7], [4].

Una de las soluciones más populares para implementar políticas de seguridad en escenarios de compartición de datos consiste en cifrar los datos antes de enviarlos a entidades externas, lo que implica una operación de descifrado en el lado receptor [7]. Este enfoque de seguridad debe acoplarse cuidadosamente al diseño de sistemas de compartición de datos, construidos comúnmente mediante frameworks (por ejemplo, Jenkins [8] o motores de flujo de trabajo [9]), en los que las aplicaciones de compartición de datos se integran con servicios de computación y almacenamiento en la nube en un único sistema de flujo de trabajo. Sin embargo, en escenarios del mundo real, la creación de servicios de información sensible, seguros y eficientes basados en frameworks sigue siendo una problemática abierta a la que las organizaciones terminan enfrentándose a ciertos obstáculos principales como el diseño de estos sistemas que deben acoplar los datos compartidos, los servicios de la nube y las aplicaciones requeridas [3].

Además, de considerar los distintos tipos de infraestructuras utilizadas, creando esquemas dinámicos en lugar de utilizar las tuberías estáticas de compartición de datos creados por los frameworks tradicionales [8]. Considerando la complejidad computacional asociada a los servicios de seguridad. Por ejemplo, el uso de criptosistemas asimétricos que en términos de consumo de recursos es costoso al realizar el proceso de codificación (cifrado) y decodificación (descifrado), que se traduce en *sobrecarga* [10] que afecta significativamente a la *eficiencia* de los sistemas de compartición y a la experiencia de servicio de los usuarios finales, lo que puede comprometer a la continuidad de los procedimientos de toma de decisiones (por ejemplo, diagnóstico sanitario, gestión territorial, empresarial y de mercado por citar algunos [11]).

En este artículo se presenta un framework para la construcción de sistemas eficientes y seguros de compartición de datos sensibles definidos por patrones arquitectónicos de software. Este, permite crear sistemas de intercambio de datos seguros y personalizables, integrando herramientas de compartición, seguridad, entrega/recuperación y almacenamiento en

[1]Universidad Carlos III de Madrid, Leganes,28911, Spain, e-mail: {cattorre, jcarrete}@inf.uc3m.es

[2]Cinvestav Tamaulipas, Cd, Victoria, 87130, Mexico, e-mail: {joseluis.gonzalez, miguel.morales}@cinvestav.mx

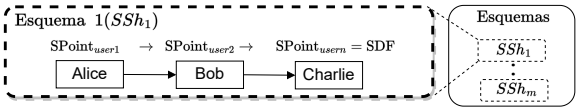


Fig. 1: Definición del esquema de compartición para usuarios finales.

diseños de estructuras de compartición de datos basados en patrones arquitectónicos de software genéricos. Para convertir estas estructuras de compartición en sistemas de compartición dinámicos, se ha creado un modelo de despliegue personalizable que combina infraestructura y políticas como código que incluye un modelo de procesamiento implícito para mejorar la eficiencia de los criptosistemas de seguridad sin alterar la codificación de los mismos. Proponiendo utilizar modelos de paralelismo y distribución, para crear dos esquemas paralelos y distribuidos.

El resto de este artículo se organiza en los siguientes apartados: en la sección 2 se describe la creación de servicios seguros de intercambio de datos eficientes y personalizables, la sección 3 describe los patrones paralelos implícitos, la sección 4 y 5 presentan el entorno experimental y los resultados obtenidos en un estudio de caso para la compartición segura de imágenes satelitales. En la sección 6 se analizan trabajos relacionados. Por último, en la sección 7 se mencionan las conclusiones del artículo.

II. Construcción eficiente y personalizable de servicios seguros de intercambio de datos como código

En esta sección, describimos un método de construcción para crear servicios de compartición de datos seguros, eficientes y personalizados. Este método considera la construcción de servicios de compartición en tres fases. En la primera, la *fase de diseño*, se considera una integración de las aplicaciones utilizadas por los participantes para compartir datos con el almacenamiento y entrega de datos en la nube, así como las aplicaciones de seguridad mediante el uso de patrones de arquitectura de software. La segunda fase, *fase de despliegue*, considera un modelo de despliegue de personalización basado en Infraestructura como código y política como código para convertir los diseños en sistemas seguros de compartición de datos. Por último, en la tercera, *fase de ejecución*, se encuentra la gestión, en tiempo de ejecución, de los patrones de paralelismo implícito para mitigar los efectos de la sobrecarga producida por las operaciones de seguridad sobre el rendimiento del servicio de compartición. La primera fase de este método considera un modelo de esquema de compartición. Este modelo permite a las organizaciones crear, a alto nivel, un diseño de esquemas de compartición (véase la definición de esquemas de compartición en la figura 1). Un *esquema de compartición* es una estructura de metadatos que describe las conexiones lógicas entre cada par de participantes en un proceso de compartición de datos. El resultado de esta fase es un esquema de compartición definido como un grafo acíclico dirigido o DAG para abreviar. Los nodos representan puntos de compartición ($SPoint$) que se asocian a un conjunto de aplicaciones (herramientas de compartición y seguridad) que debe utilizar cada participante en un esquema de compartición, mientras que las aristas representan la secuencia de las operaciones previstas para un conjunto de participantes o *usuarios* para crear un flujo de datos de compartición (véase el esquema 1 de la figura 1). De este modo, las organizaciones pueden crear los m esquemas de compartición que exige la dinámica organizativa:

$$Esquemas = \{SSh_1, SSh_2, ..., SSh_m\}.$$

donde SSh es un DAG de un conjunto de abstracciones genéricas llamadas puntos de compartición ($SPoints$), que representan los nodos en el DAG.

$$SSh_m = SPoint_1 \rightarrow SPoint_2 \rightarrow, ..., \rightarrow Spoint_n$$

Esto permite a las organizaciones asociar un SSh_m a un determinado flujo de trabajo organizativo en el que los $SPoints$ representan roles genéricos como médicos, enfermeras, etc. En los modelos tradicionales de diseño de flujos de trabajo se asignan nombres específicos a las etapas del flujo de trabajo. A diferencia, en este trabajo se produce un modelo $\{Clave, Valor\}$ en el que se puede asociar un SSh_m a un flujo de trabajo organizativo determinado y los $SPoitnts$ a los participantes en dicho flujo de trabajo. En la práctica, cuando un esquema se asocia a un flujo de trabajo de organización determinado y los $SPoitnts$ se asocian a participantes específicos, se producirá un flujo de datos de compartición (SDF):

$$SDF = SPoint_{user_1} \rightarrow SPoint_{user_2} \rightarrow, ..., \rightarrow Spoint_{user_n}$$

Esta asignación de usuarios a $SPoints$ produce, en tiempo de desarrollo, una asociación de un $SPoint$ a aplicaciones que serán utilizadas por los participantes en un esquema de compartición. Hasta este punto, un esquema de compartición basado en un DAG dado ha sido creado por las organizaciones y el esquema ha sido configurado para atender la operación de compartición producida por un conjunto de participantes.

Este comportamiento genérico funciona para las organizaciones, pero no es lo suficientemente flexible para que los participantes atiendan sus preocupaciones específicas sobre la infraestructura que están utilizando en el proceso de compartición (la infraestructura podría ser desconocida en el momento del diseño o podría cambiar en el futuro). Por lo tanto, dentro de la segunda fase se encuentra un método de construcción que incluye a la aplicación de un modelo de despliegue de personalización basado en la infraestructura y la política como código.

Este modelo permite a los usuarios finales, llevar a cabo, en tiempo de desarrollo, un proceso de personalización centrado en la definición de los parámetros de configuración tanto de los servicios de seguridad ($SecServ$) como de los servicios de gestión de datos, entrega, recuperación y almacenamiento (DRS) realizadas en una operación de compartición que se

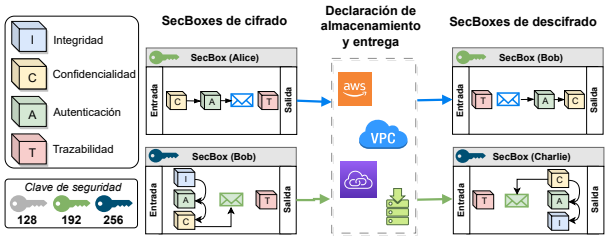


Fig. 2: Definición de seguridad, servicios de entrega y personalización de esquemas de compartición.

emplearán para configurar cada punto de compartición (*SPoint*) incluido en un esquema de compartición (véase un conjunto de bloques de criptosistemas a la izquierda de la Figura 2). La armonización de *SecServ*s con *DRS*s dentro de la abstracción genérica *SPoint* se define a alto nivel, y su realización se consigue convirtiendo esta abstracción en una estructura de metadatos lógica personalizable llamada *SecBox*, que se utilizará durante el proceso de personalización. Véanse ejemplos de personalización de *SecBoxes* en la Figura 2, que se asociaron a un conjunto de participantes en un esquema de compartición definido a alto nivel (Alice, Bob y Charlie). Una *SecBox* se utiliza para recoger la información definida por cada participante para gestionar sus operaciones de compartición, en función de las especificaciones del esquema de compartición. Así, una *SecBox* incluye información sobre requerimientos funcionales (aplicaciones de compartición), requerimientos no funcionales (sistemas de seguridad) y el contexto (detalles de la infraestructura) de cada *SPoint*. Nótese que este modelo asume que los participantes utilizarán sus aplicaciones para procesar sus datos. Esto significa que *SecBoxes* solo supervisará la entrega y recuperación seguras y eficientes de los datos intercambiados entre los *Points* de una *DRS*.

Los descriptores de funcionalidad de *SecBox* recogen información sobre la gestión de los datos entrantes. Básicamente, registra información sobre las herramientas (Ej., sistemas de archivos locales, carpetas en la nube, etc.) habilitadas para recuperar datos (véase *entrada* de *SecBoxes* descrito en la Figura 2) y las herramientas habilitadas para entregar datos cifrados al siguiente *SecBox* en el esquema de compartición (véanse las herramientas de almacenamiento y entrega declaradas por los participantes para conectar cifrar/ descifrar *SecBoxes*). Los descriptores de no-funcionalidad incluyen información sobre un conjunto de *bloques* (aplicaciones de seguridad) requeridos/elegidos por los participantes para abordar sus preocupaciones. Además, las organizaciones definen puntos de compartición genéricos, el framework produce puntos de compartición personalizados (por ejemplo, $SPoint_{user1} = Alice$), que se convierten en *SecBoxes* para que los participantes personalicen sus puntos de compartición (*SPoints*). En este proceso, eligen los servicios de seguridad necesarios para responder a sus preocupaciones específicas para cada punto de compartición (véase cómo Bob elige dos tipos de configuraciones; una para recibir datos de Alice y otra para compartir datos con Charlie). En un proceso final de adición de características de calidad mediante la recopilación de los descriptores de contexto de los *SecBoxes*, se define el tamaño de la clave criptográfica que utilizará cada par de *SecBoxes* en las operaciones de compartición en función del nivel de confianza de los participantes, o bien para la infraestructura en la que se desplegarán sus *SecBoxes*. La figura 2 muestra cómo se eligen las claves para cada punto de compartición y se asignan a los *SecBoxes* de Alice, Bob y Charlie.

Para llevar a cabo este proceso, el framework utiliza un esquema de malla de bloques, que contiene diversos algoritmos para mitigar las necesidades de cada participante. Esta malla se presenta a los usuarios finales y diseñadores como servicios (no como código) para que elijan de esta malla cuando estén configurando sus *SecBoxes* (como se ha descrito previamente al principio de esta sección). La figura 2 muestra cómo se añaden los servicios de seguridad a los *SecBoxes* (Ver Confidencialidad, Autenticación y trazabilidad en el SecBox Alice), que se utilizan en tiempo de despliegue. Dentro de la malla se encuentran criptosistemas tradicionales, como los simétricos (Advanced Encryption Standard, AES) para el cifrado masivo de datos, cifrado basados en emparejamiento bilineales (firmas cortas, aquí denominadas SSign) para los servicios de autenticación e integridad, y el cifrado basado en atributos (CP-ABE) utilizado para aplicar criptográficamente el control de acceso.

Todos estos criptosistemas pueden proporcionar cualquiera de los niveles de seguridad equivalentes a 128, 192 o 256 bits, que cumplen la mayoría de las normas aceptadas (por ejemplo, NIST[12], [13]). Tanto CP-ABE como SSign utilizan un emparejamiento bilineal computable eficiente (asimétrico). Los procedimientos de firma y verificación en SSign se implementan utilizando una instancia de función hash SHA-2. La malla también incluye un bloque de trazabilidad y verificabilidad llamado *TraceChain*, que diseñamos y propusimos específicamente para *SecBoxes* con el fin de registrar cada transacción de intercambio de información en una blockchain privada.

El resultado es la creación automática de contratos inteligentes para cada par de *SecBoxes*, así como el registro de las transacciones realizadas por cada *SecBox*. Este algoritmo permite a los usuarios finales rastrear contenidos gestionados por *SecBoxes* y realizar retos para verificar un determinado contenido compartido o intercambiado a través de una cadena de *SecBoxes*. Además, se integró un sobre digital seguro (*SDE*) [14], [7] que es una representación virtual de un paquete certificado tradicional utilizado por organizaciones y personas, en escenarios reales, para realizar un intercambio seguro de documentación con otras personas u organizaciones. En esta analogía, un *SDE* representa un paquete certificado que contiene la documentación sensible y cada *SecBox* representa un emisor o receptor de documentación certificada. Esto significa que un *SDE* también es utilizado por

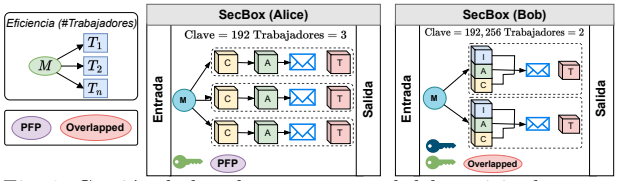


Fig. 3: Gestión de la sobrecarga general del servicio de seguridad

SecBoxes como unidad de transporte para el envío de datos. Para preservar las ventajas del diseño genérico del modelo de esquema de compartición, así como la flexibilidad de personalización del modelo de *SecBoxes*, la gestión logística de las operaciones (entrega y recuperación de *SDE*s) se coordina, en tiempo de ejecución, mediante el uso de un esquema de publicación-suscripción basado en operaciones Pub y Sub, que permiten a los usuarios finales utilizar sus *SecBoxes* para participar en diferentes esquemas de compartición. Una operación de compartición comienza cuando un participante publica un catálogo y asigna derechos a los roles de usuario (participantes válidos situados en la secuencia autorizada definida en un esquema de compartición).

Para ello se utilizan operadores lógicos (y/o). Por ejemplo, el Catálogo *MammografiasDATEx* es creado por un participante (con rol de radiólogo) y publicado para ser suscrito por participantes asociados a roles que cumplan la siguiente política: *médico o oncólogo y paciente*. Esta política es aplicada de forma automática y transparente por los *SecBoxes* utilizados por los participantes implicados en este esquema de compartición. En la fase tres, se ha definido un modelo de diseño de patrones paralelos implícitos para ayudar a los participantes con la gestión de la sobrecarga producida por sus decisiones de política de seguridad (ver Figura 3), y poder ejecutar en paralelo los bloques (servicios de seguridad) elegidos. Este modelo de patrón paralelo de procesamiento implícito produce una codificación que se incorpora al *SecBoxes* como bloque reservado.

III. Patrones paralelos implícitos

Para este trabajo se han construido dos bloques, que se añadieron a la malla, basados en dos nuevos patrones paralelos que diseñamos para los *SecBoxes*: *PFP*, un patrón paralelo implícito Pipe&Filter para procesar un gran número de archivos pequeños, y una versión *Overlapped* para asegurar un gran número de archivos grandes. A continuación se definen los patrones implementados.

A. PFP: Bloque de tuberías y filtros paralelos implícitos

El bloque *PFP* combina patrones secuenciales y paralelos conectados por un DAG. En primer lugar, los bloques de seguridad (SB) se organizan, dentro de la SecBox, en forma de patrón Pipe&Filter ($P\&F$). En este patrón, los *filtros* representan bloques de seguridad, y las *tuberías* representan bloques de entrega/recuperación (DRS). Este patrón produce la ejecución secuencial de todos los bloques de

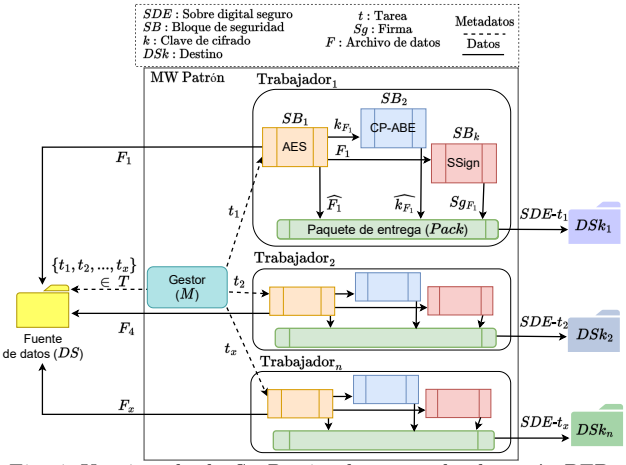


Fig. 4: Un ejemplo de *SecBox* implementando el patrón PFP, Parallel Pipe&Filter.

seguridad. Todos los *filtros* producen un conjunto de activos criptográficos que se gestionan en un patrón compartido en memoria y luego se empaquetan en un sobre digital seguro (SDE). Una representación conceptual de un sistema de seguridad creado mediante el uso de este patrón se presenta en la Figura 4. En este ejemplo, los *filtros* están asociados a bloques como AES (SB_1), CP-ABE (SB_2) y SSign (SB_k), mientras que las aristas son bloques DRS que aplican la dirección del intercambio de datos y la secuencia de ejecución de los bloques de seguridad.

En este ejemplo, el bloque AES (SB_1) extrae el archivo F_1 de DS, crea la clave de cifrado k_{F_1}, que se envía al bloque CP-ABE (SB_2) para ser cifrado y producir $\widehat{k_{F_1}}$. Al mismo tiempo, AES envía F_1 a SSign (SB_k), que produce el hash para realizar la firma de F_1 (Sg_{F_1}). Todos los SB depositan los activos criptográficos en el bloque *Pack (paquete de entrega)*, ss decir, AES, CP-ABE y SSign entregan a *Pack* los $\widehat{F_1}$, $\widehat{k_{F_1}}$ y Sg_{F1} respectivamente. El *Pack* produce un sobre digital seguro (SDE-t_1) concatenando estos activos y enviando este objeto criptográfico a un *Destino* (DSk_1 en este ejemplo). Se genera un SDE para cada fichero procesado por este patrón ($Archivo \in DS$).

En tiempo de ejecución, este patrón termina asegurando diferentes archivos simultáneamente debido a la implementación de la tubería. Por ejemplo, cuando SB_k está procesando el primer fichero (F_1), SB_2 y SB_1 están, ya procesando los ficheros F_4 y F_x respectivamente. Sin embargo, se produce una saturación cuando cada bloque de seguridad debe esperar los resultados del anterior (especialmente los últimos). Esto produce cuellos de botella y tiempos de espera para los ficheros entrantes, lo que acaba produciendo sobrecarga. El PFP también incluye un patrón paralelo Gestor/Trabajador (MWPatrón) para reducir la sobrecarga de los bloques de seguridad y DRS. En este patrón, el gestor es un bloque reservado, mientras que los trabajadores son clones del patrón Pipe&Filter descrito. En el ejemplo mostrado en la Figura 4, el gestor M distribuye los archivos a los trabajadores (ver traba-

jadores $\{trabajador_1, trabajador_2, ..., trabajador_n\}$) en forma de tareas (ver $t_1, t_2, ..., t_x \in T$ en la Figura 4).

El patrón PFP se utiliza para encadenar SBs a través de rutas de E/S, que están interconectadas a través de un SB_{pack}. El gestor se encarga de crear, coordinar y establecer el control sobre un número determinado de trabajadores. Esta asignación de tareas se realiza mediante el uso de balanceo de carga para aumentar el número de procesamiento de tareas ejecutadas en paralelo. Se utiliza una estrategia para distribuir *tareas* a los *trabajadores*. La estrategia de balanceo de carga elige trabajadores (dos en esta estrategia) de una lista de ellos, y elige aleatoriamente el que tiene la menor carga de trabajo para enviar el *archivo*. Esta estrategia de equilibrio de carga crea una distribución uniforme de tareas para los trabajadores a corto plazo, lo que resuelve el problema de tener una distribución uniforme sólo a largo plazo cuando se utiliza la selección pseudoaleatoria de trabajadores. Este balanceo de carga también asimila la heterogeneidad de rendimiento de los trabajadores, lo que resulta clave para evitar cuellos de botella en tiempo de ejecución cuando se gestionan cargas de trabajo heterogéneas (por ejemplo, diferentes tamaños de archivos y/o diferentes niveles de seguridad).

El resultado de los trabajadores puede ser enviado a un único destino (DSk) o a un conjunto de destinos, como el ejemplo mostrado en la Figura 4 ($DSk_1, DSk_2, \ldots, DSk_n$). En este patrón, los trabajadores son idénticos entre sí, pero mutuamente independientes. Como resultado, cada trabajador procesa una parte de todo el conjunto de datos ($t_x \in T$) de forma paralela. Esto permite al usuario final seleccionar el número de trabajadores (n) y una lista de servicios de seguridad ($SecServ$) que se ejecutarán dentro de una *SecBox* en función de la disponibilidad de recursos informáticos. Además de la distribución de una carga balanceada y del paralelismo, otra ventaja de este mecanismo es que se pueden añadir/eliminar fácilmente nuevos trabajadores sin realizar grandes cambios en la codificación de las aplicaciones de seguridad ejecutadas en los *SecBoxes*.

B. *Overlapped: Un patrón paralelo para la gestión de tareas pesadas de seguridad*

En el patrón overlapped, modificamos la disposición de los bloques de seguridad dentro de una *SecBox* para crear una ejecución superpuesta de los bloques de seguridad. La figura 5 muestra un ejemplo de bloques de seguridad organizados como un patrón overlapped en una *SecBox*.

Este método de paralelismo también combina dos patrones: el primero organiza secuencialmente aquellos bloques de seguridad que tienen dependencias entre sí. La figura 5 muestra cómo DAG_1 incluye SBs AES y CP-ABE que tienen una dependencia funcional con AES-KG, que en PFP se gestionaba mediante una tubería (AES-KG → AES).

El segundo DAG se diseñó para ejecutar bloques

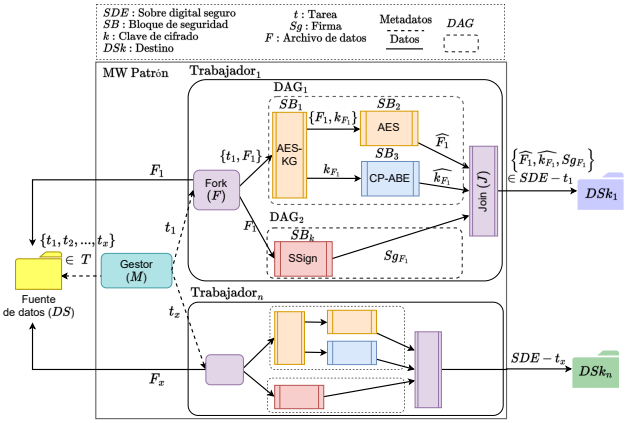


Fig. 5: Un ejemplo de *SecBox* implementando el patrón overlapped.

de seguridad sin dependencias (véase DAG_2 incluyendo un SB *SSign* en la figura 5). El segundo paso es gestionar estos patrones como un único proceso, que se clona para ser gestionado como un *trabajador* por un patrón *fork/join*. Esto significa que el *fork/join* se puede clonar de la misma forma que se realiza en el patrón PFP (ver trabajadores $\{Trabajador_1, \ldots, Trabajador_n\}$ en la Figura 5). Esto mejora el rendimiento global cuando el patrón procesa ficheros de gran tamaño, ya que los SBs evitan la espera de SBs sin dependencias, lo que incluso producirá una mejora adicional cuando se procesen dichos ficheros utilizando claves de gran tamaño. En el ejemplo mostrado en la Figura 5, se puede observar que la creación de claves AES-KG $\in DAG_1$ puede ejecutarse en paralelo con $SSign \in DAG_2$, mientras que CP-ABE y AES también pueden ejecutarse en paralelo, lo cual no era factible en PFP debido a las dependencias de las SBs AES y CP-ABE con AES-KG.

En overlapped, el *Fork SB* divide los bloques de seguridad en $DAGs$ dependientes/independientes, mientras que el *join SB* no sólo coordina la entrega de los resultados de los bloques de seguridad (DAG_1 y DAG_2), sino que también ejecuta un *paquete de entrega* ($Pack$) utilizado en *PFP*. En este punto, se pueden ejecutar otros bloques de seguridad después de producir la firma del contenido entrante t_x (por ejemplo, BCTracing para registrar la transformación de t_x en la blockchain).

El último paso en este método de paralelismo es crear un patrón gestor/trabajador para enviar tareas a diferentes trabajadores para asegurar múltiples archivos en paralelo y crear una *SecBox* con este patrón. La figura 5 también muestra cómo el M del patrón overlapped se encarga de equilibrar la carga distribuida a los clones del patrón fork/join. Es decir, se espera que el patrón overlapped produzca tres tipos de paralelismo dentro de una *SecBox*. El primero se produce cuando los bloques de seguridad se ejecutan en tuberías. El segundo se produce cuando los $DAGs$ se ejecutan en paralelo dentro de un $Trabajador$ mediante el patrón fork/join. El último se produce por el balanceo de las cargas de trabajo (archivos y tareas) realizado por el gestor del patrón

Tipo	SO	CPU	Cores	RAM	HD
Nube	CentOS 7	Intel Xeon CPU E5-2640 2.50GHz	24	64	3.5T
Fog	CentOS 7	Intel Xeon CPU E5645 2.40GHz	6	12	2T
Cliente1,2	CentOS 7	Intel Xeon CPU E5645 2.40GHz	6	12	1.5T
Cliente 3	CentOS 7	Intel Xeon CPU E5675 3.07GHz	6	24	1T
Nube	CentOS 7	Intel Xeon CPU E5-2650 2.60GHz	16	64	3T
Nube	CentOS 7	Intel Xeon CPU E5-2650 2.20GHz	24	251	3T

Tabla I: Infraestructura para experimentos.

(ejecutado por *MW*Patrón). Este tipo de patrón es muy adecuado para garantizar contenidos sensibles de gran tamaño (por ejemplo, imágenes sanitarias, imágenes por satélite, repositorios de big data, etc.) que consideren altos niveles de seguridad en las operaciones de compartición, intercambio (entrega y recuperación) y trazabilidad necesaria en los procesos de toma de decisiones comerciales y críticas. Es importante señalar que estos patrones (PFP y Overlapped) son lo suficientemente genéricos como para que los usuarios finales puedan añadir tantos bloques de seguridad como necesiten para resolver sus problemas de seguridad en escenarios de intercambio de información.

IV. Evaluación experimental

Para validar el método utilizado para crear el framework propuesto en este trabajo, desarrollamos un prototipo para construir un conjunto de servicios de compartición de información basados en *SecBoxes*. Se implementó en una nube desplegada utilizando contenedores [15] en una infraestructura informática descrita en la Tabla I. Realizamos una comparación directa del rendimiento de estos servicios compartidos con los servicios creados mediante dos soluciones de última generación.

La primera comparación se realizó con un servicio tradicional de tuberías multiseguridad [7]. Se centró en la evaluación de los parámetros críticos de los patrones paralelos propuestos en este trabajo (PFP y Overlapped) y su objetivo era determinar la eficiencia de dichos patrones para reducir la sobrecarga en las operaciones de compartición.

La segunda se centró en el rendimiento de los servicios de compartición construidos con nuestro prototipo y los servicios de tuberías construidos mediante el uso de Jenkins [16], un popular y ampliamente utilizado software de última generación que crea tuberías integrando múltiples aplicaciones en un único servicio. Jenkins también está desarrollado en Java y basado en contenedores virtuales distribuidos en una nube. Además, ejecuta las aplicaciones de una tubería en paralelo utilizando el patrón *Gestor/Trabajador*. Estas características nos permitieron realizar una comparación justa entre nuestro framework y Jenkins. El objetivo de esta evaluación es observar el impacto de los bloques reservados implícitos en el rendimiento de las tuberías creadas utilizando frameworks tradicionales y los construidos utilizando *SecBoxes*. Realizamos estos dos estudios comparativos de rendimiento basándonos en el intercambio (entrega y recuperación) y rastreo de imágenes satelitales entre un conjunto de clientes (usuarios finales).

V. Estudio de caso: imágenes satelitales

En este estudio de caso se analiza el intercambio, la entrega y la recuperación de imágenes por satélite. El repositorio incluye 1620 archivos, entre metadatos e imágenes satelitales de la Tierra captadas por la misión SMOS de la Agencia Espacial Europea (ESA). El tamaño de este repositorio es de 59 GB, con un tamaño medio de 37 MB y una desviación estándar de 95 MB.

A. Experimentos, configuraciones y métricas

Se procesaron datos y metadatos mediante servicios de intercambio de información creados con nuestro framework, y se comparó el rendimiento con tuberías secuenciales de servicios de seguridad.

Nuestros experimentos evaluaron cada nivel de seguridad recomendado dado por $\lambda \in \{128, 192, 256\}$. Para explorar el impacto en el rendimiento de la solución de seguridad, se utilizaron y evaluaron en estos experimentos diferentes números de trabajadores ($Trabajadores \in \{1, 2, 4, 6, 8, 10\}$). Cada configuración se crea utilizando el framework y luego se lanza en la nube como un servicio seguro de intercambio de información, que se utiliza para crear un esquema Pub/Sub para que dos usuarios finales intercambien datos. El primero es un cliente con el rol de productor, que comparte datos/metadatos de este repositorio. Esta tarea Pub da como resultado el aseguramiento de los datos y la entrega de cada archivo a la nube. El segundo cliente asume el rol de consumidor para recuperar lo compartido mediante el uso de tareas Sub.

El rendimiento producido por cada servicio se ha medido, capturado y evaluado en dos métricas: I) *El tiempo de servicio* mide el tiempo empleado por *SecBoxes* en la ejecución de todos los bloques de seguridad para cada configuración de cada patrón paralelo (PFP y Overlapped); II)*El tiempo de respuesta* se calcula incluyendo la suma de los tiempos de servicio de cada *SecBox* considerada en el servicio de compartición de información más los costes de intercambio de datos a través de la tubería *SecBox*.

Cada experimento se realizó 31 veces para calcular el valor medio y la mediana de las métricas evaluadas, que se utilizó para obtener la evaluación del rendimiento.

B. Análisis de rendimiento de patrones paralelos

Los experimentos de este caso de estudio se definieron para identificar tres características clave de rendimiento, que se intuían a través de los principios de diseño de los patrones paralelos utilizados en *SecBoxes*. La primera percepción es mostrar la eficiencia de los patrones paralelos en comparación con una organización tradicional en tuberías de servicios de seguridad. La segunda es observar al aumentar el número de trabajadores, la mejora del rendimiento

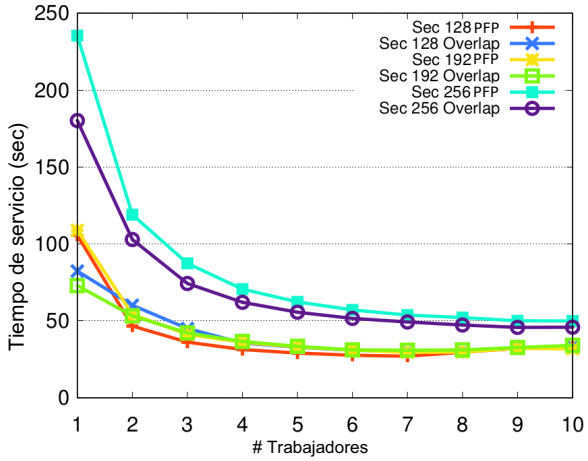


Fig. 6: Tiempo de servicio para el tratamiento de imágenes satelitales.

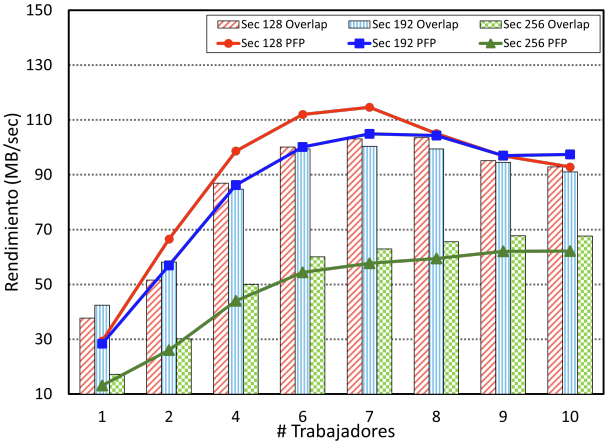


Fig. 7: Comparación de rendimiento entre patrones PFP y Overlapped.

alcanzará un nivel estable debido a la sobrecarga de gestión y la disponibilidad de recursos. La tercera es que el patrón Overlapped es más adecuado para gestionar tareas de procesamiento pesadas que los patrones PFP.

La figura 6 muestra los tiempos de servicio (eje vertical) producidos por los patrones *PFP* y *Overlapped* (véanse las figuras 4, 5) al procesar todo el contenido de las imágenes satelitales, estableciendo distinto número de trabajadores (eje horizontal). Como era de esperar, cuanto mayor es el número de trabajadores, menor es el tiempo de servicio. También se observó, que cuanto más pesados son los procesos (por ejemplo, nivel de seguridad de 256 bits), mayor es la sobrecarga en el tiempo de servicio. Tanto PFP como los patrones Overlapped produjeron una aceleración en torno al *80 %* en comparación con la versión de tubería tradicional ($Trabajadores = 1$) en los experimentos realizados en este caso de estudio.

Nuestra segunda característica de rendimiento (estabilización del rendimiento) se observó en la mejora del tiempo de servicio, analizando el comportamiento de los patrones paralelos al aumentar el número de trabajadores en *SecBoxes*. Se observó que cuanto más pesados eran los procesos, mayor era el número de trabajadores necesarios para conseguir dicha estabilización. Por ejemplo, 10 trabajadores produjeron mejor rendimiento para procesar imágenes con un nivel de seguridad de 256 bits, mientras que 5 trabajadores fueron adecuados para procesar niveles de seguridad de 192 y 128 bits.

En estos experimentos, también se observó que la mejora del rendimiento de estos patrones no sólo depende del nivel de seguridad, sino también del número de núcleos físicos asignados a los contenedores virtuales (núcleos virtuales incluidos). La figura 6 también muestra que un equipo que incluya al menos cuatro núcleos es suficiente para mejorar el rendimiento del servicio de seguridad de forma significativa para todos los experimentos realizados. También, para verificar nuestra tercera percepción sobre la gestión de tareas de procesamiento pesado, comparamos los resultados de ambos patrones para cada nivel de seguridad.

La Figura 7 muestra, en el eje vertical, el rendimiento (MB/seg) al aplicar los servicios de seguridad a las imágenes satelitales tanto para el patrón PFP como para el patrón Overlapped al variar el número de trabajadores (eje horizontal) y para los tres niveles de seguridad recomendados (para el procesamiento de los niveles de seguridad de 128, 192 y 256 bits).

En general, ambos patrones producen un mejor rendimiento que las tuberías tradicionales debido a que el número de trabajadores aumenta los cálculos concurrentes, lo que también mejora los tiempos de servicio. En el caso de 128 bits, el patrón PFP rinde mejor que el Overlapped, que alcanza un rendimiento máximo cercano a los 120 Mbps con 7 trabajadores. Para el caso del nivel de seguridad de 192 bits, el rendimiento de ambos patrones paralelos es prácticamente similar, alcanzando de nuevo el rendimiento máximo en torno a los 100 Mbps con 7 trabajadores. Al aumentar el nivel de seguridad (el tamaño de la clave), también se observa una reducción del rendimiento del patrón debido al aumento de la complejidad de los algoritmos de seguridad para gestionar claves de gran tamaño (es decir, CP-ABE y SSign). El tamaño del archivo afecta a las operaciones aritméticas relacionadas con los cálculos de emparejamientos, mientras que el tamaño de los archivos y el conjunto de datos afectan a AES y SHA. El rendimiento máximo alcanzado por los patrones que utilizan claves de gran tamaño (por ejemplo, 256 bits) se sitúa en torno a los 70 Mbps. Este cenit se obtiene cuando se utilizan 9 trabajadores.

El patrón Overlapped consigue mejores resultados que el patrón PFP, mientras que este rendimiento sigue una relación inversa cuando se utilizan niveles de seguridad de 128 y 192 bits. La razón es la misma, las operaciones CP-ABE y SSign se vuelven más complejas al aumentar el tamaño de los grupos y las operaciones aritméticas asociadas. El patrón Overlapped rinde mejor para el procesamiento pesado, lo que confirma nuestra percepción inicial. También observamos un punto de inflexión en la mejora del rendimiento producida por los patrones en todos los ex-

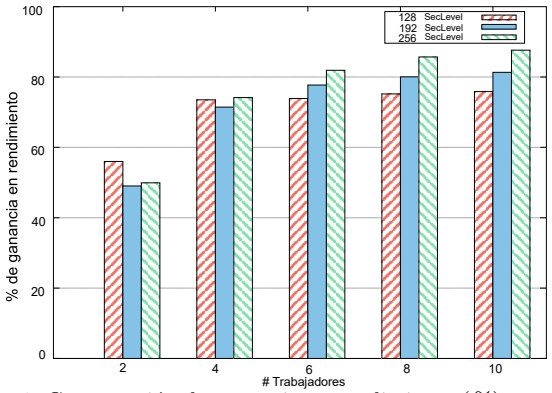


Fig. 8: Comparación de ganancia en rendimiento (%) entre el patrón PFP y una tubería tradicional

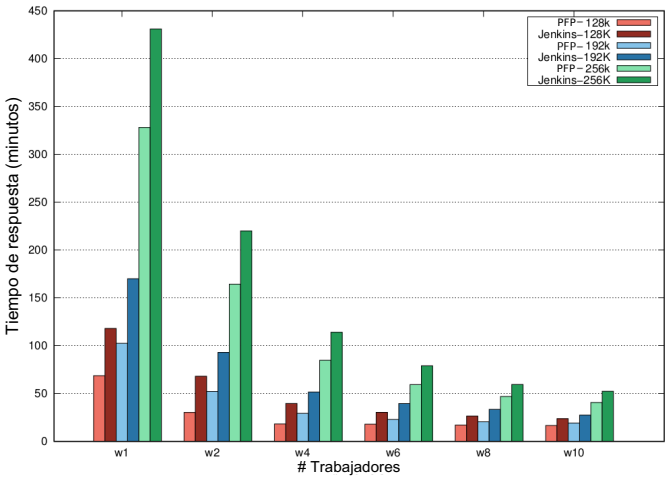


Fig. 9: Comparación del rendimiento de tuberías de servicios de seguridad construidos por Jenkins y *SecBoxes* basados en el patrón PFP.

perimentos. Este punto de inflexión surge cuando los trabajadores comienzan a causar una degradación en el rendimiento, ya que se utilizan los máximos recursos físicos (véase el rendimiento para 8 trabajadores procesando niveles de seguridad de 128 y 192 bits). Aunque esto no se observó en el patrón Overlapped utilizando 256 bits, cabe esperar que el comportamiento surja al aumentar el número de trabajadores.

A partir de ese momento, añadir más trabajadores producirá colas en los procesos de seguridad, disminuyendo así la mejora del rendimiento. En la figura 8 se muestra la ganancia en rendimiento (en el eje vertical) que se incrementa al aumentar el número de trabajadores en el patrón (eje horizontal). Como se puede observar, se consigue una mejora en el rendimiento ((MB/seg)) entre *50 %* y *85 %* al comparar el rendimiento de nuestros patrones en framework con una implementación pipeline tradicional [7] (el caso en que los trabajadores son sólo uno).

C. *Comparación de rendimiento con tuberías construidas en Jenkins.*

Esta sección presenta una comparación del rendimiento de los pipelines de seguridad creados por nuestro framework con los pipelines creados utilizando *Jenkins*. Esta comparación se realizó en condiciones similares: lenguaje de programación (Java), hardware (red privada de servidores), nube (OpenStack y contenedores), datos (imágenes satelitales), algoritmos de seguridad y patrones paralelos (es decir, patrones PFP).

Los contenedores virtuales utilizados por Jenkins y nuestro framework se desplegaron en tres clases de dispositivos: en la primera hay dispositivos para la gestión de los datos *fuente* en Fog (niebla), es decir, los datos que deben ser compartidos, entregados y / o recuperados por *SecBoxes*. La segunda clase son los dispositivos llamados *clientes*, que son ejecutados por los usuarios finales para recuperar / entregar contenidos compartidos. La tercera clase comprende los ordenadores que se utilizan para crear una nube en la que se desplegaron tanto las tuberías *SecBoxes* construidas con nuestro framework como las tuberías de contenedores virtuales construidas por Jenkins.

Para comparar la eficacia de los patrones de nuestro framework, comparamos el patrón PFP con una tubería construida en Jenkins utilizando un patrón Gestor/Trabajador paralelo, ya que este patrón está habilitado para ser utilizado en Jenkins. Para realizar una comparación justa, ambas tuberías incluyen la misma organización de seguridad y entrega/recuperación, así como bloques de almacenamiento para producir el mismo comportamiento de procesamiento en las tuberías. Esta comparación nos permitió medir el impacto de los esquemas de gestión de nuestro framework como el balanceo de carga, la gestión implícita de patrones y la coordinación de intercambio de datos de E/S, que son diferentes de los utilizados por Jenkins. Los experimentos se realizaron variando el número de trabajadores para ambas soluciones (eje horizontal) y produciendo una configuración para cada nivel de seguridad considerado en esta evaluación (*128, 192 y 256*).

La figura 9 muestra esta comparación del tiempo de servicio de *pipelines* de seguridad construidos mediante Jenkins con el patrón PFP de una *SecBox*. El tiempo de servicio producido por el patrón PFP es, en todos los casos, notablemente mejor que los tiempos de servicio producidos por los *pipelines* construidos usando Jenkins. Por ejemplo, el patrón PFP produjo una ganancia de rendimiento superior al 50 % en comparación con el pipeline de Jenkins para el nivel de seguridad de 128 bits cuando se utilizan sólo dos trabajadores. Esta ganancia se reduce al 30 % al aumentar el número de trabajadores a 10.

Esto demuestra una vez más que, incluso con un ordenador no muy potente que incluye sólo cuatro núcleos, SecBoxes rinde mejor que una solución tradicional similar utilizada por los desarrolladores para crear pipelines, que se desplegaron y evaluaron de forma muy similar en cuanto a organización y ejecución en paralelo. En el caso del nivel de seguridad de 192 bits, la ganancia máxima de los patrones PFP se obtiene cuando se utilizan dos o incluso cuatro trabajadores. La ganancia mínima de rendimiento se obtiene cuando se utiliza un nivel de seguridad de 256 bits, en torno al 20 %–30 % para las distintas configuraciones. Esto concuerda con resultados anteriores

en los que, para tareas de procesamiento pesado (por ejemplo, procesamiento de contenidos con un nivel de seguridad de 256 bits), los patrones Overlapped ofrecían un mejor rendimiento que los PFP en este tipo de escenarios.

La experimentación reveló que el framework de patrones paralelos no sólo mejora el rendimiento de las aplicaciones de seguridad para el procesamiento de contenidos sensibles, sino que además este framework proporciona un mejor rendimiento que las soluciones existentes y tradicionales para la construcción de pipelines (Jenkins en esta evaluación experimental). Este caso de estudio también demostró la viabilidad de utilizar nuestro framework para construir *SecBoxes* para proporcionar, en escenarios reales, servicios seguros y eficientes de *intercambio de información*. Esta evaluación también proporciona información a los participantes en los sistemas de intercambio sobre el ajuste fino de sus *SecBoxes* en función de sus recursos computacionales, los costes relacionados con los niveles de seguridad y la carga de trabajo de los archivos que se comparten con otros participantes.

VI. Trabajos relacionados

En este trabajo, exploramos una nueva estrategia para crear sistemas de intercambio de datos confidenciales, eficientes y seguros mediante el uso de paralelismo basado en tareas y procesos, y habilitando la seguridad basada en construcciones ABE, en aplicaciones de *intercambio de información*, firmas digitales y SHA para integridad, AES para privacidad y blockchain para trazabilidad, todo ello particularmente bajo el concepto de sobre digital, para hacerlos aptos para un uso práctico.

Las propuestas disponibles en el estado del arte se centran principalmente en un determinado servicio de seguridad y no consideran la sobrecarga que producen las operaciones seguras que afectan la experiencia del servicio de los usuarios finales y los procedimientos de toma de decisiones [17]. Para DET-ABE [18] y AES4Sec [7] el principal problema en criptografía han sido los elevados costos computacionales que se traducen en tiempos de respuesta más altos observados por los usuarios finales, lo que resulta en un problema de eficiencia para que las organizaciones apliquen estas técnicas en escenarios prácticos.

J. L. González et al. propuso SecFilter [18] un motor de seguridad, basado en sobres digitales de cifrado simétrico, cifrado basado en atributos y firmas digitales para garantizar los servicios de seguridad de confidencialidad, integridad y autenticación para proteger los archivos antes de enviarlos a un almacenamiento en la nube. Laster et al. [8] propuso un generador de flujos de trabajo continuos, que permite la integración de tuberías y contenedores virtuales. También se basa en un modelo de paralelismo implícito, que permite una comparación justa y directa con los patrones de paralelismo propuestos en este documento. La comparación se realizó bajo las mismas condiciones, hardware, mecanismos de aseguramiento y datos.

Sin embargo, no existe un estudio sobre la definición de datos eficientes y seguros mediante patrones de arquitectura de software y políticas como código para abordar soluciones editables o personalizables, considerando la seguridad múltiple y el paralelismo basado en tareas y procesos. Nuestro trabajo aborda esta brecha particular con respecto a los diseños de arquitectura de software para sistemas de software de seguridad distribuidos. Hasta donde sabemos, la personalización como modelo de implementación que combina infraestructura y política como código y convierte los diseños de canalización en servicios en diferentes infraestructuras no ha sido suficientemente explorado, que es el alcance principal del framework presentado en este documento.

VII. Conclusiones

Este artículo presenta un framework para que organizaciones y usuarios finales construyan sistemas seguros de intercambio de datos eficientes y personalizables. El framework permite a las organizaciones diseñar tuberías que integran aplicaciones de compartición, entrega/recuperación y almacenamiento con herramientas de seguridad mediante el uso de patrones genéricos de arquitectura de software, por medio de un modelo de despliegue personalizable que utiliza infraestructura como código para convertir los diseños de tuberías en servicios desplegables, dentro de infraestructuras definidas por los participantes en los procesos de compartición de datos, y también establecen políticas de seguridad heterogéneas para gobernar los flujos de datos compartidos. Además del modelo de procesamiento implícito, basado en patrones paralelos para mitigar la sobrecarga producida por la ejecución de operaciones de compartición segura. La evaluación experimental reveló que el framework es lo suficientemente flexible como para crear múltiples combinaciones de servicios de compartición segura, que muestran un rendimiento significativamente más eficiente que otras soluciones similares del estado del arte. En el estudio de caso, los *SecBoxes* produjeron una ganancia de rendimiento de alrededor del 80 % en comparación con las tuberías de seguridad tradicionales. También reveló mejoras de rendimiento entre el 40 % y el 85 % para diferentes tipos de infraestructuras, lo cual es clave para implementar servicios de intercambio de información basados en la nube en escenarios del mundo real. En este estudio se obtuvo una ganancia de rendimiento de hasta el 50 % en una comparación directa con el constructor de tuberías paralelas de última generación Jenkins [8]. Las comparaciones cuantitativas y cualitativas entre los *SecBoxes* y las soluciones actuales del estado del arte revelaron la flexibilidad y eficiencia de los *SecBoxes* para gestionar múltiples escenarios reales de compartición, intercambio y trazabilidad de información confidencial.

Agradecimientos

El presente trabajo ha sido financiado parcialmente por el proyecto del Ministerio de Ciencia e Innovación "New Data Intensive Computing Methods for High-End and Edge Computing Platforms (DECIDE)" con número de referencia PID2019-107858GB-I00 y el proyecto "Plataforma tecnológica para la gestión, aseguramiento, intercambio y preservación de grandes volúmenes de datos en salud y construcción de un repositorio nacional de servicios de análisis de datos de salud" con número 41756 por PRONACES-CONACYT.

Referencias

[1] Erik Brynjolfsson, John J Horton, Adam Ozimek, Daniel Rock, Garima Sharma, and Hong-Yi TuYe, "Covid-19 and remote work: An early look at us data," Tech. Rep., National Bureau of Economic Research, 2020.

[2] Y. Zhang, C. Xu, X. Lin, and X. S. Shen, "Blockchain-based public integrity verification for cloud storage against procrastinating auditors," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2019.

[3] Nabeil Eltayieb, Rashad Elhabob, Alzubair Hassan, and Fagen Li, "A blockchain-based attribute-based signcryption scheme to secure data sharing in the cloud," *Journal of Systems Architecture*, vol. 102, pp. 101653, Jan. 2020.

[4] Merrill Warkentin and Craig Orgeron, "Using the security triad to assess blockchain technology in public sector applications," *International Journal of Information Management*, vol. 52, pp. 102090, 2020.

[5] Chenlin Huang, Wei Chen, Lu Yuan, Yan Ding, Songlei Jian, Yusong Tan, Hua Chen, and Dan Chen, "Toward security as a service: A trusted cloud service architecture with policy customization," *Journal of Parallel and Distributed Computing*, vol. 149, pp. 76–88, 2021.

[6] Lingfang Zeng, Bharadwaj Veeravalli, and Xiaorong Li, "Saba: A security-aware and budget-aware workflow scheduling strategy in clouds," *Journal of Parallel and Distributed Computing*, vol. 75, pp. 141–151, 2015.

[7] Miguel Morales-Sandoval, Jose Luis Gonzalez-Compean, Arturo Diaz-Perez, and Victor J. Sosa-Sosa, "A pairing-based cryptographic approach for data security in the cloud," *Int. J. Inf. Secur.*, vol. 17, no. 4, pp. 441–461, Aug. 2018.

[8] Brent Laster, *Jenkins 2: Up and Running: Evolve Your Deployment Pipeline for Next Generation Automation*, .ºReilly Media, Inc.", 2018.

[9] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame, "Nextflow enables reproducible computational workflows," *Nature biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[10] Wei Dai, Yarkın Doröz, Yuriy Polyakov, Kurt Rohloff, Hadi Sajjadpour, Erkay Savaş, and Berk Sunar, "Implementation and evaluation of a lattice-based key-policy abe scheme," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1169–1184, 2017.

[11] JL Gonzalez-Compean, Victor J Sosa-Sosa, Arturo Diaz-Perez, Jesus Carretero, and Ricardo Marcelin-Jimenez, "Fedids: a federated cloud storage architecture and satellite image delivery service for building dependable geospatial platforms," *International Journal of Digital Earth*, vol. 11, no. 7, pp. 730–751, 2018.

[12] Damien Giry, "Nist report on cryptographic key length and cryptoperiod (2020)," 2020.

[13] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid, *Recommendation for Key Management: Part 1-General (Revision 3)*, NIST Special Publication 800-57. National Institute of Standards and Technology, Technology Administration, 2012.

[14] Burton Rosenberg, *Handbook of Financial Cryptography and Security*, Chapman and Hall/CRC, 1st edition, 2010.

[15] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: A state-of-the-art review," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, 2019.

[16] Joseph Muli and Arnold Okoth, *Jenkins Fundamentals: Accelerate deliverables, manage builds, and automate pipelines with Jenkins*, Packt Publishing, 2018.

[17] Anton V. Uzunov, Eduardo B. Fernandez, and Katrina Falkner, "Security solution frames and security patterns for authorization in distributed, collaborative systems," *Computers & Security*, vol. 55, pp. 193 – 234, 2015.

[18] J.L. Gonzalez-Compean, Oscar Telles, Ivan Lopez-Arevalo, Miguel Morales-Sandoval, Victor J. Sosa-Sosa, and Jesus Carretero, "A policy-based containerized filter for secure information sharing in organizational environments," *Future Generation Computer Systems*, vol. 95, pp. 430 – 444, 2019.

Resource orchestration in federated fog environments with SDN and blockchain

Carlos Núñez-Gómez[1], Francisco M. Delicado[2], Carmen Carrión[2] y Blanca Caminero[2]

Abstract— **Fog computing offers decentralized computing and storage resources as a complement to the centralized approach of cloud computing. On the other hand, blockchain provides a decentralized and immutable ledger that can support the execution of arbitrary logic through smart contracts. Thus, blockchain smart contracts are employed as the foundation for a truly decentralized, autonomous, and resilient fog resource orchestrator. However, the potentially enormous number of geographically distributed nodes that compose this layer can threaten the viability of orchestration. Additionally, fog nodes may exhibit highly dynamic workloads, which can result in the orchestrator redistributing services among them. Therefore, it is also necessary to dynamically support network connections to these services regardless of their location.**

Software-defined networking (SDN) can be integrated into the orchestrator to achieve seamless service management. To address the two mentioned issues, the S-HIDRA architecture is proposed. It integrates SDN support within a blockchain-based orchestrator for container-based services in fog environments, aiming to provide low network latency and high service availability. Furthermore, a domain-based architecture is suggested as a potential scenario to address the distributed geographical nature of fog environments. The functionality of S-HIDRA has been validated through an implementation as a proof of concept.

Keywords— **Federated fog computing, Internet of Things (IoT), resource orchestration, Software Defined Networks (SDN).**

I. Introduction

FOG computing represents a developing paradigm that merges cloud computing and the Internet of Things (IoT) [1]. It aims to bring resources, including computing power, storage, and memory capacity, closer to the network's edges. Specifically, fog computing relies on fog nodes that have limited resources to establish distributed cloud services across widely dispersed edge networks. The time has come to contemplate decentralization and distribution as crucial aspects of the future generation of IoT services and, more importantly, a distributed architecture for orchestrating container-based services in fog computing to ensure resilience and fault-tolerance [2]. In this context, blockchain emerges as a natural solution for decentralizing fog computing and addressing certain challenges related to fog orchestration, such as security and auditability [3].

To address the architectural changes in the fog virtualization infrastructure and the associated challenges in network communication, our work proposes S-HIDRA, a federated blockchain-based architecture designed to facilitate the orchestration of containerized services in broader Fog-IoT environments. This framework incorporates a novel network protocol that dynamically optimizes the configuration of network devices (such as switches and routers) in real time.

In this research paper, our focus is on evaluating the efficiency of the decentralized architecture proposed for Fog-IoT environments, leveraging the adoption of blockchain and Software Defined Networking (SDN) [4]. By adding SDN capabilities, S-HIDRA streamlines our previous work to avoid static management of network traffic among fog nodes and IoT devices, thereby dynamizing network traffic towards the orchestrated containers. In particular, we introduce an intra-domain protocol to orchestrate container services in distributed fog architectures using smart contracts and SDN virtual services. This protocol is implemented and evaluated through real-world experiments conducted on a testbed, using real applications. Additionally, to effectively handle the geographical distribution inherent in fog environments, we propose a global federated fog architecture. This architecture utilizes S-HIDRA to partition scenarios featuring numerous end-devices and fog nodes distributed across multiple geographical locations into distinct domains. These domains are seamlessly coordinated through the utilization of a global blockchain mechanism.

The rest of the paper is organized as follows. Section II presents the related work. Then, Section III introduces S-HIDRA as a distributed fog computing management architecture based on blockchain and SDN. Section IV describes the operation model of the proposed architecture. In Section V, the evaluation results are presented. Finally, conclusions are included in Section VI.

II. Related work

Numerous proposals have been put forth to integrate blockchain into fog/edge architectures with the goal of bolstering the security of decentralized environments for IoT applications. These proposals also aim to effectively manage distributed IoT data, ensuring integrity, authentication, and privacy [5], [6].

Smart contracts can also be deployed between clients and service providers to validate Service Level Agreements (SLA) compliance for on-demand resource usage, as in [7]. As an example, in BlockEdge [8], the authors propose to use blockchain

[1]High-Performance Networks and Architectures Group (RAAP), Albacete Research Institute of Informatics (I3A), University of Castilla-La Mancha, e-mail: carlos.nunez@uclm.es.

[2]Department of Computing Systems, University of Castilla-La Mancha, e-mail: {francisco.delicado, carmen.carrion, mariablanca.caminero}@uclm.es.

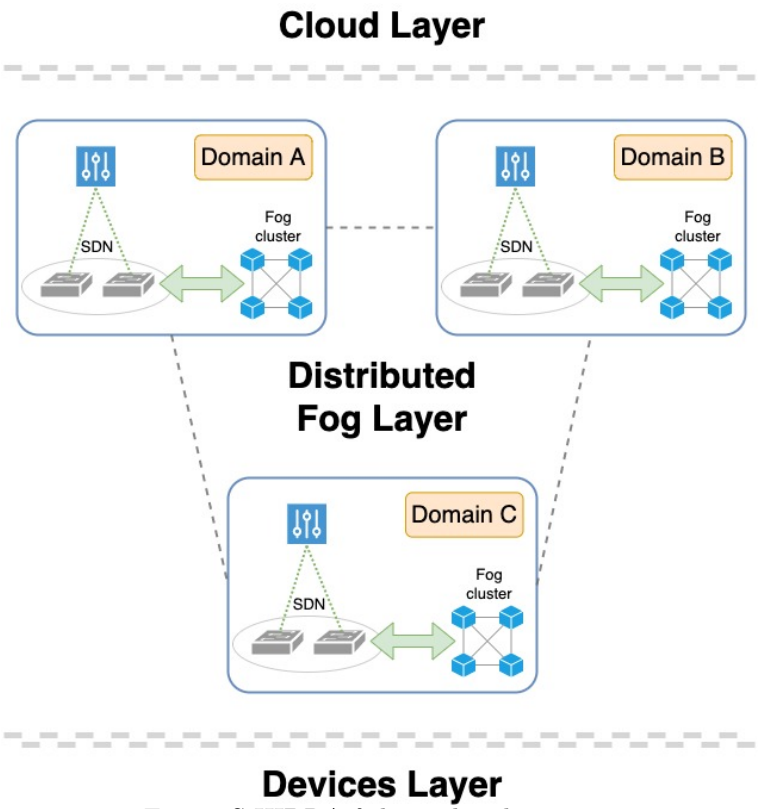


Fig. 1: S-HIDRA federated architecture.

as a means for auditing the different processes involved in an industrial IoT application. Focusing more specifically on resource management, the EdgeChain framework [9] proposes a credit-based resource management system based on a permissioned blockchain and a currency system.

Other works focus on researching the security, existing attacks and mitigation mechanisms related to SDN technology [10]. The authors in [11] tackle the problem of Distributed Denial of Service (DDoS) attack mitigation in SDN with a two-level strategy.

Thus, according to the reviewed literature, very few authors have worked on the resource management of fog/edge architectures by integrating blockchain and SDN technologies.

III. S-HIDRA: Global overview

The aim of the proposed architecture is to provide Quality of Service (QoS) to connected IoT devices by abstracting the orchestration of container-based services and network services from administrators, application designers, and fog nodes. Also, given that IoT devices are often spread across a vast geographical area, it is crucial to consider federation approaches to address scalability challenges and prioritize local data exchanges. By adopting these approaches, we can efficiently manage the growing number of IoT devices while ensuring that data transfers are localized whenever possible.

Thus, the primary objective of S-HIDRA is to combine the dynamic and programmable nature of SDN with secure, fault-tolerant, autonomous, and auditable resource orchestration based on blockchain technology. To achieve this objective, we propose distributed scenarios that are divided into various management and orchestration domains. For example, these domains can include smart buildings involving multiple unrelated companies or departments, smart campuses encompassing different schools or research groups, or distributed computing platforms. These interconnected domains rely on a peer-to-peer distributed ledger or blockchain to maintain a global state. Additionally, each domain maintains its own local state for performing service orchestration tasks and programmatically controlling network traffic to these services.

The high-level layered architecture of S-HIDRA is illustrated in Figure 1. The Distributed Fog Layer comprises multiple domains, facilitating the distribution of computing and network operations throughout the fog layer and contributing to the development of a federated fog infrastructure. Each domain consists of a local cluster of fog nodes, where S-HIDRA enhances the existing blockchain-based orchestration platform [12] by incorporating SDN capabilities. This integration enables the dynamic management of network traffic towards containerized services, facilitating programmability within the network architecture. Consequently, traffic can be efficiently controlled based on the current network state, allowing for load balancing, replication, and migration of containerized services.

As pointed out before, federation among different domains is supported by means of a global blockchain, coined as the Fog Layer Blockchain (FLB). The FLB can be implemented as a permissioned blockchain deployed as a consortium manner among the system domains, or even an open per-

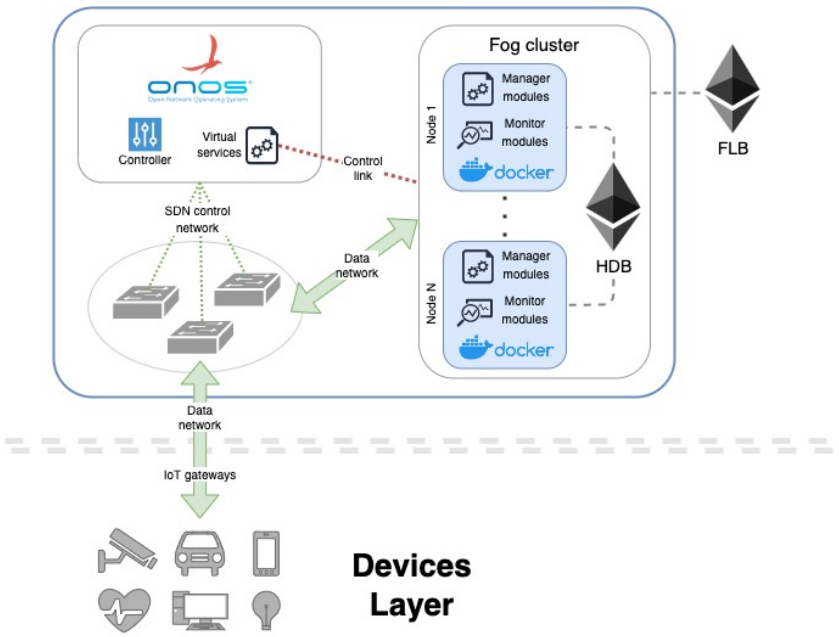


Fig. 2: S-HIDRA domain architecture.

missionless blockchain accessible worldwide. Regardless of the scenario, the FLB incorporates the Inter-Domain Registry (IDR) functional module and corresponding smart contract, responsible for overseeing the global domain state. This state comprises shared public metadata essential for domains to interact with each other. The IDR manages various types of metadata, including domain registration data, details regarding inter-domain migrations, IP addresses and port information for publicly accessible services, domain reputation scores, node blacklists, and more.

In the proposed fog architecture, each SDN controller has the primary responsibility of managing network devices within a specific domain. To facilitate the exchange of network information among SDN controllers, we adopt a flat model or horizontal architecture, similar to the one described in [4]. The FLB governs the inter-domain SDN behavior at the application level. As a result, domain SDN controllers can interact with one another to record and retrieve network information related to containerized services and orchestration events that take place within their respective domains.

Focusing in the intra-domain level, a domain is composed of (1) a cluster of fog nodes connected through a local permissioned blockchain called HIDRA Domain Blockchain (HDB), and (2) an SDN networking layer in which an SDN controller is responsible for managing a subset of Distributed Fog Layer network devices. Irrespective of the scale and complexity of network devices and topology within a domain, an SDN controller possesses the capability to dynamically manage traffic to and from fog nodes. It achieves this by querying the local domain state stored in the HDB in a secure and decentralized way. The local domain state encompasses the information and metadata maintained by the functional modules of S-HIDRA.

Figure 2 illustrates an overview of an S-HIDRA domain, showcasing the essential components and necessary intra-domain connections. As previously mentioned, each domain comprises an SDN controller and a cluster of fog nodes. For the SDN controller, we use ONOS [13], a prominent open-source SDN controller renowned for its high availability, performance, and scalability. ONOS enables the modular development and deployment of SDN applications while offering distributed control capabilities. The SDN approach divides network traffic into control, data, and application planes, providing a comprehensive framework for efficient network management.

An S-HIDRA domain cluster is composed of N fog nodes that carry out resources and services orchestration tasks following a well-defined, consensus-based workflow protocol (see Section IV). Both FLB and HDB blockchains are implemented using the Ethereum technology, so the smart contracts related to their functional modules are executed through the EVM. Although the SDN approach divides traffic into different planes or networks, fog nodes belonging to the data plane have a *control link* that allows them to send management instructions to the ONOS application plane, specifically to the virtual services application. This allows network traffic to be modeled according to the orchestration decisions made. Note that all system fog nodes implement the same software client required for inter-domain and intra-domain communication (i.e. communication with FLB and HDB blockchains, SDN controllers and Docker daemons), and for resource monitoring and rule enforcement.

The orchestration plane view, depicted in Figure 3, provides an overview of the key components of S-HIDRA nodes and their connection to the SDN plane. In the figure, we can also identify components associated with our previous research work, HIDRA [12]. HIDRA serves as the foundation for the domain-based distributed architecture proposed

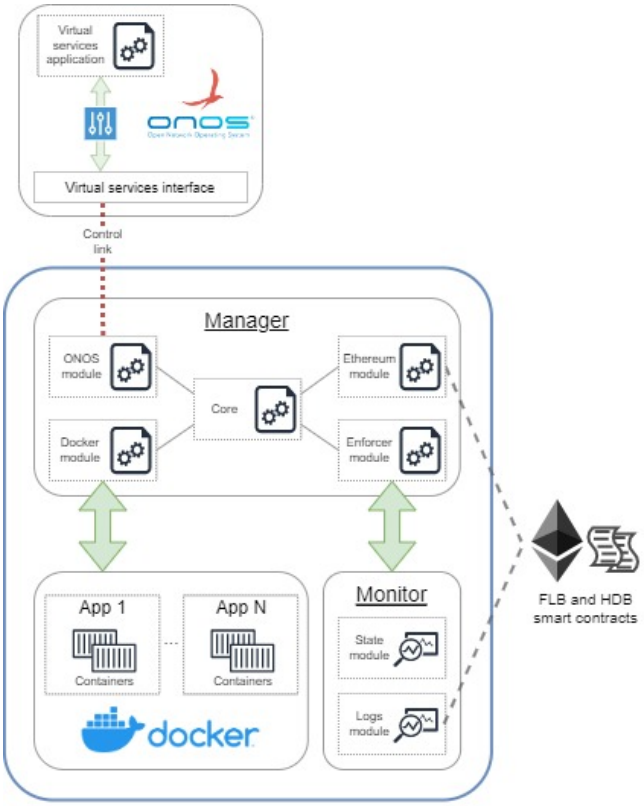


Fig. 3: Orchestration plane view.

by S-HIDRA. Within HIDRA, a local P2P network is established among HIDRA clusters to facilitate the sharing and synchronization of the cluster state. To achieve this, a permissioned Ethereum blockchain is deployed over the P2P network, enabling the execution of smart contracts. Additionally, HIDRA employs the Monitor component to monitor and isolate node resources, such as CPU, memory, and disk storage, while leveraging Docker virtualization service for containerization. Some other key components, such as the Manager, in HIDRA had limited functionality for resource orchestration. These components have been further enhanced and extended in S-HIDRA to provide more robust resource management capabilities.

Below, we provide a detailed overview of the essential components required for the distributed SDN-based orchestration in S-HIDRA:

- **SDN controller**. Each domain employs an ONOS controller responsible for managing the network devices within that domain. Inter-domain synchronization of network information is facilitated through the FLB at the SDN application level. Consequently, the ONOS controllers in S-HIDRA establish interconnections, forming a distributed SDN data store. At the intra-domain level, local clusters can be formed by replicating instances of ONOS controllers [14].
- **Virtual services application**. Each S-HIDRA domain deploys a local instance of the virtual services application. This ONOS application allows to abstract the network configuration from the deployed containerized services. A virtual service is linked to a single containerized service and it is responsible for allocating and configuring an IP, port, and protocol (TCP/UDP) in order to expose the containerized service within a domain, regardless of the fog nodes this service is running on. The virtual services application also provides other features such as replication and load balancing between different replicas of the same containerized service (which may be running on different fog nodes).
- **Virtual services interface**. Fog nodes within S-HIDRA play a pivotal role in making distributed decisions regarding the placement of containerized services. Additionally, these fog nodes undertake the registration and management of virtual services within their respective S-HIDRA domains. Consequently, a link interface becomes crucial in facilitating the exchange of network information among S-HIDRA nodes, catering to both inter-domain and intra-domain SDN states. To enable this communication, the ONOS platform offers the capability to implement and expose custom REST APIs. These APIs act as intermediaries between ONOS applications and the external environment, providing a means for S-HIDRA nodes to send and query network information effectively.
- **Functional modules/smart contracts.** S-HIDRA divides the control logic into five functional modules, each of them implementing one or more EVM smart contracts that distribute the system logic among all participants. Thus, fog nodes share and query data and exchange control messages related to orchestration tasks by using blockchain transactions. These functional modules are: (1) the *Distributed Device Registry (DDR)* which registers and authenticates fog nodes, (2) the *Distributed Event Logger (DEL)* responsible for managing orchestration events, (3) the *Distributed Container Registry (DCR)* focused on storing the historical and desired state in terms of resources and containerized services, (4) the *Distributed Reputation System (DRS)* which calculates the reputation score of nodes according to their actions, and (5) the *Inter-Domain Registry (IDR)* to globally share data generated by the intra-domain workflows. Modules (1)-(4) are executed in isolation through the HDB, and module (5) is shared among domains via the FLB.
- **ONOS module**. Each S-HIDRA node's software client needs a module to facilitate interaction with SDN capabilities. The ONOS module within the Manager component assumes responsibility for managing requests to the virtual services application via the virtual services interface. When deploying, migrating, or deleting containerized services within a domain, the workflow protocol outlined in Section IV dictates the timing and fog node that triggers the

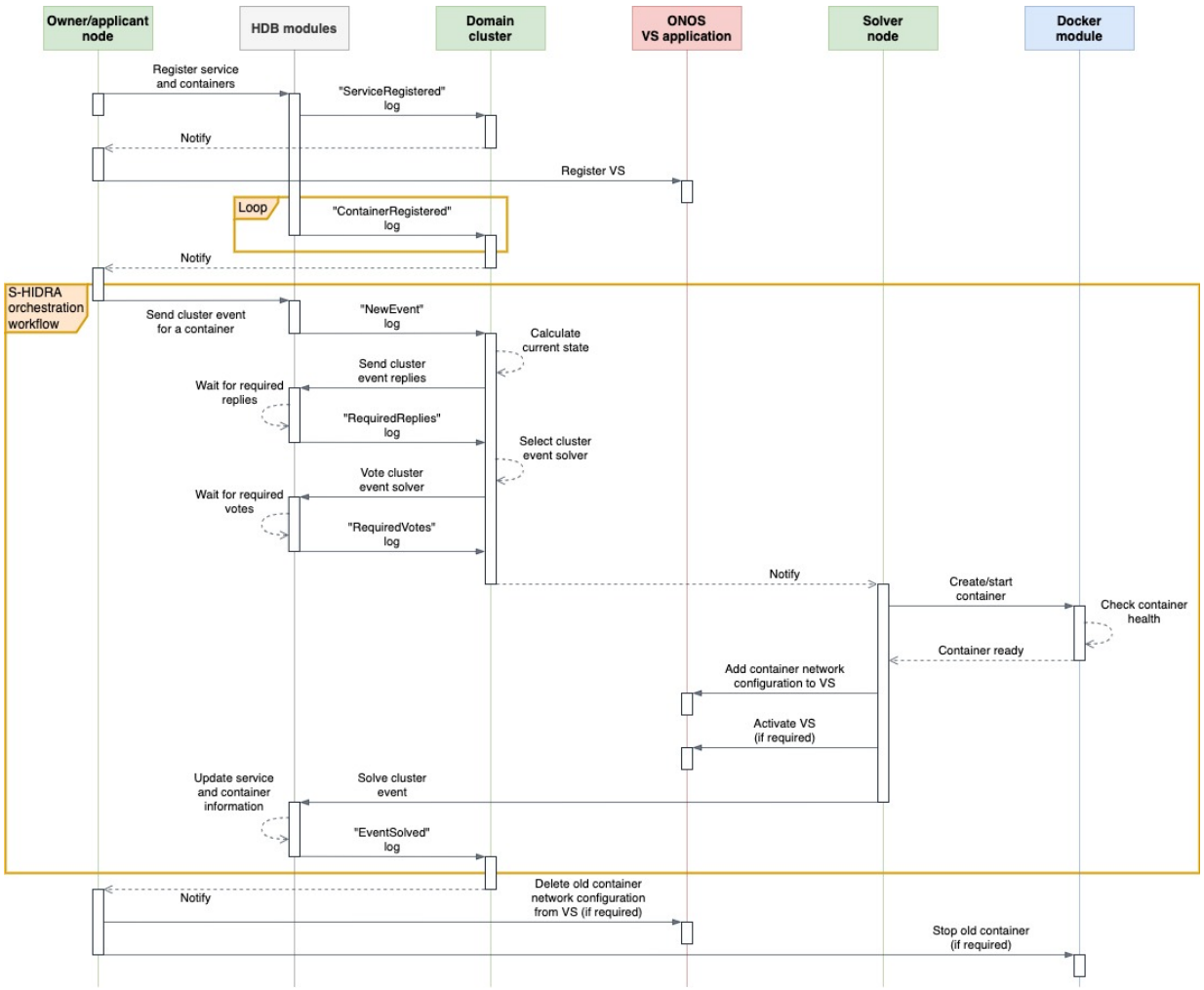


Fig. 4: S-HIDRA workflow protocol.

ONOS module to create, modify, or delete virtual services associated with the containerized service.

- **Docker module**. The deployment, migration, and execution times of containers vary based on their type and complexity. In S-HIDRA, these time factors play a crucial role in synchronizing containers and virtual services. During the migration process of a container, the virtual service linked to the containerized service remains the same. However, there may be short periods of downtime until the migrated container becomes active and healthy on the destination node (referred to as the solver node). Such downtimes can impact the availability of containerized services. To address this, the Docker module incorporates a *container health check system*, which continuously monitors the internal state of containers and notifies fog nodes of their current state. This feature ensures that the synchronization of containers and virtual services remains robust and aligns with the workflow protocol.

The Ethereum and logs modules, shown in Figure 3, play a vital role in facilitating the interaction and synchronization of fog nodes within the S-HIDRA architecture. They are only components that directly engage with the blockchain. The Ethereum module assumes the responsibility of signing and transmitting transactions, as well as interacting with the functional modules of S-HIDRA. On the other hand, the logs module monitors and captures any EVM (Ethereum Virtual Machine) asynchronous log emitted by a smart contract in response to incoming transactions. To establish seamless connectivity among fog nodes at both intra-domain and inter-domain levels, S-HIDRA necessitates the integration of these modules with both the HDB and the FLB.

IV. S-HIDRA: Workflow protocol

Figure 4 shows the steps required to orchestrate containerized services at intra-domain level. The entities participating in the workflow protocol are shown at the top of the figure. From left to right, (1) a domain fog node acting as a containerized service owner or as an applicant node with problems, (2) the HDB functional modules/smart contracts, (3) a cluster representing all fog nodes of a domain, (4) the SDN capabilities including the ONOS virtual services application and virtual services interface, (5) the solver node selected by the cluster to carry out the tasks required by the owner/applicant node, and finally, (6) the Docker module.

Within the S-HIDRA architecture, domains pos-

sess the capability to synchronize information through cluster events, facilitated by the DEL functional module and the log exchange protocol implemented in the HDB smart contracts. While the DEL module supports various types of cluster events, we will now shift our focus specifically to the container deployment and migration processes. The deployment process starts with a fog node that requires the creation of a new containerized service. In this case, the node assumes the owner role and takes control of this service, regardless of which cluster node executes the service's containers. To link the containerized service with an SDN virtual service, first, the owner node registers the metadata of the new service and its containers in the DCR module. Once the metadata registration transaction has been sent and validated by the HDB, multiple asynchronous logs are generated: a *ServiceRegistered* log that notifies the owner in order to register a new virtual service in ONOS linked to the containerized service, and a *ContainerRegistered* log for each service container.

At this stage, the containerized service has been successfully registered, but none of its containers have been orchestrated or executed. Additionally, the registered virtual service remains inactive. The subsequent step involves executing an S-HIDRA orchestration workflow for each registered container. Note that a container migration process would also start at this point. In this case, the node assumes the role of an applicant when it detects issues with its resources or running containers. In both container orchestration and migration processes, the S-HIDRA workflow commences with a node triggering a DEL event specific to a container. This event is depicted as a cluster event in Figure 4. After the event transaction is sent and validated, the DEL smart contract generates a *NewEvent* log, notifying the other cluster nodes of the event occurrence.

Each node, upon receiving the event notification, generates a report containing its current state and promptly sends it as a reply to the DEL event. The DEL contract awaits the collection of the required event replies before throwing a *RequiredReplies* log, signaling the continuation of the orchestration workflow. The number of replies necessary depends on the initial cluster configuration.

Now, the cluster has been synchronized and all fog nodes know the current state. Thus, it is possible to elect the solver node that will host the container in a decentralized manner. To this end, each cluster node selects a solver node considering the current cluster state and sends a vote transaction to the DEL contract. Similar to the wait-for-replies phase, the orchestration workflow waits until the DEL module has received the number of votes required by the initial cluster configuration. Once the votes have been collected, the DEL contract notifies the elected solver node via a *RequiredVotes* log.

At this point, the cluster has the information required to create the owner node container or replicate the applicant node container. In any case, the solver node instantiates the container using the Docker module and configures it according to the metadata registered by the owner node. In a container migration process, the cluster will be executing two instances of the applicant node container for a short time (redirecting for now all device requests through the old instance). Regardless of the event type, the Docker module's health check system is responsible for monitoring the health of the container. When the system reports a healthy container state, the solver node sends the network configuration of the new container instance (node IP, exposed port, and protocol) to the virtual service previously registered in ONOS. Note that not all containers need to register their network configuration in a virtual service, some containers work internally linked via Docker. Subsequently, once the virtual service has been updated, the solver node activates it if required.

To complete the cluster event, the solver node sends a solve transaction to the DEL contract. This transaction can only be sent by the solver node. A solve transaction closes a cluster event but also changes the cluster state depending on the event sent. Although domain devices could perform requests to the containerized service just after activating the virtual service (before solving the event), the HDB functional modules are the only ones responsible for maintaining the desired cluster state. It means, for example, that the DCR decides which containerized services are active, inactive, or deleted, regardless of the state of the ONOS virtual services application.

To finalize the S-HIDRA orchestration workflow, the DEL contract notifies the cluster nodes via an *EventSolved* log that the cluster event has been solved. At this point, the cluster nodes may carry out multiple actions in order to seal the new cluster state: nodes or service availability checks, send metadata to the FLB and IDR module, reputation calculations, etc. The applicant node may also perform cleaning actions related to the problematic/old container instance. Thus, the virtual service now only redirects network traffic from devices to the new container hosted by the solver node, allowing the applicant node to release resources.

V. Evaluation results

To evaluate the anticipated functional and non-functional characteristics of S-HIDRA, we have created a testbed that emulates a real-world domain within the S-HIDRA framework.

Figure 5 shows the key components of a domain composed of three fog nodes interconnected through an OpenFlow switch controlled by an ONOS controller 2.6.0. Both the OpenFlow switch and the fog nodes are implemented in Raspberry Pi devices.

The image depicted in Figure 5 illustrates the essential elements of a network domain comprising three fog nodes. These nodes are interconnected through an OpenFlow switch, controlled by an ONOS controller version 2.6.0. Both the Open-

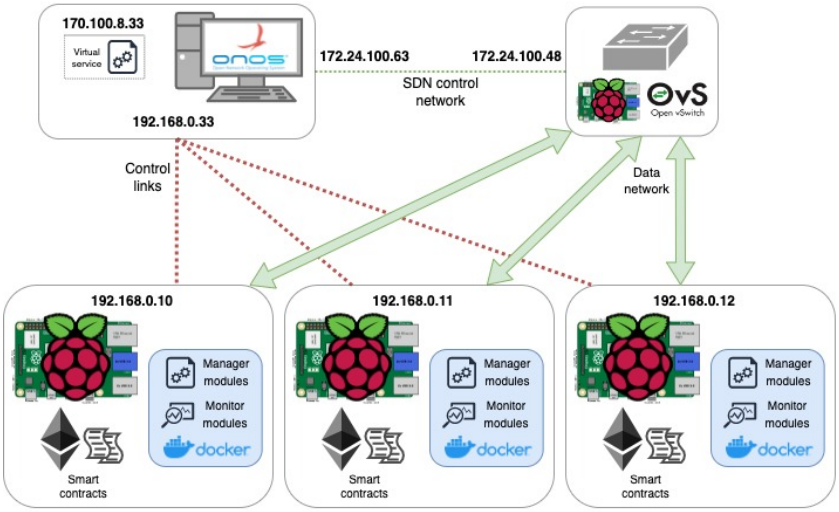


Fig. 5: S-HIDRA testbed.

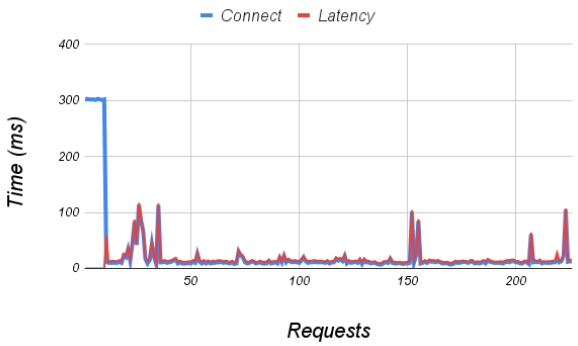


Fig. 6: Network measurement during an NGINX container deployment.

Flow switch and the fog nodes are implemented using Raspberry Pi devices.

The Raspberry Pi performing the switch role runs Open vSwitch (OVS) 2.10.7 in order to virtualize the networking capabilities of an SDN switch. The SDN components are deployed on a Debian 10 virtual machine with 2.5GHz quad-core CPU and 4GB of memory.

The testbed includes three different IP networks: (1) the *data network* (192.168.0.0/24), (2) the *SDN control network* (172.24.100.0/23), and (3) the custom network addresses representing the virtual services (e.g. 170.100.8.33). The initial cluster configuration also specifies the ONOS controller network address (192.168.0.33) that acts as management endpoint for the *control links* between cluster nodes and the virtual services interface.

Figure 6 shows the *connect times* and *latency times* during a container deployment process. *Connect times* represent the time taken to establish the TCP connection with the endpoint, while *latency times* include the *connect times* plus the endpoint computation time and the time until after receiving the first response. During the initial test, an NGINX web server is deployed as a container. Each request directed to the container's endpoint initiates a new TCP connection, as the observer cluster node uses a different source port for each request. Consequently, a new SDN flow rule is generated and installed on the OVS for every request. The test results indicate that there are peaks in the *connect times* at the beginning of the test, primarily due to the 300ms timeouts occurring before the container is fully operational. However, once the deployment instruction for the container is transmitted and the S-HIDRA orchestration workflow is completed, the NGINX web server starts successfully receiving requests. Generally, the *latency times* average around 16ms, although occasional peaks may occur depending on the saturation level of the SDN components.

Figure 7 shows the percentage of requests and latency during the first test. The results obtained reveal that despite the peaks 95% of requests remain below 30ms.

The upcoming tests focus on the container migration process, which is not only a critical but also a resource-intensive orchestration task. This process entails initiating the service on the solver node and rerouting traffic flows to it. It is crucial to ensure a seamless transition, as the service should only be halted on the applicant node once it is fully operational on the solver node. Consequently, monitoring the availability of the service during S-HIDRA testing becomes imperative.

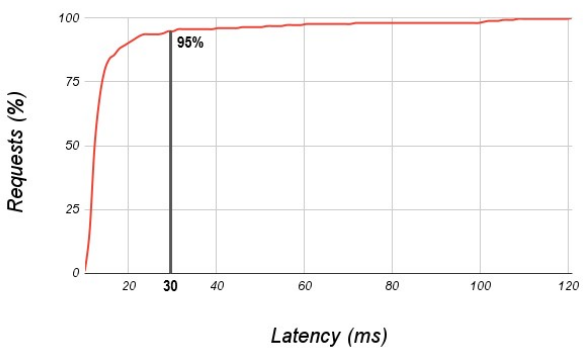


Fig. 7: Percentage of requests and latency during an NGINX container deployment.

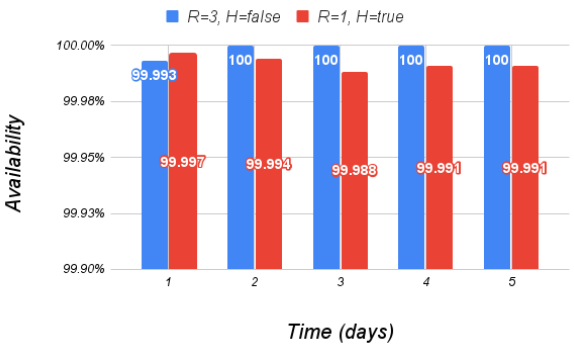


Fig. 8: Availability per day according to request rate and container health check.

Figure 8 shows the percentages of successful requests sent to an NGINX web server container during two availability tests performed over five consecutive days. In these tests, different configurations were compared based on the time between consecutive requests sent to the container endpoint (R parameter – secs), and on the use of the health check system when the S-HIDRA orchestration workflow executes a container (H parameter – true/false). In both tests, the availability percentages are around 99.99%. The obtained results demonstrate that the orchestration workflow maintains synchronization with the state of the ONOS virtual services application. This synchronization significantly reduces the loss of requests during the orchestration of containerized services.

Additional results, such as number of migrations or details on latencies, can be found in [15].

VI. Conclusions

In this paper, we have introduced S-HIDRA, an architecture designed for resource orchestration of containerized services in decentralized fog computing environments. Building upon prior research on the blockchain-based HIDRA orchestrator, we have developed S-HIDRA to cater to geographically extensive fog computing environments that are divided into distinct domains of devices/nodes. By leveraging blockchain technology, S-HIDRA inherits essential attributes such as immutability, availability, and transparency.

To enable dynamic and programmable management of network traffic for decentralized containerized services, we propose integrating SDN capabilities into the HIDRA orchestrator and S-HIDRA domains. Furthermore, we have implemented a testbed that emulates an S-HIDRA domain to assess the viability of our proposal. The obtained results demonstrate effective orchestration, along with low latency levels and high availability of containerized services throughout the container deployment and migration processes.

Our future research will involve exploring specific use cases in which the S-HIDRA proposal can be applied, such as smart cities or campuses. Additionally, we plan to develop an advanced reputation system that can quantify the behavior of nodes and domains. This enhanced reputation system will enable more sophisticated decision-making processes when orchestrating containerized services within the S-HIDRA framework.

Acknowledgements

This work was supported under PID2021-123627OB-C52 project, funded by MCIN/AEI/10.13039/501100011033 and by European Regional Development Fund (ERDF) "A way to make Europe", and under 2023-GRIN-34056 Consolidated Group Grant, funded by the University of Castilla-La Mancha.

References

[1] K. Velasquez, D. Abreu, M. Assis, C. Senna, D. Aranha, L. Bittencourt, N. Laranjeiro, M. Curado, M. Vieira, E. Monteiro, and E. Madeira, "Fog orchestration for the Internet of Everything: state-of-the-art and research challenges," *J Internet Serv Appl*, vol. 9, no. 14, 2018.

[2] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Comput. Surv.*, vol. 55, no. 7, dec 2022.

[3] H. L. Cech, M. Großmann, and U. R. Krieger, "A Fog Computing Architecture to Share Sensor Data by Means of Blockchain Functionality," in *2019 IEEE International Conference on Fog Computing (ICFC)*, June 2019, pp. 31–40.

[4] Yustus Eko Oktian, SangGon Lee, HoonJae Lee, and JunHuy Lam, "Distributed SDN controller system: A survey on design choice," *Computer Networks*, vol. 121, pp. 100–111, 2017.

[5] Yongkai Fan, Guanqun Zhao, Xia Lei, Wei Liang, Kuan-Ching Li, Kim-Kwang Raymond Choo, and Chunsheng Zhu, "SBBS: A secure blockchain-based scheme for IoT data credibility in fog environment," *IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 9268–9277, 2021.

[6] H. Baniata and A. Kertesz, "A Survey on Blockchain-Fog Integration Approaches," *IEEE Access*, vol. 8, pp. 102657–102668, 2020.

[7] P. Kochovski, V. Stankovski, S. Gec, F. Faticanti, M. Savi, D. Siracusa, and S. Kum, "Smart Contracts for Service-Level Agreements in Edge-to-Cloud Computing," *Journal of Grid Computing*, vol. 18, no. 4, pp. 673–690, dec 2020.

[8] T. Kumar, E. Harjula, M. Ejaz, A. Manzoor, P. Porambage, I. Ahmad, M. Liyanage, A. Braeken, and M. Yliantila, "BlockEdge: Blockchain-Edge Framework for Industrial IoT Networks," *IEEE Access*, vol. 8, pp. 154166–154185, 2020.

[9] J. Pan, J. Wang, A. Hester, I. Alqerm, Y. Liu, and Y. Zhao, "EdgeChain: An Edge-IoT Framework and Prototype Based on Blockchain and Smart Contracts," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4719–4732, June 2019.

[10] Yunhe Cui, Qing Qian, Chun Guo, Guowei Shen, Youliang Tian, Huanlai Xing, and Lianshan Yan, "Towards DDoS detection mechanisms in software-defined networking," *Journal of Network and Computer Applications*, vol. 190, pp. 103156, 2021.

[11] Zakaria Abou El Houda, Abdelhakim Senhaji Hafid, and Lyes Khoukhi, "Cochain-SC: An Intra- and Inter-Domain DDoS Mitigation Scheme Based on Blockchain Using SDN and Smart Contract," *IEEE Access*, vol. 7, pp. 98893–98907, 2019.

[12] C. Núñez-Gómez, B. Caminero, and C. Carrión, "HIDRA: A Distributed Blockchain-Based Architecture for Fog/Edge Computing Environments," *IEEE Access*, vol. in press, 2021.

[13] Open Networking Foundation, "Open Network Operating System (ONOS)," 2014.

[14] Open Networking Foundation, "Cluster Coordination," `https://wiki.onosproject.org/display/ONOS/Cluster+Coordination`, 2016, Accessed: 2023-05-26.

[15] C. Núñez-Gómez, C. Carrión, B. Caminero, and F. M. Delicado, "S-HIDRA: A blockchain and SDN domain-based architecture to orchestrate fog computing environments," *Computer Networks*, vol. 221, pp. 109512, 2023.

Revisitando la ley de Amdahl para la consolidación de servidores virtuales

Carlos Juiz [1], Belen Bermejo [2]

Resumen— **Las tecnologías de virtualización son ampliamente utilizadas en los centros de datos, especialmente en la computación en la nube. Ello permite facilitar la gestión del centro de datos, reducir el número de máquinas físicas, su enfriamiento, y consecuentemente el espacio y el consumo de potencia eléctrica. Por tanto, conocer hasta cuánto consolidar servidores virtuales en una máquina física es un desafío fundamental para los administradores de sistemas. Este artículo presenta un método de evaluación del rendimiento de la consolidación con reparto temporal de la carga, en cualquier tipo de virtualización. El método permite determinar la sobrecarga debida a la virtualización, la escalabilidad y eficiencia de distintas consolidaciones en un mismo servidor, compararlas con otros servidores físicos y compararlas con otro tipo de virtualizaciones.**

Palabras clave— **Evaluación del rendimiento, Servidores, Virtualización, Consolidación, Ley de Amdahl.**

I. Introducción

La computación en la nube y los centros de datos se han convertido en una parte importante de nuestra vida diaria debido a los servicios a escala que proveen [1]. Así, los centros de datos pueden proporcionar la ilusión de poseer recursos ilimitados para los usuarios. La tecnología de virtualización proporciona varias características a los proveedores de la nube, como la multiplexación de recursos, migración, consolidación de servidores y redimensionamiento de las máquinas virtuales (VM) [2]. Utilizando estas características, los proveedores de la nube pueden proporcionar recursos bajo demanda para sus usuarios y clientes. Sin embargo, la enorme demanda de esos recursos obliga a los proveedores a encontrar una forma de reducir el espacio, la potencia eléctrica, enfriamiento del centro de datos y en consecuencia reducir los costes de alojar numerosas máquinas físicas (PM) en ellos. La técnica de consolidación de servidores consiste en el alojamiento empaquetado de varias VM en el menor número de PM.

A pesar de las ventajas antes expresadas, el precio a pagar por la virtualización es empeorar el rendimiento del servidor por la sobrecarga (*overhead*) añadida, inherente al software de virtualización, el manejo de un número VM consolidadas y el modo de reparto de la carga de trabajo en las mismas [3]. En consecuencia, la consolidación de servidores virtuales intenta empaquetar una cantidad de VM en el menor número de PM para optimizar la utilización de recursos del centro de datos, pero a la vez tomar en consideración la cantidad de VM por PM, para que la virtualización no esté sobrexplotada (VM *sprawling*). Este fenómeno consiste en la proliferación descontrolada de VM en los centros de datos, produciendo aumento de la gestión y el mantenimiento. El *sprawling* se produce normalmente por sobreprovisionamiento, falta de mantenimiento proactivo de la localización de VM (VM *allocation*) o su migración descontrolada (VM *migration*) [4].

Por otro lado, existen tanto estandarización *de facto* como *de iure* para los servidores físicos de los centros de datos, por ejemplo, para determinar su rendimiento, pero no cuánto se pueden virtualizar. Es decir, no se conoce cómo establecer, de forma comparativa, cuántas máquinas físicas (PM) se deben establecer o cuántas máquinas virtuales (VM) consolidar en una máquina física para una determinada carga de trabajo. Este conocimiento es crucial, no solo para establecer un estándar de rendimiento para máquinas consolidadas, sino también para los administradores de sistemas y el dimensionamiento de los centros de datos.

Este artículo está organizado de la siguiente forma. En la sección II se comenta el trabajo relacionado con la temática de la investigación. La sección III profundiza en los sistemas de virtualización y su sobrecarga (*overhead*) inherente. La sección IV se relata la contribución principal de los autores, con el planteamiento del problema investigado, la formulación teórica propuesta y su experimentación. Se demuestra la aplicabilidad de los conceptos clásicos y nuevos que se incluyen: aceleración (*speedup*), eficiencia e isoeficiencia, consolidación óptima, eficonsolidación, isoconsolidación y sobrecarga añadida. En la sección V se explica cómo se han realizado los experimentos que se han mostrado en la sección anterior. Finaliza el artículo con las secciones correspondientes a temas de discusión y conclusiones.

II. Trabajo relacionado

Se han realizado diversos estudios para abordar el efecto de la sobrecarga de consolidación y la degradación del rendimiento en los sistemas consolidados, así como de la calidad del servicio. También para comparar el rendimiento de VM y contenedores, ya que estos últimos necesitan un software más ligero de desplegar y mantener, a costa de soportar tareas de ejecución mucho menores [5]. En [6], los autores clasificaron los estudios de investigación existentes sobre comparaciones de rendimiento entre hipervisores de VM y contenedores. Sin embargo, las comparaciones de rendimiento se realizaron desde la perspectiva de la aplicación, sin considerar las sobrecargas soportadas por la PM. De forma similar en [7], los au-

[1]Dpto. de Ciencias Matemáticas e Informática, Universitat de les Illes Balears, e-mail: cjuiz@uib.es

[2]Dpto. de Ciencias Matemáticas e Informática, Universitat de les Illes Balears, e-mail: belen.bermejo@uib.es

tores compararon el rendimiento de KVM y Docker, concluyendo que Docker ofrece un mejor rendimiento desde la perspectiva de la aplicación. En [8], se planteó la viabilidad de los contenedores en aplicaciones de alto rendimiento. Los trabajos anteriores consideraron el rendimiento desde la perspectiva de la aplicación, ejecutado en una máquina virtual o un contenedor. En cambio, en nuestro trabajo, consideramos el rendimiento desde la perspectiva del servidor físico, que contiene y soporta cualquier virtualización en centros de datos. En [9], los autores estudiaron los factores más influyentes que afectan a los servidores consolidados. También, revisaron el estado del arte de la investigación sobre la gestión de la sobrecarga (*overhead*) de rendimiento de VM para revelar sus causas. Considerando estos factores, en [5], los autores clasificaron los tipos de sobrecarga de consolidación y se propuso un método general para estimar los valores de los tiempos de ejecución, incluyendo esas sobrecargas. Esta clasificación y el método se puede aplicar a sistemas con una variedad de características de servidor y carga de trabajo (*workload*), a máquinas virtuales o contenedores. Posteriormente, en [10], se generalizó el método de estimación del tiempo de ejecución, para considerar combinaciones anidadas de sucesivos niveles de consolidación, principalmente, de contenedores dentro de máquinas virtuales, a su vez alojadas en máquinas físicas.

En este trabajo se van a revisitar y reinterpretar conceptos básicos de paralelismo, básicamente las leyes de aceleración, eficiencia por máquina, eficiencia del tiempo de ejecución, así como otros nuevos conceptos, adaptados a la consolidación de VM en PM. Es un trabajo no exento de riesgos, puesto que revisitar conceptos básicos y reinterpretarlos, tampoco facilita la selección de trabajos relacionados, por un lado, debido a la ya extendida bibliografía y su popularidad, por otro, la incertidumbre de errar en la reinterpretación. El espacio limitado del artículo no permite citar a todas las referencias que sería justo recoger, pero quizás [11] pueda servir como resumen sobre las definiciones clásicas de aceleración (*speedup*), eficiencia, isoeficiencia y las archiconocidas leyes de Amdahl [12] y Gustaffson [13], reunidas en una sola interpretación gracias a [14]. Pero es [15], donde Gunther propone modificar la ley de Amdahl en su propuesta USL (*Universal Scalability Law*) la que inspira este trabajo, ya que el capítulo de virtualización y sobre todo consolidación parece inconcluso. Este trabajo pretende reutilizar conceptos básicos de paralelismo y reunirlos para determinar el grado de consolidación óptimo de VM en PM y así encontrar un equilibrio entre número de máquinas paralelas y su rendimiento.

III. Sistema de virtualización y sobrecarga

La virtualización permite administrar el centro de datos de una manera más flexible al agregar una capa de software administrador de máquina virtual (VMM) o hipervisor. El hipervisor se encarga de crear y administrar máquinas virtuales (VM), que son capaces de ejecutar entornos de ejecución aislados. La tecnología de virtualización se puede implementar desde el punto de vista del sistema (se puede implementar directamente en la parte superior del hardware) o desde el punto de vista del proceso (se necesita un sistema operativo para desplegarse). En este trabajo, nos centramos en el punto de vista del sistema. Particularmente, estamos interesados en máquinas virtuales de sistema y técnicas de virtualización completa, paravirtualización y virtualización asistida por hardware, así como en los diferentes tipos de hipervisores, tipo-I y tipo-II, y en menor grado contenedores (ver figura 1) [10]. En aras de mayor brevedad la mayoría de los experimentos se centrarán en el tipo I, aunque es generalizable al tipo II y en menor medida a los contenedores, tal como se explicará más adelante.

Dado que la virtualización agrega una capa adicional (el hipervisor), la sobrecarga (*overhead*) es inherente a cualquier implementación de esta tecnología. Sin embargo, la virtualización se puede poner en práctica de muchas maneras diferentes y determina la magnitud de la sobrecarga [5]. Además, la consolidación de máquinas virtuales agrega otro tipo de sobrecarga a medida que aumenta la cantidad de máquinas virtuales consolidadas, de forma análoga (aunque diferente) a los problemas de coherencia y comunicación en paralelismo. El grado de consolidación determina la degradación del rendimiento (por la cantidad de sobrecarga) y, como consecuencia, la calidad de servicio que experimentan los usuarios. La magnitud del valor de la sobrecarga de la consolidación de máquinas virtuales, por otro lado, depende de factores como la tecnología del hipervisor y la carga de trabajo ejecutada, entre otros [3].

En cualquier caso, cuanto mayor sea el número de VM consolidadas dentro de una misma PM, mayor será la sobrecarga porque el hipervisor se encarga de administrarlas, demandando simultáneamente diferentes recursos.

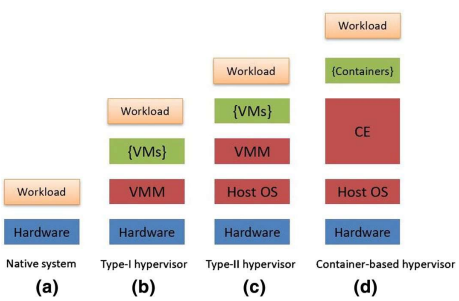


Fig. 1: Arquitectura nativa frente (a) arquitectura de virtualización: hipervisor (b) tipo I, hipervisor (c) tipo II y (d) hipervisor basado en contenedores

Una vez estudiado cómo cuantificar la magnitud de la sobrecarga, para cualquier configuración de servidores virtuales, es crucial conocer hasta cuánto consolidar VM en una PM, es decir la escalabilidad de la consolidación. Este artículo presenta un método evaluación del rendimiento de la consolidación de cual-

quier tipo de servidores virtuales, revisitando la ley de Amdahl. El método permite determinar la escalabilidad y eficiencia de distintas consolidaciones en un mismo servidor, compararlas con otros servidores físicos y compararlas con otro tipo de virtualizaciones.

IV. Planteamiento, formalización y experimentación

A. Aceleración de PM sobre VM (speedup)

Dado que el objetivo de este trabajo es proporcionar un método para determinar la calidad de la consolidación de servidores y su escalabilidad, la carga de trabajo adicional que debe realizar un servidor físico, para admitir la consolidación, existe independientemente del tipo de hipervisor. Cada hipervisor necesita diferentes requisitos de software para ser implementado, pero podemos describir las diferentes capas de interés como se muestra en la Figura 1. En aras de la simplicidad y claridad de nuestro método, modelamos nuestros escenarios teóricos y empíricos tal como se muestra en la Figura 2 [5].

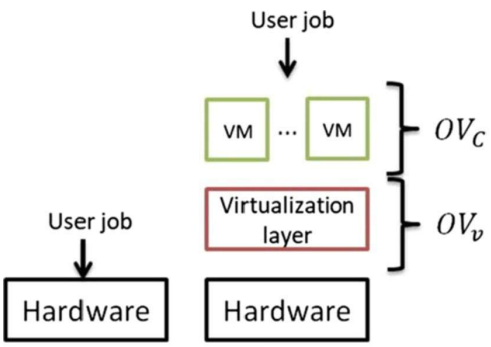


Fig. 2: Tipo de sobrecargas en el rendimiento de las VM

Dados N servidores físicos homogéneos (PM) que ejecutan en paralelo una carga de trabajo de CPU y memoria (de tamaño total T_m) y un servidor físico, idéntico a los anteriores, que ejecuta esa misma carga dividida en N máquinas o servidores virtuales (VM), se pueden definir (ver Figura 3):

- $T_{PM}(N)$ el tiempo medio de ejecución de las subtareas de tamaño Tm/N de la carga de trabajo por los N servidores físicos (PM).
- $T_{VM}(N)$ el tiempo medio de ejecución de las mismas subtareas del mismo tamaño Tm/N de la carga de trabajo por los N servidores virtuales (VM).

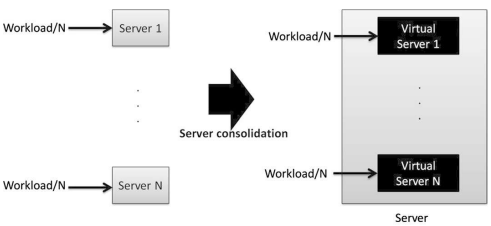


Fig. 3: Consolidación con servidores homogéneos

El factor de mejora o aceleración [11] de las PM homogéneas sobre las VM consolidadas en una PM idéntica puede calculase clásicamente como la aceleración (*speedup*):

$$S(N) = \frac{T_{VM}(N)}{T_{PM}(N)} \tag{1}$$

Evidentemente las PM homogéneas paralelas son más rápidas que una sola PM paralelizando por virtualización. El factor de mejora será mayor o menor dependiendo sobre todo de los recursos de los servidores, relativo al tamaño de la subtarea ejecutable de tamaño Tm/N, y las características de virtualización del servidor.
Un caso particular es la aceleración que se produce con un solo servidor debido al software de virtualización:

$$S(1) = \frac{T_{VM}(1)}{T_{PM}(1)} > 1 \tag{2}$$

Así podemos definir la sobrecarga del software de virtualización como O_v, necesario para poder tener al menos una VM en una PM (aunque a efectos funcionales de un servidor, sea poco útil). Expresado en tiempos medios de ejecución:

$$S(1) = \frac{T_{VM}(1)}{T_{PM}(1)} = \frac{T_{PM}(1) + OV_v}{T_{PM}(1)} = 1 + \frac{OV_v}{T_{PM}(N)} \tag{3}$$

Es decir, que el tiempo de ejecución de la carga de trabajo en la PM es el patrón sobre el que se calcula la aceleración y OV_v es independiente de N.
De forma análoga, podemos expresar la aceleración de N PM sobre N VM consolidadas en una PM, expresada en tiempos medios de ejecución:

$$S(N) = \frac{T_{VM}(N)}{T_{PM}(N)} = \frac{T_{PM}(N) + OV_v + OV_c(N)}{T_{PM}(N)} \tag{4}$$

Donde $OV_c(N)$ es la sobrecarga que supone tener más de una VM consolidada en la misma PM. Esa sobrecarga es debida a las interacciones de la gestión de las VM por parte del VMM y cualquier tiempo adicional sobre la ejecución de la carga de trabajo, debido a que las VM se ejecutan en paralelo en una sola PM.
Por tanto, en el método se definen tres tiempos de ejecución comunes a la virtualización: el tiempo de ejecución en la máquina física (PM) equivalente o carga de trabajo efectiva (*workload*), la sobrecarga fija de la capa de virtualización, cuyo valor se reparte entre todas las VM y la sobrecarga variable dependiente del número de VM consolidadas en una PM, es decir, servidores virtuales ejecutándose paralelamente en un solo servidor físico (ver Figura 4 [5]).

Fig. 4: Descomposición del tiempo $T_{VM}(N)$

Esta definición de la aceleración permite expresar-

la separando partes de la aceleración:

$$S(N) = 1 + \frac{OV_v}{T_{PM}(N)} + \frac{OV_c(N)}{T_{PM}(N)} \quad (5)$$

Si expresamos la aceleración en fracciones, tendremos tres partes, la parte paralela de trabajo útil (que como Amdahl expresaría mejora un 100 %, y por tanto paraleliza linealmente), la fracción correspondiente al OV_v, que notaremos γ, y la fracción correspondiente al OV_c multiplicado por el número de máquinas, que notaremos como δ:

$$S(N) = \frac{1}{\frac{1}{N} + \gamma + N\delta} = \frac{N}{1 + N(\gamma + N\delta)} \quad (6)$$

Donde Nson el número de máquinas en paralelo tanto PM como VM. Para el caso particular de la aceleración de una PM sobre una VM alojada en una PM idéntica (N=1), se puede despejar γ, ya que sólo hay OV_v y no hay OV_c,

$$S(1) = \frac{1}{1+\gamma} \quad (7)$$

Luego:

$$\gamma = \frac{1}{S(1)} - 1 \quad (8)$$

Sustituyendo este valor en $S(N)$, operando se despeja δ.

$$\delta = \frac{\frac{\frac{N}{S(N)} - 1}{N} - \gamma}{N} = \frac{(\frac{1}{S(N)} - \frac{1}{N}) - \gamma}{N} \quad (9)$$

Expresado con aceleraciones y N máquinas:

$$\delta = \frac{(\frac{1}{S(N)} - \frac{1}{N}) - (\frac{1}{S(1)} - 1)}{N} \quad (10)$$

Sin duda la fórmula de δ, recuerda vagamente a como determinar α en la ley de Amdahl, o Gustaffson, y el uso de dos parámetros a la USL de Gunther.
En definitiva, si se dispone de los valores de S(1), S(N) y $T_{PM}(N)$, se pueden calcular las sobrecargas OV_v y OV_c. Fijémonos que siempre que haya software de virtualización $S(1) > 1$, luego $\gamma < 1$. Recordemos que es debido a que establecimos la aceleración $S(N)$ como la que se produce de las N PM paralelas sobre las N VM paralelas en una sola PM, y que incluso con S(1) ya hay OV_v. Sin embargo δ=0, para N=1 puesto que para S(1) no hay consolidación de VM, es decir $OV_c(1) = 0$, mientras que para $N > 1$ y $\delta > 0$ ya se produce sobrecarga de $OV_c(N)$.
A partir de ahora iremos ilustrando la formulación propuesta y los conceptos a explicar con diferentes ejemplos reales en sistemas a prueba (*System Under Test, SUT*). En la tabla I se muestra un ejemplo de los valores de los tiempos de ejecución, $T_{PM}(N)$ y $T_{VM}(N)$, medidos en segundos (por razones de espacio solo se muestran 4 decimales significativos, pero los cálculos se han hecho con 8 dígitos de precisión) con carga ejecutada del benchmark Sysbench y un tamaño de 100K números primos ($T_m = 100K$), en una PM de 16 CPU y 8 GB de RAM.

En la figura 5, se muestran las aceleraciones correspondientes en la tabla I. Se puede observar que inicialmente las PM paralelas aceleran superlinealmente, al principio, y casi linealmente hasta S(5) = 4.7772, cayendo luego hasta un valle con aceleraciones entre 2.5 y 2.0, donde las VM consolidadas se hacen algo más eficientes, para que las PM vuelvan remontar después en S(11) = 3.5378, cuando las carga de trabajo se han dividido subtareas paralelas mucho menores, es decir, que son tan pequeñas que el OV_c ya pesa mucho en el tiempo de ejecución $T_{VM}(N)$.
Es muy interesante observar que la caída de aceleración comparada de N PM contra N VM, como ocurre al dividir la carga de $T_m = 100K$ entre N=6, las subtareas se ajustan mucho mejor a la consolidación que en tamaños anteriores. Sobre ello se redundará en las subsecciones posteriores para determinar cuándo consolidar o no, desde el punto de vista de esa aceleración comparada. En la Figura 6, se muestran

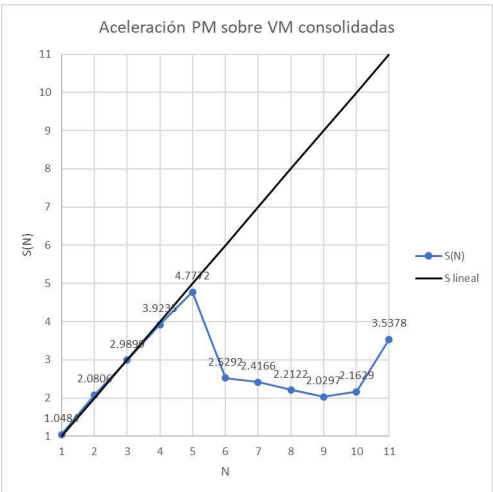


Fig. 5: Aceleración de N PM paralelas sobre consolidación de N VM

distintos tamaños Tm del benchmark y su incidencia en las aceleraciones S(N) para las 9 primeras máquinas del ejemplo de la tabla I. Se evidencia que cuanto mayor es Tm, mayor linealidad toman las aceleraciones S(N). De forma coloquial, podríamos decir que las aceleraciones de las PM sobre las VM consolidadas se van "estirando" hasta llegar a la linealidad con tamaños mayores.

Es muy interesante observar los valores que van tomando $S(N)$ y δ para indicar cómo va variando la eficiencia de la consolidación y la escalabilidad de la misma, con el número incremental de máquinas (N), tal como se apreciará en las siguientes subsecciones.

B. Eficiencia e Isoeficiencia

Una vez conocida la aceleración $S(N)$ de las PM paralelas sobre las VM consolidadas, se puede establecer la eficiencia por PM:

$$E(N) = \frac{S(N)}{N} \quad (11)$$

En el caso de una sola máquina, la eficiencia de las PM sobre las VM es superlineal debido a la sobre-

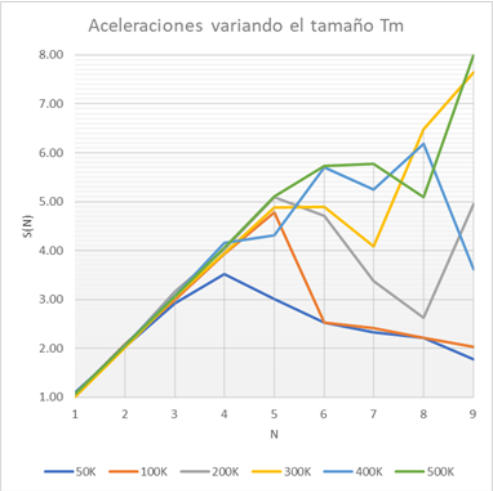


Fig. 6: Aceleraciones de N PM paralelas sobre consolidación de N VM variando el tamaño del problema T_m

Tabla I: Tiempos de Ejecución, Sobrecargas y Fracciones de Sobrecarga

$T_{PM}(N)$	$T_{VM}(N)$	OV_v/N	$OV_c(N)$	γ	δ
24.2532	25.4260	1.1728	0.0000	-0.0461	0.0000
9.2707	19.2885	0.0241	9.9694	-	0.0133
5.2972	15.8380	0.0161	10.4924	-	0.0157
3.5626	13.9777	0.0120	10.3667	-	0.0127
2.6253	12.5415	0.0096	9.8678	-	0.0110
2.0386	5.1560	0.0080	3.0690	-	0.0458
1.5668	3.7863	0.0069	2.1711	-	0.0452
1.3093	2.8964	0.0060	1.5387	-	0.0466
1.1296	2.2928	0.0053	1.1148	-	0.0475
1.0060	2.1759	0.0048	1.1215	-	0.0408
0.8890	3.1450	0.0043	2.2077	-	0.0216

carga OV_v.

$$E(1) = S(1) = \frac{1}{1+\gamma} > 1 \tag{12}$$

Esto no es una anomalía, sino una característica al establecer la aceleración de una PM frente a esa misma PM con una sobrecarga (por empaquetar una VM), puesto que la misma produce ineficiencia. En cualquier caso, la eficiencia para cualquier número N de máquinas se obtiene dividiendo por N su aceleración:

$$E(N) = \frac{1}{1+N(\gamma+N\delta)} \tag{13}$$

Y análogamente:

$$E(N) = \frac{1}{N} + \frac{OV_v}{NT_{PM}(N)} + \frac{OV_c(N)}{NT_{PM}(N)} \tag{14}$$

Se puede calcular la relación el valor de la isoeficiencia $C(N)$ para un número N de VM consolidadas en una PM [11] como:

$$C(N) = \frac{\frac{1}{S(N)}}{1-\frac{1}{S(N)}} \tag{15}$$

y por tanto el tiempo de ejecución de la PM se puede relacionar con las sobrecargas mediante,

$$T_{PM}(N) = C(N)[OV_v + OV_c(N)] \tag{16}$$

Tabla II: Aceleración, Eficiencia por PM, Isoeficiencia, Eficiencia de las Tareas y su Tiempo de Ejecución en PM

S(N)	E(N)	C(N)	$E_{VM}(N)$	T_{PM} (calculado)
1.0483	1.0483	20.6797	0.9538	24.2532
2.0805	1.0402	0.9254	0.4806	9.2707
2.9898	0.9966	0.5025	0.3344	5.2972
3.9234	0.9808	0.3420	0.2548	3.5626
4.7771	0.9554	0.2647	0.2093	2.6253
2.5291	0.4215	0.6539	0.3953	2.0386
2.4165	0.3452	0.7059	0.4138	1.5668
2.2121	0.2765	0.8249	0.4520	1.3093
2.0297	0.2255	0.9711	0.4926	1.1296
2.1629	0.2162	0.8599	0.4623	1.0060
3.5377	0.3216	0.3940	0.2826	0.8890

Si tomamos $T_{PM}(N)$ como el tiempo de ejecución útil (efectiva), que OV_v es el tiempo de ejecución inútil de sobrecarga, por añadir una capa de software que permita paralelizar VM, y OV_c es el tiempo de ejecución inútil atribuible a la sincronización, comunicación y otros retrasos para disponer de N VM paralelas en una sola PM, podemos establecer la eficiencia de las tareas de ejecución de tamaño T_m en N máquinas como [11]:

$$E_{VM}(N) = \frac{T_{PM}(N)}{T_{VM}(N)} = \frac{T_{PM}(N)}{T_{PM}(N) + OV_v + OV_c(N)} \tag{17}$$

En la tabla II se muestran la aceleración, eficiencia por PM, isoeficiencia, eficiencia del benchmark y el tiempo de ejecución de las PM, a partir de las sobrecargas calculadas, para el mismo ejemplo de del servidor de la tabla I. El T_{PM} se ha calculado con la fórmula de la isoeficiencia, a modo de demostración.

C. Escalabilidad

En el caso de la virtualización, podemos definir que la consolidación es escalable para un determinado número de máquinas N, si la eficiencia $E(N)$ se mantiene constante. Como se ha observado en la tabla II, la eficiencia, y por ello $C(N)$, no es una constante. De hecho, lo que más llama la atención de la consolidación es cómo se comporta la aceleración, y por tanto la eficiencia por PM y la eficiencia de la ejecución de las subtareas en las VM.
En la Figura 6 pudimos observar la gráfica de diente de sierra típica en la aceleración de las PM paralelas sobre las VM consolidadas en una PM idéntica. Tal como se puede observar, la aceleración pasa por una serie de máximos y mínimos locales a medida que aumenta N (dependiendo de Tm/N).
De hecho, los valores cartesianos de $S(N)$, N y el origen, forman un triángulo rectángulo, donde la arcotangente equivale al ángulo agudo (en radianes) correspondiente a la razón entre los catetos opuesto y adyacente, es decir la eficiencia $E(N)$:

$$\theta = tan^{-1}(\frac{S(N)}{N}) = tan^{-1}(E(N)) \tag{18}$$

Transformando el ángulo θ de la arcotangente de la eficiencia de radianes a grados, se visibiliza cómo

Tabla III: Aceleración, Eficiencia por PM, Linealidad, Fracción de Sobrecarga δ y Eficonsolidación

N	$S(N)$	$E(N)$	θ^o	δ	EC(N)
1	1.0483	1.0483	46.3523	0.0000	0.0000
2	2.0805	1.0402	46.1313	0.0133	0.0040
3	2.9898	0.9966	44.9032	0.0157	0.0172
4	3.9234	0.9808	44.4466	0.0127	0.0168
5	4.7771	0.9554	43.6943	0.0110	0.0185
6	2.5291	0.4215	22.8569	0.0458	0.1044
7	2.4165	0.3452	19.0460	0.0452	0.1004
8	2.2121	0.2765	15.4572	0.0466	0.0964
9	2.0297	0.2255	12.7091	0.0475	0.0914
10	2.1629	0.2162	12.2046	0.0408	0.0832
11	3.5377	0.3216	17.8286	0.0216	0.0660

va cambiando la linealidad de la aceleración $S(N)$ con distintas consolidaciones de N VM.

$$\theta^o = tan^{-1}(E(N)) \cdot \frac{180}{\pi} \tag{19}$$

Este ángulo θ^o permite determinar la calidad de la consolidación debido al valor de la eficiencia de las PM. De este modo podemos clasificar las consolidaciones:

- Si $\theta^o \geq 45$, las N PM paralelas son más eficientes que la consolidación de N VM, puesto que $E(N) \geq 1$. Entonces, las N PM aceleran linealmente, e incluso superlinealmente sobre las VM consolidadas. Los tamaños del problema Tm/N aún son demasiado grandes para ejecutarse en paralelo con una buena eficiencia en las N VM, comparado con las N PM.
- Si $\theta^o < 45$, las PM paralelas son menos eficientes con el incremento de N, puesto que su aceleración es infralineal, puesto que $E(N) < 1$. Las N VM consolidadas pueden ser una solución alternativa a las PM, para paralelizar subtareas de tamaño Tm/N, en términos de $S(N)$ y $E_{VM}(N)$.

En la tabla III se muestran los distintos valores de θ^o para las consolidaciones de la figura 5, así mismo se incluyen los valores de $S(N)$, $E(N)$ y δ. Fijémonos en la incidencia de N en el ángulo a medida que nos alejamos del origen de coordenadas.

D. Consolidación óptima

Observando las distintas N consolidaciones para una misma PM, la determinación de la consolidación óptima de N VM en la misma PM pasa por las siguientes consideraciones:

- Cuanto más alto es el valor de N, mayor número de máquinas físicas paralelas o virtuales consolidadas en una instalación. En el caso de N PM en paralelo, para un centro de datos, se presentan varias desventajas de carácter no funcional. Entre otras, el aumento del coste, espacio y potencia eléctrica con el aumento de N. Estas desventajas, se tornan ventajas para N VM consolidadas, pero a cambio de mayor tiempo de ejecución (debido a la propia virtualización y la sobrecarga de la consolidación) y en consecuencia mayor energía consumida, por esa mayor duración del tiempo de ejecución [16]. Se trata pues de encontrar un equilibrio (*trade-off*) en el intercambio de tiempo de ejecución y por tanto energía de las VM, a cambio de potencia, espacio y coste de las PM.
- Sin embargo, usar más VM consolidadas de las necesarias también produce el fenómeno conocido como *sprawling*, en los centros de datos, produciendo gestión y mantenimiento adicionales.

Teniendo en cuenta las dos consideraciones anteriores, para un tamaño T_m de carga de trabajo:

- ¿Cómo establecer el número óptimo de PM en paralelo?
- ¿Cómo establecer el número óptimo de VM consolidadas en una PM, idéntica a las anteriores?

Para responder a la primera cuestión, podemos fijarnos en el ejemplo de la figura 5 o la tabla II, para darnos cuenta puede haber varios máximos locales de la aceleración $S(N)$. De hecho, si la PM tiene suficientes recursos, al ir dividiendo la carga de trabajo en subtareas cada vez más pequeñas en paralelo, llega un momento que la aceleración de N PM máquinas paralelas puede ser muy alta por acumulación de $OV_c(N)$ en las N VM. Por tanto, en general, seleccionar el número de PM paralelas que tengan mayor aceleración puede conducirnos a elegir un N elevado, con los perjuicios antes abordados.
Llamaremos $N*$ al número de máquinas mínimo que produce la máxima aceleración $S(N^*)$ (máximo local) de las máquinas PM paralelas sobre sus correspondientes virtuales VM consolidadas. Por tanto, N^* es hasta dónde podría ser más interesante desplegar PM paralelas y así repartir en subtareas, dado un problema de tamaño T_m en N^* PM. Podría decirse que $N*$ corresponde al límite de paralelismo de las PM para reducir el tiempo de ejecución por máquina, al dividir T_m entre $N*$, con un rendimiento, espacio y potencia eléctrica a consumir óptimos.
Puede haber máximos locales $S(N)$ más altos con, pero a costa de tener menor eficiencia. En el ejemplo de la figura 5, N^* corresponde a 5 PM, con una aceleración de 4.7771 sobre las 5 VM consolidadas en una PM, un ángulo θ^o de 43.69°, correspondiente a una eficiencia por PM de 0.95, además de que δ es el valor (positivo) más bajo de la muestra (ver tabla III). Es posible que haya valores δ incluso más bajos, y aceleraciones más altas (ver repunte al final de la gráfica), pero a costa de $N \gg N*$ y $\theta^o \ll 45$ (es decir con muchas más PM paralelas y eficiencias más bajas).
El proceso para determinar N^*, sería encontrar el máximo local que todavía acelera casi linealmente, o lo que es lo mismo su eficiencia por PM es $E(N) \approx 1$. Fijémonos que con T_m muy grandes eso puede llevarnos a seleccionar un N^* alto (gran cantidad de PM paralelas). En consecuencia, para conocer hasta dónde paralelizar, con PM frente a VM, no basta

sólo con observar la aceleración, sino tomar el máximo local con el número de máquinas mínimo, para ello nos sirve $E(N)$.
Para contestar a la segunda cuestión, es decir, para elegir la consolidación de VM óptima, se podría elegir la aceleración mínima, puesto que es cuando las PM paralelas son menos eficientes que las VM paralelas. Siguiendo tabla II, $S(9)$ es la mínima aceleración de las PM sobre las VM, con un $C(9)$ máxima y consecuentemente una $E_{VM}(9)$ también máximo (sin considerar $S(1)$ puesto que no hay consolidación). Sin embargo, se deben seleccionar suficientes máquinas para una eficiencia óptima, pero no demasiadas VM para no producir *sprawling*. Para ello hemos definido el indicador que hemos denominado eficonsolidación $EC(N)$:

$$EC(N) = \frac{[E(1) - E(N)]}{N} = \frac{[NS(1) - S(N)]}{N^2} \quad (20)$$

En la tabla III se pueden ver distintos valores de EC(N). Así denominaremos N^+ al número de VM a consolidar óptimo, ya que $EC(N^+)$ maximiza la diferencia entre aceleraciones con sobrecarga y minimiza el número de máquinas consolidadas.

$$EC(N^+) = max(\frac{[E(1) - E(N)]}{N}) \quad (21)$$

Continuando con el ejemplo de la figura 7, N^+ corresponde a 6 VM con una con una aceleración de solo 2.5292 de 6 PM paralelas sobre las 6 VM consolidadas, un ángulo θ^o de 22.85˚, correspondiente a una eficiencia de 0.42 y el valor de *eficonsolidación* más alto $EC(N^+)$ de la muestra. También es uno de los valores de δ más altos, aunque no el máximo (ver tabla III). De hecho, hay valores δ incluso más altos, y aceleraciones más bajas, tal como hemos señalado, por ejemplo, S(9)=2.0297, pero a costa consolidar 3 VM adicionales. De todos modos, valores cercanos a $EC(N^+)$ son opciones secundarias a considerar por un administrador de centro de datos.

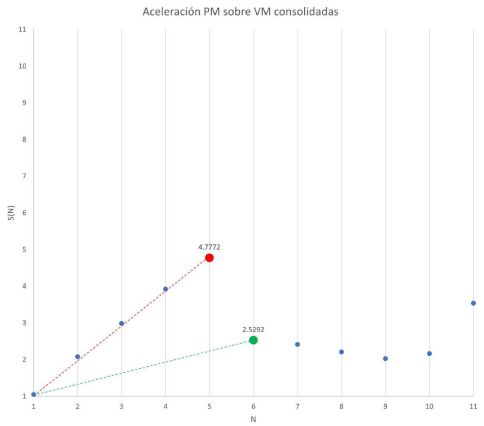


Fig. 7: N^*, N^+ sus ángulos respecto a la aceleración lineal.

E. Isoconsolidación

Cuando distintas PM disponen de distintos recursos (en nuestro caso CPU y memoria) sus tiempos de ejecución suelen ser distintos, aún consolidando del mismo modo, utilizando el mismo *benchmark* y el mismo VMM. Sin embargo, puede pasar que N^* y/o N^+ sean similares. A este fenómeno lo hemos denominado *isoconsolidación*.

Tabla IV: Aceleración, Eficiencia por PM, Eficiencia VM, Linealidad y Eficonsolidación (Tm=100K)

SUT	Dell T430	Dell T330	Ryzen 7	Lenovo ST550	Fujitsu RX500
#CPU	16	8	12	20	48
Procesador	Intel Xeon E5-2600 v4	Intel Xeon E3-1200 v6	AMD Ryzen 7 5800X	Intel Xeon Platinum v2	Intel Xeon E7-4800 v3
RAM (GB)	8	16	32	128	1024
S(N*)	4.7771	11.5259	4.9033	3.7081	4.6381
N*	5	10	5	4	5
E(N*)	0.9554	1.1525	0.9806	0.9270	0.9276
θ^o en $N*$	43.6943	49.0548	44.4405	42.8320	42.8499
δ en $N*$	0.0110	0.0018	0.0182	0.0109	0.0089
$S(N^+)$	2.5291	11.5239	2.4975	1.6604	2.2715
N^+	6	12	7	9	8
$E(N^+)$	0.4215	0.9603	0.3567	0.1844	0.2839
$EC(N^+)$	0.1044	0.0060	0.1055	0.0933	0.0932
θ^o en N^+	22.8569	43.8407	19.6358	10.4533	15.8518
$\delta en N^+$	0.0458	0.0025	0.0492	0.0572	0.0430

En la tabla IV se comparan valores de PM con distintos recursos de CPU y RAM. Observando $S(N^*)$, $S(N^+)$, N^* y N^+, se pueden determinar algunas consolidaciones iguales. Por ejemplo, podemos ver que consolidar N^*= 5 VM es óptimo para casi todas las PM, excepto una, para el tamaño de problema seleccionado. También se confirma que el benchmark es más hambriento en CPU que en RAM, puesto que los servidores T430 y T330 son idénticos excepto en esos recursos invertidos. También se observa que el T330 no es apto para consolidar VM, frente al T430 (doble número de CPU mitad de N^* y N^+), para este tamaño de problema ($T_m = 100K$).

En la figura 8, se muestran las aceleraciones de las PM sobre las VM consolidadas hasta 10 máquinas, para los 5 SUT experimentados.

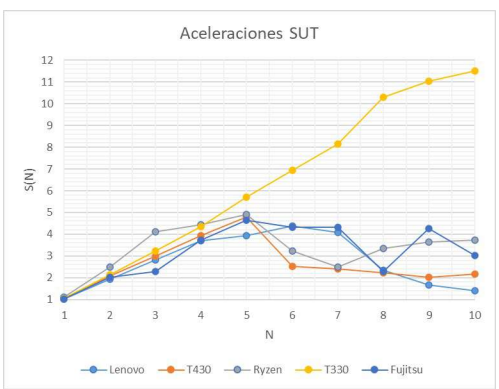


Fig. 8: Aceleración de distintas PM sobre consolidación de N VM.

En la figura 9 podemos ver que, si el tamaño del problema es la mitad, el Dell T330 empieza a comportarse de forma similar a los demás SUT seleccionados. Aunque la eficiencia de la consolidación de las N VM mejora, no hay isoconsolidación con los otros SUT, puesto que N^*= 8 y N^+ = 10. Por lo tanto, reduciendo Tm a la mitad solo han descendido tanto N^* y N^+ dos máquinas, respectivamente. Parece que no es la memoria la que limita el rendimiento de la consolidación, sino el número de CPU para un tamaño de problema dado, al comparar el servidor Dell T330 con el servidor Dell T430, ejecutando el *benchmark* elegido.

Seleccionando tamaños distintos, en el ejemplo mostrado en la figura 5, se puede observar la sen-

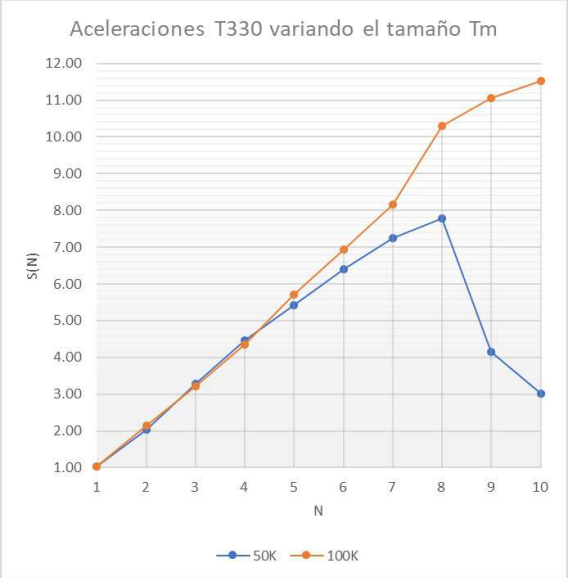


Fig. 9: Aceleración del T330 con distintos tamaños del problema

sibilidad al tamaño del problema en N^* y N^+ (ver tabla V). En definitiva, al repartirse una mayor carga de trabajo por máquina, las PM paralelas van ganando eficiencia sobre las VM consolidadas, y viceversa. En correspondencia, hacer el tamaño más grande aumenta el ángulo de linealidad θ^o en N^*, y lo contrario en N^+, mientras que δ va cayendo con el aumento de N.

Tabla V: Aceleración, Eficiencia, Linealidad y Eficonsolidación variando el tamaño del problema

	Dell T430			
T_m	10K	50K	100K	200K
S(N*)	4.4929	3.5138	4.7771	5.0974
N*	5	4	5	5
E(N*)	0.8985	0.8784	0.9554	1.0194
$E_{VM}(N*)$	0.2225	0.2845	0.2093	0.1961
θ^o en $N*$	41.9422	41.2977	43.6943	45.5530
δ en N*	0.0153	0.0296	0.0110	0.0009
$S(N^+)$	1.1990	2.5253	2.5291	2.6271
N^+	2	6	6	8
$E(N^+)$	0.5995	0.4208	0.4215	0.3283
$E_{VM}(N^+)$	0.8339	0.3959	0.3953	0.3806
EC(N^+)	0.2288	0.1118	0.1044	0.0850
θ^o en N^+	30.9446	22.8262	22.8569	18.1800
δ en N^+	0.1940	0.0522	0.0458	0.0330

F. Sobrecargas añadidas

En subsecciones anteriores, se ha propuesto un método general para determinar y cuantificar la sobrecarga en las consolidaciones de servidores, específicamente, en la consolidación de máquinas virtuales de tipo I. En la tabla VI, se muestra un ejemplo de los diferentes tiempos de ejecución variando N, para virtualizaciones de tipo I y tipo II para el servidor Dell T430, que es el que hemos tomado como ejemplo.
En el caso del tipo II, añadir niveles de software ralentiza aún más los tiempos de ejecución (de hecho, se consumen los recursos CPU y RAM más que en el tipo I), y en el caso de los contenedores se haría mucho mayor para tamaños de carga de trabajo grandes (ver Figura 1).
Sin embargo, el tipo de virtualización no impide aplicar la formulación de las secciones anteriores, puesto que sólo hay que considerar la nueva sobrecarga fija del sistema operativo añadida a OV_v, es decir variando γ. En la tabla VII, se puede observar, desplegar hipervisores de tipo II en reparto temporal, agota los recursos rápidamente por la sobrecarga añadida del sistema operativo, no siendo posible desplegar tantas VM, el alcance de N se reduce pues, pero toda la formulación es igualmente aplicable (ver tabla VII).

Tabla VI: Aceleración, Eficiencia por PM, Linealidad, Fracción de Sobrecarga δ y Eficonsolidación en Virtualización tipo II

$S(N)$	$E(N)$	θ^o	$EC(N)$	γ	δ
1.0808	1.0808	47.2260	0.0000	-0.0748	0.0000
2.1399	1.0699	46.9363	0.0054	-	0.0210
3.0436	1.0145	45.4141	0.0221	-	0.0233
3.9488	0.9872	44.6309	0.0234	-	0.0195
5.1746	1.0349	45.9833	0.0091	-	0.0136
4.3358	0.7226	35.8532	0.0597	-	0.0231

Tabla VII: Tiempos de Ejecución con distintos tipos de Virtualización para el Servidor DELL T430 (T=100K).

N	T_{PM}	T_{VM-I}	T_{VM-II}
1	24.2532	25.4260	26.2150
2	9.2707	19.2885	19.8390
3	5.2972	15.8380	16.1230
4	3.5626	13.9777	14.0680
5	2.6253	12.5415	13.5850
6	2.0386	5.1560	8.8390
7	1.5668	3.7863	-
8	1.3093	2.8964	-
9	1.1296	2.2928	-

V. Experimentación

En secciones anteriores, propusimos un método general para determinar y la sobrecarga, aceleración, eficiencias y nuevos conceptos para las consolidaciones de servidores, específicamente, en la consolidación de máquinas virtuales con cargas intensivas de CPU.
Cada sistema bajo prueba (SUT) ejecuta la carga de trabajo del benchmark Sysbench y su comportamiento es monitoreado. Todos los tiempos están medidos en segundos. Comparamos la ejecución de la carga de trabajo entre la máquina física PM y el servidor consolidado VM. Variamos el número de VM que están alojados en la PM, y toda la carga de trabajo se distribuye uniformemente entre el conjunto de servidores (división temporal de carga en subtareas iguales). Por lo tanto, la sobrecarga en la virtualización se calcula comparando la ejecución de la carga de trabajo que se equilibra entre N servidores físicos (PM), con la ejecución de una carga de trabajo idéntica equilibrada entre N servidores virtuales (VM), alojados en el mismo servidor físico (ver Figura 3).
Para demostrar el contenido teórico de la aceleración,

eficiencia, isoeficiencia, y los conceptos creados por nosotros, como *eficonsolidación* e *isoconsolidación*, usamos un extenso conjunto de valores obtenidos de experimentos. La configuración experimental se compone de 5 tipos de servidores físicos que utilizan la mayoritariamente la familia de CPU Intel Xeon (pero no la única): un Dell PowerEdge T430, con 16 CPU físicas, 8 GB de RAM y Ubuntu Server 16.04 como el sistema operativo, un Dell PowerEdge T330, con 8 CPU físicas, 16 GB de RAM y Ubuntu Server 16.04 como el Sistema operativo, lo mismo que para un Lenovo ST550 con 20 CPU con 128GB de RAM y un Fujitsu RX5000 de 48 CPU con 1024 GB de RAM. También se ha experimentado con procesadores AMD, concretamente un Ryzen 7 (ver tabla IV). Para la virtualización, implementamos KVM como hipervisor Tipo I y Virtual Box como Tipo II. A todas las máquinas virtuales se les asigna la misma cantidad de CPU que el servidor físico, 1 GB de RAM virtual, y Ubuntu Server 16.04 como sistema operativo invitado. Las cargas de trabajo ejecutadas son del benchmark Sysbench, los cuales son intensivos en CPU (cálculo de números primos). Es importante señalar que en este trabajo la carga de trabajo ejecutada solicita el 100 % utilización de la CPU física, que representa la saturación de la CPU. Todos los experimentos se han realizado con reparto temporal y no espacial de los recursos. Hay que tener en cuenta que estamos cuantificando las sobrecargas del sistema como un todo (todos sus componentes) bajo una carga de trabajo intensiva de CPU ejecución [17]. Aunque el uso de los contenedores comprende la virtualización más liviana, esta investigación no se refiere al despliegue, puesta en marcha, etc., o cualquier otra actividad de máquinas virtuales o contenedores. Este trabajo de investigación se centra en método para cuantificar la sobrecarga relativa durante el tiempo de ejecución de la carga de trabajo de aplicaciones intensivas de CPU [18]. Los tiempos de ejecución de las PM y VM se han obtenido realizando múltiples ejecuciones, para todas las configuraciones, obteniendo tiempos medios de ejecución con una desviación estándar menor al 5 % Es decir, los experimentos se realizaron una cantidad específica de veces que asegura la significación estadística.

VI. Discusión

Esta investigación se ha centrado en utilizar las leyes de la aceleración en paralelismo, para determinar el grado de consolidación óptimo para servidores virtuales, utilizando cargas de intensivas de CPU y en menor grado memoria RAM. No se han tenido en cuenta otro tipo de cargas, hipervisores o *benchmarks* para la experimentación que los mencionados. No obstante, hemos experimentado con otros *benchmarks* de CPU (stress-ng y SPEC CPU) y aunque cuantitativamente los resultados son distintos para distintas cargas y sobrecargas, la formulación sería aplicable igualmente.

Puesto que el estudio se basa en aceleraciones de una máquina sobre ella misma, con la misma carga, el mismo hipervisor y las mismas condiciones, la comparación es justa. Como es natural, si se consideran los tiempos de ejecución, cada máquina tiene un rendimiento diferente en función de sus recursos, sobre todo de CPU, frente a una carga de trabajo de un tamaño determinado. Así, servidores con pocos recursos toleran bien la virtualización si la carga es baja, y al revés, los servidores con más recursos toleran las sobrecargas mejor, para un tamaño determinado.

Precisamente la carga de trabajo se reparte uniformemente en subtareas paralelas, pero podría haberse escalado en multitareas concurrentes, puesto que el resultado sería análogo tal como se describe en [15].

Toda la investigación se ha realizado con servidores homogéneos, para poder establecer el grado de virtualización de una forma comparable, no solo entre distintas opciones de la misma PM, sino entre PM distintas. La heterogeneidad de máquinas se ha estudiado anteriormente, así como su anidamiento, por ejemplo, usando contenedores dentro de VM consolidadas en PM tal como se describe en [10], pero usando tiempos de ejecución y no aceleraciones, ni eficiencias, como en este trabajo de investigación.

VII. Conclusiones y trabajo futuro

Este trabajo propone el uso de conceptos básicos de paralelismo, adaptarlos para cuantificar y representar las sobrecargas de consolidación de máquinas virtuales en máquinas físicas. La formalización planteada se ha experimentado extensivamente, con distintas máquinas físicas y distintos tamaños de tareas para verificar el planteamiento teórico. También se ha intentado reflejar gráficamente, de forma intuitiva, la distancia al ideal de paralelismo lineal, cuando se usan las herramientas software para paralelizar máquinas. De forma literal, las VM proveen de la virtualidad de desplegar más máquinas de las que realmente se disponen físicamente. No obstante, tienen ventajas inherentes, como el ahorro de espacio, menor necesidad de enfriamiento por máquina, descenso de la potencia eléctrica desperdiciada, a cambio de sobrecargas, que provoca un aumento de la energía por un mayor tiempo de ejecución.

Los resultados de esta investigación permiten establecer hasta dónde paralelizar un servidor físico y hasta dónde consolidar máquinas virtuales de una forma óptima, pero adaptable a las circunstancias que los administradores de sistemas soportan en los centros de datos. Es decir, que se pueden fijar qué cantidad de eficiencia de consolidación se desea, para no sobrepasar un acuerdo de nivel de servicio, con respecto a los tiempos de ejecución deseados.

Tal como se refleja en el texto, la investigación aquí descrita es fácilmente aplicable a otros tipos de virtualización, siempre que se trate de máquinas completas y no procesos, como los contenedores. La tecnología de virtualización impulsa la consolidación de VM o contenedores. A pesar de su funcionalidad similar, existen diferencias significativas entre ellos en términos de rendimiento (medido por el tiempo medio de ejecución), seguridad, implementación y por-

tabilidad. Estas diferencias afectan las decisiones de consolidación al elegir entre máquinas virtuales o contenedores, y su grado de consolidación. Tradicionalmente, los servidores se consolidan mediante la asignación de máquinas virtuales o contenedores a un servidor físico. Sin embargo, la combinación de máquinas virtuales y contenedores en el mismo servidor físico puede mitigar los inconvenientes de ambos. En consecuencia, también es posible consolidar primero los contenedores en máquinas virtuales y luego consolidar estas máquinas virtuales en máquinas físicas. En un trabajo anterior, los autores propusieron un método para cuantificar la magnitud del tiempo de sobrecarga de consolidación a partir el tiempo de medio de ejecución de una tarea en la PM, comparándolo con un número (N), o bien de VM, o bien de contenedores. Posteriormente, se generalizó el método de estimación del tiempo de sobrecarga de consolidación de servidores para cualquier combinación (configuración) de consolidación arbitraria de VM y contenedores, independientemente de su configuración o anidamiento.

Es uno de los trabajos futuros de esta investigación, profundizar en el anidamiento de contenedores y consolidación. Este trabajo es parte de un proyecto que pretende estandarizar las medidas de rendimiento (y energía) en consolidación de máquinas virtuales desplegadas en máquinas físicas para poder establecer una comparativa efectiva, del mismo modo que ya se puede realizar con los servidores físicos. El estado del arte, en este sentido, señala que es crucial determinar el rendimiento de la sobrecarga para poder establecer si la energía añadida por un rendimiento menor compensa el ahorro de potencia por virtualizar. Los indicadores al uso de los servidores físicos deberán entonces modificarse, en consonancia con la consolidación, para poder hacer esa comparativa. También eso es parte del trabajo futuro de esta investigación. Aunque evidentemente queda fuera del objetivo de este trabajo, en un caso de estudio realizado con máquinas virtuales en una empresa real, se ha extendido el análisis con un tipos de carga transaccional de CPU y memoria, cuyas prestaciones se han medido con otras métricas basadas en teoría de colas. Los resultados no contradicen, sino complementan lo aquí expuesto.

VIII. Agradecimientos

Este trabajo forma parte del proyecto TED2021-132695B-I00, financiado por MCIN / AEI / 10.13039 / 501100011033 y por la Unión Europea "NextGenerationEU" / PRTR. También agradecemos a la Universidad de Sevilla y al Hasso-Plattner Institute, por el uso de sus SUT en distintos proyectos de I+D.

Referencias

[1] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al., "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, no. 13, pp. 2009, 2009.

[2] Wenting Wang, Haopeng Chen, and Xi Chen, "An availability-aware virtual machine placement approach for dynamic scaling of cloud applications," in *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*. IEEE, 2012, pp. 509–516.

[3] Belen Bermejo, Carlos Juiz, and Carlos Guerrero, "Virtualization and consolidation: a systematic review of the past 10 years of research on energy and performance," *The Journal of Supercomputing*, vol. 75, no. 2, pp. 808–836, 2019.

[4] Maik Lindner, Fiona McDonald, Barry McLarnon, and Philip Robinson, "Towards automated business-driven indication and mitigation of vm sprawl in cloud supply chains," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE, 2011, pp. 1062–1065.

[5] Belen Bermejo and Carlos Juiz, "On the classification and quantification of server consolidation overheads," *The Journal of Supercomputing*, vol. 77, no. 1, pp. 23–43, 2021.

[6] MinSu Chae, HwaMin Lee, and Kiyeol Lee, "A performance comparison of linux containers and virtual machines using docker and kvm," *Cluster Computing*, vol. 22, no. Suppl 1, pp. 1765–1775, 2019.

[7] Prashant Ramchandra Desai, "A survey of performance comparison between virtual machines and containers," *Int. J. Comput. Sci. Eng*, vol. 4, no. 7, pp. 55–59, 2016.

[8] John Paul Martin, A Kandasamy, and K Chandrasekaran, "Exploring the support for high performance applications in the container runtime environment," *Human-centric Computing and Information Sciences*, vol. 8, pp. 1–15, 2018.

[9] Nikolaus Huber, Marcel von Quast, Fabian Brosig, Michael Hauck, and Samuel Kounev, "A method for experimental analysis and modeling of virtualization performance overhead," in *Cloud Computing and Services Science*. Springer, 2012, pp. 353–370.

[10] Belen Bermejo and Carlos Juiz, "A general method for evaluating the overhead when consolidating servers: performance degradation in virtual machines and containers," *The Journal of Supercomputing*, vol. 78, no. 9, pp. 11345–11372, 2022.

[11] Kai Hwang and Naresh Jotwani, *Advanced computer architecture: parallelism, scalability, programmability*, vol. 199, McGraw-Hill New York, 1993.

[12] Gene M Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.

[13] John L Gustafson, "Fixed time, tiered memory, and superlinear speedup," in *Proceedings of the Fifth Distributed Memory Computing Conference (DMCC5)*. IEEE Press, 1990, pp. 1255–1260.

[14] Yuan Shi, "Reevaluating amdahl's law and gustafson's law," *Computer Sciences Department, Temple University (MS: 38-24)*, 1996.

[15] NJ Gunther, "Guerrilla capacity planning: A tactical approach to planning for highly scalable applications and services. 2007," *Google Scholar Google Scholar Digital Library Digital Library*.

[16] Carlos Juiz and Belen Bermejo, "The c i s 2: a new metric for performance and energy trade-off in consolidated servers," *Cluster Computing*, vol. 23, no. 4, pp. 2769–2788, 2020.

[17] Emiliano Casalicchio, "A study on performance measures for auto-scaling cpu-intensive containerized applications," *Cluster Computing*, vol. 22, no. 3, pp. 995–1006, 2019.

[18] Rajdeep Dua, A Reddy Raja, and Dharmesh Kakadia, "Virtualization vs containerization to support paas," in *2014 IEEE International Conference on Cloud Engineering*. IEEE, 2014, pp. 610–614.

Desarrollo de una infraestructura para la investigación en *cloud computing*

Lucía Pons[1], Salvador Petit[1], Julio Pons[1], María E. Gómez[1] y Julio Sahuquillo[1]

Resumen— **Los sistemas en la nube despliegan una amplia variedad de recursos compartidos y albergan un gran número de aplicaciones. Para realizar investigación dentro de este ámbito, se suele emplear un sistema reducido que oculta la enorme complejidad de los sistemas reales y proporciona flexibilidad para gestionar los recursos. A pesar de ser más sencillos y a menor escala, estos sistemas experimentales deben incluir los principales componentes (*hardware* y *software*) de los sistemas de computación en la nube o *cloud computing* para proporcionar resultados representativos.**

Este trabajo presenta Stratus, una plataforma experimental que se está utilizando actualmente para llevar a cabo investigación en *cloud computing*. A diferencia de otras plataformas existentes, Stratus es la única plataforma que ofrece conjuntamente todas las características principales de estos sistemas: uso de máquinas virtuales (MV) para ejecutar las aplicaciones de los usuarios, despliegue de los tres tipos de nodos de la nube (servidor, cliente y almacenamiento) y gestión de los principales recursos compartidos del sistema (CPU, espacio de caché, ancho de banda de memoria, red y disco). Además, Stratus implementa un gestor de *software* para facilitar la investigación y ayudar al diseño de políticas orientadas a mejorar la calidad de servicio.

Palabras clave— ***Cloud computing*, gestión de recursos compartidos, virtualización, plataforma experimental**

I. Introducción

Los sistemas en la nube o *cloud* albergan una amplia variedad de aplicaciones de los usuarios, por lo que necesitan ofrecer altas capacidades de computación y almacenamiento. Para ello, y a medida que el *cloud* evoluciona con el tiempo, las plataformas *cloud* constan de una variedad de nodos, cada uno compuesto por un conjunto de recursos. Para mejorar el rendimiento y reducir costes, la mayoría de estos recursos (por ej., núcleos, memoria principal y almacenamiento) se comparten entre las aplicaciones de los usuarios. En la nube pública, para proporcionar aislamiento y privacidad a los usuarios, sus aplicaciones se ejecutan en máquinas virtuales (MV) [1]. Estas características hacen que las plataformas *cloud* sean sistemas complejos de implantar, no sólo desde el punto de vista del *hardware*, sino también desde la perspectiva del *software*, ya que necesitan soportar la virtualización del *hardware* real. Además, deben proporcionar funcionalidades para satisfacer los requisitos de los sistemas *cloud*, como la eficiencia de los recursos, el cumplimiento de los acuerdos de nivel de servicio (SLA) [2] y el soporte para servir a múltiples clientes.

[1]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: {lupones,spetit,jpons,megomez,jsahuqui}@disca.upv.es

Para poder llevar a cabo investigación en el ámbito del *cloud computing*, muchas empresas desarrollan una plataforma experimental a pequeña escala para reducir la complejidad de los sistemas reales, proporcionar flexibilidad, y controlar la carga de trabajo. Por ejemplo, permite comprobar si se puede incumplir el SLA al aplicar una política de gestión de recursos y sacar conclusiones antes de implantarla en una plataforma real. Sin embargo, desarrollar una plataforma experimental de este tipo es todo un reto, ya que debe ser capaz de proporcionar resultados representativos. Para ello, debe ofrecer tres características principales: i) incluir los principales tipos de nodos (servidor, cliente y almacenamiento), ii) proporcionar aislamiento con máquinas virtuales a las aplicaciones de los usuarios, y iii) ofrecer la capacidad de gestionar los principales recursos compartidos (por ej., memoria principal, caché de último nivel, red, ...).

En los últimos años se han desarrollado múltiples plataformas o sistemas experimentales para llevar a cabo investigaciones en *cloud computing*. Sin embargo, la mayoría de ellas están compuestas por una única máquina [3], [5], [6], [10], [11], no proporcionan virtualización [3], [5], [6], o no consideran la gestión de componentes importantes como la red [3], [5], [6] y el almacenamiento remoto [5], [6], [7], [8], [9].

En este trabajo se presenta Stratus, una plataforma experimental que cumple con las tres características mencionadas anteriormente y que está siendo utilizada actualmente para llevar a cabo investigación en *cloud computing* [12], [13]. En cuanto al *hardware*, Stratus incluye tres tipos principales de nodos: un nodo servidor que aloja las aplicaciones de los usuarios, un nodo cliente que lanza peticiones al servidor y un nodo de almacenamiento. De este modo, se tienen en cuenta las latencias de red. En cuanto al *software*, Stratus implementa la pila de *software* completa (por ejemplo, QEMU, Libvirt, ...) para proporcionar aislamiento a las aplicaciones de los usuarios o clientes mediante el uso de MV.

Un componente importante de Stratus es el *manager*, el *software* que integra tres funcionalidades principales: 1) la gestión y el control de la ejecución de las MV y las aplicaciones que albergan, 2) la monitorización de los contadores de prestaciones y la utilización de los recursos del sistema, y 3) el reparto de los principales recursos compartidos del sistema usando tecnologías disponibles en las máquinas experimentales.

Por último, Stratus soporta la ejecución tanto de cargas de trabajo que siguen el modelo cliente-servidor (por ejemplo, TailBench [14], CloudSuite

Tabla I: Resumen de la infraestructura experimental utilizada en los trabajos orientados a la gestión de recursos.

Trabajo	**Año**	**MV?**	**Tipo de nodo**			**Gestión de recursos**				
			Servidor	Cliente	Almacenamiento	CPU	LLC	Memoria (BW)	Disco	Red
ServerMore [3]	2021	✓	✓	×	✓	✓	✓	✓	×	×
Skynet [4]	2021	×, contenedores	✓	✓	✓	✓	×	×	✓	✓
Alita [5]	2020	✓	✓	×	×	✓	✓	✓	×	×
CLITE [6]	2020	×	✓	×	×	✓	✓	✓	✓	×
PARTIES [7]	2019	×, contenedores	✓	✓	×	✓	✓	✓	✓	✓
Scavenger [8]	2018	✓	✓	✓	×	✓	✓	×	×	✓
Vertical Elasticity [9]	2018	×, contenedores	✓	✓	×	✓	✓	×	✓	✓
Stratus	2023	✓	✓	✓	✓	✓	✓	✓	✓	✓

[15]) como de cargas de trabajo *best-effort* (por ejemplo, stress_ng [16], iperf [17] o SPEC CPU [18]).

II. Soluciones actuales

A medida que los sistemas *cloud* evolucionan con el tiempo, el número y la variedad de aplicaciones, así como las herramientas disponibles, aumentan continuamente. Esta situación ha impulsado a los investigadores a crear nuevos sistemas para llevar a cabo la investigación.

A gran escala, se han construido sistemas como Grid'500 [19], Cloudlab [20] y Chamaleon [21] para permitir a los investigadores realizar experimentos en sistemas distribuidos geográficamente. En estas plataformas experimentales, los usuarios solicitan recursos durante un tiempo limitado. Estos recursos pueden ser configurados para su uso (por ej., desplegar una pila de *software* personalizada).

Este trabajo se centra en los sistemas experimentales diseñados para la investigación a pequeña escala, sin depender de sistemas externos. La tabla I resume las principales características (si soportan virtualización con MV, el tipo de nodos que incluye la plataforma y los recursos que permite gestionar) de un subconjunto representativo de sistemas experimentales. Para facilitar la comparación, la última fila de la tabla muestra las características de plataforma experimental Stratus propuesta en este trabajo. Como se puede observar, ningún trabajo relacionado ha hecho uso de una plataforma experimental que incluya todas las características incluidas en Stratus.

Stratus tiene nodos de tipo servidor, cliente y almacenamiento, utiliza MV para ejecutar las aplicaciones de los usuarios y permite gestionar (monitorizar y repartir) los principales recursos del sistema. Como puede verse en la tabla, sólo el sistema utilizado para evaluar Skynet [4] incluye los tres tipos de nodos que despliega Stratus. Algunas propuestas como [3] ejecutan servidor y clientes en la misma máquina mientras que otras propuestas [5], [6] utilizan plataformas experimentales de un solo nodo. Esto implica que no se tiene en cuenta la interferencia que causa la red.

En cuanto a la virtualización, solo tres de los trabajos relacionados utilizan MV. Dos de ellos [3], [8] también utilizan Linux KVM para desplegar MV como en este trabajo. Algunos trabajos utilizan contenedores, que permiten obtener un mejor rendimiento a costa de un peor aislamiento. Existen herramientas *software* para lograr aislar los recursos cuando se hace uso de contenedores (por ejemplo, cgroups). Sin embargo, los contenedores se ven obligados a utilizar el mismo kernel que la máquina física o el *host*, por lo que el aislamiento no es posible a nivel de kernel.

En cuanto a la gestión de los recursos compartidos, sólo PARTIES [7] tiene en cuenta todos los recursos compartidos de forma similar a Stratus, pero la plataforma usada en PARTIES carece de un nodo de almacenamiento y hace uso de contenedores en lugar de MV. De todos los recursos, la CPU (los núcleos del procesador) es el único recurso considerado en todos los trabajos. Por otro lado, la red y el disco son los recursos compartidos menos estudiados.

Aparte de las plataformas analizadas en la tabla I, existen otros trabajos de investigación que hacen uso de plataformas experimentales, pero estas se centran en estudiar un recurso especifico. Less Provisioning [22] y Twig [11] se centran en la asignación de recursos de la CPU, ajustando dinámicamente los recursos de la CPU en función de la utilización y los valores recogidos por los contadores de prestaciones *hardware*, respectivamente. ReTail [10] también se centra en los recursos de la CPU, pero gestiona estos recursos ajustando la frecuencia de la CPU. QWin [23] se ideó para garantizar la latencia de cola como objetivo de nivel de servicio (SLO) de los servidores de almacenamiento distribuido mediante la repartición de los núcleos entre las aplicaciones de los usuarios. Por último, LIBRA [24] propone una plataforma experimental para la gestión dinámica del ancho de banda de la memoria principal.

III. Visión general de Stratus

La figura 1 presenta el esquema de la plataforma experimental de Stratus. Stratus está compuesto de tres nodos principales: servidor, cliente y almacenamiento. El nodo servidor actúa como la parte del servidor en las aplicaciones cliente-servidor. Este nodo ejecuta las MV que alojan las aplicaciones de servidor, que son gestionadas por el *software manager*, que se encarga de gestionar los recursos y las aplicaciones en Stratus (más detalles en la Sección IV). El nodo cliente es un nodo auxiliar que emula el comportamiento de los clientes mediante la ejecución de aplicaciones cliente que realizan peticiones a las MV en el nodo servidor. Por último, el nodo de almace-

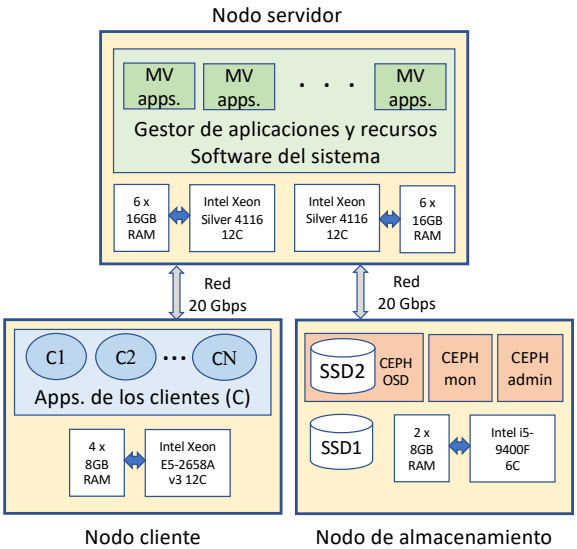


Fig. 1: Esquema de la plataforma experimental Stratus.

Tabla II: Especificaciones *hardware* de los nodos.

Nodo	**Procesador**		**Memoria**
	Modelo	**#Núcleos (#Hilos)**	
Servidor	2x Intel Xeon Silver 4116	48 (96)	12 x DDR4-2666 16GB DIMMs
Cliente	Intel E5-2658A	12 (24)	1 x 32GB DIMM
Almacenamiento	Intel i5-9400F	6 (6)	2 x DDR4-2666 16GB DIMMs

namiento proporciona recursos de almacenamiento remoto para las MV. Los nodos cliente y de almacenamiento están interconectados al nodo servidor con enlaces dedicados de 20 Gbps.

El diseño de Stratus se ha realizado en base a dos ejes principales: el *hardware* desplegado y el *software* del sistema. Esta sección presenta y motiva las opciones de diseño que se han tomado para cada eje.

A. Hardware utilizado

Tipos y número de nodos.
Una decisión de diseño clave es seleccionar los tipos y número de nodos que componen la plataforma experimental. Por un lado, debe evitarse la enorme complejidad de gestionar un elevado número de nodos que se encuentran en entornos reales. Por otro lado, los resultados proporcionados deben ser representativos de escenarios reales. Un sistema en nube incluye dos tipos principales de nodos: nodos de computación y nodos de almacenamiento. Dentro de los nodos de computación, algunos deben de emular el comportamiento de los clientes para aquellas aplicaciones que siguen el modelo cliente-servidor. Teniendo en cuenta estas dos cuestiones, se ha optado por minimizar el número de nodos, incluyendo un nodo por cada uno de los tipos de nodo existentes en entornos *cloud* reales.

Especificaciones del *hardware*. En cuanto a los nodos de servidor y almacenamiento, deben ser representativos de los nodos típicos de los sistemas

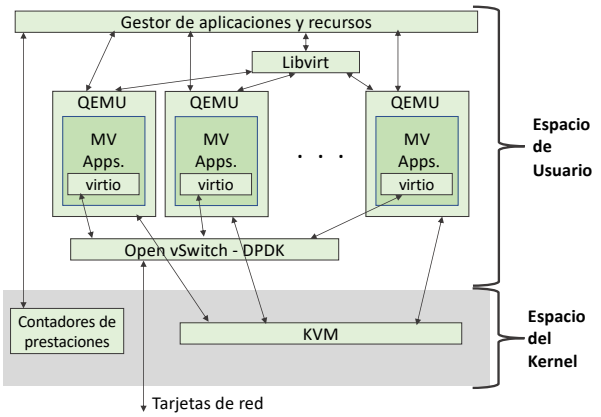


Fig. 2: *Software* del sistema Stratus en el nodo servidor.

cloud. La tabla II muestra las especificaciones (procesador y memoria principal) de cada uno de los nodos. Las especificaciones de los nodos de servidor y almacenamiento pueden considerarse representativas de los nodos implementados en sistemas de nube reales. Por ejemplo, Google Cloud [25], Amazon EC2 [26] Huawei Elastic Cloud [27] utilizan procesadores Intel Xeon Scalable que incluyen de decenas a cientos de GB de capacidad de memoria principal.

En cuanto a los medios de almacenamiento en el nodo de almacenamiento (mostrado en la Figura 1), se compone de dos discos SSD. El primero (SSD1) almacena el sistema operativo del nodo de almacenamiento y el *software* del sistema, mientras que el segundo (SSD2) de 960 GB se dedica exclusivamente a actuar como almacenamiento remoto para las aplicaciones que se ejecutan en las MV del nodo servidor. Para ello, se hace uso de Ceph [28], un *software* de almacenamiento distribuido de código abierto que se instala habitualmente en entornos de nube.

Por último, los nodos cliente y de almacenamiento están interconectados al nodo servidor con dos enlaces dedicadas de 20 Gbps. Más concretamente, el nodo servidor dispone de dos tarjetas de red de 20 Gbps (doble puerto, 10 Gbps por puerto) que se conectan a los nodos cliente y de almacenamiento.

B. Software del sistema

Para diseñar el entorno experimental de Stratus, se han analizado los principales componentes de OpenStack [29]. OpenStack es una plataforma de tecnología para computación en nube de código abierto. En la actualidad, es popularmente utilizada para la gestión de servicios virtuales tanto en nubes públicas como privadas. OpenStack es un sistema complejo, con múltiples componentes que soportan diferentes tipos de dispositivos (por ejemplo, dispositivos de red, dispositivos de almacenamiento, etc.) de múltiples proveedores. Para evitar lidiar con tal complejidad, construimos un sistema más simple que incluye los principales componentes de *software* que se pueden encontrar en un despliegue típico de OpenStack. La mayor simplificación radica en los niveles superiores de *software*, cuyo objetivo es reducir la com-

plejidad de la gestión, pero que tienen un impacto insignificante o nulo en el rendimiento del sistema y gestión de recursos.

La figura 2 presenta los principales componentes *software* del nodo servidor de Stratus y sus interacciones. Estos componentes, que se encargan de gestionar las MV y las interconexiones de red, se describen a continuación.

Infraestructura de las MV. La ejecución y gestión de las MV implica una compleja pila de *software*, en la que se distinguen tres niveles principales: el hipervisor, el gestor de virtualización y el sistema operativo de MV y las aplicaciones que se ejecutan. El hipervisor es el sistema operativo instalado en la máquina física. Actualmente se utiliza en la industria un amplio conjunto de hipervisores tanto de código abierto como propietarios. Ejemplos de hipervisores de código abierto son Linux KVM [30] y Xen [31]. El primero es una de las tendencias actuales de la industria, y está siendo utilizado por Amazon [32] y Google [33]. Este último también cuenta con el apoyo de Amazon. El gestor de virtualización se refiere a la plataforma de *software* que gestiona los recursos de *hardware* de la máquina física y los distribuye entre las MV. Un ejemplo de gestor de virtualización es Libvirt [34]. Algunos virtualizadores, como QEMU [35], también soportan tanto KVM como Xen. Por último, el sistema operativo de la MV y las aplicaciones de los usuarios. Pueden ser tanto propietarios como de código abierto (por ejemplo, una distribución de servidor Linux que ejecute varios servicios de Internet).

***Software* de red.** Para interconectar las máquinas virtuales con las tarjetas de interfaz de red físicas (NIC) del nodo servidor, se utiliza un *conmutador virtual*. El conmutador virtual se configura con Open vSwitch (OvS) [36]. Las NICs emuladas en las MV (es decir, las NICs virtio [37]) y cada NIC física (ambos puertos) en el nodo servidor se acceden desde el conmutador virtual a través de Data Plane Development Kit (DPDK) [38]. DPDK permite la transferencia directa de paquetes entre las NIC virtio y las NIC físicas, sin pasar por la pila de red del kernel del sistema operativo anfitrión. Esta configuración aumenta el rendimiento de la red en comparación con el mecanismo de reenvío de paquetes por defecto implementado en el kernel de Linux.

IV. *Manager*: gestor de recursos y aplicaciones de Stratus

Proporcionar un entorno experimental que permita automatizar la configuración y ejecución de experimentos es crucial a la hora de llevar a cabo investigación. El objetivo del gestor de recursos y aplicaciones de Stratus, referido como *manager*, es ayudar al investigador en esta tarea, proporcionándole una interfaz amigable. Para ello, implementa tres funciones principales i) gestionar y controlar la ejecución de una o varias MV, cada una de las cuales ejecuta una aplicación, ii) monitorizar los contadores de prestaciones *hardware* y la utilización de los recursos del sistema, iii) repartir los recursos del sistema y asignarlos a las MV.

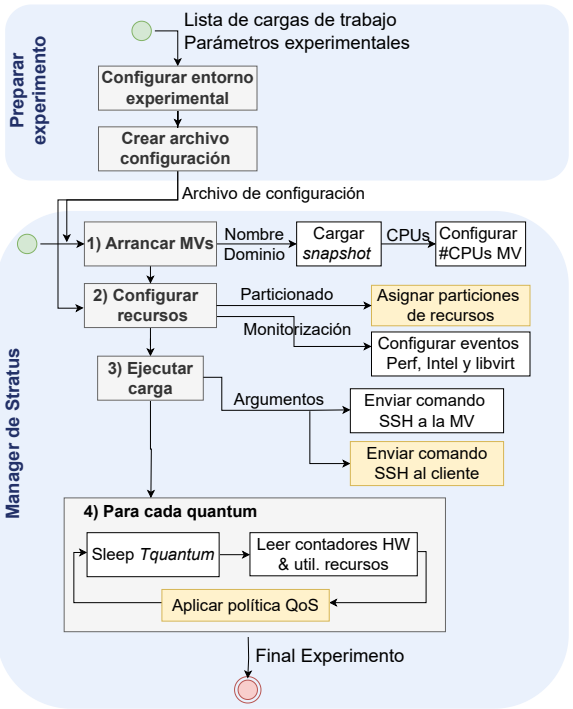


Fig. 3: Diagrama de bloques de las acciones para ejecutar un experimento con el *manager* de Stratus. Las cajas amarillas representan las acciones opcionales.

A. *Ejecución de experimentos*

Para ilustrar cómo el *manager* realiza los experimentos, la figura 3 muestra un diagrama de bloques con los principales pasos realizados al lanzar una o más MV junto con las aplicaciones que se ejecutarán en ellas (*parejas MV-aplicación*). A continuación, cada uno de los pasos se discute en detalle.

A.1 Preparación del experimento

Configuración del entorno experimental. Para fijar las condiciones del experimento, se debe configurar el entorno experimental. Por ejemplo, habilitar o deshabilitar el *prefetcher*, fijar la frecuencia del procesador para evitar la variabilidad entre experimentos, o sincronizar los relojes de las máquinas cliente y servidor usando el protocolo NTP (Network Time Protocol).

Crear archivo de configuración. Antes de ejecutar el experimento, es necesario definir la carga de trabajo (es decir, las MV y las aplicaciones que se ejecutarán en ellas) y las condiciones experimentales (por ej., eventos de prestaciones a monitorizar, duración del *quantum*). Estas se definen mediante plantillas MAKO [39]. En este punto también se puede definir si una MV se le asigna una porción de uno o varios recursos. Estas plantillas con las cargas de trabajo a ejecutar y todos los parámetros del experimento se usan para generar el fichero de configuración del experimento que usa el *manager* para lanzarlo.

A.2 Funciones realizadas por el *manager*

Preparar las MV para su ejecución. El primer paso que realiza el *manager* es preparar y arrancar las MV definidas en el fichero de configuración. Se arrancan y se dejan en un estado *estable*, es decir, listas para ejecutarse. Una vez arrancadas las VMs, se puede modificar el número de núcleos asignados a cada MV (es decir, vCPU) en caso de que se vaya a ejecutar una aplicación paralela (multihilo o multiproceso) y se requiera más de un núcleo.

Configuración de la monitorización y partición de recursos. Con QEMU, cada vCPU se asocia con un identificador de proceso o PID del sistema operativo de la máquina física. Estos PID son necesarios para monitorizar los contadores de prestaciones con Perf individualmente para cada núcleo (es decir, vCPU) de la MV. Del mismo modo, la monitorización del ancho de banda de memoria y LLC se realiza también a nivel de PID. El resto de recursos, red y ancho de banda de disco se monitorizan a nivel de MV. El *manager* también permite asignar a cada MV una parte de un o varios recursos compartidos. Por lo tanto, si está especificado en el archivo de configuración, se asigna la cuota del recurso a la MV.

Inicio de la ejecución de aplicaciones en las MV. Cuando las MV están operativas y listas para empezar a ejecutar las aplicaciones, se envía un comando SSH a cada MV para iniciar la ejecución de la aplicación. El *manager* está adaptado para soportar la ejecución de aplicaciones cliente-servidor (por ejemplo, la *suite* de *benchmarks* TailBench), así como cargas de trabajo *best-effort* (por ej., la *suite* de *benchmarks* de SPEC CPU). En el caso de una carga de trabajo cliente-servidor, se envía un comando SSH al nodo cliente para iniciar la ejecución de los clientes, que envían peticiones al servidor (ya en ejecución).

Realizar acciones en cada *quantum*. Una vez iniciada la ejecución, el *manager* ejecuta el *bucle principal* (ver Figura 3) durante el resto del tiempo de ejecución. En cada *quantum* o intervalo de tiempo (indicada en la configuración), se recopilan datos de los contadores de prestaciones y los recursos monitorizados. Además, el *manager* está adaptado para permitir implementar y aplicar políticas para garantizar y mejorar la calidad de servicio (QoS). Por ejemplo, políticas que gestionen el uso compartido de recursos entre MV [7], [8], [5], predigan interferencias entre MV [9], [6], [10], [13], [40] o programen MV [41].

Fin de la ejecución. El bucle principal finaliza cuando el *manager* detecta que todas las MV han terminado de ejecutar sus aplicaciones, momento en el que apaga las MV en ejecución. Todos los datos recogidos de los contadores de prestaciones y recursos del sistema se almacenan en ficheros CSV, listos para ser procesados. Además, con fines de caracterización y depuración, también se recopilan estadísticas y datos dentro de las MV. Por ejemplo, en las cargas de trabajo de Tailbench, los clientes informan de datos como la latencia de las peticiones.

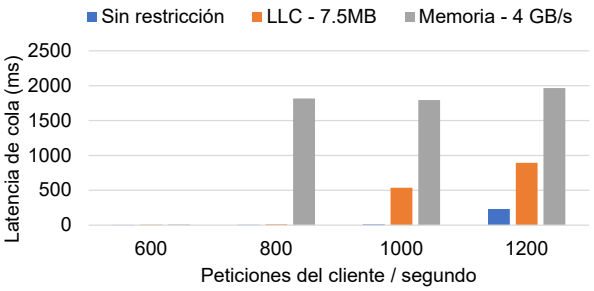


Fig. 4: Efecto de la reducción del espacio de la caché de último nivel (LLC) y la restricción del consumo del ancho de banda de memoria en la latencia de cola (percentil 95) de la aplicación *cloud* `img-dnn` de Tailbench.

B. Gestión de los principales recursos compartidos

Las aplicaciones de los usuarios compiten por los recursos compartidos en los sistemas *cloud*. Esto significa que el rendimiento de una determinada aplicación (o MV) dependerá de las aplicaciones con las que se ejecute al mismo tiempo. En otras palabras, de la parte del recurso que sea capaz de utilizar. En consecuencia, saber en qué medida el rendimiento de una aplicación dada se ve afectado al variar la cantidad de recursos asignados es una información de valor para los proveedores *cloud*.

En los últimos años, los procesadores de servidor se han dotado de tecnologías avanzadas que permiten monitorizar y repartir los principales recursos del sistema. A modo de ejemplo, la figura 4 compara cómo la latencia de cola (percentil 95) de la aplicación *cloud* de latencia crítica `img-dnn` de Tailbench [14] se ve afectada ante tres eventos independientes: i) sin restricción, ii) limitar el espacio de caché (7.5 MB de 16.5 MB) y iii) limitar en ancho de banda de memoria a 4 GB/s. Como se puede observar, al limitar el consumo de estos dos recursos a `img-dnn`, la latencia de cola crece al aumentar el número de peticiones por encima de 800, especialmente si se limita el ancho de banda de memoria, lo que hace que soporte menos peticiones antes de saturarse.

A continuación, se explica cómo se implementa la monitorización y partición de cada recurso compartido en Stratus sin depender de ninguna herramienta externa.

Utilización de la CPU. La utilización de la CPU (núcleo) representa el porcentaje de tiempo que una CPU está activa. Es una métrica crucial en entornos *cloud*, ya que en los sistemas productivos la utilización de la CPU es baja (menos del 20 %) la mayor parte del tiempo [42], [43], y por lo tanto, muchas estrategias de aprovisionamiento de recursos [44], [8], [22] persiguen incrementar la utilización de la CPU, a fin de realizar un mejor aprovechamiento de los recursos. Para obtener la utilización de cada CPU, utilizamos los datos recogidos del archivo `/proc/stat`, que informa de las estadísticas sobre la actividad del kernel agregada desde que el sistema arranca por primera vez. Para asignar las vCPU de las MV a los núcleos lógicos de la máquina física, Stratus utiliza la API de Libvirt [34].

Caché de último nivel (*last level cache* o LLC). Debido a la alta latencia de acceso a la memoria principal en caso de fallo de la LLC, está caché es uno de los recursos compartidos más críticos en los procesadores multinúcleo actuales. Recientemente, algunos fabricantes de procesadores como Intel han desarrollado tecnologías que permiten monitorizar y gestionar la LLC. En los procesadores Intel, estas tecnologías se conocen como Cache Monitoring Technology (CMT) y Cache Allocation Technology (CAT) [45]. La distribución de la LLC (a nivel de vías de caché) se realiza mediante clases de servicio (CLOS), que pueden definirse como grupos de aplicaciones (PID) o como grupos de núcleos lógicos a los que se asigna una partición de la LLC.

Ancho de banda de memoria. El ancho de banda de la memoria puede afectar considerablemente al rendimiento o la capacidad de respuesta de las aplicaciones. Por ejemplo, en un sistema de servidor con diferentes MV que acceden a la memoria principal, la interferencia entre MV puede crecer significativamente y hacer que las MV más sensibles a la memoria rindan por debajo de un nivel aceptable, comprometiendo la calidad de servicio. Los procesadores recientes de Intel de la familia Xeon Scalable introducen la Memory Bandwidth Allocation Technology (MBA) [46], que permite distribuir el ancho de banda de memoria entre las aplicaciones en ejecución. De forma similar a CAT, MBA funciona utilizando CLOS. Es decir, los límites de ancho de banda de MBA sólo se aplican a CLOS, a los que el usuario puede asignar tareas (PID) o núcleos.

Ancho de banda del disco. Muchas cargas de trabajo operan con archivos de *big data* o bases de datos que no pueden cargarse completamente en la memoria principal. En consecuencia, estas cargas de trabajo dependen constantemente del sistema de E/S para acceder al disco y cargar/almacenar los datos requeridos. La supervisión y partición de este recurso es, por tanto, de gran interés. El acceso de E/S a los discos puede monitorizarse utilizando la herramienta virsh o la API de Libvirt. Ambos mecanismos ofrecen la misma funcionalidad y permiten monitorizar el número de operaciones o bytes y la duración de las operaciones de lectura y escritura.

Ancho de banda de red. Las MV que se ejecutan en la misma máquina física comparten recursos de red cuyo ancho de banda y latencia desempeñan un papel importante en la calidad de servicio de las aplicaciones de usuario. En consecuencia, los recursos de red deben ser monitorizados y distribuidos para minimizar las interferencias entre MV. El número de paquetes de red o bytes que pasan por una interfaz de red puede monitorizarse con la API de Libvirt.

V. Conclusiones

La investigación en sistemas *cloud* es cada vez más popular. Sin embargo, antes de implantar soluciones en sistemas *cloud* públicos, la investigación y las pruebas deben realizarse en plataformas experimentales controladas. Las soluciones existentes hacen uso de plataformas experimentales para evaluar su trabajo, pero estas plataformas no incluyen todas las principales características de los sistemas de nube reales (tipos de nodos, virtualización, gestión de recursos) y, por tanto, no proporcionan resultados representativos.

Este trabajo presenta Stratus, una plataforma experimental utilizada para llevar a cabo investigación y experimentos controlados en el ámbito del *cloud computing*. A diferencia de las plataformas experimentales utilizadas en trabajos existentes, Stratus cumple con todas las características de los entornos *cloud* en términos de despliegue de *hardware* y *software*. Además, Stratus implementa un gestor de aplicaciones y recursos que asiste al investigador en la ejecución de experimentos y en la gestión de recursos, lo cual es clave para diseñar políticas orientadas a la calidad de servicio que mitiguen las interferencias entre MV.

Agradecimientos

Este trabajo ha sido financiado por Huawei Cloud, por el Ministerio de Universidades mediante la beca FPU18/01948, por el Ministerio de Ciencia e Innovación junto al FEDER europeo a través de los proyectos PID2021-123627OB-C51 y TED2021-130233B-C32.

Referencias

[1] Jyotiprakash Sahoo, Subasish Mohapatra, and Radha Lath, "Virtualization: A Survey on Concepts, Taxonomy and Associated Security Issues," in *Proceedings of ICCNT*, 2010, pp. 222–226.

[2] Damián Serrano, Sara Bouchenak, Yousri Kouki, Frederico Alvares de Oliveira Jr., Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana Arantes, and Pierre Sens, "SLA guarantees for cloud services," *Future Generation Computer Systems*, vol. 54, pp. 233–246, 2016.

[3] Amoghavarsha Suresh and Anshul Gandhi, "ServerMore: Opportunistic Execution of Serverless Functions in the Cloud," in *Proceedings of SoCC*, 2021, pp. 570–584.

[4] Yannis Sfakianakis, Manolis Marazakis, and Angelos Bilas, "Skynet: Performance-driven Resource Management for Dynamic Workloads," in *Proceedings of CLOUD*, 2021, pp. 527–539.

[5] Quan Chen, Shuai Xue, Shang Zhao, Shanpei Chen, Yihao Wu, Yu Xu, Zhuo Song, Tao Ma, Yong Yang, and Minyi Guo, "Alita: Comprehensive Performance Isolation through Bias Resource Management for Public Clouds," in *Proceedings of SC20*, 2020, pp. 1–13.

[6] Tirthak Patel and Devesh Tiwari, "CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical Jobs for Warehouse Scale Computers," in *Proceedings of HPCA*, 2020, pp. 193–206.

[7] Shuang Chen, Christina Delimitrou, and José F. Martínez, "PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services," in *Proceedings of ASPLOS*, 2019, pp. 107–120.

[8] Seyyed Ahmad Javadi, Amoghavarsha Suresh, Muhammad Wajahat, and Anshul Gandhi, "Scavenger: A Black-Box Batch Workload Resource Manager for Improving Utilization in Cloud Environments," in *Proceedings of SoCC*, 2019, pp. 272—285.

[9] Shashank Shekhar, Hamzah Abdel-Aziz, Anirban Bhattacharjee, Aniruddha Gokhale, and Xenofon Koutsoukos, "Performance Interference-Aware Vertical Elasticity for Cloud-Hosted Latency-Sensitive Applications," in *Proceedings of CLOUD*, 2018, pp. 82–89.

[10] Shuang Chen, Angela Jin, Christina Delimitrou, and José F. Martínez, "ReTail: Opting for Learning Simplicity to Enable QoS-Aware Power Management in the Cloud," in *Proceedings of HPCA*, 2022, pp. 155–168.

[11] Rajiv Nishtala, Vinicius Petrucci, Paul Carpenter, and Magnus Sjalander, "Twig: Multi-Agent Task Management for Colocated Latency-Critical Cloud Services," in *Proceedings of HPCA*, 2020, pp. 167–179.

[12] Lucía Pons, Josué Feliu, José Puche, Chaoyi Huang, Salvador Petit, Julio Pons, María E. Gómez, and Julio Sahuquillo, "Effect of Hyper-Threading in Latency-Critical Multithreaded Cloud Applications and Utilization Analysis of the Major System Resources," *Future Generation Computer Systems*, vol. 131, pp. 194–208, 2022.

[13] Lucía Pons, Josué Feliu, Julio Sahuquillo, María E. Gómez, Salvador Petit, Julio Pons, and Chaoyi Huang, "Cloud White: Detecting and Estimating QoS Degradation of Latency-Critical Workloads in the Public Cloud," *Future Generation Computer Systems*, vol. 138, pp. 13–25, 2023.

[14] H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, 2016, pp. 1–10.

[15] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi, "Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware," *Proceedings of ASPLOS*, 2012.

[16] Canonical Ltd, "Ubuntu manpage: stress-ng," Available at `https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html`, 2020, Accessed: 2022-11-20.

[17] ESnet, NLANR, DAST, "iPerf tool for network bandwidth measurements," Available at `https://iperf.fr/`, 2020, Accessed: 2022-11-20.

[18] "Why would a cloud computing company use the SPEC CPU2017 benchmark suite?," Available at `https://www.spec.org/cpu2017/publications/DO-case-study.html`, 2017, Accessed: 2019-08-02.

[19] Sébastien Badia, Alexandra Carpen-Amarie, Adrien Lèbre, and Lucas Nussbaum, "Enabling large-scale testing of IaaS cloud platforms on the grid'5000 testbed," in *Proceedings of the 2013 International Workshop on Testing the Cloud*, 2013, pp. 7–12.

[20] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, et al., "The Design and Operation of CloudLab," in *USENIX Annual Technical Conference*, 2019, pp. 1–14.

[21] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S Gunawi, Cody Hammock, et al., "Lessons learned from the chameleon testbed," in *Proceedings of USENIX ATC*, 2020, pp. 219–233.

[22] Binlei Cai, Keqiu Li, Laiping Zhao, and Rongqi Zhang, "Less Provisioning: A Hybrid Resource Scaling Engine for Long-Running Services With Tail Latency Guarantees," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1941–1957, 2022.

[23] Liuying Ma, Zhenqing Liu, Jin Xiong, and Dejun Jiang, "QWin: Core Allocation for Enforcing Differentiated Tail Latency SLOs at Shared Storage Backend," in *Proceedings of ICDCS*, 2022, pp. 1098–1109.

[24] Ying Zhang, Jian Chen, Xiaowei Jiang, Qiang Liu, Ian M. Steiner, Andrew J. Herdrich, Kevin Shu, Ripan Das, Long Cui, and Litrin Jiang, "LIBRA: Clearing the Cloud Through Dynamic Memory Bandwidth Management," in *Proceedings of HPCA*, 2021, pp. 815–826.

[25] "Google Cloud Compute Engine - CPU platforms [online]," Available at `https://cloud.google.com/compute/docs/cpu-platforms`, 2022, Accessed: 2022-11-14.

[26] "Amazon's EC2 [online]," Available at `https://aws.amazon.com/ec2/instance-types/?nc1=h_ls`, 2022, Accessed: 2022-11-14.

[27] "Huawei Elastic Cloud Server (ECS) [online]," Available at `https://www.huaweicloud.com/intl/en-us/product/ecs.html`, 2022, Accessed: 2022-11-14.

[28] Sage Weil, Scott Brandt, Ethan Miller, Darrell Long, and Carlos Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System," in *Proceedings of OSDI*, 2006, pp. 307–320.

[29] Omar Sefraoui, Mohammed Aissaoui, and M. Eleuldj, "OpenStack: Toward an Open-source Solution for Cloud Computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, Mar. 2012.

[30] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori, "kvm: the Linux virtual machine monitor," in *Proceedings of the Linux symposium*, 2007, pp. 225–230.

[31] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, "Xen and the art of virtualization," *ACM SIGOPS operating systems review*, vol. 37, no. 5, pp. 164–177, 2003.

[32] "Amazon web services [online]," Available at `https://aws.amazon.com/ec2/faqs/?nc1=h_ls`, 2022, Accessed: 2022-11-28.

[33] "Google compute engine faq [online]," Available at `https://cloud.google.com/compute/docs/faq`, 2022, Accessed: 2022-11-28.

[34] "Libvirt: The virtualization api [online]," Available at `https://libvirt.org`, 2022, Accessed: 2022-11-28.

[35] "Qemu [online]," Available at `https://www.qemu.org`, 2022, Accessed: 2022-11-28.

[36] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado, "The Design and Implementation of Open vSwitch," in *Proceedings of NSDI*, 2015, pp. 117–130.

[37] Rusty Russell, "virtio: towards a de-facto standard for virtual I/O devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.

[38] "Dpdk [online]," Available at `https://www.dpdk.org/`, 2022, Accessed: 2022-11-28.

[39] Michael Bayer et al., "Mako Templates," Available at `http://www.makotemplates.org/`, 2019.

[40] Ru Jia, Yun Yang, John Grundy, Jacky Keung, and Li Hao, "A systematic review of scheduling approaches on multi-tenancy cloud platforms," *Information and Software Technology*, vol. 132, pp. 106478, 2021.

[41] Zhe Wang, Chen Xu, Kunal Agrawal, and Jing Li, "Adaptive scheduling of multiprogrammed dynamic-multithreading applications," *Journal of Parallel and Distributed Computing*, vol. 162, pp. 76–88, 2022.

[42] Chengzhi Lu, Kejiang Ye, Guoyao Xu, Cheng-Zhong Xu, and Tongxin Bai, "Imbalance in the cloud: An analysis on Alibaba cluster trace," in *Proceedings of Big Data*, 2017, pp. 2884–2892.

[43] Qixiao Liu and Zhibin Yu, "The Elasticity and Plasticity in Semi-Containerized Co-Locating Cloud Workload: A View from Alibaba Trace," in *Proceedings of SoCC*, 2018, pp. 347–360.

[44] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini, "Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms," in *Proceedings of SOSP*, 2017, pp. 153–167.

[45] Intel, "Improving real-time performance by utilizing cache allocation technology," *Intel Corporation, April*, 2015.

[46] Andrew H., Abbasi, Khawar M., Marcel C., "Introduction to Memory Bandwidth Allocation," Available at `https://software.intel.com/en-us/articles/introduction-to-memory-bandwidth-allocation`, 3 2019.

Lenguajes, compiladores y herramientas de programación y ejecución paralela

Maleabilidad MPI basada en la eficiencia paralela

Sergio Iserte[1], Victor Lopez[1], Marta Garcia-Gasulla[1] y Antonio J. Peña[1]

Resumen— **Este artículo presenta la integración del recolector de métricas de rendimiento TALP en la biblioteca para maleabilidad DMR.**

Tradicionalmente, las aplicaciones científicas que hacen uso de la computación de altas prestaciones no pueden reasignar recursos una vez asignados, lo que dificulta la adaptabilidad de las cargas de trabajo. La gestión dinámica de recursos, especialmente a través de la maleabilidad de procesos MPI, se plantea como una solución para esta limitación. Gracias a la integración de TALP, DMR ofrecerá nuevas políticas de reconfiguración basadas en métricas de rendimiento como la eficiencia paralela, el balance de carga o la eficiencia de comunicaciones.

Palabras clave—**Recursos dinámicos, maleabilidad de procesos, métricas de rendimiento**

I. Introducción

En los centros de computación de altas prestaciones (HPC, del inglés *High Perfomance Computing*), las aplicaciones se ejecutan en clústeres donde cientos o miles de peticiones compiten por los mismos recursos. Los recursos se solicitan mediate trabajos que además de los requisitos necesarios también contienen los comandos para ejecutar las aplicaciones. El conjunto de todos los trabajos en el sistema conforma la carga de trabajos.

Así pues, adaptar la carga de trabajo a los recursos disponibles dependiendo de las necesidades de las aplicaciones puede aumentar considerablemente la utilización de recursos y la productividad global del sistema en términos de número de trabajos completados por unidad de tiempo. Esta adaptación se puede hacer de forma dinámica, es decir, durante la ejecución de los trabajos, gracias a la maleabilidad.

La mayor parte de las aplicaciones científicas encontradas en centros HPC se ejecutan en paralelo en distintos nodos de cómputo siguiendo el modelo de computación de memoria distribuida basado en el sistema de paso de mensajes MPI (del inglés *Message Passing Interface*).

Tradicionalmente, estas aplicaciones HPC solicitan una serie de recursos computacionales para su ejecución que una vez se les han sido asignados no son reasignados hasta la finalización de los respectivos trabajos. Esta restricción previene la implantación de cargas de trabajos adaptativas que negocien dinámicamente la asignación de recursos. La gestión dinámica de recursos se plantea como una solución a esta rigidez a la hora de procesar cargas de trabajos. Una técnica para aportar dinamismo a la gestión de recursos es la maleabilidad de procesos MPI. Esta maleabilidad permite a las aplicaciones cambiar, en tiempo de ejecución la configuración de procesos, en repetidas veces. Este mecanismo es esencial para disponer de cargas de trabajos que se adapten a las circunstancias en constante cambio que se encuentran en los clústeres HPC.

La gestión dinámica de recursos es un tema actual de gran relevancia en el que hay un creciente interés gracias a la iniciativa europea *EuroHPC Joint Undertaking*[1] y la financiación de grandes proyectos europeos como: DEEP-SEA[2], ADMIRE[3], EUPilot[4], TIME-X[5] y REGALE[6].

En este trabajo se presenta una herramienta de maleabilidad MPI capaz de tomar decisiones de reconfiguración de procesos basadas en métrica de eficiencia paralela tomadas en tiempo de ejecución. En concreto, este artículo describe como la biblioteca DMR (del inglés, *Dynamic Management of Resources*) ha integrado la herramienta TALP (del inglés, *Tracking Application Live Performance*) que permite la medición de la eficiencia paralela para aplicaciones HPC. Por este motivo, DMR adquiere conocimiento sobre el comportamiento, en términos de rendimiento, eficiencia y balanceo de carga, de los trabajos para así poder determinar acciones de expansión o contracción de trabajos de forma más inteligente.

El resto del artículo está estructurado del siguiente modo: En la Sección II, se presentan los fundamentos de la maleabilidad los esfuerzos previos en este tema y se describe en profundidad DMR y TALP. En la Sección III, se explica cómo se ha realizado la integración de las dos herramientas y su desarrollo. La Sección IV contiene un ejemplo de uso y como se ha utilizado DMR+TALP en el código de la implementación distribuida del método Jacobi. Finalmente, la Sección V culmina el artículo con una serie conclusiones extraídas tras los primeros experimentos y establece cuáles podrían ser las futuras líneas de trabajo.

II. Fundamentos

Actualmente, se pueden encontrar diferentes estrategias de maleabilidad de procesos MPI. Esta sección revisa estos enfoques junto con los marcos en los que se implementan.

Por un lado, las estrategias de C/R (en inglés, *Checkpoint/Restart*) para la maleabilidad que para

[1]Barcelona Supercomputing Center (BSC), e-mail: {sergio.iserte, victor.lopez, marta.garcia, antonio.pena}@bsc.es.

[1]https://eurohpc-ju.europa.eu
[2]https://www.deep-projects.eu
[3]https://www.admire-eurohpc.eu
[4]https://eupilot.eu
[5]https://www.time-x-eurohpc.eu
[6]https://regale-project.eu

la redistribución de datos durante la reconfiguración almacena en disco el estado de la aplicación. Algunos ejemplos son Process Checkpointing and Migration (PCM) API [1], Scalable C/R (SCR) [2], Stop Restart Software (SRS) [3] o ReSHAPE [4].

Por otro lado, en la bibliografía también se encuentran soluciones de maleabilidad que redistribuyen los datos sin necesidad de almacenarlos en disco. En su lugar, los datos se transfieren entre las memorias principales de los distintos nodos de cómputo mediante mensajes. Por ejemplo, los autores [5] presentan una técnica de reconfiguración para aplicaciones MPI basada en la biblioteca ULFM (del inglés, *User-Level Fault Mitigation*) [6], que admite el uso de la rutina estándar *MPI_ Comm_ spawn* y la eliminación dinámica de procesos. Este tipo de maleabilidad se ha seguido estudiando en [7] donde se desarrolla una infraestructura para la ejecución elástica de aplicaciones MPI basándose en Slurm[7] y MPICH[8].

A pesar de la variedad de soluciones en maleabilidad, la toma de decisiones no ha sido un tema demasiado estudiado. Por lo que las políticas de reconfiguración han sido más bien simples, basándose en criterios como el estado del clúster o información de rendimiento dada por el usuario. El uso de información obtenida de la misma ejecución se ha estudiado anteriormente en la herramienta Flex-MPI [8] que ofrece una política de reconfiguración consciente del rendimiento que automáticamente reconfigura el trabajo MPI para ajustarlo a los parámetros de eficiencia definidos. Flex-MPI [9] no permite que los procesos iniciales finalicen durante la ejecución. Por lo tanto, solo se pueden reducir los procesos dinámicos después de una expansión. Para superar estas restricciones y seguir disponiendo de la medición de rendimiento en tiempo de ejecución se ha integrado TALP en DMR.

Un estado del arte de maleabilidad más extenso se puede encontrar en [10].

A continuación, se detallan las bibliotecas integradas para este proyecto: DMR proporciona la maleabilidad de procesos, mientras que TALP la medición de la eficiencia.

A. DMR

Concretamente, en este trabajo se ha utilizado DMR para la gestión dinámica de procesos [11]. DMR ha demostrado ser una herramienta de maleabilidad competitiva gracias a sus diversos casos de éxito publicados [12], [13] y los que están en marcha. DMR consta de dos componentes principales: un gestor de recursos (RMS, del inglés *Resource Manager System* y un *runtime* distribuido paralelo basado en MPI [14]. DMR enlaza ambos componentes mediante una capa de comunicación que permite la ejecución de aplicaciones maleables en una carga de trabajos para clústeres HPC. Slurm el RMS utilizado por DMR y, como en otras soluciones, es el responsable de monitorizar la utilización de recursos y las solicitudes de los trabajos. DMR ha extendido Slurm para dotarlo de la capacidad de gestionar trabajos maleables y administrar recursos dinámicos [15]. En resumen, la maleabilidad de trabajos en DMR funciona de la siguiente manera:

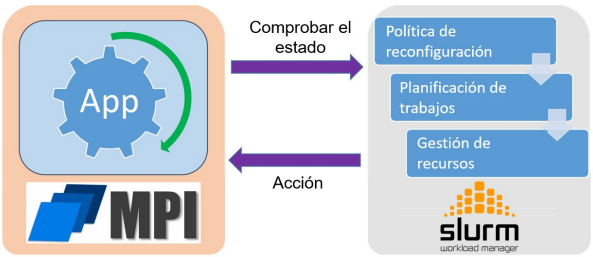


Fig. 1: Esquema de la comunicación de una aplicación maleable con DMR.

- Cuando una aplicación se está ejecutando, tiene que ponerse en contacto periódicamente con el RMS para mostrar su voluntad de ser reconfigurada.
- Esta comunicación se establece en un punto de sincronización, definido por los usuarios, donde se puede iniciar la operación de reconfiguración.
- A continuación, Slurm evalúa el estado global del sistema para decidir si realizar esa reconfiguración y comunicar esa decisión al *runtime* de MPI.
- Si esta llamada resuelve cambiar el tamaño del trabajo, Slurm reasigna los recursos y devuelve el número resultante de procesos, lo que puede significar expandir o reducir el trabajo.
- Finalmente, el *runtime*, con las directrices de la aplicación, redistribuye los datos entre los procesos y el trabajo continúa la ejecución con la nueva configuración de procesos.

La Figura 1 esquematiza la comunicación que establece DMR entre la aplicación y el RMS.

Además, la implementación actual de DMR acomoda tres modos de programación con diferentes grados de libertad al solicitar una acción de cambio de reconfiguración:

- Las aplicaciones pueden **sugerir encarecidamente** una operación específica. Por ejemplo, para expandir un trabajo, los usuarios pueden definir el número "mínimo"de nodos solicitados a un valor mayor que el número de nodos asignados de ese trabajo.
- Determinando un **número óptimo** de procesos para ejecutar una aplicación. Los usuarios pueden configurar una aplicación maleable con un número de procesos deseado del que no se puede bajar. Sin embargo, se puede decidir conceder una expansión hasta el máximo especificado.
- **Máxima expansión**:
  - Se expandirá un trabajo siempre que haya suficientes recursos disponibles si:
    - No hay ningún trabajo pendiente de ejecución en la cola.
    - No se puede ejecutar ningún trabajo pen-

[7] https://slurm.schedmd.com
[8] https://www.mpich.org

diente debido a la insuficiencia de recursos disponibles.
- Se reducirá un trabajo si hay algún trabajo en cola que se pueda iniciar con parte de los recursos asignados al trabajo a contraer.

A modo de resumen de la política de reconfiguración original en DMR, la Figura 2 esquematiza la toma de decisiones. Como se puede apreciar, la política de reconfiguración no es consciente del rendimiento de la aplicación en ejecución en ningún momento.

B. TALP

TALP es uno de los módulos de la biblioteca DLB (del inglés, *Dynamic Load Balancing*) [16], [17]. El modulo TALP [18] recoge metricas de rendimiento relacionadas con la paralelizacion MPI durante la ejecucion de la aplicacion de manera transparente para el usuario.

Las metricas recogidas por TALP pueden ser visualizadas al finalizar la ejecución, pero tambien ofrece una API que permite consultarlas en tiempo de ejecucion. Esta API será la que utiliza DMR para obtener las metricas de eficiencia paralela de la aplicacion.

Las metricas recogidas por TALP son las llamadas "metricas POP" [19], este conjunto de metricas estan organizadas de manera jerarquica y son multiplicativas. Es decir, el valor de la metrica padre es igual al producto de las metricas hijas.

En la Figura 3 podemos encontrar representada la jerarquia de metricas POP. Las metricas que TALP puede recoger de esta jerarquia son: eficiencia paralela *Parallel Efficiency*), balance de carga(*Load Balance*) y eficiencia de comunicación (*Communication Efficiency*).

La eficiencia paralela indica que porcentaje del tiempo consumido se ha usado para hacer calculo útil (por ejemplo, tiempo fuera de comunicaciones MPI); el balance de carga indica el tiempo que se ha perdido por esperar a un proceso que tiene más carga de trabajo que los demás; y la efficiencia de comunicación corresponde al tiempo dedicado a esperar una comunicación de otros procesos o por los mismos sobrecostes de usar comunicación MPI.

Una solución específica para la aplicación de dinámica de fluidos computacional (CFD, del inglés *Computational Fluid Dynamics*) Alya [20], que emplea COMP Superscalar (COMPSs) [21] junto con la biblioteca TALP [18], para permitir la maleabilidad consciente del rendimiento. Mediante COMPS se explota el paralelismo directamente desde el código secuencial, lo que permite a los usuarios evitar cualquier problema relacionado con la concurrencia. Alya utiliza técnicas de C/R que aprovecha COMPS para reconfigurar la aplicación y redistribuir los datos. Las reconfiguraciones solo se producen si TALP indica que proporcionará una reducción en el tiempo de ejecución para la expansión o una eficiencia similar con menos recursos.

III. Metodología

La biblioteca DMR se ha extendido para integrar las funcionalidades de medición ofrecidas por TALP. Gracias a la API de TALP la integración ha sido directa, añadiendo las llamadas a las funciones correspondientes para obtener la información de la aplicación en ejecución.

El Código 1 describe de forma genérica el esqueleto de una aplicación iterativa maleable implementada con DMR. Existen tres zonas de interés para el/la desarrollador/a: inicialización, reconfiguración y finalización. Se inicializa el entorno de maleabilidad junto con las estructuras de datos necesarias por la propia aplicación mediante `DMR_INIT`. A `DMR_INIT` se le debe de proporcionar la función para la inicialización de los datos del programa, junto con las funciones para recibir los datos tanto en el caso de expansión como la contracción del trabajo.

El Algoritmo 1 describe la etapa de inicialización dentro de la biblioteca DMR. Inicialmente, se comprueba que los procesos se han lanzado con el comando `mpirun` o bien desde una invocación a la función `MPI_Comm_spawn` (línea 2). En el primer caso, los procesos no tienen un comunicador padre (línea 10). En caso de que los procesos tengan un comunicador padre, deberán recibir los datos de los procesos en el comunicador padre y desconectarse de él (líneas 4-8), para seguir con la ejecución del programa. Es en este último supuesto donde se desactiva la monitorización llevada a cabo por TALP para no tener en cuenta a la hora de calcular las métricas de eficiencia la recepción de datos por parte de los nuevos procesos (línea 3). Una vez terminada la transferencia de datos, se reactivan las mediciones (línea 9).

Tras esta etapa, comienza la ejecución de las iteraciones. Para cada iteración, se realizará el cómputo propio de la aplicación y se llevará a cabo la consulta para ver si hay que reconfigurar el trabajo en `DMR_RECONFIGURATION`. Nótese que la reconfiguración se invoca después de la realización del cómputo en esa iteración, aunque también podría llevarse a cabo siempre y cuando los procesos estén sincronizados. Para la reconfiguración es necesario conocer cómo se envían los datos en caso de expansión y contracción de trabajo, por este motivo, se proporcionan estas funciones. Como en la etapa de inicialización, en el Algoritmo 2 se puede observar cómo se desactiva la monitorización de métricas por parte de TALP durante las operaciones de maleabilidad (línea 1). Estas operaciones corresponden a la petición de reconfiguración (línea 2) y, en caso de tener que reconfigurar el trabajo, enviar los datos a los nuevos procesos (líneas 3-9). Durante la petición de reconfiguración (línea 2) se capturan las métricas de eficiencia calculadas para ese instante y son transferidas a Slurm para que las pueda utilizar en la política de reconfiguración habilitada. Tan sólo en el caso de que no haya que reconfigurar la aplicación, se reactivará la monitorización de las métricas TALP (línea 11). Para no añadir sobrecostes adicionales, no se reactivarán en caso contrario ya que los procesos terminarán su eje-

Fig. 2: Esquema de la política de reconfiguración de DMR.

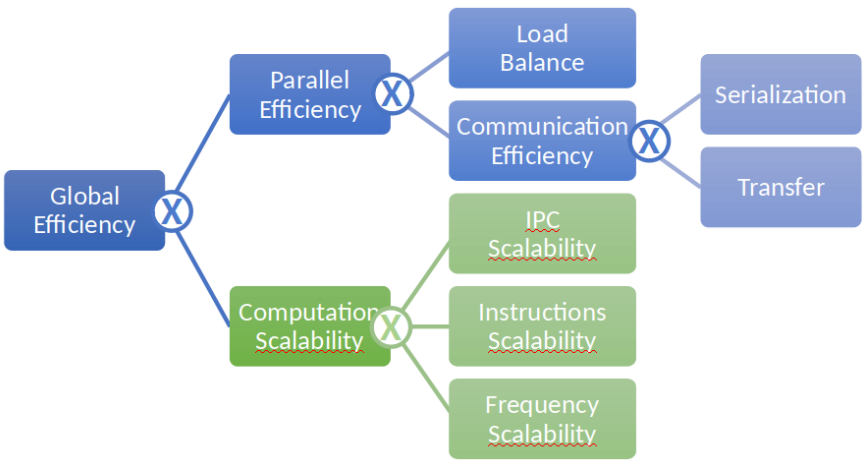


Fig. 3: Jerarquia de metricas POP.

```
1  void main() {
2    DMR_INIT(user_init(), r_ex(), r_sh());
3    for (int i; i<ITERATIONS; i++) {
4      compute();
5      DMR_RECONFIGURATION(s_ex(), s_sh());
6    }
7    DMR_FINALIZE(user_free());
8  }
```

Listing 1: Código maleable en DMR.

Algoritmo 1 Inicialización DMR

```
1:  parentComm ← MPI_Comm_get_parent()
2:  if parentComm ≠ MPI_COMM_NULL then
3:      DLB_MonitoringRegionStop()
4:      if commWorldSize > parentCommSize then
5:          r_ex()
6:      else
7:          r_sh()
8:      MPI_Comm_disconnect(parentComm)
9:      DLB_MonitoringRegionStart()
10: else
11:     user_init()
```

Algoritmo 2 Reconfiguración DMR

```
1:  DLB_MonitoringRegionStop()
2:  action ← DMR_Reconfiguration()
3:  if action = expand then
4:      s_ex()
5:      DMR_Detach()
6:  else
7:      if action = shrink then
8:          s_sh()
9:          DMR_Detach()
10:     else
11:         DLB_MonitoringRegionStart()
```

Algoritmo 3 Finalización DMR

```
1:  user_free()
2:  DMR_Finalize()
```

cución cuando completen el envío de datos.

Finalmente, la aplicación terminará liberando la memoria reservada, entre otras rutinas específicas a realizar antes de acabar la ejecución, indicando cómo hacerlo a `DMR_FINALIZE` (véase el Algoritmo 3).

Para este trabajo se ha sustituido la política de reconfiguración descrita en la Figura 2, por una que aproveche la información capturada por TALP. Particularmente, se ha diseñado una política que pretende mantener una eficiencia paralela sostenida en el tiempo dentro de un rango, tal y como muestra la Figura 4. Así pues, si tras completar una iteración, TALP reporta una eficiencia paralela menor que el límite máximo del rango de eficiencia definido, se programará una contracción a la anterior configuración de procesos posible. Por el contrario, si la eficiencia registrada es mayor que la definida en el límite inferior del rango, se programará una expansión a la siguiente configuración de procesos posible. Por ejemplo, si la aplicación soporta maleabilidad lineal con factor de reconfiguración 2, significa que el trabajo podrá expandirse y contraerse conforme a los valores de las potencias de 2, es decir, en cada paso se podrá dividir a la mitad o duplicar el número de

```
1  int main(int argc, char *argv[]) {
2      int world_size, world_rank, size =
         SIZE;
3      MPI_Init(&argc, &argv);
4      MPI_Comm_rank(MPI_COMM_WORLD,
         &world_rank);
5      MPI_Comm_size(MPI_COMM_WORLD,
         &world_size);
6      double *a, *b, *x;
7      DMR_INIT(initialize_data(&a, &b, &x,
         size), recvEx(&a, &x, &b, &size),
         recvSh(&a, &x, &b, &size));
8      DMR_Set_parameters(1, 8, 0);
9      DMR_Inhibit_iter(3);
10     for (; DMR_it < ITERATIONS; DMR_it++) {
11         run_jacobi(a, x, b, size);
12         DMR_RECONFIGURATION(sendEx(a, x,
             b, n), sendSh(a, x, b, n));
13     }
14     DMR_FINALIZE(free_data(&a, &b, &x));
15     MPI_Finalize();
16     return 0;
17 }
```

Listing 2: Código de Jacobi maleable en DMR.

procesos.

IV. Resultados

Para comprobar el correcto funcionamiento de la integración de TALP en DMR se ha utilizado la implementación MPI del código de Jacobi en [22]. El método de Jacobi es un método iterativo usado para resolver sistemas de ecuaciones lineales. La función principal de la versión maleable se puede ver en el Código 2. Este código incluye las llamadas MPI necesarias para iniciar (línea 3) y finalizar (línea 15) el entorno, al igual que las llamadas a DMR para habilitar y configurar la maleabilidad. En concreto, se han configurado los límites de la maleabilidad en un mínimo de un proceso y un máximo de ocho (línea 8) con `DMR_Set_parameters`. Con esa misma función no se ha definido ninguna preferencia de configuración de procesos (valor cero en el tercer argumento), así el RMS cuenta con mayor autonomía para planificar acciones de reconfiguración. Para esta aplicación, también se han definido periodos de tres iteraciones en los que se inhibe la maleabilidad (línea 9) para evitar constantes reconfiguraciones de procesos.

Así pues, los vectores de datos declarados en la línea 6 se inicializan en la línea 7 mediante DMR y son utilizados a lo largo de la ejecución. Además, estos vectores se redimensionan dependiendo de la configuración de procesos ya que el tamaño global del problema es constante. Este redimensionamiento de las estructuras de datos se lleva a cabo en las funciones de recepción (línea 7), mientras que en las funciones de envío se calcula como redistribuirlas (línea 12).

La aplicación ha sido evaluada en nueve nodos del supercomputador Marenostrum IV del Barcelona Supercomputing Center (BSC). Cada nodo de este supercomputador está equipado con dos procesadores Intel Xeon Platinum 8160 (24 núcleos a 2.10 GHz cada uno) con un total de 48 núcleos y 96 GB de RAM. Los nodos están interconectados a través de una red de interconexión Intel Omni-Path a 100 Gbit/s. En cuanto al software, la biblioteca DMR se ha ejecutado con MPICH 3.2 y Slurm 17.02. Uno de los nueve nodos hospeda el demonio controlador de Slurm, mientras que los ocho restantes actúan como nodos de cómputo.

Para el caso de estudio en cuestión, el paralelismo intra-nodo se implementa con OpenMP, por lo que se ejecuta un proceso de 48 hilos por nodo hasta un máximo de ocho nodos.

La aplicación se lanza inicialmente con un proceso y hasta la tercera iteración no se consulta el estado para la maleabilidad (la inhibición se ha habilitado). Tras la tercera iteración TALP reporta una eficiencia paralela ≈ 1, por lo que la política de reconfiguración de Slurm planifica una expansión a dos procesos. DMR lleva a cabo la expansión y se continúa con el procesamiento de las iteraciones. Tras tres iteraciones más configuradas en dos procesos, se vuelve a planificar una expansión a cuatro procesos, ya que la eficiencia reportada es mayor que el umbral definido. Tras las iteraciones iniciales y varias expansiones y contracciones, la eficiencia se estabiliza en $\approx 0{,}99$ utilizando ocho procesos hasta la última iteración cuando termina el programa. Es al inicio de la ejecución de Jacobi cuando se observan mayores variaciones en la eficiencia paralela. Parece ser que hasta que la solución no empieza a converger, la eficiencia no se estabiliza. Durante esa transición, DMR ajusta el número de procesos para que los requisitos en eficiencia se cumplan. A continuación, se muestran ejemplos de las transiciones en la configuración de procesos para varias ejecuciones llevadas a cabo:

$$1 \to 2 \to 4 \to 2 \to 1 \to 2 \to 4 \to 8 \ldots\ldots \quad (1)$$

$$1 \to 2 \to 4 \to 2 \to 1 \to 2 \to 4 \to 8 \ldots\ldots \quad (2)$$

$$1 \to 2 \to 4 \to 8 \to 4 \to 2 \to 1 \to 2 \to 4 \to 8 \ldots\ldots \quad (3)$$

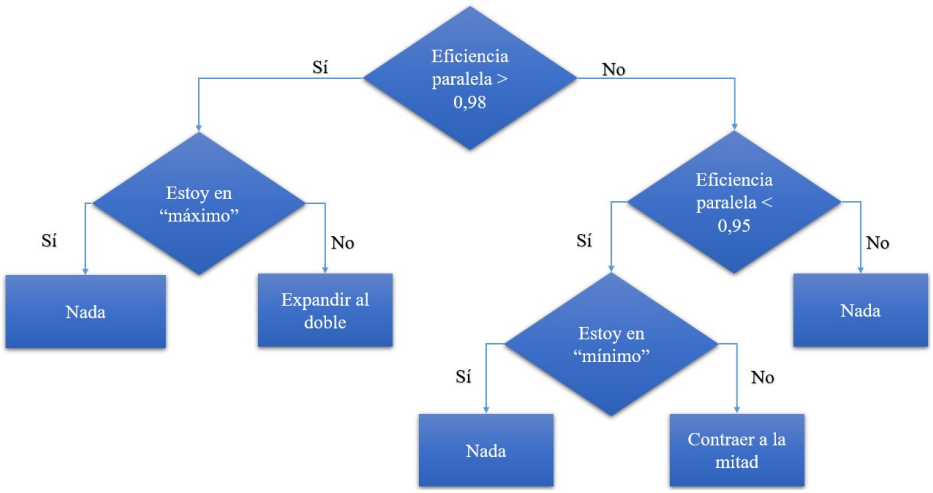


Fig. 4: Esquema de la política de reconfiguración de DMR consciente de la eficiencia paralela.

V. Conclusiones

En este artículo se ha presentado la integración de TALP, una herramienta de medición de eficiencia paralela, en la biblioteca para maleabilidad de procesos DMR. Esta integración ha permitido el diseño de una política de reconfiguración de trabajos consciente de la eficiencia en tiempo real del trabajo en ejecución, para así, poder tomar decisiones de escalabilidad en el número de procesos y recursos asignados.

La evaluación del sistema presentado se ha llevado a cabo aprovechando el método iterativo Jacobi para la resolución de ecuaciones lineales. Los resultados muestran que DMR va ajustando el tamaño del trabajo (en términos de número de procesos) dependiendo de la eficiencia obtenida en cada iteración. El algoritmo, a medida que converge, muestra mayor eficiencia por lo que es posible asignarle más recursos para acelerar su ejecución.

Esto ha sido tan solo un estudio preliminar que será extendido con políticas de reconfiguración más inteligentes que tengan en cuenta más métricas y el estado de otros trabajos y de los propios recursos.

Agradecimientos

El proyecto *The European PILOT* ha recibido financiación de *European High-Performance Computing Joint Undertaking (JU)* en virtud del acuerdo de subvención n.º 101034126. La JU recibe apoyo del programa de investigación e innovación Horizonte 2020 de la Unión Europea y de España, Italia, Suiza, Alemania, Francia, Grecia, Suecia, Croacia y Turquía. Además, *The European PILOT*, PCI2021-122090-2A también cuenta con financiamiento conjunto del MCIN/AEI/10.13039/501100011033 y del UE NextGenerationEU/PRTR.

La investigación que ha producido este artículo recibió financiamiento del proyecto DEEP-SEA del programa EuroHPC de la Comisión Europea bajo el acuerdo de subvención 955606 y la subvención PCI2021-121958 financiada por MCIN/AEI/10.13039/501100011033 y por "European Union NextGenerationEU/PRTR".

Referencias

[1] Kaoutar El Maghraoui, Boleslaw K Szymanski, and Carlos Varela, "An Architecture for Reconfigurable Iterative MPI Applications in Dynamic Environments," in *International Conference on Parallel Processing and Applied Mathematics*, 2006, pp. 258–27.

[2] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski, "Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC10)*, nov 2010.

[3] S. S. Vadhiyar and J. J. Dongarra, "SRS - A Framework for Developing Malleable and Migratable Applications for Distributed Systems," *Parallel Processing Letters*, vol. 2, pp. 291–312, 2002.

[4] Rajesh Sudarsan and Calvin J. Ribbens, "ReSHAPE: a Framework for Dynamic Resizing and Scheduling of Homogeneous Applications in a Parallel Environment," in *Proceedings of the International Conference on Parallel Processing*, 2007.

[5] Pierre Lemarinier, Khalid Hasanov, Srikumar Venugopal, and Kostas Katrinis, "Architecting Malleable MPI Applications for Priority-driven Adaptive Scheduling," in *Proceedings of the 23rd European MPI Users' Group Meeting on - EuroMPI 2016*, New York, New York, USA, sep 2016, pp. 74–81, ACM Press.

[6] W. Bland, A. Bouteiller, T. Herault, G. Bosilca, and J. Dongarra, "Post-failure Recovery of MPI Communication Capability: Design and Rationale," *International Journal of High Performance Computing Applications*, vol. 27, no. 3, pp. 244–254, jun 2013.

[7] Isaías Comprés, Ao Mo-Hellenbrand, Michael Gerndt, and Hans-Joachim Bungartz, "Infrastructure and API Extensions for Elastic Execution of MPI Applications," in *Proceedings of the 23rd European MPI Users' Group Meeting on - EuroMPI 2016*. 2016, pp. 82–97, ACM Press.

[8] Gonzalo Martín, Maria-Cristina Marinescu, David E. Singh, and Jesús Carretero, "FLEX-MPI: an MPI Extension for Supporting Dynamic Load Balancing on Heterogeneous Non-dedicated Systems," in *Euro-Par Parallel Processing*, aug 2013, pp. 138–149.

[9] Gonzalo Martín, David E. Singh, Maria-Cristina Marinescu, and Jesús Carretero, "Enhancing the Performance of Malleable MPI Applications by Using Performance-aware Dynamic Reconfiguration," *Parallel Computing*, vol. 46, pp. 60–77, jul 2015.

[10] Jose I. Aliaga, Maribel Castillo, Sergio Iserte, Iker Martín-Álvarez, and Rafael Mayo, "A Survey on Malleability Solutions for High-Performance Distributed Computing," *Applied Sciences*, vol. 12, no. 10, 2022.

[11] Sergio Iserte, Rafael Mayo, Enrique S. Quintana-Ortí, Vicenç Beltran, and Antonio J. Peña, "DMR API: Improving Cluster Productivity by Turning Applications into Malleable," *Parallel Computing*, vol. 78, pp. 54–66, oct 2018.

[12] Sergio Iserte and Krzysztof Rojek, "A Study of the Effect

of Process Malleability in the Energy Efficiency on GPU-based Clusters," *The Journal of Supercomputing*, pp. 1–20, oct 2019.

[13] Sergio Iserte, Héctor Martínez, Sergio Barrachina, Maribel Castillo, Rafael Mayo, and Antonio J Peña, "Dynamic Reconfiguration of Noniterative Scientific Applications," *The International Journal of High Performance Computing Applications*, p. 109434201880234, sep 2018.

[14] S. Iserte, R. Mayo, E.S. Quintana-Orti, and A.J. Pena, "DMRlib: Easy-coding and Efficient Resource Management for Job Malleability," *IEEE Transactions on Computers*, 2020.

[15] Sergio Iserte, *High-throughput Computation through Efficient Resource Management*, Ph.D. thesis, Universitat Jaume I, Castelló de la Plana, nov 2018.

[16] M. Garcia, J. Corbalan, and J. Labarta, "LeWI: A Runtime Balancing Algorithm for Nested Parallelism," in *Parallel Processing, 2009. ICPP '09. International Conference on*, Sept 2009, pp. 526–533.

[17] Marta Garcia, Jesus Labarta, and Julita Corbalan, "Hints to Improve Automatic Load Balancing with LeWI for Hybrid Applications," *J. Parallel Distrib. Comput.*, vol. 74, no. 9, pp. 2781–2794, sep 2014.

[18] Victor Lopez, Guillem Ramirez Miranda, and Marta Garcia-Gasulla, "TALP: A Lightweight Tool to Unveil Parallel Efficiency of Large-Scale Executions," in *Proceedings of the 2021 on Performance EngineeRing, Modelling, Analysis, and VisualizatiOn STrategy*, New York, NY, USA, 2021, PERMAVOST '21, p. 3–10, Association for Computing Machinery.

[19] "A Generic Performance Analysis Technique Applied to Different CFD Methods for HPC, author=Garcia-Gasulla, Marta and Banchelli, Fabio and Peiro, Kilian and Ramirez-Gargallo, Guillem and Houzeaux, Guillaume and Ben Hassan Saïdi, Ismaïl and Tenaud, Christian and Spisso, Ivan and Mantovani, Filippo, journal=International Journal of Computational Fluid Dynamics, volume=34, number=7-8, pages=508–528, year=2020, publisher=Taylor & Francis," .

[20] G Houzeaux, R M Badia, R Borrell, D Dosimont, J Ejarque, M Garcia-Gasulla, and V López, "Dynamic Resource Allocation for Efficient Parallel CFD Simulations," Tech. Rep., Barcelona Supercomputing Center, 12 2021.

[21] Rosa M. Badia, Javier Conejero, Carlos Diaz, Jorge Ejarque, Daniele Lezzi, Francesc Lordan, Cristian Ramon-Cortes, and Raul Sirvent, "COMP Superscalar, an interoperable programming framework," *SoftwareX*, vol. 3-4, pp. 32–36, 12 2015.

[22] Edmond Jajaga and Jolanda Kllobocishta, "MPI Parallel Implementation of Jacobi," in *ICT Innovations*, 2012, pp. 449–458.

Generación de Micro-kernels para Multiplicación de Matrices con Exo

Adrián Castelló[1], Julian Bellavita[2], Grace Dinh[3], Yuka Ikarashi[4] y Héctor Martínez[5]

Resumen— **La optimización de la multiplicación de matrices (o GEMM) ha sido una necesidad durante las últimas décadas. Actualmente, esta operación se considera el buque insignia de bibliotecas de álgebra lineal tales como BLIS, OpenBLAS o Intel OneAPI debido a su uso generalizado en una gran variedad de aplicaciones científicas. La GEMM suele implementarse siguiendo la filosofía GotoBLAS, que aplica una división por bloques a los operandos de la GEMM y utiliza una serie de bucles anidados para mejorar el rendimiento. Estos enfoques extraen la máxima potencia computacional de las arquitecturas a través de pequeñas piezas de código de alto rendimiento orientadas al hardware, denominadas micro-kernels. Sin embargo, esta estructura obliga a los desarrolladores a generar, con un esfuerzo no desdeñable, un micro-kernel dedicado para cada nuevo hardware.**

En este trabajo, presentamos, paso a paso, el procedimiento para generar micro-kernels con el compilador Exo para arquitecturas ARM, cuyo rendimiento se aproxima (o incluso supera) al de los micro-kernels desarrollados manualmente con funciones intrínsecas o lenguaje ensamblador. Nuestra solución también permite mejorar la portabilidad del código, ya que únicamente hay que modificar las instrucciones dependientes del hardware.

Palabras clave— **Generación de código, Computación de altas prestaciones, Exo, Algebra lineal, micro-kernels**

I. Introducción

En las últimas décadas, se ha hecho un esfuerzo incesante por desarrollar implementaciones de alto rendimiento de bibliotecas de álgebra lineal (LA del inglés linear algebra). Estas bibliotecas se han diseñado para una amplia variedad de arquitecturas, como procesadores vectoriales, procesadores multinúcleo, unidades de procesamiento gráfico (GPU) y, más recientemente, arquitecturas y aceleradores de dominios específicos. Este esfuerzo nada desdeñable suele provenir de los principales proveedores de hardware, produciendo algunos productos relevantes como Intel OneAPI, AMD AOCL, IBM ESSL, ARMPL y NVIDIA cuBLAS, así como del mundo académico, con paquetes de software como GotoBLAS2 [1], OpenBLAS [2], BLIS [3] y ATLAS [4].

La multiplicación de matrices (GEMM) es el núcleo computacional insignia sobre el que se basan las bibliotecas de LA. Más concretamente, el rendimiento general de estas bibliotecas se debe a una pequeña pieza de código, llamado micro-kernel, que es el enlace entre el algoritmo GEMM y el hardware subyacente. Además, la GEMM es también una operación clave para las aplicaciones de aprendizaje profundo (DL, del inglés Deep Learning) que utilizan redes neuronales profundas (DNNs) como las redes convolucionales para el procesamiento de señales y la visión por ordenador [5], [6]. Desafortunadamente, estas bibliotecas de LA presentan algunas limitaciones:

1. Las rutinas optimizadas son específicas del hardware. Este es el caso de los productos Intel, IBM o ARM. En menor medida, también se aplica a GotoBLAS2, OpenBLAS y BLIS, que utilizan una colección de micro-kernels orientados al hardware [7].
2. El desarrollo de micro-kernels altamente optimizados para GEMM requiere profundos conocimientos de computación de alto rendimiento y arquitectura de ordenadores.
3. Una nueva arquitectura implica un nuevo conjunto de pruebas y depuración para el desarrollo de micro-kernels para extraer la máxima potencia computacional del nuevo hardware.
4. El código de estos micro-kernels suele utilizar macros, plantillas para aumentar la productividad así como técnicas de programación de alto nivel. Por lo tanto, el mantenimiento de las bibliotecas queda en manos de los desarrolladores originales.
5. El software existente en las bibliotecas omite algunos casos relevantes como, por ejemplo, soporte para aritmética de coma flotante de media precisión (16 bits) o aritmética de enteros.
6. La implementación de la GEMM en estas librerías es subóptima porque el micro-kernel único suele estar optimizado para casos de grandes matrices "cuadradas". Por lo tanto, las GEMM rectangulares tal y como aparecen en DL suelen presentar un rendimiento inferior ya que los casos frontera se tratan dentro del código del micro-kernel principal.

En este trabajo, abordamos estas limitaciones demostrando que es posible generar automáticamente un conjunto de micro-kernels para la GEMM utilizando Exo [8]. Esta solución alternativa tiene las siguientes ventajas:

1. A alto nivel, el micro-kernel de la librería es "reemplazado" por una colección de códigos en C generado por Exo donde cada micro-kernel se encargará de un tamaño diferente.
2. Usando el backend apropiado, la generación/optimización puede ser fácilmente especializada para diferentes tipos de datos, lo que mejora aún más la portabilidad y mantenibilidad

[1]Universitat Politécnica de Valencia, e-mail: adcastel@disca.upv.es
[2]Cornell University, e-mail: jb2695@cornell.edu
[3]UC Berkeley, e-mail: dinh@berkeley.edu
[4]MIT CSAIL, e-mail: yuka@csail.mit.edu
[5]Universidad de Córdoba, e-mail: el2mapeh@uco.es

de la solución.

3. Al adaptar el tamaño del micro-kernel al problema, es posible superar las implementaciones de alto rendimiento de la GEMM de las bibliotecas existentes ampliamente aceptadas.
4. El proceso de optimización para cada problema se reduce enormemente, limitándose a evaluar una serie de micro-kernels generados automáticamente.
5. El generador de micro-kernels para ARM Neon está disponible públicamente en `https://github.com/adcastel/EXO_ukr_generator`.

Además, el trabajo desarrollado ha contribuido al código EXO añadiendo dos instrucciones intrínsecas Neon y el soporte para los tipos de datos de coma flotante de 16 bits para ARM[1].

El resto del documento se organiza de la siguiente manera. La sección II resume el algoritmo BLIS para la GEMM e introduce el lenguaje de programación específico EXO, además analiza algunos trabajos existentes sobre la generación de código; la sección III presenta, paso a paso, el proceso de generar códigos micro-kernels para la arquitectura ARM optimizados utilizando EXO; la sección IV evalúa y compara los micro-kernels generados con otros micro-kernels comúnmente utilizados en tres escenarios diferentes; y la sección V resume el trabajo realizado y sus contribuciones.

II. Antecedentes

A. Implementación de la GEMM en BLIS

Considere la GEMM $C = C + AB$, donde los operandos son matrices con las siguientes dimensiones: $m \times k$, $k \times n$, y $m \times n$ para A, B, y C, respectivamente. BLIS (así como otras bibliotecas LA) sigue el enfoque de GotoBLAS [1] para descomponer esta operación en tres bucles anidados alrededor de dos *rutinas de empaquetado* y un *macro-kernel*. En BLIS, el macro-kernel se descompone en dos bucles adicionales alrededor de un *micro-kernel*, consistiendo este último en un único bucle que realiza un *outer-product* por iteración. Las figuras 1 y 2 muestran el algoritmo base de BLIS para la GEMM, que incluye los seis bucles, las dos rutinas de empaquetamiento y el micro-kernel.

```
1  for (jc=0; jc<n; jc+=nc)  // Loop L1
2   for (pc=0; pc<k; pc+=kc) {    // L2
3    // Pack B
4    Bc := B(pc:pc+kc-1,jc:jc+nc-1);
5    for (ic=0; ic<m; ic+=mc) {   // L3
6     // Pack A
7     Ac := A(ic:ic+mc-1,pc:pc+kc-1);
8     for (jr=0; jr<nc; jr+=nr)   // L4
9       for (ir=0; ir<mc; ir+=mr) // L5
10       // Micro-kernel          // L6
11       C(ic+ir:ic+ir+mr-1,
12         jc+jr:jc+jr+nr-1)
13         += Ac(ir:ir+mr-1,0:kc-1)
14         *  Bc(0:kc-1,jr:jr+nr-1);
15   }}
```

Fig. 1: Pseudo-código del algoritmo de la GEMM en BLIS.

[1]El soporte para FP16 no estaba en el repositorio EXO en el momento de escribir este documento, pero está disponible en `https://github.com/adcastel/exo`

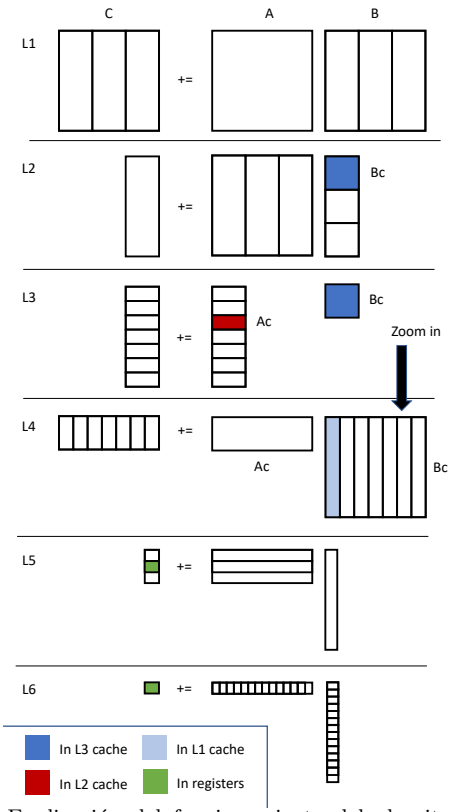


Fig. 2: Explicación del funcionamiento del algoritmo de la GEMM en BLIS.

Los tres bucles más externos del algoritmo recorren las dimensiones n–, k–, y m– del problema, particionando los operandos de la matriz para ajustarse a la jerarquía de caché del procesador. Los parámetros de configuración de la caché n_c, k_c, y m_c, adaptados a la arquitectura del procesador de destino [9], favorecen que el empaquetado B_c permanezca en la caché L3 y el empaquetado A_c en la caché L2 durante la ejecución del micro-kernel, mientras que C pasa de la memoria principal a los registros del procesador, como se muestra en la Figura 2. La función de las rutinas de empaquetado es que se pueda acceder a los datos de A_c, B_c desde el micro-kernel de forma secuencial.

Los micro-kernels son fragmentos de código orientado al hardware que suelen codificarse en lenguaje ensamblador o instrucciones intrínsecas. Dentro del micro-kernel, se actualiza un bloque de $m_r \times n_r$ elementos de C dentro del bucle k_c. Los operandos para la actualización son porciones de los buffers A_c y B_c, llamadas A_r y B_r y cuyas dimensiones son $m_r \times k_c$ y $k_c \times n_r$, respectivamente. Los valores de m_r y n_r se utilizan habitualmente para denominar el micro-kernel. BLIS proporciona micro-kernels especializados para muchos procesadores de AMD, Intel, ARM e IBM. Sin embargo, desarrollar un nuevo micro-kernel orientado al hardware para cada nueva arquitectura supone un costoso esfuerzo de desarrollo, por lo que es común que las bibliotecas solamente proporcionen un único micro-kernel por arquitectura.

B. Entornos de Compilación y Optimización

La gran cantidad de tiempo que supone desarrollar micro-kernels optimizados para diversas arquitecturas se debe, en gran medida, a la necesidad de realizar optimizaciones específicas para cada una de ellas con el fin de obtener el máximo rendimiento, Para resolver este problema, se han aplicado enfoques de generación automática de código con el fin de lograr la portabilidad del rendimiento en arquitecturas existentes y nuevas, incluidas las de propósito general o aceleradores de propósito específico [10] [11], con una intervención mínima del programador [12].

En concreto, los lenguajes y entornos de compilación programables por el usuario, como Halide [13], MLIR [14], TVM [15] o Exo [8], proponen una clara separación de actuación entre la definición de una operación (por ejemplo, una multiplicación de matrices) y el *planificador*, o conjunto de optimizaciones aplicadas a la operación. Esta división permite que las optimizaciones comunes (por ejemplo, *particionado de bucles*) se distancien de las optimizaciones específicas de la arquitectura como, por ejemplo, la vectorización.

Asimismo, los compiladores tradicionales también deben recibir una especificación del hardware que describa el conjunto de instrucciones del mismo (incluidas las vectoriales o intrínsecas). Por ejemplo Halide, TVM y LLVM, integran las especificaciones de hardware en el compilador, lo que requiere la especificación manual de generación de código para admitir nuevo hardware. Exo [8], por el contrario, externaliza la definición de intrínsecas del hardware tomando como entrada librerías definidas por el usuario. Por ejemplo, la Figura 3 proporciona definiciones para las funciones intrínsecas ARM Neon `vstlq_f32` y `vfmaq_laneq_f32` en Python semánticamente equivalentes; estas definiciones son utilizadas por el compilador para generar llamadas automáticamente. Una especificación de hardware completa consiste en definiciones similares para cada intrínseca así como definiciones para niveles de la jerarquía de memoria (en este caso, DRAM y registros Neon) y tipos de datos (en este caso, `f32` floats).

```
@instr("vst1q_f32(&{dst_data}, {src_data});")
def neon_vst_4xf32(dst: [f32][4] @ DRAM,
                   src: [f32][4] @ Neon):
  assert stride(src, 0) == 1
  assert stride(dst, 0) == 1

  for i in seq(0, 4):
    dst[i] = src[i]

@instr("{dst_data} = vfmaq_laneq_f32(
  {dst_data}, {lhs_data}, {rhs_data}, {l});")
def neon_vfmla_4xf32_4xf32(
  dst: [f32][4] @ Neon, lhs: [f32][4] @ Neon,
  rhs: [f32][4] @ Neon, l: index):

  assert stride(dst, 0) == 1
  assert stride(lhs, 0) == 1
  assert stride(rhs, 0) == 1
  assert l >= 0
  assert l < 4

  for i in seq(0, 4):
    dst[i] += lhs[i] * rhs[l]
```

Fig. 3: Especificación de funciones intrínsecas en Exo.

Un grave inconveniente para la integración del código generado por algunos de estos entornos es la dependencia de las bibliotecas existentes. Por ejemplo, el código generado por TVM utiliza un conjunto de objetos TVM que puede obligar a reescribir gran parte (o toda) la pila de software de una aplicación. Además, puede incurrir en sobrecargas en tiempo de ejecución para convertir los tipos de datos a los soportados por estas bibliotecas. Sin embargo, herramientas como Exo generan código C que puede utilizarse en todas las bibliotecas existentes.

C. Autoajuste y Optimización de Código

Al tratar los *planificadores* como entradas, los lenguajes programables por el usuario facilitan la exploración automática de espacios de programación mediante técnicas de autoajuste [16], [17]. Por ejemplo, AutoTVM [18], como parte de TVM, realiza una exploración completa de un espacio de búsqueda definido por una plantilla especificada por el usuario. Aunque estos métodos de búsqueda han demostrado un gran rendimiento en la práctica, deben enfrentarse a un espacio de búsqueda de posibles parámetros de ajuste que resulta muy costoso desde el punto de vista computacional. Además, esta búsqueda, crece exponencialmente con la dimensión del espacio de diseño, así como las dificultades de generalización (tanto para problemas de distintos tamaños como para distintos objetivos de hardware) y de explicación para los desarrolladores.

Recientemente, el trabajo realizado en [9] como parte del proyecto BLIS, ha demostrado que el uso de modelos analíticos para la selección de los parámetros de configuración es una forma eficaz de lograr un alto rendimiento sin necesidad de autoajuste. La sustitución del autoajuste por soluciones basadas en modelos también se ha explorado con éxito en [2], [19], [20], [21].

Nuestro trabajo, centrado en la generación de micro-kernels, difiere de [22] que utiliza MLIR para describir las primeras experiencias con el algoritmo GEMM completo, de [23] que propone esquemas avanzados de autoajuste para la GEMM, y de [24] que utiliza un script Python y macros en C para construir micro-kernels. En concreto, utilizamos Exo para ampliar y analizar en mayor profundidad la generación manual de micro-kernels para la GEMM e integrar el código resultante en un algoritmo de la GEMM similar a BLIS.

III. Generación de Código

En esta sección, explicamos, paso a paso, cómo construir desde cero un código de micro-kernel para la GEMM con Exo para la arquitectura ARMv8. Las dimensiones del micro-kernel generado son de 8×12 al igual que el existente en la biblioteca BLIS para esta arquitectura específica. Para cada fase de construcción, presentamos primero las instrucciones de Exo empleadas y explicamos el código intermedio resultante. También mostramos cómo ampliar la generación de código para manejar otras

características de los micro-kernels. El código para la generación paso a paso está disponible en `https://github.com/adcastel/EXO_ukr_generator`.

A. Generación de Micro-kernels

La Figura 4 muestra el aspecto del código inicial de un micro-kernel con la memoria almacenada por columnas (tal y como es común en las bibliotecas de LA) basado en el *outer product* (como en BLIS). Para cumplir las características del algoritmo GEMM de BLIS, hemos modificado el código inicial del micro-kernel de la siguiente manera: 1) Dado que el lenguaje C almacena los datos por filas, transponemos las dimensiones de la matriz C; 2) BLIS utiliza empaquetamientos de datos para los operandos A y B del micro-kernel que garantiza el acceso consecutivo a los datos. Por lo tanto, para tener esto en el operando A_c, también intercambiamos sus dimensiones. El oprando B_c ya se accede consecutivamente por lo que no se necesitan cambios; 3) Reorganizamos los bucles internos con la estructura k, j, i para que coincida con la estructura deseada. Observe que los operandos A_c y B_c se reservan en una estructura unidimensional en el algoritmo que llama al micro-kernel, por lo que aunque transpongamos los tamaños de los datos de C y A_c no hay riesgo en el patrón de acceso.

Para empezar con la explicación del código en la Figura 4, la primera línea (`@proc`) le dice al compilador EXO que la siguiente función es programable, y por lo tanto producirá un código en lenguaje C. Los argumentos de la función también muestran algunos aspectos del lenguaje EXO. En primer lugar, el nombre del argumento es seguido por ":" y el tipo, que puede ser `size, scalar, vector`, o `matrix`. Para los datos que requieren asignación de memoria, debemos especificar su ubicación. En este ejemplo, asignamos los búferes A_c, B_c y C a RAM utilizando la notación `@ DRAM`. Esta versión cubre todas las combinaciones de valores alfa y beta. Por lo tanto, los buffers C_b y B_a situados en las líneas 8 y 9 se utilizan para el cálculo de $C * beta$ y $B_c * alpha$, respectivamente. En concreto, las líneas 12–14 calculan los resultados C_b y las líneas 17–19 calculan los resultados B_a.

Las líneas 22–25 realizan el resultado del micro-kernel ejecutando $C_b = C_b + A_c \times B_a$. Finalmente, las líneas 28–30 devolverán el resultado del cálculo a la matriz C.

Para simplificar la explicación, a partir de este punto optimizaremos una versión específica del micro-kernel. En concreto, aplicaremos la transformación al código mostrado en la Figura 5 que corresponde al micro-kernel cuando los valores *alpha* y *beta* son ambos iguales a 1.

Para la optimización del código inicial, se requerirían más funciones de programación para los bucles C_b y B_a (líneas 12–14, 17–19, y 28–30), que serían equivalentes a las que se muestran a partir de este punto para otros bucles.

Micro-kernel Básico. En primer lugar, generamos la función a programar y especializamos el código generado especificando que queremos utilizar los valores de 8 y 12 para los argumentos M_R y N_R, respectivamente. La Figura 6 muestra el código de usuario y la representación generada. Las líneas 3 y 4 obtienen la versión inicial del micro-kernel y establecen las variables M_R y N_R, respectivamente. El código generado ha cambiado las variables por sus valores (por ejemplo, las iteraciones del segundo bucle ahora van de 0 a 12 en lugar de N_R).

```
1  @proc
2  def ukernel_ref( MR: size, NR: size, KC: size,
3    alpha: f32[1], Ac: f32[KC, MR] @ DRAM,
4    Bc: f32[KC, NR] @ DRAM, beta: f32[1],
5    C: f32[NR, MR] @ DRAM,
6    ):
7    # Tmp buffers for C * beta and B * alpha
8    Cb: f32[NR,MR] @ DRAM
9    Ba: f32[KC,NR] @ DRAM
10
11   # Cb = C * beta
12   for cj in seq(0, NR):
13     for ci in seq(0, MR):
14       Cb[cj,ci] = C[cj,ci] * beta[0]
15
16   # Ba = Bc * alpha
17   for bk in seq(0, KC):
18     for bj in seq(0, NR):
19       Ba[bk,bj] = Bc[bk,bj] * alpha[0]
20
21   # Cb += Ac * Ba
22   for k in seq(0, KC):
23     for j in seq(0, NR):
24       for i in seq(0, MR):
25         Cb[j, i] += Ac[k,i] * Ba[k,j]
26
27   # C = Cb
28   for cj in seq(0, NR):
29     for ci in seq(0, MR):
30       C[cj,ci] = Cb[cj,ci]
```

Fig. 4: Codigo del micro-kernel de la GEMM en formato EXO.

```
1  @proc
2  def ukernel_ref( MR: size, NR: size, KC: size,
3    alpha: f32[1], Ac: f32[KC, MR] @ DRAM,
4    Bc: f32[KC, NR] @ DRAM, beta: f32[1],
5    C: f32[NR, MR] @ DRAM,
6    ):
7
8    # C += Ac * Bc
9    for k in seq(0, KC):
10     for j in seq(0, NR):
11       for i in seq(0, MR):
12         C[j, i] += Ac[k,i] * Bc[k,j]
```

Fig. 5: Versión simplificada del micro-kernel en formato EXO.

```
1  # USER CODE
2  MR, NR = 8, 12
3  p = rename(ukernel_ref, "uk{}x{}".format(MR,NR))
4  p = p.partial_eval(MR,NR)
5
6  # RESULTING EXO GENERATED CODE
7  def uk_8x12( KC: size, alpha: f32[1] @ DRAM,
8    Ac: f32[KC, 8] @ DRAM, Bc: f32[KC, 12] @ DRAM,
9    beta: f32[1] @ DRAM, C: f32[12, 8] @ DRAM
10   ):
11
12   # C += Ac * Bc
13   for k in seq(0, KC):
14     for j in seq(0, 12):
15       for i in seq(0, 8):
16         C[j, i] += Ac[k, i] * Bc[k, j]
```

Fig. 6: Versión 1 del micro-kernel generado en formato EXO.

Estructura de Bucles. En las líneas 2 y 3 de la Figura 7 dividimos los bucles `i` y `j` para adaptarlos a la longitud del vector de la arquitectura. Como el procesador NVIDIA Carmel basado en ARM utiliza registros vectoriales de 128 bits y, con tipo de da-

tos de 32 bits, la longitud del vector es 4. Además, el algoritmo de la GEMM utilizado en este trabajo garantiza que los tamaños de búfer A_c y B_c sean múltiplos de los valores M_R y N_R, respectivamente. Esta acción da lugar a los bucles anidados `it`, `itt`, `jt` y `jtt` localizados en las líneas 14–17. Además, EXO ha generado automáticamente el acceso a los datos en C, A_c, y B_c para que coincida con la nueva estructura de bucle (líneas 18–20).

```
1  # USER CODE
2  p = divide_loop(p,'i', 4, ['it','itt'],...)
3  p = divide_loop(p,'j', 4, ['jt','jtt'],...)
4
5  # RESULTING EXO GENERATED CODE
6  def uk_8x12( KC: size, alpha: f32[1] @ DRAM,
7    Ac: f32[KC, 8] @ DRAM, Bc: f32[KC, 12] @ DRAM,
8    beta: f32[1] @ DRAM, C: f32[12, 8] @ DRAM
9    ):
10
11   # C += Ac * Bc
12   for k in seq(0, KC):
13     # Loop splits to match the vector length (4)
14     for jt in seq(0, 3):
15       for jtt in seq(0, 4):
16         for it in seq(0, 2):
17           for itt in seq(0, 4):
18             C[jtt + 4 * jt, itt + 4 * it] +=
19               Ac[k, itt + 4 * it] *
20               Bc[k, jtt + 4 * jt]
```

Fig. 7: Versión 2 del micro-kernel generado en formato EXO.

Matriz C. La Figura 8 muestra uno de los pasos más complejos en la generación del micro-kernel, la vinculación de la matriz C a los registros vectoriales, que incluye: declaración, carga y almacenamiento de los resultados. En concreto, el proceso es el siguiente:

1. Mapear la matriz C con un registro vectorial (líneas 3 y 4), que se refleja en la línea 51.
2. Redimensionar el registro vectorial a una estructura de 3 dimensiones donde cada dimensión corresponde a las iteraciones de cada bucle. En concreto, la primera instrucción (línea 7) redimensiona la declaración al tamaño de la longitud del vector, que en este caso es 4; la línea 8 completa el tamaño de la dimensión M_R; y la línea 9 está ligada a la dimensión N_R de C. El resultado de estas líneas puede verse en la línea 34, donde aparece la asignación final.
3. La sentencia `lift_alloc` de la línea 12 mueve la declaración de los registros de C a la parte superior del código generado.
4. Las líneas 15–18 mueven la carga y el almacenamiento de la matriz C fuera del bucle de cálculo (líneas 37–43 y líneas 56–62, respectivamente).
5. Las líneas 21 y 22 sustituyen los bucles de carga y almacenamiento por instrucciones intrínsecas. Las líneas 40 y 59 demuestran estas sustituciones.
6. La línea 25 establece la variable de registro C al tipo `Neon`.

Operandos A_c y B_c. La Figura 9 aplica las acciones para generar las cargas de A_c y B_c a registros. Observe que el código mostrado utiliza el nombre X_c por simplicidad ya que estas acciones deben realizarse para ambos operandos. El procedimiento para cada operando es el siguiente:

```
1  # USER CODE
2  #  1) Map C buffer to vectorial register C_reg
3  Cp = 'C[4 * jt + jtt, 4 * it + itt]'
4  p = stage_mem(p, 'C[_] += _', Cp, 'C_reg')
5
6  #  2) Build a 3D structure of C_reg
7  p = expand_dim(p, 'C_reg', 4, 'itt', ...)
8  p = expand_dim(p, 'C_reg', MR//4, 'it', ...)
9  p = expand_dim(p, 'C_reg', NR, 'jt*4+jtt', ...)
10
11 #  3) Move the register declaration to the top
12 p = lift_alloc(p, 'C_reg', n_lifts=5)
13
14 #  4) Move C load and store from the k-loop
15 p = autofission(p, p.find('C_reg[_]=_').after(),
16                  n_lifts=5)
17 p = autofission(p, p.find('C[_]=_').before(),
18                  n_lifts=5)
19
20 #  5) Replace the loops with Neon intrinsics
21 p = replace(p, 'for itt in _:_', neon_vld_4xf32)
22 p = replace(p, 'for itt in _:_', neon_vst_4xf32)
23
24 #  6) Set the C_reg memory to Neon
25 p = set_memory(p, 'C_reg', Neon)
26
27 # RESULTING EXO GENERATED CODE
28 def uk_8x12( KC: size, alpha: f32[1] @ DRAM,
29   Ac: f32[KC, 8] @ DRAM, Bc: f32[KC, 12] @ DRAM,
30   beta: f32[1] @ DRAM, C: f32[12, 8] @ DRAM
31   ):
32
33   # Registers for C
34   C_reg: f32[12, 2, 4] @ Neon
35
36   # Load C to registers
37   for jt in seq(0, 3):
38     for jtt in seq(0, 4):
39       for it in seq(0, 2):
40         neon_vld_4xf32(
41             C_reg[4 * jt + jtt, it, 0:4],
42             C[4 * jt + jtt, 4 * it:4 * it + 4]
43         )
44
45   # C += Ac * Bc
46   for k in seq(0, KC):
47     for jt in seq(0, 3):
48       for jtt in seq(0, 4):
49         for it in seq(0, 2):
50           for itt in seq(0, 4):
51             C_reg[jt * 4 + jtt, it, itt] +=
52               Ac[k, itt + 4 * it] *
53               Bc[k, jtt + 4 * jt]
54
55   # Store C from registers
56   for jt in seq(0, 3):
57     for jtt in seq(0, 4):
58       for it in seq(0, 2):
59         neon_vst_4xf32(
60             C[jtt + 4 * jt, 4 * it:4 * it + 4],
61             C_reg[jtt + 4 * jt, it, 0:4]
62         )
```

Fig. 8: Versión 3 del micro-kernel generado en formato EXO.

1. Mapear la matriz X_c a un registro vectorial (línea 3).
2. Redimensionar el registro vectorial a una estructura de dos dimensiones donde la primera dimensión es la longitud del vector y la segunda dimensión es el bucle más externo (líneas 6 y 7). El resultado de estas líneas se puede ver en las líneas 32 y 33 donde aparece la asignación final.
3. La sentencia `lift_alloc` traslada la declaración de los registros fuera del bucle k.
4. Las líneas 13 y 14 mueven la carga de los operandos al inicio del bucle k.
5. La línea 17 reemplaza los bucles `xtt` con instrucciones de carga vectorial.
6. La línea 20 establece la variable de registro de tipo `Neon`.

```
# USER CODE
#  1) Map Xc buffer to vectorial register X_reg
p = bind_expr(p, 'Xc[_]','X_reg')

#  2) Build a 2D structure of X_reg
p = expand_dim(p, 'X_reg' , 4, 'xtt', ...)
p = expand_dim(p, 'X_reg', XR//4, 'xt', ...)

#  3) Move the register declaration to the top
p = lift_alloc(p, 'X_reg', n_lifts=5)

#  4) Move the Xc load to the k-loop
p = autofission(p, p.find('X_reg[_]=_').after(),
                n_lifts=4)

#  5) Replace the xtt loop by Neon intrinsics
p = replace(p, 'for xtt in _:_', neon_vld_4xf32)

#  6) Set the X_reg memory to Neon
p = set_memory(p, 'X_reg', Neon)

# RESULTING EXO GENERATED CODE
def uk_8x12( KC: size, alpha: f32[1] @ DRAM,
  Ac: f32[KC, 8] @ DRAM, Bc: f32[KC, 12] @ DRAM,
  beta: f32[1] @ DRAM, C: f32[12, 8] @ DRAM
  ):

  # Registers for C and load C as in the
  # previous figure.  Omitted for brevity

  # Registers for Ac and Bc
  A_reg: R[2, 4] @ Neon
  B_reg: R[3, 4] @ Neon
  for k in seq(0, KC):
    # Load Ac and Bc to registers
    for it in seq(0, 2):
      neon_vld_4xf32(
        A_reg[it, 0:4],
        Ac[k, 4 * it:4 + 4 * it])
    for jt in seq(0, 3):
      neon_vld_4xf32(
        B_reg[jt, 0:4],
        Bc[k, 4 * jt:4 + 4 * jt])
    for jt in seq(0, 3):
      for jtt in seq(0, 4):
        for it in seq(0, 2):
          for itt in seq(0, 4):
            C_reg[jtt + 4 * jt, it, itt] +=
              A_reg[it, itt] * B_reg[jt, jtt]

  # Store C from registers as in the
  # previous figure.  Omitted for brevity
```

Fig. 9: Versión 4 del micro-kernel generado en formato Exo.

Cómputo del Micro-kernel. La Figura 10 ilustra este paso. Reordenamos los bucles `jtt` y `it` del cálculo (línea 2) para que el acceso a los valores de registro B_c sea secuencial. La línea 3 sustituye el bucle más interno para la sentencia `fmla` como se muestra en las líneas 24–26.

Desenrollado de Bucle. Aunque esta es una técnica que algunos compiladores aplican por defecto, también es posible hacerlo manualmente en Exo. La Figura 11 muestra un ejemplo de desenrollado para los bucles que cargan los operandos A_c y B_c en registros. Lo configuramos con las sentencias `unroll_loop` de las líneas 2 y 3, dando como resultado las líneas 21–25.

Codigo C Generado. Para asegurarnos de que el código generado no sólo está optimizado en lenguaje C, sino que además la compilación a ensamblador se realiza correctamente, hemos compilado el código con el comando `gcc-10 -S`, y el código ensamblador resultante para el bucle k se muestra en la Figura 12. Esta salida está tan optimizada como la implementada a mano en BLIS.

```
# USER CODE
p = reorder_loops(p,'jtt it')
p = replace(p, 'for itt in _: _',
                neon_vfmla_4xf32_4xf32)

# RESULTING EXO GENERATED CODE
def uk_8x12( KC: size, alpha: f32[1] @ DRAM,
  Ac: f32[KC, 8] @ DRAM, Bc: f32[KC, 12] @ DRAM,
  beta: f32[1] @ DRAM, C: f32[12, 8] @ DRAM
  ):

  # Registers for C, Ac and Bc and loads as in
  # previous figures.  Omitted for brevity

  # C += Ac * Bc
  for k in seq(0, KC):
    # Load Ac and Bc to registers as in
    # previous figures. Omitted for brevity

    # Computation with registers
    for jt in seq(0, 3):
      for it in seq(0, 2):
        for jtt in seq(0, 4):
          neon_vfmla_4xf32_4xf32(
            C_reg[jtt + 4 * jt, it, 0:4],
            A_reg[it, 0:4], B_reg[0:4, jt], jtt)

  # Store C from registers as in
  # previous figures.  Omitted for brevity
```

Fig. 10: Versión 5 del micro-kernel generado en formato Exo.

```
# USER CODE
p = unroll_loop(p,'it')
p = unroll_loop(p,'jt')

# RESULTING EXO GENERATED CODE
def uk_8x12( KC: size, alpha: f32[1] @ DRAM,
  Ac: f32[KC, 8] @ DRAM, Bc: f32[KC, 12] @ DRAM,
  beta: f32[1] @ DRAM, C: f32[12, 8] @ DRAM
  ):

  # Registers for C and load C as in
  # previous figures.  Omitted for brevity

  # Registers for Ac and Bc
  A_reg: R[2, 4] @ Neon
  B_reg: R[3, 4] @ Neon

  # C += Ac * Bc
  for k in seq(0, KC):
    # Unrolled loads from Ac and Bc to registers
    neon_vld_4xf32(A_reg[0,0:4], Ac[k,0:4+0])
    neon_vld_4xf32(A_reg[1,0:4], Ac[k,4:4+4])
    neon_vld_4xf32(B_reg[0,0:4], Bc[k,0:4+0])
    neon_vld_4xf32(B_reg[1,0:4], Bc[k,4:4+4])
    neon_vld_4xf32(B_reg[2,0:4], Bc[k,8:4+8])
    # Computation with registers
    for jt in seq(0, 3):
      for it in seq(0, 2):
        for jtt in seq(0, 4):
          neon_vfmla_4xf32_4xf32(
            C_reg[jtt + 4 * jt, it, 0:4],
            A_reg[it, 0:4], B_reg[0:4, jt], jtt)

  # Store C from registers as in
  # previous figures.  Omitted for brevity
```

Fig. 11: Versión 6 del micro-kernel generado en formato Exo.

B. Casos Frontera

Uno de los problemas del enfoque de un único micro-kernel por arquitectura adoptado por las bibliotecas de LA es la degradación del rendimiento cuando las dimensiones del micro-kernel no coinciden con las optimizadas. Esta situación se denomina casos frontera. Soluciones como BLIS utilizan una versión no especializada del micro-kernel para estas situaciones porque entienden que no tienen impacto en el rendimiento para problemas de gran tamaño. Sin embargo, los escenarios HPC más recientes, como DL, están repletos de estos escenarios. Un ejemplo claro son los tamaños de la primera capa del modelo

```
.L3:
  ldp     q1, q0, [x3]
  add     x0, x0, 1
  ldp     q4, q3, [x4]
  add     x3, x3, 32
  ldr     q2, [x4, 32]
  add     x4, x4, 48
  fmla    v12.4s, v1.4s, v4.s[0]
  fmla    v10.4s, v1.4s, v4.s[1]
  fmla    v8.4s, v1.4s, v4.s[2]
  fmla    v30.4s, v1.4s, v4.s[3]
  fmla    v11.4s, v0.4s, v4.s[0]
  fmla    v9.4s, v0.4s, v4.s[1]
  fmla    v31.4s, v0.4s, v4.s[2]
  fmla    v29.4s, v0.4s, v4.s[3]
  fmla    v28.4s, v1.4s, v3.s[0]
  fmla    v26.4s, v1.4s, v3.s[1]
  fmla    v24.4s, v1.4s, v3.s[2]
  fmla    v22.4s, v1.4s, v3.s[3]
  fmla    v27.4s, v0.4s, v3.s[0]
  fmla    v25.4s, v0.4s, v3.s[1]
  fmla    v23.4s, v0.4s, v3.s[2]
  fmla    v21.4s, v0.4s, v3.s[3]
  fmla    v20.4s, v1.4s, v2.s[0]
  fmla    v18.4s, v1.4s, v2.s[1]
  fmla    v16.4s, v1.4s, v2.s[2]
  fmla    v6.4s, v1.4s, v2.s[3]
  fmla    v19.4s, v0.4s, v2.s[0]
  fmla    v17.4s, v0.4s, v2.s[1]
  fmla    v7.4s, v0.4s, v2.s[2]
  fmla    v5.4s, v0.4s, v2.s[3]
  cmp     x1, x0
  bne     .L3
```

Fig. 12: Ensamblador generado con el compilador gcc-10 a partir del micro-kernel obtenido con Exo.

convolucional ResNet50-v1.5, donde después de aplicar el método im2row, las dimensiones de la gemm resultantes son $12544, 64$, y 147 para M, N, y K, respectivamente.

Usando Exo, y suponiendo que los empaquetados de BLIS están en el algoritmo de la gemm, todo lo que tenemos que hacer es utilizar el código que se muestra en la Figura 6 y cambiar los valores de M_R y N_R para que coincida con los casos especiales. Posteriormente, la ejecución de todos los pasos generará un nuevo micro-kernel que coincide con esos valores.

Sin embargo, es posible que no necesitemos el proceso de empaquetado porque los datos ya están con el patrón de acceso deseado o el tamaño del problema es lo suficientemente pequeño como para que el coste del empaquetado no merezca la pena. En este caso, deberíamos ajustar el procedimiento de generación del micro-kernel (por ejemplo, no empaquetar A) de la siguiente manera:

1. El bucle `i` de la Figura 7 no debe dividirse.
2. El mapeo entre A_c y `A_reg` cambiará y las dimensiones de este último coincidirá con el valor de M_R.
3. Dentro del bucle k, `A_reg` se difundirá con los valores A_c.
4. El cálculo utilizará el `neon_vfmadd_4xf32_4xf32`, que computa el valor `A_reg` por todo el `B_reg`.

Con estos cambios, generaremos un micro-kernel que no necesita empaquetado de A.

C. Portabilidad entre Arquitecturas

Una de las ventajas de la autogeneración es que, una vez construido el patrón de pasos para obtener un código, basta con cambiar las intrínsecas invocadas. Concretamente, con Exo sólo se debe cambiar el tercer argumento en la instrucción `replace` en el código de usuario para generar el código para la arquitectura deseada. Si la nueva arquitectura proporciona una interfaz de programación de aplicaciones (API del inglés application programming interface) con la misma funcionalidad, este es el único cambio necesario. Sin embargo, es posible que una instrucción intrínseca utilizada en este ejemplo no esté presente en la API de otro hardware (por ejemplo, ARM Neon `vfmaq_laneq_f32`). En ese escenario, se utiliza un desarrollo similar al presentado cuando no se dispone de empaquetado de datos.

Como ejemplo sencillo que ilustra la portabilidad, cambiando la línea 21 de la Figura 8 de `replace(p, 'for itt in _: _', neon_vld_4xf32)` por `replace(p, 'for itt in _: _', _mm512_loadu_ps)` cambiará la carga intrínseca de ARM Neon a Intel AVX512.

D. Tipos de Datos

Generar micro-kernels para diferentes tipos de datos es tan fácil como utilizar la función `set_precision` para cada asignación de memoria y registro en el código. Por ejemplo, `set_precision(p, A_reg, "f16")` utilizará registros de coma flotante de 16 bit para la variable `A_reg`. Esta característica soluciona la falta de códigos especializados para distintos tipos de datos en las bibliotecas de LA existentes.

IV. Evaluación de Rendimiento

En esta sección evaluamos el rendimiento de las rutinas para la gemm y los micro-kernels de 8×12 comparando tres implementaciones de micro-kernels diferentes: 1)`Neon`: un microkernel implementado con instrucciones `Neon` y desarrollado a mano; 2)`BLIS`: el micro-kernel de BLIS v.0.9; y 3) Exo, el código generado automáticamente presentado en la Sección III. Los experimentos de esta sección se realizaron en un único núcleo del procesador NVIDIA Carmel (ARM v8.2) integrado en una placa NVIDIA Jetson AGX Xavier, utilizando aritmética de coma flotante IEEE de 32 bits (FP32).

Los experimentos incluyen tres tipos de problemas para la gemm: rendimiento aislado del micro-kernel (incluidos los casos frontera); matrices cuadradas de gran tamaño; y problemas altamente "rectangulares". El código para ejecutar la prueba en modo aislado está disponible en `https://github.com/adcastel/driver_uk_blis`, y las pruebas cuadradas y rectangulares están disponibles en `https://github.com/martineh/gemm_blis_family.git`.

A. Modo Aislado

En este experimento, demostramos que los micro-kernels generados por Exo son tan eficientes como los codificados a mano. Para ello, ejecutamos los micro-kernels directamente durante 5 segundos y luego calculamos los GFLOPS. Las columnas 8×12 de la Figura 13 muestran el rendimiento cuando el

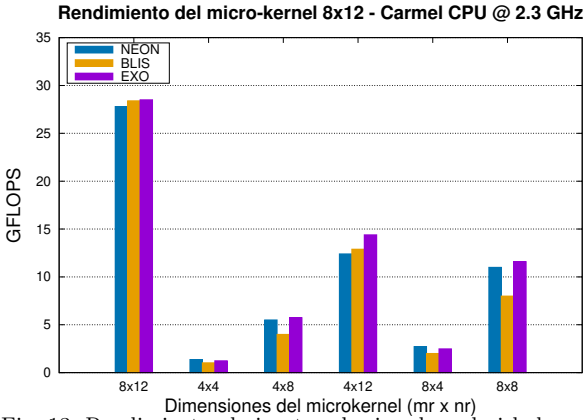


Fig. 13: Rendimiento al ejecutar el micro-kernel aislado para distintos tamaños de M_R y N_R.

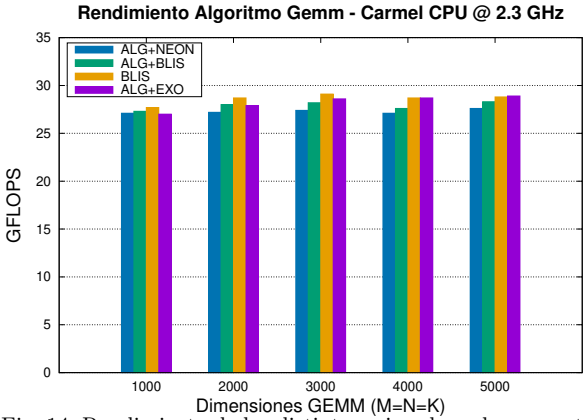


Fig. 14: Rendimiento de los distintos micro-kernels con matrices cuadradas.

micro-kernel se invoca a su máximo rendimiento. Para esta prueba, hemos fijado K_c en 512, que es el valor que utiliza BLIS para esta arquitectura ARM. Existen pequeñas diferencias entre las tres soluciones. En primer lugar, `NEON` es más lento que `BLIS`, y la principal diferencia es que el primero está escrito con intrínsecas Neon mientras que el segundo está en ensamblador. Exo es ligeramente superior porque sólo soporta el caso exacto de 8×12, mientras que los otros dos micro-kernels también incluyen la lógica para los casos frontera.

Las demás columnas representan el rendimiento al llamar a los micro-kernels con distintos casos frontera. Mientras que `NEON` y `BLIS` ejecutan el mismo micro-kernel que en el caso de 8×12, Exo se beneficia de la facilidad para generar diferentes tamaños de micro-kernel y, por tanto, se ejecuta un micro-kernel especializado para cada tamaño del problema. Este enfoque es claramente la mejor solución para superar estos escenarios.

B. Matrices Cuadradas

La Figura 14 muestra el rendimiento para una ejecución completa del algoritmo GEMM con los distintos micro-kernels. Las columnas con el prefijo `ALG+` indican que hemos utilizado una realización basada en BLIS del algoritmo GEMM que también incluye el modelo teórico presentado en [9] para optimizar los empaquetamientos. El sufijo de estas columnas indica el micro-kernel (o micro-kernels en el caso de Exo) utilizados. Además, la columna etiquetada como `BLIS` es el rendimiento al llamar a la función GEMM de la biblioteca BLIS.

`BLIS` obtiene mejores resultados en este caso porque el algoritmo GEMM utilizado en la biblioteca BLIS implementa la precarga de datos dentro del micro-kernel que no se utiliza en la variante `ALG+BLIS`. La combinación `ALG+`Exo supera a los otros `ALG+`. Si consideramos que los empaquetamientos de los operandos del micro-kernel son iguales debido al modelo, la única diferencia es el uso de micro-kernels diferentes (Exo) o un micro-kernel para todos los casos. En concreto, para el enfoque de Exo hemos utilizado (además del micro-kernel 8×12) el micro-kernel 8×4 para los tamaños de problema 1.000 y 4.000 y el micro-kernel 8×8 para los tamaños de problema 2.000 y 5.000.

Id.	Número en VGG16	m	n	k
1	01	50,176	64	27
2	03	50,176	64	576
3	06	12,544	128	576
4	08	12,544	128	1,152
5	11	3,136	256	1,152
6	13/15	3,136	256	2,304
7	18	784	256	2,304
8	20/22	784	512	4,608
9	25/27/29	196	512	4,608

Tabla I: Dimensiones de la GEMM al aplicar el método IM2ROW a las capas convolucionales del modelo VGG16 con el número de imágenes a 1.

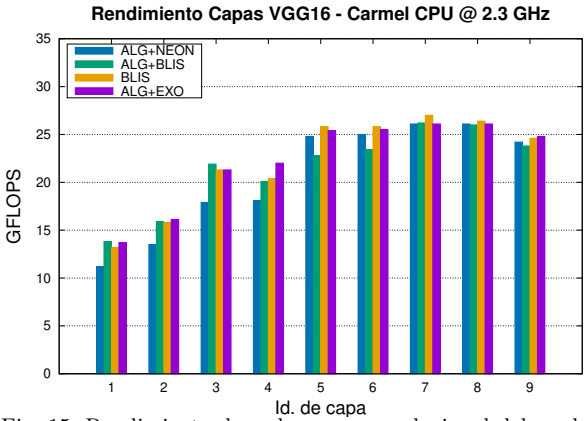


Fig. 15: Rendimiento de cada capa convolucional del modelo VGG16 con distintas versiones de micro-kernels

C. Matrices Rectangulares

Dado el interés actual en la inferencia en DL, las dimensiones de este experimento se eligen para ser las que se obtienen aplicando la transformación IM2ROW [25] a las capas de convolución en el modelo DNN VGG16 en forma de una GEMM. El número de muestras para el escenario de inferencia se establece en 1 muestra por lote. Dado que algunas capas comparten los mismos parámetros dando lugar a problemas idénticos de GEMM, los resultados para estos sólo se reportan una vez; ver Tabla I como referencia.

La Figura 15 refleja las ventajas de los micro-kernels ad hoc para casos frontera. La implementación `ALG+`Exo es la mejor opción para para 3 capas, `BLIS` con precarga en cuatro, mientras que el `ALG+BLIS` es el mejor en dos de ellas.

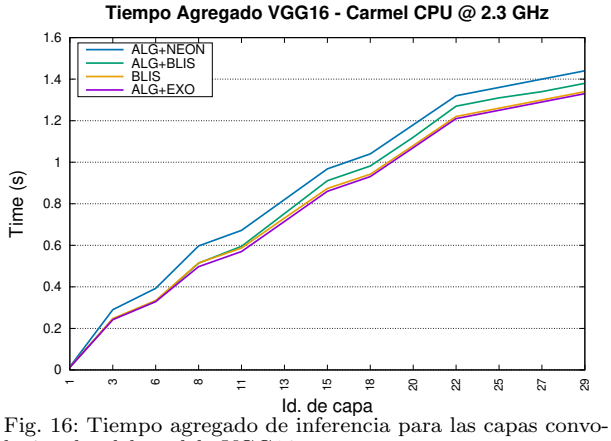


Fig. 16: Tiempo agregado de inferencia para las capas convolucionales del modelo VGG16

Para poner estos resultados en términos de rendimiento absoluto, la Figura 16 muestra el tiempo agregado para toda la ejecución del modelo. Aunque la diferencia es pequeña, el mejor rendimiento se consigue utilizando `ALG+`Exo y `BLIS` (con precarga), seguido de `ALG+BLIS` y `ALG+NEON`.

V. Conclusiones

En este artículo, hemos abordado el problema del enfoque monolítico adoptado por las bibliotecas de LA mediante la generación de código C de micro-kernels específicos y orientados al hardware utilizando la herramienta Exo para la arquitectura ARM8. Hemos descrito, paso a paso, el proceso para construir un generador de micro-kernels desde cero que produce un código C con un rendimiento cercano (o incluso mejor) que el propio de la biblioteca BLIS. También hemos proporcionado las explicaciones para adaptar el generador de micro-kernels a diferentes requisitos de software, como diferentes tipos de datos. Hemos analizado la comparación de rendimiento en tres escenarios diferentes: ejecución de micro-kernels, multiplicaciones de matrices grandes y cuadradas, y las operaciones GEMM rectangulares generadas por los modelos convolucionales frente a micro-kernels basados en instrucciones intrínsecas de Neon y en ensamblador. El generador de micro-kernels está disponible en `https://github.com/adcastel/EXO_ukr_generator`. Además, este trabajo ha contribuido al código de la herramienta Exo con el soporte de algunas características ARM y del tipo de datos de coma flotante de 16-bits. Como trabajo futuro, planeamos abordar la generación de otras piezas de código (o micro-kernels) para librerías LA o códigos de dominios específicos como códigos convolucionales. Además, se pretende adaptar este generador para arquitecturas RISC-V.

Agradecimientos

Este trabajo no hubiera sido posible sin la activa colaboración del Dr. Gilbert Bernstein de la University of Washington y Dr. Jonathan Ragan-Kelley del Massachusetts Institute of Technology. El trabajo de Adrián Castelló está soportado por la ayuda FJC2019-039222-I financiada por MCIN/AEI/10.13039/ 501100011033 del Ministerio de Ciencia e Innovación. Yuka Ikarashi recibe financiación de Funai Overseas Scholarship, Masason Foundation fellowship, y Great Educators fellowship. Héctor Martínez está financiado por la *Consejería de Transformación Económica, Industria, Conocimiento y Universidades de la Junta de Andalucía.*

Referencias

[1] K. Goto and R. A. van de Geijn, "Anatomy of a high-performance matrix multiplication," *ACM Trans. Math. Softw.*, vol. 34, no. 3, pp. 12:1–12:25, May 2008.

[2] Z. Xianyi, W. Qian, and Z. Yunquan, "Model-driven level 3 BLAS performance optimization on Loongson 3A processor," in *2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)*, 2012.

[3] F. G. Van Zee and R. A. van de Geijn, "BLIS: A framework for rapidly instantiating BLAS functionality," *ACM Trans. Math. Softw.*, vol. 41, no. 3, pp. 14:1–14:33, 2015.

[4] R. C. Whaley and J. J. Dongarra, "Automatically tuned linear algebra software," in *Proc. ACM/IEEE Conference on Supercomputing*, ser. SC '98. USA: IEEE Computer Society, 1998, p. 1–27.

[5] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[6] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 65:1–65:43, 2019.

[7] F. G. V. Zee, T. M. Smith, B. Marker, T. M. Low, R. A. V. D. Geijn, F. D. Igual, M. Smelyanskiy, X. Zhang, M. Kistler, V. Austel, J. A. Gunnels, and L. Killough, "The BLIS framework: Experiments in portability," *ACM Trans. Math. Softw.*, vol. 42, no. 2, Jun. 2016. [Online]. Available: https://doi.org/10.1145/2755561

[8] Y. Ikarashi, G. L. Bernstein, A. Reinking, H. Genc, and J. Ragan-Kelley, "Exocompilation for productive programming of hardware accelerators," in *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, ser. PLDI 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 703–718. [Online]. Available: https://doi.org/10.1145/3519939.3523446

[9] T. M. Low, F. D. Igual, T. M. Smith, and E. S. Quintana-Ortí, "Analytical modeling is enough for high-performance BLIS," *ACM Trans. Math. Softw.*, vol. 43, no. 2, pp. 12:1–12:18, Aug. 2016.

[10] T. Moreau, T. Chen, Z. Jiang, L. Ceze, C. Guestrin, and A. Krishnamurthy, "VTA: an open hardware-software stack for deep learning," *CoRR*, vol. abs/1807.04188, 2018. [Online]. Available: http://arxiv.org/abs/1807.04188

[11] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proceedings of the 58th Annual Design Automation Conference (DAC)*, 2021.

[12] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, and D. Qian, "The deep learning compiler: A comprehensive survey," *CoRR*, vol. abs/2002.03794, 2020. [Online]. Available: https://arxiv.org/abs/2002.03794

[13] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 519–530. [Online]. Available: https://doi.org/10.1145/2491956.2462176

[14] C. Lattner, J. A. Pienaar, M. Amini, U. Bondhugula, R. Riddle, A. Cohen, T. Shpeisman, A. Davis, N. Vasilache, and O. Zinenko, "MLIR: A compiler infrastructure for the end of moore's law," *CoRR*, vol. abs/2002.11054, 2020. [Online]. Available: https://arxiv.org/abs/2002.11054

[15] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Q. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: end-to-end optimization stack for deep learning," *CoRR*, vol. abs/1802.04799, 2018. [Online]. Available: http://arxiv.org/abs/1802.04799

[16] R. T. Mullapudi, A. Adams, D. Sharlet, J. Ragan-Kelley, and K. Fatahalian, "Automatically scheduling halide image processing pipelines," *ACM Trans. Graph.*, vol. 35, no. 4, jul 2016. [Online]. Available: https://doi.org/10.1145/2897824.2925952

[17] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen, J. E. Gonzalez, and I. Stoica, "Ansor: Generating High-Performance Tensor Programs for Deep Learning," arXiv, Tech. Rep., arXiv:2006.06762 [cs, stat] type: article. [Online]. Available: http://arxiv.org/abs/2006.06762

[18] T. Chen, L. Zheng, E. Q. Yan, Z. Jiang, T. Moreau, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Learning to optimize tensor programs," *CoRR*, vol. abs/1805.08166, 2018. [Online]. Available: http://arxiv.org/abs/1805.08166

[19] K. Yotov, X. Li, M. J. Garzarán, D. Padua, K. Pingali, and P. Stodghill, "Is search really necessary to generate high-performance BLAS?" *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, vol. 93, no. 2, 2005.

[20] A. Olivry, G. Iooss, N. Tollenaere, A. Rountev, P. Sadayappan, and F. Rastello, "IOOpt: automatic derivation of i/o complexity bounds for affine programs," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. ACM, jun 2021. [Online]. Available: https://doi.org/10.1145/3453483.3454103

[21] Q. Huang, M. Kang, G. Dinh, T. Norell, A. Kalaiah, J. Demmel, J. Wawrzynek, and Y. S. Shao, "Cosa: Scheduling by constrained optimization for spatial accelerators," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 554–566.

[22] U. Bondhugula, "High performance code generation in MLIR: an early case study with GEMM," *CoRR*, vol. abs/2003.00532, 2020. [Online]. Available: https://arxiv.org/abs/2003.00532

[23] Y. Zhang, *Parallel solution of integral equation-based EM problems in the frequency domain*. IEEE Press, 2009.

[24] G. Alaejos, A. Castelló, H. Martínez, P. Alonso-Jordá, F. D. Igual, and E. S. Quintana-Ortí, "Micro-kernels for portable and efficient matrix multiplication in deep learning," *The Journal of Supercomputing*, pp. 1–24, 2022.

[25] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *International Workshop on Frontiers in Handwriting Recognition*, 2006.

Gestión dinámica de procesos y comunicadores en aplicaciones MPI maleables

Javier Fernández Muñoz[1], Alberto Cascajo García[1] y Jesús Carretero Pérez[1]

Resumen— **Se puede definir una aplicación maleable como aquella que puede aumentar o reducir los recursos utilizados en base a la variación de la carga de trabajo. Estas aplicaciones suelen utilizar las funcionalidades de los gestores de trabajos para gestionar los recursos. En el caso de las aplicaciones MPI, la última versión del estándar propone algunas opciones para incrementar los procesos de la misma. Sin embargo no deja claro como utilizar esas mismas funcionalidades para reducir dichos procesos. En este trabajo se presenta una estrategia compatible con la última versión del estándar MPI para desarrollar aplicaciones MPI maleables que permita añadir y eliminar recursos al nivel de nodo de computo de forma discrecional. La evaluación realizada muestra que la estrategia propuesta tiene un rendimiento y escalabilidad aceptables para la funcionalidad que proporciona.**

Palabras clave— **MPI, Maleabilidad, Aplicaciones reconfigurables, Gestión de recursos HPC**

I. Introducción

Los sistemas de computación de alto rendimiento (HPC) han crecido enormemente en los últimos años. Esto obliga a las aplicaciones a adaptarse a los nuevos sistemas donde el número de procesadores crece sin parar, conectados por redes de comunicaciones cada vez más rápidas.

Estos sistemas están diseñados para dar servicio a la vez a un número cada vez mayor de aplicaciones. Para ello, se utilizan aplicaciones de gestión de recursos y trabajos que permiten seleccionar las aplicaciones a ejecutar en cada instante. A medida que aumentan los recursos de los sistemas HPC, es necesario ejecutar un mayor número de aplicaciones simultaneas para mantener un uso eficiente de los recursos. Esto a su vez complica la planificación simultanea de múltiples trabajos con necesidades de recursos, tiempos de ejecución y cargas de trabajo muy diferentes entre si.

Una posible solución es la introducción de tareas maleables. Una tarea maleable se define como aquella que puede aumentar o reducir sus recursos durante su ejecución, en función de sus necesidades y/o las necesidades de planificación global del sistema. La maleabilidad puede aumentar las posibilidades del planificador para aumentar la efectividad del sistema. Por ejemplo, una aplicación que esta en ejecución puede aumentar sus recursos en el momento que otras aplicaciones terminen y así reducir su tiempo de ejecución total. Otro ejemplo seria que mientras el planificador esta esperando a que se liberen suficientes nodos para ejecutar una aplicación de gran tamaño, las aplicaciones en ejecución pueden utilizar temporalmente los nodos libres para mejorar su rendimiento con la condición de liberarlos en el momento que se considere que la aplicación en espera tiene suficientes recursos para ser ejecutada.

Uno de los entornos de desarrollo más utilizado para aplicaciones HPC es MPI. Esta plataforma facilita enormemente el desarrollo de aplicaciones paralelas distribuidas gracias a elementos como los comunicadores MPI que permiten conectar colectivamente todos los procesos de la aplicación. El entorno MPI se diseño inicialmente para el lanzamiento de aplicaciones con un número fijo de procesos totalmente conectados entre si. Ahora bien, con el paso del tiempo se ha introducido en el estándar diversas funcionalidades que permiten generar dinámicamente nuevos procesos y conectarlos a la aplicación MPI. Sin embargo hasta el momento no se ha establecido en el estándar un solución definitiva para el desarrollo de aplicaciones maleables. Por un lado, el estándar MPI propone algunas opciones para incrementar los procesos de una aplicación. Sin embargo no deja claro cual es la forma correcta para reducir dicho número de procesos.

Existen otras opciones similares a la maleabilidad como pueden ser técnicas basadas en el checkpointing y el relanzamiento de las aplicaciones. Estas técnicas son más fáciles de implementar pero carecen de la flexibilidad y rendimiento de las aplicaciones propiamente maleables [1]. En la actualidad, las aplicaciones maleables son poco frecuentes [2].

En este trabajo se presenta una estrategia que es compatible con la última versión del estándar MPI (en concreto la versión 4.0 [3]) y que permite el desarrollo de aplicaciones MPI maleables. Estas técnicas permiten una maleabilidad ajustable al nivel de nodo de computo. Además, permite aprovecharse de la integración existente entre los entornos de desarrollo MPI modernos y los gestores de trabajos HPC existentes.

La sección II ilustra varias propuestas del estado del arte relacionadas. La sección III muestra los principales conceptos utilizados en este trabajo. La sección IV presenta la estrategia propuesta para el desarrollo de aplicaciones MPI maleables. La sección V muestra la evaluación de las técnicas propuestas. Por último, la sección VI muestra las conclusiones y trabajos futuros.

II. Estado del arte

En la literatura existen diversas propuestas para desarrollar aplicaciones MPI maleables. Por un lado, existen diversas opciones basadas en el relanzamiento de la aplicación una vez se ha realizado un check-

[1]Dpto. de Informática,Universidad Carlos III de Madrid, e-mail:{javier.fernandez, alberto.cascajo, jesus.carretero}@uc3m.es

point de los datos. Por ejemplo, está la propuesta denominada PCM (Process Checkpointing and Migration)[4]. Este trabajo concibe la maleabilidad como una sucesión de operaciones de dividir e integrar procesos a base de añadir llamadas del entorno PCM en la propia aplicación. Otras opciones que siguen un esquema parecido incluyen: la técnica de Checkpointing/relanzamiento escalable [5], la propuesta denominada SRS (Stop Restart Software) [6] y la de Adaptive MPI (AMPI)[7]. En general estas técnicas suelen seguir la estrategia de guardar los datos en disco, terminar los procesos existentes, relanzar una nueva serie de procesos y recuperar los datos del disco. Existe otra propuesta [8] que combina AMPI con el gestor de recursos Torque/Maui y lo extiende para su uso con trabajos maleables.

Otra opción es, por ejemplo, Elastic MPI [9] que incluye una infraestructura y un conjunto de extensiones de MPI para la ejecución de aplicaciones MPI maleables. Esta implementación esta basada en MPICH y Slurm así que su principal desventaja es ser totalmente dependiente de ambos.

Otra propuesta del estado del arte es FlexMPI [10], [11] la cual implementa extensiones para facilitar la implementación de aplicaciones maleables. FlexMPI también proporciona monitorización de las aplicaciones, diferentes estrategias automáticas para seleccionar que procesos modificar y cuando, y proporciona redistribución de datos con balanceo de carga.

ULFM (User Level Failure Migration) [12] es otra de las opciones existentes. ULFM es una extensión no estándar de MPI para incorporar maleabilidad integrando los mecanismos de tolerancia a fallos incluidos en ULFM con los mecanismos de reconfiguración dinámica existentes en MPI (*MPI_Comm_spawn*)

Mención especial merece el trabajo realizado en [13]. Este trabajo analiza en detalle todas las funcionalidades presentes en el estándar MPI que pueden ser relevantes para el desarrollo de aplicaciones maleables, así como analizar diferentes estrategias para implementar dichas aplicaciones.

III. Antecedentes

A. Gestión dinámica de procesos en MPI

Al principio, el estándar MPI no permitía la creación de nuevos procesos durante la ejecución de una aplicación. A partir de la versión 2 se han añadido nuevas funcionalidades para ampliar dinámicamente los procesos de una aplicación MPI. En su concepto más puro, una aplicación MPI esta basada en el modelo SPMD (Single Program Multiple Data). Sin embargo, con la inclusión de estas nuevas funcionalidades, las aplicaciones pueden seguir tanto un modelo SPMD, con número dinámico de procesos, como cambiar a un modelo MPMD (Multiple programs Multiple Data) donde los nuevos procesos pueden ejecutar un programa totalmente distinto del original.

El estándar MPI (hasta la version 4.0) implementa dos estrategias distintas para gestionar dinámicamente el numero de procesos. Estas estrategias se pueden enumerar como:

1. Creación de nuevos procesos, engendrados desde el propio código de la aplicación MPI, e integrados posteriormente como parte de dicha aplicación.
2. Lanzamiento de dos aplicaciones MPI, de forma independiente, que se integran posteriormente en una única aplicación.

En el primer caso, la semántica es similar a la creación de un proceso en UNIX. Para ello se utiliza la llamada *MPI_Comm_spawn* o *MPI_Comm_spawn_multiple*. Estas llamadas permiten lanzar una nuevo grupo de procesos, ejecutando el programa indicado como parámetro (el mismo o uno distinto). Este nuevo grupo de procesos se comporta en muchos aspectos como una nueva aplicación MPI; como por ejemplo al disponer de su propio comunicador *MPI_COMM_WORLD*. La principal diferencia con el lanzamiento de una nueva aplicación, es que la llamada *MPI_Comm_spawn* (procesos padre) y la llamada *MPI_Comm_get_parent* (procesos hijo) devuelven un comunicador especial (intercomunicador), para integrar ambos grupos de procesos en una única aplicación.

En el segundo caso la semántica es similar a la utilizada para los sockets. Primero, se parte de dos aplicaciones MPI que están ya ejecutando, una actuando como servidor, y otra como cliente. En la aplicación servidor, un proceso abre un puerto de comunicaciones, con la llamada *MPI_Open_port*, y obtiene su dirección textual. A continuación, sus procesos ejecutan la llamada *MPI_Comm_accept*, esperando una solicitud de conexión. El texto que contiene la dirección del puerto es compartido, por cualquier medio, con la aplicación cliente. Finalmente, los procesos de la aplicación cliente ejecutan la llamada *MPI_Comm_connect*, creando la conexión con la aplicación servidor. En este caso, ambas aplicaciones reciben, también como resultado, un comunicador especial (intercomunicador).

En ambos casos, los nuevos procesos se comportan, de forma real o virtual, como una nueva aplicación MPI, para integrarse con la ya existente. Esta integración se realiza, utilizando una funcionalidad de MPI conocida como intercomunicadores, en contraposición con los comunicadores normales o intracomunicadores. Un intercomunicador, permite conectar dos intracomunicadores, de dos aplicaciones MPI distintas, mediante una conexión punto a punto. Cualquier comunicación que use un intercomunicador será, siempre, entre procesos situados a ambos lados de dicha conexión. Es posible convertir un intercomunicador en un intracomunicador. Sin embargo el estándar no garantiza que su rendimiento sea equivalente al de un intracomunicador puro.

A continuación, enumeraremos las características más relevantes de ambas estrategias, relacionadas con el objetivo de crear y gestionar aplicaciones MPI maleables:

- La llamada *MPI_Comm_spawn* (procesos padre)

y la llamada *MPI_Init* (procesos hijo) son, potencialmente, colectivas entre si. Además la llamada *MPI_Comm_get_parent* solo generar comunicadores válidos para la "sesión"principal (*MPI_Init*/ *MPI_Finalize*). Esto implica que no se puede usar los comunicadores creados con *MPI_Comm_spawn* en una sesión MPI propiamente dicha.
- *MPI_Finalize* es una llamada colectiva entre todos los procesos que están, en ese momento, conectados por un comunicador. Además, un proceso no se puede desconectar del comunicador *MPI_COMM_WORLD* salvo al realizar la llamada *MPI_Finalize*. Esto implica que todos los procesos que pertenezcan a un mismo grupo (ya sean creados al inicio o mediante *MPI_Comm_spawn*) tienen que terminar a la vez.
- Las llamadas *MPI_Comm_accept* y *MPI_Comm_connect* se pueden realizar tantas veces como se quiera entre dos grupos cualesquiera de procesos. Da igual si son de la misma aplicación MPI o no.

B. Integración con el planificador de trabajos (Slurm)

El estándar MPI se diseño para facilitar las comunicaciones y demás interacciones entre los procesos de una aplicación. Sin embargo, no contempla la gestión de dichos procesos, ni de los recursos necesarios para su creación/destrucción. La labor de gestionar estos recursos suele ser realizada por un planificador de trabajos, de los cuales, Slurm, es uno de los más conocidos y utilizados.

Slurm (Simple Linux Utility for Resource Management) es un sistema de gestión de recursos para entornos HPC. Este proporciona una plataforma flexible para administrar tareas, asignar recursos, y programar trabajos en un entorno distribuido. Además, facilita la administración eficiente de los recursos, la monitorización de trabajos, y la implementación de políticas de asignación personalizadas. Slurm utiliza colas de trabajo para administrar y priorizar las tareas enviadas por los usuarios. Las tareas se ejecutan en función de las políticas de asignación. Estas pueden tener en cuenta factores como la prioridad del usuario, la disponibilidad de recursos, y las restricciones de tiempo.

Slurm se fundamenta en dos operaciones básicas, que se complementan entre si. Estas operaciones son:

- Reserva de los recursos solicitados por la aplicación: En especial la reserva de nodos de computo, pero también pueden ser CPUs individuales, memoria, etc. A una reserva determinada de recursos se le denomina *job* o trabajo. Esta operación se puede realizar con el comando *salloc* de Slurm.
- Creación y gestión de los procesos de una aplicación: Requiere que haya una reserva de recursos previa, y los procesos creados utilizaran todos o parte de dichos recursos. A una ejecución determinada de los procesos de una aplicación se le denomina *job step* o etapa del trabajo. Un trabajo puede ejecutar múltiples etapas de forma paralela, repartiendo los recursos, o secuencial. Esta operación se puede realizar con el comando *srun* de Slurm.

A continuación, enumeraremos las características más relevantes de las operaciones de Slurm, para la creación y gestión de aplicaciones MPI maleables:

- Slurm permite que un trabajo en marcha pueda aumentar sus recursos dinámicamente, en especial el número de nodos. Para ello, es necesario solicitar la reserva de recursos con un nuevo trabajo y, cuando este sea concedido, integrar el nuevo trabajo como parte del antiguo, para así sumar los recursos de los dos.
- Slurm permite que un trabajo en marcha pueda reducir sus recursos, en especial el número de nodos. El único requisito es que dichos recursos no estén siendo utilizados, en este momento, por una de las etapas del trabajo. Esta operación se realiza con el comando *scontrol* de Slurm.
- Slurm permite que un trabajo ejecute tantas etapas de trabajo como se quiera, ya sea de forma paralela (repartiendo los recursos), o secuencial.
- Hasta donde llega nuestro conocimiento, Slurm no permite reducir los recursos asignados a una etapa de trabajo, al menos, hasta que dicha etapa es finalizada en su totalidad.

En lo relacionado con el entorno MPI, la forma en que dicho entorno se integra con los distintos gestores de trabajos, y en especial con Slurm, es muy variada. Cada implementación de MPI tiene su propia manera de integrarse con los gestores de trabajos. En general, existen tres estrategias generales para realizar esta integración.

- Nula integración: Es la opción más básica. El gestor de trabajos se encarga únicamente de la reserva de los recursos, y es el entorno de MPI el que se encarga, en exclusiva, de la creación y gestión de los procesos de la aplicación.
- Integración a medida: Los mecanismo de creación y gestión de procesos, tanto del entorno MPI concreto como del gestor de trabajos especifico, están integrados entre si. De esta forma, ambos pueden configurarse para delegar en el otro esta labor o hacerla ellos mismos. Esta integración se realiza ad-hoc para cada pareja de entorno MPI/gestor de trabajos.
- Integración en base a un estándar: En este caso el funcionamiento es similar al caso anterior con la salvedad que se utiliza un estándar para realizar la integración. El más utilizado es el estándar PMI (Process Management Interface), creado para proporcionar una interfaz de gestión y coordinación de procesos, en entornos de computación de alto rendimiento. Existen existen tres versiones (PMI-1, PMI-2[14], y

PMIx[15]); siendo la última, un desarrollo pensado para abordar los desafíos de los sistemas de gran escala (exascale).

El principal punto a considerar, es que tanto los gestores de trabajos (Slurm o otro), como los propios entornos MPI, suelen considerar los grupos de procesos (creados inicialmente o creados mediante*MPI_Comm_spawn*) como una unidad indivisible, que debe crearse y destruirse de forma única. Por ejemplo, cuando el entorno de MPI delega en Slurm la creación de los procesos, cada grupo de procesos es creado como una única etapa de trabajo, es decir con una llamada a *srun*.

Hay que recordar que el objetivo es el siguiente, que las aplicaciones MPI maleables puedan crear y eliminar procesos como proxy de la incorporación o liberación de recursos del sistema. Para conseguir este objetivo con el estándar MPI actual (MPI 4.0) es necesario que la granularidad de dicha maleabilidad sea limitada a grupos de procesos enteros (cada grupo con su propio comunicador *MPI_COMM_WORLD*). Por ejemplo, si queremos que la granularidad sea a nivel de nodo, hay que crear un grupo de procesos por nodo (ya sea como una aplicación MPI nueva o con una llamada a *MPI_Comm_spawn*) de forma que a la hora de eliminarlos también se pueda hacer nodo por nodo.

IV. Propuesta para la implementación de aplicaciones MPI maleables

Una aplicación MPI maleable debe permitir que, durante su ejecución, se puedan crear nuevos procesos aprovechando los recursos reservados para ello. Además, debe permitir también la destrucción de dichos procesos, con el fin de liberar dichos recursos. Esto implica, además, la posibilidad de crear un comunicador global que abarque todos los procesos de la aplicación, tanto los originales, como los creados dinámicamente. Por último, es necesario poder ajustar, o cambiar, dinámicamente dicho comunicador para incluir o eliminar procesos.

En este trabajo, se presenta una propuesta para la implementación de aplicaciones MPI maleables, que cumple con estos criterios. Esta propuesta esta pensada para ser ejecutada en un cluster HPC con reservas a nivel de nodo, lo cual es un caso muy habitual. En concreto, los requisitos que se tuvieron en cuenta, a la hora de realizar este diseño, son los siguientes:

- La aplicación debe reservar un número inicial de nodos al comenzar su ejecución.
- La aplicación podrá reservar nuevos nodos, para aumentar su número de procesos, de forma dinámica.
- La aplicación podrá liberar nodos, siempre que primero finalice los procesos de dicho nodo.
- La aplicación podrá elegir que nodos quiere eliminar.
  NOTA: esta condición tuvo que limitarse a la eliminación de nodos reservados dinámicamente, debido a las restricciones ya explicadas en III

Como se comentó en III-A, el estándar MPI (hasta la versión 4.0) tiene algunas limitaciones que dificultan la consecución de estas condiciones. La principal, es la necesidad de crear y/o destruir simultáneamente todos los procesos que conforman un grupo de procesos. En este contexto nos referimos a un grupo de procesos, como el conjunto de procesos que están (o pueden estar) comunicados por un comunicador *MPI_COMM_WORLD*. Estos grupos de procesos se crean en dos situaciones:

- Al ejecutar una nueva aplicación MPI, normalmente, mediante el comando *mpirun*. El grupo se conforma por todos los procesos creados por este comando.
- Al ejecutar una llamada colectiva a *MPI_Comm_spawn*. El grupo se conforma por todos los procesos creados directamente por esta llamada.

Basándonos en este concepto, el de grupo de procesos, vamos a desarrollar la estrategia para implementar aplicaciones maleables. Dicha estrategia se puede resumir con los siguiente pasos:

- Inicialmente, se crea la aplicación MPI con el número de nodos mínimo que se considere aceptable.
- En cada proceso de ampliación, se creará un grupo de procesos (vía *MPI_Comm_spawn*), por cada nuevo nodo que se quiera añadir.
- Cuando todos los procesos de un nodo sean finalizados, dicho nodo será liberado.

Cada vez que se modifica el número de procesos y/o nodos es necesario actualizar el comunicador global. Este solo debe incluir los procesos que permanecen después de la actualización. El comunicador global se consigue combinado los comunicadores básicos (*MPI_COMM_WORLD* y/o *MPI_COMM_SELF*) de cada uno de los grupos de procesos involucrados. La forma de combinarlos es obteniendo intercomunicadores de dos comunicadores cualquiera (básicos o compuestos) y convertirlos en intracomunicadores o comunicadores normales.

Existen dos formas de conseguir un intercomunicador de dos comunicadores. Por un lado, al crear un nuevo grupo de procesos con *MPI_Comm_spawn* se combina el comunicador de los procesos padre, con el *MPI_COMM_WORLD* de los procesos hijo. Este estrategia es perfecta cuando se quieren añadir nodos. Sin embargo, es muy poco flexible a la hora de eliminar un nodo. La razón es que solo se puede eliminar nodos desconectando dicho comunicador y usando uno previo. La idea seria volver al comunicador previo que tenga el mismo número de nodos que se desea mantener, y desconectar todos los comunicadores posteriores al elegido. Así, los nodos eliminados son siempre los últimos en ser añadidos.

La segunda forma de combinar dos comunicadores es utilizando las llamadas *MPI_Comm_accept* y *MPI_Comm_connect*. Estas permiten conectar dos comunicadores de dos grupos de procesos de la mis-

ma o de distinta aplicación. Esta técnica se puede hacer, de forma incremental, hasta conseguir un comunicador global que conecte todos los grupos de procesos. La gran ventaja de esta técnica es que se puede realizar tantas veces como se quiera. Por tanto los grupos de procesos se pueden reorganizar a voluntad, por ejemplo, cada vez que se añada o elimine un nodo. Además dicha reorganización se puede realizar, incluyendo o dejando fuera cualquier nodo que se desee. El único requisito es disponer de un medio para distribuir las direcciones de los puertos abiertos en todos los nodos. La mejor solución es el uso de un comunicador global ya existente, como el obtenido con la llamada *MPI_Comm_spawn*.

El último punto es comentar como desconectar un comunicador creado de forma incremental. Para que la desconexión sea completa no es suficiente con desconectar el último comunicador creado. Por contra, es necesario desconectar, además, todos los comunicadores previos hasta llegar a los comunicadores básicos (aquellos que no se pueden desconectar salvo con *MPI_Finalize*). En esta trabajo proponemos guardar el histórico de todos los comunicadores hasta que, llegado el momento, se desconecten todos al mismo tiempo.

El Algoritmo 1 presenta la estrategia para aumentar los nodos en una aplicación MPI maleable. Este se basa en el uso de la llamada *MPI_Comm_spawn* para aumentar el numero de procesos. Pero, en lugar de crear un único grupo con todos los procesos, se propone crear, en un bucle, tantos grupos de procesos como nodos se quiere añadir. El comunicador final se consigue fusionando de forma incremental el comunicador original con los comunicadores de cada nuevo grupo de procesos.

Por otro lado el Algoritmo 2 presenta la estrategia para reducir los nodos en una aplicación MPI maleable. La idea es finalizar los procesos de los nodos que se quieren liberar y, posteriormente, crear un nuevo comunicador con los grupos de procesos restantes. Para ello, este algoritmo construye el comunicador de forma incremental siguiendo la estructura de un árbol binario. Así, inicialmente, todos los grupos de procesos se emparejan. Cada par se combina para conseguir un comunicador entre ambos. Estos comunicadores se emparejan de nuevo para crear comunicadores compuestos más grandes. Este proceso se repite hasta que todos los procesos esta contenidos en un único comunicador global. Dado su estructura de árbol binario, la complejidad de este algoritmo es de orden O(log_n), siendo n el número de nodos o grupos de procesos.

Algorithm 1 Añadir nuevos nodos en una aplicación MPI maleable

Entrada: Comm_Global, Nuevos_Nodos
Salida : Comm_Global

```
// Si es proceso hijo, conectarlo
si es_proceso_hijo entonces
    Inter_Comm = MPI_Get_parent();
    Nuevos_Nodos = MPI_Receive(Inter_Comm);
    Comm_Global = MPI_Merge(Inter_Comm);
fin
// Repetir si hay nodos que añadir
mientras No_vacio(Nuevos_Nodos) hacer
    Nodo = coger_primero(Nuevos_Nodos);
    borrar_primero(Nuevos_Nodos);
    Inter_Comm=MPI_spawn(nodo,Comm_Global);
    MPI_Send(Nuevos_Nodos, Inter_Comm);
    Comm_Global = MPI_Merge(Inter_Comm);
fin
```

Algorithm 2 Eliminar nodos en una aplicación MPI maleable

Entrada: Comm_Global, Nodos_Borrados
Salida : Comm_Global

```
// Seleccionar si es un nodo a eliminar
Grupo = Coger_Grupo_Proc(Comm_Global);
Borrado = Esta_En(Grupo, Nodos_Borrados);
// Abrir un puerto en cada grupo no borrado
Rango = MPI_Rank(MPI_COMM_WORLD);
si Rango==0 Y NO(Borrado) entonces
    Puerto = MPI_open_port();
fin
// Compartir las direcciones de puertos
Vec_Puerto=MPI_Allgather(Puerto,Comm_Global);
// Borrado->desconectar el comm. y terminar
si Borrado entonces
    MPI_Disconnect(Comm_Global);
    MPI_Finalize();
    exit(0);
fin
// Coger los grupos que permanecen
Vec_Grupo_Val = coger_validos(Vec_Puerto);
Grupo_Val = posicion(Grupo, Vec_Grupo_Val);
Num_Grupo_Val = tamaño(Vec_Grupo_Val);
// Unir los MPI_COMM_WORLD -> Comm_Global
Comm_Local = MPI_COMM_WORLD;
mientras Num_Grupo_Val > 1 hacer
    Indice = Grupo_Val % Num_Grupos_Val;
    si Indice == Grupo entonces
        Inter = MPI_accept (Comm_local);
        Comm_local = MPI_Merge(Inter);
    fin
    en otro caso
        Port=coger_puerto(Vec_Puerto, Indice);
        Inter=MPI_connect (Port, Comm_local);
        Comm_local = MPI_Merge(Inter);
    fin
    Num_Grupo_Val = Num_Grupo_Val / 2;
fin
// Desconectar el comm. global
MPI_Disconnect(Comm_Global);
// Obtener el proximo comm. global
Comm_Global = Comm_local;
```

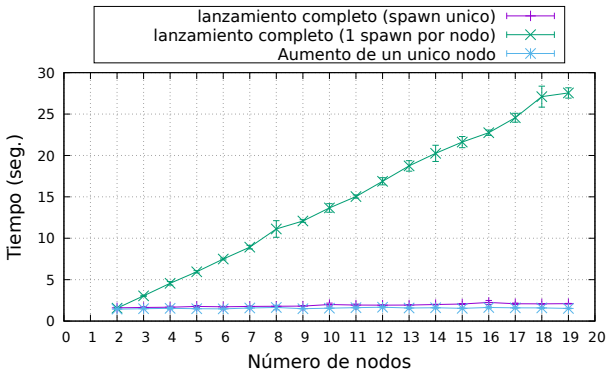


Fig. 1: Inclusión de nuevos nodos y generación del comunicador asociado (spawn).

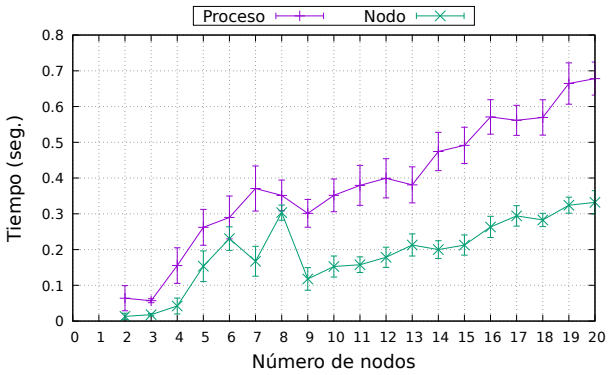


Fig. 2: Regeneración del comunicador global usando accept/-connect.

V. Resultados experimentales

En este apartado vamos a evaluar el rendimiento de la solución propuesta, para el desarrollo de aplicaciones MPI maleables. En primer lugar, evaluaremos cual es el rendimiento y, especialmente, la escalabilidad de la estrategia elegida para aumentar los nodos de la aplicación. En segundo lugar se evaluará el rendimiento de la estrategia elegida para eliminar nodos, reorganizando el comunicador global. Por ultimo, también se evaluará el coste de desconectarse de un comunicador creado con ambas estrategias.

La arquitectura del sistema empleado para la evaluación consiste en un cluster de 68 nodos, donde cada uno incluye 2 procesadores Intel(R) Xeon(R) CPU E5-2697 a 3.4 Ghz, cada uno con 18 cores. Cada nodo incluye, además, 128 GB de memoria RAM. La red de interconexión es una red Omni-Path a 100 Gb/s. A nivel software, el gestor de proceso es Slurm versión 22.05 y para el entorno MPI se ha utilizado MPICH version 4.0.3. Las pruebas se ha realizado empleando entre 1 y 20 nodos, lo que da un total de entre 36 y 720 procesos.

La Figura 1 muestra el tiempo empleado en incluir de 1 a 18 nuevos nodos a una aplicación que ejecuta en un único nodo. Por un lado, muestra cuanto tarda en lanzar todos los nodos en una única llamada *MPI_Comm_spawn*. Por otro, muestra el tiempo empleado en lanzar los nodos uno a uno, mediante sucesivas llamadas a *MPI_Comm_spawn*. Por último, también muestra cuanto se tarda en incluir un nuevo nodo, cuando la aplicación ya tiene entre 1 y 18 nodos.

Como se puede ver en la gráfica los tiempos en aumentar cualquier número de nodos, mediante una única llamada a *MPI_Comm_spawn*, son prácticamente iguales (alrededor de 1.5 segundos). Esto, además, es independiente del numero de nodos que ya estén incluidos en la aplicación. En cambio, cuando se añaden los nodos uno a uno, el tiempo se incrementa de manera proporcional. La conclusión es que cada llamada *MPI_Comm_spawn* tiene un coste similar, independientemente del número de nodos involucrados. Esto hace que la estrategia elegida, incrementar la aplicación nodo por nodo, pueda tener un coste importante a medida que el número de nodos aumenta. Esto a pesar de ser la única solución encontrada para poder eliminar los nodos de forma discrecional.

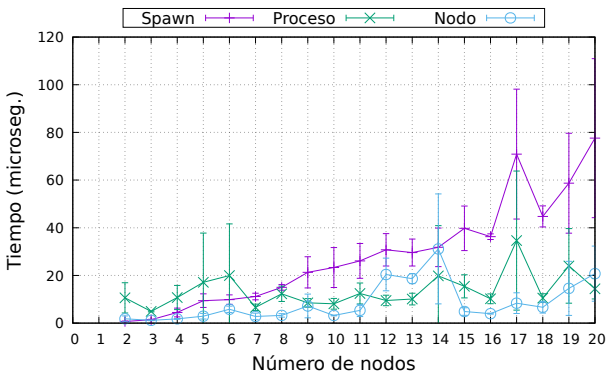


Fig. 3: Desconexión de un comunicador global en función del método de generación.

La Figura 2 muestra el tiempo necesario en componer un nuevo comunicador global, de la misma forma que se haría cuando se desee eliminar un nodo. La aplicación, que fue construida nodo a nodo, incluye desde 2 a 20 nodos. De esta forma cada nodo tiene su propio grupo de procesos con un comunicador *MPI_COMM_WORLD* de 36 procesos. Además cada proceso tiene su propio comunicador *MPI_COMM_SELF*. El comunicador global se construye usando *MPI_Comm_accept* y *MPI_Comm_connect* siguiendo el esquema de árbol binario. En la gráfica se presentan dos resultados: el primero combinado los comunicadores *MPI_COMM_WORLD* de cada nodo, el segundo combinado los comunicadores *MPI_COMM_SELF* de cada proceso.

Los resultados muestran que la estrategia seguida permite una alta escalabilidad. El tiempo empleado es claramente dependiente del número de operaciones de conexión realizadas (mayor en el esquema por procesos que en el de nodos). Aun así, el esquema de conexión en árbol binario reduce considerablemente el número de operaciones, a medida que aumenta el número de nodos o procesos, lo cual es esperable de un algoritmo de complejidad O(log_n). Además los tiempo se sitúan por debajo del segundo, incluso cuando se enlazan 720 procesos de forma individual.

Por último la Figura 3 muestra el tiempo necesario en desconectar un comunicador global creado con cualquiera de las técnicas anteriores, incluyendo todos los comunicadores intermedios empleados en generarlo. Por un lado se presenta el tiempo de desconexión de los comunicadores generados con la primera estrategia (*MPI_Comm_spawn*) y por otro el tiempo de desconexión de los comunicadores creados de forma binaria con la segunda estrategia (*MPI_Comm_accept* y *MPI_Comm_connect*), partiendo tanto de los nodos como de los procesos individuales.

El resultado implica unos tiempo de desconexión despreciables (por debajo del los 100 microsegundos). En cualquier caso, se observa que a mayor número de comunicadores auxiliares que desconectar, mayor es el tiempo requerido. Así, el comunicador incremental generado con *MPI_Comm_spawn* requiere un tiempo mayor que los generados de forma binaria con *MPI_Comm_accept* y *MPI_Comm_connect*.

VI. Conclusiones

En este trabajo se presenta una metodología para desarrollar aplicaciones MPI maleables que permitan añadir y eliminar recursos a voluntad. Para ello se utilizan las facilidades presentes en el estandar MPI 4.0 y su integración con gestores de trabajos como Slurm. Las propuestas se basan en granularizar la maleabilidad, en base a grupos de procesos que comparten su propio comunicador *MPI_COMM_WORLD*. Estos grupos se crean cuando se inicia una nueva aplicación MPI, o cuando se utiliza la llamada *MPI_Comm_spawn*. Se ha presentado dos estrategias para aumentar/reducir los recursos de una aplicación MPI maleable. Estas estrategias permiten aumentar los nodos asignados a una aplicación, al tiempo que permite eliminarlos posteriormente. La evaluación realizada muestra que la estrategia tiene un buen rendimiento y escalabilidad al reducir nodos pero no así al aumentarlos ya que el tiempo es proporcional al número de nodos añadidos. Como trabajo futuro se pretende investigar nuevas técnicas que permitan mejora la escalabilidad al incrementar los nodos, pero sin perder el grado de maleabilidad conseguido.

Agradecimientos

Este trabajo se ha desarrollado dentro del marco de la Unión Europea (European High-Performance Computing Joint Undertaking) mediante el projecto "Adaptive multi-tier intelligent data manager for Exascale" con identificación No 956748 - ADMIRE - H2020-JTI - EuroHPC-2019-1, y la agencia Española de Investigación con identificación PCI2021-121966.

Referencias

[1] Jimmy Aguilar Mena, "Methodology for malleable applications on distributed memory systems," 2022.

[2] David E. Bernholdt, Swen Boehm, George Bosilca, Manjunath Gorentla Venkata, Ryan E. Grant, Thomas Naughton, Howard P. Pritchard, Martin Schulz, and Geoffroy R. Vallee, "A survey of mpi usage in the us exascale computing project," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 3, pp. e4851, 2020, e4851 cpe.4851.

[3] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 4.0*, June 2021.

[4] Kaoutar El Maghraoui, Boleslaw K. Szymanski, and Carlos Varela, "An architecture for reconfigurable iterative mpi applications in dynamic environments," in *Parallel Processing and Applied Mathematics*, Roman Wyrzykowski, Jack Dongarra, Norbert Meyer, and Jerzy Waśniewski, Eds., Berlin, Heidelberg, 2006, pp. 258–271, Springer Berlin Heidelberg.

[5] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.

[6] Sathish S. Vadhiyar and Jack J. Dongarra, "Srs: A framework for developing malleable and migratable parallel applications for distributed systems," *Parallel Processing Letters*, vol. 13, no. 02, pp. 291–312, 2003.

[7] Chao Huang, Orion Lawlor, and L. V. Kalé, "Adaptive mpi," in *Languages and Compilers for Parallel Computing*, Lawrence Rauchwerger, Ed., Berlin, Heidelberg, 2004, pp. 306–322, Springer Berlin Heidelberg.

[8] Suraj Prabhakaran, Marcel Neumann, Sebastian Rinke, Felix Wolf, Abhishek Gupta, and Laxmikant V. Kale, "A batch system with efficient adaptive scheduling for malleable and evolving applications," in *2015 IEEE International Parallel and Distributed Processing Symposium*, 2015, pp. 429–438.

[9] Isaías Comprés, Ao Mo-Hellenbrand, Michael Gerndt, and Hans-Joachim Bungartz, "Infrastructure and api extensions for elastic execution of mpi applications," in *Proceedings of the 23rd European MPI Users' Group Meeting*, New York, NY, USA, 2016, EuroMPI 2016, p. 82–97, Association for Computing Machinery.

[10] Gonzalo Martín, Maria-Cristina Marinescu, David E. Singh, and Jesús Carretero, "Flex-mpi: An mpi extension for supporting dynamic load balancing on heterogeneous non-dedicated systems," in *Euro-Par 2013 Parallel Processing*, Felix Wolf, Bernd Mohr, and Dieter an Mey, Eds., Berlin, Heidelberg, 2013, pp. 138–149, Springer Berlin Heidelberg.

[11] David E. Singh and Jesus Carretero, "Combining malleability and i/o control mechanisms to enhance the execution of multiple applications," *Journal of Systems and Software*, vol. 148, pp. 21–36, 2019.

[12] Pierre Lemarinier, Khalid Hasanov, Srikumar Venugopal, and Kostas Katrinis, "Architecting malleable mpi applications for priority-driven adaptive scheduling," in *Proceedings of the 23rd European MPI Users' Group Meeting*, New York, NY, USA, 2016, EuroMPI 2016, p. 74–81, Association for Computing Machinery.

[13] Iker Martín-Álvarez, José I Aliaga, Maribel Castillo, Sergio Iserte, and Rafael Mayo, "Dynamic spawning of mpi processes applied to malleability," *The International Journal of High Performance Computing Applications, First published online (2023)*.

[14] Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Jayesh Krishna, Ewing Lusk, and Rajeev Thakur, "Pmi: A scalable parallel process-management interface for extreme-scale systems," in *Recent Advances in the Message Passing Interface*, Rainer Keller, Edgar Gabriel, Michael Resch, and Jack Dongarra, Eds., Berlin, Heidelberg, 2010, pp. 31–41, Springer Berlin Heidelberg.

[15] PMIx Administrative Steering Committee, "Process management interface for exascale (pmix) standard, version 5.0," `https://pmix.org/uploads/2023/05/pmix-standard-v5.0.pdf`, 2023, [Online; accessed 07-Jun-2023].

Acelerando la extracción de los parámetros del modelo de diodo único para paneles solares

X. Moreno-Vassart[1], F. Javier Toledo[2], Victoria Herranz[2], Jose M. Blanes[3], Vicente Galiano[1]

Resumen— **En este artículo se describen las mejoras que aportan una nueva implementación del método Two-step Linear Least-Squares (TSLLS) de extracción de parámetros del modelo de único diodo utilizado como modelo de comportamiento de un panel fotovoltaico. El modelo TSLLS inicialmente está desarrollado en su versión para matlab y se puede hacer uso de ella mediante una interfaz web (https://www.pvmodel.umh.es). El desarrollo de una versión en un lenguaje como Python es crucial tanto para el escalado de herramientas web a tiempo real así como para la gestión y monitorización de paneles solares mediante sistemas empotrados. La nueva librería desarrollada permite obtener la mejor aproximación posible de los cinco parámetros y la obtención de los puntos característicos de la curva V-I en un tiempo más corto sin hacer uso de métodos menos exactos. La precisa y rápida extracción de estos puntos permite un mejor seguimiento del punto de máxima potencia (MPP) y posibilita un mayor rendimiento de una instalación solar. Esta versión en Python mejora los tiempos de ejecución de la versión anterior en MATLAB® casi 2.5 veces y aumenta su posibilidad de utilización en sistemas de código abierto sin necesidad de licencia.**

Palabras clave— **Modelo Único Diodo,Energía Fotovoltaica, Procesamiento de Datos, Algoritmos Paralelos, Seguimiento del Punto de Máxima Potencia, Optimización**

I. Introducción

El uso de combustibles fósiles como fuente de energía para transporte, industria y generación de energía eléctrica tiene un impacto negativo sobre el medio ambiente y la economía mundial. En los últimos años, por ejemplo, la dependencia del petroleo se ha relacionada con el aumento del coste de la electricidad. Las energías renovables deben ser por tanto la solución energética tanto de la industria como de la sociedad en general. La energía solar es una de las fuentes renovables las más accesibles y democratizadas.

En sistemas fotovoltaicos, la modelización de los paneles solares se ha convertido en una herramienta clave para la predicción y optimización de su rendimiento, situando el panel en el Punto de Máxima Potencia (MPP), incrementando la energía generada. Existen algoritmos capaces de encontrar los parámetros SDM con gran precisión y rapidez para condiciones ambientales fijas. Por ejemplo, [1], [2], [3], [4] han obtenido en los últimos años los mejores resultados de la literatura en la extracción de parámetros SDM por minimización en función de la distancia corriente.

El modelo de único diodo (*Single Diode Model*, SDM), también conocido como modelo de cinco parámetros, ha sido el más utilizado en las últimas décadas y, dado un mínimo de iluminación, es capaz de generar una curva teórica que reproduce, con mucha precisión, la curva V-I de la mayoría de paneles solares. Es importante recalcar que esas curvas, y por tanto los parámetros del modelo, dependen también de otros factores como la temperatura y la irradiancia [5] o las condiciones del panel, como por ejemplo sombras, suciedad, degradación [6], etc. De hecho, también hay trabajos que proporcionan fórmulas sobre cómo varían los parámetros en función de la temperatura y/o la irradiancia [7], [8], [9], aunque las fórmulas existentes todavía no pueden contemplar todas las variables posibles ni todas las tecnologías.

La sección II cubre el modelo de único diodo y las ecuaciones que lo rigen. Este análisis prosigue (sec. III) con una explicación del método de extracción de parámetros SDM mediante el método *Two-step Linear Least Squares* (TSLLS), el algoritmo cuya optimización es el sujeto de estudio de este artículo. El artículo continúa (sec. IV) con un repaso a la herramienta web que hace uso de la librería de extracción de parámetros y los beneficios que aportan la mejora. Más adelante se exponen los cambios realizados en esta implementación del TSLLS (sec. V) y las medidas de tiempos de ejecución de las distintas versiones (sec. VI).

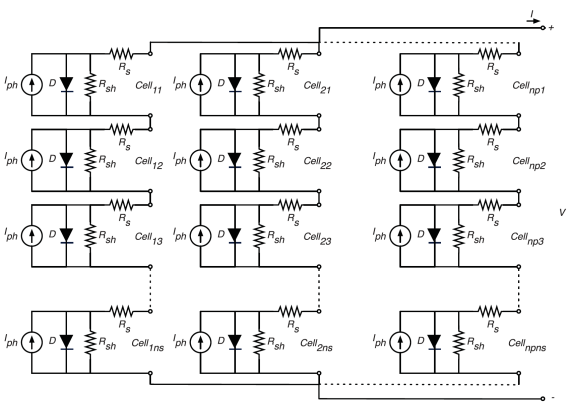


Fig. 1: Modelo de únido diodo (SDM).

[1]Dpto. de Ingeniería de Computadores, Universidad Miguel Hernández de Elche, e-mail: {xmoreno-vassart,vgaliano}@umh.es

[2]Centro de Investigación Operativa, Universidad Miguel Hernández de Elche, e-mail: {javier.toledo,mavi.herranz}@umh.es

[3]Grupo de Electrónica Industrial, Universidad Miguel Hernández de Elche, e-mail: jmblanes@umh.es

II. Modelo de Único Diodo

El modelo de único diodo describe el funcionamiento eléctrico de un panel fotovoltaico con el circuito de la Figura 1 y sus características vienen descritas por 5 parámetros:

- $I_{ph} = n_p I_{ph}^{celda}$: Fotocorriente
- $I_{sat} = n_p I_{sat}^{celda}$: Corriente de saturación inversa del diodo
- $R_s = \frac{n_s}{n_s} R_s^{celda}$: Resistencia en serie
- $R_{sh} = \frac{n_s}{n_p} R_{sh}^{celda}$: Resistencia de derivación
- n: Factor de idealidad del diodo

Partiendo del circuito anterior – una generalización para múltiples celdas en serie y paralelo – y resolviendo el circuito aplicando la ley de Kirchoff de corrientes obtenemos $I = n_p(I_{ph} - I_D - I_{sh})$ donde $I_D = n_p\, I_{sat} \left(e^{\frac{\frac{V}{n_s} + \frac{IR_s}{n_p}}{nV_T}} - 1 \right)$, $I_{sh} = n_p \frac{\frac{V}{n_s} + \frac{IR_s}{n_p}}{R_{sh}}$ y $V_T = \frac{kT_c}{q}$. T es la temperatura de la celda en Kelvin, k la constante de Boltzmann ($1{,}3806503E-23\frac{J}{K}$) y q la carga del electrón ($1{,}602E19C$). Podemos expresar por tanto la curva V-I con la siguiente ecuación

$$I = n_p \left[I_{ph} - I_{sat} \left(e^{\frac{\frac{V}{n_s} + \frac{IR_s}{n_p}}{nV_T}} - 1 \right) - \frac{\frac{V}{n_s} + \frac{IR_s}{n_p}}{R_{sh}} \right] \tag{1}$$

donde I es la corriente del panel medida en Amperios y V la diferencia de potencial en Voltios. La resolución de esta ecuación no lineal de forma precisa y eficiente es un campo de estudio de mucha actividad, con propuestas que hacen uso de distintos algoritmos [10], [3], medidas de error [11] y como en el caso de este artículo, optimizaciones en la implementación de algoritmos preexistentes. Este artículo se centra en el método TSLLS (sec. III) porque ha obtenido los parámetros con el menor error RMSE (*Root-Mean-Square Error*) hasta la fecha en dos casos de estudio de referencia [12].

III. Two-step Linear Least Squares

El método objeto de estudio en este artículo es el *Two-step Linear Least Squares* (TSLLS), que obtiene los parámetros del modelo de único diodo partiendo de las cordenadas de un mínimo de 5 puntos de la curva V-I y no depende de la estimación de los puntos característicos I_{sc} (corriente de cortocircuito), V_{oc} (voltaje de circuito abierto), y punto de máxima potencia (V_{mpp}, I_{mpp}) , ni de un aproximación inicial de la solución. A partir de la ecuación 1 , mediante un cambio de variable podemos reescribirla del siguiente modo:

$$I = A - B(C^V D^I - 1)EV \tag{2}$$

Donde definimos 5 parámetros geométricos de la siguiente manera

$$\begin{aligned} A &= n_p I_{ph} \frac{R_{sh}}{R_{sh} + R_s} \\ B &= n_p I_{sat} \frac{R_{sh}}{R_{sh} + R_s} \\ C &= e^{\frac{1}{n_s n V_T}} \\ D &= e^{\frac{R_s}{n_p n V_T}} \\ E &= \frac{n_p}{n_s} \frac{1}{R_{sh} + R_s} \end{aligned} \tag{3}$$

Si somos capaces de resolver los parámetros geométricos 3, seremos capaces de recuperar los parámetros originales de la ecuación 2:

$$\begin{aligned} I_{ph} &= A \frac{1}{n_p} \frac{ln(C)}{ln(C) - Eln(D)} \\ I_{sat} &= B \frac{1}{n_p} \frac{ln(C)}{ln(C) - Eln(D)} \\ n &= \frac{1}{ln(C) n_s V_T} \\ R_{sh} &= \frac{n_p}{n_s} \left(\frac{1}{E} - \frac{ln(D)}{ln(C)} \right) \\ R_s &= \frac{n_p}{n_s} \frac{ln(D)}{ln(C)} \end{aligned} \tag{4}$$

El método TSLLS resuelve por separado pero no de forma independiente la parte lineal $A + B - EV$ que corresponde a la asíntota oblicua y la parte exponencial $BC^V D^I$. El primer paso es resolver la ecuación lineal

$$I \approx A + B - EV \tag{5}$$

Tomando L puntos desde el inicio (al menos 2), se ajusta con mínimos cuadrados la ecuación 5 con $K = A + B$, obteniendo así la solución (K, E). Existe un método conocido de resolución mediante la pseudoinversa de Moore-Penrose de Λ_L, es decir $\Lambda_L^T(\Lambda_L^T\Lambda_L)^{-1}$. En este caso Λ_L es una matriz de segundo orden de la forma

$$\Lambda_L = \begin{bmatrix} 1 & -V_1 \\ \vdots & \vdots \\ 1 & -V_L \end{bmatrix} \tag{6}$$

Además esta solución es única cuando el rango de Λ_L es 2, cierto siempre que $V_j \neq V_k$, siempre que no tomemos dos valores repetidos en las medidas. La solución (K, E) se puede por tanto expresar como

$$\begin{bmatrix} K \\ E \end{bmatrix} = (\Lambda_L^T \Lambda_L)^{-1} \Lambda_L^T \begin{bmatrix} I_1 \\ \vdots \\ I_L \end{bmatrix} \tag{7}$$

El segundo paso es resolver la parte exponencial, que podemos reescribir en función de K y E de la siguiente manera

$$BC^V D^I = K - EV - I \tag{8}$$

y tomando logaritmos

$$ln(B) + Vln(C) + Iln(D) = ln(K - EV - I) \quad (9)$$

Tomando los M últimos puntos (al menos 3), obtenemos los parametros B, C y D que mejor se ajustan a la ecuación 9. Del mismo modo que en el paso anterior, se resuelve con la pseudo-inversa de Γ_M

$$\Gamma_M = \begin{bmatrix} 1 & V_{N-M+1} & I_{N-M+1} \\ \vdots & \vdots & \vdots \\ 1 & V_N & I_M \end{bmatrix} \quad (10)$$

La solución se puede expresar por tanto como

$$\begin{bmatrix} ln(B) \\ ln(C) \\ ln(D) \end{bmatrix} = (\Omega_M^T \Omega_M)^{-1} \Omega_M^T \begin{bmatrix} ln(K - I_{N-M+1} - EV_{N-M+1}) \\ \vdots \\ ln(K - I_N - EV_N) \end{bmatrix} \quad (11)$$

$\Gamma_M^T \Gamma_M$ es una matriz cuadrada de orden 3 y por la geometría de las curvas V-I siempre tiene inversa y es única.

El método TSLLS aplica los pasos anteriores para los conjuntos (L, M) con

$$\begin{aligned} L &\in [2, N-3] \\ M &\in [3, L+1] \end{aligned} \quad (12)$$

siendo N el número de muestras de la medida de la curva V-I. Buscando el quinteto de parámetros que minimiza la raíz del error cuadrático medio (RMSE) de la curva de intensidades obtenida con dichos parámetros (I^P y la medida I) para todos los valores j del conjunto (L, M).

$$RMSE = \sqrt{\frac{1}{N} \sum_{j=1}^{N} \left(I_j^P - I_j\right)^2} \quad (13)$$

Esta solución es buena de por sí, pero es posible mejorarla con un ajuste de curvas utilizando la solución anterior como semilla. Se hace uso de los algoritmos *Levenberg-Marquardt* y *Trust region reflective* para obtener el mejor resultado.

Por último obtenemos los puntos característicos (V_{oc}, I_{sc}, V_{mpp}, I_{mpp}) con los parámetros del modelo único diodo obtenidos.

IV. Servicio web de extracción de parámetros SDM y otras aplicaciones

Implementar el algoritmo descrito en la sección III no es tarea fácil, y una mala implementación obtendría parámetros erróneos e inesperados. Por este motivo, los autores han desarrollado una plataforma web que obtiene los parámetros y otras funcionalidades como la solución con un modelo más sencillo (modelo de Forma Reducida) y la obtención de las combinaciones de cinco parámetros del SDM partiendo de los datos del fabricante. La Figura 2 muestra la página principal de dicha web.

El servicio web esta desarrollado en Python con un apariencia sencilla y limpia que permite a los usuario una rápida comprensión de sus funcionalidades.

Fig. 2: Página principal de https://pvmodel.umh.es

Los cálculos se realizan en el servidor a petición de los usuarios que acceden mediante un navegador. El servicio web recoge los datos del usuario, los transforma y almacena en un formato apropiado y, una vez guardados, lanza el interprete de MATLAB®(ver Figura 3). Cuando los cálculos han terminado, se recogen desde Python y se almacenan los resultados en una base de datos, listos para representarlos en HTML y enviarlos al cliente. MATLAB®también genera gráficas que son mostradas al usuario (ver Figura 5).

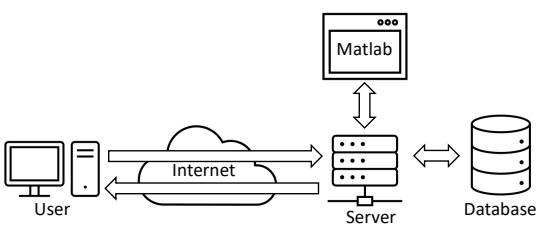


Fig. 3: Arquitectura de servicio web

El formulario de entrada se puede ver en la Figura 4. El usuario puede introducir medidas manualmente, incluyendo la temperatura, el número de celdas en serie (n_s) y el número de celdas en paralelo (n_p) o mediante un archivo separado tabuladores que represente las tensiones e intensidades del panel solar. El gráfico de las medidas es generado automáticamente.

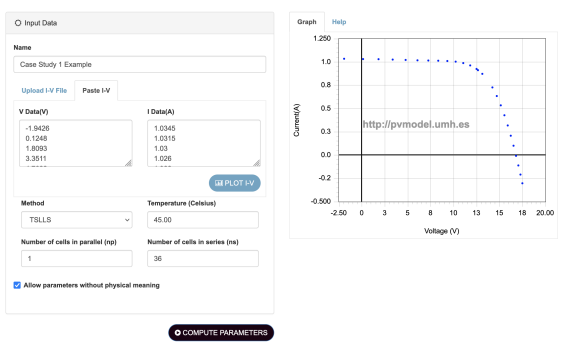


Fig. 4: Recogida de datos de entrada

La Figura 5 muestra los cinco parametros SDM obtenidos (I_{ph}, I_{sat}, n, R_{sh}, R_s), la raíz del error cuadrático medio (RMSE) y el error absoluto medio (MAE) entre la medida de corriente del modulo fotovoltaico y la corriente estimada con el modelo de único diodo. Los puntos característicos de la curva V-I también son obtenidos a partir del modelo y son presentados al usuario. Estos son la tensión de circuito abierto (V_{oc}), la corriente de cortocircuito (I_{sc}), y el punto de máxima potencia (V_{mpp},I_{mpp}). El usuario también tiene disponible las soluciones antes y después del refinamiento por ajuste de curvas explicado en la sección III.

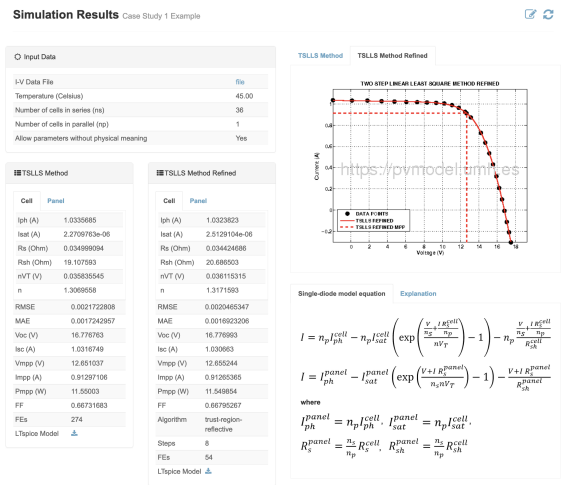


Fig. 5: Visualización de resultados

La nueva implementación en Python elimina la dependencia del interprete de MATLAB®, y también permite un mejor escalado de la aplicación web. Es más eficiente compartir los resultados de la extracción de parámetros SDM cuando la aplicación y la implementación de los cálculos utilizan la misma tecnología, facilitando su paso por memoria compartida y evitando almacenar representaciones intermedias. Además, la nueva versión permite la ejecución en paralelo de la extracción de múltiples curvas de forma nativa, en un mismo interprete de Python.

Estos cambios no solo tienen repercusiones en el servicio web, son esenciales en aplicaciones *in situ*, en instalaciones fotovoltaicas. Obtener los parámetro SDM es una tarea costosa y por tanto cualquier mejora es beneficiosa en sistemas empotrados, reduciendo su consumo y los requerimientos hardware necesarios para un seguimiento del MPP basado en el modelo de único diodo a tiempo real. Una granja solar de gran escala puede permitirse el uso de hardware propio de un centro de datos, con alto número de núcleos. Realizar un buen uso de estos sistemas de alta prestaciones requiere del paralelismo. La versión propuesta hace uso de la técnica *Single Instruction Multiple Data* (SIMD) para garantizar un uso eficiente de los recursos disponibles. Además elimina los costes de licencia de un software privativo como es MATLAB®, coste particularmente significativo en instalaciones distribuidas con múltiples controladores independientes.

V. Metodología

La implementación en Python incluye múltiples mejoras sobre el código original. En primer lugar evita calcular los parámetro geométricos C y D siempre que sea posible, prefiriendo almacenar y utilizar $ln(C)$ y $ln(D)$ respectivamente. Esto no solo implica un ahorro computacional, al limitar las operaciones y transformaciones de datos innecesarias, también mejora la precisión de los resultados, reduciendo los errores por operaciones sucesivas y almacenando resultados más pequeños en coma flotante, más cercanos a 1 donde la distancia entre números en coma flotante consecutivos es menor (*unit in the last place* - ULP).

Se trata de una implementación basada en cálculos vectoriales que hacen uso de librerías de altas prestaciones implementadas en C y Fortran: *Basic Linear Algebra Subprogram* (BLAS)[13] y *Linear Algebra Package* (LAPACK)[14]. Estas librerías expuestas en Python por NumPy y Scipy, entre otras, reducen la sobrecarga computacional que impone el uso de un lenguaje interpretado. Esto también es parcialmente cierto en MATLAB® pero el rediseño ha permitido hacer un uso más intensivo de la programación vectorial.

Además, han sido eliminados los comportamientos de máquina de estados presentes en las versiones anteriores, permitiendo una ejecución sin contexto. Esto facilita la paralelización de las operaciones y permite la extracción de parámetros de múltiples curvas V-I simultanea de forma sencilla y eficiente.

VI. Resultados

Los tiempos de ejecución son fuertemente dependientes del número de muestras de las medidas de la curva de Voltaje-Intensidad, por ese motivo las medidas realizadas han utilizado una misma muestra de una base de datos de más de un millón de curvas. Se han elegido de forma aleatoria 10 curvas V-I distintas que contienen 180 y 184 puntos de una colección de medidas ofrecidas por el Instituto Nacional de Energías Renovables del Estados Unidos (*National Renewable Energy Laboratory*, NREL). Las medidas se han realizado en un equipo con un procesador Intel® Core™ i7-1255U, 32Gb de RAM a 3200MHz y un disco duro de estado sólido de 512GB.

Tabla I: Tiempos de ejecución de la extracción de parámetros con TSLLS de 10 curvas V-I.

Implementación	Tiempo medio (s)
original (MATLAB®)	$4{,}9935 \pm 0{,}71251$
original (Python)	$26{,}240 \pm 1{,}52405$
nueva (Python)	$2{,}0356 \pm 0{,}32243$

Las medidas de la Tabla I representan el valor medio obtenido en 5 ejecuciones de la muestra seleccionada sin tener cuenta el tiempo de inicio de los interpretes para tres implementaciones: implementación original en matlab, implementación en Python resultante de una traducción literal del codigo matlab, y la optimización del código Python mediante las optimizaciones descritas en la Sección V. La nueva versión

es capaz de extraer parámetros SDM 2.45 veces más rápido que la original en MATLAB® y más de 12 veces que la versión anterior en Python, un *speedup* de 12.89.

VII. Conclusiones

La cantidad de muestras es una limitación intrínseca y presente en todos los algoritmos de extracción actualmente en uso. En el caso del TSLLS la dependencia es cuadrática (eq. 12), por tanto con una complejidad temporal $O(n^2)$. Los trabajos de optimización se centran por tanto en cada una de esas iteraciones.

La nueva implementación consigue mejorar los tiempos de ejecución de las versiones originales en manteniendo la rapidez de prototipado que ofrecen los lenguajes interpretados, y sin la complejidad añadida que pueden tener lenguajes de más bajo nivel a la hora de implementar esta clase de algoritmos. El uso de librerías vectoriales de altas prestaciones reduce el significativamente el coste adicional del interprete y de la gestión de memoria con recolector de basura. La versión actual tiene una ventaja adicional respecto a las dos anteriores, limita el acceso a memoria secundaria, beneficio que es particularmente útil para servicios web multiusuario. Por otro lado, Python permite la ejecución del método TSLLS en multitud de plataformas en servidores, clusters HPC y sistemas empotrados sin requerir licencia ni costes adicionales. El coste de esta abstracción adicional no es despreciable, especialmente cuando las curvas V-I contienen pocos puntos. Sin embargo, el coste del control de flujo en Python es superior y el primer elemento a optimizar en trabajos futuros. Además, esta versión puede servir de referencia para una implementación sucesiva en un lenguaje compilado con un mínimo de abstracción y sin recolector de basura (C++ o Rust) ya sea para su ejecución nativa o para obtener una representación intermedia como *WebAssembly* (Wasm) independiente de la arquitectura, que pueda ejecutarse tanto en dispositivos empotrados como servidores de altas prestaciones (por ejemplo mediante WASIX) o en el cliente, mediante el navegador web.

Agradecimientos

Dr. F. Javier Toledo, Dr. V. Galiano, Dr. Victoria Herranz and Dr. José M. Blanes have received funding from grant TED2021-130025B-I00 funded by MCIN/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR. Dr. F. Javier Toledo's work has received funding from the Ministerio de Ciencia e Innovación of Spain (PGC2018-097960-B-C21), the government of the Valencian Community (PROMETEO/2021/063) and the European Union (ERDF, "A way to make Europe"). Dr. V. Galiano's work has received funding from the Valencian Ministry of Innovation, Universities, Science and Digital Society (Generalitat Valenciana) under Grant CIAICO/2021/278 and from Grant PID2021-123627OB-C55 funded by MCIN/AEI/ 10.13039/501100011033 and, by "ERDF A way of making Europe".

Referencias

[1] Antonino Laudani, Francesco Riganti-Fulginei, and Alessandro Salvini, "High performing extraction procedure for the one-diode model of a photovoltaic panel from experimental i–v curves by using reduced forms," *Solar Energy*, vol. 103, pp. 316–326, 05 2014.

[2] Alejandro Angulo Cárdenas, Miguel Carrasco, Fernando Mancilla-David, Alexandre Street, and Roberto Cárdenas, "Experimental parameter extraction in the single-diode photovoltaic model via a reduced-space search," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 2, pp. 1468–1476, 2017.

[3] F. J. Toledo, J. M. Blanes, and V. Galiano, "Two-step linear least-squares method for photovoltaic single-diode model parameters extraction," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 8, pp. 6301–6308, 2018.

[4] Jie Xu, "Separable nonlinear least squares search of parameter values in photovoltaic models," *IEEE Journal of Photovoltaics*, vol. 12, no. 1, pp. 372–380, 2022.

[5] S. Yadir, R. Bendaoud, A. EL-Abidi, H. Amiry, M. Benhmida, S. Bounouar, B. Zohal, H. Bousseta, A. Zrhaiba, and A. Elhassnaoui, "Evolution of the physical parameters of photovoltaic generators as a function of temperature and irradiance: New method of prediction based on the manufacturer's datasheet," *Energy Conversion and Management*, vol. 203, pp. 112141, 2020.

[6] Sofiane Kichou, Elif Abaslioglu, Santiago Silvestre, Gustavo Nofuentes, Miguel Torres-Ramírez, and Aissa Chouder, "Study of degradation and evaluation of model parameters of micromorph silicon photovoltaic modules under outdoor long term exposure in jaén, spain," *Energy Conversion and Management*, vol. 120, pp. 109–119, 2016.

[7] Efstratios I. Batzelis and Stavros A. Papathanassiou, "A method for the analytical extraction of the single-diode pv model parameters," *IEEE Transactions on Sustainable Energy*, vol. 7, no. 2, pp. 504–512, 2016.

[8] Amit Jain and Avinashi Kapoor, "Exact analytical solutions of the parameters of real solar cells using lambert w-function," *Solar Energy Materials and Solar Cells*, vol. 81, no. 2, pp. 269–277, 2004.

[9] W. De Soto, S. A. Klein, and W. A. Beckman, "Improvement and validation of a model for photovoltaic array performance," *Solar Energy*, vol. 80, no. 1, pp. 78–88, 2006.

[10] Antonino Laudani, Fernando Mancilla-David, Francesco Riganti-Fulginei, and Alessandro Salvini, "Reduced-form of the photovoltaic five-parameter model for efficient computation of parameters," *Solar Energy*, vol. 97, pp. 122–127, 2013.

[11] F. Javier Toledo, Vicente Galiano, Jose M. Blanes, Victoria Herranz, and Efstratios Batzelis, "Photovoltaic single-diode model parametrization. an application to the calculus of the euclidean distance to an i–v curve," *Mathematics and Computers in Simulation*, 2023.

[12] T. EASWARAKHANTHAN, J. BOTTIN, I. BOUHOUCH, and C. BOUTRIT, "Nonlinear minimization algorithm for determining the solar cell parameters with microcomputers," *International Journal of Solar Energy*, vol. 4, no. 1, pp. 1–12, 1986.

[13] L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al., "An updated set of basic linear algebra subprograms (blas)," *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 135–151, 2002.

[14] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

Parte 2.- VII Jornadas de Computación Empotrada y Reconfigurable

Aplicaciones de los sistemas empotrados

Servitization of a Neuromorphic Robotic Arm: A Microservices Approach

Adalberto Farias[1], Enrique Piñero-Fuentes[2],Antonio Rios-Navarro[2], Alejandro Linares-Barranco[2], Sergio Segura[2],

Resumen— **Neuromorphic engineering includes a new paradigm of neuro-inspired computation that goes beyond classical Von Neumman or Harvard-type architectures. In-memory computing, information encoding through spikes (or events), and the new approach to artificial intelligence that neuromorphic computing offers are making an impact on the industrial fabric. Low latency and low power consumption are the main features of this research field. Robotic devices are usually controlled using either proprietary software or ad-hoc low-level programs, which hinders their shared use and integration with other applications. In this work, a remote service is presented for allowing access control and arbitration in the use of robots as a shared service. This service allows the design and execution of trajectories in robot and Cartesian coordinate spaces in a simple and convenient way for the user. The ED-Scorbot robotic arm is presented as a use case for this service. Concepts such as closed-loop neuromorphic motor control or direct robot kinematics are abstracted for the user, allowing the creation of unsupervised learning iterations of closed-loop trajectories or the creation of datasets for supervised training. Moreover, the servitization is based on a heterogeneous ecosystem composed of microservices that use different platforms/languages to provide a novel Web experience to users, while still being flexible to allow easy addition of new robotic arms.**

Palabras clave— **Servitization, Neuromorphic motor control, FPGA, Robotic arm, API specification**

I. Introduction

THE use of robots is more common nowadays than ever. In most activities they are used, we have benefits such as efficiency, precision, and cost reduction. The reduction in energy and latency can still be improved. One factor that greatly impacts this success is how easily end users can put the robot to work properly in their tasks. Neuromorphic controllers take advantage of low power and low latency in the robotic field [1] [2]. Nevertheless, current neuromorphic robots are managed in research labs in a local way [3]. They usually require specialized hardware and software controllers [4] only available at research labs. Remote interaction is not always possible and this restricts other researchers to explore the benefits of these robots and controllers. In this context, servitization has been viewed as an inflection point, affecting the business model and bringing significant productivity gains[5], as it provides a general, intuitive, independent, portable, and (usually) scalable way of offering functionalities.

[1]Systems and Computing Department, Federal University of Campina Grande, e-mail: adalberto@computacao.ufcg.edu.br

[2]Smart Computer Systems Research and Engineering Lab (SCORE), Research Institute of Computer Engineering (I3US), University of Seville, Spain, e-mail: epinerof@us.es.

Making the use of a robotic arm easy and wide is a challenge that involves several efforts. For example, the robot's platform should be easy and intuitive for developers/engineers, that is, it must be easy to programmatically put the robot to work. The functionalities and behaviour must be clearly defined and must be minimally stable. Fault tolerance or failure recovery features should be present. And finally, it should be easy to use the robot in wider contexts (an ecosystem or integrated with other applications). However, Commercial Off-The-Shelf (COTS) robots provide their own solutions to command their own hardware, and these can be more or less standardized. For example, the Summit XL Robot manufacturer offers a ROS-compatible controller [1], while another manufacturer like RobotAnno's SJ602 [2] offers both a standalone desktop application to send commands to the robot and a Robotic Operating System (ROS) compatible controller (bought separately). This allows for the exploration of this kind of servitization work in the world of robotics.

This work focuses on the integration effort by providing a solution that represents a service context in which a robotic arm can be easily integrated and used, both locally and remotely. The adopted neuromorphic robotic arm – ED-Scorbot – is already used as a service [6]. However, it requires extra knowledge for remotely controlling the arm, such as establishing a direct connection via SSH with the authorised machine, sending commands to the robot by using low-level tools (scripts and programs), applying specific data conversions, and sometimes handling low-level parameters of the arm motor controllers.

The demand to evolve this scenario to a new service model has inspired our solution.

The servitization ecosystem proposed in this work makes the ED-Scorbot available as a Web application to users that need to know only basic details about the arm (business information). Servitization is the transformational process whereby a company shifts from a product-centric to a service-centric business model and logic[7]. Concerning software, this changes the paradigm from "buying, installing, using and maintaining the application working in your own infrastructure"to "paying for using an application installed and maintained in another infrastructure".

In this sense, our solution pushes the arm's service to a richer and more up-to-date model: Software as

[1]https://robotnik.eu/products/mobile-robots/summit-xl-en-2/

[2]http://www.robotanno.com/en/commercial-robotic-arm150/sj605-a-robot-arm.html

a Service[8] (SaaS). As the provided user interface is specific for this kind of robot, it is applicable to robotic arms of the same family (or characteristics), whose movement information is based on the notion of joint coordinates (a list of values for each joint).

SaaS has impacted the way the information technology world thinks about its role in a wider context, that is, what is the role of providers of computing services to the rest of the enterprise? This has influenced the rise of an effective opportunity to change the focus from deploying and supporting applications to managing the services provided by applications.

A common materialisation of SaaS is the microservice architectural style[9], which is an approach to develop an application as a suite of small services, each running in its own process and available through some communication mechanism (often an HTTP resource API[3]). Each compound service can be implemented in an arbitrary programming language, concentrates on specific business capabilities, and is independently deployable.

The main contribution of this work is an ecosystem based on the microservices architectural pattern that allows remote control of a robotic arm hosted at the University of Seville. The solution promotes the abstraction of details about the arm, allows easy addition of new robotic arms (neuromorphic or not), and provides an intuitive, scalable, and portable tool to control the arm (a Web-based user interface). This has led to important results from the user point of view as well as specification standards and represents a first step in jointing different computing areas towards evolving a proof of concept (a robotic arm) to a product (a remote service).

The next section provides the background of our work for the ED-Scorbot and the microservices. Section III is focused on servitization and tests. Then, section IV presents some discussions. Finally, section V concludes the paper.

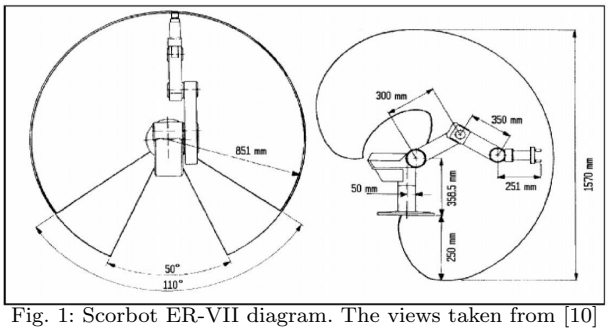


Fig. 1: Scorbot ER-VII diagram. The views taken from [10]

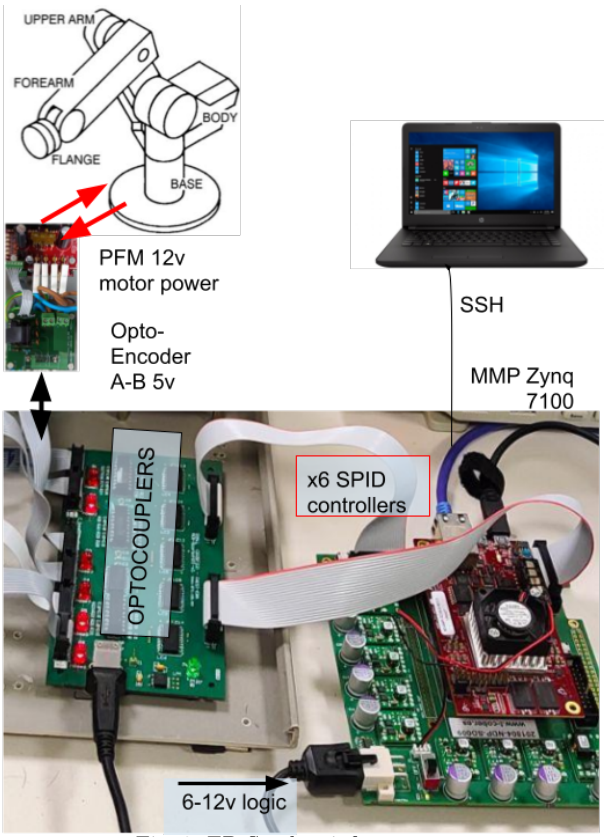


Fig. 2: ED-Scorbot infrastructure

II. Background

A. ED-Scorbot Robotic Arm

This robot consists of a Scorbot ER-VII model, six-axis with 12V DC motors with dual-channel optical encoders to record the movements of these axes. A descriptive image of the robot can be seen in Fig.˜1. Each joint's motor is managed with a SPID [4] spike-based controller, modified for position control, and driven by a set of six H-bridges that are powered through an external 12V-25A power supply, as shown in Fig. 2. These H-bridges allow the arrival of the expanded spikes at the SPID output to the motors while maintaining a Pulse-Frequency-Modulation (PFM).

The spike-based controller of this robot (1 $SPID$ per joint, 6 $SPID$ in total) is built inside the Mini-Module-Plus Zynq-7000® All Programmable SoC (PSoC) Plus 7Z100 (MMP) module from AvNet™, mounted on the DocSoC [11] board, developed in the Robotics and Computer Technology group at the University of Seville. In addition, a second board is used that contains several optocouplers and buffers that allow connection to the DC motors, as well as feedback from the optical encoders of the motor position, the microswitches that indicate the starting position of the robot, and the sensors of the limit switches [6]. This feedback is critical, as the robot has no other mechanism such as encoding the current position of each joint, so it has to be implemented in the FPGA. In this implementation, a 16-bit counter per joint is used as an absolute reference for its position and is used in parallel to close the control loop within the neuromorphic PFM controller. Thus, a mapping can be created between the commanded pulsing input, the desired angles to be achieved for each joint, and its equivalent position expressed in its counters (see [6] for more details). If a consistent homing procedure is executed after every system reset, accurate position monitorization can be achieved. Only joints one to four are taken into account because the other two joints do not affect the Cartesian position of the

[3]API: Application Programming Interface.

arm end-effector.

The MMP board is based on a PSoC that includes both an FPGA and an embedded computer, running its own operating system (OS), using a single board that will act both as a controller and as a platform from which the controller is commanded. This heterogeneous platform consists primarily of two separate parts: the processing system (PS) and the programmable logic (PL). The PS has two ARM cores on which a Petalinux OS [4] is running. On the other hand, the PL has an FPGA of the Kintex 7 family.

A simplified block diagram of the hardware platform is shown in Fig. 3. The left side depicts the PS where the OS is running and which has access to some peripherals such as USB, Ethernet and others. The Ethernet interface is used to provide internet access to the PS. This provides a connection to the platform via SSH, thus having an embedded system that can be remotely controlled comfortably. On the right side, we have the PL where the IP modules of the 6 SPIDs have been deployed. The connection between PS and PL is made through a standard Advanced eXtended Interface (AXI) bus.

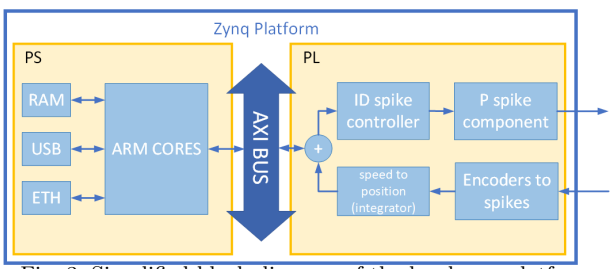


Fig. 3: Simplified block diagram of the hardware platform

The OS running in PS also contains a service for receiving commands for the robot and transmitting its information. This service is based on MQTT[12] (Message Queuing Telemetry Transport), a lightweight, publish-subscribe, machine-to-machine, network protocol for message queuing, designed for connections to remote locations that have devices with resource constraints or limited network bandwidth, such as in the Internet of Things (IoT).

B. Microservices

Regarding basic functioning requirements, the ED-Scorbot arm controller presents the following items:

- It has a cyclic behaviour to be available and responsive to new interactions from client applications via asynchronous messages.
- It is able to inform clients about its `ERROR` state and current owner all the time.
- Only one user can be the owner at a time.
- Only the owner can request to move the arm (point or trajectory), cancel a trajectory execution or disconnect from the arm.
- After accepting a new owner, the arm must be moved to its home position. During this procedure, the arm cannot execute other functions.

[4] `https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html`

In terms of the state machine, Fig. 4 represents the behaviour of the controller microservice.

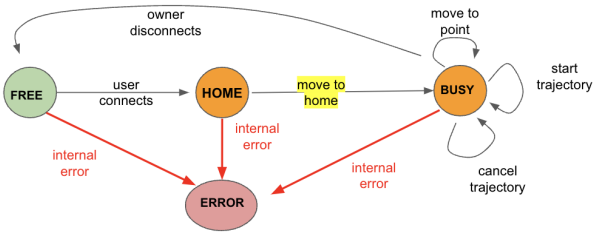


Fig. 4: The basic state machine of ED-Scorbot

Upon initialisation, we must wait until the physical arm is working properly and only then the controller become available for the Web service. If a user requests to connect to the arm and the arm is `FREE`, it accepts the user as the current owner and changes to state `HOME`. After that, the controller automatically moves the arm to its home position, changes the internal state to `BUSY`, and accepts requests from the owner to move to a point, apply a trajectory, cancel trajectory execution, or disconnect from the arm. Only the latter functionality changes the robotic arm state to `FREE` again.

As the controller is associated with a physical device, it can go to an `ERROR` state if some internal problem occurs. This is abstractly represented by the red arrows leading to the `ERROR` state where the controller cannot execute all functions that forward commands to the arm and requires human intervention (to restart the services or the physical arm).

The state machine of Fig. 4 has been used as a reference to implement the controller microservice. However, the ED-Scorbot arm has already been used remotely with different technologies.

Fig. 5 shows the previous servitization model, where the coupled MMP board contains the hardware infrastructure to execute platform-specific programs. A set of C/C++/Python functions contain several tools (scripts and low-level functions) providing support for high-level programming. The MQTT server is the controller component and also works as a layer providing communication features and transparency regarding the underlying tools.

The SSH/GUI component is a service providing access to users. Thus, although the MQTT service has communication features, it is not used directly by final users. Instead, users connect with an SSH server and get access to tools and scripts that are able to control the arm by receiving messages from the MQTT service. Moreover, there is a graphical tool that runs as a desktop application and allows users to interact with the arm (also sending messages to the MQTT service). This tool has several convenience features, including high-level (final users) and low-level (engineers) views and manipulation.

The servitization model proposed by this work offers a novel user experience while still maintaining the previous servitization model working.

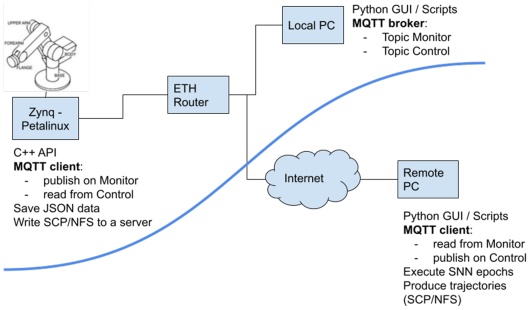


Fig. 5: The previous servitization

III. Servitization

To provide a more sophisticated remote control for the ED-Scorbot arm, we used an ecosystem (illustrated by Fig. 6) composed by independent microservices where each service has a well-defined role (high cohesion) and adheres to a contract established by API specifications (low coupling).

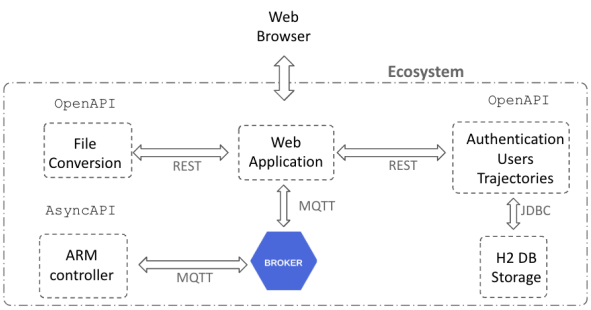


Fig. 6: The ED-Scorbot ecosystem

The Browser is at the user level and executes the Web application provided by the corresponding service. Internally, each component of the ecosystem has its role explained as follows:

- **Broker** - the asynchronous communication middleware responsible to handle all messages over MQTT protocol. We adopted the Eclipse Mosquitto broker [13] in our solution.
- **ARM Controller** - the service responsible to control the arm, implementing all features related to configuration, fault tolerance, error recovery, and business logic. Broadly, it receives requests, performs all validations, executes commands in the physical arm using underlying tools (platform-dependent executables and scripts), and sends responses associated with the requests. It has been implemented as an asynchronous C/C++ service according to a well-defined specification described in subsection III-B.
- **Authentication, Users, Trajectories and Storage** - a compound microservice with features for storing data in H2 database [14], authentication, users management and trajectories manipulation. We used Java Spring Boot framework [15] to implement it as a synchronous RESTful [16] service following the specification detailed in subsection III-A.1.
- **File conversions** - a microservice providing features for applying conversions between binary and JSON/TSV format that are accepted by the user interface. These binary files contain trajectory data manipulated by the Numpy library and need to be converted prior to their use in our tool. This service has been implemented in Flask/Python and follows the synchronous specification described in subsection III-A.2.
- **Web Application** - the top-level microservice that provides the graphical user interface. It interacts with all other services (synchronous and asynchronous) to offer remote control of the arm as a Web application. We have used the Angular [17] framework to implement this microservice and followed the API specifications detailed in subsections III-A and III-B to implement communication layers responsible to interact with the other microservices.

Let us now discuss the specifications of all microservices inside our ecosystem.

A. Synchronous Services Specifications

The synchronous microservices are adherent to specifications defined according to the OpenAPI 3.0 standard [18], where interfaces of RESTful APIs establish a contract with well-defined operation signatures, parameters, response, and abstract behaviour, so that humans and computers can discover and understand the capabilities of services without accessing any source code or documentation.

In the following, we detail the specifications establishing abstract models of the two synchronous microservices.

A.1 Authentication, Users and Trajectories

The first specification covers the functionalities of a microservice responsible for authentication, users, and trajectory manipulation. The functionalities related to **authentication** focus on handling access to the ecosystem and encompasses two routes in this functionality: `/authenticate` that validates credentials and `/signup` to provide support for new registrations. New users are registered but they are disabled automatically and, therefore, must wait until an administrator confirm and grant access.

Concerning **users**, the specification establishes basic support for adding, updating, and removing as follows. When associated with the HTTP `GET` method, the route `/users` returns the list of all users. On the other hand, when invoked using the `POST` method it adds a new user. And when requested through the `PUT` method, the route `/users/{username}` updates a specific user. It is worth pointing out that users have two possible roles: `ADMIN` or `USER`, meaning privileged and normal users, respectively. Depending on the role, each functionality can behave differently. For example, normal users are able only to access their own information, while privileged users can manipulate the information of any user.

Regarding **trajectories**, the functionalities allow obtaining all trajectories, adding and removing trajectories. When invoked using the `GET` method, the route `/trajectories` return all saved trajectories. And when associated with the `POST` method, it saves a new trajectory. The route `/trajectories/timestamp` associated with the `DELETE` method deletes a trajectory (identified by its timestamp). We highlight that any trajectory must be associated with a single user. Thus, its manipulation depends on the role of the logged user: privileged users can handle any trajectory, whereas normal users have access only to their own trajectories.

The data involved in all functionalities are defined by the structures illustrated in Fig. 7. They are associated with a functionality/route they are used.

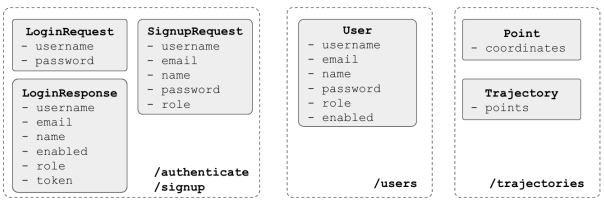


Fig. 7: Data related to authentication, users and trajectories

Note that a trajectory is composed of `points`, where each point contains `coordinates`. Mathematically, the coordinates of a point are a list of values:

$$[x_1, x_2, ..., x_n, t],$$

where x_i is the value to be assigned to the i^{th} joint of the arm. This value can be expressed in angles (degrees or radians) or in reference values. On the other hand, the last coordinate t is optional and represents a waiting time information (in milliseconds), that is, the time to wait when starting moving the arm to the point $(x_1, x_2, ..., x_n)$. This guarantees that the next movement must wait t milliseconds after the arm started moving to the point $(x_1, x_2, ..., x_n)$. Furthermore, this also provides a way to smooth trajectories. The arm has an average time to reach a point and the user is free to use a smaller waiting time during movements to produce a smooth transition between two consecutive movements.

The complete specification is available at SwaggerHub [5] and provides more details.

A.2 File Conversions

The second specification includes functionalities related to applying convenient conversions to the content of specific files. This support is necessary because most of the robotic arm's users are familiar with the `Numpy` library and can write trajectories programmatically in order to generate graphic representations or to train machine learning models. Nevertheless, the Web tool handles trajectories in JSON or in TSV formats, with point coordinates expressed in degrees.

The functionalities established by the second synchronous specification cover two simple tasks: one for **loading** the content of a `.npy` (binary) file containing a trajectory and returns it as a JSON object, and another for **converting** all points of a trajectory among *degrees*, *radians* or *reference* values.

For more information about this specification please refer to its SwaggerHub page[6].

B. The Asynchronous Specification

The way that some microservices communicate through a **broker** in our ecosystem is established by using the AsyncAPI standard [19], as it allows to describe of communication interfaces of Event-Driven Architectures as an API where functionalities, payload, and behaviour can be abstractly defined.

Concerning the ED-Scorbot ecosystem, the asynchronous API defined the way the arm controller communicates (receives and sends) messages, and the user interface agrees on the same model to send asynchronous requests and receive the results. In this sense, those applications have facets to behave as publishers and subscribers at the same time.

The Arm Controller is an application embedded in the arm's board with two main capabilities: manipulating the physical arm and providing remote control. Its specification defines all characteristics using concepts such as `channels` (or `topics`) and payloads, which are related to asynchronous systems.

The way we have designed the asynchronous specification makes the ecosystem scalable using a single broker: new arms become available automatically at startup by communicating their meta information on channel/topic `metainfo` at service initialisation.

Furthermore, each arm has a **name** that allows its identification in our ecosystem. In this sense, the **name** has a syntactic and semantic role to connect clients and arms.

The functionalities of the controller are grouped in topics that are explained as follows.

The topic `metainfo` is used to request or transmit metainformation about arms. Recall from Section II-A that a physical arm has joints that move according to a range (in angles or reference values) that establishes limits to be respected by users. Hence, the meta information of an arm is represented by a pair $(name, joints)$, where $name$ is the robot name and $joints$ is a list of pairs containing the minimum and maximum values of a joint. For example, the joints $[(min_1, max_1), (min_2, max_2), (min_3, max_3)]$ denote a robotic arm containing three joints whose minimum and maximum values are given by min_i and max_i, respectively.

The topic `name/commands` allows interactions between clients and the arm identified by `name`. Clients can send messages to connect, move the arm, disconnect from the arm, and cancel trajectory execution using this topic.

[5] https://app.swaggerhub.com/apis-docs/ADALBERTOCAJUEIRO_1/ed-scorbot-service_api/1.0.0

[6] https://app.swaggerhub.com/apis-docs/ADALBERTOCAJUEIRO_1/ed-scorbot_api/1.0.0

Finally, the topic `name/moved` is used to communicate the robotic arm's positions. Each time the arm moves to a point (even during a trajectory execution) it notifies the clients about its current position.

Regarding the payload of messages, the structures depend on the topic they are used. Fig. 8 shows the structures grouped according to the topics.

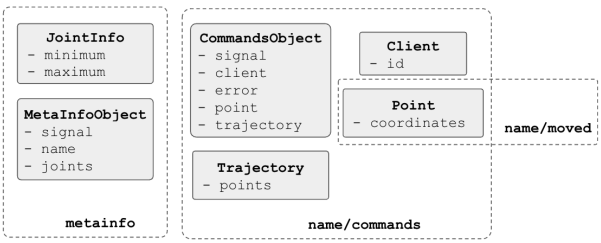


Fig. 8: Data manipulated by the asynchronous service

The `MetaInfoObject` defines the payload structure of topic `metainfo` and encapsulates three items: (*i*) an integer `signal` indicating that the user has requested arm's meta information (`1`) or the controller has sent its meta information to users (`2`), (*ii*) the controller `name` so that clients can identify an arm conveniently, and (*iii*) the information of all `joints` as a list of `JointInfo` (pair with minimum and maximum values for a joint).

As previously mentioned, the `Point` structure contains a list of coordinates possibly including time information in the last coordinate. This structure is used in topic `name/commands` to allow users to send points to the arm and in topic `name/moved` to allow the controller to notify clients about the last position of the arm.

The `Trajectory` object contains a list of points used only on topic `name/commands`.

As the controller must guarantee that the arm is controlled by only one user at a time, it maintains an instance of the `Client` object to keep the arm's owner. This instance is updated conveniently during connections and disconnections with the arm. For our context, a `Client` has only an identifier.

Finally, the `CommandsObject` represents the payload of topic `name/commands` and contains the following internal structure:

`signal` - an integer value indicating an action meaning requests from clients to controller or responses in the opposite direction. Its possible values are listed in table I.
`client` - the client involved in the message
`error` - a flag indicating if the physical arm is responsive or nor. This abstractly represents physical or hardware problems that put the physical arm in an inconsistent or blocked state and, therefore, cannot be available to users.
`point` - the point to move the arm. This field has sense only when `signal = 7`.
`trajectory` - the trajectory to execute. This field has sense only when `signal = 8`.

Complete documentation about the asynchronous specification can be found at SwaggerHub[7]).

Table I: Commands signals

Signal	Description
3	An arm status request from users
4	Controller communicates arm's status
5	Connect request from uers
6	Controller accepts a user as its owner and notifies all users
7	A request from the owner to move the arm to a specific point
8	A request from the owner to execute a trajectory
9	A request from the owner to cancel the current trajectory execution
10	The controller notifies its owner about the current trajectory cancellation
11	A request from the owner requested to disconnect from the arm
12	The controller confirmed the owner has disconnected
13	The controller notifies the owner that the arm has been moved to the home position

C. Implemented Microservices

All microservices of our ecosystem have been implemented using different technologies and are available as independent projects at Github, with detailed instructions about compiling, installing and running.

The microservice related to authentication, users, and trajectories [8] has been implemented in Java. The functionalities described by the specification are implemented as routes that are available over HTTP.

The microservice related to file conversions~[9] is implemented in Python and is available over HTTP.

The microservice implementation for the arm controller [10] uses auxiliary libraries such as Mosquitto[11] and Niels Lohmann[12] to interact with the Mosquitto MQTT broker and to convert objects into JSON (and vice-versa), respectively. It is, indeed, an implementation of a simulated robotic arm, proposed as basic implementation to be evolved to any target platform. This effort involves modifying the code to interact with the physical arm and embedding the microservice directly into the arm's infrastructure.

The Web application [13] has been implemented in Angular and is the top-level service of the ecosystem, providing a new user experience on controlling the ED-Scorbot arm. The interaction with other services was implemented using design patterns to reach transparency about the internal structure of the ecosystem and low coupling between application

[7]https://app.swaggerhub.com/apis-docs/ADALBERTOCAJUEIRO_1/ed-scorbot_async/1.0.0
[8]https://github.com/adalbertocajueiro/edscorbot-java
[9]https://github.com/adalbertocajueiro/edscorbot-py
[10]https://github.com/adalbertocajueiro/edscorbot-c-cpp
[11]https://mosquitto.org/
[12]https://github.com/nlohmann/json
[13]https://github.com/adalbertocajueiro/edscorbot-angular

modules that depend on the interaction with other microservices. This minimises the effort to adjust the user interface when any API specification changes.

We describe our user interface by explaining its functionalities separately.

The application requires user authentication to allow access to the ecosystem. The login support is exhibited as the first screen requiring credentials or allowing new registrations (see Fig. 9). By default, new users are disabled until some administrator grants access.

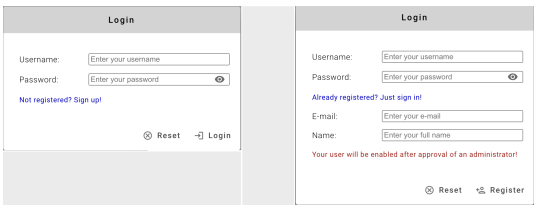


Fig. 9: Login/signup screen

The main screen (Fig. 10) contains a navigation bar at the left to provide access to the functionalities, a status bar at the top to show the available robots, a button to connect to the arm, and icons to show information about logged user and to perform a logout.

The users' manipulation page is also exhibited by Fig. 10 and allows user management activities such as enabling/disabling, changing roles, resetting passwords, and so on. If the logged user has administrator privileges it is possible to handle all users. Otherwise, only its own information can be accessed and modified.

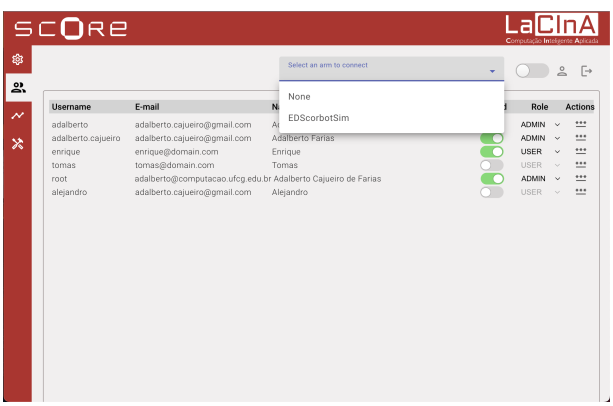


Fig. 10: Users management screen

The screen providing functionalities for converting the content of trajectory files is depicted in Fig. 11. As long as any available robot is chosen in the status bar, the functionalities become enabled. The user must first click the `upload` button and choose a `.npy` file containing trajectories compatible with the chosen arm. The application automatically loads the content and exhibits it at left. Then the user can choose the source and the target types (degrees, radians, or reference values) and apply the conversion. The application shows the converted content at right. Finally, the user can export to JSON or TSV by using the convenient button.

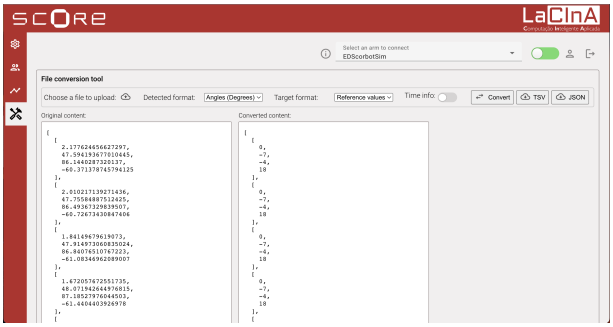


Fig. 11: Conversions screen

The movements screen is presented in Fig. 12 and provides functionalities for building points and trajectories, moving the arm, and viewing the points before sending them to the arm.

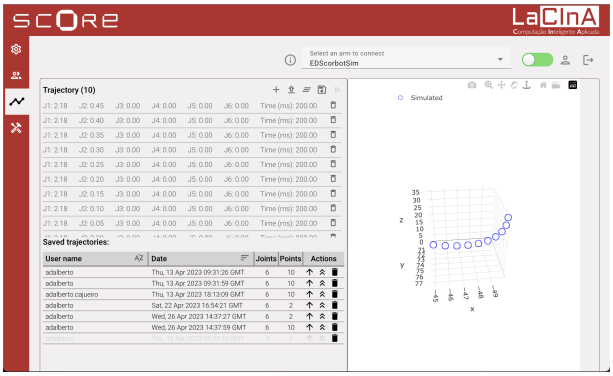


Fig. 12: Movements screen

The screen is composed of three parts. The support for building trajectories is on the top-left region. It contains buttons for adding points individually, loading points from a JSON or a TSV file, cleaning the current list of points, saving the list of points as a new trajectory, and sending points to the arm.

The addition of points is possible only if the user has selected an arm. This can be made by building individual points or loading them from a JSON or TSV file. Once the list of points is not empty, the user can change the order of points (by drag and drop), clear the list or save it as a new trajectory.

The 3D viewer component (right region) plots the points in a spatial view so that the user can see the positions (produced by applying the kinematics function of the chosen arm) before sending them to the arm. It also allows some interactions such as zoom-in, zoom-out, rotations, etc.

To execute a point or a trajectory the user must first connect to the arm. This is achieved by clicking the toggle button on the status bar. When the arm accepts a new owner, it exhibits a progress spinner indicating that the arm is moving to the home position. When the arm reaches its home position the spinner automatically disappears, indicating that the user has become the arm's owner and that the arm is ready to execute commands from its owner.

The points added by the user are plotted as filled circles in the corresponding positions. After sending them to the arm, the viewer receives the real po-

sitions one by one and plots them as filled circles. Along the execution, the user sees two traces: one for the simulated points and another for the real points in the same spatial Cartesian plane.

The tool also provides facilities to handle saved trajectories (the bottom-left region of Fig. 12). Depending on the logged user, the exhibited trajectories can change. For each exhibited trajectory, it is possible to see its time stamp, the number of joints, and the number of points. Moreover, for each enabled trajectory the user can replace the entire list of points to be executed with the trajectory, or can add all points of the trajectory to the list, or can delete the trajectory.

D. Adding a new Robot to the Ecosystem

The ecosystem has been designed to allow additions of new arms, as long as they are adherent to the asynchronous API specification and specific adjustments are made in two microservices: *file conversion* and *Web application*.

Although the point representation is generic, the conversion of coordinates from angles to reference values or vice-versa depends on the characteristics of the motors of each joint. Thus, developers of new arms must provide these functions in a specific file of the *file conversion* service and link them with the specific arm to make it work[14]. For details and an example about it, please refer to the microservice documentation.

The adjustments in the *Web application* are related to the direct kinematics function to convert a robot point (coordinates for joints) into a spatial point (3D coordinates) so that they can be plotted suitably. This can be achieved by providing an implementation of the direct kinematics function in a specific file and associating it with the arm. For details and an example about it, please refer to the service documentation.

IV. Discussion and future work

The new servitization model developed in this work has brought important concrete results:

- It provided a more sophisticated, cohesive, low coupled, heterogeneous and scalable ecosystem to control robotic arms. Besides providing a distributed processing, our heterogeneous ecosystem hides complexity for final users so that they have the impression of using a single application.
- The new user interface provides a new metaphor through a single application handling several aspects in a transparent way. Moreover, the use of modern technologies and concepts makes the Web application more intuitive, attractive and portable to control robotic arms.
- The flexibility for adding new robots is an important result, reached by using suitable design patterns, technologies and the SaaS idea.
- We have successfully used our solution considering the ED-Scorbot arm as our case study with scope in development tests. Our current effort is to perform user tests by making the ecosystem available to external users that will provide feedback to improve the user experience.
- The use of API specifications represents an important result that provided a better connection of microservices. By describing syntactical elements of services, specifications allow one to establish well-defined contracts to be followed by implementations.

Regarding future directions, we can list the following points that deserve to be explored:

- The synergy among embedded development, neuromorphic computing, and servitization has revealed a huge horizon to be explored because their combination changes the way applications are conceived, specified, implemented, and deployed. Embedded development brings benefits where robots, for example, can become more and more useful to humans. Neuromorphic computing, on the other hand, has impacted many areas by using a new model closer to the human brain. This brings more power to the machine learning field. And servitization has changed the way that the world thinks about programs as services instead of applications.
- A deeper study about similar robotic arms and variability can improve the flexibility and scalability of the ecosystem as it allows one to understand better the common characteristics and the differences between arms and face them as a product family.
- Automatic code generation from API specifications also deserves attention as it can impact the maintenance of the ecosystem. The communication between microservices is the key aspect involved in this issue. Currently, changes in the API specifications cause efforts to maintain the consistency of microservices with specifications. The automatic code derivation will reduce (or even eliminate) this effort.
- The use of formal specifications for arm's controllers is also a hot topic to be explored because it complements the API specification (syntactic) with a behavioural description (semantic). This allows one to have all artifacts necessary to completely generate the arm's controller code and enables all benefits associated with formal specifications such as automatic test generation and automatic verification of properties via model checking. This is an important contribution towards the correctness of robotic arms.

V. Conclusions

In this paper, we have used concepts of Software as a Service to elaborate a novel servitization model for the ED-Scorbot robotic arm, with the aim of enabling the use of a specific neuromorphic robot in

[14]By adding the arm's name and its functions to a hash map structure.

a remote and easy way. The robot can now be used in-the-loop of higher-level neuromorphic algorithms from international research groups, as it is done in the SMALL EU project. The solution is based on an ecosystem composed of heterogeneous and independent microservices that provide all functionalities to be offered in the entire system.

We have achieved a low coupling of components by using the concept of API specifications (synchronous and asynchronous) to be followed by the microservices. At the same time, the microservices have high cohesion as each of them deals with a specific aspect/functionality. The first microservice provides support for authentication, users and trajectory manipulation, and persistence for the entire ecosystem. The second microservice provides facilities to handle trajectory files written in other formats that need to be converted before being used. The third microservice represents an implementation of a simulated robotic arm and is intended to be a *baseline* to be evolved and embedded in the arm's platform. We have evolved our implementation, embedded it into the ED-Scorbot platform, and used it in our ecosystem.

Finally, the fourth microservice is the top-level service that communicates with all others to provide a high-level Web application as a modern user interface for remotely controlling robotic arms.

By using API specifications, design patterns, and moderns resources of Web development, our solution is scalable in the sense that any new arm will be automatically available in the ecosystem as long as they are adherent to the asynchronous API specification, publish its metainformation at startup and its conversions and kinematics functions are informed to the conversion and to the web application microservices.

Acknowledgments

This work has been partially funded by the national projects MINDROB (PID2019-105556GB-C33/AEI/10.13039/501100011033), ATENEA (PID2021-126227NB-C22 funded by MCIN/AEI/10.13039/501100011033/FEDER, UE), IRIS (TED2021-131023B-C21 funded by MCIN/AEI/10.13039/501100011033 and by European Union "NextGenerationEU"/PRTR), and CHIST-ERA H2020 grant SMALL (PCI2019-111841-2/AEI/10.13039/501100011033). Adalberto's work is supported by the Brazilian standard university staff training program. The work of Enrique Piñero-Fuentes was supported by the predoctoral grant "Formación de Profesorado Universitario (FPU)"from the Ministry of Education, Culture and Sport of the Government of Spain.

References

[1] Giacomo Indiveri and Timothy K Horiuchi, "Frontiers in neuromorphic engineering," 2011.

[2] Rasmus Stagsted, Antonio Vitale, Jonas Binz, Leon Bonde Larsen, Yulia Sandamirskaya, et al., "Towards neuromorphic control: A spiking neural network based pid controller for uav," RSS, 2020.

[3] Alejandro Linares-Barranco, Fernando Perez-Peña, Angel Jimenez-Fernandez, and Elisabetta Chicca, "Edbiorob: A neuromorphic robotic arm with fpga-based infrastructure for bio-inspired spiking motor controllers," *Frontiers in Neurorobotics*, vol. 14, pp. 96, 2020.

[4] Angel Jimenez-Fernandez, Gabriel Jimenez-Moreno, Alejandro Linares-Barranco, Manuel J. Dominguez-Morales, Rafael Paz-Vicente, and Anton Civit-Balcells, "A neuro-inspired spike-based pid motor controller for multi-motor robots with low cost fpgas," *Sensors*, vol. 12, no. 4, pp. 3831–3856, 2012.

[5] Jochen Wirtz, Paul Patterson, Werner Kunz, Thorsten Gruber, Vinh Lu, Stefanie Paluch, and Antje Martins, "Brave new world: Service robots in the frontline," 01 2018.

[6] Salvador Canas-Moreno, Enrique Pinero-Fuentes, Antonio Rios-Navarro, Daniel Cascado-Caballero, Fernando Perez-Pena, Francisco De Asis Gomez-Rodriguez, and Alejandro Linares-Barranco, "Towards spike-based motor control fpga integration forneuromorphic robotic arms for learning applications throughremote access," *Submited to Springer Autonomous Robots*, 2021.

[7] Christian Kowalkowski, Heiko Gebauer, Bart Kamp, and Glenn Parry, "Servitization and deservitization: Overview, concepts, and definitions," *Industrial Marketing Management*, vol. 60, 01 2017.

[8] Gianpaolo Carraro and Fred Chong, "Software as a service: An enterprise perspective," 2006.

[9] Martin Fowler and James Lewis, "Microservices," 2014.

[10] Eshed Robotec, "Scorbot er-vii user manual," Eshed Robotec Limited, Dec. 1998.

[11] Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Iulia-Alexandra Lungu, Moritz B. Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu, and Tobi Delbruck, "Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 3, pp. 644–656, 2019.

[12] OASIS Standard, "Mqtt version 5.0," August 2019.

[13] Roger A. Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, no. 13, pp. 265, 2017.

[14] "H2 database engine," https://www.h2database.com/, 2022.

[15] et. al Phillip Webb, Dave Syer, "Spring Boot Reference Documentation," https://docs.spring.io/spring-boot/docs/current/reference/html/.

[16] Michael Przybilski, "Rest - representational state transfer," Tech. Rep., ?

[17] "Angular Framework," https://angular.io.

[18] OpenAPI Initiative, "OpenAPI Specification," https://github.com/OAI/OpenAPI-Specification.

[19] AsyncAPI Initiative, "AsyncAPI Specification 2.6.0," https://www.asyncapi.com/docs/reference/specification/v2.6.0.

Una implementación de la Teoría de control de bucle cerrado basado en eventos dispersos

Santiago Díaz-Romero[1], Salvador Canas-Moreno[1], Enrique Piñero-Fuentes[1], Antonio Ríos-Navarro[1 2], Marcin Paluch [3], Alejandro Linares-Barranco [1 2]

Resumen— **Los sistemas de control de motores basados en eventos o neuroinspirados comunican la información mediante pulsos, ráfagas instantáneas de corriente eléctrica que codifican la información de forma muy eficiente, tal como ocurre en los organismos biológicos. En este trabajo se propone una implementación software para una teoría de control motor neuromórfico en lazo cerrado basado en eventos dispersos. Este paradigma permite desarrollar, entre otros, controladores PID pulsantes. El algoritmo de control se ha incluido en un simulador desarrollado en Python, y disponible públicamente, de un carro robotizado que controla un péndulo invertido. Las implementaciones propuestas se pueden portar fácilmente a cualquier FPGA o placa basada en microcontrolador y permiten controlar el robot real. Hemos probado y verificado la funcionalidad, obteniendo los parámetros equivalentes a las constantes de proporcionalidad, integración y derivación para este controlador pulsante. Los resultados empíricos demuestran la validez tanto de la teoría, en simulación, como de la implementación en un robot real, en el ámbito pulsante. En la simulación, para el control de ángulo mediante un PID, con un tiempo mínimo de reacción de 1ms del controlador, el péndulo se mantiene en equilibrio a lo largo del recorrido del carro más de 5min, registrando el paso por los 0 grados más de 2800 veces. Para el péndulo real usado, el controlador LQR y las constantes de control óptimas, con un tiempo mínimo de reacción de 6ms del controlador, el péndulo se mantiene en equilibrio a lo largo del recorrido del carro así como ante perturbaciones leves y moderadas sin observarse caída durante todos los experimentos realizados.**

Palabras clave— **Ingeniería Neuromórfica, Control motor, Controlador Neuromórfico, Control basado en eventos, Python, FPGA, Arduino.**

I. Introducción

TRADICIONALMENTE , el control de los sistemas electrónicos de potencia ha estado dominado por los controladores analógicos. Los controladores digitales se han ido mejorando constantemente en términos de coste y facilidad de uso en las últimas décadas, haciendo que este tipo de controladores sean más atractivos para sustituir a los controladores analógicos en muchos dispositivos [1]. La implementación de controladores digitales incluye ventajas potenciales como una flexibilidad significativamente mejorada, menor tiempo de diseño, programabilidad, eliminación de componentes discretos de sintonización, mayor fiabilidad, integración más sencilla y la posibilidad de incluir una variedad de características de rendimiento [2].

[1]Dpto. de Arquitectura y Tecnología de Computadores, Universidad de Sevilla, e-mail: sdiaz5@us.es

[2]Smart Computers Systems Research and Engineering (SCORE). University of Seville

[3]Instituto de Neuroinformática, Universidad de Zurich, e-mail: marcin.p.paluch@gmail.com

En los controladores digitales, uno de los aspectos principales es la frecuencia de muestreo de la señal, que permite la discretización. La elección de una frecuencia de muestreo adecuada depende de la dinámica de los sistemas de lazo abierto, así como de la dinámica deseada del sistema controlado. Un sistema con una dinámica rápida necesita una frecuencia de muestreo elevada para garantizar la estabilidad del sistema controlado, a costa de una mayor potencia de cálculo. Además, la entrada controlada de un sistema dinámico se actualiza tradicionalmente en cada paso de tiempo independientemente de la amplitud del error, es decir, de forma periódica. Cuando un sistema controlado se encuentra en un punto de consigna estable, evidentemente no hay necesidad de muestrear los datos, actualizar el controlador y el actuador. De hecho, hacerlo es energéticamente ineficiente.

Esta falta de eficiencia unida a una menor capacidad computacional es el punto de partida de los sistemas de control dirigidos por eventos [3], [4]. Estos sistemas son muy interesantes por muchas razones. Una de ellas es el hecho de que son muy similares a la forma en que un sistema nervioso biológico se comporta como controlador, como por ejemplo el humano. El sistema de control motor humano se basa en eventos o impulsos en lugar de consignas periódicas [5]. Sólo se emprende una nueva acción de control cuando la señal de medición se ha desviado lo suficiente de la consigna deseada. Otra razón interesante es la utilización de recursos. Un controlador integrado suele implementarse con un sistema operativo en tiempo real que admite programación concurrente. Los recursos de la CPU se comparten entre los núcleos/hilos de tal manera que parece como si cada tarea se estuviera ejecutando de forma independiente. Por lo tanto, utilizar recursos de la CPU para realizar cálculos de control cuando no ha ocurrido nada significativo en el proceso es claramente un desperdicio innecesario de recursos. También podemos aplicar esta idea a los recursos de comunicación. El ancho de banda de comunicación disponible en un sistema distribuido es limitado. Utilizarlo para enviar datos de forma temporizada es claramente un desperdicio de ancho de banda. La última razón es que este tipo de sistemas son excelentes candidatos para la computación de bajo consumo y baja latencia, ya que el consumo de energía es inferior al de sistemas similares y su latencia también es mucho menor, al igual que ocurre en los sistemas biológicos.

En los sistemas de control basados en eventos, se

envía un evento cada vez que la señal muestreada aumenta o disminuye en un determinado umbral. Cuando la señal no cambia, intrínsecamente no hay más actualizaciones. Estos sistemas de control también presentan inconvenientes. Por ejemplo, cuando transmiten el valor de la señal dentro de un evento, ya que limita el rango dinámico del sistema, es decir, la relación entre el mayor y el menor valor que puede asumir una señal dentro de un sistema de control. Esto se debe a que la transferencia de datos debe tener la misma representación que los datos del sistema. Así, si se quiere controlar un sistema con una gran precisión en un amplio rango de valores, dicho sistema de control necesitaría una red con un gran ancho de banda para transmitir los valores de la señal [6].

En este trabajo presentamos una implementación del algoritmo dirigido por eventos propuesto en [7] (en adelante SECLOC). Este algoritmo propone un método de discretización no lineal basado en la duración entre eventos, demostrando que el método alcanza una precisión arbitraria independientemente de la amplitud de consigna sin aumentar el ancho de banda de transmisión de datos de la red. El método disminuye el número total de muestras necesarias para estimar los estados de un sistema dinámico y la tasa de actualización de un actuador, haciéndolo más eficiente energéticamente. La implementación de hardware propuesta en este artículo se refiere a la particularización del algoritmo mediante el uso de una discretización basada en eventos logarítmicos con base 1,05. Se ha seleccionado esta base ya que en el artículo original demostró que proporcionaba buenos resultados.

El resto del artículo se organiza como sigue. En la sección II se ofrece una breve explicación de la teoría del SECLOC. En la sección III se describen las distintas implementaciones desarrolladas para este artículo. En la sección IV se describe el experimento del péndulo invertido. Y en la sección V se presentan las conclusiones.

II. El algoritmo SECLOC

Una gran cantidad de sistemas dinámicos usan controladores discretos. La elección del tiempo de muestreo es fundamental en el diseño de estos controladores. Un tiempo de muestreo muy bajo aumenta la precisión a cambio de un mayor costo en computo y energía. Por el contrario, tiempos de muestreo muy bajos pueden reducir el consumo pero también reducen la región donde el control es estable y por tanto la utilidad del mismo puede verse mermada o anulada para una aplicación concreta. Además, el uso de un tiempo de muestreo fijo, hace que se consuman recursos incluso cuando se ha alcanzado el valor de la consigna, lo que lo hace poco eficiente energéticamente.

En el algoritmo SECLOC la discretización se realiza de forma no lineal. En la discretización no lineal (a diferencia de la discretización lineal), el paso necesario para generar eventos disminuye a medida que la señal de control se aproxima al valor consignado. Es decir, el número de eventos generados tiende a infinito conforme el error absoluto se aproxima a 0. Por tanto, se hace necesario detener la generación de eventos cuando el error absoluto es menor que un valor umbral establecido (Figura 1) añadiendo un periodo refractario o de reposo.

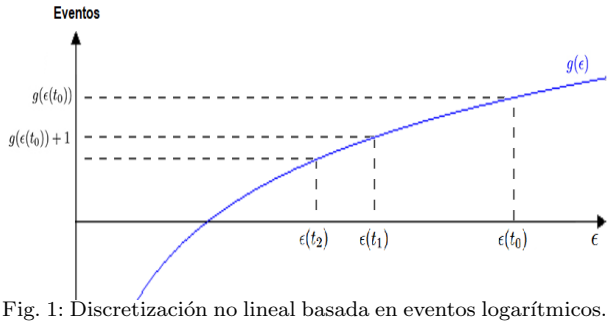


Fig. 1: Discretización no lineal basada en eventos logarítmicos.

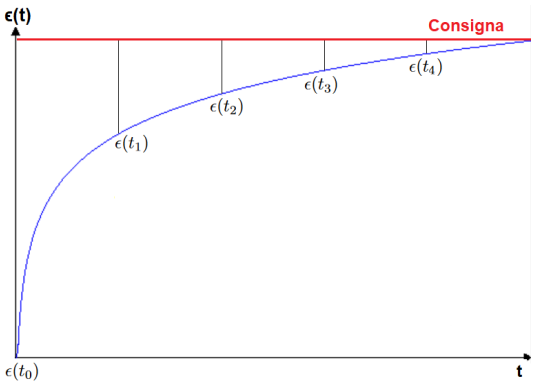


Fig. 2: Evolución temporal del sistema.

Una candidata a la función de discretización g debe cumplir tres condiciones:

1. $g : \mathbb{R}^{+,*} \to \mathbb{R}$
   $x \mapsto g(x)$
2. g debe ser continua y estrictamente monotónica en $\mathbb{R}^{+,*}$.
3. $\lim_{x\to 0} g(x) = \begin{cases} -\infty & \text{si } dg/dx > 0 \\ +\infty & \text{si } dg/dx < 0 \end{cases}$

Se ha seleccionado una función logarítmica de base b de $\epsilon(t)$ para g y que cumple los requisitos anteriores:

$$\forall b \in \mathbb{R}^+, b \neq 1$$

$$g(\epsilon) = \frac{\ln|\epsilon|}{\ln b} = log_b|\epsilon| \tag{1}$$

De este modo, es posible generar una serie temporal como:

$$\forall b \in \mathbb{R}^+, b \neq 1, i \in \mathbb{N}^+$$

$$\left\{ ti \left| \left| \frac{\ln|\epsilon(t_i)|}{\ln b} - \frac{\ln|\epsilon(t_{i-1})|}{\ln b} \right| = 1 \right\} \tag{2}$$

Para indicar si el error ha aumentado o disminuido entre dos eventos consecutivos, se añade a la ecuación información sobre la polaridad:

$$b > 1$$

$$(t_i, p_i = +1) \left| \left| \frac{\epsilon(t_i)}{\epsilon(t_{i-1})} \right| = b \right. \tag{3}$$

$$(t_i, p_i = -1)\left|\frac{\epsilon(t_{i-1})}{\epsilon(t_i)}\right| = b \tag{4}$$

Es necesario tener en cuenta que se ha definido de este modo debido a la elección realizada para la base (1.05). Es posible realizar modificaciones a la ecuación para tomar una base $0 < b < 1$[7].

De esta forma, el error a lo largo del tiempo queda definido como:

$$b > 1$$

$$\epsilon(t) = \epsilon(t_0) + (-1)^{s_N} \sum_{i=1}^{N} (b^{p_i} - 1)\epsilon(t_{i-1})H(t - t_i)) \tag{5}$$

Donde $H(x)$ es la función escalón de Heaviside.

Despejando de (3), se puede escribir el valor del error de la función en el instante t_i usando una recursión geométrica:

$$\epsilon(t_i) = \frac{\epsilon(t_0)}{b^i} \tag{6}$$

De manera que para cualquier valor finito de $\epsilon(t_i)$:

$$\lim_{n \to +\infty} \epsilon(t_i) = 0$$

Esto demuestra que es posible alcanzar una precisión arbitraría mediante la discretización logarítmica descrita. Una precisión arbitraria, es necesaria para garantizar que el sistema de control alcanza un valor de error que pueda considerarse nulo (punto en que se alcanza el valor consignado, Figura 2).

Como se ha comentado con anterioridad, es necesario incluir un periodo refractario, ya que una vez ϵ sea 0, el tiempo entre eventos disminuye también a 0. Es decir, el número de eventos sería (teóricamente) ∞. De esta forma se define el periodo refractario, como el tiempo mínimo entre dos eventos que el sistema es capaz de generar. Este término nace en neurociencia como una propiedad de la actividad neuronal [8]. Se debe tener en cuenta que la ratio entre el error pasado y el actual puede haber cambiado en varias ocasiones durante el periodo refractario. Por este motivo se añade un último parámetro que representa el número de veces que se sobrepasa el valor consignado entre dos eventos r_n:

$$\begin{aligned} r_i &= \left\lfloor \frac{\ln\left(\epsilon(t_{i-1})/\epsilon(t_i)\right)}{\ln b} \right\rfloor \quad si \;\; p_i = -1 \\ &= \left\lfloor \frac{\ln\left(\epsilon(t_i)/\epsilon(t_{i-1})\right)}{\ln b} \right\rfloor \quad si \;\; p_i = +1 \end{aligned} \tag{7}$$

Finalmente, la ecuación incluyendo r_n puede escribirse como:

$$b > 1$$

$$\epsilon(t) = \epsilon(t_0) + (-1)^{s_N} \sum_{i=1}^{N} (b^{-p_i r_i} - 1)\epsilon(t_{i-1})H(t - t_i) \tag{8}$$

Esta discretización permite beneficiarse de una representación basada en eventos en el contexto del control de sistemas. Además, permite aplicar, cuando surgen estos eventos, el algoritmo de control deseado. En este trabajo usamos el controlador PID (Proporcional, Integral y Derivativo) para mantener el péndulo en un rango mínimo de oscilación sobre su posición de equilibrio y un controlador LQR (Linear–Quadratic Regulator)[9] para controlar el sistema tanto en ángulo como en posición.

III. Implementación

En primer lugar se ha partido de una implementación del algoritmo SECLOC en C para Arduino, de los autores de [7], y se ha portado a Python y adaptado a fin de poder parametrizar el controlador en un simulador para su validación. Posteriormente, se ha aplicado el algoritmo al control en ángulo de un péndulo invertido con la finalidad de implementarlo posteriormente en un péndulo invertido robotizado con las mismas características. Para ello se ha utilizado el software CartPoleSimulation[1], desarrollado en lenguaje Python por el Instituto de Neuroinformática de la Universidad de Zurich. La interfaz del simulador puede observarse en la Figura 4. Este software implementa las ecuaciones dinámicas del robot y permite programar cualquier tipo de control y comprobar su desempeño en un simulador. Adicionalmente puede hacer uso de un módulo hardware, basado en microcontrolador, que actúa de puente entre el software y un carro físico(Figura 3). Esta fase ha permitido explorar diferentes rangos de las constantes del PID usadas en el SECLOC para una comportamiento óptimo en el control del péndulo.

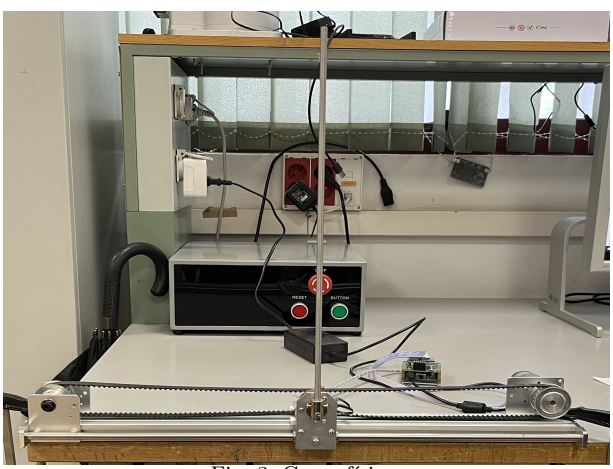


Fig. 3: Carro físico.

El software ha sido desarrollado tomando como base un modelo concreto de carro físico (Figura 5), lo que permite utilizar los resultados obtenidos directamente sobre el mismo sin la necesidad de volver a hallar los parámetros del control PID. También puede configurarse para emular el comportamiento de un carro con características definidas por el usuario.

La implementación software puede dividirse en dos partes principales: la generación de eventos mediante el algoritmo SECLOC propiamente dicho y la generación de la señal de control para el controlador PID

[1] https://github.com/SensorsINI/CartPoleSimulation

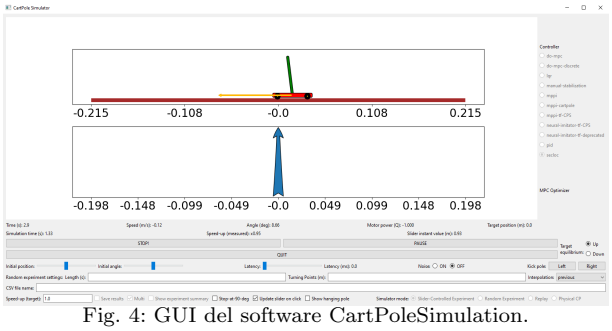


Fig. 4: GUI del software CartPoleSimulation.

aplicado a una señal discreta en el tiempo.

Una versión simplificada de la implementación de la generación de eventos puede observarse en el algoritmo 1.

La señal de control PID para la discretización logarítmica basada en eventos propuesta puede escribirse como:

$$u(t_k) = K_p\epsilon(t_k) + K_i I(t_k) + K_d\frac{d\epsilon(t_k)}{dt} \tag{9}$$

Dónde cada uno de los términos asociados a los errores pueden calcularse como:

$$\epsilon(t_k) = (-1)^{S_k}(\epsilon(t_{k-1}) + (b^{p_k} - 1)) \tag{10}$$

$$\frac{d\epsilon(t_k)}{dt} \approx \epsilon(t_{k-1})\frac{(-1)^{S_k}(b^{p_k} - 1)}{t_k - t_{k-1}} \tag{11}$$

$$\begin{aligned} I(t_k) = \int_0^{t_k} \epsilon(\sigma)d\sigma \approx \\ \approx I(t_{k-1}) + \epsilon(t_{k-1})\frac{2 + (-1)^{s_k}(b_k^p - 1)}{2(t_k - t_{k-1})} \end{aligned} \tag{12}$$

Algorithm 1 Generación de eventos SECLOC

```
error ← consigna − posicion
if |error| > banda_muerta then
    if tiempo ≥ periodo_ref then
        if error_ant = 0 then
            aum_ratio ← base
            signo_ratio ← sign(error)
        end if
        if (error ≠ 0 & error_ant ≠ 0) then
            aum_ratio ← |error/error_ant|
            red_ratio ← 1/aum_ratio
            signo_ratio ← aum_ratio
        end if
        if aum_ratio ≥ log_base then
            cambio_signo ← signo_ratio
            error_ant ← error
            polaridad ← 1
            emite_evento()
        else if red_ratio ≥ log_base then
            cambio_signo ← signo_ratio
            error_ant ← error
            polaridad ← −1
            emite_evento()
        end if
    end if
end if
```

La implementación en software de dicha señal de control es una conversión directa de las ecuaciones anteriores.

Se puede observar que se ha añadido una banda muerta en la que el controlador no actúa. Esto se ha hecho para ayudar a filtrar el ruido del sensor, ya que este puede distorsionar en gran medida el comportamiento del filtro.

Para la implementación hardware se cuenta con una placa de control basada en un microcontrolador STM32F103C8T6 que puede funcionar como puente entre el software de simulación y el carro físico (Figura 5). De esta forma, es posible utilizar el controlador que se ha implementado en el simulador directamente sobre el carro sin necesidad de realizar ninguna codificación adicional.

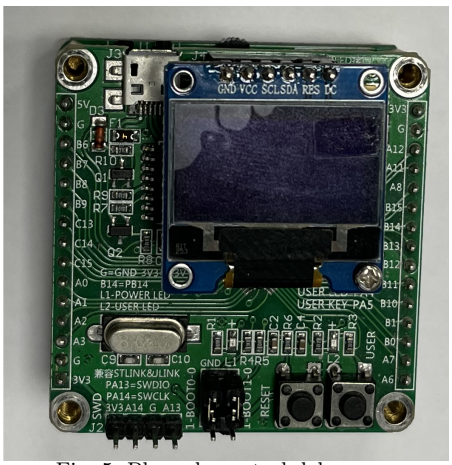


Fig. 5: Placa de control del carro.

IV. Experimentos en Péndulo Invertido

Las características principales con las que el software está configurado por defecto se corresponden con el mencionado carro físico. Estas pueden consultarse en la Tabla I:

Tabla I: Características del carro.

Propiedad	Valor
Longitud utilizable de la pista	220 mm
Longitud del péndulo	395 mm
Longitud del carro	44 mm
Masa del péndulo	87 g
Masa del carro	230 g
Fuerza máxima del motor	2,62 N

Debido a las características de la generación de eventos propuesta, no ha sido posible utilizar ningún método clásico para el ajuste de parámetros de un controlador PID, como los métodos de Ziegler-Nichols[10] o Cohen-Coon.

Para facilitar la tarea de obtención de las constantes del PID, se ha añadido al simulador la posibilidad de realizar simulaciones en bucle. En este modo el simulador empieza realizando la simulación con las constantes PID que se encuentran definidas en un fichero de configuración. Cuenta las veces que el ángulo del péndulo cambia de signo y por tanto, pasa por 0 grados (posición de equilibrio) y detiene

la simulación cuando el péndulo cae por debajo de la horizontal. Una vez finalizada la simulación, muestra el número de veces que el péndulo ha pasado por 0 grados e indica las constantes actuales del PID. Después, aumenta una de las constantes y reinicia la simulación hasta haber probado todas las combinaciones posibles en el rango indicado. El tamaño del incremento de las constantes PDI, así como los límites superiores e inferiores de cada una, son configurables individualmente.

Haciendo uso de esta herramienta, se ha configurado el controlador para que la búsqueda de las constantes quede definida con los parámetros que aparecen en la Tabla II. En todos los experimentos el ángulo de inicio θ se ha definido en 0,10 radianes (5,73 grados) para ayudar a iniciar el movimiento. Por otro lado, el periodo refractario se ha establecido en $1ms$ y la banda muerta en $0,001rad$ (este parámetro no influye en la simulación, únicamente en su aplicación real).

Tabla II: Configuración de búsqueda de constantes PID.

Parámetro	min	max	paso
K_p	0	100	10
K_i	0	1	0,1
K_d	0	100	10

En la Figura 6 se muestra de manera visual, la distribución completa de los resultados obtenidos para los $11^3 = 1331$ experimentos realizados por el simulador.

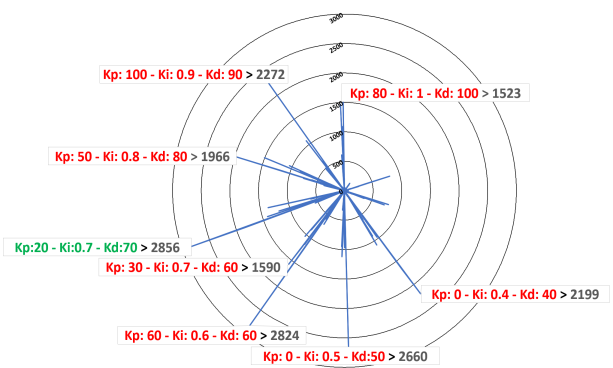


Fig. 6: Resultados de la búsqueda de constantes del PID.

Puede observarse cómo el controlador de constantes $K_p = 20$, $K_i = 0,7$ y $K_d = 70$ es el que obtiene mejor resultado con un total de 2856 pasos por 0. Posteriormente, se ha realizado una simulación con estas constantes para comprobar el resultado en detalle. Esta se ha traducido en 340 segundos de control antes de alcanzar uno de los extremos del carro y caer como consecuencia de la colisión, los resultados de esta simulación pueden verse en Figura 7.

Aunque con esta parametrización, durante la simulación, el péndulo mantiene la verticalidad el tiempo indicado, analizando los resultados de la simulación en detalle, el sistema parece tender a la inestabilidad. Por este motivo se decidió repetir la experimentación ignorando completamente el error en la posición del carro y basándose únicamente en la estabilidad del

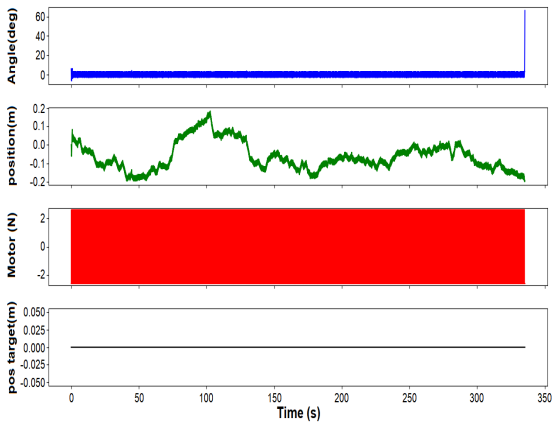


Fig. 7: Resultados PID $K_p = 20$, $K_i = 0,7$ y $K_d = 70$.

ángulo ya que esta es realmente la variable a controlar. De entre todas las opciones $K_p = 750$, $K_i = 0$ y $K_d = 0$ fue el que produjo mejores resultados, siendo capaz de equilibrar y mantener la posición del péndulo durante aproximadamente 2 segundos, mostrando un comportamiento estable (Figura 8). Aunque, como es obvio, el carro golpea rápidamente el límite y cae.

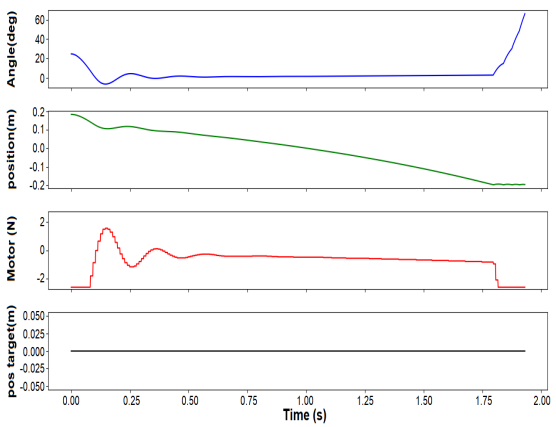


Fig. 8: Resultados PID $K_p = 750$, $K_i = 0$ y $K_d = 0$.

Posteriormente, para paliar estos efectos, se ha implementado un controlador LQR en el simulador utilizando la generación de eventos propuesta por el algoritmo SECLOC y la misma base, periodo refractario y banda muerta que en el controlador PID. El controlador LQR suele ser muy efectivo en sistemas no lineales, como es el caso de un péndulo invertido. El mayor inconveniente respecto al PID es que el controlador LQR requiere disponer de un modelo del sistema. Por este motivo, se ha hecho uso de las expresiones analíticas derivadas de las ecuaciones de Euler-Lagrange. Las descripciones de estas, así como del funcionamiento del controlador LQR, se han obtenido de [11] y [12]. En síntesis, en los controladores LQR, deben parametrizarse las matrices de coste Q y R (en lugar de las constantes K_p, K_i y K_d). Esta elección queda generalmente a criterio del diseñador, por lo que al igual que en el caso anterior, los valores

de Q y R se han hallado a base de prueba y error.

En este caso, no ha sido necesaria la realización de pruebas sistemáticas ya que se han obtenido buenos resultados muy rápidamente. Se ha priorizado mantener el equilibrio por encima de mantener la posición del carro y se han seleccionado los siguientes valores de Q y R:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad y \qquad R = 1$$

Con estos parámetros, los resultados obtenidos durante la simulación fueron excelentes, estabilizando el péndulo tanto en ángulo como en posición y manteniendo la estabilidad ante perturbaciones.

Debido a los buenos resultados obtenidos por el controlador LQR en el simulador, se ha implementado este mismo en el hardware.

Para su implementación en el sistema hardware ha sido necesaria la realización de algunos cambios, ya que el sistema presenta algunas limitaciones a la frecuencia máxima a la que puede trabajar. Por este motivo, se ha aumentado el periodo refractario a $6ms$ y se ha añadido una banda muerta de $0,01rad$ para ignorar el ruido generado por los sensores.

A pesar del aumento del periodo refractario y de la inclusión de una banda muerta, el péndulo ha podido ser controlado en ángulo y posición de manera efectiva.

V. Conclusiones

Se han implementado dos controladores diferentes basados en el algoritmo SECLOC para controlar un péndulo invertido, uno basado en un controlador PID y otro en un controlador LQR. El algoritmo ha demostrado su funcionalidad con ambos controladores, si bien el controlador PID no ha podido controlar el péndulo invertido en ángulo y posición. Esto se debe a una limitación propia de los controladores PID, no del algoritmo de generación de eventos SECLOC.

Por otro lado, la implementación del controlador LQR ha sido capaz de controlar tanto el ángulo como la posición del carro, arrojando excelentes resultados tanto en su despliegue en el simulador como en el carro físico, a pesar de las limitaciones impuestas en este último.

Estos buenos resultados dejan la puerta abierta a una futura implementación hardware del algoritmo en una FPGA (Field Programmable Gate Array). Esto permitiría eliminar las limitaciones físicas impuestas por el sistema actual y utilizarlo para controlar un sistema real. También es posible estudiar si éste método de generación de eventos puede ser aplicado a otros algoritmos de control más sofisticados que permitan un control más óptimo del sistema o el control de sistemas más complejos.

Agradecimientos

Esta trabajo ha sido parcialmente soportado por los proyectos MINDROB (PID2019-105556GB-C33), financiado por MCIN/AEI/10.13039/501100011033, y CHIST-ERA H2020 grant SMALL (PCI2019-111841-2), financiado por MCIN/AEI/10.13039/501100011033. El trabajo de Enrique Piñero-Fuentes fue soportado por la beca predoctoral "Formación de Profesorado Universitario(FPU)"del Ministerio de Educación, Cultura y Deporte del Gobierno de España. Los autores agradecen la colaboración con el equipo de Tobi Delbruck del Instituto de Neuroinformática de Zurich por el uso del robot.

Referencias

[1] P Murphy, M Xie, Y Li, M Ferdowsi, N Patel, F Fatehi, A Homaifar, and F Lee, "Study of digital vs analog control," in *Power Electronics Seminar Proceedings (CPES Center for Power Electronics Systems)*. Citeseer, 2002, pp. 203–206.

[2] Aleksandar Prodic and Dragan Maksimovic, "Digital pwm controller and current estimator for a low-power switching converter," in *COMPEL 2000. 7th Workshop on Computers in Power Electronics. Proceedings (Cat. No. 00TH8535)*. IEEE, 2000, pp. 123–128.

[3] RC Dorf, M Farren, and C Phillips, "Adaptive sampling frequency for sampled-data control systems," *IRE Transactions on Automatic Control*, vol. 7, no. 1, pp. 38–47, 1962.

[4] R Tomovic and G Bekey, "Adaptive sampling based on amplitude sensitivity," *IEEE Transactions on automatic control*, vol. 11, no. 2, pp. 282–284, 1966.

[5] Carver Mead and Mohammed Ismail, *Analog VLSI implementation of neural systems*, vol. 80, Springer Science & Business Media, 1989.

[6] Laurentiu Hetel, Christophe Fiter, Hassan Omran, Alexandre Seuret, Emilia Fridman, Jean-Pierre Richard, and Silviu Iulian Niculescu, "Recent developments on the stability of systems with aperiodic sampling: An overview," *Automatica*, vol. 76, pp. 309–335, 2017.

[7] Pierre Daye, Sio-Hoi Ieng, and Ryad Benosman, "A theory for sparse event-based closed loop control," *Frontiers in neuroscience*, vol. 13, pp. 827, 2019.

[8] H. W. Heiss, "Human physiology. edited by robert f. schmidt and gerhard thews. springer-verlag, berlin-heidelberg-new york (1983) 725 pages, 569 figures, approx. $39.20 isbn: 3-540-11669-9," *Clinical Cardiology*, vol. 6, no. 9, pp. A43–A44, 1983.

[9] H Kwakernaak and Raphael Sivan, *Linear Optimal Control Systems*, John Wiley & Sons, Nashville, TN, Jan. 1972.

[10] J. G. Ziegler and N. B. Nichols, "Optimum Settings for Automatic Controllers," *Transactions of the American Society of Mechanical Engineers*, vol. 64, no. 8, pp. 759–765, 12 2022.

[11] Steven L Brunton and J Nathan Kutz, *Data-driven science and engineering*, Cambridge University Press, Cambridge, England, Feb. 2019.

[12] Jean-Pierre Corriou, *Process control*, Springer, London, England, 2004 edition, Mar. 2004.

[13] Erick L Oberstar, "Fixed-point representation & fractional math," *Oberstar Consulting*, vol. 9, 2007.

[14] Wayne T Padgett and David V Anderson, "Fixed-point signal processing," *Synthesis Lectures on Signal Processing*, vol. 4, no. 1, pp. 1–133, 2009.

[15] Yuen Fong Chan, Mehrdad Moallem, and Wei Wang, "Design and implementation of modular fpga-based pid controllers," *IEEE transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1898–1906, 2007.

[16] Jai Karan Singh, Mukesh Tiwari, and Vishal Sharma, "Design and implementation of i2c master controller on fpga using vhdl," *IJET*, vol. 4, no. 4, 2012.

Implementación del acelerador neuromórfico de SNNs ReckON en MPSoC

Alejandro Linares Barranco[1], Luciano Prono[2], Thomas Limbacher[3], Robert Lengenstein [3], Giacomo Indiveri[4], Charlotte Frenkel[5]

Resumen— **La ingeniería neuromórfica estudia, modela e implementa los sistemas neuronales biológicos en simuladores, sistemas reconfigurables e incluso en microchips para mejorar el estado del arte en la resolución de ciertos problemas donde los sistemas neuronales biológicos son más eficientes y requieren de un menor consumo. Con el auge de la inteligencia artificial, los modelos de neuronas biológicas se están utilizando para implementar redes neuronales que puedan aprender ciertas tareas tras una fase de entrenamiento. Un conjunto de estas redes son las redes neuronales pulsantes que se apoyan en el modelo simplificado de neurona presentado en el ámbito de la ingeniería neuromórfica, la neurona que integra y dispara (Integrate and Fire). Son varios los aceleradores que han surgido para implementar estas redes. El sistema ReckON permite tanto entrenar como ejecutar una red neuronal pulsante. Este sistema ha sido descrito con un lenguaje de descripción de hardware y se ha fabricado un ASIC a partir de él. En este trabajo se ha tomado esta descripción del hardware y se ha adaptado para su implementación en un MPSoC de Xilinx basado en FPGA y sistema computador multiprocesador empotrado. Se describen los circuitos necesarios para su correcto funcionamiento y uso desde un notebook de Jupyter que se ejecuta en el sistema empotrado de la plataforma Pynq ZU.**

Palabras clave— **SNNs, ingeniería neuromórfica, FPGA, MPSoC, Jupyter notebook**

I. Introducción

La ingeniería neuromórfica (NE) es un campo de investigación en rápido crecimiento que se enfoca en el diseño y desarrollo de sistemas computacionales inspirados en la biología del cerebro humano. En particular la NE imita la forma en que las neuronas, el elemento básico de procesamiento de un sistema neuronal, se comunican entre ellas. Esta comunicación se realiza con impulsos eléctricos, que son el resultado de una acumulación de actividad en la neurona, denomidado potencial de membrana, provocada por la ponderación de la actividad pulsante de entrada en las sinapsas, que aplican un peso sinaptico configurable para permitir un ajuste en la activdad de salida. Un pulso de salida es el resultado de un potencial de acción que ocurre cuando el potencial de membrana sobrepasa un umbral, también configurable. Uno de los enfoques más populares en NE es la utilización de redes neuronales, que son sistemas computacionales capaces de aprender y realizar tareas complejas mediante el procesamiento de datos de entrada. Pero en su versión pulsante.

Las redes neuronales pulsantes son una variante particular de las redes neuronales, que utilizan pulsos en lugar de señales analógicas o digitales para procesar la información. A diferencia de las redes neuronales tradicionales, que se basan en la suma ponderada de entradas y la aplicación de una función de activación no lineal [1], las redes neuronales pulsantes, al utilizar pulsos para transmitir información a través de la red [2], les permite procesar la información de manera mucho más eficiente que las redes neuronales tradicionales, y con un menor consumo [3].

La implementación de aceleradores de redes neuronales pulsantes en hardware es un área de investigación en constante evolución que ha atraído mucha atención en los últimos años y utilizado técnicas de diseño tanto analógicas como digitales, tanto síncronas como asíncronas, basados en ASICs [4] [5] [6] [7] [8], FPGAs [9] [10] o NOC [11]. Los aceleradores de redes neuronales son dispositivos especializados diseñados para realizar los cálculos de las redes neuronales de manera eficiente y en paralelo. Los aceleradores de redes neuronales pulsantes aprovechan las propiedades únicas de las redes neuronales pulsantes para lograr un rendimiento aún mayor y una eficiencia energética mejorada en comparación con los aceleradores de redes neuronales tradicionales [3].

Uno de los principales desafíos en la implementación de aceleradores de redes neuronales pulsantes es el diseño de circuitos que puedan manejar pulsos de alta frecuencia. Los pulsos en las redes neuronales pulsantes son típicamente de corta duración y con baja latencia entre ellos cuando se agrupan en ráfagas, lo que requiere que los circuitos sean capaces de procesar información respetando esta latencia mínima exigible. Típicamente se usan técnicas de multiplexación temporal y comunicación asíncrona para recibir y extraer la actividad de miles de neuronas de un chip a otro. El más común es el Address-Event-Representation (AER) [12].

Para abordar estos desafíos, los investigadores han desarrollado varias técnicas de diseño y herramientas de simulación para la implementación de hardware de redes neuronales pulsantes y su correcto entrenamiento o aprendizaje de los pesos sinapticos adecuados para cada desafío donde aplicarlos. Estas técnicas incluyen el diseño de circuitos de reloj, el uso de algoritmos de sincronización, el diseño de circuitos de alta velocidad y la utilización de tecnologías de semiconductores avanzadas. Por otro lado, es de suma

[1]Dpto. de Arquitectura y Tecnología de Computadores, Universidad de Sevilla, e-mail: alinares@atc.us.es
[2]Dpto. of Electronics and Telecommunications, Politecnico di Torino, e-mail: luciano.prono@polito.it
[3]Institute of Theoretical Computer Science, Graz University of Technology, e-mail: thomas.limbacher@tugraz.at
[4]Institute of Neuroinformatics, University of Zurich, e-mail: giacomo@ini.uzh.ch
[5]Dept of Microelectronics, TU Delft, e-mail: c.frenkel@tudelft.nl

importancia el uso de simuladores como Nengo [13], Bindsnet [14], Slayer [15], snnTorch [16] o Norse [17], entre otros, que nos permiten explorar la arquitectura SNN adecuada a nuestro problema, así como la técnica de aprendizaje más adecuada de forma previa a la implementación hardware.

Además, los investigadores han demostrado que los aceleradores de redes neuronales pulsantes tienen un gran potencial para su uso en una amplia gama de aplicaciones, como el reconocimiento de voz y de imágenes, la clasificación de datos y el control de robots. Estas aplicaciones requieren un procesamiento de datos en tiempo real y una alta eficiencia energética, lo que hace que los aceleradores de redes neuronales pulsantes sean una solución ideal.

En este artículo nos centramos en el acelerador ReckON de redes neuronales pulsantes, cuyos autores ofrecen la arquitectura en abierto (open-source) en leguaje de descripción de hardware para simulación lógica [1]. En concreto, hemos abordado la implementación para un sistema empotrado basado en sistema programable multiprocesador de Xilinx, la Zynq UltraScale+ de la placa Pynq-ZU, que permite interaccionar con el hardware a través de un fichero de comandos escrito en Python y que es ejecutado en el propio sistema empotrado a través de un servidor de Jupyter. En la siguiente sección describimos resumidamente el acelerador ReckON. En la sección III explicamos los detalles y circuitos adicionales que se han necesitado para la implementación. En la sección IV presentamos los resultados y por último las conclusiones.

II. El acelerador de SNNs ReckON

La capacidad de aprender dependencias temporales a corto y largo plazo en un hardware de SNN es uno de los elementos que faltaban para dotar de robustez a los dispositivos autónomos de vanguardia en aplicaciones como el reconocimiento de gestos, el procesamiento del habla y la robótica cognitiva [8]. Para resolver este reto, ReckON propone un procesador de red neuronal recurrente (RNN) pulsante que permite el aprendizaje no-supervisado durante segundos con una resolución temporal de milisegundos. ReckON utiliza trazas de elegibilidad (ETs) de propagación hacia adelante para una aproximación bioinspirada al algoritmo de retropropagación en el tiempo (BPTT), que es local tanto en el espacio como en el tiempo, lo que reduce drásticamente los requisitos de memoria. Este acelerado utiliza dispersión (sparsity) en los datos de entrada y actualizaciones de los pesos para minimizar el procesamiento. Y Reckon utiliza el bus AER, también usados en los sensores neuromórficos, como la retina [18] y la cóclea [19], para el procesamiento de tareas de diagnóstico. Estas características permiten demostrar el entrenamiento de datos temporales con bajo consumo y baja latencia. [8].

La figura 1 muestra un diagrama simplificado de la arquitectura del acelerador ReckON. Cuenta con

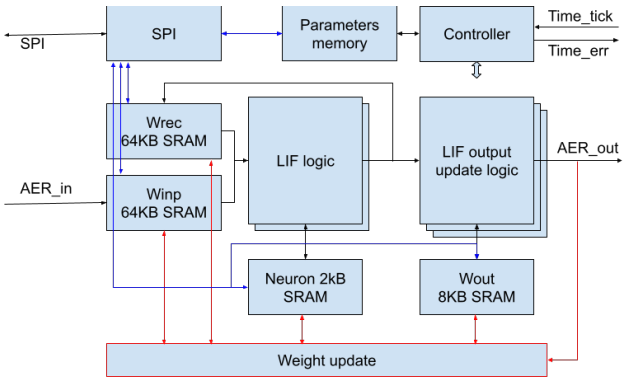


Fig. 1: Diagrama de bloques del acelerador ReckON (simplificado de [8])

entrada y salida AER pulsante y un bus de configuración SPI. La señal $Time\_tick$ le permite sincronizar los pasos de procesamiento neuronal interno con un temporizador externo, el cual está acompasado con los datos de entrada. Este acelerador require de varios bloques de memoria internos tanto para almacenar los pesos sinápticos (W_{rec}, W_{inp}, W_{out}), como el estado de las neuronas ($NeuronSRAM$). Cuenta con un mecanismo autónomo de actualización de los pesos de la red inferida ($Weight\_update$) y la lógica que emula la evolución de las neuronas integra-dispara (LIF) y su salida pulsante (LIF output).

Para su correcto funcionamiento, el acelerador require de una fase inicial de configuración, por medio del bus SPI, de los parámetros y pesos de la red a inferir. Posteriormente, diferentes trenes de pulsos son recibidos y aprendidos en una fase de entrenamiento, para posteriormente poder trabajar en fase de inferencia.

El trabajo de Frenkel e Indiveri [8] ofrece un ejemplo de simulación en Verilog, disponible en GitHub (ver apartado 1), en el que se realizan varias épocas de un entrenamiento y prueba de 50 muestras de trenes de pulsos para clasificar entre dos posibles clases usando 40 neuronas de entrada, 100 neuronas en una única capa oculta, y 2 neuronas de salida.

III. Implementación Hardware / Software

ReckON se ha implementado en un MPSoC de Xilinx, modelo Zynq Ultrascale+ XCZU5EG, para la tarjeta de prototipado Pynq ZU [2]. Este chip de Xilinx consiste en un sistema computador empotrado basado en cuatro cores ARM, junto con una FPGA en la que desplegar el circuito deseado y comunicarlo con el sistema computador. La conexión entre los ARM y la FPGA se realiza mediante el bus AXI. Los dispositivos Zynq™ UltraScale+™ MPSoC ofrecen escalabilidad con cuatro procesadores de 64 bits ARM Cortex A53, a la vez que combinan control en tiempo real con motores blandos y duros con dos procesadores ARM Cortex R5F, y un procesador de gráficos tipo GPU ARM MALI 400MP, además de la lógica programable. Estos sistemas permiten posibilidades ilimitadas para aplicaciones como 5G Wireless, sistemas avanzados de asistencia al conductor de nueva

[1] https://github.com/ChFrenkel/ReckOn/

[2] https://xilinx.github.io/PYNQ-ZU/

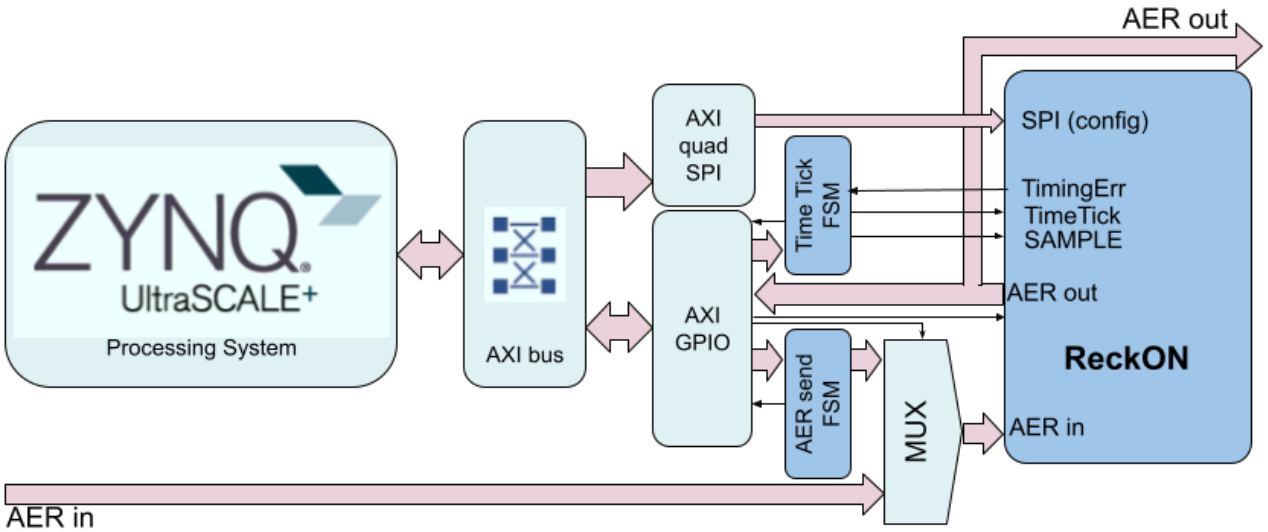


Fig. 2: Diagrama de bloques del despliegue de ReckON en Zynq Ultrascale+

generación (ADAS) e Internet de las Cosas Industrial.

Para integrar ReckON en este dispositivo se han tenido en cuenta varias cuestiones importantes. En primer lugar, el código verilog del acelerador, tomado del repositorio público (github) referenciado anteriormente, ha sido usado para el flujo de diseño de un ASIC por los creadores. Este verilog, por tanto, no incluye las memorias internas necesarias, ya que dependen de la tecnología elegida para la implementación. En segundo lugar, el circuito requiere de una señal de reloj pseudo-biológica que permite la correcta evolución tanto de la fase de aprendizaje como de inferencia de la SNN planteada ($Time\_tick$). Cuando el circuito detecta que la temporización de los eventos de entrada o del propio reloj pseudo-biológico es incorrecta, activa una señal de error ($Time\_err$), requiriendo el acelerador de una atención especial.

Se ha usado la herramienta Vivado de Xilinx, versión 2021.2, en un sistema Ubuntu 18.04.

Para abordar correctamente la implementación en FPGA se han tomado las siguientes decisiones de diseño (ver figura 2):

- La entrada de eventos AER_in de ReckON podrá venir tanto del exterior de la tarjeta como desde el procesador a través de una interfaz AXI-GPIO, de forma que la depuración y pruebas del acelerador sea más sencillo de implementar sin requerir de sistemas AER externos. La selección se hará a través de multiplexores (MUX) de control.
- Debido a que el control de la temporización requerida debe ser muy preciso, al usar la interfaz AXI-GPIO esta precisión temporal se pierde. Por ello, se han desarrollado dos máquinas de estados finitas ($Time\_tickFSM$ y $AER\_sendFSM$) que nos permiten recuperar esta precisión en los momentos requeridos. Estas máquinas se describen más adelante.
- Para realizar las pruebas del acelerador, la salida de eventos AER_out puede llegar o bien a través de la interfaz AXI-GPIO, desde el software que se esté ejecutando en los ARM, o bien desde los puertos AER de salida de la tarjeta PynqZU.
- ReckON requiere de una interfaz de configuración basada en el bus SPI para configurar tanto la red, los pesos de la misma y los diferentes parámetros de configuración de que dispone. Para este control se ha desplegado la interfaz AXI-quad-SPI de Xilinx en modo SPI básica.
- Se ha añadido un IP de Xilinx, denominado ILA (no representado en la figura por simplificidad), que actúa como un analizador lógico que nos permite conectar cuantas señales internas sean necesarias para visualizarlas en una interfaz gráfica de Vivado en modo de cronograma. Este ILA permite configurar condiciones de disparo para la captura del cronograma a depurar.

La máquina de estados $Time\_tickFSM$ recibe desde el código en ejecución en los ARM el número de pulsos o "ticks"que deben enviarse al acelerador cada vez que llegue un nuevo evento. Estos ticks representan el tiempo entre eventos y deben tener una distancia temporal precisa en el acelerador, ya que la base del aprendizaje en estos sistemas es la distancia y correlación temporal entre los eventos que llegan a las neuronas de la capa de entrada. Cuando esta FSM ha concluido su tarea, vuelve a avisar al código de los ARM (cuaderno Jupyter en nuestro caso) para que continúe con los estímulos. Al concluir la ráfaga de eventos que representa una de las clases a aprender, el sistema requiere de la activación de una señal en un instante de tiempo muy preciso, que se utiliza como señal de activación del aprendizaje (SAMPLE) y que se acompaña de un evento especial de entrada que representa la clase a aprender para la ráfaga recibida. Este control también recae en nuestra implementación en esta FSM.

Por otro lado, la máquina de estados $AER\_sendFSM$ recibe a través del AXI-GPIO el siguiente evento a enviar a ReckON y se encarga de gestionar las señales del protocolo de comunicación AER (REQ y ACK) con la mínima latencia, para evitar errores temporales ($Time\_err$).

Las memorias SRAM que require ReckON se han añadido como IP-cores de librería en la herramienta Vivado cumpliendo las restricciones que el verilog que describe a ReckON nos imponía respecto a tamaño de la memoria, número de puertos y registros de entrada / salida para estabilizar los datos.

La plataforma Pynq-ZU de Xilinx cuenta con un servidor Jupyter Notebook en el sistema operativo Petalinux ya precargado. De esta forma, conectando la tarjeta Pynq-ZU a un ordenador (usando USB), se puede abrir un navegador web en éste que abriendo el puerto 9090 de la dirección IP asiganda a la tarjeta (192.168.3.1 por defecto), nos abre el servicio Jupyter Notebook en ejecución en la Pynq-ZU. Desde este servicio se permite subir a la memoria de la Pynq los archivos de configuración de la FPGA generados en Vivado para programar y usar la FPGA desde un notebook escrito en Python. La figura 3 muestra una captura de pantalla del entorno con el comienzo de nuestro código Python. Se puede apreciar a la izquierda los archivos usados (la carpeta ̈datacontiene los archivos de pesos y de eventos), y a la derecha la secuencia de comandos que permite la carga de la FPGA (usando la librería Overlay), así como la definición del manejador del AXI-GPIO, del bus SPI a través del AXI-quad-SPI y la definición de las rutas de los archivos de pesos eventos de entrenamiento y de test.

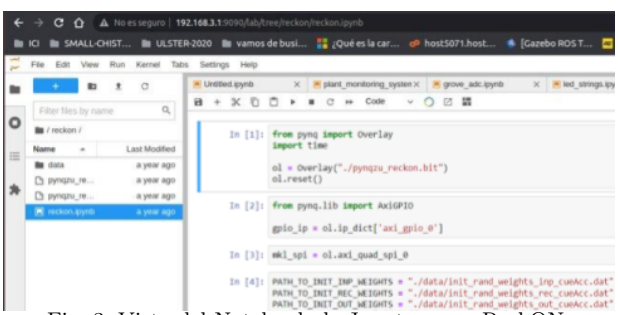


Fig. 3: Vista del Notebook de Jupyter para ReckON

IV. Resultados experimentales

Para comprobar el correcto funcionamiento del acelerador se ha trasladado la funcionalidad del fichero tbench.sv disponible en el repositorio del ReckON a un cuaderno de Jupyter. Tanto el flujo del tbench.sv como la reproducción usada en el cuaderno de Jupyter, responden al diagrama de flujo de la figura 4. En este test se ha usado una RSNN (SNN recurrente) con 40 neuronas en la capa de entrada, 100 neuronas en la capa oculta (con conexiones recurrentes a ellas mismas), y 2 neuronas en la capa de salida, como se ve en la figura 6 Tras una fase de inicialización y configuración de los parámetros y pesos del ReckON en la que se utiliza el bus SPI masivamente, pasamos a un par de bucles anidados en los que recorremos un número de épocas (epochs) en el entrenamiento para repetir los estímulos y pruebas que se tienen de forma alterna. Cada secuencia de eventos con una duración de 2250 ms cada uno comienza activando la señal SAMPLE. Cada secuencia de eventos

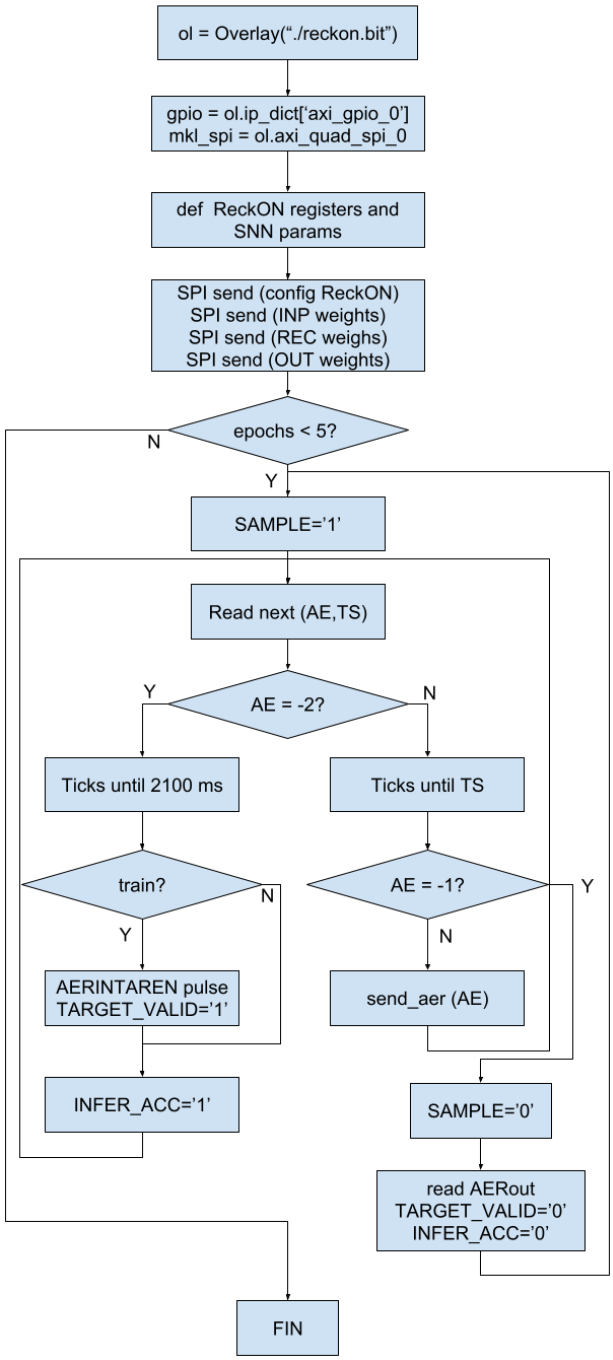


Fig. 4: Diagrama de flujo del test al ReckON

se compone de la muestra de eventos (AE,TS), siendo AE la dirección del evento a escribir en AER_in y TS el tiempo de espera entre eventos consecutivos. Existen dos tipos de eventos especiales: el -2, que nos indica en su TS el valor del AE que se debe aprender para la actual secuencia de eventos, cuando se trata de una muestra de entrenamiento. Este -2, cuando se trata de una muestra de test, indica la salida esperada de la SNN configurada en ReckON, lo cual nos servirá para comprobar posteriormente, al recibir el AER_out, si la red acertó o no. Este evento especial -2 obliga a activar, en entrenamiento, una señal de aprendizaje (TARGET_VALID) hasta el final de la secuencia, así como INFER_ACC, y un pulso en la señal AERINTAREN, como puede verse en el dia-

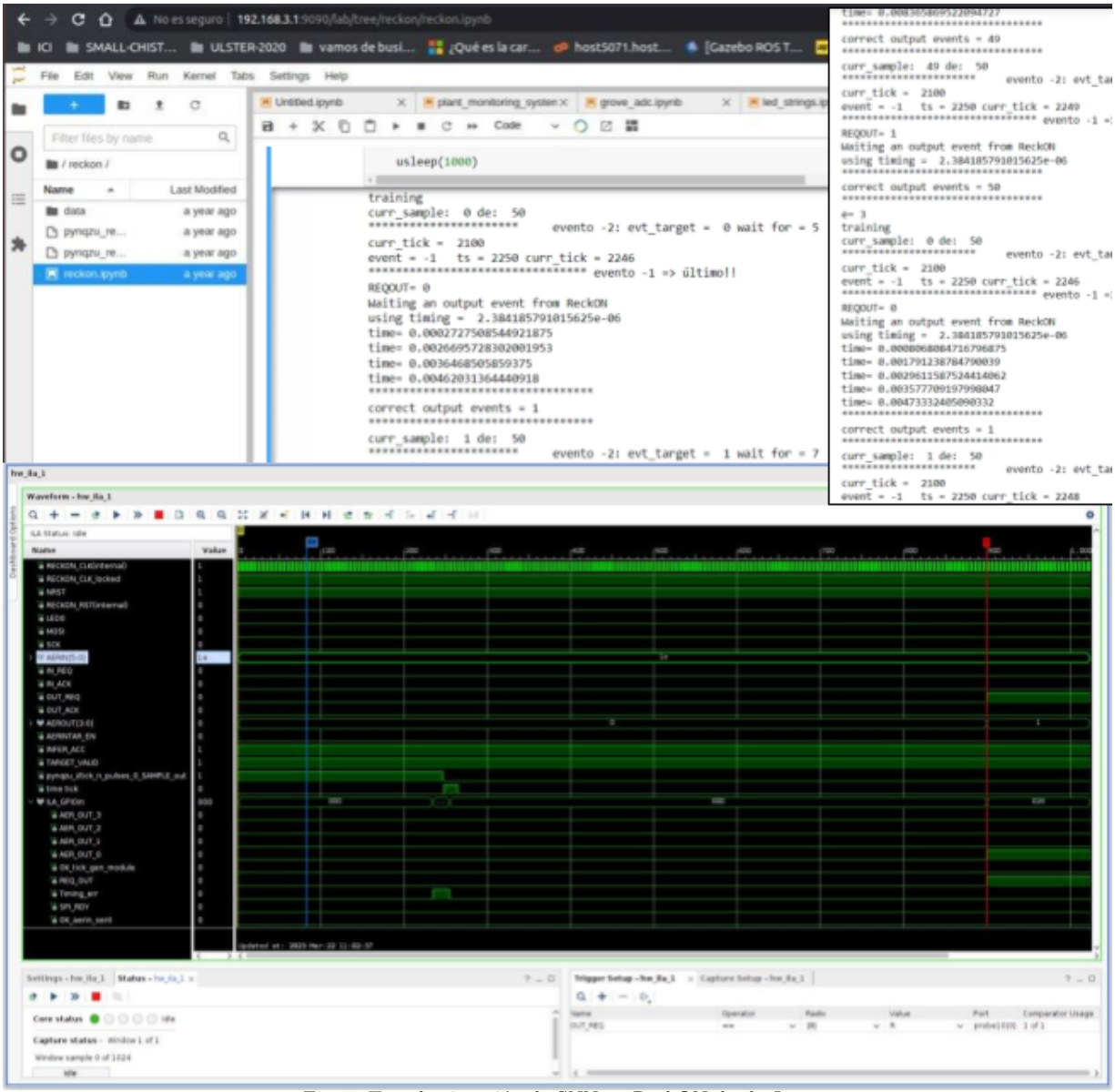


Fig. 5: Test de ejecución de SNN en ReckON desde Jupyter

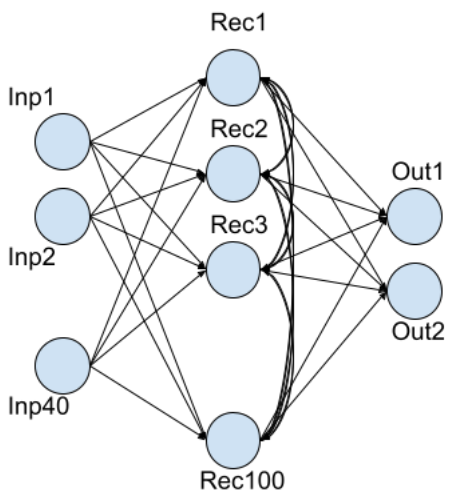


Fig. 6: Test de ejecución de SNN en ReckON desde Jupyter

grama de flujo. Cuando estamos en fase de test, sólo se activa la señal INFER_ACC hasta el final de la secuencia. Cada vez que una muestra se ha procesado (han pasado los 2250 ms), se debe desactivar la señal SAMPLE y esperar el envío de un evento de salida de ReckON indicando la clase ganadora. Para comenzar a procesar una nueva secuencia de eventos hay que volver a activar la señal SAMPLE. Tras finalizar el número de épocas configurado, el test habrá concluido. Las secuencias de eventos se leen de los archivos 'dataset_cueAcc_train.dat' y 'dataset_cueAcc_test.dat'. El primer dato de estos archivos indica el número de muestras (secuencias de eventos) que contiene el fichero. En este test se trata de 50 secuencias de entrenamiento y 50 de test.

En la figura 5 puede verse la salida impresa del test ejecutado en el cuaderno de Jupyter, así como una captura del cronograma capturado por Vivado con el ILA que contiene el diseño. El comienzo de la salida impresa del testbench muestra el flujo para la primera secuencia. Puede observarse cómo tras una serie de eventos AE se lee del fichero .dat un evento -2, cuya AE esperada es 0. Posteriormente, al cumplirse el tiempo de 2250ms se obtiene del fi-

Resource	Utilization	Available	Utilization %
LUT	40058	117120	34.20
LUTRAM	442	57600	0.77
FF	8814	234240	3.76
BRAM	46	144	31.94
DSP	144	1248	11.54
IO	20	252	7.94
BUFG	8	352	2.27
PLL	1	8	12.50

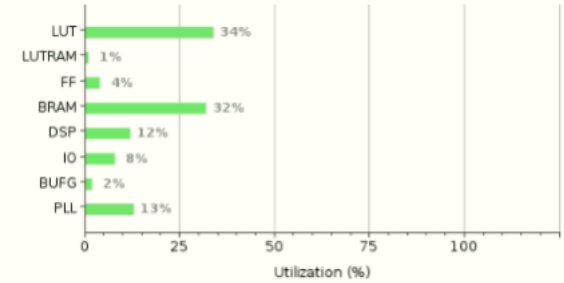


Fig. 7: Recursos de la Zynq Ultrascale+ consumidos por RecKON

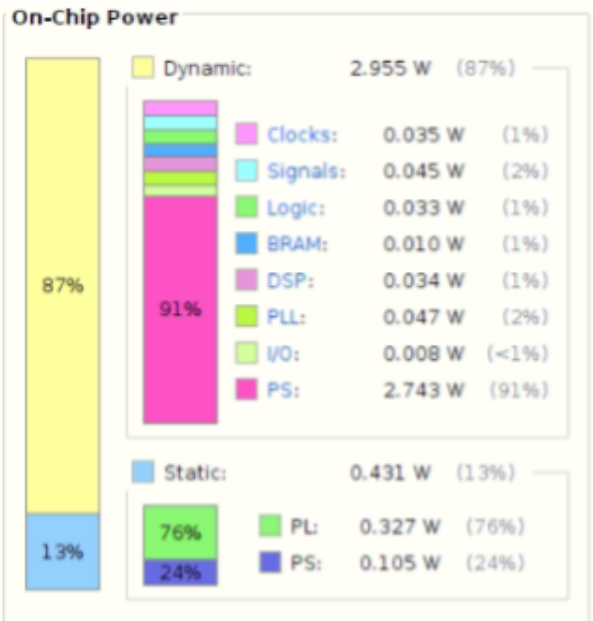


Fig. 8: Consumo energético de ReckON en la Zynq Ultrascale+

chero el evento -1, pasando al estado de recepción de eventos. Al recibirse el evento se comprueba que su dirección coincide con la esperada y se pasa a la siguiente muestra de eventos. Se ha añadido en la figura, en la parte derecha el final de la época 2 y comienzo de la época 3. Puede comprobarse que la tasa de acierto de la SNN probada es del 100 %, pues se han acumulado 50 eventos de salida idénticos a los esperados.

El ReckON, tras sintetizarlo en la FPGA de nuestra Zynq Ultrascale+ ha requerido algo más del 30 % de los recursos LUT y BRAM del chip. Se muestra un resumen de la ocupación en la figura 7. Por otro lado, se ha estimado el consumo de potencia del MPSoC usando la herramienta Vivado, resultando en algo más de 200mW el consumo dinámico del circuito en la parte FPGA del chip. El consumo total, incluyendo el sistema computador empotrado, no llega a los 3,4W. El resumen de la estimación de consumo se aprecia en la figura 8

V. Conclusiones

En este trabajo se ha implementado, para el sistema multiprocesador con lógica programable de Xilinx Zynq UltraScale+ de la tarjeta Pynq-ZU, el acelerador de redes neuronales pulsantes, ReckON, descrito en verilog para ASIC y disponible en un repositorio público. Se presentan las diferentes técnicas y circuitos empleados para esta implementación y su posible uso desde un servidor de cuadernos de Jupyter para utilizar el acelerador desde un código escrito en Python. Las pruebas realizadas corresponden con el ejemplo de test hecho en verilog por los creadores, comprobándose el correcto funcionamiento de la prueba. El consumo dinámico del sistema está entorno a los 200mW y los recursos necesarios del chip de Xilinx alrededor del 30 %. El trabajo futuro consiste en utilizar este acelerador para clasificar otras informaciones pulsantes utilizando redes SNN con diferente número de neuronas en las capas de entrada, oculta y de salida, estableciendo las reglas de conexionado recurrente necesarias y modificando los pesos a través de herramientas de simulación de SNNs, como son snnTorch o Bindsnet. Esta implementación en FPGA abre la puerta para futuras aplicaciones de EdgeAI neuromórficas de un bajo consumo y baja latencia.

Agradecimientos

Esta trabajo ha sido parcialmente soportado por el proyecto MINDROB (PID2019-105556GB-C33) financiado por MCIN/AEI/10.13039/501100011033, y CHIST-ERA H2020 grant SMALL (PCI2019-111841-2), financiado por MCIN/AEI/10.13039/501100011033. Los autores agradecen la colaboración con los equipos de Charlotte Frenkel de la TU Delft y de Robert Leingenstein de la TU Graz.

Referencias

[1] M. Rabinovich, R. Huerta, and V. Afraimovich, "Dynamics of sequential decision making," *Physical Review Letters*, vol. 97, 2006.

[2] Wolfgang Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[3] Peter U. Diehl, Guido Zarrella, Andrew Cassidy, Bruno U. Pedroni, and Emre Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, 2016, pp. 1–8.

[4] Jun Sawada, Filipp Akopyan, Andrew S Cassidy, Brian Taba, Michael V Debole, Pallab Datta, Rodrigo Alvarez-Icaza, Arnon Amir, John V Arthur, Alexander Andreopoulos, Rathinakumar Appuswamy, Heinz Baier, Davis Barch, David J Berg, Carmelo Di Nolfo, Steven K Esser, Myron Flickner, Thomas A Horvath, Bryan L Jackson, Jeff Kusnitz, Scott Lekuch, Michael Mastro, Timothy Melano, Paul A Merolla, Steven E Millman, Tapan K Nayak, Norm Pass, Hartmut E Penner, William P Risk, Kai Schleupen, Benjamin Shaw, Hayley Wu, Brian Giera, Adam T Moody, Nathan Mundhenk, Brian C Van Essen, Eric X Wang, David P Widemann, Qing Wu, William E Murphy, Jamie K Infantolino, James A Ross, Dale R Shires, Manuel M Vindiola, Raju Namburu, and Dharmendra S Modha, "TrueNorth ecosystem for brain-inspired computing: scalable systems, software, and applications,"

in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 130–141.

[5] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *Biomedical Circuits and Systems, IEEE Transactions on*, vol. 12, no. 1, pp. 106–122, Feb. 2018.

[6] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, J Bussat, R Alvarez-Icaza, JV Arthur, PA Merolla, and K Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[7] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[8] Charlotte Frenkel and Giacomo Indiveri, "Reckon: A 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales," in *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, 2022, vol. 65, pp. 1–3.

[9] Kit Cheung, Simon R. Schultz, and Wayne Luk, "A large-scale spiking neural network accelerator for fpga systems," in *Artificial Neural Networks and Machine Learning – ICANN 2012*, Alessandro E. P. Villa, Włodzisław Duch, Péter Érdi, Francesco Masulli, and Günther Palm, Eds., Berlin, Heidelberg, 2012, pp. 113–120, Springer Berlin Heidelberg.

[10] Alireza Khodamoradi, Kristof Denolf, and Ryan Kastner, "S2n2: A fpga accelerator for streaming spiking neural networks," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, New York, NY, USA, 2021, FPGA '21, p. 194–205, Association for Computing Machinery.

[11] M.M. Khan, D.R. Lester, L.A. Plana, A. Rast, X. Jin, E. Painkras, and S.B. Furber, "Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008, pp. 2849–2856.

[12] "The address-event representation communication protocol AER 0.02," Caltech internal memo, Feb. 1993.

[13] Eric Hunsberger and Chris Eliasmith, "Training spiking deep networks for neuromorphic hardware," 2016.

[14] Hananel Hazan, Daniel J. Saunders, Hassaan Khan, Devdhar Patel, Darpan T. Sanghavi, Hava T. Siegelmann, and Robert Kozma, "Bindsnet: A machine learning-oriented spiking neural networks library in python," *Frontiers in Neuroinformatics*, vol. 12, 2018.

[15] Sumit Bam Shrestha and Garrick Orchard, "SLAYER: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., pp. 1419–1428. Curran Associates, Inc., 2018.

[16] Jason K Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu, "Training spiking neural networks using lessons from deep learning," *arXiv preprint arXiv:2109.12894*, 2021.

[17] Christian Pehle and Jens Egholm Pedersen, "Norse - A deep learning library for spiking neural networks," Jan. 2021, Documentation: https://norse.ai/docs/.

[18] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128 × 128 1.5 % contrast sensitivity 0.9 % fpn 3 s latency 4 mw asynchronous framefree dynamic vision sensor using transimpedance preamplifiers," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 3, pp. 827–38, 2013.

[19] A. Jimenez-Fernández et al., "A binaural neuromorphic auditory sensor for fpga: A spike signal processing approach," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 4, pp. 804–18, 2017.

Minimizacion paralela de modelos parametricos a partir de nubes de puntos

Nahuel E. Garcia-D'Urso[1], Jorge Azorin-Lopez[1], Andres Fuster-Guillo[1] y Akira Llorens Montero [1]

Resumen— **En este trabajo se describe un algoritmo para la obtención de modelos paramétricos 3D de forma paralela a partir de nubes de puntos. La estimación de los modelos paramétricos se realiza mediante un proceso de minimización de la posición y la forma del cuerpo. La función de minimización se basa en cálculos de la distancia entre puntos. Los puntos son seleccionados utilizando el algoritmo de los vecinos mas cercanos y el ángulo entre las normales. Además, el uso de un modelo plantilla, junto a la paralelización del método descrito, ayuda a acelerar y mejorar la precisión del modelo paramétrico obtenido. Además, el uso de técnicas de paralelización hace que el método mejores en un 124 %. El método ha sido probado en tres datasets diferentes.**

Palabras clave— **Modelo paramétrico, modelo 3D del cuerpo humano, sensores RGB-D, paralelización**

I. Introducción

La estimación de un modelo 3D paramétrico del cuerpo humano, como SCAPE [1], SMPL [2] o sus variantes, se basa en la estimación de un conjunto de variables que definen la postura y la forma del cuerpo humano. Aunque existen numerosos artículos que estiman un modelo paramétrico 3D a partir de imágenes 2D, hay pocos que lo hagan a partir de un modelo 3D obtenido de uno o varios sensores RGB-D. Esto puede deberse a la escasez de conjuntos de datos reales del cuerpo humano en 3D [3], [4], aunque hoy en día hay conjuntos de datos artificiales disponibles, como SURREAL [5], GHS3D [6], UP [7] o MPII Human Shape [8]. Esto podría deberse al hecho de que los modelos 3D, obtenidos de cámaras RGB-D, a veces no están completos. Zonas como la de las axilas o entrepierna puedes no disponer de puntos debido a oclusiones. Además, se requiere un procesamiento significativo para que estos modelos 3D se puedan utilizar en otras tareas, como la automatización de mediciones antropométricas o su incorporación en entornos de desarrollo de videojuegos. Pata abordar estos problemas, proponemos un método basado en la minimización de los parámetros SMPL. El método descrito en este trabajo se ha realizado utilizando técnicas de paralelización, lo cual permite la obtención de los modelos paramétricos de forma más rápida.

En contraste, los modelos paramétricos 3D del cuerpo humano ofrecen una utilización conveniente en diversas aplicaciones. Los modelos paramétricos permiten la implementación de algoritmos automatizados debido a que los índices en la nube de puntos están ordenados siempre igual para todos los modelos. Como resultado, también se utilizan para tareas de minería de datos [9]. Los modelos paramétricos también se utilizan para llenar de puntos las zonas donde las cámaras RGB-D no han podido obtener datos. También son utilizados en la industria del textil [10], [11].

Hoy en día, el industria que más se beneficia del uso de modelos paramétricos es la industria textil. En este área los modelos paramétricos permiten que las personas se puedan probar diferentes prendas de ropa sin tener que comprarlas. Además, modelos paramétricos como SMPL o STAR permiten una gran definición y precisión lo que conlleva a que las personas además pueden comprobar si la prenda de ropa se adecua con su cuerpo [12], [13]. Por otro lado, estos modelos permiten la creación de avatares virtuales, generación de entornos con personas [14]. Otra de las áreas donde más se están utilizando estos modelos es en la Realidad Virtual y Aumentada.

En este trabajo, proponemos un método para generar modelos 3D del cuerpo humano a partir de escaneos obtenidos con cámaras RGB-D de forma paralela. El método consta de 4 fases como se muestra en la Figura 1. En la fase inicial se realiza la adquisición del modelo 3D utilizando el sistema desarrollado en el proyecto Tech4Diet[1]. En este proyecto se cuenta con una cabina equipada con 13 cámaras RGB-D Intel RealSense [15]. También se realiza un preprocesamiento de las nubes de puntos. En la segunda fase se utiliza la red neuronal BPS [16] para obtener un template que facilite la minimización posterior. En la tercera fase se minimizan los parámetros del modelo paramétrico SMPL [**?**] con respecto al template obtenido en la fase anterior. Esta fase se utiliza para tener un punto de partida en la última parte del método. En esta última parte se realiza la minimización del modelo SMPL utilizando ya el escaneo 3D obtenemos con las cámaras.

Nuestras principales contribuciones son: (1) Proponemos un pipeline para obtener modelos 3D paramétricos del cuerpo humano a partir de escaneos capturados con cámaras RGB-D. (2) Un método capaz de mejorar las áreas en el escaneo donde falta información. (3) Estimación rápida gracias al uso de templates y paralelización.

II. Trabajo Relacionado

En los últimos años, ha habido un creciente interés en la estimación de modelos paramétricos 3D del cuerpo humano a partir de diferentes inputs. Una

[1]Departamento de Tecnología Informática y Computación, Universidad de Alicante, España, e-mail: {nahuel.garcia, jazorin, fuster, akira}@ua.es.

[1]https://tech4d.ua.es/

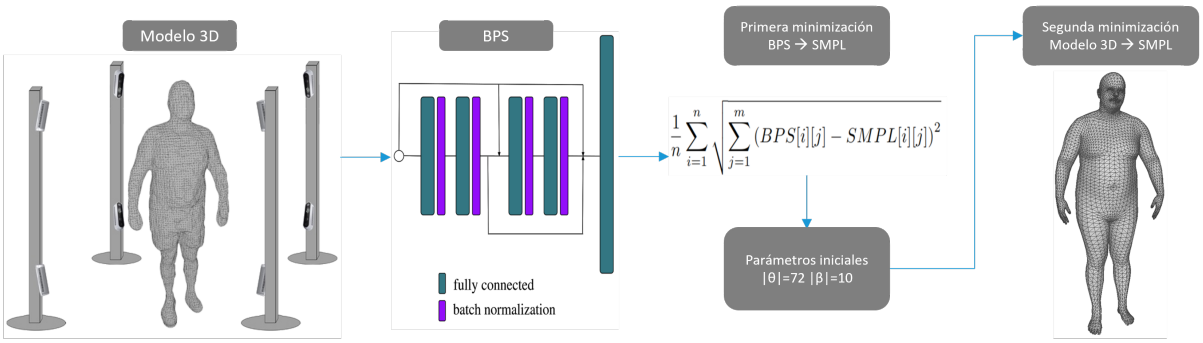


Fig. 1: Pipeline de la estimación del modelo paramétrico 3D del cuerpo humano.

de las razones de esto es que los modelos paramétricos del cuerpo humano son cada vez más precisos y realistas. El modelo paramétrico SCAPE representó un gran avance en esta área y posteriormente, el modelo paramétrico SMPL se convirtió en el más utilizado debido a su alta precisión en la forma y postura del cuerpo humano. En la actualidad, se pueden encontrar trabajos en los que se mejoran estos dos modelos paramétricos mencionados. SMPL-X [17] incorpora la posibilidad de articular y deformar tanto la mano como la cara de la persona. Además, el modelo STAR [18] representa una mejora sobre SMPL, ya que este modelo paramétrico permite deformaciones corporales más realistas gracias al aprendizaje de formas de mezcla correctivas.

Estas mejoras en los modelos paramétricos ha llevado a un aumento en el número de trabajos enfocados en estimar o generar modelos paramétricos 3D del cuerpo humano a partir de imágenes 2D o 3D. En particular, Bogo et al. [19] fueron los primeros en reconstruir con éxito un cuerpo humano de alta definición a partir de una sola imagen utilizando el modelo paramétrico SMPL. A partir de este enfoque, SMPL-X [17] predice características de imágenes 2D utilizando OpenPose [20] y posteriormente minimizando el modelo SMPL-X a estas características.

Los avances recientes en los modelos paramétricos han llevado a un aumento en el número de trabajos enfocados en estimar o generar modelos 3D del cuerpo humano a partir de imágenes o escaneos obtenidos con cámaras RGB-D. En particular, Bogo et al. [19] fueron los primeros en reconstruir con éxito un cuerpo humano de alta definición a partir de una sola imagen utilizando el modelo paramétrico SMPL. A partir de este enfoque, SMPL-X [17] predice características de imágenes 2D utilizando OpenPose [20] y posteriormente ajusta el modelo SMPL-X a estas características. Otros métodos utilizan redes neuronales para lograr reconstrucciones del cuerpo humano. Un ejemplo de esto se presenta en [21], donde los autores utilizaron una combinación de una red neuronal de regresión profunda y una rutina de optimización iterativa. El trabajo presentado en [22] utiliza imágenes como entradas para predecir los parámetros del modelo corporal SMPL. Para lograr esto, los autores utilizaron una red neuronal convolucional (CNN) seguida de un encoder y un regresor.

En cuanto a los trabajos donde se reconstruye un cuerpo humano a partir de una imagen 3D como entrada, muchos de ellos obtienen los parámetros del modelo realizando una regresión sobre la nube de puntos. En el trabajo presentado en [23], los autores proponen un método en el que aprenden las transformaciones bidireccionales entre el espacio de pose y el espacio canónico. Otro trabajo notable es el de [10]. Su método se basa en la minimización punto a punto del escaneo con el modelo SMPL. Además, demuestran su eficacia comparando las medidas realizadas en el modelo obtenido con las medidas antropométricas reales del cuerpo.

III. Metodología

En esta sección presentamos el método y materiales que hemos utilizado para lograr los objetivos mencionados anteriormente. En primer lugar, explicaremos el sistema de adquisición utilizado y el preprocesamiento que se ha aplicado al modelo 3D para la generación del modelo paramétrico. En segundo lugar, detallaremos el paso intermedio donde se genera un template 3D utilizando el método BPS. Luego, explicaremos la obtención de los parámetros iniciales a partir de la minimización realizada utilizando el template. Finalmente, explicaremos la minimización final donde se obtienen los parámetros que generan un modelo paramétrico con la misma forma y pose que el modelo original.

A. Obtención y Preprocesado del Modelo 3D.

Para la reconstrucción del cuerpo humano, la persona debe estar posicionada en el centro de una cabina equipada con 13 sensores Intel Real-Sense D435, como se muestra en la Figura 2. La persona es capturada por todas las cámaras de forma síncrona, reduciendo los errores debido al movimiento entre frames.

Después de obtener las nubes de puntos, se aplican varios filtros para reducir el ruido (filtro mediano, filtro bilateral y eliminación estadística de valores atípicos). Después de aplicar los diferentes filtros, se realiza un registro rígido de las nubes de puntos para unificarlas. Para ello, se utilizan las matrices de transformación obtenidas al realizar una calibración extrínseca del sistema de cámaras. Además, dado que el individuo debe estar parado sobre un escalón, se calcula un plano a nivel de los pies para eliminar

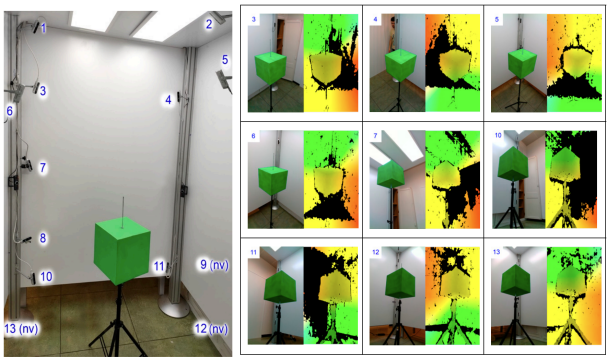


Fig. 2: Cabina con 13 sensores RGB-D dispuestos en cuatro postes de aluminio a diferentes alturas, "nv"significa cámara no visible en esta imagen (izquierda). Ejemplo de adquisición de imagen de color y profundidad (derecha).

todos los puntos debajo de él.

B. *Estimación de una plantilla intermedia utilizando una red neuronal BPS.*

Después de obtener y preprocesar el escaneo 3D del cuerpo humano, se genera un template intermedio utilizando el método propuesto por [16]. Basis Point Sets (BPS) es una técnica que codifica un conjunto de puntos en representaciones de distancia fija. Esto implica seleccionar k puntos fijos en el espacio y calcular los vectores desde cada punto hasta el punto más cercano en la nube de puntos. Estos puntos fijos permanecen consistentes para cualquier input. El vector de distancia resultante se utiliza como entrada para una red neuronal DenseNet [24], que consta de dos bloques de capas completamente conectadas. La salida de la red es la predicción de las posiciones de los vértices de un template similar a la nube de puntos de entrada.

C. *Primera Minimización: BPS a SMPL*

Dado que las personas escaneadas con el sistema de cámaras RGB-D se encuentran siempre en una posición de .ª", con los brazos hacia abajo y un poco separados del cuerpo, se ha inicializado el modelo SMPL con esa misma posición. Esto se hace con fin de obtener resultados más rápidos y óptimos.

En cuanto a la función de minimización 1 utilizada en esta parte del pipeline, es la distancia euclídea promedio entre cada vértice del template y el modelo SMPL.

$$\frac{1}{n}\sum_{i=1}^{n}\sqrt{\sum_{j=1}^{m}(BPS_{i,j}-SMPL_{i,j})^2} \qquad (1)$$

D. *Segunda Minimización*

Una vez realizada la minimización inicial y obtenidos los parámetros de pose y forma intermedios, se realiza la minimización con respecto al modelo obtenido de los sensores RGB-D. En este caso, al minimizar utilizando un escaneo en lugar de un template, los modelos no tienen el mismo número de puntos. Además, cada escaneo tiene un número diferente de vértices. Por lo tanto, antes de usar la ecuación 1, se deben seleccionar los puntos correspondientes en el modelo SMPL para cada punto en el modelo escaneo, habiendo repetición. Para lograr esto, se sigue el siguiente proceso:

En primer lugar, se definen las diferentes partes del cuerpo tanto para el modelo SMPL como para el modelo escaneado. Este paso se realiza para mejorar la velocidad del algoritmo en los pasos posteriores. Luego, se realiza un registro rígido para alinearlos en la misma posición, utilizando el método Iterative Closest Point (ICP). Para cada punto en el modelo escaneado que pertenece a una parte del cuerpo (cadera, cintura, mano derecha, ...), se calculan los cinco vecinos más cercanos en el modelo SMPL. Esta búsqueda solo se realiza entre puntos que pertenecen a la misma parte del cuerpo tanto en el modelo SMPL como en el escaneado, evitando una búsqueda en todo el cuerpo cada vez. Se elige el punto válido como aquel con el valor más pequeño entre la distancia y el ángulo de las normales, utilizando la siguiente ecuación 2. En esta ecuación, v_1 representa un punto en el escaneo 3D, $v_2^{1,2,3,4,5}$ se refiere a los cinco vecinos más cercanos, $(\hat{v}1\cdot\hat{v}2^{1,2,3,4,5})$ se refiere a la multiplicación entre los vectores unitarios de los puntos mencionados y $\perp_{-1}^{\top 1}$ delimita los valores entre -1 y 1 cuando el producto escalar de los vectores unitarios es mayor a 1 o menor a 1 respectivamente. Finalmente, se calcula la distancia euclídea promedio, como se hizo en el paso anterior (ecuación 1), entre los puntos que obtuvieron los mejores resultados y los puntos correspondientes en el modelo escaneado.

$$\text{mín}(\sqrt{(v_1-v_2^{1,2,3,4,5})^2}\cdot 2\cdot|\arccos((\hat{v}_1\cdot\hat{v}_2^{1,2,3,4,5})\perp_{-1}^{\top 1})|) \qquad (2)$$

IV. Paralelización

Se ha aplicado paralelización tanto en la función que asigna las partes del cuerpo en el modelo escaneado como en la función objetivo que se aplica durante la minimización.

En lo que se refiere a la paralelización de la función de asignación de partes se ha utilizado una paralelización mediante CPU. En este caso cada proceso aplica el algoritmo para cada punto y el resultado es almacenado en un *array* que contiene el id del punto y la parte del cuerpo que representa.

Sobre la función objetivo, 1, se ha aplicado paralelización mediante GPU para el calculo de las distancias. En este caso, cada hilo se encarga de calcular la distancia euclídea para un solo punto.

V. Experimentación

En esta sección, presentamos la evaluación de nuestro método de estimación de modelo paramétrico de cuerpo humano en 3D. En primer lugar, describimos los conjuntos de datos y las métricas utilizadas, y luego presentamos los resultados obtenidos.

A. *Conjuntos de datos y métricas de evaluación*

Hemos realizado la adquisición de 8 cuerpos utilizando el sistema de cámara RGBD-D descrito en

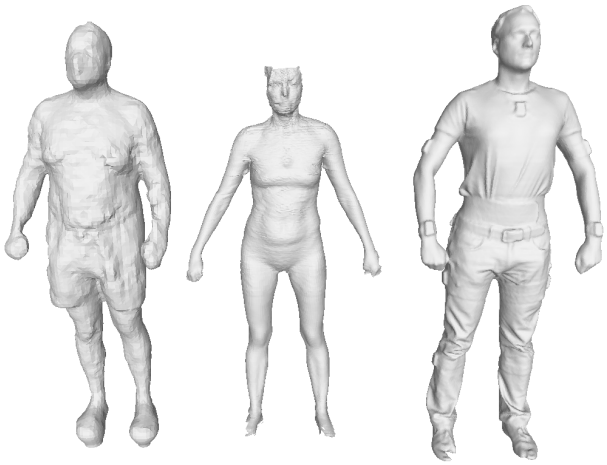


Fig. 3: Ejemplo de modelos 3D provenientes de los conjuntos Tech4Diet, NOMO3D y TNT15.

[15]. Además, hemos analizado los resultados en dos conjuntos de datos públicos, TNT15 [25] y NOMO3D [10]. El primer conjunto de datos consiste en un conjunto de datos (datos de video, siluetas, datos de IMU, matrices de proyección y mallas) obtenidos con 8 cámaras RGB y 10 IMU. El último conjunto de datos consta de 400 escaneos obtenidos con el sensor TC2, que utiliza los sensores de profundidad Intel RealSense R200.

Hemos evaluado la estimación del modelo de cuerpo 3D utilizando varias métricas. Dado una escaneo 3D de referencia obtenido con un sistema de cámara RGB-D y una estimación, hemos calculado la distancia de Chamfer 3, que es la suma de las distancias al cuadrado entre las correspondencias de los vecinos más cercanos, y la distancia de Hausdorff 4, que es la mayor de todas las distancias euclídeas desde un punto en una nube de puntos hasta el punto más cercano en la otra nube de puntos. En la Figura V-D, se muestran los resultados visuales de las estimaciones de cuerpo obtenidas.

$$Chamfer(X,Y) = \sum_{x \in X} \min_{y \in Y} ||x-y||_2^2 + \sum_{y \in Y} \min_{x \in X} ||x-y||_2^2 \tag{3}$$

$$Hausdorff(X,Y) = \max \left\{ \sup_{x \in X} d(x,Y), \sup_{y \in Y} d(X,y) \right\} \tag{4}$$

Se ha utilizado como algoritmo de minimización L-BFGS-B. Para la primera minimización se ha establecido un nivel de tolerancia de 0,001 y un máximo de iteraciones de 100. Para la segunda minimización se ha utilizado un número máximo de iteraciones de 25. En este caso, el proceso de minimización de los parámetros SMPL termina cuando la distancia euclídea entre las nubes de puntos en menor a 9 cm. Si al terminal la minimización no se alcanza ese valor, se vuelve a minimizar partiendo de los valores obtenidos en la anterior minimización. El número máximo de iteraciones globales es de 20. Esto se realiza porque cada vez que termina una minimización se vuelven a calcular los mejores puntos utilizando las diferentes partes del cuerpo, como está explicado en 2.

Para la paralelización de los algoritmos se han utilizado las librerías de Numba [26] y Joblib [27]. La CPU utilizada es la Intel i7-10750H con 12 núcleos y 5.0 GHz. Mientras que la GPU utilizada es la NVIDIA GeForce GTX 1660 Ti Mobile.

B. Resultados

Hemos evaluado nuestro método de estimación de modelos 3D paramétricos en un total de 32 escaneos 3D. Los modelos utilizados varían significativamente en calidad, desde escaneos de alta calidad hasta aquellos con datos faltantes en ciertas áreas, como se ilustra en la Figura 3.

C. Resultados Cuantitativos

Los resultados obtenidos aplicando el método de estimación paramétrico propuesto en este trabajo se muestran en la Tabla I. Al observar los resultados en la métrica de distancia Chamfer, se puede ver que las estimaciones del modelo paramétrico en el conjunto de datos TNT15 arrojan los mejores resultados, con una distancia media de 2,23 cm. Aunque también se obtienen buenos resultados en los otros dos conjuntos de datos, con ambos cercanos a una distancia media de 2,5 cm. En cuanto a la distancia de Hausdorff, el conjunto de datos TNT15 también arroja los mejores resultados. En este caso, se obtienen resultados más pobres en los conjuntos de datos NOMO3D y T4D. Se puede observar que para el conjunto de datos TNT15, se obtienen los peores resultados al estimar modelos 3D de mujeres. Una de las razones de esto es que es difícil para BPS y nuestro método capturar con precisión la región de la cabeza cuando se trata de cabello largo.

En lo que se refiere a la mejora de tiempo obtenido, se han realizado múltiples ejecuciones del método sobre uno de los modelos del dataset de T4D. En la tabla II se puede observar como el uso de técnicas de paralelización, ya sea con CPU o GPU, disminuye el tiempo de ejecución de este método.

D. Resultados Cualitativos

En la figura 4 se pueden observar tres estimaciones del modelo SMPL realizadas utilizando el método descrito en este trabajo. En las imágenes ubicadas en el centro y la derecha, se puede ver el modelo estimado sobre la nube de puntos de la escaneo de entrada.

VI. Conclusiones

Presentamos un método para estimar modelos paramétricos 3D basado en la minimización de parámetros utilizando una función que selecciona los mejores puntos para calcular la distancia entre dos nubes de puntos. Además, nuestro método se basa en el uso de la red neuronal BPS para obtener parámetros intermedios que facilitan la minimización final. Como

Tabla I: Rendimiento del método de estimación para los conjuntos de datos T4D, TNT15 y NOMO3D. Las métricas se expresan en centímetros.

Dataset	**Nº de Modelos**		**Distancia Chamfer**			**Distancia Hausdorff**		
	Mujer	Hombre	Mujer	Hombre	Media	Mujer	Hombre	Media
T4D	0	8	0	2.73	2.73	0	12.35	12.35
TNT15	2	2	2.65	2.51	2.58	8.60	6.11	7.35
NOMO3D	10	10	2.67	2.23	2.45	9.96	10.13	10.04

Tabla II: Comparativa de tiempos obtenidos en cada una de las secciones del código que se han paralelizado. Todos los tiempos están en segundos.

		Tiempo medio por iteración
Primera Minimización	Paralelizado	14.4 ± 0.052
	Sin paralelizar	26.6 ± 1.81
Segunda Minimización	Paralelizado	199 ± 1.99
	Sin paralelizar	350 ± 13.4
Selección de las partes del cuerpo	Paralelizado	5.31 ± 0.14
	Sin paralelizar	39.1 ± 1.02

se muestra en la sección V, hemos realizado la estimación de 32 cuerpos escaneados pertenecientes a tres conjuntos de datos diferentes (T4D, NOMO3D y TNT15). Las diferentes métricas calculadas muestran que los modelos estimados tienen una gran similitud con la realidad. Además, el haber aplicado técnicas de paralelización ha permitido obtener tiempos de ejecución mucho más bajos que si se ejecutara de forma no paralela.

Nuestro método abre varias direcciones para trabajos futuros. Los modelos paramétricos 3D del cuerpo humano nos permiten obtener medidas antropométricas automáticamente o utilizarlos en diferentes motores de juegos, como Unity. Dado que hoy en día existen muchos trabajos que obtienen un modelo paramétrico a partir de imágenes 2D, sería interesante comparar nuestro método con estos métodos.

Agradecimientos

Este trabajo fue apoyado por la Agencia Estatal de Investigación (AEI) de España bajo la subvención PID2020-119144RB-I00 financiada por MCIN / AEI / 10.13039 / 501100011033, y también ha sido desarrollado con el apoyo de valgrAI - Valencian Graduate School and Research Network of Artificial Intelligence de la Comunidad Valenciana y la Generalitat Valenciana, cofinanciado por la Unión Europea.

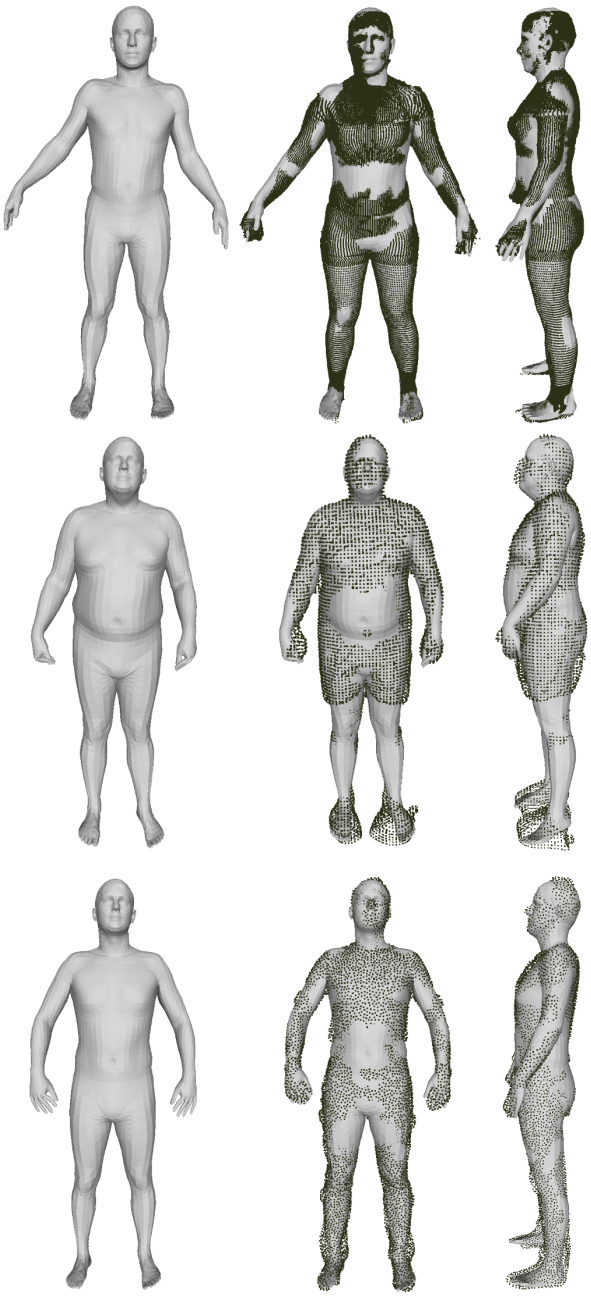


Fig. 4: En la parte izquierda se muestra el modelo SMPL estimado. En el centro y a la derecha se muestra el modelo SMPL estimado con puntos de referencia de la realidad (Figura 3). "De arriba abajo, podemos ver un modelo estimado de cada uno de los conjuntos de datos utilizados: NOMO3D, T4D y TNT15, respectivamente".

Referencias

[1] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis, "Scape: Shape completion and animation of people," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 408–416, jul 2005.

[2] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black, "SMPL: A skinned multi-person linear model," *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, vol. 34, no. 6, pp. 248:1–248:16, Oct. 2015.

[3] Vincent Leroy, Jean-Sebastien Franco, and Edmond Boyer, "Multi-view dynamic shape refinement using local temporal integration," 10 2017, pp. 3113–3122.

[4] Jinlong Yang, Jean-Sébastien Franco, Franck Hétroy-Wheeler, and Stefanie Wuhrer, "Estimation of human body shape in motion with wide clothing," in *European Conference on Computer Vision*, 2016.

[5] Gül Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J. Black, Ivan Laptev, and Cordelia Schmid, "Learning from synthetic humans," in *CVPR*, 2017.

[6] Hongyi Xu, Eduard Gabriel Bazavan, Andrei Zanfir, Bill

Freeman, Rahul Sukthankar, and Cristian Sminchisescu, "Ghum & ghuml: Generative 3d human shape and articulated pose models," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (Oral)*, 2020, pp. 6184–6193.

[7] Christoph Lassner, Javier Romero, Martin Kiefel, Federica Bogo, Michael J. Black, and Peter V. Gehler, "Unite the people: Closing the loop between 3d and 2d human representations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, July 2017.

[8] Leonid Pishchulin, Stefanie Wuhrer, Thomas Helten, Christian Theobalt, and Bernt Schiele, "Building statistical shape spaces for 3d human modeling," *Pattern Recognition*, 2017.

[9] Péter Tamás and Marianna Halasz, "Human body measuring and 3d modelling," 01 2009.

[10] Song Yan, Johan Wirta, and Joni-Kristian Kämäräinen, "Anthropometric clothing measurements from 3d body scans," 2019.

[11] Qianli Ma, Jinlong Yang, Anurag Ranjan, Sergi Pujades, Gerard Pons-Moll, Siyu Tang, and Michael J. Black, "Learning to dress 3d people in generative clothing," in *Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[12] Raquel Vidaurre, Igor Santesteban, Elena Garces, and Dan Casas, "Fully Convolutional Graph Neural Networks for Parametric Virtual Try-On," *Computer Graphics Forum (Proc. SCA)*, 2020.

[13] Hsuan-I Ho, Lixin Xue, Jie Song, and Otmar Hilliges, "Learning locally editable virtual humans," 2023.

[14] Yan Zhang, Mohamed Hassan, Heiko Neumann, Michael J. Black, and Siyu Tang, "Generating 3d people in scenes without people," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[15] Andrés Fuster-Guilló, Jorge Azorín-López, Marcelo Saval-Calvo, Juan Miguel Castillo-Zaragoza, Nahuel Garcia-DUrso, and Robert B. Fisher, "Rgb-d-based framework to acquire, visualize and measure the human body for dietetic treatments," *Sensors*, vol. 20, no. 13, 2020.

[16] Sergey Prokudin, Christoph Lassner, and Javier Romero, "Efficient learning on point clouds with basis point sets," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 4332–4341.

[17] Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A. A. Osman, Dimitrios Tzionas, and Michael J. Black, "Expressive body capture: 3D hands, face, and body from a single image," in *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10975–10985.

[18] Ahmed A. A. Osman, Timo Bolkart, and Michael J. Black, "STAR: Sparse trained articulated human body regressor," in *Computer Vision - ECCV 2020*, Cham, Aug. 2020, vol. 6 of *Lecture Notes in Computer Science, 12351*, pp. 598–613, Springer.

[19] Federica Bogo, Angjoo Kanazawa, Christoph Lassner, Peter Gehler, Javier Romero, and Michael J. Black, "Keep it SMPL: Automatic estimation of 3D human pose and shape from a single image," in *Computer Vision – ECCV 2016*. Oct. 2016, Lecture Notes in Computer Science, Springer International Publishing.

[20] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," 2019.

[21] Nikos Kolotouros, Georgios Pavlakos, Michael J. Black, and Kostas Daniilidis, "Learning to reconstruct 3d human pose and shape via model-fitting in the loop," 2019.

[22] Muhammed Kocabas, Nikos Athanasiou, and Michael J. Black, "Vibe: Video inference for human body pose and shape estimation," 2020.

[23] Shunsuke Saito, Jinlong Yang, Qianli Ma, and Michael J. Black, "Scanimate: Weakly supervised learning of skinned clothed avatar networks," 2021.

[24] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016.

[25] Timo von Marcard, Gerard Pons-Moll, and Bodo Rosenhahn, "Human pose estimation from video and imus," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 8, pp. 1533–1547, Jan. 2016.

[26] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015, pp. 1–6.

[27] Joblib Development Team, "Joblib: running python functions as pipeline jobs," 2020.

[28] MMPose Contributors, "Openmmlab pose estimation toolbox and benchmark," `https://github.com/open-mmlab/mmpose`, 2020.

[29] Connor Z. Lin, Koki Nagano, Jan Kautz, Eric R. Chan, Umar Iqbal, Leonidas Guibas, Gordon Wetzstein, and Sameh Khamis, "Single-shot implicit morphable faces with consistent texture parameterization," in *ACM SIGGRAPH 2023 Conference Proceedings*, 2023.

Arquitecturas eficientes

Evaluación del rendimiento de técnicas de encriptación clásicas y post-cuánticas en dispositivos IoT de bajo consumo

Aritz Herrero, Javier Navaridas, Jose A. Pascual [1]

Resumen— **Este artículo investiga distintas técnicas de encriptación para su uso en dispositivos de bajo consumo destinados a ser desplegados en entornos IoT y Edge. El objetivo es encontrar un punto dulce con respecto al consumo de recursos y el rendimiento en este tipo de dispositivos. En particular, nos centramos en dos placas de la empresa Nordic Semiconductor: la NRF52840-DK, de menores prestaciones y basada en ARM Cortex-M4 y la NRF5340-DK, que utiliza un ARM Cortex-M33. Evaluamos diversas implementaciones, tanto software como hardware, de los algoritmos clásicos AES y RSA. Nuestra evaluación incluye, además, diversas implementaciones de los algoritmos post-cuánticos Kyber, NTRU y Saber. Los resultados muestran que los algoritmos post-cuánticos tienen un consumos de energía órdenes de magnitud mayor que los clásicos (AES). En concreto, el algoritmo post-cuántico más eficiente de nuestro estudio resulta ser Kyber.**

Palabras clave— **Dispositivos de bajo consumo IoT, Ciberseguridad, Criptografía clásica, Criptografía post-cuántica, Aceleración por hardware.**

I. Introducción

Atendiendo a los requisitos introducidos por las nuevas normativas, estándares y guías de buenas prácticas en materia de protección de datos y de las comunicaciones, los fabricantes de procesadores están desarrollando distintas familias de productos destinados a sistemas con recursos limitados en cuanto a consumo y dimensiones. En la actualidad, la arquitectura ARM es una de las principales tecnologías utilizadas en los procesadores integrados en los dispositivos IoT y Edge, gracias a su bajo consumo de energía con respecto a su rendimiento. ARM está comenzando a integrar en sus distintas familias de productos, capacidades de aceleración hardware embebidas, destinadas a liberar la CPU de la carga que supone el procesamiento de las operaciones necesarias para proteger las comunicaciones inalámbricas. En este artículo se busca evaluar el uso y la idoneidad de este tipo de técnicas hardware frente a las implementaciones software actuales como un mecanismo de mejora del rendimiento de los dispositivos.

Por otro lado, cuando no se puede contar en el diseño de los dispositivos IoT con procesadores de gran capacidad de cómputo, la sobrecarga que supone para el dispositivo garantizar la protección de los datos y las comunicaciones hace que en muchos casos sea inviable su implementación para cumplir con las normativas y estándares exigibles a este tipo de sistemas. Por ello es importante cerciorarse de sí los distintos dispositivos son capaces de llevar a cabo el cifrado y desencriptado de los datos y entender el impacto que estos supondrían en las prestaciones y los recursos de los dispositivos.

Además, la aparición de los primeros sistemas de computación cuántica y la cada vez más cercana disponibilidad de sistemas cuánticos de gran escala suponen una amenaza importante para las infraestructuras de información global. La criptografía de clave pública, ampliamente utilizada en Internet hoy en día, se basa en problemas matemáticos difíciles de resolver con métodos de computación clásica. Sin embargo, los sistemas criptográficos basados en estos problemas difíciles, como AES y RSA, podrían ser fácilmente descifrados por medio de la computación cuántica. Esto traerá consigo la obsolescencia de los sistemas de seguridad actuales y tendrá un impacto dramático en todas las industrias que dependan de información crítica.

La criptografía post-cuántica trata de evitar dichos problemas de seguridad por medio de algoritmos que sean resistentes a los ataques tanto de computadoras clásicas como cuánticas. Con el objetivo de identificar dichas implementaciones actuales, y elegir la más completa y segura para que pase a ser el pilar que sostenga los nuevos sistemas de seguridad, el National Institute of Standards and Technology (NIST) está impulsando el desarrollo de nuevas técnicas criptográficas post-cuánticas por medio de una competición que decidirá el nuevo estándar criptográfico a emplear en el futuro [1].

Por todo esto, nuestro estudio también incluye la evaluación del rendimiento de varios de los algoritmos recogidos en dicha competición. En particular, en este trabajo nos centraremos en los algoritmos criptográficos Kyber, NTRU y Saber, ya que los tres fueron designados entre los finalistas de la segunda ronda de la competición[1].

II. Selección de algoritmos

Para realizar nuestro análisis, se han seleccionado varios algoritmos de criptografía clásica y varios algoritmos post-cuánticos para su evaluación. Entre los algoritmos *clásicos* se han seleccionado aquellos que se encuentran implementados en el hardware que proporcionan las placas y que nos va a permitir compararlos con diferentes implementaciones software.

[1]Dpto. de Arquitectura y Tecnología de Computadores, Universidad del País Vasco — UPV/EHU

[1]Posteriormente, NTRU y Saber fueron desechados tras la tercera ronda debido a su similitud con Kyber y a sus peores prestaciones, en línea con los resultados de nuestro análisis.

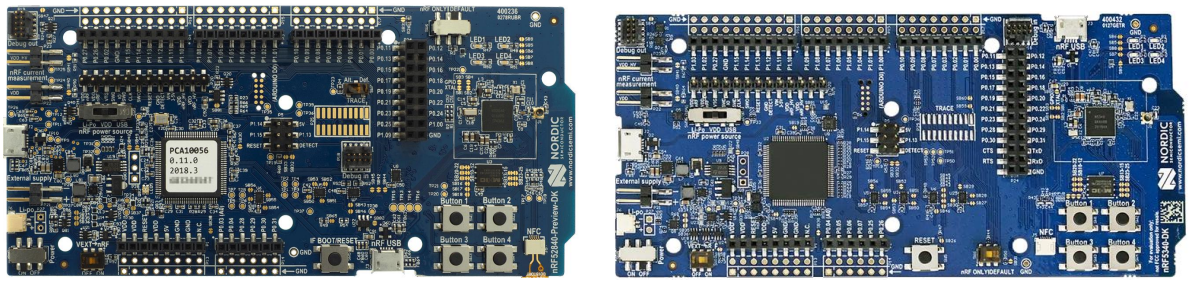


Fig. 1: Placas utilizadas en nuestro estudio. NRF52840-DK (izquierda) y NRF5340-DK (derecha)

Con el objetivo de cubrir tanto algoritmos de criptografía simétrica como de criptografía asimétrica, se han considerado los siguientes algoritmos, enumerados junto con algunas de sus características:

- Algoritmos de cifrado simétrico AES128-CBC y AES256-CBC: Estándar AES [2] de 128 y 256 bits y modo CBC.
- Algoritmos de cifrado asimétrico RSA [3] con tamaño de clave de 2048 bits

Además, debido a la importancia que están adquiriendo los algoritmos post-cuánticos [1], se ha decidido incorporarlos en el banco de pruebas. Los avances que se están realizando en computación cuántica van a permitir *romper* algoritmos de cifrado asimétrico (RSA y ECC) en el futuro próximo. Debido a ello, el NIST ha organizado una competición de la que saldrán los algoritmos que reemplazarán a los algoritmos utilizados en la actualidad y que, en un futuro, deberán ser también soportados por sistemas embebidos que traten con datos críticos.

La integración de estos algoritmos se ha realizado utilizando módulos externos que permiten añadirlos a nuestro banco de pruebas de manera sencilla. En concreto, se implementaron los siguientes algoritmos de encriptación post-cuántica:

- Algoritmo de encapsulado de claves Kyber-Crystal [4].
- Algoritmo de encapsulado de claves NTRU [5].
- Algoritmo de cifrado asimétrico Saber [6].

III. Configuración experimental

Con el objetivo de que el banco de pruebas desarrollado fuera representativo de los empleados en las instalaciones industriales de nuestro entorno, se adquirieron dos placas de la empresa Nordic Semiconductor: la NRF52840-DK de capacidad limitada y la NRF5340-DK que dispone de mayores prestaciones. Además, para medir el consumo energético se ha adquirido también el módulo NRF6707 de la misma empresa, el cual permite obtener mediciones del consumo en tiempo real. La Figura 1 muestra una imagen de las placas utilizadas.

En concreto, la NRF52840-DK dispone de un microcontrolador ARM Cortex-M4 a 64 MHz, con 1 MB de Flash y 256 KB de RAM. La NRF5340-DK, por su parte, ofrece mejores prestaciones al disponer de un microcontrolador ARM Cortex-M33 a 128 MHz, con 1 MB de Flash y 512 KB de RAM. Ambas placas incluyen aceleradores Arm CryptoCell de las familias 310 y 312, respectivamente. El resto de componentes de las placas, orientados principalmente a las comunicaciones inalámbricas, son similares y tienen un impacto despreciable en las métricas que capturamos en nuestro estudio.

Para la evaluación de los algoritmos criptográficos hemos optado por el desarrollo desde cero de un entorno de pruebas que permite la integración, ejecución y evaluación de manera sencilla de cualquier algoritmo criptográfico. Para ello se ha diseñado y desarrollado un banco de pruebas sobre el nRF-Connect-SDK [7] desarrollado por Nordic y que utiliza el sistema operativo de tiempo real Zephyr [8].

Nuestro banco de pruebas está diseñado para evaluar algoritmos con y sin aceleración hardware y proporciona tres métricas:

1. Tiempo de ejecución en ciclos
2. Tamaño en memoria de la implementación
3. Consumo de energía

Así mismo, se ha creado una herramienta para recoger los datos de las placas durante la ejecución de los algoritmos y las lecturas del consumo energético del módulo de energía. Esto permite la automatización del proceso de recogida de resultados.

IV. Evaluación del rendimiento

Para realizar las pruebas de rendimiento, se carga el código relativo al algoritmo que se quiere ejecutar y se desactivan el resto de los algoritmos con el objetivo de medir el consumo de memoria de manera más precisa.

Las tablas de uso de memoria flash y SRAM así como el tamaño de stack y heap se han calculado con los límites necesarios para que funcione cada algoritmo, aunque a la hora de utilizar la placa de medición de consumo, nRF52840 requiere en algunos casos un heap mayor para evitar corrupción en los mensajes enviados a través de la interfaz RTT y UART.

De cara a interpretar los resultados hay que tener en cuenta que un segundo equivale a 323.768 tics y que las placas funcionan con un voltaje de 3,3 voltios. El número de iteraciones (bloques) que se han utilizado en los experimentos de cada uno de los algoritmos es el siguiente:

- AES: 63000 iteraciones
- RSA: 30 iteraciones
- Kyber: 150 iteraciones
- NTRU: 40 iteraciones
- Saber: 100 iteraciones

Tabla I: Consumo de memoria (en bytes) de AES 128 en la placa nRF52840.

nRF52840	Backend	FLASH	SRAM	Stack
AES 128	Cxx	61720	20864	2048
	Oberon	79136	29664	2048
	Vanilla	71016	30688	3072

Tabla II: Consumo de memoria (en bytes) de AES 128+256 en la placa nRF5340.

nRF5340	Backend	FLASH	SRAM	Stack
AES 128+256	Cxx	64448	19496	2048
	Oberon	80452	28280	2048
	Vanilla	72204	29296	3072

Tabla III: Tiempos de ejecución de AES en segundos.

	Backend	Encrypt	Decrypt
M4 128	Cxx	11,92	12,08
	Oberon	13,53	14,30
	Vanilla	10,59	14,95
M33 128	Cxx	8,10	8,60
	Oberon	12,75	13,56
	Vanilla	15,10	11,74
M33 256	Cxx	8,35	8,74
	Oberon	15,10	16,09
	Vanilla	9,13	13,28

Tabla IV: Consumo de energía de AES (en mJ).

	Backend	Encrypt	Decrypt	Total
M4 128	Cxx	146,15	147,69	**293,84**
	Oberon	152,79	161,47	314,26
	Vanilla	128,46	179,72	308,18
M33 128	Cxx	90,01	95,01	**185,02**
	Oberon	129,61	139,43	269,04
	Vanilla	175,86	138,15	314,02
M33 256	Cxx	92,67	96,96	**189,63**
	Oberon	151,57	161,76	313,33
	Vanilla	106,71	156,41	263,12

A. Resultados del algoritmo AES

El algoritmo AES se ha evaluado utilizando 3 implementaciones distintas. Por un lado, se han evaluado dos implementaciones software (Oberon y Vanilla) y otra que utiliza el hardware de aceleración del que disponen las placas (Cxx). En el caso de la placa que utiliza el procesador M4, solo la versión AES de 128 bits se encuentra implementada en hardware. En el caso de la que dispone del chip M33, se encuentran disponibles en hardware tanto la versión de 128 bits como la de 256 bits.

Los experimentos se han realizado cifrando y descifrando 63000 bloques de 16 bytes y utilizando un heap de tamaño 8192 bytes en todos los casos. Los resultados obtenidos claramente muestran que el uso del hardware de aceleración para AES en dispositivos embebidos supera en rendimiento al uso de implementaciones software.

Tabla V: Consumo de memoria (en bytes) de RSA 2048.

	Backend	FLASH	SRAM	Stack	Heap
M4	Cxx	88156	22112	3072	8192
	Vanilla	80940	40074	4096	16394
M33	Cxx	89332	20736	3072	8192
	Vanilla	81940	38706	4096	16394

Tabla VI: Tiempos de ejecución de RSA en segundos.

	Backend	Encrypt	Decrypt
M4	Cxx	0,1586	3,8116
	Vanilla	0,6497	27,5561
M33	Cxx	0,1468	3,7396
	Vanilla	0,4987	21,5528

Tabla VII: Consumo de energía de RSA (en mJ).

	Backend	Encrypt	Decrypt	Total
M4	Cxx	2,02	51,73	**53,75**
	Vanilla	8,39	345,40	353,79
M33	Cxx	1,92	55,29	**57,20**
	Vanilla	7,41	311,15	318,56

Si nos fijamos en el consumo de memoria (Tablas I y II) claramente se ve que el uso de primitivas hardware reduce el uso total de memoria entre un 17 % y un 26 %. Dado que este tipo de sistemas suele disponer de poca memoria RAM, su utilización permitirá el uso de esta memoria libre para otras tareas.

En cuanto al tiempo de ejecución necesario para cifrar y desencriptar (Tabla III), la implementación hardware supera en prácticamente todos los casos a las implementaciones software, en algunos casos de manera substancial: con un tiempo de ejecución de hasta un 45 % menor. Las diferencias son en general más evidentes en la placa que utiliza el procesador M33, ya que este incorpora una versión más rápida del acelerador criptográfico de ARM.

Para finalizar vamos a analizar el consumo de energía que se encuentra detallado en la Tabla IV. En este caso, el uso del acelerador criptográfico da siempre como resultado un menor consumo de energía, con reducciones que pueden superar el 40 %. Cabe destacar que la placa con el M33 resulta mucho más eficiente en términos de energía que la placa que utiliza el procesador M4, especialmente al hacer uso del acelerador criptográfico.

B. Resultados del algoritmo RSA (PKCS21)

En esta sección se presentan los resultados obtenidos utilizando criptografía asimétrica. Para ello, se han evaluado dos implementaciones del algoritmo RSA, una software (Vanilla) y otra que utiliza aceleración hardware (Cxx).

Los experimentos consisten en el cifrado y desencriptado de 30 bloques de 128 bytes. Debido a que la generación de la clave es muy costosa y de duración

Tabla VIII: Especificación de nivel de seguridad propuesta por el NIST que compara el coste de realizar ataques para romper cifrados con AES y claves de 128, 192 y 256 bits.

	128	192	256	Superior
KYBER_512	X			
KYBER_768		X		
KYBER_1024			X	
KYBER_512_90S	X			
KYBER_768_90S		X		
KYBER_1024_90S			X	
NTRU_HPS_2048_509	X			
NTRU_HPS_2048_677		X		
NTRU_HPS_4096_821			X	
NTRU_HPS_4096_1229				X
NTRU_HRSS_701		X		
NTRU_HRSS_1373				X
LIGHTSABER	X			
SABER		X		
FIRESABER			X	

variable, este tiempo no se tiene en cuanta a la hora de medir el tiempo de ejecución y los consumos de energía.

Los resultados obtenidos para RSA son similares a los obtenidos para AES, pero, en este caso, la ganancia obtenida por usar aceleración hardware es mucho mayor. Por un lado, en la Tabla V, se observa una reducción del consumo de memoria de aproximadamente un 14 %. Sin embargo, la mayor ganancia se puede apreciar tanto en el tiempo de ejecución (Tabla VI) como en el consumo de energía de las placas (Tabla VII). Esto era lo esperado debido a que RSA es un algoritmo muy pesado. El proceso de cifrado se realiza aproximadamente 4 veces más rápido utilizado una cuarta parte de la energía. El proceso de desencriptado obtiene mejoras aún mayores al realizarse unas 6 veces más rápido utilizando una sexta parte de la energía.

C. Criptografía post-cuántica

Pasamos a analizar los 3 algoritmos de criptografía post-cuántica: Kyber [4], NTRU [5] y Saber [6], incluyendo múltiples versiones de cada uno de ellos. Con el objetivo de realizar una comparación entre los diferentes algoritmos, los experimentos se han llevado a cabo realizando un número de iteraciones concreto para cada algoritmo (150, 40 y 100, respectivamente) debido a que el tiempo de ejecución entre ellos es variable.

Más adelante, una vez obtenidos los datos de consumo de energía, se calculará el consumo de energía por iteración (mJ/Nº iter), y de esta manera, se compararán los diferentes algoritmos, incluyendo AES. Para realizar esta comparación se utilizará el nivel de seguridad objetivo utilizando la Tabla VIII que muestra el nivel de seguridad equivalente de los algoritmos post-cuánticos evaluados frente a AES con diferentes tamaño de clave. Al igual que con RSA, la generación de claves no se considera a la hora de realizar las mediciones.

Tabla IX: Consumo de memoria de Kyber.

	Variant	FLASH	SRAM	Stack	Heap
M4	KYBER_512	64088	32704	10240	8192
	KYBER_768	64008	44416	20480	8192
	KYBER_1024	64216	56288	30720	8192
	KYBER_512_90S	102120	42944	20480	8192
	KYBER_768_90S	102040	44416	20480	8192
	KYBER_1024_90S	102248	56288	30720	8192
M33	KYBER_512	65056	31328	10240	8192
	KYBER_768	64992	43040	20480	8192
	KYBER_1024	65200	54912	30720	8192
	KYBER_512_90S	103104	41568	20480	8192
	KYBER_768_90S	103024	43040	20480	8192
	KYBER_1024_90S	103232	54912	30720	8192

Tabla X: Tiempos de ejecución de Kyber en segundos.

	Variant	Encrypt	Decrypt
M4	KYBER_512	3,41	3,64
	KYBER_768	5,40	5,66
	KYBER_1024	8,04	8,34
	KYBER_512_90S	4,79	5,27
	KYBER_768_90S	8,55	9,20
	KYBER_1024_90S	13,57	14,39
M33	KYBER_512	2,86	3,10
	KYBER_768	4,48	4,79
	KYBER_1024	6,66	7,03
	KYBER_512_90S	4,32	4,77
	KYBER_768_90S	7,64	8,24
	KYBER_1024_90S	12,08	12,83

Tabla XI: Consumo de energía de Kyber (en mJ).

	Variant	Encrypt	Decrypt	Total
M4	KYBER_512	41,39	42,47	83,86
	KYBER_768	66,12	66,95	133,07
	KYBER_1024	99,89	100,24	200,13
	KYBER_512_90S	56,86	61,60	118,46
	KYBER_768_90S	103,13	109,53	212,66
	KYBER_1024_90S	164,75	172,78	337,52
M33	KYBER_512	32,21	34,24	66,45
	KYBER_768	50,63	53,50	104,13
	KYBER_1024	75,07	78,82	153,89
	KYBER_512_90S	48,82	53,38	102,20
	KYBER_768_90S	86,37	92,31	178,68
	KYBER_1024_90S	137,47	144,42	281,90

C.1 Algoritmo Kyber

Los experimentos de Kyber incluyen tres tamaños de clave, así como dos implementaciones del algoritmo. La primera es la versión estándar del algoritmo (KYBER), mientras que la segunda es una variante híbrida que utiliza algoritmos clásicos (AES-256 y SHA2) cómo parte de su implementación.

Si nos fijamos en el consumo de memoria (Tabla IX), podemos ver que al aumentar el tamaño de clave se incrementa el uso de memoria en aproximadamente 22 KB. La única excepción es el caso del algoritmo híbrido (90S) que al pasar de 512 a 768 solo se incrementa 2 KBytes. En general, la versión híbrida requiere un uso substancialmente mayor de Flash, aproximadamente un 33 % más.

Tabla XII: Consumo de memoria de NTRU.

	Variant	FLASH	SRAM	Stack	Heap
M4	NTRU_HPS_2048_509	70348	52320	30720	8192
	NTRU_HPS_2048_677	70508	63328	40960	8192
	NTRU_HPS_4096_821	69996	74528	51200	8192
	NTRU_HPS_4096_1229	70364	86752	61440	8192
	NTRU_HRSS_701	70184	63968	40960	8192
	NTRU_HRSS_1373	71132	108992	81920	8192
M33	NTRU_HPS_2048_509	71332	50944	30720	8192
	NTRU_HPS_2048_677	71492	61952	40960	8192
	NTRU_HPS_4096_821	70980	73144	51200	8192
	NTRU_HPS_4096_1229	71348	85384	61440	8192
	NTRU_HRSS_701	71152	62584	40960	8192
	NTRU_HRSS_1373	72116	107600	81920	8192

Tabla XIII: Tiempos de ejecución de NTRU en segundos.

	Variant	Encrypt	Decrypt
M4	NTRU_HPS_2048_509	1,34	2,41
	NTRU_HPS_2048_677	2,18	4,43
	NTRU_HPS_4096_821	2,88	6,04
	NTRU_HPS_4096_1229	5,49	12,34
	NTRU_HRSS_701	2,05	4,52
	NTRU_HRSS_1373	7,16	18,33
M33	NTRU_HPS_2048_509	1,04	2,09
	NTRU_HPS_2048_677	1,66	3,57
	NTRU_HPS_4096_821	2,19	4,84
	NTRU_HPS_4096_1229	4,48	10,74
	NTRU_HRSS_701	1,53	3,64
	NTRU_HRSS_1373	6,13	16,56

Tabla XIV: Consumo de energía de NTRU (en mJ).

	Variant	Encrypt	Decrypt	Total
M4	NTRU_HPS_2048_509	15,15	27,15	42,30
	NTRU_HPS_2048_677	25,07	51,33	76,40
	NTRU_HPS_4096_821	33,39	70,69	104,08
	NTRU_HPS_4096_1229	63,31	142,62	205,93
	NTRU_HRSS_701	24,09	52,61	76,70
	NTRU_HRSS_1373	83,89	211,35	295,24
M33	NTRU_HPS_2048_509	12,05	24,64	36,69
	NTRU_HPS_2048_677	19,59	42,64	62,23
	NTRU_HPS_4096_821	25,92	57,72	83,64
	NTRU_HPS_4096_1229	52,25	125,76	178,01
	NTRU_HRSS_701	18,15	43,40	61,55
	NTRU_HRSS_1373	70,93	190,03	260,96

Pasando al tiempo de ejecución, reportado en la Tabla X, podemos ver que aumentar el tamaño de la clave incrementa esta métrica entre un 48 % y un 78 %. Además, la variante híbrida (90S) es entre un 40 % y un 82 % más lenta que la estándar, con diferencias que incrementan con el tamaño de clave. Cabe destacar que, a diferencia de RSA, el tiempo de cifrado y de desencriptado son similares entre sí para todas las configuraciones, siendo el desencriptado ligeramente más lento con diferencias siempre menores al 10 %.

Finalmente, en términos de energía (Tabla XI), podemos apreciar que el consumo de los procesos de cifrado y desencriptado se incrementa gradualmente con el tamaño de clave, entre el 50 % y el 80 %. Así mismo, el algoritmo híbrido (90S) consume entre un 30 % y un 65 % más que la versión estándar.

C.2 Algoritmo NTRU

Los experimentos de NTRU incluyen dos implementaciones del algoritmo. Por un lado, se evalúa NTRU_HPS, la versión original de NTRU [5], para la cual se utilizan conjuntos de parámetros que incrementan gradualmente la seguridad. También incluimos en nuestra evaluación NTRU_HRSS, una variante propuesta más recientemente [9] que simplifica ligeramente la versión original. Para esta segunda versión se utilizan dos conjuntos de parámetros correspondientes a seguridad de 192 bits y Superior.

Con respecto al consumo de memoria, mostrado en la Tabla XII, podemos ver que el uso de Flash es similar en todas las configuraciones, independientemente de la variante y el nivel de seguridad. Sin embargo, el consumo de memoria sí que depende de estos. Con HPS, cada incremento en el nivel de seguridad requiere entre 20 y 22 KBytes extra de memoria. En el caso de HRSS, pasar de protección de 192 bits a Superior requiere un consumo extra de 86 Kbytes. Dada la poca capacidad de recursos de las placas que estamos evaluando, la cantidad de memoria libre para el resto de módulos puede limitar drásticamente la practicidad de los diseños soportados por la placa.

Si nos fijamos en los tiempos de ejecución, expuestos en la Tabla XIII, podemos ver que aumentar el nivel de protección incrementa esta métrica entre un 32 % y un 122 %. La diferencia entre variantes es despreciable para seguridad de 192 bits, pero para protección Superior, HRSS es entre un 30 % y un 54 % más lento que HPS. El tiempo de desencriptado es entre 1,8 y 2,7 veces mayor al de cifrado para todas las variantes.

Los resultados de energía (Tabla XIV) son muy similares a los de tiempo de ejecución: aumentar la seguridad incrementa el consumo entre un 33 % y un 118 % mientras que el proceso de desencriptado consume entre 1,8 y 2,7 veces lo que el de cifrado. Igualmente, el uso de HRSS con seguridad Superior supone un incremento del consumo de entre un 31 % y un 51 % con respecto a HPS, mientras que con 192 bits, las diferencias son minúsculas.

C.3 Algoritmo Saber

Los experimentos con el algoritmo Saber incluyen tres variantes (conjuntos de parámetros) que representan niveles de seguridad crecientes [6]. LightSaber es la versión ligera y corresponde a un nivel de seguridad de 128 bits. Saber es la configuración estándar, correspondiente a 192 bits de seguridad. Finalmente, FireSaber es la versión que ofrece mayor seguridad, 256 bits.

Con respecto al consumo de memoria, mostrado en la Tabla XV, podemos ver que, al igual que ocurría con NTRU, el consumo de Flash no se ve afectado por el nivel de seguridad, mientras que el consumo de memoria sí que se ve afectado. En concreto, Saber requiere unos 2 KB más que Lightsaber, mientras que pasar de Saber a FireSaber supone 22 KB extra de memoria.

Si nos fijamos en los tiempos de ejecución, expues-

Tabla XV: Consumo de memoria de Saber.

	Variant	FLASH	SRAM	Stack	Heap
M4	LIGHTSABER	64632	42720	20480	8192
	SABER	64388	44128	20480	8192
	FIRESABER	64488	55808	30720	8192
M33	LIGHTSABER	65616	41344	20480	8192
	SABER	65372	42752	20480	8192
	FIRESABER	65472	54432	30720	8192

Tabla XVI: Tiempos de ejecución de Saber en segundos.

	Variant	Encrypt	Decrypt
M4	LIGHTSABER	3,94	4,62
	SABER	7,39	8,43
	FIRESABER	11,80	13,22
M33	LIGHTSABER	3,12	3,67
	SABER	5,91	6,76
	FIRESABER	9,35	10,55

Tabla XVII: Consumo de energía de Saber (en mJ).

	Variant	Encrypt	Decrypt	Total
M4	LIGHTSABER	44,40	50,66	95,06
	SABER	82,39	91,77	174,16
	FIRESABER	133,52	146,43	279,95
M33	LIGHTSABER	36,33	42,38	78,71
	SABER	68,45	78,03	146,48
	FIRESABER	110,24	173,63	283,87

tos en la Tabla XVI, podemos ver que aumentar el nivel de protección supone que la ejecución cueste entre un 56 % y un 89 % más. El tiempo de desencriptado es ligeramente mayor que el de cifrado (entre el 12 % y el 18 %).

Finalmente, pasamos a discutir los efectos en el consumo de energía, mostrado en la Tabla XVII. Podemos ver que aumentar el nivel de seguridad incrementa el consumo entre un 56 % y un 89 %. Si comparamos el consumo de cifrado y de desencriptado podemos ver que este segundo es ligeramente mayor: entre el 9 % y el 17 %).

C.4 Comparación entre algoritmos post-cuánticos

En las Tablas XVIII, XIX y XX se compara el consumo de energía de cada uno de los algoritmos post-cuánticos agrupados por el nivel de seguridad que ofrecen (128, 192, 256 bits, respectivamente). Cuando es posible, a modo de referencia, se compara también con el consumo de energía del algoritmo AES.

El análisis de los resultados claramente muestra que las implementaciones de AES, incluso las versiones software, tienen un consumo de energía órdenes de magnitud menor que la de los algoritmos post-cuánticos para el mismo nivel de seguridad.

Si comparamos los algoritmos post-cuánticos entre ellos, el más eficiente en términos de consumo de energía es Kyber en todos los casos, siendo los consumos de NTRU y Saber similares entre sí, pero entre un 60 % y un 113 % mayores que Kyber.

Tabla XVIII: Consumo de energía con seguridad de 128 bits.

128 bit security	M4 (µJ)	M33 (µJ)
AES 128 Cxx	4,7	2,9
AES 128 Oberon	5,0	4,3
AES 128 Vanilla	4,9	5,0
KYBER_512	559,1	482,3
KYBER_512_90S	662,9	597,1
NTRU_HPS_2048_509	1058,5	1012,5
LIGHTSABER	950,3	896,4

Tabla XIX: Consumo de energía con seguridad de 192 bits.

192 bit security	M4 (µJ)	M33 (µJ)
KYBER_768	728,0	608,6
KYBER_768_90S	976,0	847,6
NTRU_HPS_2048_677	1326,0	1210,5
NTRU_HRSS_701	1304,6	1173,2
SABER	1326,4	1217,4

Tabla XX: Consumo de energía con seguridad de 256 bits.

256 bit security	M4 (µJ)	M33 (µJ)
AES 256 Cxx	—	3,0
AES 256 Oberon	—	5,0
AES 256 Vanilla	—	4,2
KYBER_1024	957,0	772,1
KYBER_1024_90S	1389,1	1189,5
NTRU_HPS_4096_821	1540,0	1367,7
FIRESABER	1846,6	1642,2

V. Conclusiones

A modo de resumen, las contribuciones de nuestro trabajo son las siguientes:

- Banco de pruebas compuesto por sendas tarjetas de Nordic con componentes de aceleración hardware más Profiler KIT versión 1. Este banco de pruebas está desplegado en un entorno de laboratorio de la UPV/EHU.
- Conjunto de algoritmos criptográficos clásicos desplegados en nuestro banco de pruebas, funcionando con y sin aceleración hardware.
- Conjunto de algoritmos post-cuánticos desplegados en nuestro banco de pruebas.
- Evaluación de los algoritmos basada en métricas de rendimiento y eficiencia: memoria requerida, tiempo de ejecución y consumo de energía.

Los resultados obtenidos muestran que el uso de aceleradores hardware para las primitivas criptográficas acarrea beneficios tanto en tiempo de ejecución, como en consumo energético. AES obtiene mejoras en tiempo de ejecución de hasta un 45 %, reduciendo el consumo energético hasta en un 40 %. Estos beneficios son aún mayores para RSA, que puede reducir el tiempo de ejecución y el consumo de las operaciones de cifrado hasta 4 veces y de las operaciones de desencriptado hasta 6 veces. En ambos casos también se reduce, de manera significativa, el consumo de memoria.

Con respecto a las implementaciones de algoritmos post-cuánticos los resultados son dispares. Por un lado, vemos que este tipo de dispositivos de bajo consumo posee suficientes recursos para poder ejecutar este tipo de algoritmos. Sin embargo, nuestros resultados sugieren que las necesidades en términos de capacidad de cómputo y de consumo de energía requeridos para la ejecución de algoritmos post-cuánticos es muy superior a la de los clásicos, lo que puede limitar su practicidad y, por tanto, su utilización por la comunidad. Esto puede representar un importante vector de ataque dadas las vulnerabilidades existentes tras la obsolescencia de los sistemas actuales.

Si realizamos una comparación de los distintos algoritmos post-cuánticos estudiados, podemos ver que Kyber es el algoritmo que obtiene mejores resultados en general, mientras que NTRU y Saber ofrecen rendimientos similares entre sí. Estos resultados se alinean con los reportados por NIST [1].

Agradecimientos

This work is supported by the Basque Government (projects ELKARTEK21/89 and IT1504-22) and by the Spanish Ministry of Economy and Competitiveness MINECO (PID2019-104966GB-I00). Dr. Javier Navaridas is a Ramón y Cajal fellow from the Spanish Ministry of Science, Innovation and Universities (RYC2018-024829-I).

Referencias

[1] G Alagic, D Apon, D Cooper, et al., "Status report on the third round of the NIST post-quantum cryptography standardization process," *US Department of Commerce, NIST*, 2022.

[2] M Dworkin, E Barker, J Nechvatal, et al., "Advanced encryption standard (AES)," *Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD*, 2001-11-26 2001.

[3] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[4] J Bos, L Ducas, E Kiltz, et al., "CRYSTALS - kyber: A CCA-secure module-lattice-based KEM," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018, pp. 353–367.

[5] J Hoffstein, J Pipher, and JH Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory: 3rd Intl. Symposiun, ANTS-III Portland, Oregon, USA, June 21–25*, 1998, pp. 267–288.

[6] JP D'Anvers, A Karmakar, S Sinha Roy, and F Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," in *10th Intl. Conf. on Cryptology in Africa, Marrakesh, Morocco, May 7–9,*, 2018, pp. 282–305.

[7] Nordic Semiconductor, "nRF Connect SDK," `https://www.nordicsemi.com/Products/Development-software/nRF-Connect-SDK`, Accessed: 2023-05-25.

[8] Zephyr project, "Zephyr RTOS kernel," `https://docs.zephyrproject.org/latest/introduction/index.html`, Accessed: 2023-05-25.

[9] A Hülsing, J Rijneveld, J Schanck, and P Schwabe, "High-speed key encapsulation from NTRU," in *Cryptographic Hardware and Embedded Systems – CHES 2017*, 2017, pp. 232–252.

Protección de comunicaciones entre vehículos autónomos mediante el uso de códigos de corrección de errores

J. Gracia-Morán*, A. Vicente-García**, L.J. Saiz-Adalid*

***Resumen*—La reducción de tamaño de la tecnología CMOS ha permitido aumentar las prestaciones de los sistemas empotrados. Sin embargo, esta disminución de tamaño implica un aumento en la tasa de fallos. En este sentido, cobra gran importancia la protección, de forma sencilla y rápida, de los datos que procesan estos sistemas.**

Uno de los aspectos más importantes de los sistemas empotrados es su consumo de energía. Por ejemplo, a la hora de transmitir un dato, si este se recibe de forma errónea, hay que pedir su retransmisión, lo que conlleva un gasto de energía no previsto. Es por ello por lo que parece interesante proteger los datos a intercambiar.

Si bien los códigos de corrección de errores (ECC) se han utilizado tradicionalmente para proteger sistemas de memoria o para transmisiones masivas de datos, también se pueden utilizar para proteger datos de pocos bits transmitidos por una red de comunicaciones. En este trabajo, utilizando dos prototipos de vehículos autónomos no tripulados basados en Arduino que se comunican entre sí, y mediante diferentes ECC, hemos estudiado la confiabilidad de la comunicación entre ellos. Con el estudio de diferentes ECC se pretende valorar y analizar la mejora obtenida en la confiabilidad, así como el sobrecoste en el que se incurre, cuestión importante en sistemas empotrados.

Palabras Clave—Arduino, Códigos de Corrección de Errores, Confiabilidad, Sistemas Empotrados, Transmisión de datos

I. Introducción

El incremento de prestaciones conseguido con el aumento en la escala de integración de la tecnología CMOS ha acarreado también una reducción en la confiabilidad de los sistemas integrados [1]. De los diferentes métodos de tolerancia a fallos existentes [2], los códigos de corrección de errores (ECC) [3] permiten una gran cobertura de fallos [4][5][6][7][8][9] a un coste reducido.

En trabajos previos ya hemos estudiado el impacto de la inclusión de ECC en sistemas empotrados [10][11]. En estos trabajos anteriores se protegían los datos críticos del sistema, de los cuales dependía su correcto funcionamiento. Estos datos se mantenían en memoria dentro de un único sistema, pero ¿qué pasa si estos datos críticos se tienen que compartir con otros sistemas empotrados? Si un dato se corrompe durante su transmisión, pueden ocurrir dos cosas. Si el equipo receptor identifica el dato como erróneo, pedirá la retransmisión del mismo, con el consiguiente gasto de tiempo y energía tanto en el emisor como en el receptor. Si no lo identifica como erróneo, el dato erróneo será procesado como correcto, con efectos no previstos.

* Instituto ITACA, Universitat Politècnica de València, 46022 Valencia, España.
e-mail: { jgracia, ljsaiz } @ itaca.upv.es
** Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, 46022 Valencia, España.
e-mail: advigar1 @ inf.upv.es

En el primer caso, para ahorrar tanto tiempo como energía, lo ideal sería que, una vez identificado el dato como erróneo, el receptor tenga la capacidad suficiente como para corregir los errores producidos. De esta forma, no haría falta reenviar los datos.

El segundo caso es más grave, ya que puede provocar la pérdida de control del sistema y causar un comportamiento errático. Esto se puede evitar si se diseña un mecanismo de tolerancia a fallos ajustado al comportamiento del sistema.

Una posible solución para ambos casos sería utilizar ECC para proteger el dato a enviar. Tradicionalmente, los códigos de corrección de errores se han utilizado para proteger sistemas de memoria, sensibles a distintos mecanismos físicos que pueden provocar errores en la información almacenada. También se utilizan en transmisiones masivas de datos, como la distribución de contenidos *online*, televisión digital o comunicaciones satelitales, pero es menos frecuente su uso en comunicaciones de pocos bits. Si utilizamos esta técnica de tolerancia a fallos, que ya hemos visto que consigue excelentes resultados a un coste muy bajo [10][11], se podrían ahorrar retransmisiones de datos.

En el presente trabajo, hemos realizado este estudio. Para ello, hemos utilizado dos prototipos de vehículos autónomos no tripulados, ambos equipados con un sistema de conducción basado en Arduino y distintos sensores, y que se comunican entre ellos mediante módulos Bluetooth. Los vehículos funcionarán uno detrás de otro, siguiendo un circuito. El primer vehículo dará las órdenes al segundo. La idea es que este segundo vehículo debe obedecer las órdenes enviadas por el primer vehículo, sin que haya colisiones entre ellos.

Para estudiar la tolerancia a fallos y mejorar la confiabilidad de la comunicación entre ambos vehículos se utilizarán códigos correctores de errores para proteger la información transmitida. Además, introduciremos diversos fallos mediante la técnica de inyección de fallos. Con la prueba de diferentes ECC se pretende valorar y analizar la mejora obtenida en la confiabilidad, así como la sobrecarga introducida, aspecto importante en sistemas empotrados como los utilizados en este trabajo.

A continuación, la Sección II resume el sistema empotrado utilizado, así como los ECC implementados. En la Sección III se describen las pruebas realizadas para comprobar la validez de la propuesta. La Sección IV presenta los diferentes resultados obtenidos durante la evaluación del sistema, estudiando principalmente el tamaño del código y su tiempo de ejecución. Y finalmente, la Sección V concluye este trabajo.

II. Descripción del sistema utilizado

A. *Sistema empotrado utilizado*

Para comprobar el funcionamiento de nuestra propuesta, hemos utilizado dos pequeños vehículos, equipados cada uno de ellos con una placa basada en Arduino UNO R3 y diversos sensores y actuadores [12][13]. Esta placa tiene las siguientes particularidades:

- Microcontrolador ATmega328p.
- Memoria flash de 32 KB, SRAM de 2 KB y EEPROM de 1KB.
- Velocidad del reloj de 16MHz.

La idea de funcionamiento del sistema es la siguiente. Un primer vehículo (ver Fig. 1) circulará siguiendo un determinado circuito. Este vehículo está equipado con los sensores necesarios para el seguimiento de la línea que marca el circuito, además de un módulo bluetooth que utilizará para intercambiar información con el vehículo 2 (ver Fig. 2). Estos datos son los que se deben proteger, pues el funcionamiento del segundo vehículo depende de esta información. El segundo vehículo también está equipado con sensor para el seguimiento de línea y módulo bluetooth.

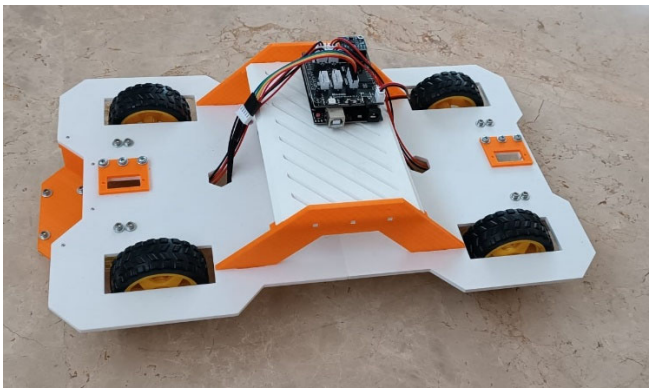


Fig. 1. Vehículo 1 [12] (vehículo maestro) utilizado en las pruebas.

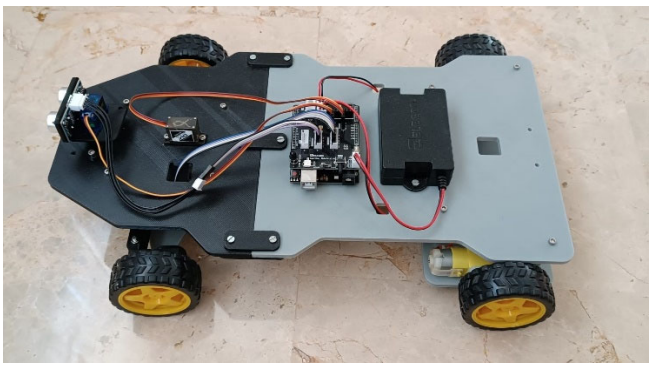


Fig. 2. Vehículo 2 [13] (vehículo esclavo) utilizado en las pruebas.

La Fig. 3 muestra un esquema del comportamiento del sistema. La idea es que el vehículo 1 indicará al vehículo 2 lo que debe hacer mediante una serie de comandos. En el caso de que el vehículo 2 detecte un error en el comando recibido, y no pueda corregirlo, pedirá la retransmisión de la última orden. La Tabla I muestra los comandos a enviar, así como su codificación en binario.

A la hora de codificar los datos con el ECC correspondiente, surgen dos alternativas. Por un lado, al ser sólo cuatro los comandos utilizados, se podría tener una tabla con el dato y los bits de código. En este caso, la tabla será muy pequeña, pero si el número de comandos es elevado habría que considerar el tamaño de la tabla y reconsiderar si merece la pena o no. En cualquier caso, a costa de la memoria necesaria para almacenar dicha tabla, reducimos el tamaño del programa (al no tener que implementar el codificador) y, más importante, ahorramos tiempo de procesamiento, al no tener que codificar los datos con el ECC cada vez que se envían. Así, se protegen los datos durante su transmisión. Sin embargo, en entornos críticos, esta implementación puede dar problemas, pues la memoria es el componente que más fallos acumula [14].

La segunda opción es codificar los comandos con el ECC cada vez que se envían, protegiendo así el comando tanto durante su almacenamiento en memoria como durante la transmisión. Para lograr este aumento en la tolerancia a fallos, es necesario implementar el codificador, lo que conlleva un aumento en el tiempo de procesamiento y un mayor tamaño del programa. Dependiendo del tamaño de la tabla de la opción anterior, esto se puede traducir en una mayor utilización de memoria.

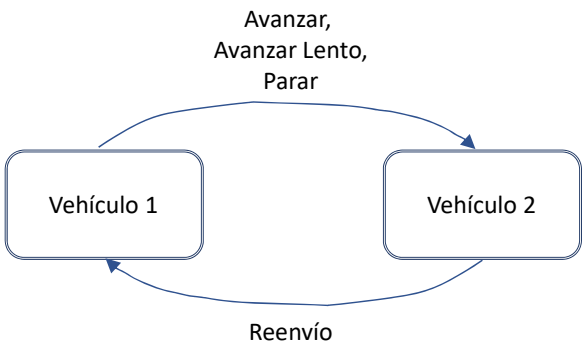


Fig. 3. Funcionamiento básico sistema comunicaciones.

TABLA I
COMANDOS A ENVIAR Y SU CODIFICACIÓN EN BINARIO

COMANDO	Codificación
Reenvío	0000
Avanzar	0001
Avanzar Lento	0010
Parar	0011
Retroceder	0100

B. *Códigos de Corrección de Errores utilizados*

Los datos enviados mediante bluetooth tienen una longitud de ocho bits. De esta forma, podemos codificar los diferentes comandos con cuatro bits, y utilizar otros cuatro bits para los bits de paridad. Con este formato, se podría ampliar el repertorio de comandos actuales si fuera necesario. En futuros trabajos queremos estudiar cómo afectaría a las prestaciones del sistema el tener que enviar más de una palabra de ocho bits.

Así pues, los ECC utilizados en este trabajo y que nos permitirán comprobar la viabilidad de nuestra propuesta son los siguientes:

Hamming SEC-DED (8,4)

El código de Hamming es un código SEC (*Single Error Correction*) [15], que puede corregir un bit erróneo con operaciones de codificación y decodificación sencillas y rápidas, así como con la redundancia más baja. La distancia de Hamming de estos códigos es 3. La matriz de paridad para la versión (7, 4) de este ECC (4 bits de datos codificados en una palabra de 7 bits) se muestra a continuación:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (1)$$

Por otra parte, el código de Hamming extendido permite corregir un error en un bit, o detectar dos bits en error añadiendo para ello únicamente un bit de paridad más. Este último bit calcula la paridad par de toda la palabra codificada. Al extender el código con este bit, se aumenta la distancia de Hamming a 4, lo que permite la cobertura indicada. De esta forma, se dice que los códigos de Hamming extendidos son códigos SEC-DED (*Single Error Correction – Double Error Detection*).

Hsiao SEC-DED (8,4)

Los códigos Hsiao son una versión optimizada de los códigos de Hamming extendidos [16]. En este caso, la matriz de paridad de un código Hsiao debe cumplir los siguientes requerimientos:

- Cada columna tiene que ser diferente y distinta de cero.
- Cada columna contiene un número impar de 1s.
- El número total de 1s en la matriz de paridad debe ser mínimo.
- El número de 1s de cada fila debe ser idéntico, o lo más cercano posible, al número medio de 1s.

La matriz de paridad utilizada en este trabajo es la siguiente (2):

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \quad (2)$$

El resultado de diseñar un código con estas características es que los errores simples generarán síndromes únicos, y con un número impar de unos. Por el contrario, los errores dobles generan síndromes con un número par de unos, que además se pueden repetir (distintos errores dobles pueden generar el mismo síndrome). De esta forma, se puede distinguir fácilmente entre errores simples y dobles. Los primeros se pueden corregir y los segundos solo se pueden detectar.

SEC-DAEC-TAED (8, 3)

En cuanto al tercer ECC que queremos estudiar, queríamos diseñar un código utilizando una metodología propia, presentada en [17], con una tolerancia a fallos diferente a los dos anteriores, siempre teniendo en cuenta que el tamaño máximo de la palabra codificada es de ocho bits. En concreto, queríamos reforzar la corrección y detección de errores adyacentes. Se define el error adyacente como un error múltiple donde todos los bits erróneos son contiguos, siendo este el tipo de error múltiple más frecuente [18][19]. El problema que hemos encontrado es que no existe un ECC capaz de corregir dos errores adyacentes con cuatro bits de paridad para cuatro bits de datos. Así pues, hemos optado por diseñar un ECC para tres bits de datos y cinco bits de paridad. Con este nuevo requerimiento, y utilizando nuestra metodología, sí que hemos podido diseñar un ECC capaz de corregir errores simples y dobles adyacentes, así como de detectar errores triples adyacentes (*SEC: Single Error Correction – DAEC: Double Adjacent Error Correction – TAED: Triple Adjacent Error Detection*). La matriz de paridad de este ECC es la siguiente (3):

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

III. Descripción de las Pruebas Realizadas

A. Análisis de la cobertura de los códigos

Antes de poner en marcha los vehículos con los ECC, se ha hecho un análisis de las coberturas de corrección y detección de cada código. Para ello, se ha utilizado la técnica de inyección de fallos. Dado que son bien conocidos, para el código Hamming extendido y el código de Hsiao no se ha hecho un análisis exhaustivo, simplemente se ha comprobado que cumplen con su descripción (corregir errores simples y detectar errores dobles), para descartar fallos en la implementación.

Dado que el código SEC-DAEC-TAED (8, 3) es nuevo y diseñado específicamente para esta aplicación, se ha hecho un estudio más exhaustivo de su cobertura. Para ello se inyectan sistemáticamente todos los posibles errores de un tipo determinado, y se comprueba cuántos de ellos se corrigen o detectan. Es importante tener en cuenta que en este análisis no se considera la frecuencia con la que aparecen los distintos tipos de error, sino comprobar todos los posibles errores de un tipo, y determinar qué porcentaje de los mismos es corregido o detectado.

Como era de esperar, el código funciona correctamente cuando no se produce ningún error. Esto es importante, hay que asegurarse de que sea así. Aunque, como se ha dicho, no se entra a valorar la frecuencia, lo más frecuente es que no se produzcan errores. También es capaz de corregir todos los errores de un solo bit. Estos dos casos son comunes y aparecen tanto en la gráfica de cobertura de errores aleatorios (Fig. 4) como en la de errores adyacentes (Fig. 5).

Las diferencias vienen en los errores múltiples, donde se distingue entre aleatorios (todas las posibles combinaciones) y adyacentes (errores múltiples donde los bits erróneos son contiguos). Por ejemplo, hay 28 combinaciones distintas de errores de dos bits en un dato de ocho bits. Como puede observarse en la Fig. 4, se corrige el 25%, es decir, solo 7 de esos 28 casos. Efectivamente, solo hay 7 combinaciones de errores dobles adyacentes en un dato de 8 bits. Dado que el código está preparado para corregirlos, son esas las combinaciones que suponen el 25% en la gráfica de errores aleatorios y, a la vez, suponen el 100% de errores adyacentes de dos bits (ver Fig. 5).

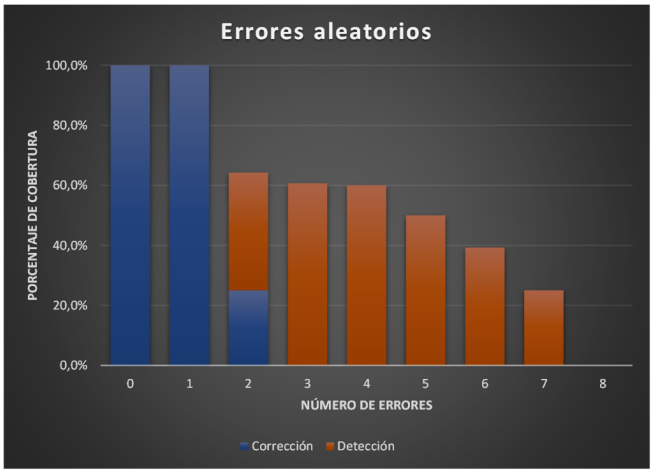


Fig. 4. Cobertura de corrección y detección de errores aleatorios.

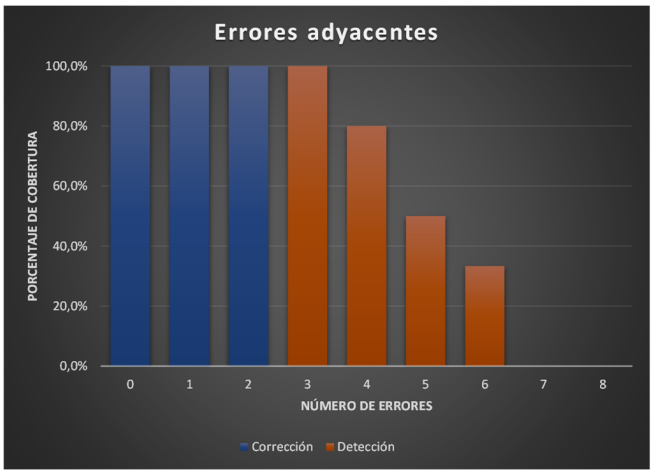


Fig. 5. Cobertura de corrección y detección de errores adyacentes.

Los errores triples adyacentes se detectan al 100%, pero solo se detecta el 61% si se consideran todos los posibles errores triples. A partir de aquí, se producen porcentajes de detección descendentes conforme aumenta el número de errores. Obviamente, el código no está diseñado para encontrarse con esas situaciones, por lo que es un comportamiento esperado.

En definitiva, el análisis de cobertura del código SEC-DAEC-TAED (8, 3) permite determinar que cumple con los objetivos para los que se ha diseñado.

B. Puesta en marcha y análisis de comportamiento

Para comprobar el comportamiento de los vehículos y su comunicación en ausencia/presencia de fallos, se han diseñado una serie de pruebas. Mediante el circuito de la Fig. 6, se analiza el comportamiento del sistema en función del código corrector de errores implementado en cada momento, y del tipo de errores inyectados. En este circuito, los dos vehículos autónomos siguen la línea (este comportamiento ya está incluido en su programación), mientras el vehículo director gestiona el ritmo de marcha (marcha o paro, adelante o atrás, y velocidad) del vehículo seguidor, a través de los comandos enumerados anteriormente.

Las pruebas consisten en enviar, de forma cíclica, las cuatro instrucciones secuencialmente, con un lapso de tres segundos entre comandos. Una vez enviado el cuarto comando el proceso vuelve a empezar desde el primero, repitiendo el ciclo (ver Fig. 7). De esta forma, se comprueban las instrucciones individualmente, con tiempo suficiente para ver el comportamiento en el circuito propuesto. Esta secuencia de instrucciones permite, además, evitar que el vehículo seguidor (más rápido) alcance al vehículo director, por las configuraciones de cada vehículo y la manera en la que toman las curvas. El comportamiento correcto del sistema puede verse en el vídeo disponible en este enlace: https://youtu.be/ymK4kNL-A9E.

Se han realizado diversas campañas de inyección de fallos sobre las comunicaciones entre ambos vehículos, para todas las configuraciones y códigos analizados. En los casos en que los fallos inyectados entran dentro de las coberturas de corrección de cada código, los errores se han corregido, interpretando correctamente las instrucciones. En los casos en que son detectados, pero no corregidos, se ha solicitado el reenvío de la instrucción, con lo que se puede recuperar si no se producen errores posteriores. En ambos casos, el comportamiento es correcto. Sin embargo, las inyecciones de fallos no cubiertos por cada código causan que los comandos no se interpreten correctamente, provocando un inadecuado comportamiento, como el que puede verse en este vídeo: https://youtu.be/TFcBIAZURxE.

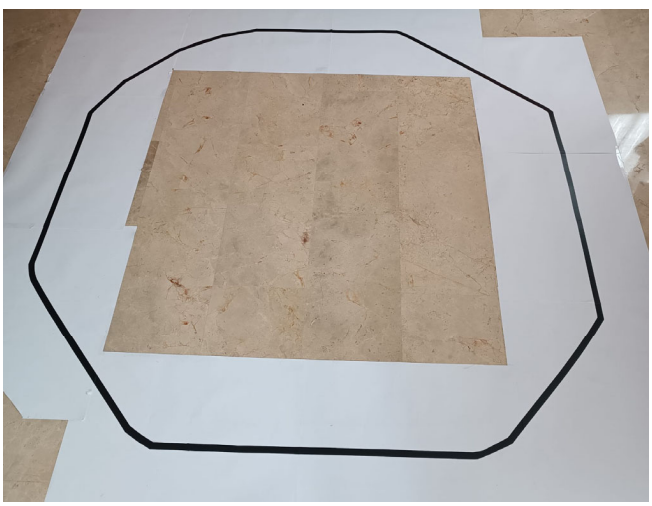


Fig. 6. Circuito de pruebas.

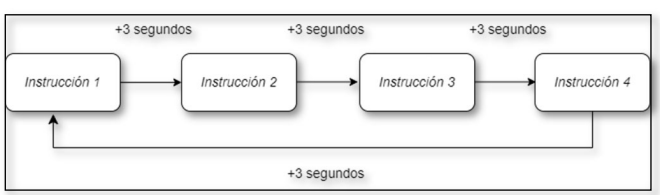


Fig. 7. Ciclo de envío de comandos.

IV. Evaluación del Sistema Empotrado

A. Análisis del tamaño del código

La Tabla II muestra los resultados obtenidos para las dos versiones comentadas anteriormente, es decir, sin implementar el codificador, protegiendo de esta forma únicamente el dato durante su transmisión, e implementando el codificador para proteger los comandos tanto de fallos en la memoria como en el canal de comunicaciones. La primera fila de esta tabla muestra el tamaño del software sin ningún tipo de ECC implementado.

TABLA II
TAMAÑO DEL SOFTWARE (EN BYTES)

ECC IMPLEMENTADO	VEHÍCULO 1				VEHÍCULO 2			
	SIN CODIFICADOR		CON CODIFICADOR		SIN CODIFICADOR		CON CODIFICADOR	
	SOFTWARE	DATOS	SOFTWARE	DATOS	SOFTWARE	DATOS	SOFTWARE	DATOS
SOFTWARE SIN PROTEGER	8870	647	--	--	6270	354	--	--
HAMMING (8, 4)	9112	723	11314	1262	8220	1100	8476	1100
HSIAO (8, 4)	9152	763	10146	1014	9660	920	9864	906
SEC-DAEC-TAED (5, 3)	9154	765	10128	1007	9522	919	9664	905

TABLA III
TIEMPO DE EJECUCIÓN (EN MILISEGUNDOS)

ECC IMPLEMENTADO	VEHÍCULO 1		VEHÍCULO 2	
	SIN CODIFICADOR	CON CODIFICADOR	SIN CODIFICADOR	CON CODIFICADOR
SOFTWARE SIN PROTEGER	12	--	32	--
HAMMING (8, 4)	14	23	180	183
HSIAO (8, 4)	12	22	144	150
SEC-DAEC-TAED (5, 3)	14	23	153	156

Como se puede ver, el software sin codificador ocupa menos espacio que el software con codificador. Este es un resultado esperado, pues al añadir el codificador se añade software (las fórmulas necesarias para aumentar la tolerancia a fallos del sistema).

Particularizando este análisis, vemos que los diferentes ECC implementados en el vehículo 1 (vehículo maestro) presentan un tamaño muy parecido, tanto en el tamaño del software como en los datos necesarios para su correcta ejecución. Al añadir el codificador, crecen las diferencias. En concreto, el código SEC-DAEC-TAED es el que menos memoria ocupa, aun presentando la mayor capacidad de corrección de errores (aunque para menos bits de datos).

Con respecto al vehículo 2 (vehículo esclavo), presenta unos resultados más variados que el vehículo 1. Esto se debe a que este vehículo (en ambas versiones, con y sin codificador) tiene que decodificar más comandos que el vehículo 1. Además, también implementa un diálogo algo más complejo que el primer vehículo.

En general, podemos ver que el tamaño del software en ambos vehículos no tiene problemas de espacio, lo que permitiría añadir más sensores o ECC con una mayor tolerancia a fallos.

B. Análisis del tiempo de ejecución

La Tabla III muestra los tiempos de ejecución del software, calculados como la media de 20 ejecuciones del software completo.

Lo primero que podemos apreciar es la gran diferencia de tiempos que hay entre los dos vehículos. Esto se debe principalmente a dos causas. La primera es el tiempo que el segundo vehículo debe esperar la transmisión del comando por parte del vehículo maestro.

La segunda razón es debida al tiempo de procesamiento de los comandos por parte del segundo vehículo. Hay que recordar que el sentido de la comunicación es principalmente del vehículo maestro al vehículo esclavo (es mayor la información que va del vehículo 1 al vehículo 2). Sólo cuando el segundo vehículo detecta algún fallo, el primer vehículo ejecuta el decodificador para comprobar que la petición de reenvío del último comando es correcta.

Aun así, vemos que el tiempo de ejecución es muy bajo en ambos vehículos, lo que permite utilizar los ECC en este tipo de comunicaciones sin que el sistema pierda funcionalidades ni se retrase la respuesta de ambos vehículos.

V. CONCLUSIONES

En este trabajo se ha estudiado la protección de las comunicaciones entre dos vehículos autónomos mediante ECC. Hemos podido comprobar que esta protección es viable. Si bien los ECC utilizados tienen una tolerancia a fallos más bien simple, este trabajo es el primer paso de cara a estudiar el uso de ECC más complejos, y con una mayor cantidad de datos a transmitir.

Aunque los ECC implementados en este trabajo fueron diseñados originalmente para su implementación en hardware con el fin principal de proteger los datos almacenados en memoria, hemos podido comprobar que es posible realizar una protección software de las comunicaciones, con una sobrecarga asumible.

El siguiente paso será probar con sistemas más complejos, que incluyan más sensores y actuadores, y con una mayor cantidad de datos a transferir.

AGRADECIMIENTOS

Proyecto PID2020-120271RB-I00 financiado por MCIN/AEI /10.13039/501100011033

REFERENCIAS

[1] The International Technology Roadmap for Semiconductors 2015. [Online]. Disponible en: http://www.itrs2.net/itrs-reports.html.

[2] I. Koren and C. Krishna, "Fault-Tolerant Systems", 2nd Edition, Elsevier Science, 2020.

[3] E. Fujiwara, Code Design for Dependable Systems: Theory and Practical Application, Ed. Wiley-Interscience, 2006.

[4] R. W. Hamming, "Error detecting and error correcting codes," Bell System Technical Journal, vol. 29, pp. 147–160, 1950.

[5] C.L. Chen and M.Y. Hsiao, "Error-correcting codes for semiconductor memory applications: a state-of-the-art review", IBM Journal of Research and Development, vol. 58, no. 2, pp. 124–134, March 1984.

[6] L. Saiz-Adalid et al., "Modified Hamming Codes to Enhance Short Burst Error Detection in Semiconductor Memories", 2014 Tenth European Dependable Computing Conference (EDCC), pp. 62-65, 2014.

[7] J. Gracia-Moran, L.J. Saiz-Adalid, D. Gil-Tomás, and P.J. Gil-Vicente, "Improving Error Correction Codes for Multiple Cell Upsets in Space Applications", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26(10), pp. 2132-2142, October 2018.

[8] A. Sánchez-Macián, P. Reviriego, J. Tabero, A. Regadío and J.A. Maestro, "SEFI protection for Nanosat 16-bit Chip On-Board Computer Memories", IEEE Transactions on Device and Materials Reliability, vol. 17(4), pp. 698-707, December 2017.

[9] D.C.C. Freitas et al., "Error Coverage, Reliability and Cost Analysis of Fault Tolerance Techniques for 32-bit Memories used on Space Missions", 2020 21st International Symposium on Quality Electronic Design, pp. 250-254, March 2020.

[10] J. Gracia-Morán, P. Martín-Tabares, and L.J. Saiz-Adalid, "Estudio del impacto de la inclusión de Códigos Correctores de Errores en un Sistema Empotrado", V Jornadas de Computación Empotrada y Reconfigurable (JCER2020/2021), Jornadas SARTECO 20/21, pp. 647-651, Septiembre 2021.

[11] J. Gracia-Morán and L.J. Saiz-Adalid, "Análisis del impacto de la inclusión de Códigos Correctores de Errores en un Sistema Empotrado basado en Arduino", VI Jornadas de Computación Empotrada y Reconfigurable (JCER2022), Jornadas SARTECO 2022, pp. 713-718, Septiembre 2022.

[12] J. Lázaro Tarazón, "Diseño e implementación mediante aplicaciones CAD e impresión 3D de un coche a escala guiado por Arduino", Trabajo Final de Grado, Universitat Politècnica de València, 2021, http://hdl.handle.net/10251/170866

[13] R. García Armero, "Desarrollo de un prototipo a escala de un vehículo para pruebas de laboratorio mediante impresión 3D", Trabajo Final de Grado, Universitat Politècnica de València, 2021, http://hdl.handle.net/10251/173301

[14] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule", IEEE Trans. Electron Devices, vol. 57, no. 7, pp. 1527–1538, Julio 2010.

[15] R. W. Hamming, "Error detecting and error correcting codes". Bell System Technical Journal, vol. 29, pp. 147–160, 1950.

[16] M.Y. Hsiao, "A class of optimal minimum odd-weight column SECDED codes," IBM Journal of Research and Development, vol. 14, no. 4, pp. 395–401, July 1970.

[17] Luis-J. Saiz-Adalid et al., "Flexible Unequal Error Control Codes with Selectable Error Detection and Correction Levels", 32th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2013), pp. 178-189, 2013.

[18] G. Tsiligiannis et. al., "Multiple Cell Upset Classification in Commercial SRAMs", IEEE Transactions on Nuclear Science, vol. 61, no. 4, August 2014.

[19] M. Wirthlin, D. Lee, G. Swift, and H. Quinn, "A method and case study on identifying physically adjacent multiple-cell upsets using 28-nm, interleaved and SECDED-protected arrays," IEEE Transactions on Nuclear Science, vol. 61, no. 6, pp. 3080–3087, Dec. 2014.

Evaluación de la robustez de una red neuronal desarrollada para generar un acelerador HW

Juan Carlos Ruiz García[1], David de Andrés Martínez[1] y Joaquín Gracia Morán[1]

Resumen—
El uso combinado de lenguajes de programación de alto nivel con herramientas de automatización de diseño electrónico, facilita el desarrollo de modelos de redes neuronales sintetizables sobre lógica programable. Aunque específicos, los aceleradores HW así producidos pueden optimizarse para ofrecer un buen balance entre prestaciones, área de silicio ocupada y consumo. Esto los hace especialmente interesantes en ámbitos donde los recursos disponibles son limitados, como en el IoT o la automoción. Sin embargo, su interés final puede variar en cada contexto atendiendo a los requisitos que se impongan en materia de seguridad funcional y que fijarán el grado de robustez esperado del modelo. Este artículo propone una metodología de evaluación, mediante inyección de fallos, que permite determinar la robustez de una red neuronal frente a alteraciones accidentales de sus pesos y sesgos. El modelo considerado es el de una red neuronal convolucional de tipo Lenet-5 implementada en C que posee un error de predicción del 1.83 %. Los resultados muestran que, en ocasiones, la modificación de un único bit en uno de los pesos/sesgos considerados puede llevar a la red a realizar predicciones incorrectas. No obstante, se constata que, en líneas generales, su naturaleza estocástica permite a la red tolerar el efecto de la mayor parte de los fallos inyectados, con un impacto mínimo, aunque no despreciable, en su precisión, y por tanto, en su seguridad funcional.

Palabras clave— **Evaluación de robustez, Inyección de fallos, Redes Neuronales, Modelos HW comportamentales.**

I. Introducción

Nuestra dependencia de la tecnología crece a medida que los dispositivos electrónicos que nos rodean se hacen más inteligentes y se interconectan para ofrecer servicios de mayor valor añadido. El aumento de sus capacidades de procesamiento ha permitido a estos dispositivos tratar (parte de) sus datos y tomar (algunas) decisiones localmente. Esta sinergia entre inteligencia artificial y computación en la frontera está impulsando avances tecnológicos significativos en áreas como la automatización industrial, la atención médica inteligente o la conducción autónoma [1].

El desarrollo de sistemas inteligentes en la frontera plantea desafíos a la hora de lograr un equilibrio óptimo entre rendimiento y eficiencia [2]. Por una parte, es necesario ajustar el tamaño de los modelos de red neuronal considerados a la memoria disponible. Por otra parte, se debe optimizar su eficiencia computacional, sin perder demasiada precisión, y minimizar su consumo de energía.

Para abordar estos desafíos, muchos sistemas empotrados integran aceleradores hardware optimizados para cálculo tensorial [3]. Descartando las NPUs (del inglés *neuronal processing units*) y GPUs (del inglés *graphics processing units*) por su alto consumo energético, estos aceleradores pueden ser TPUs (del inglés *tensor processing units*) o estar implementados sobre lógica programable (FPGAs, del inglés *field-programmable gate arrays*). Las TPUs ofrecen un buen balance entre rendimiento y eficiencia energética, pero carecen de flexibilidad para adaptarse a la características de cada red neuronal. Por otro lado, las FPGAs ofrecen dicha flexibilidad, pero a costa de implementar aceleradores HW más específicos.

Las herramientas de automatización de diseño electrónico (*Electronic Design Automation (EDA)* en inglés) ocultan parte de esta complejidad al desarrollador, al menos durante las primeras etapas del diseño de la red neural. Estas herramientas permiten generar modelos descritos en lenguajes de descripción de hardware (*Hardware Description Languages (HDL)* en inglés) a partir de programas desarrollados en lenguajes de alto nivel. Luego, estos modelos HDL se implementan sobre lógica programable, para producir un acelerador específico, o sobre cualquier otro tipo de acelerador hardware disponible [4].

La mayor parte de las herramientas EDA permiten producir diseños que optimicen consumo, área ocupada y/o frecuencia de funcionamiento. Sin embargo, cuando el diseño tiene requisitos de seguridad funcional, la evaluación de su robustez resulta también primordial [5]. Estándares como el IEC 61508 [6] o el ISO 26262 [7] recomiendan afrontar esta evaluación utilizando técnicas de inyección de fallos.

Este artículo aborda la evaluación de la robustez de una red neuronal frente a fallos de memoria que afecten a sus pesos y sesgos. Los retos tratados son:

- ¿Qué fallos inyectar y por qué?
- ¿Qué metodología de inyección de fallos utilizar?
- Y, ¿Cómo se ven afectadas las salidas de la red?

La sección II enmarca este trabajo en un esfuerzo por ofrecer aceleradores HW confiables para redes neuronales. El procedimiento de inyección de fallos propuesto se detalla en III , el caso de estudio en IV y los resultados en V. Finalmente, la contribución concluye en VI.

II. Evaluación de la robustez de aceleradores HW

Un acelerador HW implementado en lógica programable se puede comenzar a desarrollar utilizando lenguajes de programación de alto nivel. Estos lenguajes permiten expresar de manera eficiente la lógica y el flujo de datos de las redes neuronales, pero no permiten capturar y explotar el paralelismo existente en el HW en el que dichas redes se ejecutarán.

[1]Instituto ITACA, Universitat Politècnica de València, e-mail: jcruizg, ddandres,jgracia}@disca.upv.es.

El flujo de diseño *semicustom* establece que, en primer lugar, el programa de la red neuronal debe convertirse, siguiendo un proceso de síntesis de alto nivel [8], en un modelo HDL, lo que significa que se genera una representación del mismo a nivel de red de puertas lógicas. Finalmente, esta representación se hace corresponder con los elementos físicos de la tecnología de implementación y se definen las interconexiones entre dichos elementos. El modelo de implementación resultante acelera la ejecución de la red original al poderse ejecutar en hardware. La implementación final de este modelo para lógica programable se denomina *bitstream*.

La evaluación de la robustez de una red neuronal desarrollada tal y como hemos descrito requerirá del estudio del comportamiento en presencia de fallos de los distintos modelos intermedios que el flujo de diseño *semicustom* contempla generar. Sin embargo, el nivel de detalle entre modelos es tal que, en la práctica, no es viable cubrirlos todos con ayuda de una única metodología de inyección de fallos.

El proceso de inyección de fallos más rápido, y que ofrece resultados más representativos, es el que se lleva a cabo sobre el modelo de implementación de la red [9]. Sin embargo, la complejidad de estos modelos es tan grande que su simulación conlleva tiempos de experimentación prohibitivos. Por ello, la inyección de fallos suele acelerarse mediante la ejecución de los modelos sobre hardware reconfigurable, lo que se conoce como emulación de fallos [10]. El problema de la emulación de fallos es que requiere establecer una correspondencia biunívoca entre los elementos especificados a nivel comportamental, que constituyen el objetivo de los fallos a inyectar, y sus homólogos a nivel estructural [11], algo que no siempre se puede conseguir. En ausencia de esa correspondencia, la emulación de fallos debe hacerse "a ciegas" lo que dificulta considerar fallos más allá del simple bitflip en la memoria de configuración de la FPGA [12].

Inyectar fallos en el programa de la red neuronal, o en su modelo HDL [13], es temporalmente menos costoso. En el caso del programa, la inyección se realizaría al 100 % vía software, mientras que en el caso de los modelos HDL se usarán técnicas de simulación de fallos [14]. Sin embargo, la inyección a estos niveles tan abstractos plantea dudas sobre la representatividad de los modelos utilizados, lo cuál es un problema cuando esas dudas se trasladan a los resultados de las campañas de inyección [15]. La gran ventaja es que, al trabajar con los primeros modelos de la red disponibles, es posible obtener estimaciones tempranas de la robustez de la red en desarrollo, lo que reduce el coste de corregir los problemas que se encuentren o de plantear alternativas de diseño.

Más allá de los pros y contras de la inyección de fallos en cada tipo de modelo, el objetivo es el de enriquecer el flujo de diseño *semicustom* a todos los niveles con herramientas aptas para la evaluación de la robustez de los diseños utilizados. En este artículo nos centramos en el primero de estos diseños, el programado utilizando un lenguaje de alto nivel, por ser éste un nivel normalmente obviado por los desarrolladores SW, que no suelen interesarse en el desarrollo de aceleradores HW, y poco tratado por los diseñadores HW, que prefieren partir de modelos HDL tradicionales para el diseño de sus aceleradores.

III. Metodología de inyección de fallos

Comenzaremos precisando cuál es el modelo de red neuronal considerado (III-A), y continuaremos detallando qué fallos vamos a inyectar sobre dicho modelo (III-B), cómo lo vamos a hacer (III-C) y finalmente, cuál es el procedimiento que seguiremos para coordinar los procesos de inyección y observación que son necesarios de cara a obtener trazas de la ejecución de la red considerada (III-D).

A. Modelo de red neuronal considerado

Con independencia de su función concreta, una red neuronal es siempre una estructura de procesamiento de información [16] que consiste en una serie de nodos interconectados (*neuronas*) organizados en capas. La primera capa es la capa de entrada y es la que recibe los datos de entrada. A medida que los datos se propagan a través de la(s) capa(s) intermedia(s), las neuronas realizan cálculos matemáticos (normalmente de multiplicación y acumulación de resultado) para procesar la información y generar salidas. La última capa, conocida como capa de salida, produce los resultados finales. Durante el entrenamiento, se ajustan los pesos y sesgos de las conexiones entre las neuronas para reducir el error del modelo, es decir para mejorar su capacidad de proporcionar respuestas correctas para los datos de entrada.

La inferencia es lo que sigue al entrenamiento, y permite utilizar el modelo ya entrenado para realizar predicciones o tomar decisiones basadas en nuevos datos de entrada. Para ello, el proceso de inferencia se sustenta en el uso de las conexiones, los pesos y los sesgos "aprendidos" durante el entrenamiento de la red. Por tanto, el valor de esos pesos y sesgos resulta crítico para el correcto funcionamiento de la red, y por ello, serán el objetivo de los fallos que vamos a considerar y que pasamos a detallar.

B. Modelos de fallo y objetivos de la inyección

Por cuestiones de eficiencia, los pesos y sesgos de las red neuronales suelen almacenarse en la memoria durante la ejecución de la red neuronal. Sin embargo, es bien sabido que la reducción del tamaño y del consumo de las celdas y chips de memoria incrementan la sensibilidad de dichas memorias a fenómenos ligados, entre otros, al estrés térmico, al ruido electromagnético o en situaciones especiales al efecto de las radiaciones ionizantes en los semiconductores [17].

Estos fallos tienen potencial para poder afectar significativamente al comportamiento de una red neuronal. De hecho, cuando una celda de memoria se ve afectada por un fallo, la integridad del bit del dato almacenado se puede ver comprometida, lo que puede generar un efecto en cascada que se propage por las distintas capas de la red neuronal y acabe afectando

a sus salidas. Las consecuencias pueden ser asumibles si la predicción que finalmente emite la red no se ve alterada, o no serlo si la predicción resulta incorrecta.

Los modelos de fallo son representaciones simplificadas de los fenomenos que acabamos de describir. En el caso de las memorias estos modelos suelen ser de naturalez accidental aunque, por su prevalencia, pueden clasificarse como fallos transitorios o permanentes [18]:

- Los *bit-flips* son fallos transitorios que ocurren cuando se produce un cambio en el estado de un bit en la memoria, lo que puede dar lugar a datos incorrectos en la información almacenada. Estos errores pueden ser causados por diversas razones, como la radiación cósmica, la interferencia electromagnética o las imperfecciones en los componentes hardware. Por su naturaleza, estos fallos ocurren, pero no prevalecen, con lo que cualquier reescritura de la celda de memoria afectada los eliminará.
- Los fallos de *pegado a 0 o a 1* son fallos de naturaleza permanente. Esto significa que un fallo de pegado a 0/1 (*stuck-at 0/1* en inglés) nunca desaparece. Cuando estos fallos ocurren afectan a una celda de memoria, y por tanto a un bit de algún valor almacenado, haciendo que se mantenga permanentemente en un estado bajo/alto, o cero/uno, sin importar la entrada que apliquemos. Estos fallos pueden ser causados por fenómenos de desgaste y electromigración que pueden afectar a las celdas de memoria a lo largo del tiempo. El desgaste se refiere al deterioro gradual de las celdas debido al uso y a la exposición a factores ambientales. Por otro lado, la electromigración es un fenómeno relacionado con el movimiento de los átomos en los conductores, causado por el flujo de corriente, lo que puede provocar el debilitamiento o la interrupción de las conexiones eléctricas.

La multiplicidad de los fallos es también otro factor a tener en cuenta ya que, cada vez con mayor frecuencia y, debido al tamaño tan reducido de las celdas de memoria, el número de celdas que se pueden ver afectadas por un fallo puede ser superior a uno. Por ello en este artículo consideramos, además de fallos simples, tanto fallos dobles como fallos triples adyacentes. En el primer caso, fallo doble adyacente, el bit-flip, o pegado a 0/1, afecta a 2 bits del dato almacenado en memoria. En el segundo caso, el número de bits afectados es 3.

La tabla I sintetiza todo lo dicho en este punto. Una vez definidos los fallos que se van a considerar abordemos ahora la problemática de cómo emular sus efectos en la red neuronal.

C. Metodología de inyección

A la hora de modificar un bit en un peso o un sesgo de los utilizados en la red neuronal hay que resolver básicamente tres problemas:

1. Recuperar el valor (peso/sesgo) a modificar;
2. Realizar la modificación de dicho valor en base al tipo de fallo a inyectar;
3. Actualizar el valor original con el nuevo valor “inyectado”;

Para dar una solución al primero de los problemas hay que entender que estos pesos/sesgos están almacenados en vectores en memoria. El número de dimensiones de estos vectores depende de multiples factores que están más allá del interés de esta contribución, pero lo importante es que con independencia de sus dimensiones, los vectores se almacenan en memoria de manera lineal, es decir, localizando una fila tras otra y, para una fila dada, un elemento tras otro. Por tanto, si los accesos se programan en el inyector de forma lineal todos los vectores, con independencia de su número de dimensiones y el valor de las mismas, podrán ser tratados de la misma forma. La idea es que todos los vectores sean manipulados como vectores unidimensionales y con un número de elementos igual al producto de todas las dimensiones del vector original. La figura 1 muestra la definición de un vector de pesos y dos ejemplos de funciones en C que se podrían implementar para poder realizar un acceso lineal a los elementos de dicho vector tratándolo como un vector unidimensional, gracias al uso de un puntero al primer elemento del vector y de desplazamientos relativos al mismo.

```
float KERNEL_CONV_2[CONV2_KERNELS][CONV1_KERNELS][CONV2_KERNEL_HEIGHT][CONV2_KERNEL_WIDTH]

int getTensorLength(TensorID id) {
        switch (id) {

            case CONV_2_KERNELS:
                    return CONV2_KERNELS * CONV2_KERNEL_WIDTH * CONV2_KERNEL_HEIGHT;
                    break;

float* getTensorPTR(TensorID id){
        switch (id) {
            case CONV_2_KERNELS:
                    return &(KERNEL_CONV_2[0][0][0][0]);
```

Fig. 1: Funciones para el acceso lineal a un vector en C

Una vez resuelto el problema de acceso a los vectores que almacenan los pesos y sesgos de la red, el siguiente paso es definir una función para modificar el bit (o bits) del valor objetivo y hacer dicha modificación atendiendo al tipo de fallo a inyectar. Para ello utilizaremos una máscara con un número de bits igual o superior al número de bits del elemento a modificar. Obviamente este número de bits variará de una red a otra en función de si la red trabaja con pesos y sesgos en coma flotante de simple o doble precisión o con valores enteros, algo común cuando la red se cuantiza.

La máscara que se utilizará indicará con un ‘1’ lógico en un bit que el bit correspondiente del dato objetivo debe modificarse, y con un ‘0’ que no se debe alterar. En el caso de un bit-flip, la operación a aplicar será una XOR lógica entre la máscara y el valor objetivo de la inyección. Para un pegado a 0, será una AND lógica y para un pegado a 1, se aplicará una OR lógica. La especificación de las máscaras concretas a utilizar en cada experimento es uno de los parámetros de entrada del procedimiento experimental que se detallará en el apartado III-D de esta sección.

Tabla I: Resumen de fallos considerados, multiplicidad de los fallos y objetivos de las inyecciones

Modelos de fallo	Prevalencia	Multiplicidad de los fallos	Objetivos de la inyección
Bit-flip	Transitorio	Simple	Pesos y sesgos de la red neuronal
Pegado a 0 Pegado a 1	Permanente Permanente	Simple, Doble y Triple adyacente	

La figura 2 proporciona la implementación de una función en C que puede servir para inyectar fallos en datos de entrada de tipo entero con signo de 8 y 32 bits, de tipo entero sin signo de 32 bits, o en números en coma flotante de 32 bits. Por otro lado, la máscara que la función aplica es de 32 bits y como puede apreciarse, el valor del elemento es leído y transformado sistemáticamente en un entero sin signo de 32 bits antes ser manipulado utilizando la máscara. Así, una vez modificado, el valor "'inyectado' es finalmente promocionado nuevamente a su tipo de origen. Estas promociones de tipo son necesarias para que las operaciones lógicas puedan aplicarse bit a bit.

```
void inject(
        InjectionElement* injectionElement,
        FAULT_TYPE faultType,
        uint32_t mask
){
        uint32_t* ui32PTR;
        int8_t* ui8PTR;
        uint32_t valToSet;
        switch((*injectionElement).elementType) {
                case INT8:
                        ui8PTR = (int8_t*) (*injectionElement).element;
                        valToSet = *ui8PTR;
                        break;
                default:
                        ui32PTR = (uint32_t*) (*injectionElement).element;
                        valToSet = *ui32PTR;
                        break;
        }
        if (DEBUG) printf("> injectFault on value %X  \n", valToSet);
        switch(faultType){
                case NO_FAULT:
                        break;
                case BITFLIP:
                        if (DEBUG) printf("> flipping bits\n");
                        valToSet = valToSet ^ (mask);   //flipping bits
                        break;
                case STUCK_AT_0:
                        if (DEBUG) printf("> stucking at 0 bits\n");
                        valToSet = valToSet & ~(mask);  // clearing bits
                        break;
                case STUCK_AT_1:
                        if (DEBUG) printf("> stucking at 1 bits\n");
                        valToSet = valToSet | (mask);   // setting bits
                        break;
        }
        if (DEBUG) printf("> value after injection %X \n", valToSet);
        switch((*injectionElement).elementType) {
                case FLOAT:
                        *(float *) (*injectionElement).element = (float) valToSet;
                        break;
                case UINT32:
                        *(uint32_t *) (*injectionElement).element = (uint32_t) valToSet;
                        break;
                case INT32:
                        *(int32_t *) (*injectionElement).element = (int32_t) valToSet;
                        break;
                case INT8:
                        *(int8_t *) (*injectionElement).element = (int8_t) valToSet;
                        break;
                case UNKNOWN:
                        break;
        }
}
```

Fig. 2: Ejemplo de función para la inyección de fallos en C

En la figura 2 se observa también que el dato objetivo de la inyección es suministrado en una estructura que contiene un puntero al elemento a modificar. Por tanto, al estar referenciado mediante un puntero, cuando se actualiza el valor del dato objetivo lo que realmente se está actualizando es el vector de pesos o sesgos que se desea modificar. Esta aproximación resuelve el tercer problema que se había planteado anteriormente y que consiste en actualizar el valor original del elemento objetivo por el valor resultante del proceso de inyección.

D. Procedimiento experimental

La lanzadera es el punto de entrada al inyector y es la encargada de proporcionar al coordinador todos los parámetros necesarios para configurar un experimento de inyección (ver figura 3). Como se aprecia en la figura 4 la lanzadera permite especificar el tipo de fallo a inyectar, el fichero de máscaras a utilizar, los vectores objetivo de la inyección, los parámetros de entrada para la red neuronal (rango de imágenes y fichero con las imágenes en el ejemplo), así como el fichero de etiquetas para dichos parámetros:

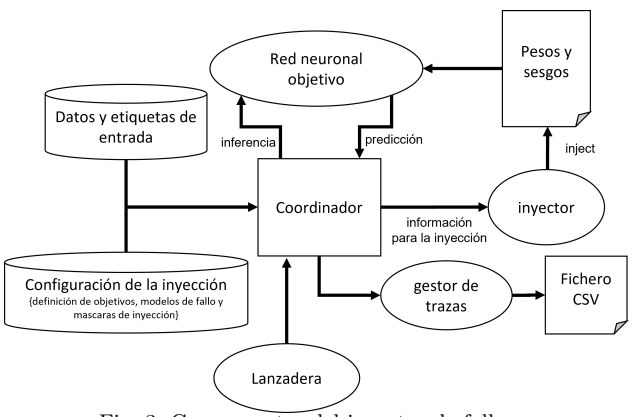


Fig. 3: Componentes del inyector de fallos

- Los tipos de fallos a inyectar son los introducidos en el punto III-B con la salvedad de permitir adicionalmente inhibir la inyección de un fallo en un experimento. Esto permite realizar experimentos en los que la red neuronal objetivo simplemente realice una inferencia con los datos de entrada proporcionados. La traza de ejecución que así se obtenga consituirá una ejecución de referencia, o *golden run*, que permitirá posteriormente determinar si los experimentos con fallos inyectados difieren o no de los experimentos en los que no se inyectó ningún fallo. La figura 5 muestra un extracto del fichero CSV de salida que genera el inyector, y en el que, como puede observarse, se refleja: el identificador de la entrada considerada (un identificador de imagen en nuestro caso), el tensor objetivo (0 cuando no lo hay), el elemento del tensor sobre el que se inyecta (XX si no se hace), la máscara utilizada, el fallo introducido (NF significa No Fallo), la salida que proporciona la red neuronal, la salida de la etapa softmax (etapa de clasificación final existente en muchas redes neuronales,ver Figura IV), la predicción ofrecida por la red, y la etiqueta, es decir, la respuesta correcta, y esperada, para la entrada considerada. En el ejemplo concreto de la figura 5, la entrada que se muestra es para la imagen 123 que representa un 6

```
minilenet_fi FaultType MaskFile TensorMin:TensorMax ImagesFile ImageMin:ImageMax LabelsFile

        FaultType           > NF | BF | S0 | S1
        MaskFile            > Default: ../data/mask_singlefault_all32bits.data
        TensorMin:TensorMax > Range of tensors (from 1 to 9 for minilenet)
                              0 stands for NO_TENSOR
                              1 stands for CONV1_WEIGHTS
                              2 stands for CONV1_BIAS
                              3 stands for CONV2_WEIGHTS
                              4 stands for CONV2_BIAS
                              5 stands for FC1_WEIGHTS
                              6 stands for FC1_BIAS
                              7 stands for FC2_WEIGHTS
                              8 stands for FC2_BIAS
        ImagesFile           > Default: ../data/input_images_file.plain
        ImageMin:ImageMax   > Range of images (from 1 to 10000 for the default imnages file)
        LabelsFile          > Labels file
*****************
        Examples:
                > Inject Bifflips using the info of default Maskfile in Tensors 1 to 2
                  using default image file and images from 123 to 145
                  [./minilenet_fi BF ../data/singlefault_all32bits.mask 1:2 ../data/input_images_file.plain 123:145 ../data/input_images_file.labels]
                > Inject double adjacent Bifflips using the masks specified in ../data/doubleadjacentfaults_all32bits.mask
                  in tensors using default image file and images from 123 to 145
                  [./minilenet_fi BF ../data/doubleadjacentfaults_all32bits.mask 1:2 ../data/input_images_file.plain 123:145 ../data/input_images_file.labels]
                > Execute GOLDEN RUN use file ../data/goldenrun_all32bits.mask for no Tensor
                  using default image file and considering images from 123 to 145
                  [./minilenet_fi NF ../data/goldenrun_all32bits.mask 0:0 ../data/input_images_file.plain 123:145 ../data/input_images_file.labels]
```

Fig. 4: Lanzamiento del inyector desde consola

A	B	C	D	E	F	G	H	I
IMGID	TENSORID	ELEMID	MASK	FAULT	OUTPUT	SOFTMAX	PRED	LABEL
123	0	XX	XXXXXXXX	NF	4.0031299591,0.0076000765,-3.1213324070,-0.0000027854,0.0000000512,0.0000000022		6	6

Fig. 5: Información recabada en la traza de un experimento *golden run*

(valor de la columna LABEL) y para la que la red neuronal ha predicho que es un 6 (valor de columna PRED). Cabe señalar que actualmente el inyector asume que el comportamiento del proceso de inferencia de la red neuronal es determinista, es decir que con los mismos estímulos de entrada la predicción que emitirá la red será siempre la misma.

- Como ya se avanzó en el apartado III-C, las máscaras, y más concretamente, los ficheros de máscara, forman también parte de los parámetros de entrada al procedimiento experimental que estamos describiendo. La figura 6 muestra un extracto de un fichero de máscara diseñado para inyectar errores triples adyacentes en los 32 bits de cada uno de los datos considerados. Cada entrada del fichero de máscara tiene el formato P:XX:YYY, donde P indica que la máscara se especifica por posición, XX la posición del bit a partir del cual se aplica la máscara, e YYY es la máscara a aplicar especificada en binario. En el caso de la entrada P:01:111 se está indicando que se debe inyectar el modelo de fallos considerado (bit-flip o pegado a 0/1) a partir del bit 1 de cada elemento y hasta contar 3 bits. Por tanto, el proceso de inyección alterará los bits 1, 2 y 3 de todos los elementos de todos los vectores que se especifiquen. Para inyectar un fallo sólo en el bit 12 de los datos, y sólo ese, especificaríamos la máscara P:12:1, y para inyectar un error doble adyacente en los bits 7 y 8 de los datos, la máscara sería P:7:11. Es cierto que la inyección podría aplicarse también sobre bits no consecutivos o elegidos aleatoriamente, pero ese tipo de fallos no han sido considerados en este artículo. Por ejemplo, la máscara P:4:101101, indica que los bits 4, 6, 7 y 9 de los datos a manipular son los que deben verse afectados por los fallos considerados.
- A la hora de especificar el objetivo de la inyección se define un rango de tensores utilizando la nomenclatura TX:TY, lo que indica que uno tras otro el valor de todos los elementos de los tensores especificados en el rango se irán alterando de acuerdo al tipo de fallo y máscara especificados. Para que esto sea posible se ordenan todos los tensores (vectores que contienen pesos o sesgos) y a cada uno se la asocia un valor Z de tal manera que TZ identifique a cada tensor durante los experimentos de inyección. Esto queda claro si observamos el ejemplo que muestra la figura 4.

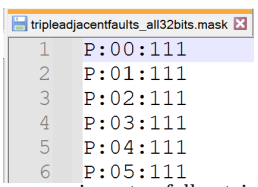


Fig. 6: Máscaras para inyectar fallos triples adyacentes

- La especificación de los parámetros de entrada de la red es un aspecto delicado y que es difícil tratar de manera genérica. En su versión actual, el inyector procesa redes neuronales de tipo convolucional y por ello los parámetros de entrada que se consideran son: un fichero de imágenes y un rango de imágenes que dentro de dicho fichero identifican a las imágenes a procesar.
- Finalmente, también hay que suministrar un fichero de etiquetas. Las etiquetas son los oráculos de las predicciones que realiza la red neuronal. Para cada imagen, el coordinador del inyector recibe una etiqueta que refleja, como ya se ha mencionado, la traza de salida. Así a posteriori podemos saber si una predicción incorrecta de la red viene motivada por el fallo inyectado o si la red simplemente falla por la imprecisión inherente a su proceso de inferencia.

Volviendo a la figura 3, y con toda la información proporcionada por la lanzadera, el coordinador lee los ficheros suministrados, extrayendo de los mismos

```
69          Args* argstyped = (Args*)additionalArgs;
70          ExperimentArgs* expArgs = &(argstyped->expArgs);
71          InjectionArgs* injArgs = &(argstyped->injArgs);
72
73          if ((expArgs->imageIndexMin <= imageIndex) && (imageIndex <= expArgs->imageIndexMax)){
74                  //for (int img=fieargs.imageIndexMin; img<= fieargs.imageIndexMax; img++){
75                  (*injArgs).imageIndex = imageIndex;
76                  if ((*injArgs).faultType!=NO_FAULT) {
77                          for (int tensor = (*expArgs).tensorIdMin; tensor <= (*expArgs).tensorIdMax; tensor++){
78                                  (*injArgs).tensorID = getTensorID(tensor);
79                                  TensorInfo tensorInf = getTensorInfo((*injArgs).tensorID);
80                                  for (int elem = 0; elem < tensorInf.length; elem++){
81                                          (*injArgs).element = elem;
82                                          float* ePtr = getPointerToTensorElement(tensorInf, (*injArgs).element);
83                                          for (int m=0; (m < expArgs->masks.length); m++){
84                                                  (*injArgs).mask = ((uint32_t*)expArgs->masks.data)[m];
85                                                  float valBeforeInjection = *ePtr;
86                                                  inject(ePtr, (*injArgs).faultType, (*injArgs).mask);
87                                                  //inference section begins
88                                                  InferenceResult result = inference(image);
89                                                  //inference section ends
90                                                  printOutputs((*injArgs), result, labels);
91                                                  setTensorElement(tensorInf, (*injArgs).element, valBeforeInjection);
92
93                                          }
94                                  }
95                          }
96                  }else{
97                          //inference section begins
98                          InferenceResult result = inference(image);
99                          //inference section ends
100                         printOutputs((*injArgs), result, labels);
101                 }
102         }
```

Fig. 7: Ejemplo de coordinador de experimentos de inyección

los datos concretos de entrada a la red, su etiquetas y las máscaras de inyección a utilizar durante el experimento. Con esto y con la información suministrada por la lanzadera, se elije un tensor y un elemento del tensor y se remite toda la información al *inyector*, que es el encargado de emular el fallo. Una vez se ha inyectado el error, se informa al coordinador que lanza el proceso de inferencia para el dato de entrada en la red neuronal objetivo. A continuación, se observan las salidas y el gestor de trazas genera el fichero CSV del experimento.

Este proceso experimental se repite para cada máscara especificada, para cada elemento de los tensores elegidos, y para cada una de las imágenes consideradas. La figura 7 proporciona un listado C con un ejemplo real de implementación de este proceso iterativo. Como puede observase en la línea 76, el tratamiento que aplica el coordinador será distinto dependiendo de si hay que inyectar fallos durante un experimento. La función *printOutputs* (ver líneas 90 y 100 del listado) es la función que implementa la obtención de trazas. Cabe también señalar que en la línea 91, el valor original del elemento que ha sido modificado por el fallo se restaura a su valor original sin fallo. Esto es hace así porque, de momento, el efecto acumulativo de los fallos no se está considerando. Por tanto, en cada experimento, sólo se inyecta un fallo, y su efecto en el dato alterado se elimina al final de cada experimento, antes de realizar una nueva inyección.

IV. Caso de estudio

La metodología de inyección propuesta se va a utilizar para evaluar la robustez de una implementación propia de la red neuronal Lenet-5 [19] y que denominaremos *Minilenet*. La arquitectrura de esta red es la que muestra la figura 8. El motivo que nos ha llevado a implementar esta red en C es la necesidad de disponer de modelos que puedan sintetizarse mediante HLS para posteriormente generar un acelerador hardware. El disponer del modelo original nos permitirá, en el futuro, adaptarlo para incluir mecanismos de tolerancia a fallos para hacer frente a los problemas que se detecten experimentalmente.

Como en el caso de Lenet-5, Minilenet es una red neuronal que hace uso de capas convolucionales y submuestreo (*max_pooling*) para el procesamiento y reconocimiento de imágenes. Más concretamente, la red implementa 2 etapas de convolución (que extraen 3 y 6 características de la imagen, respectivamente, mediante *kernels* 5x5), cada una con una función de activación de rectificación lineal (ReLU) conectada a una etapa de submuesteo, y 2 capas finales completamente conectadas. Con esta configuración la red actúa como un clasificador que identifica números manuscritos del 0 al 9, extraídos de la base de datos MNIST [20]. Su precisión es del 98.17 %.

Pasemos a analizar los resultados de la campaña de inyección llevada a cabo sobre este caso de estudio.

V. Experimentos y resultados

Tal y como muestra la tabla II, Minilenet hace uso de un total de 45079 pesos/sesgos que pueden verse afectados por fallos. Cada uno de estos valores se representa mediante un número en coma flotante de 32 bits, lo que resulta en casi 1.5 millones de bits que deben considerarse como posible objetivo de un fallo. Como la carga de trabajo ejecutada consta de 50 imágenes extraídas del repositorio MNIST [20], el total de experimentos a realizar asciende a más de 72 millones. Considerando que se han repetido estos experimentos para cada modelo de fallo considerado (bit-flip con multiplicidad simple, y pegado a 0 y pegado a 1, con multiplicidad simple, doble y triple), se han inyectado un total de 504 millones de fallos.

Fig. 8: Arquitectura de Minilenet

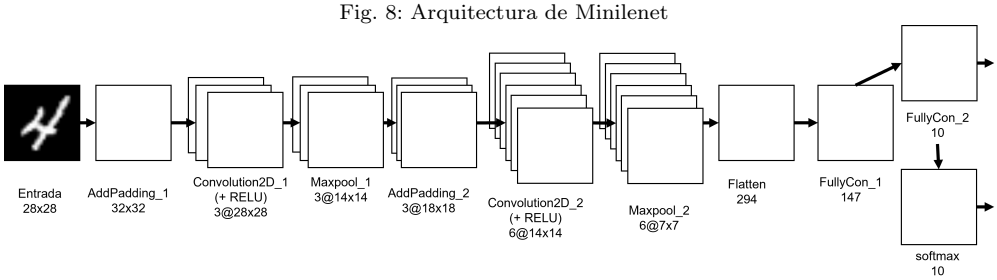


Tabla II: Inyecciones realizadas, en total, y por cada capa de Minilenet, para cada uno de los modelos de fallo considerados.

Etapa de Minilenet	Pesos/Sesgos	Inyecciones por imagen	Inyecciones totales (50 imágenes)
Pesos capa convolución 1	75	2400	120000
Sesgo capa convolución 1	3	96	4800
Pesos capa convolución 2	150	4800	240000
Sesgo capa convolución 2	6	192	9600
Pesos capa fully connected 1	43218	1382976	6914880
Sesgo capa fully connected 1	147	4704	235200
Pesos capa fully connected 2	1470	47040	2352000
Sesgo capa fully connected 2	10	320	16000
Total Minilenet	45079	1442528	72126400

Los resultados experimentales obtenidos se han procesado para clasificar las respuestas de Minilenet de acuerdo a los siguientes modos de avería:

- *Sin efecto*: La red clasifica correctamente la imagen y la probabilidad asociada a esta clasificación dista menos de 10 puntos porcentuales de la obtenida en ausencia de fallos.
- *SDC-10*: La red clasifica correctamente la imagen, pero la probabilidad asociada a esta clasificación dista entre 10 y 20 puntos porcentuales de la obtenida en ausencia de fallos.
- *SDC-20*: La red clasifica correctamente la imagen, pero la probabilidad asociada a esta clasificación dista más de 20 puntos porcentuales de la obtenida en ausencia de fallos.
- *Avería*: La red no clasifica la imagen correctamente.

Cabe señalar que la experimentación se ha llevado a cabo haciendo uso de una muestra de 50 imágenes para las cuales se sabía que Minilenet ofrecía, en ausencia de fallos, una predicción correcta.

Los diferentes modos de avería obtenidos para cada uno de los modelos de fallo considerados, tanto a nivel global de Minilenet, como en cada una de sus capas, se muestran en las Figuras 9, 10 y 11.

La robustez intrínseca de Minilenet frente a fallos de tipo pegado a 0, pegado a 1 y bit-flip es del 99.999 %, 98.334 % y 98.331 %, respectivamente, lo que equivale a una probabilidad de avería del 0.001 %, 1.666 % y 1.667 %, respectivamente. Estos datos reflejan la probabilidad de que el proceso de inferencia de Minilenet se vea alterado, o no, por los fallos introducidos. Las figuras 9, 10 y 11 proporcionan esta estimación de robustez en el caso general (ver entrada *Minilenet* en las figuras), y también pormenorizan ese valor para el caso de que el fallo inyectado afecte a cada una de las etapas de la red. Así, en el caso de los fallos de pegado a 1 y bit-flip, el 3.14 % y el 3,24 %, respectivamente, de los fallos inyectados sobre los pesos que utiliza la etapa de convolución 1 han provocado una avería. Sin embargo, el porcentaje de avería se reduce a sólo el 0.12 % en caso de que el fallo inyectado sea un pegado a 0.

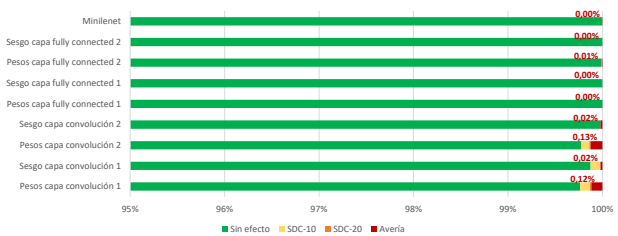


Fig. 9: Distribución de los modos de avería frente a pegado a 0 en cada capa de Minilenet y en su conjunto

Si estudiamos más en detalle la Figura 9, constatamos que el peor de los casos, es decir, si los fallos de pegado a 0 afectan a la etapa de convolución 2, el porcentaje de averías es del 0.13 % y el de SDC-10 de un 0.1 %. Esto parece indicar, por un lado, que la mayor parte de los bits de los pesos y sesgos ya están a '0' y, por tanto, no se ven afectados por estos fallos. Por otro lado, aquellos bits que se encuentran a '1' en los pesos y sesgos no ocupan posiciones relevantes dentro del formato del número en coma flotante y, por tanto, su cambio a cero resulta simplemente en una reducción/aumento de su valor que no afecta significativamente al conjunto de las operaciones en las que intervienen. Estas hipótesis podrán verificarse cuando estudiamos a continuación con mayor detalle, la distribución de las averías dependiendo del bit afectado.

En el caso de los fallos de tipo pegado a 1, se observa que sí que afectan en mayor grado al correcto funcionamiento de Minilenet, como muestra la Figura 10. Los elementos más sensibles son los pesos de

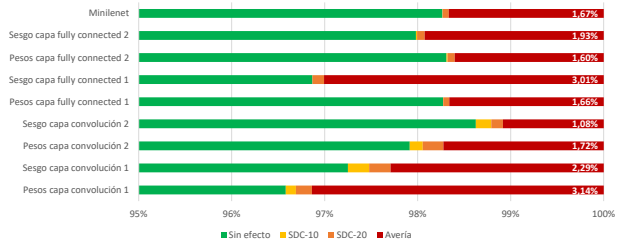


Fig. 10: Distribución de los modos de avería frente a pegado a 1 en cada capa de Minilenet y en su conjunto

la primera etapa de convolución y los sesgos de la primera etapa *fully connected*, que provocan un poco más del 3 % de averías en cada una de estas capas. Aquí pueden apreciarse otros modos de avería como SDC-10 y SDC-20, aunque con valores no superiores al 0.23 % para las capas más afectadas. Siguiendo el mismo razonamiento que en el caso anterior, si los fallos de tipo pegado a 1 sí consiguen afectar al funcionamiento de Minilenet, cabe suponer que los bits relevantes del formato de los números en coma flotante estaban almacenaban originalmente un '0', y que esto ha supuesto un incremento/decremento significativo en las operaciones involucradas.

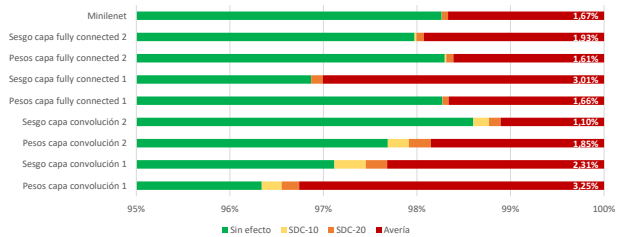


Fig. 11: Distribución de los modos de avería frente a bit-flip en cada capa de Minilenet y en su conjunto

Finalmente, como muestra la Figura 11, los fallos de tipo bit-flip son los más peligrosos para Minilenet, ya que siempre invierten el valor del bit afectado y, por tanto, aglutinan todos los casos efectivos contemplados para pegado a 0 y pegado a 1. A pesar de que, en primera instancia, pudiera parecer que los fallos que afecten a la última capa deberían presentar el mayor impacto, por proporcionar directamente la salida de Minilenet, parece que las probabilidades de clasificación obtenidas son tan distantes entre categorías que la red puede tolerar de manera intrínseca la mayor parte de estos fallos. Así, para Minilenet, los resultados parecen indicar que la primera etapa de convolución y la etapa *fully connected* son las más sensibles a los fallos inyectados, ya que los datos afectados se propagarán por el resto de capas del sistema y, posiblemente, puedan afectar a un mayor número de operaciones y modificar suficientemente las probabilidades de clasificación final.

Estas tendencias se mantienen en los resultados derivados de la inyección de fallos dobles y triples adyacentes. Al considerar fallos dobles, la robustez de Minilet es del 99.998 % y del 96.556 % para fallos de tipo pegado a 0 y pegado a 1, respectivamente, lo que supone doblar la probabilidad de avería para fallos de tipo pegado a 1. Estos números son prácticamente idénticos al considerar fallos triples, donde la robustez de Minilenet se reduce a 96.429 % para fallos de tipo pegado a 1 y se mantiene para fallos de tipo pegado a 0.

La Figura 12 muestra la distribución de las averías para Minilenet en su conjunto y para cada una de sus capas, atendiendo al bit que se ha visto afectado por el fallo. Para poder analizar la gráfica es preciso recordar el formato de los números en coma flotante de simple precisión (32 bits) IEEE-754 [21], donde el bit 31 representa el signo, los bits 30-23 representan el exponente (en exceso 127), y los bits 22-0 representan la mantisa (con un '1' implícito).

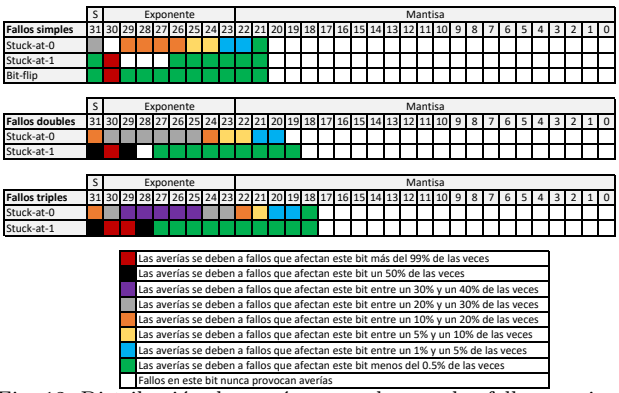


Fig. 12: Distribución de averías causadas por los fallos considerados en cada bit de los pesos/sesgos de Minilenet

Así, podemos observar cómo, para fallos simples, la práctica totalidad de las averías provocadas por fallos de tipo pegado a 1 y bit-flip derivan de la alteración del bit 30 del peso/sesgo considerado. Este bit es el de mayor peso del exponente y, si tenemos en cuenta que los bits 29-27 están a '1' (un pegado a 1 nunca provoca averías, pero sí un pegado a 0, entre un 15 % y un 20 % de las veces), el número representado cambia de 4.5E-5 a 1.4E+34, aproximadamente. Por ello, las operaciones involucradas se verán, con toda probabilidad, afectadas por el fallo y provocarán, con mayor facilidad, una clasificación errónea de la imagen.

En segundo lugar, el bit 31, que corresponde al bit de signo, podría parecer otro candidato importante a la hora de provocar averías. Sin embargo, los números reales representados no son lo suficientemente grandes como para provocar grandes variaciones y únicamente en 288 ocasiones llegan a provocar averías. En el caso de los fallos de tipo pegado a 0, que en conjunto provocan muy pocas averías, este es el bit más sensible.

En conjunto, puede observarse como únicamente el bit de signo, los bits del exponente y, en algunos casos, los dos primeros bits de la mantisa, llegan a provocar averías en Minilenet. Así, fallos simples que afecten a los bits 20-0 de la mantisa nunca provocarán averías.

Sin embargo, al considerar fallos múltiples, puede observarse cómo el bit de signo pasa a ser uno de los bits críticos, junto a los bits 30 y 29, especialmente para fallos de tipo pegado a 1. Esto se debe a que, ahora, no solo se está modificando el bit de signo,

sino también el bit adyacente, que corresponde al bit de mayor peso del exponente y, como se ha indicado anteriormente, esto provocará grandes desviaciones en el valor del peso/sesgo afectado.

Para fallos de tipo pegado a 1, el 28 sigue sin verse afectado por fallos dobles (al ser '1' y estar rodeado de bits a '1'), pero los fallos triples provocan que pase a ser uno de los bits más sensibles.

Los fallos de tipo pegado a 0 siguen provocando menor número de averías, pero la sensibilidad de los bits del exponente se ve afectada en mayor medida, ya que muchos de ellos están por defecto '1' y, aunque el modificarlos de forma aislada no presenta en muchos casos ningún efecto, al modificar varios de ellos en ráfaga sí tiene un impacto en la clasificación proporcionada.

Con esto, queda patente la capacidad de la red para tolerar, de forma intrínseca la ocurrencia de fallos simples, aunque existe al menos un 1.67 % de casos en los que el fallo derivará en una avería, lo que no es aceptable para sistemas críticos. Es más, estos fallos pueden acumularse en el tiempo, o pueden ocurrir de forma múltiple en el espacio, lo que incrementa notablemente su efecto en la clasificación (3.56 % de averías para fallos triples adyacentes). Por ello, deben definirse mecanismos adaptados a cada una de las redes neuronales diseñadas que permitan tolerar y, en la medida de los posible corregir, el efecto de estos fallos.

VI. Conclusiones

La naturaleza estocástica de una red neuronal le permite ser prácticamente immune a la ocurrencia de fallos de memoria que puedan afectar a sus pesos y sesgos. Los resultados experimentales avalan dicha afirmación, aunque también señalan ciertos cuellos de botella que pueden poner en tela de juicio el uso de este tipo de soluciones en la frontera.

Aunque es cierto que la robustez de la red es de prácticamente el 100 % frente a fallos permanentes de pegado a 0, esta robustez se ve reducida en un 1,67 % si el fallo es permanente de pegado a 1 o transitorio de tipo bit-flip. De hecho los resultados muestran que, para el caso de la red neuronal bajo estudio, el modelo de fallos bit-flip es suficiente para realizar una estimación global de su robustez, porque con los valores de sus pesos y sesgos, el efecto de los fallos permanentes es un subconjunto del efecto del bit-flip. Esto permite reducir de manera significativa del número de inyecciones a realizar. En el caso de nuestro caso de estudio, pasaríamos de 216 millones de inyecciones a sólo 72 millones en el caso de considerar sólamente fallos simples de tipo bit-flip sin reducir la representatividad de la conclusiones de nuestro estudio.

Estos resultados cobran gran relevancia cuando en lugar de pensar en fallos de naturaleza accidental, uno piensa en errores de memoria de tipo bit-flip inyectados maliciosamente. Tal y como hemos visto, bastará con modificar adecuadamente un único bit (de preferencia el 30) de un peso o sesgo para que, en determinadas circunstancias, la red neuronal se averíe, y siga así, mientras el ataque persista. Este es un aspecto complementario al presentado en este trabajo que se podría abordar de cara a futuro para, utilizando la metodología de inyección de fallos propuesta, identificar la debilidades de la red y definir metodologías de tolerancia, o al menos de mitigación, a posibles intrusiones.

Siguiendo con la vista puesta en el futuro, resulta también de interés el estudiar qué efecto puede tener en la robustez de una red neuronal convolucional el uso de mecanismos de optimización, como la cuantización. Este mecanismo permite a la red trabajar con pesos y sesgos de tipo entero en lugar de hacerlo con valores en coma flotante. Evidentemente, esto permite reducir las necesidades energéticas y de memoria de la red, así como acelerar el procesamiento de las entradas, al trabajar con artimética entera. Pero, ¿cómo afectará el cambio a la robustez de la red? Algo similar se plantea con otras optimizaciones, como la poda o el clustering.

En lo relativo a los modelos de fallo considerados, sólo hemos arañado la superfice, puesto que, como ya se ha comentado, no hemos tenido en cuenta el efecto acumulativo que los fallos permanentes pueden tener en el sistea, y por tanto, no hemos estudiado todavía las consecuencias que ese efecto puede tener sobre el comportamiento de la red neuronal. Además, hemos visto que el porcentaje de averías en caso de fallos múltiples adyacentes se incrementa hasta alcanzar el 3.56 %, pero ¿qué ocurriría si los fallos múltiples no son adyacentes y afectan simultáneamente a elementos distintos en una misma etapa o a etapas distintas de la red?.

Plantear estas preguntas nos permite avanzar, no sólo en la comprensión de cómo funcionan los algoritmos de aprendizaje profundo, sino también en el conocimiento de cómo fallan, algo de la máxima importancia para poder depositar una confianza justificada en su uso. Además todos estos estudios nos encaminan hacia la definición, implementación y verificación de mecanismos de detección, tolerancia y/o corrección de fallos (y ataques) que encontremos para permitir un uso confiable de las redes neuronales en entornos de funcionamiento hostil.

Agradecimientos

El presente trabajo ha sido realizado en el marco del proyecto PID2020-120271RB-I00 financiado por MCIN/AEI /10.13039/501100011033.

Referencias

[1] Saad Motahhir, *Smart Embedded Systems and Applications*, 08 2022.

[2] Tomas Bures, Danny Weyns, Bradley Schmer, Eduardo Tovar, Eric Boden, Thomas Gabor, Ilias Gerostathopoulos, Pragya Gupta, Eunsuk Kang, Alessia Knauss, Pankesh Patel, Awais Rashid, Ivan Ruchkin, Roykrong Sukkerd, and Christos Tsigkanos, "Software Engineering for Smart Cyber-Physical Systems: Challenges and Promising Solutions," *SIGSOFT Softw. Eng. Notes*, vol. 42, no. 2, pp. 19–24, June 2017.

[3] Luke Kljucaric and Alan D. George, "Deep-learning inferencing with high-performance hardware accelerators,"

ACM Trans. Intell. Syst. Technol., may 2023, Just Accepted.

[4] D. O'Loughlin, A. Coffey, F. Callaly, D. Lyons, and F. Morgan, "Xilinx vivado high level synthesis: Case studies," 01 2014, pp. 352–356.

[5] Gesina Schwalbe and Martin Schels, "A Survey on Methods for the Safety Assurance of Machine Learning Based Systems," in *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, Toulouse, France, Jan. 2020.

[6] Heinz Gall, "Functional safety IEC 61508 / IEC 61511 the impact to certification and the user," in *The 6th ACS/IEEE International Conference on Computer Systems and Applications, AICCSA 2008, Doha, Qatar, March 31 - April 4, 2008*. 2008, pp. 1027–1031, IEEE Computer Society.

[7] Mohamad Gharib, Andrea Ceccarelli, Paolo Lollini, and Andrea Bondavalli, "A cyber-physical-social approach for engineering functional safety requirements for automotive systems," *J. Syst. Softw.*, vol. 189, pp. 111310, 2022.

[8] Nan Wu, Hang Yang, Yuan Xie, Pan Li, and Cong Hao, "High-level synthesis performance prediction using gnns: Benchmarking, modeling, and advancing," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, New York, NY, USA, 2022, DAC '22, p. 49–54, Association for Computing Machinery.

[9] Ilya Tuzov, David de Andrés, and Juan Carlos Ruiz, "Simulating the effects of logic faults in implementation-level vital-compliant models," *Computing*, vol. 101, no. 2, pp. 77–96, 2019.

[10] David de Andrés, Juan-Carlos Ruiz-Garcia, Daniel Gil, and Pedro J. Gil, "Fault emulation for dependability evaluation of VLSI systems," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 4, pp. 422–431, 2008.

[11] Ilya Tuzov, David de Andrés, and Juan Carlos Ruiz, "Reversing FPGA architectures for speeding up fault injection: does it pay?," in *18th European Dependable Computing Conference, EDCC 2022, Zaragoza, Spain, September 12-15, 2022*. 2022, pp. 81–88, IEEE.

[12] Corbin Thurlow, Hayden Rowberry, and Michael Wirthlin, "Turtle: A low-cost fault injection platform for sram-based fpgas," in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2019, pp. 1–8.

[13] Juan Carlos Baraza, Joaquin Gracia, Sara Blanc, Daniel Gil, and Pedro J. Gil, "Enhancement of fault injection techniques based on the modification of VHDL code," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 6, pp. 693–706, 2008.

[14] Daniel Gil, Juan Carlos Baraza, Joaquín Gracia, and Pedro Joaquín Gil, *VHDL Simulation-Based Fault Injection Techniques*, pp. 159–176, Springer US, Boston, MA, 2003.

[15] Roberto Natella, Domenico Cotroneo, João Durães, and Henrique Madeira, "On fault representativeness of software fault injection," *IEEE Trans. Software Eng.*, vol. 39, no. 1, pp. 80–96, 2013.

[16] Yoshua Bengio, Yann LeCun, and Geoffrey E. Hinton, "Deep learning for AI," *Commun. ACM*, vol. 64, no. 7, pp. 58–65, 2021.

[17] Cristian Constantinescu, "Impact of deep submicron technology on dependability of VLSI circuits," in *2002 International Conference on Dependable Systems and Networks (DSN 2002), 23-26 June 2002, Bethesda, MD, USA, Proceedings*. 2002, pp. 205–209, IEEE Computer Society.

[18] Pedro Gil, Jean Arlat, Henrique Madeira, Yves Crouzet, Tahar Jarboui, Karama Kanoun, Thomas Marteau, Joao Duraes, Marco Vieira, Daniel Gil, Juan-Carlos Baraza, and Joaquín Gracia, "Fault Representativeness," Tech. Rep., Dependability Benchmarking project, 2002.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[20] Yann LeCun, Corinna Cortes, and CJ Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[21] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[22] Ana Pereira and Carsten Thomas, "Challenges of machine learning applied to safety-critical cyber-physical systems," *Machine Learning and Knowledge Extraction*, vol. 2, no. 4, pp. 579–602, 2020.

[23] ISO, "Road vehicles – Functional safety," 2011.

[24] Gregory A. Northrop and Pong-Fei Lu, "A semi-custom design flow in high-performance microprocessor design," in *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*. 2001, pp. 426–431, ACM.

[25] Issam W. Damaj, "High-level synthesis," *CoRR*, vol. abs/1905.02076, 2019.

[26] Annachiara Ruospo, Lucas Matana Luza, Alberto Bosio, Marcello Traiola, Luigi Dilillo, and Ernesto Sanchez, "Pros and cons of fault injection approaches for the reliability assessment of deep neural networks," in *2021 IEEE 22nd Latin American Test Symposium (LATS)*, 2021, pp. 1–5.

Uso de códigos de corrección de errores asimétricos en un sistema empotrado

J. Gracia-Morán*, J.C. Ruiz-García*, L.J. Saiz-Adalid*

***Resumen*—Durante estos últimos años, el desarrollo tecnológico ha propiciado un aumento en las prestaciones de los sistemas digitales, pero a costa de reducir su confiabilidad. Por ejemplo, y gracias a la continua reducción de tamaño de la tecnología CMOS, los sistemas de memoria proporcionan hoy en día una gran capacidad de almacenamiento a costa de un aumento en su tasa de fallos. Este progreso tecnológico también ha permitido el desarrollo de sistemas microprocesadores con una amplia gama de aplicaciones a un precio muy asequible, como pueden ser los sistemas basados en Arduino. Este tipo de sistemas soporta la conexión de una gran cantidad de sensores y actuadores, lo que permite crear una infinidad de sistemas complejos. Pero, ¿qué pasa si queremos utilizar un sistema basado en Arduino en un entorno crítico? ¿Cómo se protegen los datos de este sistema? Y dentro de un dato, ¿todos los bits erróneos producen los mimos efectos?**

Ampliamente usados, los códigos de corrección de errores (ECC) se utilizan para proteger sistemas de memoria. Generalmente, cuando se diseña un mecanismo de tolerancia a fallos basado en ECC, todos los bits de datos tienen la misma protección. Sin embargo, en algunas aplicaciones y entornos, esto puede no ser necesario. En este trabajo se presenta un ECC Asimétrico, es decir, un ECC capaz de ofrecer diferentes niveles de protección dentro de un dato.

Palabras Clave—Códigos de Corrección de Errores Asimétricos, Confiabilidad, Fallos Simples, Fallos Múltiples, Sistemas Empotrados

I. Introducción

El aumento en la escala de integración de la tecnología CMOS ha permitido aumentar las prestaciones de los sistemas digitales pero a costa de un incremento en su tasa de fallos [1]. Por ejemplo, en memorias, se pueden producir fallos simples o múltiples causados por el impacto de una partícula de radiación cósmica [2][3].

Un método común para proteger sistemas de memoria son los códigos de corrección de errores (ECC) [4], los cuales permiten una amplia variedad de cobertura de fallos [5][6][7][8][9][10].

En trabajos anteriores [11][12], estudiamos el impacto de la introducción de códigos de corrección de errores en un sistema empotrado. Si bien los ECC utilizados en estos trabajos están diseñados originalmente para su implementación en hardware, vimos que su implementación en software es viable y que, además, se puede conseguir una gran cobertura de fallos sin introducir una gran sobrecarga.

Un aspecto importante de los ECC utilizados en [11][12], y que suele ser habitual en este tipo de sistemas tolerantes a fallos basados en ECC, es que protegen a todos los bits por igual. Es decir, la protección frente a fallos se implementa suponiendo que todos los bits van a sufrir la misma tasa de fallos, o se asume que el efecto de los errores es igual de inocuo o dañino para el sistema, sea cual sea el bit afectado.

Sin embargo, no siempre es necesaria esta protección [13][14]. En determinadas aplicaciones, la diferencia entre un resultado obtenido con datos correctos y uno obtenido a partir de datos afectados por fallos en determinados bits puede ser asumible. En este sentido, el resultado obtenido, aunque no sea el esperado, puede ser tomado como correcto, no siendo considerado como una avería al no inducir en el sistema un comportamiento peligroso o indeseado.

Para este tipo de aplicaciones, una posible solución de tolerancia a fallos basada en ECC sería el uso de Códigos de Corrección de Errores Asimétricos [4][15]. Este tipo de ECC permite implementar diferentes niveles de control de errores dentro de una misma palabra de datos, de tal manera que algunas partes de dicha palabra estarán más protegidas que otras.

En el presente trabajo, hemos implementado y estudiado cómo afecta un código de corrección de errores Asimétrico en un sistema basado en Arduino [12]. En primer lugar, y utilizando la técnica de inyección de fallos, hemos estudiado cuál es el efecto de un fallo de tipo *bit-flip* en un determinado bit de las variables críticas del sistema. A partir de estos resultados, se ha diseñado un ECC Asimétrico utilizando la metodología presentada en [15]. Este tipo de ECC es capaz de dividir la palabra de datos en diferentes áreas, y proporcionar una protección frente a la ocurrencia de fallos distinta para cada una de estas áreas. Por último, hemos estudiado la sobrecarga en el tiempo de ejecución y en la memoria ocupada por la solución que implica el añadir estos ECC, y lo hemos comparado con algunos de los ECC utilizados en [12].

A continuación, la Sección II resume el sistema empotrado utilizado y los fallos más comunes en memoria. La Sección III muestra los resultados obtenidos al inyectar fallos en el sistema empotrado sin haber implementado ningún tipo de protección. La Sección IV describe el ECC Asimétrico presentado en este trabajo y resume la metodología empleada en su diseño, así otros ECC implementados para comparar. La Sección V presenta los diferentes resultados obtenidos durante la evaluación del sistema tolerante a fallos. La Sección VI resume las lecciones aprendidas con este trabajo. Y finalmente, la Sección VII concluye este trabajo.

II. Descripción del sistema utilizado

A. Sistema empotrado

Tal y como se describió en [12], el sistema usado está basado en la placa Arduino UNO R3 [23], al que le hemos añadido un módulo LCD1602 (pantalla LCD de 16 columnas y 2 filas), un sensor de temperatura y humedad DHT11 y un LED RGB, tal y como se puede ver en la Fig. 1.

* Instituto ITACA, Universitat Politècnica de València, 46022 Valencia, España.
e-mail: { jgracia, jcruizg, ljsaiz } @ itaca.upv.es

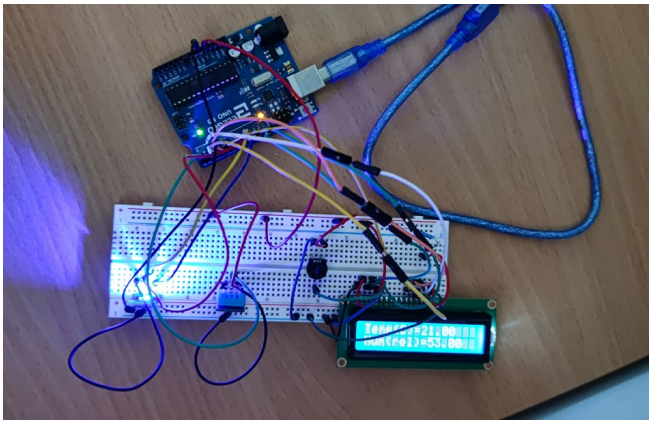


Fig. 1. Sistema empotrado utilizado.

El funcionamiento del sistema es sencillo. El sensor de temperatura y humedad proporciona datos de 32 bits en formato de coma flotante de simple precisión que hay que proteger. En concreto, se utiliza el formato IEEE754 de simple precisión [22], tal y como se puede ver en la Fig. 2. Estos valores se utilizarán para activar diferentes colores del LED RGB, y también se mostrarán en la pantalla LCD.

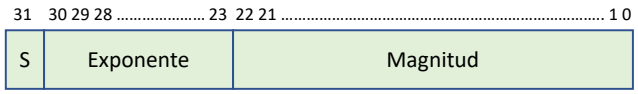


Fig. 2. Formato IEEE754 de simple precisión.

B. *Fallos considerados*

Mediante el uso de ECC, hemos protegido las variables críticas (que almacenan los datos proporcionados por el sensor de temperatura y humedad) contra la ocurrencia de fallos que pueden afectar al contenido de las celdas de memoria que las almacenan. Estos fallos pueden inducir errores *simples* o *múltiples*. El *error simple* afecta a un único bit, mientras que el *error múltiple* perturba a varios bits, que pueden ser *adyacentes* (bits contiguos), *no adyacentes,* o de *ráfaga* (afecta a un grupo de bits contiguos, de los que únicamente sabemos que el primer y el último bit son erróneos, pero no conocemos el estado de los bits intermedios).

III. Inyección de Fallos en el Sistema Empotrado sin Protección

Con el fin de comprobar los efectos de un fallo en un determinado bit, se han realizado varias campañas de inyección de fallos sistemática en las variables críticas del sistema [21]. Para ello, se han inyectado fallos simples y múltiples adyacentes (de hasta 8 bits) en todos los bits de las variables críticas.

Por cuestiones de espacio, hemos agrupado los resultados obtenidos en bytes. Las Tablas I y II muestran un resumen de los resultados producidos sin ningún tipo de protección implementado en el sistema empotrado. En concreto, se puede ver el rango de diferencias entre los valores medidos y el valor de referencia (sin fallos inyectados). Más que fijarnos en los valores concretos mostrados en las tablas, hay que ver la tendencia de estos. En ambas tablas, vemos el mismo comportamiento. A partir de un determinado bit, la diferencia entre el valor sin error y el valor con error se dispara. Y esta diferencia es mayor cuanto más significativo es el bit. Como se ha comentado, esto se debe a que el dato se almacena en memoria en el formato IEEE754 de simple precisión. Un cambio en los bits de menor peso implica cambios en los bits con menor influencia de la magnitud, lo que provoca cambios mínimos en los valores almacenados. Sin embargo, el error en los bits de mayor peso provoca enormes cambios en el valor de las variables críticas, sobre todo cuando se modifican los bits de mayor peso del exponente.

Hay que remarcar, especialmente, los errores múltiples que afectan al bit 29. En estos casos, el sistema se para, y no reacciona, teniendo que ser reiniciado (marcado con un (*) en las Tablas I y II).

TABLA I
DIFERENCIAS ENTRE EL VALOR CON ERROR Y EL VALOR SIN ERROR EN LA VARIABLE TEMPERATURA (SIN ECC)

Nº Fallos	31..24	23..16	15..8	7..0
1	-40 .. 10^{20}	-10 .. 0.1	0 .. 0.06	0
2	-20 .. 10^{15} (*)	-17.5 .. 0.4	0 .. 0.19	0
3	10^{4} .. 10^{18} (*)	0.9 .. 20	0 .. 0.44	0
4	10^{19} .. 10^{9} (*)	1.9 .. 10^{4}	0.01 .. 0.94	0
5	10^{18} (*)	1.9 .. 10^{8}	-0.06 .. 0.02	0 .. 0.01
6	(*)	-0.01 .. 10^{18}	0.03 .. 1.94	0 .. 0.02
7	(*)	5.9 .. 10^{18}	-2.06 .. 0.06	0 .. 0.03
8	(*)	-7.6 .. 10^{18}	-2.03 .. 5.94	0 .. 0.06

TABLA II
DIFERENCIAS ENTRE EL VALOR CON ERROR Y EL VALOR SIN ERROR EN LA VARIABLE HUMEDAD (SIN ECC)

Nº Fallos	31..24	23..16	15..8	7..0
1	-116 .. 10^{21} (*)	-16 .. 62	0 .. 0.13	0
2	-57 .. 10^{16} (*)	-4 .. 406	0 .. 0.38	0
3	-28.5 .. 10^{18} (*)	-28.5 .. 278	0.01 .. 0.88	0
4	10^{8} .. 10^{17} (*)	1.75 .. 10^{3}	-0.12 .. 0.94	0 .. 0.01
5	10^{18} (*)	-40.5 .. 10^{8}	0.03 .. 1.88	0 .. 0.02
6	(*)	-39 .. 10^{18}	-0.03 .. 5.88	0 .. 0.03
7	(*)	-37.5 .. 10^{18}	-2.12 .. 5.94	0 .. 0.06
8	(*)	-38 .. 10^{18}	-18.12 .. 5.97	0 .. 0.12

IV. Códigos de Corrección de Errores

A. *ECC Asimétrico*

Como hemos visto en la sección anterior, el efecto de un bit erróneo en el sistema depende de su posición. Los errores en los bits menos significativos introducen valores erróneos asumibles, mientras que los errores en los bits más significativos pueden llegar a causar la parada del sistema, algo inasumible en sistemas críticos.

Para solucionar estos efectos, en este trabajo hemos implementado un ECC Asimétrico [4][15] con tres zonas de protección (detección y corrección) diferenciadas. La primera zona la constituyen los 16 bits más significativos de las variables críticas protegidas, mientras que la segunda abarca sus 16 bits menos significativos. La tercera zona es la definida por los bits de paridad que el código añade. La matriz de paridad de este ECC se muestra a continuación:

$$\mathbf{H} = \begin{bmatrix} 10000000111111111111111100011101101000 11 \\ 01000000000000000000000001111001111011 00 \\ 00100000000000000000000011011100011100 01 \\ 00010000000000000000000011001011001001 10 \\ 00001000000000000000000001001111110010 01 \\ 00000100000000000000000001101110100100 10 \\ 00000010101010101010101010111111010001 00 \\ 00000001010101010101010101111101010001 1000 \end{bmatrix}$$

Para diseñar este ECC Asimétrico, hemos utilizado la metodología presentada en [15], y que se resume a continuación.

Tal y como se acaba de ver, un ECC se puede describir mediante su matriz de paridad **H**. El diseño y la búsqueda de una matriz **H** que cumpla con unos determinados requisitos de tolerancia a fallos pueden ser muy complejos. Nuestra metodología se basa en la búsqueda de patrones de error a corregir y/o detectar, los cuales se especifican mediante un conjunto dado de vectores de error. Esta metodología se ha utilizado tanto para diseñar ECC con distintas coberturas de fallos dentro de la misma palabra de datos (códigos FUEC [15]), como para crear diferentes familias de códigos (por ejemplo, [8][16][17][18][19][20]). A continuación, se describe brevemente la metodología utilizada, cuyo resumen se puede ver en la Fig. 3.

Una vez fijados el número de bits de la palabra de código (n) y de la palabra de datos (k), se seleccionan los patrones de error a detectar y/o corregir. Con estos datos, se inicia la búsqueda de la matriz de paridad **H** que cumpla que cada error corregible tenga un síndrome diferente, y cada error detectable genere un síndrome diferente a todos los síndromes generados por los errores corregibles

Utilizando el algoritmo de búsqueda recursivo de la Fig. 3b, se busca la matriz **H** que describa el código. Este algoritmo verifica matrices parciales y agrega una nueva columna solo si la matriz anterior satisface los requisitos. Existen $2^{(n-k)-1}$ combinaciones posibles para cada columna, pues las columnas agregadas deben ser distintas de cero. Se busca que las matrices tengan el menor número de unos (pues así se disminuye el área y/o el retardo). Por ello, primero se comprueban las columnas con un número menor de unos.

A partir de la matriz de paridad **H**, es bastante sencillo obtener las ecuaciones lógicas para los circuitos codificadores y el cálculo del síndrome en los circuitos decodificadores. Para relacionar cada síndrome con el vector de error correspondiente, se implementa también una tabla de búsqueda.

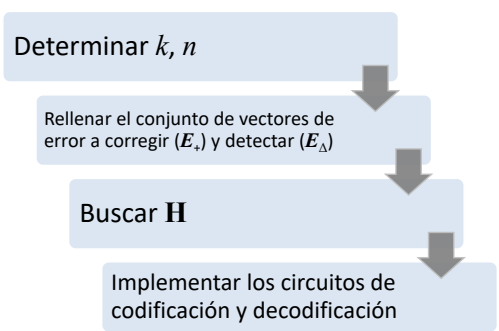
La especificación de las capacidades de tolerancia a fallos del ECC Asimétrico que se propone en este trabajo son las que se muestran en la Tabla III. La interpretación de la tabla debe hacerse por filas, es decir, que los eventos que se detallan en una fila ocurren simultáneamente. Como puede observarse, nuestro ECC es capaz de corregir:

- Dos errores simples, producidos uno en los bits de paridad y otro en los bits de mayor peso de la palabra de datos.
- Un error doble adyacente en los bits de mayor peso de la palabra de datos. Este conjunto de errores corregibles también incluye el error que afecta a los bits 16 y 15, es decir, el último más significativo y el primero menos significativo.
- Un error simple en los bits de mayor peso de la palabra de datos.

Por otro lado, nuestra propuesta también es capaz de detectar:

- Un error simple en los bits de paridad.
- Un error simple en los bits de menor peso de la palabra de datos.
- Un error doble adyacente en los bits de los bits de paridad.
- Dos errores simples, uno en los bits de paridad y otro en los bits de menor peso de la palabra de datos.
- Un error doble adyacente en los bits de menor peso de la palabra de datos.

Cabe destacar que todo error corregido es un error detectado, pero no todo error detectado puede ser corregido. Esto es algo importante a tener en cuenta a la hora de interpretar el contenido de las casillas en blanco de la Tabla III. Así, por ejemplo, el código propuesto detecta, pero no corrige, errores simples en los 16 bits menos significativos del dato protegido. Por otra parte, para poder corregir errores simples en los 16 bits más significativos de dicho dato debe, obviamente, poder detectarlos, aunque esto no aparezca explícitamente en la Tabla III.



(a)

```
Procedure CheckPartialMatrix partial_H (n-k) × ncols  /* ncols ∈ [1..n] */
    SyndromeSet = {}
    For each error vector e in E+
        partial_e = (e0, e1, ..., encols-1)
        If HammingWeight(e) = HammingWeight(partial_e)
            newSyndrome = CalculateSyndrome(partial_H × Transpose(partial_e))
            If newSyndrome in SyndromeSet then Return    /* Wrong partial matrix */
            Else Add newSyndrome to SyndromeSet
        End if
    End for
    For each error vector e in EΔ
        partial_e = (e0, e1, ..., encols-1)
        If HammingWeight(e) = HammingWeight(partial_e)
            newSyndrome = CalculateSyndrome(partial_H × Transpose(partial_e))
            If newSyndrome in SyndromeSet then Return    /* Wrong partial matrix */
            Else Do Nothing   /* newSyndrome not stored in this case */
        End if
    End for
    If ncols = n
        Add partial_H to SolutionsSet
        Return
    Else
        For each possible new_column    /* n- k bits, excluding the all 0 combination */
            CheckPartialMatrix[partial_H | new_column] (n- k) × (ncols+1)
        End for
    End if
End procedure
```

(b)

Fig. 3. Metodología de búsqueda de matrices de paridad basada en el conjunto de patrones de error a corregir y/o detectar: (a) Diagrama de flujo; (b) Algoritmo de búsqueda (extraído de [15]).

TABLA III
CAPACIDADES DE DETECCIÓN Y CORRECCIÓN DEL ECC ASIMÉTRICO PROPUESTO

CORRECCIÓN			
Bits de paridad	**Dato de 32 BITS**		
	16 bits más significativos	**16 bits menos significativos**	**Resultado**
Error simple	Error simple		Sin error en los bits de datos
	Error Doble Adyacente		Sin error en los bits de datos
	Error simple		Sin error en los bits de datos
DETECCIÓN			
Bits de paridad	**16 bits más significativos**	**16 bits menos significativos**	**Resultado**
Error simple			Sin error en los bits de datos
		Error simple	Error < 0.6% del bit implícito
Error Doble Adyacente			Sin error en los bits de datos
Error Simple		Error Simple	Error < 0.6% del bit implícito
		Error Doble Adyacente	Error < 0.6% del bit implícito

La última columna de la Tabla III muestra el resultado esperado en función del error que se haya producido. Es importante remarcar que, en principio, la detección de errores no se utiliza para implementar un mecanismo de reintento de la operación, sino para asegurar que esos errores no alteran de forma indeseada los bits más significativos en el proceso de decodificación (de forma similar a lo que sucede en el código SEC de Hamming cuando se produce un error doble). Más adelante se comenta la posibilidad de combinar ambas alternativas en función de los requerimientos del sistema.

Teniendo esto en cuenta, el valor "Error < 0.6% del bit implícito" que aparece en algunas filas de la detección se obtiene de la siguiente manera: dado que los datos a proteger son números reales, estos se almacenan memoria en variables de tipo *float* en C, siguiendo el estándar IEEE754 de simple precisión [22] (mostrado en la Fig. 2). Puesto que el código garantiza la integridad de los 16 bits de mayor peso, si se cumplen las hipótesis de fallo consideradas para el diseño, el mayor error no corregible sería un error doble adyacente que afectase a los bits 15 y 14 de la palabra de datos. Considerando que el bit 31 representa el signo, los bits 30 a 23 el exponente, y el resto de bits representan la mantisa normalizada, donde el bit implícito tiene valor 1, el bit 22 representa el valor ½, el 21 el valor ¼, y así sucesivamente con potencias negativas de 2. En el peor de los casos, el error introducido sería 1/256 + 1/512 ≈ 0.005859, o sea, inferior al 0.6%.

Es importante tener en cuenta que este porcentaje se calcula respecto del valor del bit más significativo del número representado, que será el bit implícito una vez normalizado. Dicho de otra forma, el valor del bit más significativo es $2^{exponente}$. Por ejemplo, si el valor es 12.5, en binario es 1100.1. El bit más significativo vale 8. En este caso, el error será inferior al 0.6% de 8.

En cuanto a las ventajas que presenta este tipo de ECC, la principal, sin duda, es la disminución de operaciones a realizar. La mayor parte de las operaciones de codificación y decodificación se concentran en los bits de mayor peso y en los de paridad, lo que provoca un menor número de operaciones globales, pues los bits de menor peso se protegen menos. De esta forma, se mejora la latencia general del ECC. Además, este menor número de operaciones también causa una menor sobrecarga en el espacio de almacenamiento necesario.

B. Otros Códigos de Corrección de Errores

En la Tabla IV se puede ver un resumen de algunos de los ECC implementados en [12], y que vamos a utilizar para comparar con el ECC Asimétrico propuesto en este trabajo. Con el fin de que esta comparación sea lo más completa posible, se han seleccionado varios ECC con diferentes propiedades de tolerancia a fallos.

Por un lado, tenemos el código SEC-DED-8AEC. Este ECC, que protege palabras de datos de 32 bits, está pensado inicialmente para reducir la latencia de la decodificación, a costa de una gran redundancia. El resto de ECC presentan una buena combinación entre redundancia y tolerancia a fallos para proteger palabras de datos de 16 bits.

V. EVALUACIÓN DEL SISTEMA EMPOTRADO TOLERANTE A FALLOS

A. Resultados de la inyección de fallos

Las Tablas V y VI muestran los resultados de los diferentes experimentos de inyección de fallos con las variables protegidas mediante el ECC Asimétrico. Al igual que las Tablas I y II, las Tablas V y VI muestran el rango de error cometido con respecto al valor de referencia (sin fallos inyectados). Como se puede ver, fallos simples y dobles adyacentes provocan diferencias mínimas en los bits de menor peso, errores que en este caso son asumibles. Incluso, se consigue superar el fallo doble adyacente que afecta al bit 29 (los casos en los que el sistema no responde están marcados también con un (*)).

TABLA IV
CAPACIDADES DE TOLERANCIA A FALLOS DE LOS ECCS IMPLEMENTADOS EN [12]

ECC	**Capacidad de Tolerancia a Fallos**	**Redundancia**
SEC-DED-8AEC [12]	**Corrección**: Errores de ráfaga de 8 bits; errores adyacentes de 8 bits **Detección**: Errores aleatorios de 2 bits; errores de ráfaga de 12 bits	32
FUEC-DAEC [12]	**Corrección**: Errores simples, errores dobles adyacentes **Detección**: Errores de ráfaga de 3 y 4 bits	7
FUEC-TAEC [12]	**Corrección**: Errores simples; errores dobles adyacentes; errores de ráfaga de 3 bits **Detección**: Errores de ráfaga de 4 bits	8
FUEC-QUAEC [12]	**Corrección**: Errores simples; errores dobles adyacentes; errores de ráfaga de 3 y 4 bits	9

TABLA V
DIFERENCIAS ENTRE EL VALOR CON ERROR Y EL VALOR SIN ERROR EN LA VARIABLE TEMPERATURA (ECC ASIMÉTRICO)

Nº Fallos	31..24	23..16	15..8	7..0
1	0	0	-0.06 .. 0.03	0
2	0	0	-0.05 .. 0.03	0
3	-25 .. 10^{23} (*)	-12 .. 10^{20}	0.03 .. 0.44	0
4	10^{9} .. 10^{19} (*)	-48 .. 10^{6}	0.01 .. 10^{11}	0
5	10^{18} (*)	-10^{8} .. 10^{3}	-18 .. 329.5	0
6	(*)	-24 .. 10^{18}	-24 .. 0.04	-0.01 .. 0
7	(*)	-20 .. 10^{18}	-21 .. 10^{11}	-0.01 .. 0.01
8	(*)	-24 .. 10^{8}	-56 .. 10^{11}	-0.02 .. 0.01

TABLA VI
DIFERENCIAS ENTRE EL VALOR CON ERROR Y EL VALOR SIN ERROR EN LA VARIABLE HUMEDAD (ECC ASIMÉTRICO)

Nº Fallos	31..24	23..16	15..8	7..0
1	0	0	0 .. 0.06	0
2	0	0	0 .. 0.09	0
3	-27 .. 10^{23} (*)	-14 .. 10^{20}	0.01 .. 0.44	0
4	10^{9} .. 10^{19} (*)	-47 .. 10^{6}	0.01 .. 10^{11}	0
5	10^{18} (*)	-10^{8} .. 10^{3}	-20 .. 404	0 .. 0.01
6	(*)	-27 .. 10^{18}	-27 .. 1	0 .. 0.02
7	(*)	-25 .. 10^{18}	-24 .. 10^{11}	0 .. 0.03
8	(*)	-27 .. 10^{8}	-56 .. 10^{11}	0 .. 0.06

En cuanto a fallos múltiples que afectan a un número mayor de bits, de tres en adelante, podemos ver que la variación es algo mayor con respecto a los casos sin protección del ECC. Esto es debido a que el síndrome que se genera en estos casos provoca una corrección en bits que no son erróneos, al ser errores no contemplados en las hipótesis de fallos. Si estos fallos se van a producir de forma habitual, habría que rehacer el ECC para contemplar estos casos.

B. Comparación con otros ECC

En las Tablas VII, VIII, IX, X, XI y XII podemos ver los resultados de la inyección de fallos en el sistema estando este protegido por los ECC FUEC-DAEC, FUEC-TAEC y FUEC-QUAEC. Como se ha comentado previamente, hemos inyectado hasta 8 fallos adyacentes, que es la máxima capacidad de tolerancia a fallos del código SEC-DED-8AEC. Esto hace que todos los fallos inyectados hayan sido tolerados por este último ECC.

Como se puede ver en estas tablas, todas siguen la misma tendencia mostrada en las Tablas V y VI. Todos los ECC toleran los fallos especificados en sus hipótesis de fallos (filas con todos sus valores a 0). También se puede observar que los errores son mayores cuanto más significativo es el bit afectado. De hecho, se puede ver en todos los casos que las diferencias entre el valor de referencia y el valor erróneo cuando el fallo afecta a los 8 bits menos significativos es casi despreciable.

TABLA VII
DIFERENCIAS ENTRE EL VALOR CON ERROR Y EL VALOR SIN ERROR EN LA VARIABLE TEMPERATURA (FUEC-DAEC)

Nº Fallos	31..24	23..16	15..8	7..0
1	0	0	0	0
2	0	0	0	0
3	10^{4} .. 10^{18} (*)	-26 .. 28	0.01 .. 0.11	0
4	10^{9} .. 10^{19} (*)	-26 .. 10^{4}	0.01 .. 0.88	0
5	-28 .. 10^{18} (*)	-23 .. 10^{28}	0.02 .. 1.88	0 .. 0.02
6	(*)	-26 .. 10^{28}	0.03 .. 3.88	0 .. 0.02
7	(*)	-23 .. 10^{28}	-0.12 .. 3.88	0 .. 0.02
8	(*)	-20 .. 10^{18} (*)	-8 .. 3.98	0 .. 0.06

TABLA VIII
DIFERENCIAS ENTRE EL VALOR CON ERROR Y EL VALOR SIN ERROR EN LA VARIABLE HUMEDAD (FUEC-DAEC)

Nº Fallos	31..24	23..16	15..8	7..0
1	0	0	0	0
2	0	0	0	0
3	10^{3} .. 10^{18} (*)	-32 .. 10^{2}	0 .. 0.22	0
4	10^{8} .. 10^{17} (*)	-30 .. 10^{3}	-0.25 .. 0.22	0 .. 0.01
5	10^{18} (*)	-44 .. 10^{28}	-0.25 .. 1.75	0 .. 0.05
6	(*)	-31 .. 10^{28}	-6.25 .. 0.25	0 .. 0.05
7	(*)	-37 .. 10^{27}	-2.25 .. 5.75	0 .. 0.05
8	(*)	-37 .. 10^{28} (*)	-16 .. 3.98	0 .. 0.12

TABLA IX
DIFERENCIAS ENTRE EL VALOR CON ERROR Y EL VALOR SIN ERROR EN LA VARIABLE TEMPERATURA (FUEC-TAEC)

Nº Fallos	31..24	23..16	15..8	7..0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	-29 .. 10^{19} (*)	-27 .. 10^{4}	0 .. 0.12	0
5	-29 .. 10^{21}	-20 .. 10^{18}	-0.12 .. 0.12	0
6	(*)	(*)	-0.12 .. 94.5	0 .. 0.06
7	(*)	(*)	-0.12 .. 94.5 (*)	0 .. 0.03
8	(*)	(*)	-0.01 .. 94.5 (*)	0 .. 0.06

TABLA X
DIFERENCIAS ENTRE EL VALOR CON ERROR Y EL VALOR SIN ERROR EN LA VARIABLE HUMEDAD (FUEC-TAEC)

Nº Fallos	31..24	23..16	15..8	7..0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	10^{8} .. 10^{17} (*)	-39 .. 10^{3}	0 .. 0.23	0 .. 0.01
5	-62 .. 10^{19} (*)	-49 .. 10^{6}	-0.25 .. 0.24	0 .. 0.01
6	(*)	(*)	-49 .. 0.25	0 .. 0.13
7	(*)	(*)	-44 .. 7.5 (*)	0 .. 0.12
8	(*)	(*)	-44 .. 8 (*)	0 .. 0.24

TABLA XI
DIFERENCIAS ENTRE EL VALOR CON ERROR Y EL VALOR SIN ERROR EN LA VARIABLE TEMPERATURA (FUEC-QUAEC)

Nº Fallos	31..24	23..16	15..8	7..0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	10^{18} (*)	-28 .. 10^{12}	0 .. 0.12	0 .. 0.05
6	(*)	-28 .. 10^{16}	-28 .. 0.12	0 .. 0.08
7	(*)	-24 .. 10^{18} (*)	-28 .. 0.12	0 .. 0.08
8	(*)	-10^{18} .. 10^{14} (*)	-16 .. 10^{5}	0 .. 0.08

TABLA XII
DIFERENCIAS ENTRE EL VALOR CON ERROR Y EL VALOR SIN ERROR EN LA VARIABLE HUMEDAD (FUEC-QUAEC)

Nº Fallos	31..24	23..16	15..8	7..0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	-60 .. 10^{18} (*)	-60 .. 10^{14}	0 .. 0.24	0 .. 0.11
6	(*)	-62 .. 10^{17}	-62 .. 0.25	0 .. 0.15
7	(*)	-10^{18} .. 10^{14} (*)	-60 .. 47	0 .. 0.16
8	(*)	-10^{18} .. 10^{14}	-60 .. 10^{7}	0 .. 0.16

Aquellas celdas marcadas con (*) significa que el sistema no respondía ante el fallo inyectado. En los experimentos de inyección de fallos realizados para esta sección, podemos comprobar que hay un número mayor de estos casos que utilizando el ECC Asimétrico. Esto es debido a que los ECC FUEC-DAEC, FUEC-TAEC y FUEC-QUAEC utilizan fórmulas más complejas tanto en la codificación como en la decodificación. Estas fórmulas hacen un uso más intensivo de los bits dañados, lo que provoca la no respuesta del sistema.

También hay que tener en cuenta que el código FUEC-DAEC detecta fallos triples adyacentes, y el ECC FUEC-TAEC detecta fallos cuádruples adyacentes. En estos casos, y aunque el resultado final no sea correcto, el sistema lo sabe. De esta forma, estos datos erróneos se pueden ignorar.

C. Análisis de la redundancia de los ECC

En la Tabla XIII se puede ver el número de bits de paridad (o bits redundantes) que utiliza cada ECC estudiado en este trabajo, así como la longitud de la palabra de datos que protegen. Como se puede ver, el menor número de bits de código es el del ECC Asimétrico introducido en este trabajo. El número de bits redundantes que utiliza un ECC es uno de sus parámetros más determinante, puesto que estos bits se añaden a cada palabra de datos cuando el ECC es implementado en hardware, por lo que interesa que su número sea bajo.

En este trabajo, los diferentes ECC se han implementado en software, y además, se protegen únicamente dos variables de 32 bits, lo que provoca que el número de bits de código no sea un parámetro tan crítico como sería en su implementación en hardware. Sí que habría que tener muy en cuenta este número de bits si el número de variables a proteger fuese mayor.

TABLA XIII
REDUNDANCIA DE LOS ECCS

ECC	Nº bits de datos	Nº bits de código
ECC Asimétrico	32	8
SEC-DED-8AEC [12]	32	32
FUEC-DAEC [12]	16	7
FUEC-TAEC [12]	16	8
FUEC-QUAEC [12]	16	9

D. Análisis del tamaño del código

Como se puede ver en la Tabla XIV, todos los codificadores ocupan la misma cantidad de memoria en todos los ECC estudiados. Esto se debe a:

i) Las fórmulas utilizadas para la codificación son relativamente sencillas;

ii) El entorno de programación de Arduino permite utilizar funciones para realizar operaciones de tipo bit de forma rápida y fácil.

En la decodificación, se pueden ver mayores diferencias. Por un lado, el menor tamaño ocupado se corresponde con el ECC Asimétrico y el FUEC-QUAEC. Esto se produce por:

i) Las fórmulas de decodificación de estos dos ECC son más sencillas que para el resto de los ECC.

ii) La detección de errores ocupa mucho espacio (se utilizan fórmulas más complicadas). Esto se puede ver en el código FUEC-QUAEC, que no tiene detección, por lo que su decodificador ocupa muy poco espacio. En el caso de nuestro ECC Asimétrico, las fórmulas de detección son más simples que las del resto de ECC con características de detección de fallos. Es por esta razón que el decodificador ocupa tan poco espacio.

En cuanto al tamaño global, vemos que sólo el FUEC DAEC presenta un tamaño menor que el ECC Asimétrico. Como se verá en el siguiente subapartado, este menor tamaño conlleva un mayor tiempo de ejecución.

E. Análisis del tiempo de ejecución

La Tabla XV muestra los tiempos de ejecución del software, calculados como la media de 20 ejecuciones del programa completo (adquisición de datos, codificación y decodificación de las variables críticas, activación del LED RGB y muestra de datos en la pantalla LCD), siguiendo el mismo proceso que en [12].

Con respecto al proceso de codificación, todos los ECC tienen un retardo menor de 0.3 ms, siendo la codificación más lenta la correspondiente al ECC Asimétrico. Esto es debido al hecho de tener el menor número de bits redundantes. En este caso, las operaciones son algo más complejas que las del resto de ECC, lo que conlleva un mayor tiempo en la codificación.

Esta mayor complejidad en el codificador queda compensada con el tiempo que se necesita en la decodificación. Tal y como se puede ver en la Tabla XV, el tiempo en la decodificación del ECC Asimétrico es el menor, junto con el del FUEC-QUAEC. De esta forma, el tiempo total de ejecución del sistema de control con el ECC Asimétrico es el menor de todos los sistemas protegidos, siendo apenas algo mayor de 1 ms con respecto al sistema sin proteger.

TABLA XIV
TAMAÑO DEL SOFTWARE (EN BYTES)

ECC Implementado	Codificación		Decodificación		Total Software	
	Software	Datos	Software	Datos	Software	Datos
Software sin Proteger	--	--	--	--	9554	399
ECC Asimétrico	444	9	444	9	14200	401
SEC-DED-8AEC [12]	444	9	12496	9	25324	399
FUEC-DAEC [12]	444	9	3050	9	13118	401
FUEC-TAEC [12]	444	9	4988	9	15070	401
FUEC-QUAEC [12]	444	9	444	9	16306	399

TABLA XV
TIEMPO DE EJECUCIÓN (EN MILISEGUNDOS)

ECC Implementado	*Codificación*	*Decodificación*	*Total Software*
Software sin Proteger	--	--	532.751
ECC Asimétrico	0.280	0.319	533.818
SEC-DED-8AEC [12]	0.216	1.713	536.509
FUEC-DAEC [12]	0.164	0.706	534.426
FUEC-TAEC [12]	0.169	0.923	534.919
FUEC-QUAEC [12]	0.165	1.076	535.240

VI. LECCIONES APRENDIDAS

Este trabajo es un claro ejemplo de lo que se denomina en inglés "*Approximate Fault Tolerance*" o "*Dependable enough*" [24]. Podemos definir estos conceptos como la capacidad de conseguir mejores prestaciones, con respecto al tiempo de ejecución y la energía consumida, a cambio de relajar la confiabilidad del sistema. En estos casos, se asume que existe una tolerancia a fallos inherente. Un ejemplo es el que hemos visto en este trabajo, en el que los errores no tratados en los bits de menor peso se manifiestan como una degradación de la precisión, o incluso no son apreciables.

La gran ventaja de este tipo de tolerancia a fallos es que se puede utilizar para lograr un bajo consumo de energía, una disminución de la latencia y un alto rendimiento, aspectos importantes en los sistemas empotrados.

Otra particularidad a tener en cuenta son las características del sensor de temperatura. En este trabajo se ha utilizado el sensor DHT11 [25], el cual permite una lectura de datos cada 2 segundos, por lo que no hay problemas de tiempo en la codificación/decodificación. Se podrían utilizar sensores más rápidos, como el SHT15 [26] o el MLX90614 [27]. El primero permite una lectura cada segundo, por lo que tampoco habría problemas para ejecutar el software asociado al ECC. En cambio, el segundo sensor permite una lectura cada 500 ms. En este caso, sí que es importante que el tiempo de ejecución del algoritmo de control sea lo más rápido posible, como ocurre con el ECC Asimétrico, el cual apenas añade 1 ms al algoritmo original.

En este segundo caso, incluso se podría cambiar la aproximación a la tolerancia a fallos. Al analizar el dato, si el error es asumible, se puede corregir mediante el ECC, mientras que si el error es mayor del tolerado, se podría tomar una nueva medida. Si esta nueva medida estuviese almacenada en una variable distinta, serviría también para tolerar fallos permanentes.

VII. CONCLUSIONES

En este trabajo se ha presentado un ECC Asimétrico que divide el dato a proteger en diferentes partes, aplicándole a cada parte una tolerancia a fallos distinta, y lo hemos implementado en un sistema empotrado basado en Arduino.

Para probar su funcionamiento, en primer lugar, hemos realizado una serie de campañas de inyección de fallos sistemática, con el fin de comprobar si el efecto de un error es el mismo, sea cual sea el bit afectado. Hemos comprobado que los bits de mayor peso del dato provocan errores más graves que los errores en los bits de menor peso. Es más, hemos visto que determinados errores no hace falta corregirlos, pues su efecto queda difuminado por el sistema. En un futuro, queremos probar con sistemas más complejos, que incluyan más sensores y actuadores, y aumentar las variables a proteger. Así, habría que analizar cómo afectaría al sistema la tolerancia a fallos para proteger este mayor número de variables. También seguiremos diseñando y probando otros ECC.

AGRADECIMIENTOS

Proyecto PID2020-120271RB-I00 financiado por MCIN/AEI /10.13039/501100011033

REFERENCIAS

[1] The International Technology Roadmap for Semiconductors 2015. [Online]. Disponible en: http://www.itrs2.net/itrs-reports.html.

[2] N.G. Chechenin and M. Sajid, "Multiple cell upsets rate estimation for 65 nm SRAM bit-cell in space radiation environment", 3rd International Conference and Exhibition on Satellite & Space Missions, Mayo 2017.

[3] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule", IEEE Trans. Electron Devices, vol. 57, no. 7, pp. 1527–1538, Julio 2010.

[4] E. Fujiwara, Code Design for Dependable Systems: Theory and Practical Application, Ed. Wiley-Interscience, 2006.

[5] R. W. Hamming, "Error detecting and error correcting codes," Bell System Technical Journal, vol. 29, pp. 147–160, 1950.

[6] C.L. Chen and M.Y. Hsiao, "Error-correcting codes for semiconductor memory applications: a state-of-the-art review", IBM Journal of Research and Development, vol. 58, no. 2, pp. 124–134, March 1984.

[7] L. Saiz-Adalid et al., "Modified Hamming Codes to Enhance Short Burst Error Detection in Semiconductor Memories", 2014 Tenth European Dependable Computing Conference (EDCC), pp. 62-65, 2014.

[8] J. Gracia-Moran, L.J. Saiz-Adalid, D. Gil-Tomás, and P.J. Gil-Vicente, "Improving Error Correction Codes for Multiple Cell Upsets in Space Applications", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26(10), pp. 2132-2142, October 2018.

[9] A. Sánchez-Macián, P. Reviriego, J. Tabero, A. Regadío and J.A. Maestro, "SEFI protection for Nanosat 16-bit Chip On-Board Computer Memories", IEEE Transactions on Device and Materials Reliability, vol. 17(4), pp. 698-707, December 2017.

[10] D.C.C. Freitas et al., "Error Coverage, Reliability and Cost Analysis of Fault Tolerance Techniques for 32-bit Memories used on Space Missions", 2020 21st International Symposium on Quality Electronic Design, pp. 250-254, March 2020.

[11] J. Gracia-Morán, P. Martín-Tabares, and L.J. Saiz-Adalid, "Estudio del impacto de la inclusión de Códigos Correctores de Errores en un Sistema Empotrado", V Jornadas de Computación Empotrada y Reconfigurable (JCER2020/2021), Jornadas SARTECO 20/21, pp. 647-651, Septiembre 2021.

[12] J. Gracia-Morán and L.J. Saiz-Adalid, "Análisis del impacto de la inclusión de Códigos Correctores de Errores en un Sistema Empotrado basado en Arduino", VI Jornadas de Computación Empotrada y Reconfigurable (JCER2022), Jornadas SARTECO 2022, pp. 713-718, Septiembre 2022.

[13] K. Huang et al., “Improve Robustness of Deep Neural Networks by Coding”, 2020 Information Theory and Applications Workshop (ITA), 2020, pp. 1-7, doi: 10.1109/ITA50056.2020.9244998.

[14] B. Masnick, J. Wolf, “On linear unequal error protection codes”, IEEE Transactions on Information Theory, vol. 13(4), pp. 600–607, Octubre 1967.

[15] Luis-J. Saiz-Adalid et al., “Flexible Unequal Error Control Codes with Selectable Error Detection and Correction Levels”, 32th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2013), pp. 178-189, 2013.

[16] J. Gracia-Moran, L.J. Saiz-Adalid, J.C. Baraza-Calvo, P.J. Gil-Vicente, “Correction of Adjacent Errors with Low Redundant Matrix Error Correction Codes”, 8th Latin-American Symposium on Dependable Computing (LADC 2018), pp. 107-114, October 2018.

[17] L.J. Saiz-Adalid, J. Gracia-Morán, D. Gil-Tomás, J.C. Baraza-Calvo, P.J. Gil-Vicente, “Ultrafast Codes for Multiple Adjacent Error Correction and Double Error Detection”, IEEE Access 2019, 7, 151131–151143, doi:10.1109/ACCESS.2019.2947315.

[18] J. Gracia-Morán, L.J. Saiz-Adalid, J.C. Baraza-Calvo, D. Gil-Tomás, P.J. Gil-Vicente, “Design, Implementation and Evaluation of a Low Redundant Error Correction Code”, IEEE Latin American Transactions, Vol. 19(11), pp. 1903 – 1911, November 2021

[19] L.J. Saiz-Adalid, P. Reviriego, P. Gil, S. Pontarelli, J.A. Maestro, “MCU Tolerance in SRAMs Through Low-Redundancy Triple Adjacent Error Correction”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2015, 23, 2332–2336, doi:10.1109/TVLSI.2014.2357476.

[20] L.J. Saiz-Adalid, J. Gracia-Morán, D. Gil-Tomás, J.C. Baraza-Calvo, P.J. Gil-Vicente, “Reducing the Overhead of BCH Codes: New Double Error Correction Codes”, Electronics 2020, 9, 1897. https://doi.org/10.3390/electronics9111897.

[21] A. Benso and P. Prinetto, “Fault Injection Techniques and Tools for Embedded Systems”, Springer, 2003.

[22] “IEEE Standard for Floating-Point Arithmetic”, in IEEE Std 754-2008, pp. 1-70, 29 Agosto 2008, doi: 10.1109/IEEESTD.2008.4610935.

[23] https://www.elegoo.com/blogs/arduino-projects/elegoo-uno-r3-project-the-most-complete-starter-kit-tutorial

[24] G.S. Rodrigues, F.L. Kastensmidt, A. Bosio; “Approximate Computing for Fault Tolerance Mechanisms for Safety-Critical Applications”. In: A. Bosio, D. Ménard, O. Sentieys, (eds) Approximate Computing Techniques. Springer, 2022.

[25] https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

[26] https://www.sparkfun.com/datasheets/Sensors/SHT1x_datasheet.pdf

[27] https://www.sparkfun.com/datasheets/Sensors/Temperature/MLX90614_rev001.pdf

Vehículos aéreos no tripulados

Protocolo multi-salto para la coordinación eficiente de enjambres de drones con gran dispersión geográfica

David Clérigues[1],Jamie Wubben[1], Carlos T. Calafate[1], Juan Carlos Cano[1] y Pietro Manzoni[1]

Resumen— **Debido a la creciente popularidad de los vehículos aéreos no tripulados (VANTs), podemos observar un interés cada vez mayor en las aplicaciones multi-VANTs. Muchas aplicaciones como: la agricultura de precisión, la vigilancia de fronteras, la infraestructura móvil para el IoT, etc., pueden beneficiarse enormemente si los VANTs trabajan de manera conjunta. Sin embargo, para permitir esta cooperación, los VANTs deben ser capaces de comunicarse de forma inalámbrica entre ellos. Las comunicaciones inalámbricas, pasadas un umbral, no pueden establecerse debido a la pérdida de la señal. Si un VANT se encuentra cerca del umbral sufrirá una gran pérdida de paquetes y no podrá operar sin dificultades. Por ello, en este trabajo proponemos una estrategia que permite a los VANTs comunicarse entre sí de forma eficiente en configuraciones de enjambre basadas en un VANT maestro y varios VANTs esclavos. En concreto, proponemos un protocolo para la coordinación de dichos enjambres que integra el control de misión y la retransmisión de mensajes para situaciones donde algunos VANTs se encuentran muy alejados del maestro. Nuestro enfoque se basa en el descubrimiento de puertas de enlace que permitan a los VANTs más alejados encaminar sus mensajes hacia el maestro de manera eficiente, introduciendo una sobrecarga mínima de mensajes, y teniendo en cuenta las limitaciones del canal inalámbrico. Los resultados de rendimiento obtenidos en un entorno realista de emulación muestran que el enfoque propuesto es capaz de encontrar las puertas de enlace en sólo unos segundos, y que el tiempo total de misión usando retransmisiones multisalto experimenta un aumento muy moderado en comparación con una misión estándar.**

Palabras clave— **UAVs Swarm; ArduSim; Piggybacking; Multi-hop communications; Routing**

I. Introducción

El sector de los VANT (Vehículos Aéreos no Tripulados) ha experimentado un crecimiento exponencial en los últimos años debido a sus diversas aplicaciones en una amplia variedad de industrias, como la agricultura [1], la construcción [2], la minería [3], la vigilancia [4] y la entrega de paquetes [5]. Los VANTs ofrecen ventajas significativas en términos de eficiencia, seguridad y costes en comparación con los sistemas de aeronaves tripuladas. Sin embargo, también presentan desventajas como preocupaciones por la privacidad, la generación de ruido, y la contaminación acústica, limitaciones asociadas con las regulaciones y políticas de seguridad, y la vulnerabilidad ante riesgos de seguridad como ataques cibernéticos [6].

[1]Dpto. de Informática de Sistemas y Computadores (DISCA), Universitat Politècnica de València, Camino de Vera, S/N, 46022 Valencia, España. E-mail: daclefe1@disca.upv.es

Los enjambres de drones son una tecnología emergente que utiliza múltiples VANTs coordinados para realizar tareas específicas. Los enjambres de drones ofrecen una serie de ventajas en comparación con los VANT individuales, como la capacidad de cubrir un área mayor, la redundancia y la resiliencia del sistema, y la capacidad de realizar tareas más complejas. Los enjambres de drones también ofrecen oportunidades para la investigación en áreas como la inteligencia artificial y la robótica, y están siendo explorados para una amplia variedad de aplicaciones, desde la búsqueda y rescate hasta la entrega de paquetes y la vigilancia.

Hemos analizado distintos trabajos centrados en (i) algoritmos de encaminamiento con soporte al multi-salto, y (ii) modelos de coordinación. En primer lugar, Rehman y Wolf [7] sostienen que la retransmisión puede ser una opción técnicamente adecuada para la conectividad multi-salto en redes ad hoc inalámbricas, y comparan el rendimiento de una versión multi-salto del protocolo MAC IEEE 802.11 con el protocolo de encaminamiento ad hoc AODV. Examinando una propuesta de protocolo de enrutamiento multi-salto, Sharma [8] evaluó cómo la modificación del tamaño de los paquetes afectaba al rendimiento. Descubrió que, a medida que aumentaba el tamaño de los paquetes, el rendimiento alcanzaba un máximo en un momento determinado, y luego disminuía gradualmente. Además, Prabha [9] examinó las distintas categorías de redes ad hoc multi-salto inalámbricas, y analizó los principales avances realizados en este campo, entre los que se incluyen las redes oportunistas. Además, Pant [10] propuso un protocolo de enrutamiento multi-salto que utilizaba la agrupación en rejilla, lo que se traducía en una mayor vida útil de la red en comparación con los algoritmos existentes. Por último, Pinto [11] caracterizó las redes aéreas multi-salto de vehículos aéreos no tripulados comerciales, y dedujo el número óptimo de saltos que maximizan el rendimiento de extremo a extremo.

Por otra parte, Richards et al. [12] analizaron soluciones para la asignación autónoma de tareas y la planificación de trayectorias para una flota de VANTs, y compara dos métodos de optimización que integran la asignación de tareas y la planificación de trayectorias para abordar estos retos. Además, Souza et al. [13] propusieron un algoritmo diseñado para coordinar múltiples vehículos aéreos no tripulados en

formación de enjambre, garantizando al mismo tiempo una utilización eficiente del ancho de banda en zonas cubiertas por redes móviles. Kim et al. [14] sugirieron un esquema de control que permite la gestión en tiempo real de drones multi-salto, incluso cuando están fuera del alcance de comunicaciones del sistema de control terrestre. Chen et al. [15] investigaron técnicas de comunicación y redes adecuadas para enjambres de drones, y propusieron una solución inteligente y robusta para enjambres de drones que puede mejorar eficazmente la robustez ante fallos de nodos.

En este artículo presentamos un protocolo para organizar el enjambre con capacidades multi-salto. El enjambre utilizado sigue un patrón de coordinación maestro-esclavo (uno ordena, los otros obedecen). Mediante este protocolo, los nodos que están fuera del rango de comunicaciones del maestro son capaces de operar con normalidad. Dentro del protocolo tenemos dos estados a seguir: (i) *Descubrimiento de vecinos*, y (ii) *Configuración multi-salto*, para poder dar paso a la ejecución de la misión.

Las prestaciones han sido medidas con un simulador realista multi-VANT llamado ArduSim [16]. Las tres variantes del protocolo han sido analizadas. Primero, se ha estimado las cabeceras que estas inducen. Seguidamente, se ha medido la congestión del canal y el tiempo de misión utilizado sin emplear concatenación de mensajes (*piggybacking*) sobre las variantes. Los resultados han sido positivos para la versión basada en puertas de enlace.

La estructura del artículo es la siguiente: la Sección II detalla nuestra propuesta para alcanzar una solución capaz de coordinar enjambres en un entorno multi-salto. Seguidamente, la Sección V concreta el entorno de simulación empleado para validar nuestra solución. A continuación, la Sección IV establece los parámetros de simulación, y detalla los pasos seguidos para obtener los tiempos estimados del protocolo. Además, en la Sección V presentamos las prestaciones obtenidas empleando nuestro protocolo. Por último, la Sección VI resume las metas alcanzadas durante el desarrollo de la propuesta, y discute posibles trabajos futuros.

II. Protocolo

Un enjambre de drones debe estar coordinado en todo momento para evitar problemas durante el vuelo. La coordinación puede emplearse usando consenso, negociación, coordinación maestro-esclavo, u otros mecanismos. Para nuestro trabajo usamos el paradigma maestro-esclavo por su eficacia, sencillez y control preciso. A medida que nos alejamos del maestro es más difícil garantizar la recepción de los mensajes en los esclavos. Añadir multi-salto no solo empeora la recepción de mensajes, sino que también nos encontramos nuevos problemas como la congestión de la red, retardos excesivos, problemas de *routing*, etc. Para solventar estas dificultades hemos diseñado una solución capaz de reducir el impacto de estas penalizaciones, permitiendo al enjambre volar coordinado y cohesionado.

En primer lugar, los esclavos han sido divididos en dos roles: (a) nodos hoja (más alejados), y (b) nodos intermedios. Los nodos intermedios se encargarán de encaminar los mensajes de los nodos más alejados. Además, empezarán a transmitir sus mensajes una vez reciban un mensaje (desde maestro o esclavo más alejado). Por otro lado, las hojas no retransmiten los mensajes recibidos, y son los encargados de empezar a transmitir los mensajes de los esclavos (se explica en mayor detalle más abajo). Seguidamente, dentro del protocolo hemos definido dos estados: (a) descubrimiento de vecinos, y (b) *Configuración multi-salto* basada en puertas de enlace. Tras esto, se hace uso de una misión preestablecida donde hemos definido una serie de pasos a seguir. En las siguientes secciones detallamos el uso de cada estado y de la misión.

A. Descubrimiento de vecinos

Este es el primer estado del protocolo. Cada nodo del enjambre envía y recibe mensajes en un bucle durante un tiempo predefinido (véanse los detalles en la Sección IV). De este modo, cada nodo descubre qué nodos están dentro de su alcance de radio.

El Algoritmo 1 muestra el comportamiento del estado. Mientras el tiempo de ejecución sea inferior al tiempo establecido, se envía un mensaje que contiene el identificador de estado y el identificador de nodo; tras el envío, se lee la memoria intermedia del canal. Si hay un mensaje de estado *Descubrimiento de vecinos* en la memoria intermedia, se procesa. El procesamiento es sencillo: primero se extrae el identificador del emisor y, a continuación, este identificador se almacena en la lista de vecinos.

B. Configuración multisalto

Al final del estado de descubrimiento de vecinos, cada VANT tiene sus vecinos almacenados en una lista. Cada VANT conoce sus conexiones directas, y en base a dicha información, cada VANT determina cuántos saltos tiene hasta el maestro, y configura su puerta de enlace para llegar a él. El planteamiento es el siguiente:

- El maestro envía mensajes que incluyen la lista de todos sus vecinos.
- Cada esclavo, al recibir un mensaje, lo propaga sólo si se cumplen simultáneamente 3 condiciones:
  1. Su número de saltos es superior al del mensaje recibido.
  2. Su identificador está en la lista recibida
  3. Es capaz de alcanzar nodos aún no incluidos en la lista (añade sus vecinos y aumenta el número de saltos antes de retransmitir)

Por un lado, el maestro enviará un mensaje compuesto por: (i) su identificador, (ii) su número de saltos (cero), (iii) el identificador de estado, y (iv) su lista de vecinos. Este mensaje se enviará periódicamente hasta que se agote el tiempo. Debido a su simplicidad, no se detalla este algoritmo.

Por otro lado, los esclavos ejecutan el Algoritmo 2

Algoritmo 1: Estado Descubrimiento de vecinos

```
tiempoInicial ← tiempoActual
mientras tiempoActual − tiempoInicial < tiempoEstablecido hacer
    Enviar Mensaje Descubrimiento de vecinos (Nodo ID + Estado ID)
    msg ← Recibir Mensaje Descubrimiento de vecinos (Estado ID)
    si msg ≠ null entonces
        Vecinos ← msg[transmisorID]
    fin
fin
```

durante un tiempo limitado (más detalle en la Sección IV). En primer lugar, se lee el búfer de recepción. Si se almacena un mensaje de esta fase, se procesa. Al procesar el mensaje, se extrae el número de saltos recibidos. A continuación, se actualiza el número de saltos del nodo si (a) aún no se ha inicializado, y (b) el número de saltos recibidos es inferior en dos o más unidades. Además, el nodo que envió el mensaje se anota a sí mismo como pasarela. Con el número de saltos actualizado, se comprueba que (a) el número de saltos del nodo es mayor, (ii) su identificador está en la lista, y (iii) uno de sus vecinos aún no está en la lista. Si se cumplen las condiciones, se envía un mensaje con los valores actualizados (identificador del nodo y lista). En caso contrario, el nodo es uno de los más alejados y no retransmite (el mensaje se descarta).

El campo de número de saltos se utiliza en los mensajes de difusión. Al enviar un mensaje, el nodo adjunta su recuento de saltos. Cuando otro nodo recibe el mensaje, comprueba si su recuento de saltos local es mayor. En caso afirmativo, el mensaje se almacena en la memoria intermedia. Por el contrario, cuando se trabaja con mensajes *unicast*, hay que definir pasarelas. Por lo tanto, cuando se envía un mensaje, se entrega a un nodo pasarela específico. Al recibir el mensaje, un nodo comprueba que (i) es el receptor, y (ii) es la pasarela de su vecino. En ese caso, se almacena en el búfer para su posterior tratamiento.

C. Misión

Por último, una vez que cada nodo del enjambre ha configurado el modo multisalto, se puede realizar una misión. Esta consiste en alcanzar una serie de *waypoints*. Para alcanzar estos *waypoints*, los nodos alternan entre dos subestados: (i) *Waypoint Alcanzado*, y (ii) *Moverse al siguiente Waypoint*; dichos estados se repiten en bucle hasta alcanzar el último *waypoint*.

En el subestado *Waypoint Alcanzado*, el maestro espera a recibir al menos un mensaje de cada esclavo del enjambre. Este mensaje contiene: (i) el identificador del mensaje, (ii) un identificador de fase, y (iii) un acuse de recibo de llegada. Cuando se cumple esta condición, pasa al estado *Moverse al siguiente Waypoint*. Por otro lado, los esclavos difieren en su comportamiento. Por ejemplo, un nodo hoja comienza transmitiendo mensajes de *Waypoint Alcanzado*. Además, sólo recibe mensajes de Mover a waypoint. Al recibir un mensaje de este tipo, cambia su estado a Mover a waypoint. Los nodos intermedios no transmiten mensajes hasta que reciben un mensaje de la hoja o de su vecino. En ese momento, pueden enviar su propio mensaje *Waypoint Alcanzado*. Cuando se recibe un mensaje, se puede asociar al subestado *Waypoint Alcanzado*, o al subestado *Moverse al siguiente Waypoint*. Para los casos de *Waypoint Alcanzado*, el nodo retransmite este mensaje. Por otro lado, si se trata de un mensaje *Moverse al siguiente Waypoint*, también retransmite dicho mensaje antes de pasar a dicho estado.

El estado*Moverse al siguiente Waypoint* es más simple. El maestro envía mensajes a los esclavos para indicarles que avancen. La estructura del mensaje consta de: (i) el identificador del mensaje, (ii) un identificador de fase, y (iii) la orden de moverse. Los nodos esclavos leen el búfer del canal. Si reciben un mensaje*Moverse al siguiente Waypoint*, lo retransmiten. Este último paso no se aplica a las hojas.

El objetivo principal de esta pequeña misión planificada es comprobar la correcta coordinación del enjambre, y medir: (i) la congestión del canal, y (ii) la duración de la misión.

D. Piggybacking

El Piggybacking es una técnica de transmisión de datos utilizada en la capa de enlace. Su objetivo es agrupar un acuse de recibo junto con el siguiente paquete enviado. Este mecanismo ofrece una disminución de las transmisiones de paquetes, mejorando así la eficiencia, ya que, al utilizar el reenvío multisalto, el número de mensajes enviados y la congestión del canal pueden tender a aumentar. Además, no garantizamos que los mensajes sean recibidos por el destino. En nuestra solución adoptamos la técnica de Piggybacking para reducir el tráfico en la red, mejorando así las prestaciones. Destacar que dicha opción es viable en nuestro caso, ya que todos los mensajes de los esclavos tienen un destino único, que es el nodo maestro. De esta manera, un nodo intermedio reenviará el mensaje si, y sólo si, (i) el mensaje que tiene que enviar tiene el mismo destino que el mensaje recibido, y (ii) ambos mensajes pertenecen al mismo estado. Si no se cumplen estas condiciones, enviará el mensaje recibido sin hacer piggybacking de su propio mensaje. Otro tipo de mensaje es el que no tiene destino (*broadcast*), que suele enviar el maestro, dirigiéndose a todos los esclavos. En este caso, el piggybacking no es aplicable. Al implementar

Algoritmo 2: Estado Descubrimiento de puertas de enlaces y saltos

```
tiempoInicial ← tiempoActual
mientras tiempoActual − tiempoInicial < tiempoEstablecido hacer
    msg ← Recibir Mensaje Salto (Estado ID)
    si msg ≠ null entonces
        recbSalto ← msg[saltos]
        recbTransmisorID ← msg[senderID]
        si (saltosAlMaestro = null) or (recbSalto ≤ saltosAlMaestro − 2) entonces
            saltosAlMaestro = recbSalto + 1
            puertaAlMaestro = recbTransmisorID
        fin
        recbVANTList ← msg[Lista_VANTs]
        si VANT cumple las tres reglas entonces
            Enviar Mensaje de Saltos actualizado (Nodo ID + Estado ID + Nueva Lista + Nuevos
             Saltos)
            soyHoja = falso ;                                   // No soy el más lejano
        fin
    fin
fin
```

el comportamiento piggybacking en nuestro protocolo, hemos conseguido reducir el número de mensajes enviados y la congestión del canal, aumentando al mismo tiempo la eficiencia durante el transcurso de la misión.

III. Framework Experimental

En la actualidad el desarrollo de soluciones para VANTs tiende a basarse en dos estrategias alternativas: despliegues reales o experimentos de simulación. Los investigadores tienden a la simulación, ya que el uso de dispositivos físicos puede implicar importantes recursos económicos y materiales. Por otro lado, las simulaciones permiten conseguir un producto más sofisticado y óptimo, y posiblemente llevarlo a un entorno real de forma más segura.

Hoy en día existe una amplia gama de simuladores en el mercado. En este trabajo, nos hemos basado en el simulador ArduSim [16]. Esta herramienta de simulación soporta simulaciones en tiempo real, siendo una característica innovadora en el mercado. Las comunicaciones entre dispositivos se establecen mediante el uso de un canal inalámbrico virtual que emula las comunicaciones en entornos compartidos en tiempo real. El simulador, al ser de código abierto, permite a diferentes usuarios crear sus propias soluciones, desde protocolos hasta canales de comunicación específicos. Para más detalles, recomendamos la lectura de [16].

Finalmente, dentro del simulador, el usuario dispone de un conjunto definido de formaciones de vuelo para estructurar el enjambre. Se puede usar la formación matriz, aleatoria, circular y la linear. Esta última es la que vamos a usar para llevar a cabo nuestros experimentos.

IV. Parámetros

En esta sección definimos los parámetros de la simulación de la Sección V. Además, analizamos los estados mostrados en la Sección II para determinar el tiempo estimado de cada uno. Cada algoritmo ha sido ejecutado durante un minuto. Durante este tiempo se ha capturado los tiempos donde los nodos hojas han conseguido descubrir su puerta de enlace y número de saltos. Se ha medido el rendimiento sobre estos nodos, ya que son los más perjudicados dentro del enjambre. Los tiempos se han medido cuando las hojas están a un salto (3 nodos), a dos saltos (5 nodos), a tres saltos (7 nodos), y a cuatro saltos (9 nodos).

Los parámetros que hemos establecido son los siguientes: 300 metros con un rango fijo de comunicaciones, una distancia de 240 metros entre los drones, cuatro *waypoints* en la misión, y un número de saltos máximo de cuatro. La separación empleada ha sido la máxima posible para poder estresar la red al máximo, y medir los datos en las peores condiciones posibles.

El procedimiento de *Descubrimiento de vecinos* ha demostrado poder ejecutarse en menos de 800 milisegundos, tal y como ilustra la Figura 1. En la formación linear, tanto los nodos intermedios como el maestro siempre tienen dos vecinos, mientras que las hojas solo uno. La *Configuración multi-salto* muestra unos resultados más positivos donde el tiempo estimado está comprendido entre 0.1 segundos y 1.5 segundos. En total, al combinar ambos estados se añade una sobrecarga temporal entre 750 milisegundos y 2 segundos dependiendo la cantidad de nodos.

V. Experimentación

La misión ha sido ejecutada varias veces para recoger la suficiente información para que sea fiable. El análisis de los datos ha sido realizado usando el máximo número de saltos. Además, se han clasificado los datos para el nodo maestro, los intermedios, y las hojas, ya que cada uno tiene una función específica, y los más afectados son los intermedios.

Las Tablas I y II muestran los resultados obtenidos en envío y recepción. Las hojas envían mensa-

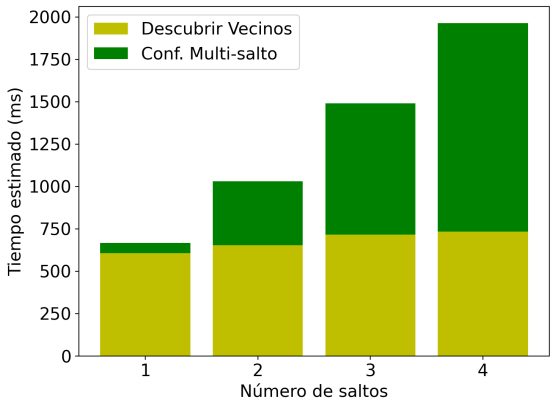


Fig. 1: Tiempo estimado para cada estado.

jes periódicamente hasta que reciben un mensaje del maestro para cambiar de fase. La recepción siempre será igual en todas las variantes. Sin embargo, durante el envío se observa como utilizar puertas de enlace con piggybacking reduce drásticamente el envío de mensajes. Seguidamente, los nodos intermedios son los más estresados, ya que estos se encargan de enviar sus mensajes y los que reciben por parte de sus vecinos. En este escenario, tanto el envío como la recepción han sido agilizadas al emplear las puertas de enlace con el piggybacking. Por último, el maestro ha mostrado un comportamiento idéntico en todas las variantes durante el envío (un resultado esperado, ya que siempre envía la misma cantidad de mensajes). Sin embargo, la recepción denota una mejoría empleando piggybacking independientemente de la variante empleada. Si no consideramos el piggybacking, las puertas de enlace siguen siendo una opción a considerar.

Tabla I: Congestión del canal durante la misión (Sin Piggybacking).

Congestión del canal		
	Sin Piggybacking	
Saltos	**Enviados**	**Recibidos**
Maestro	473	41
Intermedio	362	302
Hoja	154	4

Tabla II: Congestión del canal durante la misión (Con Piggybacking)

Congestión del canal		
	Con Piggybacking	
Saltos	**Enviados**	**Recibidos**
Maestro	473	8
Intermedio	291	282
Hoja	49	4

Cuanto más nos alejamos del maestro, más tiempo se tarda en completar la misión. Observando la Tabla III podemos corroborar como el uso de piggybacking mejora sustancialmente los tiempos de vuelo. Al no usar piggybacking notamos como los nodos hojas tardan casi 25 segundos más que el maestro en finalizar la misión, mientras que su contraparte tan solo tiene una variación de 12 segundos.

Tabla III: Tiempo total de la misión

Tiempo de vuelo total (s)		
Saltos	**Sin Piggy.**	**Con Piggy.**
0	106,81	95,91
1	109,46	99,62
2	118,46	102,16
3	128,55	105,50
4	130,14	108,56

VI. Conclusiones y trabajo futuro

Los enjambres de drones tienen un gran potencial, pero su problema principal es que estos deben estar coordinados en todo momento. Al emplear un escenario multi-salto, este problema se vuelve más tedioso de lidiar debido al aumento de la pérdida de paquetes y los retardos. Para ello, en este artículo hemos planteado una solución capaz de superar estas adversidades de forma eficiente. El protocolo para la gestión de enjambres en un entorno multi-salto consta de dos partes esenciales. En primer lugar, *Descubrimiento de vecinos* permite a cada nodo del enjambre descubrir a sus vecinos en cuestión de 750 ms aproximadamente. Seguidamente, las puertas de enlace son configuradas en cada nodo entre 50 ms y un segundo y medio (dependiendo el número de saltos). Con ambas fases introducimos una sobrecarga total de aproximadamente dos segundos. Finalmente, la experimentación ha demostrado que el protocolo es capaz de cumplir una misión guiada en un entorno multi-salto. Los resultados obtenidos han sido positivos, ya que la congestión del canal es aceptable en cada uno de los nodos y, además, el tiempo total de la misión no es elevado. Cabe destacar que el piggybacking es un mecanismo muy eficiente que ha conseguido acelerar la ejecución de la misión y reducir el envío/recepción de mensajes drásticamente. Como trabajo futuro buscamos adaptar este enfoque a un entorno real para medir unas prestaciones más especificas. Además, este entorno real irá variando, introduciendo distintos obstáculos como ruidos, obstrucciones y otros elementos que perjudiquen las comunicaciones.

Agradecimientos

Este trabajo deriva de los proyectos de I+D PID2021-122580NB-I00 y SRTC1900C007159XV0, financiados por MCIN/AEI/10.13039/501100011033 y "FEDER, Una manera de hacer Europa".

Referencias

[1] E. Raymond Hunt and Craig S. T. Daughtry, "What good are unmanned aircraft systems for agricultural remote sensing and precision agriculture?," *International Journal of Remote Sensing*, vol. 39, pp. 5345 – 5376, 2018.

[2] Javier Irizarry, Masoud Gheisari, and Bruce N. Walker, "Usability assessment of drone technology as safety inspection tools," *J. Inf. Technol. Constr.*, vol. 17, pp. 194–212, 2012.

[3] Javad Shahmoradi, Elaheh Talebi, Pedram Roghanchi, and Mostafa Hassanalian, "A comprehensive review of applications of drone technology in the mining industry," 2020.
[4] Xiufang Shi, Chaoqun Yang, Weige Xie, Chao Liang, Zhiguo Shi, and Jiming Chen, "Anti-drone system with multiple surveillance technologies: Architecture, implementation, and challenges," *IEEE Communications Magazine*, vol. 56, pp. 68–74, 2018.
[5] Dane Bamburry, "Drones: Designed for product delivery," *Design Management Review*, vol. 26, pp. 40–48, 2015.
[6] Daniela Rojas Viloria, Elyn L. Solano-Charris, Andrés Muñoz-Villamizar, and Jairo R. Montoya-Torres, "Unmanned aerial vehicles/drones in vehicle routing problems: a literature review," *International Transactions in Operational Research*, vol. 28, no. 4, pp. 1626–1657, 2021.
[7] Habib ur Rehman and Lars C. Wolf, "A multihop ieee 802.11 mac protocol for wireless ad hoc networks," *2009 29th IEEE International Conference on Distributed Computing Systems Workshops*, pp. 432–439, 2009.
[8] Namita Sharma, "Impact of varying packet size on multihop routing protocol in wireless sensor network," 2014.
[9] Chander Prabha, Surender Kumar, and Ravinder Kumar Khanna, "Wireless multi-hop ad-hoc networks: A review," *IOSR Journal of Computer Engineering*, vol. 16, pp. 54–62, 2014.
[10] Manoj Kumar Pant, Biswanath Dey, and Sukumar Nandi, "A multihop routing protocol for wireless sensor network based on grid clustering," *2015 Applications and Innovations in Mobile Computing (AIMoC)*, pp. 137–140, 2015.
[11] Geon-Hwan Kim, Jae-Choong Nam, Imtiaz Mahmud, and You-Ze Cho, "Multi-drone control and network self-recovery for flying ad hoc networks," in *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2016, pp. 148–150.
[12] Arthur G. Richards, John S. Bellingham, Michael Tillerson, and Jonathan P. How, "Coordination and control of multiple uavs," 2002.
[13] Bruno José Olivieri de Souza and Markus Endler, "Coordinating movement within swarms of uavs through mobile networks," *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pp. 154–159, 2015.
[14] Geon-Hwan Kim, Jae-Choong Nam, Imtiaz Mahmud, and You-Ze Cho, "Multi-drone control and network self-recovery for flying ad hoc networks," in *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2016, pp. 148–150.
[15] Wu Chen, Jiajia Liu, Hongzhi Guo, and Nei Kato, "Toward robust and intelligent drone swarm: Challenges and future directions," *IEEE Network*, vol. 34, pp. 278–283, 2020.
[16] Francisco Fabra, Carlos T. Calafate, Juan Carlos Cano, and Pietro Manzoni, "Ardusim: Accurate and real-time multicopter simulation," *Simulation Modelling Practice and Theory*, vol. 87, pp. 170–190, 2018.

Uso de campos de fuerza direccionales para la gestión táctica de conflictos entre UAVs

Julián Arenillas[1], Jamie Wubben[1], Enrique Hernández-Orallo[1], Carlos T. Calafate[1]

Resumen— **A medida que avanzamos hacia escenarios donde la adopción de vehículos aéreos no tripulados (VANTs o UAVs) se vuelve más masiva, se requieren soluciones inteligentes para resolver de manera eficiente los conflictos en las trayectorias de vuelo de las aeronaves para evitar posibles colisiones. Entre los diferentes enfoques posibles, la adopción de campos de fuerza virtuales es una posible solución reconocida por ser simple, distribuida y efectiva. En este trabajo estudiamos el rendimiento de un enfoque de campo de fuerza direccional (D-FFP), evaluando sus beneficios en comparación con un protocolo de campo de fuerza estándar (FFP) utilizando un simulador realista. Los resultados muestran que, en escenarios típicos asociados con corredores de tráfico aéreo, el enfoque propuesto puede reducir la sobrecarga de tiempo de vuelo en un 69,1 % (en promedio), respetando las distancias de seguridad de vuelo requeridas entre aeronaves.**

Palabras clave— **UAVs, gestión de táctica de conflictos, U-Space, campos de fuerza direccionales.**

I. Introducción

LOS vehículos aéreos no tripulados (VANTs o UAVs), coloquialmente conocidos como drones, toman un papel importante en el ámbito de las ciudades inteligentes, ofreciendo nuevas posibilidades que conllevan a crear nuevas restricciones [1]. A medida que pasa el tiempo, aumentan las tecnologías y aplicaciones que buscan hacer uso del espacio aéreo, usando drones para realizar varios servicios [2].

Este aumento en el uso de los drones, dentro de las ciudades, plantea cuestiones nuevas para permitir la integración segura y eficiente de aeronaves no tripuladas en entornos urbanos [3]. Es aquí donde entra el término U-Space, concepto emergente en la industria de aviación, que hace referencia a infraestructuras y regulaciones desarrolladas con el propósito de solventar problemas de gestión, seguridad y privacidad que puedan surgir de compartir este espacio aéreo dentro de una población. Adicionalmente, la integración mutua de la red móvil 5G y del U-Space también es un punto extra a analizar para aumentar la cobertura de comunicación entre UAVs [4].

Es por eso mismo que se plantean nuevos problemas, urgiendo sistemas para gestión de colisiones que permitan evitar choques entre drones que compartan un mismo espacio aéreo [5], y impidiendo que ocurran accidentes graves dentro de un entorno urbano.

A la hora de abordar este tipo de conflictos, suelen considerarse dos enfoques complementarios entre sí [6]: la gestión estratégica y la gestión táctica de conflictos. La gestión estratégica de conflictos se basa en analizar y estudiar los distintos obstáculos antes de que despeguen los drones, modificando las rutas de vuelo necesarias cuando se detecta un posible conflicto futuro. Por otro lado, la gestión táctica de conflictos trata de analizar los conflictos posibles una vez los drones están en pleno vuelo, modificando la trayectoria del dron en el momento que detecta una posible colisión. En este trabajo, nos centraremos en este segundo enfoque, donde, concretamente, se propondrá un método basado en campos de fuerza que permita lograr una resolución de conflictos simple y escalable a varios drones. En un trabajo anterior [7], se introdujo el protocolo FFP, o "Force Field Protocol", que establece un campo de fuerza omnidireccional alrededor de un dron de tipo multirrotor. Este protocolo ofrece buenos resultados con diferentes patrones de vuelo. No obstante, presenta un margen de mejora, que intentaremos conseguir en este artículo, adaptándolo para que ofrezca un campo de repulsión direccional. No obstante, debido al gran número de parámetros involucrados, hace falta lograr un ajuste preciso del método, lo cual conlleva realizar una gran cantidad de pruebas para validar la estabilidad de los drones al realizar una maniobra de evasión, así como para optimizar su rendimiento.

El resto de este artículo se organiza de la siguiente manera: en la siguiente Sección II mostraremos distintos trabajos relacionados en este ámbito. A continuación, en la Sección III, propondremos nuestra solución, y se detallará el entorno donde se desarrolla la solución en la Sección IV. Los resultados que se han obtenido para medir el rendimiento del protocolo desarrollado se presentan en la Sección V. Por último, en la Sección VI, se presentan las conclusiones de este artículo, y los distintos trabajos que se pueden realizar en un futuro.

II. Trabajos relacionados

El ámbito de los sistemas anticolisión ha sido estudiado para todo tipo de vehículos modernos. Existen varias soluciones y distintos enfoques, donde el artículo [8] ofrece una clasificación de la mayoría de categorías para las estrategias de evitación de colisiones, entre las que se incluyen: estrategias geométricas, uso de campos de fuerza, trayectorias de escape optimizadas y estrategias de detección y evitación.

Las estrategias geométricas se basan en la recolección de información dada por la geometría (ej. trayectoria, velocidad, posición) del dron y del obstáculo a colisionar para recalcular una nueva trayectoria. Cuando se detecta una colisión, se reajusta la trayectoria con la menor desviación posible, evadiendo dicho obstáculo al consultar la información geométri-

[1]Dpto. de Informática de Sistemas y Computadores (DISCA), Universitat Politècnica de València (UPV), Camino de Vera, S/N, 46022 Valencia. E-mail: juarpo@inf.upv.es, {jwubben,ehernandez,calafate}@disca.upv.es.

ca que ofrecen ambas partes. Un ejemplo es [9], donde se propone un sistema de Vigilancia Dependiente Automática, o ADS-B, que se utiliza para detectar otros UAVs en el aire y gestionar conflictos con un enfoque geométrico.

Con la segunda metodología, el uso de campos de fuerza, o campos potenciales artificiales, utiliza un enfoque de partículas cargadas para gestionar las colisiones. Mediante fuerzas de atracción y repulsión, se permite a un dron ser atraído o repelido por los diferentes objetivos u obstáculos que pueda encontrar en su trayectoria. En entornos estáticos, este enfoque se ejecuta correctamente; sin embargo, no es el mismo caso en obstáculos dinámicos, siendo necesaria tanto la información geométrica como la dirección de los obstáculos y drones, por lo que se aumenta la complejidad de las soluciones [8]. Un análisis de la literatura muestra que existen pocos trabajos que utilizan campos de fuerza para evitar colisiones usando drones. La mayoría solo consideran los obstáculos como objetos estáticos [10–13]. Choi et al. [14] sí que consideran tanto objetos estáticos como móviles en el aire. Utilizando un campo vectorial independiente del operador rotacional, logran evitar obstáculos satisfactoriamente. Su principal enfoque en este trabajo es solucionar el problema del mínimo local existente en los obstáculos estáticos. Kownacki et al. [15], también han utilizado un campo de fuerza para evitar colisiones. En su desarrollo, consideraron VANTs no holonómicos (p.ej. drones de ala fija) con varias tandas de simulaciones numéricas que demuestran el correcto funcionamiento de sus algoritmos. Utilizando también un VANT no holonómico, Burgos et al. [16] implementan un campo de flujo potencial para evadir colisiones junto a un campo de fuerza, permitiendo combinaciones de diferentes enfoques para lograr una resolución válida de conflictos.

Respecto a estrategias basándose en la optimización, se consideran los conflictos y sus evasiones como problemas que requieren del uso de la optimización para solucionarlos. Para ello se toma información geográfica, posiciones, tamaño de los obstáculos y más puntos clave para el cálculo de la optimización [17]. Si tenemos en cuenta la computación limitada de los UAVs, y las consecuencias de equiparlos con herramientas más potentes (aumento de peso, reducción de la batería, etc.), la optimización gana un importante papel en los cálculos de evitación de colisiones. Diferentes algoritmos han sido desarrollados para esto, por ejemplo, algoritmos inspirados en hormigas, algoritmos genéticos, optimización bayesiana, métodos basados en el descenso del gradiente, optimización por enjambre de partículas, algoritmos voraces, y las aproximaciones locales [8]. En [18], se propone un algoritmo de búsqueda de tiempo mínimo (MTS) que utiliza una optimización inspirada en el comportamiento de búsqueda de comida de las hormigas, utilizada en drones para encontrar una ruta libre de colisiones, asegurando al mismo tiempo una comunicación estable con la estación de control en tierra.

Por último, las estrategias de detección y evitación utilizan distintos sensores ubicados en los UAVs que permiten la detección de los obstáculos, y su posterior evasión. El uso de sensores reduce considerablemente la computación necesaria para detectar estos obstáculos, y así mejorar los tiempos de respuesta. Este enfoque combina muy bien con un entorno dinámico, detectando posibles obstáculos en movimiento y/o UAVs cercanos o dentro de la misma formación, sin conocer sus rutas de vuelo [8]. Wang et al. [19], proponen la utilización de un sensor LiDAR 2D en un algoritmo de detección de obstáculos estáticos y dinámicos, además del posible seguimiento de estos últimos. Este algoritmo ofrece mejoras de eficiencia en la computación y memoria necesaria en un UAV. El uso de los sensores LiDAR es común para la detección de obstáculos, gracias a su alta precisión, donde muchas soluciones y desarrollos incluyen esta tecnología. No obstante, estos sensores pueden presentar un precio elevado, permitiendo la entrada a sensores más accesibles como los detectores de proximidad vía ultrasonidos, pero requieren un manejo de datos adecuado. Balemans et al. [20], ofrecen un autocodificador que utiliza convoluciones para predecir datos LiDAR, basándose en las mediciones obtenidas por un sensor de ultrasonidos.

En la mayoría de estos trabajos mencionados, se han utilizado simuladores para realizar pruebas, donde el uso de MATLAB es mencionado con frecuencia. Con este tipo de simulaciones, se pueden realizar una gran cantidad de experimentos de forma rápida y eficiente para obtener datos preliminares. Sin embargo, a menudo se omite la complejidad real de un sistema físico, como la inercia. Así pues, en trabajos anteriores [7], hemos implementado un enfoque basado en campo de fuerza (FFP) que muestra como las colisiones pueden ser evitadas con una pequeña sobrecarga temporal. No obstante, no se tuvo en cuenta factores como la dirección del obstáculo con respecto a la dirección del vuelo. Es decir, la magnitud del vector de repulsión era la misma independientemente de si el obstáculo estaba enfrente del dron o a su lado. Esto provocaba que el dron percibiera repulsiones más exageradas frente a obstáculos que no necesitaban una evasión tan drástica. Por esa razón, en este artículo se realiza una evaluación basada en simulación realista de un protocolo de campo de fuerza direccional (D-FFP), el cual se detalla en el siguiente apartado.

III. El protocolo D-FFP

En la versión anterior de nuestro protocolo FFP o "Force Field Protocol" [7], se utiliza un planteamiento convencional donde el punto destino de vuelo actúa como una fuerza constante de atracción, imitando una fuerza gravitacional. Por otra parte, cada pareja de drones en vuelo se comporta como dos cargas eléctricas similares, desviándose entre sí de forma omnidireccional, generando más fuerza cuanto más se acercan. En este artículo ofrecemos una variación del comportamiento de los drones, aplican-

do los principios de repulsión entre dos imanes con la misma polaridad y, por ende, la fuerza de repulsión dependerá de la dirección (θ). Este comportamiento se ha modelado según la ecuación:

$$R(\theta,\mu) = \begin{cases} cos(\mu\cdot\theta) : \theta \in [-\frac{\pi}{2\cdot\mu}, \frac{\pi}{2\cdot\mu}] \\ C : \theta \notin [-\frac{\pi}{2\cdot\mu}, \frac{\pi}{2\cdot\mu}] \end{cases}$$

Esta ecuación nos permite modificar el comportamiento de la fuerza de repulsión, ajustando los parámetros C y μ que permiten determinar el tamaño de la componente omnidireccional y la anchura del lóbulo principal, respectivamente.

Esto lo representamos en la Figura 1 fijando los valores de C a 0.25, y de μ a 2. Podemos observar como los ángulos próximos a cero (donde el objetivo se encuentra a lo largo de la línea definida por el vector velocidad), el vector de repulsión alcanza valores altos, perdiendo fuerza al alejarse de esta dirección, alcanzando un valor de repulsión mínimo y constante para el resto de direcciones.

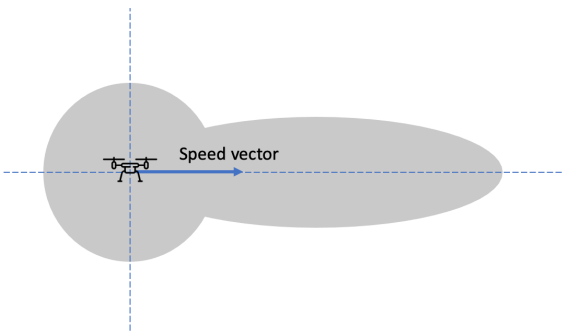


Fig. 1: Patrón de repulsión generado. ($C = 0{,}25; \mu = 2$).

IV. Entorno experimental

A la hora de ejecutar nuestros experimentos, hemos implementado ambos protocolos, tanto la versión anterior FFP como nuestra versión D-FFP, en el simulador ArduSim [21]. Este simulador nos permite resolver el anterior problema de omisión de parámetros físicos, donde entran en juego varios aspectos importantes como la inercia, factor a tener en cuenta en los cambios de dirección en tiempo real.

Para presentar y comparar las diferencias en el rendimiento de cada protocolo, hemos ideado cinco escenarios distintos que representan intersecciones entre dos drones. Estos escenarios logran crear conflictos a resolver por las técnicas de evasión de colisiones desarrolladas en los protocolos, donde se varían la situación y ángulo del conflicto entre ambos drones. Estos escenarios están representados en la Figura 2.

En términos de análisis de rendimiento, lo que se busca obtener de estos cinco escenarios es un equilibrio óptimo entre la sobrecarga del tiempo de vuelo y la distancia de seguridad entre ambos drones. En la siguiente sección, mediremos el tiempo total de vuelo de los drones, y compararemos las diferencias de tiempo entre ambos protocolos. Además, se medirá la distancia mínima entre drones, que deberá ser superior a 10 metros, distancia de seguridad establecida para compensar un error posible en el GPS (± 5 para cada uno) que pueda provocar una colisión.

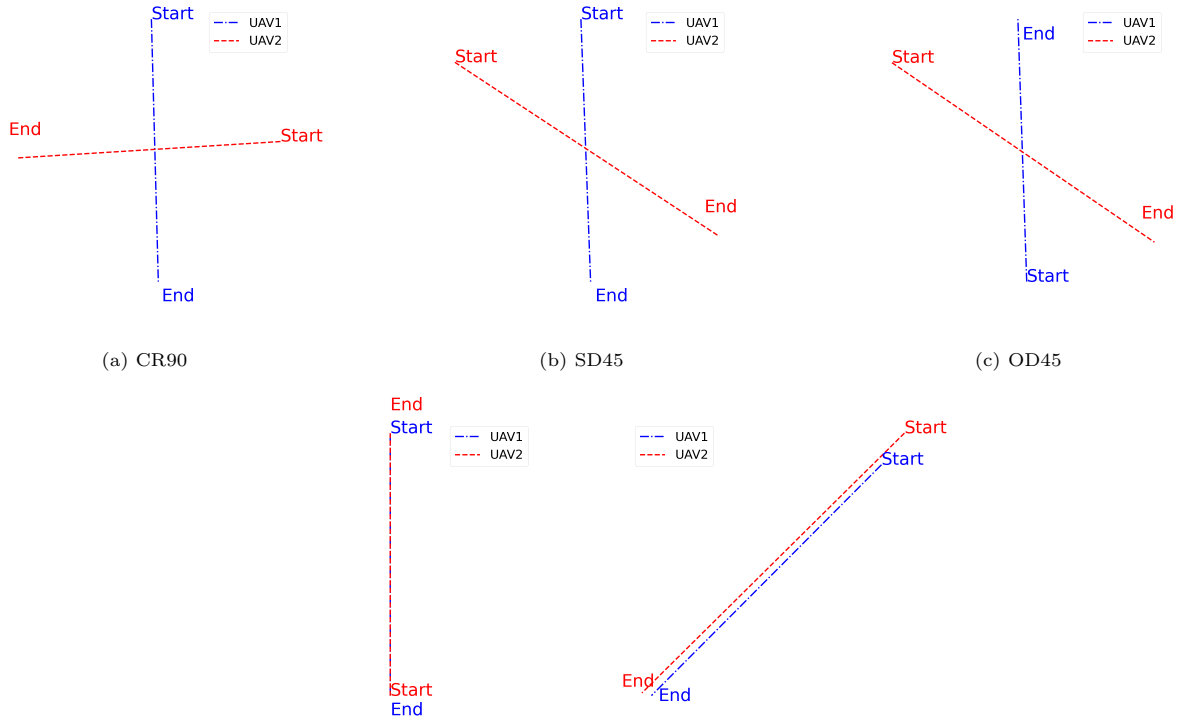
Dicho vector de repulsión se añadirá a un vector de atracción, calculado respecto a los distintos objetivos propuestos en la misión. El dron será comandando para que realice un movimiento hacia el vector resultante de sumar ambos vectores. Como el vector de repulsión es reducido a un valor constante pequeño, si no se detecta ningún obstáculo en el campo de fuerza, el vector de atracción siempre dominará al dron, moviendo al dron por los distintos puntos de la misión.

Si se detecta un obstáculo en el campo de fuerza, el vector resultante se verá modificado por el vector de repulsión, alterando la trayectoria principal del dron y realizando evasiones según la posición del obstáculo. Si el obstáculo está presente en la parte direccional del campo de fuerza, el vector de repulsión tendrá una dirección opuesta al vector de atracción, generando un contrapeso que reducirá su velocidad y lo desplazará ligeramente hacia uno de los lados. Por otra parte, si el obstáculo está presente en la parte omnidireccional, el vector de repulsión presentará una dirección opuesta al obstáculo, alejándose de forma similar al comportamiento del protocolo FFP, pero con menor fuerza.

Algoritmo 1 D-FFP

```
1:  despegue()
2:  iniciarComunicación()
3:  mientras objetivos > 0 hacer
4:      mientras !objetivoAlcanzado hacer
5:          atracción = obtenerVectorAtracción()
6:          repulsión = obtenerVectorRepulsión()
7:          resultado = sumaVectores(atracción, repulsión)
8:          moverUAV(resultado)
9:          esperar(200 milisegundos)
10:     fin mientras
11:     quitarObjetivo()
12: fin mientras
13: detenerComunicación()
14: aterrizar()
```

En el Algoritmo 1, mostramos el continuo código ejecutado del protocolo D-FFP. Como primer paso en la misión, los drones despegarán de sus posiciones iniciales, entablando comunicaciones una vez están situados en el aire. El algoritmo de evasión funcionará hasta que se cumplan todos los objetivos a sobrevolar en el aire, ejecutándose periódicamente cada 200 milisegundos. Para cada objetivo, se calculará su respectivo vector de atracción, junto al vector repulsión de todos los obstáculos presentes en el aire. Ambos vectores se sumarán para dar el vector resultado que definirá la trayectoria a desplazarse por el dron. Una vez se hayan cumplido todos los objetivos, se dará por finalizada la misión, parando todas las comunicaciones y aterrizando a los respectivos drones que ejecuten este protocolo.



(a) CR90 (b) SD45 (c) OD45

(d) HO (e) TO

Fig. 2: Representación de los cinco escenarios usados en las pruebas.

V. Resultados

En esta sección se presentan los resultados de las simulaciones, obtenidos como resultado de ejecutar ambos protocolos a evaluar en los cinco escenarios descritos.

En la Tabla I, medimos el rendimiento de cada protocolo utilizando dos métricas: la distancia mínima entre drones y la sobrecarga de tiempo, representada como ST. Para esta sobrecarga, comparamos el tiempo mínimo para completar la misión con el tiempo de ejecución resultante, donde este tiempo mínimo viene establecido por la finalización del escenario sin aplicar ningún algoritmo de gestión de conflictos.

Como podemos observar en la Tabla I, D-FFP reduce considerablemente la sobrecarga en los cinco escenarios propuestos. Por otra parte, las distancias mínimas también se ven reducidas en todos los casos, salvo el último. El último caso representa un adelantamiento entre dos drones, donde el que realiza una trayectoria más larga alcanza con mayor velocidad al otro. Al tener un campo de fuerza direccional, el dron que va a realizar la maniobra de adelantamiento detecta antes al obstáculo a evitar, a diferencia del campo omnidireccional. Esto resulta en una evasión más temprana y, por lo tanto, un aumento en las distancias mínimas.

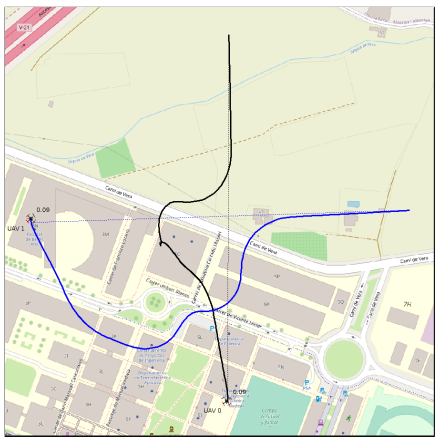
Observando el promedio, el tiempo de sobrecarga se ve reducido en 17 segundos aproximadamente, presentando un 69,1 % de reducción comparado con la media del protocolo FFP anterior. Como dato adicional, todas estas ganancias en el rendimiento se han logrado intentando respetar una mínima distancia de al menos 10 metros, según lo deseado.

Para comprender aún más como se han logrado estas mejoras, representamos de forma gráfica las trayectorias reales de dos drones en el escenario CR90 ejecutado en ArduSim, como mostramos en la Figura 3. Se puede observar que, con un vector de repulsión menor para un conflicto de 90 grados, se consigue una trayectoria de evasión menos fluctuante y más eficiente, reduciendo el tiempo de vuelo de ambos drones. En la simulación del protocolo FFP, ambos drones presentan una maniobra de evasión cuando se va a realizar la colisión, volviendo a presentar otra maniobra cuando se encuentran más al sur, después de finalizar la primera maniobra. El protocolo D-FFP disminuye el número de evasiones en 1, presentando una reducción de la velocidad del dron de trayectoria azul que permite el paso del dron con trayectoria negra. Esta reducción es generada cuando el campo de fuerza direccional del dron de trayectoria azul detecta al segundo dron, recibiendo un vector de repulsión con dirección opuesta al de atracción que, por lo tanto, reduce su vector de velocidad.

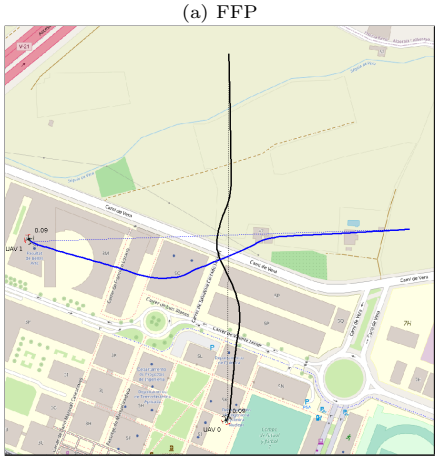
Adicionalmente, presentamos otro caso con cambios importantes en el escenario OD45, obteniendo una mejora en la trayectoria al realizar una evasión en una colisión de 45 grados entre dos drones con direcciones opuestas. La ejecución en el simulador ArduSim está representada en la Figura 4. En el caso del algoritmo FFP, podemos observar que se realiza una maniobra de evasión exagerada en ambos drones, que ocurre cuando ambos campos omnidireccionales se juntan. Visualmente, vemos que en el caso del algoritmo D-FFP, el campo direccional del dron con dirección norte presenta una repulsión muy elevada

Escenario	Mín. tiempo [s]	FFP		D-FFP	
		ST (s)	**Dist. Mín. [m]**	**ST [s]**	**Dist. Mín [m]**
CR90	46,45	27,07	62,62	5,8	20,79
SD45	45,05	34,07	92,52	10,21	22,13
OD45	44,85	24,87	26,57	4,21	12,08
HO	45,09	16,02	20,68	3,57	13,87
TO	155,77	23,27	11,02	14,91	64,31
Promedio	***67,44***	***25,06***	***42,68***	***7,74***	***26,64***

Tabla I: Resultados de la comparación de los 5 escenarios ejecutados con los protocolos FFP y D-FFP, en términos de distancia mínima entre drones y sobrecarga de tiempo (ST) respecto al tiempo mínimo.

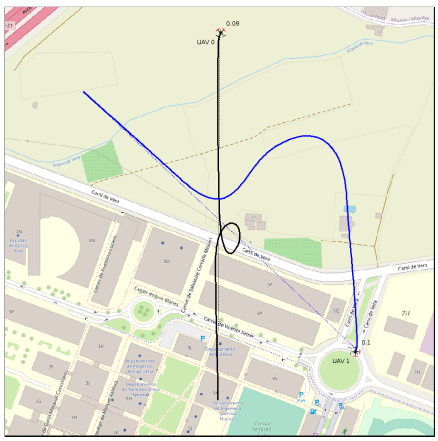


(a) FFP

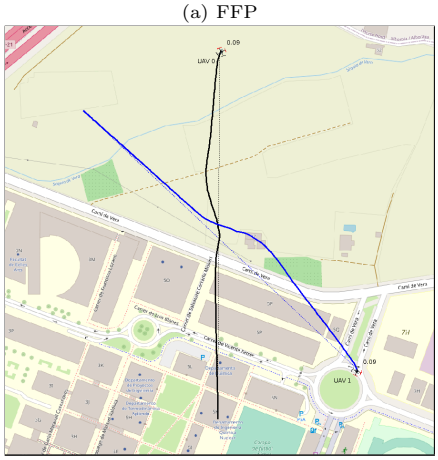


(b) D-FFP

Fig. 3: Trayectorias de vuelo de los UAVs para cada protocolo ejecutado en el escenario CR90.



(a) FFP



(b) D-FFP

Fig. 4: Trayectorias de vuelo de los UAVs para cada protocolo ejecutado en el escenario OD45.

al entrar en contacto con el otro dron, ralentizando su velocidad y permitiendo su paso, reduciendo alteraciones en la trayectoria y tiempo de la misión.

VI. Conclusiones y trabajos futuros

La masificación gradual del espacio aéreo ha introducido una nueva complejidad en la gestión de dicho espacio, ya que pueden surgir conflictos en las trayectorias de las aeronaves que pueden potencialmente dar lugar a colisiones si no se gestionan adecuadamente. En este artículo, hemos propuesto un nuevo enfoque, utilizando campos de fuerza direccionales para mejorar la gestión del espacio aéreo cuando ocurren conflictos entre aeronaves. Concretamente, mejoramos los enfoques convencionales, que se basan en el concepto de campo de fuerza, los cuales establecen patrones omnidireccionales de repulsión entre aeronaves, introduciendo un nuevo factor de direccionalidad en estos patrones. El objetivo es mejorar el rendimiento al optimizar las trayectorias de vuelo, intentando reducir al máximo las desviaciones introducidas con respecto al patrón de vuelo original.

Diferentes pruebas en escenarios de vuelo típicos muestran el potencial de esta técnica propuesta, y las mejoras respecto a las técnicas convencionales, permitiendo reducir el tiempo total de sobrecarga asociado a las maniobras de evasión de las aeronaves, siempre garantizando una distancia de seguridad entre ellas.

Como trabajo futuro, se plantearán mejoras al protocolo para permitir un funcionamiento correcto y eficiente en conflictos de carácter tridimensional. Además, se validará el protocolo propuesto mediante pruebas de vuelo reales.

Agradecimientos

El presente trabajo ha sido realizado en el marco del proyecto PID2021-122580NB-I00, financiado por MCIN/AEI/10.13039/501100011033 y por el "Fondo Europeo de Desarrollo Regional (FEDER)".

Referencias

[1] Nader Mohamed, Jameela Al-Jaroodi, Imad Jawhar, Ahmed Idries, and Farhan Mohammed, "Unmanned aerial vehicles applications in future smart cities," *Technological Forecasting and Social Change*, vol. 153, pp. 119293, 2020.

[2] Majed Alwateer and Seng W. Loke, "Emerging drone services: Challenges and societal issues," *IEEE Technology and Society Magazine*, vol. 39, no. 3, pp. 47–51, 2020.

[3] Víctor Alarcón, Manuel García, Francisco Alarcón, Antidio Viguria, Ángel Martínez, Dominik Janisch, José Joaquín Acevedo, Ivan Maza, and Aníbal Ollero, "Procedures for the integration of drones into the airspace based on u-space services," *Aerospace*, vol. 7, no. 9, 2020.

[4] Lechosław Tomaszewski, Robert Kołakowski, and Sławomir Kukliński, "Integration of u-space and 5gs for uav services," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 767–772.

[5] Enrique Aldao, Luis M. González-deSantos, Humberto Michinel, and Higinio González-Jorge, "Uav obstacle avoidance algorithm to navigate in dynamic building environments," *Drones*, vol. 6, no. 1, 2022.

[6] Flavia Causa, Armando Franzone, and Giancarmine Fasano, "Strategic and tactical path planning for urban air mobility: Overview and application to real-world use cases," *Drones*, vol. 7, no. 1, 2023.

[7] Jamie Wubben, Carlos T. Calafate, Juan-Carlos Cano, and Pietro Manzoni, "Ffp: A force field protocol for the tactical management of uav conflicts," *Ad Hoc Networks*, vol. 140, pp. 103078, 2023.

[8] Jawad N. Yasin, Sherif A. S. Mohamed, Mohammad-Hashem Haghbayan, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila, "Unmanned aerial vehicles (uavs): Collision avoidance systems and approaches," *IEEE Access*, vol. 8, pp. 105139–105155, 2020.

[9] Jung-Woo Park, Hyondong Oh, and Min-Jea Tahk, "Uav collision avoidance based on geometric approach," in *2008 SICE Annual Conference*, Loughborough, England, 09 2008, pp. 2122 – 2126, Loughborough University.

[10] Jiayi Sun, Jun Tang, and Songyang Lao, "Collision avoidance for cooperative uavs with optimized artificial potential field algorithm," *IEEE Access*, vol. 5, pp. 18382–18390, 2017.

[11] Enming Wu, Yidong Sun, Jianyu Huang, Chao Zhang, and Zhiheng Li, "Multi uav cluster control method based on virtual core in improved artificial potential field," *IEEE Access*, vol. 8, pp. 131647–131661, 2020.

[12] Ameni Azzabi and Khaled Nouri, "Path planning for autonomous mobile robot using the potential field method," in *2017 International Conference on Advanced Systems and Electric Technologies (IC_ASET), 2017, pp*,389 − −394.

[13] Changxin Huang, Wei Li, Chao Xiao, Binbin Liang, and Songchen Han, "Potential field method for persistent surveillance of multiple unmanned aerial vehicle sensors," *International Journal of Distributed Sensor Networks*, vol. 14, no. 1, pp. 1550147718755069, 2018.

[14] Daegyun Choi, Kyuman Lee, and Donghoon Kim, *Enhanced Potential Field-Based Collision Avoidance for Unmanned Aerial Vehicles in a Dynamic Environment.*

[15] Cezary Kownacki and Leszek Ambroziak, "A new multidimensional repulsive potential field to avoid obstacles by nonholonomic uavs in dynamic environments," *Sensors*, vol. 21, no. 22, 2021.

[16] Edward Burgos and Subodh Bhandari, "Potential flow field navigation with virtual force field for uas collision avoidance," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2016, pp. 505–513.

[17] B. M. Albaker and N. A. Rahim, "A survey of collision avoidance approaches for unmanned aerial vehicles," in *2009 International Conference for Technical Postgraduates (TECHPOS)*, Manhattan, New York, U.S, 2009, pp. 1–7, IEEE.

[18] Sara Pérez-Carabaza, Jürgen Scherer, Bernhard Rinner, José A. López-Orozco, and Eva Besada-Portas, "Uav trajectory optimization for minimum time search with communication constraints and collision avoidance," *Engineering Applications of Artificial Intelligence*, vol. 85, pp. 357–371, 10 2019.

[19] Wang Min, Voos Holger, and Su Daobilige, "Robust online obstacle detection and tracking for collision-freenavigation of multirotor uavs in complex environments," *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1228–1234, 2018.

[20] Niels Balemans, Peter Hellinckx, and Jan Steckel, "Predicting lidar data from sonar images," *IEEE Access*, vol. 9, pp. 57897–57906, 2021.

[21] Francisco Fabra, Carlos T. Calafate, Juan Carlos Cano, and Pietro Manzoni, "Ardusim: Accurate and real-time multicopter simulation," *Simulation Modelling Practice and Theory*, vol. 87, pp. 170–190, 2018.

Una plataforma para el diseño y evaluación de servicios UTM

Jesús Jover[1], Aurelio Bermúdez [1] y Rafael Casado[1]

Resumen— **Este artículo presenta las bases de una herramienta creada para la simulación de espacios aéreos UTM, como el definido en U-space. En concreto, la herramienta permite el desarrollo y la evaluación de servicios para la gestión del tráfico aéreo no tripulado, el diseño y la verificación del software de control de las aeronaves, y la obtención de datos y telemetría para su posterior análisis. La plataforma se construye sobre ROS, Gazebo y MATLAB, con el objetivo de ofrecer un entorno realista con una variedad de características y funcionalidades. Se presenta su estructura y funcionamiento, se proporcionan algunas indicaciones para su utilización y se muestra algún ejemplo de uso.**

Palabras clave— **Modelado, evaluación, movilidad aérea urbana (UAM), vehículos/sistemas aéreos no tripulados (UAV/UAS), gestión del tráfico aéreo no tripulado (UTM), U-space.**

I. Introducción

SE prevé que en los futuros espacios de movilidad aérea urbana (*Urban Air Mobility*, UAM) [1] el cielo de nuestras ciudades estará ocupado por multitud de vehículos áereos no tripulados (*Unmanned Aerial Vehicle/Unmanned Aerial System*, UAV/UAS). Se tratará, probablemente, de vehículos eléctricos de despegue y aterrizaje vertical (*electric Vertical Take-Off and Landing*, eVTOL), volando a baja o muy baja altitud, en muchas ocasiones de manera autónoma, y destinados al transporte de personas o bienes. Estos UAV serán operados por entidades públicas o privadas de muy diversa naturaleza.

Para gestionar de manera segura y eficiente un elevado número de UAV en el espacio aéreo urbano se hace necesario implementar mecanismos de gestión del tráfico aéreo no tripulado (*Unmanned Traffic Management*, UTM) [2]. Estos mecanismos facilitarán el intercambio de información entre UAV, operadores y otros usuarios del espacio aéreo (gestores del espacio aéreo, autoridades, etc.).

En Europa, UTM se está desarrollando en el marco de lo que se conoce como U-space [3], [4], y a través de distintos reglamentos emitidos por la Comisión Europea [5]. Tanto en el concepto de operaciones de U-space como en la regulación comunitaria se definen una serie de servicios, que son proporcionados por diferentes proveedores de servicio (de nuevo, entidades de carácter público o privado). Entre estos servicios se encuentran los de autorización, monitorización, seguimiento, o detección y resolución de conflictos.

Para desarrollar y probar todos estos servicios UTM (y cualquier otro software para UAM) antes de su despliegue en entornos reales se hace necesario disponer de plataformas de simulación como la presentada en este trabajo (U-TRAFMAN, de *U-space TRAFfic MANagement*). En concreto, esta herramienta permite simular uno o varios operadores de UAV, encargados, a su vez, de la gestión de uno o más de estos dispositivos. Además, ofrece una visualización del vuelo de dichos UAV en un entorno urbano tridimensional, de acuerdo con unos planes de vuelo preestablecidos (que incluyen secuencias de *waypoints* definiendo rutas 4D), así como un análisis posterior de la navegación.

Existen multitud de plataformas para la simulación de UAV, pero la mayoría presentan un mayor enfoque en el control o las comunicaciones de las aeronaves, y no en como UTM, U-space y su modelo de servicios funciona. Esto hace que implementar y ejecutar multitud de servicios en una simulación sea realmente complejo. Muchos de ellos, además, no ofrecen documentación alguna o incluso no están disponibles para su utilización.

En lo que se refiere a trabajos similares, en [6] se presenta una plataforma muy similar a la nuestra en cuanto a los objetos y las funcionalidades simuladas, con un gran desarrollo en la parte de la visualización gráfica del tráfico aéreo urbano. Por otro lado, y al igual que en la propuesta actual, los autores del simulador descrito en [7] emplean ROS (*Robot Operating System*) [8] para las comunicaciones y Gazebo [9] para la simulación gráfica y de físicas. El simulador está basado en la arquitectura funcional de U-space, centrándose en los servicios "in-fligth". El simulador de UTM descrito en [10] está fuertemente enfocado a la gestión de conflictos. Al igual que los anteriores, Ardusim [11] simula la operación de múltiples UAV de una manera realista, incluyendo los protocolos de comunicación entre ellos, si bien no está tan focalizado en UTM.

En este artículo se describe la arquitectura interna y la funcionalidad de la herramienta U-TRAFMAN, y se proporcionan algunos ejemplos de uso, con objeto de dar una idea de su aplicabilidad y utilidad. El resto del artículo se estructura de la siguiente manera. En primer lugar, la Sección II detalla los componentes y el funcionamiento interno de nuestra plataforma. A continuación, la Sección III describe la preparación del entorno de trabajo y el procedimiento para lanzar las simulaciones. La Sección IV presenta algunas pruebas realizadas con la plataforma. Finalmente, la Sección V recoge nuestras conclusiones y el trabajo futuro a desarrollar.

II. Arquitectura y funcionalidad

Esta sección presenta las tecnologías utilizadas por U-TRAFMAN, detalla los componentes internos de

[1]Universidad de Castilla-La Mancha, e-mail: {jesus.jover,aurelio.bermudez,rafael.casado}@uclm.es.

la arquitectura y profundiza en las funcionalidades ofrecidas.

A. Tecnologías empleadas

A.1 ROS

En ROS [8] todos los componentes de un ecosistema robótico se modelan mediante "nodos", dando lugar a una red ROS (*ROS network*). Los nodos pueden intercambian información entre ellos mediante tópicos (*topics*), servicios (*services*) y acciones (*actions*). Los tópicos siguen el modelo de comunicación de publicación/suscripción, mientras que los servicios usan el modelo petición-respuesta. Con esto, ROS permite el desarrollo de sistemas distribuidos donde la información fluye entre nodos, abstrayéndose de la tecnología de comunicación realmente existente entre ellos.

En U-TRAFMAN, ROS y su modelo de comunicación es empleado para el intercambio de información entre todas las piezas del simulador (es decir, los UAV, los operadores, los servicios UTM y otros componentes software que veremos más adelante). Cada componente implementa los tópicos y/o servicios con los que puede comunicarse con otros componentes. Un ejemplo es el tópico `uplan` disponible en cada aeronave y accesible desde `/drone/%id%/uplan` (donde `%id%` identifica a cada UAV). A través de este tópico los operadores de las aeronaves pueden enviar un plan de vuelo a las mismas.

Cuando se desarrolla para ROS, el código se organiza en paquetes (*packages*) que, a su vez, están agrupados en espacios de trabajo (*workspaces*). De esta forma, U-TRAFMAN define un único espacio de trabajo, con dos paquetes. El primero (`utrafman_main`) mantiene los componentes y los ficheros necesarios para preparar y lanzar las simulaciones, mientras que el otro paquete (`utrafman_god`) implementa un *plug-in* que controla los modelos existentes en la simulación, permitiendo añadir o eliminar aeronaves en tiempo de simulación. Cada uno de estos paquetes puede contener código para ROS, definición de mensajes, y cualquier otra información necesaria.

La estructura de la información enviada y recibida por los tópicos y servicios se debe definir haciendo uso de archivos de definición de mensajes y servicios (con extensiones .msg y .srv, respectivamente). En el caso de los tópicos, se define la información enviada en cada mensaje, mientras que para los servicios se define la información contenida en la petición y en la respuesta.

Comentar, por último, que existen multitud de herramientas que ayudan a facilitar el desarrollo en ROS, como son *rqt*, *rqt_graph* o RVIZ. Además, ROS es ampliamente usado en la comunidad robótica, de forma que dispone de API para los lenguajes más comunes.

A.2 Gazebo

Gazebo [9] es un popular simulador de dinámicas para entornos 3D con capacidad de simular de forma precisa y eficiente entornos complejos con gran población de robots. Gracias a sus motores de física, el simulador de robots Gazebo proporciona a U-TRAFMAN una simulación realista del vuelo de un grupo de UAV en un escenario tridimensional.

En concreto, el escenario a simular (*world* en la nomenclatura de Gazebo) se define en un fichero con extensión `.world`, en el que se especifica la posición, tamaño y propiedades de los diferentes modelos (*models*) que componen dicho escenario. Estos modelos pueden ser aeronaves, edificios, árboles, etc. Tanto la composición del escenario como los modelos utilizan el lenguaje SDF (*Simulation Description Format*) [12], que permite la definición de gran cantidad de propiedades en relación a las físicas, colisiones, visualización, etc.

Gazebo también permite equipar los modelos simulados con diferentes tipos de sensores, como LiDAR o cámaras. Gracias a la integración de ROS con Gazebo, estos sensores disponen de su propia funcionalidad a través de un *plug-in* de Gazebo, y tienen capacidad de enviar y recibir información a través de la red de ROS.

La funcionalidad de los modelos en Gazebo puede ser extendida mediante *plug-ins*, que son bibliotecas programadas en C++ o Python. Las API de ROS y Gazebo permiten el acceso a la información que viaja por la red, así como a los modelos de la simulación, permitiendo controlarla en su totalidad. Por ejemplo, en U-TRAFMAN las aeronaves tienen asociado un *plug-in* de control, que se encarga de que la aeronave navegue por el escenario siguiendo el plan de vuelo recibido a través del tópico `uplan` anteriormente mencionado. Más detalles sobre este *plug-in* se ofrecen en la sección II-D.

A.3 MATLAB

MATLAB [13] es un lenguaje de alto nivel y un entorno interactivo que permite realizar tareas computacionalmente intensivas. Permite el manejo matricial, el graficado de funciones y datos, así como hacer de interfaz con otros lenguajes como C, C++, Fortran o Python.

En U-TRAFMAN, MATLAB es usado principalmente con dos finalidades. Por un lado, definir todo lo necesario para llevar a cabo diversas simulaciones: creación y registro de aeronaves, operadores, generación de planes de vuelo, etc. Por otro lado, también es empleado para definir los servicios UTM presentes, como son el servicio de registro o el de monitorización, que se comentarán más adelante.

Una de las grandes ventajas de MATLAB es la disponibilidad de multitud de librerías orientadas a procesos específicos. Por otra parte, la compatibilidad con ROS está disponible a través del *add-on* "ROS Toolbox". Este implementa primitivas y acceso a la red de ROS a través de una sencilla API, haciendo el desarrollo de servicios y resto de componentes mucho más cómodo y eficiente.

En U-TRAFMAN también se emplea MATLAB para otras tareas. Un ejemplo es el procesamiento, análisis y visualización de toda la información reco-

gida durante la ejecución de las simulaciones. Otro ejemplo es el análisis de la telemetría de las aeronaves. En este sentido, más adelante detallaremos el visor de telemetría incluido en la plataforma.

Finalmente, U-TRAFMAN ofrece también la posibilidad de ejecutar los servicios UTM a través del *add-on* "Parallel Computing Toolbox". Esto permite distribuir la carga computacional en simulaciones donde estén presentes servicios UTM computacionalmente demandantes, como por ejemplo, los servicios de resolución estratégica/táctica de conflictos. Estos servicios implementan procesos complejos que requieren recursos y tiempos de respuesta garantizados. De esta forma, cada servicio UTM puede ejecutarse en un proceso diferente, o en máquinas diferentes si lo ejecutamos en un cluster, y todos ellos estarán comunicados por la red de ROS.

B. Arquitectura de U-TRAFMAN

Como se ha comentado en la sección anterior, el simulador hace uso de diferentes tecnologías. Todos estos elementos son necesarios para la ejecución de una simulación. Aunque en nuestro despliegue ROS y Gazebo suelen ejecutarse siempre en la misma máquina, no tiene porqué ser así. ROS y Gazebo pueden estar en ejecución en una máquina, mientras los servicios UTM pueden ejecutarse en una máquina diferente. El único requisito es que todas ellas se encuentren en la misma red. De esta forma, podemos distribuir la carga computacional cuando multitud de servicios UTM estén ejecutando en una simulación.

En la Figura 1 podemos observar la arquitectura en diferentes capas. Vemos como la red se encarga de comunicar todas las máquinas. La máquina 0 será la que ejecute tanto ROS como Gazebo. Este se divide en dos procesos: *gzserver*, encargado del cálculo de las físicas y dinámicas; y *gzclient*, el visor del escenario del mundo. Además, en la misma máquina se ejecuta el *plug-in* de control de cada una de las aeronaves. También se observa como podemos tener más máquinas (numeradas de 1 en adelante) ejecutando diferentes servicios UTM, desarrollados en MATLAB o en cualquier otro lenguaje, como Python. En caso de haber sido programados en MATLAB y ejecutarse en la misma máquina, como se ha comentado, existe la posibilidad de paralelizar procesos que ofrece este entorno.

C. Implementación de servicios UTM

Los servicios UTM son un conjunto de tecnologías y procedimientos utilizados para gestionar el tráfico de aeronaves en espacios aéreos de baja altitud. Son una de las partes más importantes dentro de la iniciativa europea U-space, destinada a establecer un marco regulatorio y tecnológico para habilitar operaciones seguras y eficientes de aeronaves no tripuladas.

Un ejemplo de servicio UTM es el seguimiento de aeronaves. Este servicio recibe información relativa a la posición de las aeronaves y lleva a cabo los cálculos necesarios para monitorizar y rastrear la ubicación de estas, su velocidad, altitud y trayectoria. De esta forma, la información almacenada y generada por este servicio puede ser consumida por otro servicio o servir para la toma de decisiones y coordinación del espacio aéreo.

Aunque U-space no ofrece una descripción técnica de cómo deben implementarse estos servicios, es lógico pensar que muchos de ellos seguirán la arquitectura cliente-servidor si se espera respuesta, o la arquitectura de tópicos si la información solo fluye de un origen a un destino.

Como se ha comentado, los servicios UTM en U-TRAFMAN han sido implementados en MATLAB. No obstante, como la comunicación entre los servicios se realiza mediante tópicos y servicios ROS, los servicios pueden extenderse o re-implementarse en cualquier lenguaje que soporte ROS.

Actualmente, U-TRAFMAN implementa dos servicios UTM básicos, que dan apoyo a otros, proporcionándoles información actualizada del estado del espacio aéreo. Son los siguientes:

C.1 Servicio de Registro

El servicio de registro mantiene información sobre las entidades existentes en el espacio aéreo. Cada vez que un operador, un UAV o un plan de vuelo es creado, debe quedar registrado en este servicio. Para ello, ofrece una serie de servicios ROS. De forma similar, también ofrece servicios para la consulta de datos del registro.

C.2 Servicio de Monitorización

El servicio de monitorización recibe y almacena los datos de telemetría proporcionados por las todas las aeronaves en vuelo. Este servicio está suscrito al tópico de telemetría de cada una de las aeronaves, de forma que, cada vez que un mensaje de telemetría es enviado por una aeronave, es recibido y almacenado por este servicio de monitorización. Además, proporciona también servicios para acceder y filtrar esta información.

D. Software a bordo de las aeronaves

Cada aeronave dispone de un "software embarcado"que se encarga de la navegación autónoma de la aeronave. Este software recibe los planes de vuelo que debe ejecutar a través del tópico `uplan` y se encarga de su ejecución en el tiempo indicado. De igual forma, cada aeronave, también a través de un tópico, puede enviar información sobre su estado (lo que llamamos telemetría) a su operador y a los servicios UTM.

Este comportamiento está dirigido por un software de control de bucle cerrado que se ejecuta con una frecuencia de 1000, 500 o 250 Hz. En cada ejecución, compara el estado real de la aeronave con su plan de vuelo, y aplica las modificaciones necesarias para conducir la aeronave al estado deseado, permitiendo responder a posibles desviaciones o influencias externas (viento, ruido en los sensores, etc). En el caso de un cuadricóptero, modelo de aeronave incluido en U-TRAFMAN, esto se traduce en modificar la veloci-

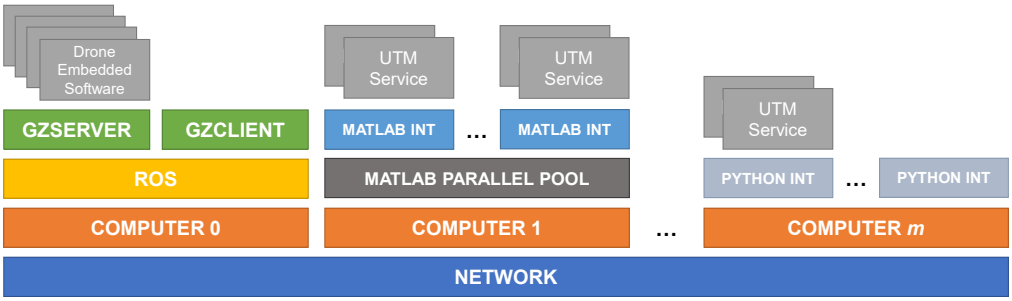


Fig. 1: Arquitectura

dad con la que las hélices giran, actuando sobre cada uno de sus motores. El software embarcado también genera los mensajes de telemetría que informan del estado de la aeronave a los servicios UTM.

Este se implementa por medio de un *plug-in* de modelo para Gazebo. Además del control, el *plug-in* también traduce las señales y velocidades de rotación de las hélices a fuerzas aplicadas sobre el modelo de la aeronave en Gazebo. Si el lector está interesado puede consultar [14] para conocer todos los detalles sobre el control incluido en las aeronaves en este simulador.

III. Preparación y ejecución de simulaciones

A. Disponibilidad del simulador

El simulador puede encontrarse de forma pública en el siguiente repositorio: `https://github.com/I3A-NavSys/utrafman_sim`. La documentación completa del mismo, incluyendo su instalación, puede encontrarse en `https://i3a-navsys.github.io/utrafman_sim/#/`.

B. Ejecución de simulaciones

En primer lugar, debemos iniciar el núcleo de ROS y Gazebo y cargar en Gazebo el escenario (el mundo) a simular. La integración de Gazebo con ROS permite lanzar ambos entornos con un único comando, en este caso: `roslaunch utrafman_main %simulacion%.launch`. El fichero `%simulación%.launch` contiene los componentes a ejecutar, y se encuentra en el directorio `/launch` del paquete `utrafman_main`. Por otro lado, el archivo del mundo se encuentra en el directorio `/world`.

Una vez ROS y Gazebo están en ejecución, el resto de tareas se llevarán a cabo desde MATLAB. Un *script* se encargará de conectarse con la red de ROS, iniciar la ejecución de los servicios UTM, insertar aeronaves en la simulación, generar las rutas que deben seguir y proceder con el envío de los planes de vuelo. Estos *scripts* se encuentra en el directorio `/src/matlab/simulations`.

Durante la simulación se pueden añadir nuevas aeronaves y eliminar las existentes. También se pueden generar nuevos planes de vuelo, que serán enviados a las aeronaves para su ejecución. Mientras una aeronave permanezca en la simulación enviará mensajes de telemetría con una frecuencia preestablecida de 1 Hz. Esta información podrá ser consumida por los servicios en tiempo real, o almacenada y analizada a posteriori.

La Figura 2 muestra el mundo definido en el fichero `generated_city.launch`. Tras ejecutar el *script* de MATLAB situado en `simulations/test_simulation.m`, ocurre lo siguiente:

1. Establecimiento de la conexión de MATLAB con la red de ROS.
2. Iniciación de los servicios UTM.
3. Registro de un operador de UAV.
4. Creación de 30 UAV, registro de los mismos en el servicio de registro e inicio del seguimiento por parte del servicio de monitorización.
5. Creación de 30 planes de vuelo y registro de los mismos en el servicio de registro.
6. Envío de los planes de vuelo a su respectivo UAV.

La Figura 3 muestra algunos UAV siguiendo su plan de vuelo en la simulación. La secuencia de interacción llevada a cabo por las distintas entidades simuladas puede observarse en la Figura 5:

1. Se crea una instancia de operador, que es enviada al servicio de registro (a través de un servicio de ROS). El servicio de registro contesta, autorizando el operador y asignándole un identificador (X).
2. Se crea una instancia de UAV, que es enviada al servicio de registro. Si se autoriza, el servicio de registro proporciona un identificador (Y) al UAV y llama al servicio de inserción de UAV en la simulación. Además, el servicio de registro informa a todo interesado sobre la inserción de un nuevo UAV en la simulación. El servicio de registro devuelve el identificador al UAV insertado.
3. El servicio de monitorización recibe la advertencia de la inserción de un nuevo UAV, y se suscribe a su tópico de telemetría.
4. Se crea una instancia de un plan de vuelo, y se envía al servicio de registro para su registro y autorización. El servicio de registro, con la ayuda de otros servicios, tomará una decisión. Por el momento, en U-TRAFMAN, todos los planes de vuelo son aceptados sin comprobaciones adicionales. El servicio de registro responderá entonces

con el plan de vuelo aceptado y su identificador (Z).

5. Con el plan de vuelo aceptado, el UAV debe comenzar a volar en el tiempo indicado en dicho plan. El operador enviará el plan de vuelo al UAV para su realización.
6. El UAV recibe el plan de vuelo y lo ejecuta, enviando continuamente (cada segundo) mensajes de telemetría.

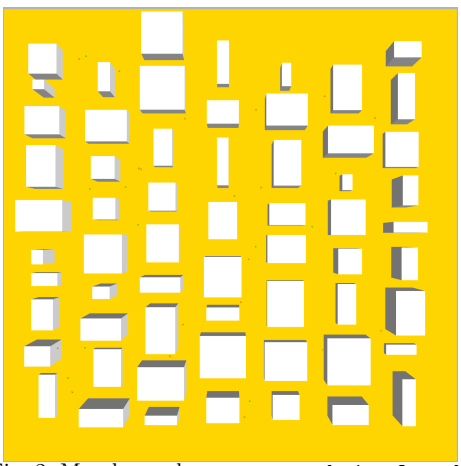


Fig. 2: Mundo usado en `generated_city.launch`.

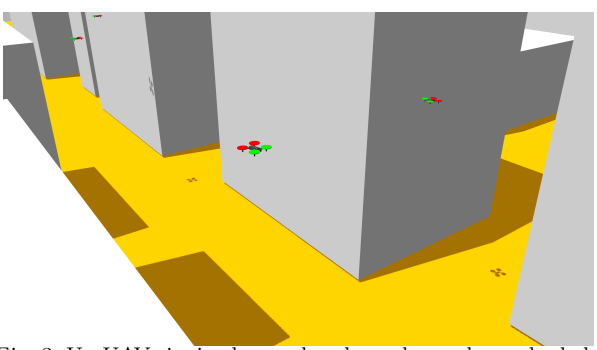


Fig. 3: Un UAV siguiendo un plan de vuelo en el mundo de la Figura 2.

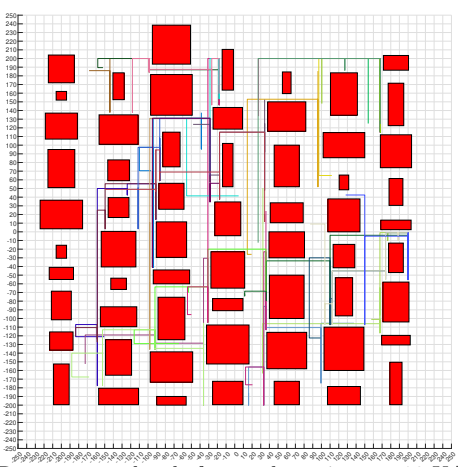


Fig. 4: Rutas generadas de forma aleatoria para 30 UAV en el mundo de la Figura 2.

IV. Pruebas y resultados

Esta sección describe algunas pruebas de escalabilidad, ejecución prolongada y la herramienta de análisis de telemetría de U-TRAFMAN. Las pruebas, y por tanto, la ejecución completa (ROS, Gazebo y MATLAB) del simulador ha sido llevada a cabo en una máquina con procesador i9-10900K, con 128 GB de memoria y sistema operativo Ubuntu 20.04.

A. Prueba de escalabilidad

La primera prueba realizada ha tenido como objetivo comprobar el límite de escalabilidad del simulador, así como el factor de tiempo real al que se puede ejecutar. Para ello, hemos hecho uso de diferentes configuraciones en el bucle de ejecución. La prueba se ha realizado en el mismo mundo mostrado en la Figura 2, de 500 x 500 metros.

El factor de tiempo real en Gazebo es una medida que indica la relación entre el tiempo simulado y el tiempo real (en el mundo real). Un factor de tiempo real menor que 1 implica que la simulación va más lenta que el tiempo real, indicando que el simulador y/o los recursos de la máquina no son capaces de procesar las tareas lo suficientemente rápido. Por otro lado, un factor de tiempo real mayor que 1 indica lo contrario, es decir, que la simulación se está ejecutando a una velocidad mayor al tiempo real. La Figura 6 muestra cómo varia el factor de tiempo real a medida que aumenta el número de UAV en la simulación, para diferentes configuraciones. La prueba se ha repetido variando de 10 a 500 la cantidad de UAV en el espacio aéreo, en pasos de 10 UAV. A medida que aumenta el número de UAV en el espacio aéreo, el factor de tiempo real es menor. Era de esperar, pues el simulador debe ejecutar el software a bordo de cada UAV. Además, en la figura se muestran tres configuraciones (series), dependiendo de la frecuencia con la que se ejecuta el software de navegación (1000Hz, 500Hz o 250 Hz). Ejecutar este software 1000 veces por segundo permite una navegación más estable de las aeronaves, pero, obviamente, penaliza el tiempo de simulación. Sin embargo, ejecutarlo 250 veces por segundo permite una navegación con un buen rendimiento, consiguiendo un factor de tiempo real en torno a 2.7 veces mayor.

B. Prueba de ejecución prolongada

La siguiente prueba llevada a cabo ha tenido como objetivo comprobar el correcto funcionamiento de la plataforma para la ejecución de simulaciones de larga duración, es decir, de más de 24 horas (de tiempo simulado).

Para ello, se ha simulado un espacio aéreo con una superficie de 500 x 500 metros y con 20 UAV en vuelo, el mismo mostrado en la Figura 2. Durante la simulación, cada UAV ha ejecutado 148 planes de vuelo, con una longitud media de aproximadamente 600 metros. Además, cada UAV ha enviado alrededor de 86,000 mensajes de telemetría (`utrafman_main/Telemetry`), que han sido almacenados por el servicio de monitorización para su posterior análisis. La distancia media recorrida por cada UAV ha sido de 95.3 ± 2.3 km y el tiempo real transcurrido han sido 5 horas y 14 minutos, concluyendo

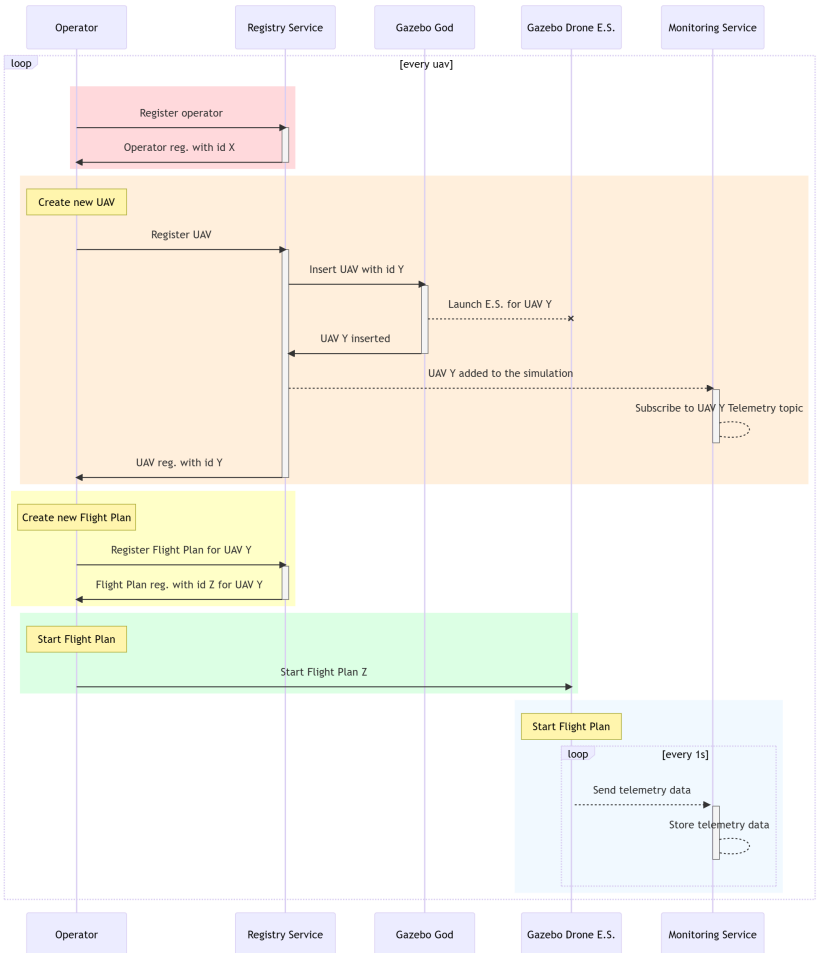


Fig. 5: Secuencia de interacción entre las distintas entidades en una simulación.

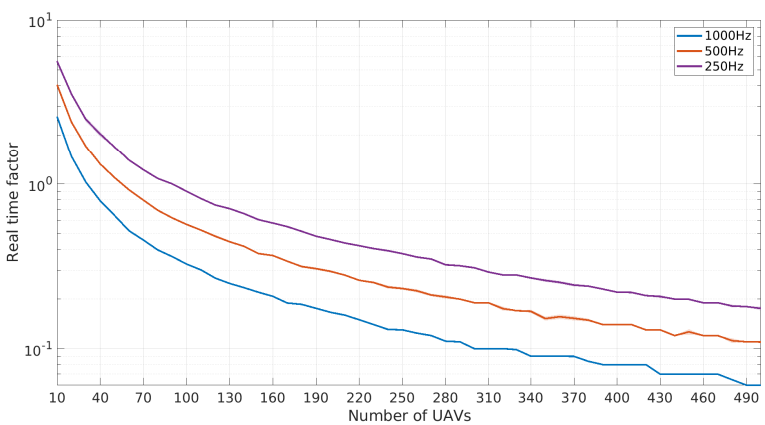


Fig. 6: Variación del factor de tiempo real en función del número de UAV en la simulación. Escala logarítmica.

en un factor de tiempo real de 4.58.

C. *Visor de telemetría*

Gracias a que todos los mensajes de telemetría que envían las aeronaves durante la simulación quedan almacenados, podemos analizar el correcto funcionamiento del simulador y/o los servicios ejecutados. Con este propósito, se ofrece la herramienta *visor de telemetría* (ver Figura 7), que permite, dado un plan de vuelo, visualizar el comportamiento de la aeronave durante su ejecución, y en comparación con lo que le fue encomendado. Esta herramienta es de gran utilidad a la hora de estudiar las consecuencias que pueda tener un cambio en el sistema de control de las aeronaves o en los servicios que modifiquen las rutas seguidas por las mismas.

El visor se descompone en 4 secciones. En la parte superior izquierda aparece una vista 3D de la ruta de referencia (la contenida en el plan de vuelo) y de la ruta realizada por la aeronave durante la simulación. Los *waypoints* de la ruta de referencia se representan mediante círculos de color rojo. En la parte inferior izquierda podemos ver tanto la rotación de la aeronave sobre el eje Z (*yaw*) como la velocidad con la que esta se produce. En la parte superior derecha podemos ver la posición en el tiempo de la aerona-

ve (descompuesta en los ejes X, Y y Z). De manera análoga, en la parte inferior derecha aparece su velocidad. En todas las gráficas podemos observar, en color naranja, la referencia aportada por el plan de vuelo, mientas que en color azul se muestra la información recogida durante la simulación.

V. Conclusiones y trabajo futuro

En este artículo hemos presentado nuestra plataforma U-TRAFMAN, enfocada al diseño e implementación de servicios para U-space, así como al control de la navegación de las aeronaves. Puesto que nuestro objetivo es poner la herramienta a disposición de la comunidad investigadora, se ha detallado su instalación, configuración y funcionamiento, y se han ofrecido medidas de su rendimiento y ejemplos de su capacidad de análisis.

Como trabajo futuro, por un lado pretendemos ampliar U-TRAFMAN con un mayor número de servicios U-space y herramientas para su verificación. Por otra parte, desarrollaremos algoritmos para el diseño de planes de vuelo óptimos y libres de colisión (gestión estratégica de conflictos), para la gestión táctica de conflictos, para la la localización de los UAV cuando los sistemas GNSS fallan o no están disponibles, o sistemas de detección y evitación de colisiones (*detect and avoid*, DAA). También pretendemos emplear U-TRAFMAN para el estudio y la planificación de vertipuertos, otra de las piezas clave de U-space.

Por último, nos planteamos la incorporación de software de control de vuelo ya existentes, como es el caso de PX4 Autopilot. Esto permitiría un navegación más configurable, además de que la comunidad y documentación existente es mayor.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Ciencia e Innovación, la Junta de Comunidades de Castilla-La Mancha, y la Unión Europea (Fondo Europeo de Desarrollo Regional), a través de los proyectos PID2021-123627OB-C52 y SBPLY/19/180501/000159, y por la Universidad de Castilla-La Mancha a través de la ayuda 2023-GRIN-34056.

Referencias

[1] Anna Straubinger, Raoul Rothfeld, Michael Shamiyeh, Kai-Daniel Büchter, Jochen Kaiser, and Kay Olaf Plötner, "An overview of current research and developments in urban air mobility – Setting the scene for UAM introduction," *Journal of Air Transport Management*, vol. 87, pp. 101852, Aug. 2020.

[2] Tao Jiang, Jared Geller, Daiheng Ni, and John Collura, "Unmanned Aircraft System traffic management: Concept of operation and system architecture," *International Journal of Transportation Science and Technology*, vol. 5, no. 3, pp. 123–135, Oct. 2016.

[3] CORUS-XUAM consortium, "U-space ConOps (edition 3.10)," `https://corus-xuam.eu/new-u-space-conops/`, July 2022, Accesed 24-05-2024.

[4] Cristina Barrado, Mario Boyero, Luigi Brucculeri, Giancarlo Ferrara, Andrew Hately, Peter Hullah, David Martin-Marrero, Enric Pastor, Anthony Peter Rushton, and Andreas Volkert, "U-Space Concept of Operations: A Key Enabler for Opening Airspace to Emerging Low-Altitude Operations," *Aerospace*, vol. 7, no. 3, pp. 24, Mar. 2020.

[5] Comisión Europea, "Reglamento de Ejecución (UE) 2021/666 de la Comisión de 22 de abril de 2021 por el que se modifica el Reglamento (UE) nº 923/2012 en lo que se refiere a los requisitos para la aviación tripulada que opera en el espacio aéreo U-Space," `https://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:32021R0666`, Accesed 24-05-2024.

[6] Jimmy Johansson Westberg, Karljohan Lundin Palmerius, and Jonas Lundberg, "UTM City—Visualization of Unmanned Aerial Vehicles," *IEEE Comput. Grap. Appl.*, vol. 42, no. 5, pp. 84–89, Sept. 2022.

[7] Jose A. Millan-Romera, Jose Joaquin Acevedo, Angel R. Castano, Hector Perez-Leon, Carlos Capitan, and Anibal Ollero, "A UTM simulator based on ROS and Gazebo," in *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, Cranfield, United Kingdom, Nov. 2019, pp. 132–141, IEEE.

[8] Open Source Robotics Foundation, "ROS - Robot Operating System," `https://ros.org/`, Accesed 20-03-2023.

[9] Open Source Robotics Foundation, "Gazebo," `https://gazebosim.org/home`, Accesed 20-03-2023.

[10] Sugjoon Yoon, Dongcho Shin, Younghoon Choi, and Kyungtae Park, "Development of a Flexible and Expandable UTM Simulator Based on Open Sources and Platforms," *Aerospace*, vol. 8, no. 5, pp. 133, May 2021.

[11] Francisco Fabra, Carlos T. Calafate, Juan Carlos Cano, and Pietro Manzoni, "ArduSim: Accurate and real-time multicopter simulation," *Simulation Modelling Practice and Theory*, vol. 87, pp. 170–190, Sept. 2018.

[12] Open Source Robotics Foundation, "SDFormat," `http://sdformat.org/`, Accesed 21-03-2023.

[13] The Mathworks, Inc., "MATLAB y Simulink para ingeniería de sistemas basada en modelos (MBSE)," `https://es.mathworks.com/solutions/model-based-systems-engineering.html`, Accesed 03-05-2023.

[14] Aurelio Bermúdez, Rafael Casado, Guillermo Fernández, María Guijarro, and Pablo Olivas, "Drone challenge: A platform for promoting programming and robotics skills in K-12 education," *International Journal of Advanced Robotic Systems*, vol. 16, no. 1, pp. 172988141882042, Jan. 2019.

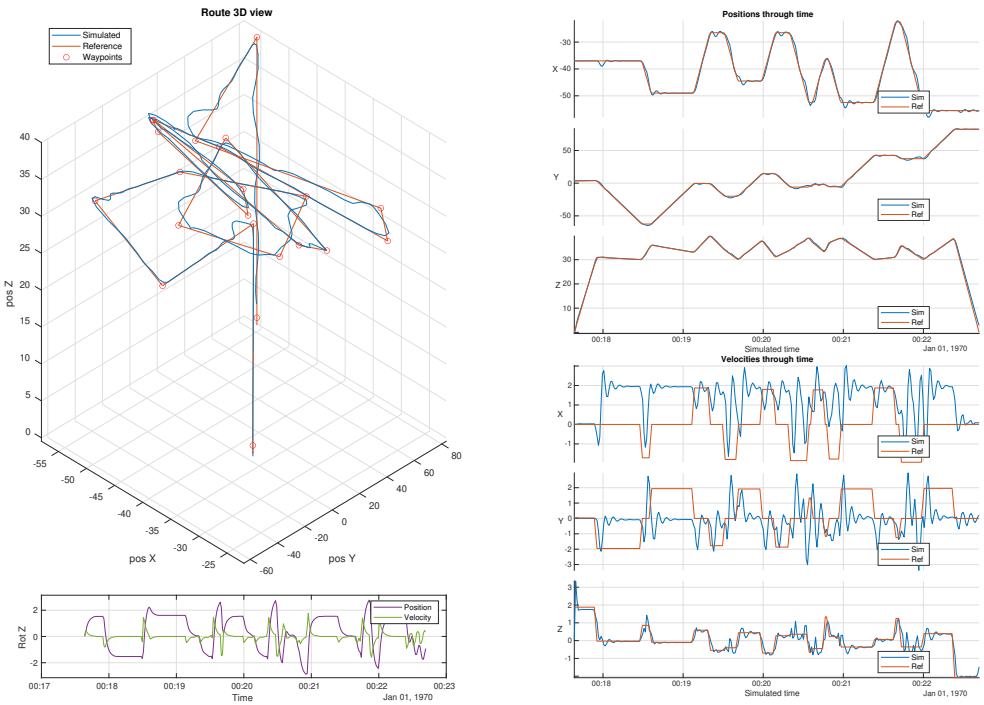


Fig. 7: Visor de telemetría de vuelo. Plan de vuelo frente a vuelo realizado.

Uso de redes ad hoc para comunicaciones entre drones: estado actual y perspectivas futuras

Iker Nebot[1], Jamie Wubben[1], Enrique Hernández-Orallo[1] y Carlos T. Calafate[1]

Resumen— **El estándar IEEE 802.11 es relevante para la creación de redes ad hoc debido a su amplia adopción, sus especificaciones técnicas y protocolos que permiten la comunicación inalámbrica eficiente, su capacidad para establecer redes de forma autónoma y su oferta de mecanismos de seguridad. Estas características hacen que el estándar IEEE 802.11 sea una opción ideal para habilitar la comunicación ad hoc en drones y otros dispositivos inalámbricos. No obstante nos encontramos que, en la práctica, hay diversas barreras que obstaculizan el uso de esta tecnología. Por esa razón, en este trabajo se realiza un estudio del estado actual de esta tecnología, analizando los diferentes dispositivos presentes en el mercado en cuanto a soporte al modo ad hoc y funcionamiento en la banda de los 5 GHz. Además, se realiza un estudio del rendimiento de esta tecnología en base a pruebas reales. Por último, se propone un Algoritmo para la definición de un identificador de red (BSSID) ad hoc en base al servicio que se está usando, que reemplazaría el actual sistema basado en un identificador aleatorio.**

Palabras clave— **IEEE 802.11, redes ad hoc, drones, evaluación de rendimiento.**

I. Introducción

LOS drones han experimentado avances significativos en los últimos años, revolucionando diversas industrias como la vigilancia, servicios de entrega y respuesta ante desastres [1]. Sin embargo, una comunicación confiable y eficiente entre los drones sigue siendo un desafío crucial, especialmente en entornos dinámicos e impredecibles. En dichos contextos, las comunicaciones ad hoc [2] surgen como una opción flexible para abordar estos desafíos. Concretamente, permiten a los drones ajustar dinámicamente su configuración de red y cambiar sus vecinos de comunicación según las condiciones del entorno [3], asegurando una conectividad continua, incluso en escenarios donde la infraestructura de comunicación tradicional es inaccesible o inestable [4].

El estándar IEEE 802.11 [5], también conocido como Wi-Fi, es en la actualidad la mejor opción tecnológica a la hora de establecer cualquier tipo de red ad hoc por varias razones. En primer lugar, el estándar IEEE 802.11 ha sido ampliamente adoptado y utilizado en dispositivos y equipos de comunicación inalámbrica en todo el mundo. Esto significa que la mayoría de los dispositivos actuales, incluidos los drones, están equipados con capacidades Wi-Fi, lo que facilita su integración en redes ad hoc basadas en este estándar. En segundo lugar, el estándar IEEE 802.11 permite que los dispositivos, como los drones, establezcan redes de forma autónoma sin depender de una infraestructura centralizada. Los dispositivos pueden descubrir y comunicarse directamente entre sí utilizando las capacidades del modo ad hoc del estándar IEEE 802.11, formando rápidamente redes dinámicas y flexibles en tiempo real.

En este contexto, el anexo "p"del estándar IEEE 802.11 [5] ha surgido como una solución de conectividad especializada para redes entre vehículos en movimiento, buscando dar soporte a aplicaciones de seguridad vial, gestión de tráfico inteligente, y conducción cooperativa. Concretamente, esta tecnología utiliza la banda de frecuencias de 5.9 GHz para comunicaciones de vehículo a vehículo (V2V) y de vehículo a infraestructura (V2I) [6], introduciendo diversas mejoras respecto al estándar base. Estas mejoras incluyen una mayor potencia de transmisión y una mayor sensibilidad de recepción, lo cual permite una mayor cobertura de la señal inalámbrica. Además, utiliza técnicas de modulación robustas, como la modulación de frecuencia ortogonal (OFDM), que ofrecen una mayor resistencia a las interferencias y al desvanecimiento de la señal en entornos móviles.

Otra característica importante del estándar IEEE 802.11p es su baja latencia. La comunicación entre vehículos requiere una respuesta rápida y en tiempo real, no solo para apoyar su operación segura, sino también para permitir la coordinación eficiente [7]. De esta manera, la baja latencia del estándar 802.11p permite una comunicación ágil entre vehículos, lo cual resulta fundamental para aplicaciones de seguridad vial y conducción cooperativa.

Las FANETs (Flying Ad Hoc Networks [8]) son redes ad hoc específicamente diseñadas para drones, donde los propios drones establecen una red de comunicación entre ellos de manera autónoma. Sin embargo, a diferencia de las redes vehiculares, las FANETs no disponen de un estándar de comunicaciones específico, lo que conlleva varios problemas.

En primer lugar, la falta de un estándar de comunicaciones dificulta la interoperabilidad y la compatibilidad entre drones de diferentes fabricantes. Cada fabricante puede implementar sus propios protocolos y mecanismos de comunicación, lo cual en la práctica impide la formación de redes ad hoc entre drones de diferentes marcas o modelos; además, el modo ad hoc no suele estar soportado por defecto. Esto restringe la capacidad de dichos drones para colaborar y comunicarse eficientemente en un entorno de FANETs diverso. Además, la falta de un estándar de comunicaciones ad hoc para FANETs dificulta la implementación de mecanismos de seguridad y gestión de calidad de servicio (QoS), ya que cada fabricante puede tener su propia implementación.

[1]Dpto. de Informática de Sistemas y Computadores (DISCA), Universitat Politécnica de València, e-mail: inebamo@posgrado.upv.es, {jwubben,ehernandez, calafate}@disca.upv.es

En este artículo se presenta una visión de la situación actual en cuanto a soporte al modo ad hoc en redes WiFi, destacando además el rendimiento de esta tecnología. En base a los datos ofrecidos, se realiza una propuesta que busca acelerar la adopción de comunicaciones ad hoc en entornos de FANETs en base a un Algoritmo de generación de BSSID único por servicio.

El resto de este documento está organizado de la siguiente manera: en la Sección II se ofrece una visión de la situación actual en cuanto a soporte al modo ad hoc en tarjeta IEEE 802.11, tanto al nivel de chip como al nivel de driver. En la Sección III se detalla el rendimiento de esta tecnología en la banda de los 5 GHz en base a experimentos reales. A continuación, en la Sección IV, se presenta un novedoso Algoritmo que permite mejorar la adopción de redes ad hoc a grande escala mediante la definición de BSSID únicos por servicio. Por último, en la Sección V, se presentan las principales conclusiones del artículo, con referencia a trabajo futuro.

II. Soporte al modo ad hoc: situación actual

En esta Sección se va a ofrecer una visión sobre el abanico de posibilidades que actualmente existe respecto al soporte al modo ad hoc entre los diferentes chipsets y drivers, haciendo hincapié en los requisitos que tenemos para su uso en drones.

A. Chips

En el mercado de los adaptadores de red inalámbricos, existe una amplia selección de chips entre la que elegir. Destacan fabricantes como Realtek o Mediatek, quienes venden sus productos a terceros que desarrollan las soluciones Wi-Fi que integran estos chips.

Sin embargo, la gran mayoría de adaptadores de red inalámbricos están diseñados para funcionar en sistemas operativos Windows, dejando de lado los GNU/Linux.

Muchos equipos de investigación y desarrolladores de software utilizan Linux en sus proyectos, gracias al rendimiento y la flexibilidad que aporta frente a las alternativas más comerciales. En nuestro caso, queremos desplegar un software de simulación, control y monitorización de enjambre de drones [9] para controlar los nodos de la red adhoc.

Los requisitos para configurar la red ad hoc y desplegar el software adicional en el ordenador de a bordo del dron son:

- Soporte desde el sistema operativo Linux.
- Chipset compatible con redes ad hoc.
- Chipset compatible con la banda de los 5GHz.
- Interfaz USB.

Tras comprobar las especificaciones de los chipsets que se muestran en las Tablas I y II, tan solo 42 de los 192 ($\approx 21,9\,\%$) son compatibles con Linux y cumplen los requisitos necesarios para el experimento.

Tabla I: Chipsets compatibles con sistemas operativos Linux agrupados por driver (de A a L)

Driver	Chipsets
acx100	Texas Instruments ACX100, ACX111, TNETW1450
adm8211	ADMtek ADM8211
airo	Aironet 4500, 4800, 340, 350
ar5523	Atheros AR5523
at76c50x-USB	Atmel AT76C503, AT76C505
ath5k	Atheros AR2413, AR2414, AR2417, AR2425, AR5210, AR5211, AR5212, AR5213, AR5413, AR5414, AR5423, AR5424
ath6kl	Atheros AR6003, AR6004
ath9k	Qualcomm Atheros chips with IEEE 802.11n support
ath9k_htc	Atheros AR9271, AR7010
ath10k	Qualcomm Atheros chips with IEEE 802.11ac support
ath11k	Qualcomm Atheros chips with IEEE 802.11ax support
atmel	Atmel at76c502, at76c504, at76c506
b43	Some Broadcom 43xx
b43legacy	Broadcom 4301, 4303, 4306
brcmfmac	Broadcom 4356, 43567, 43570, 4358, 4359, 43602, 4365, 4366, 4329, 4330, 4334, 43340, 43341, 43241, 4335, 4339, 43362, 43430, 43455, 4354, 43143, 43235, 43236, 43238, 43143, 43242, 43566, 43569
brcmsmac	Broadcom 4313, 43224, 43225
carl9170	Atheros AR9170
cw1200	ST-Ericsson CW1100, CW1200
HostAP	Intersil PRISM-II, PRISM-2.5, PRISM 3
ipw2x00	Intel 2100 and 2200
iwlegacy	Intel 3945ABG, 4965AGN
iwlwifi	Intel 6250AGN, 6200AGN, 6300AGN, 1000BGN, 5150AGN, 5100AGN, 5300AGN, 5350AGN, 6005, 6030, 6150BGN, 100BGN, 130BGN, 2000
libertas	Marvell 88W8686, 8388, 8385, 8385, 8686, 8688, 8686, 88W8388

De esta selección hay que descartar los chipsets cuyos drivers no son compatibles con las últimas versiones de Linux, no tienen soporte para arquitecturas ARM, o están descatalogados.

B. Drivers

Como se ha comentado en el apartado II-A, la mayoría de las empresas que fabrican adaptadores de red se centran en sistemas operativos Windows. Por ello, es frecuente encontrar chips que no disponen de drivers para sistemas basados en UNIX. Adicionalmente, los fabricantes suelen proteger el código de los chips que producen, lo que obliga a los usuarios a depender de drivers no oficiales, desarrollados y mantenidos por particulares, que no disponen de los mismos recursos que los fabricantes. Como consecuencia, los drivers presentan numerosos problemas de compatibilidad con otras arquitecturas o versiones avanzadas del kernel de Linux, fallos de compilación, implementaciones incompletas, o errores en los apartados menos utilizados, como las redes ad hoc o la banda de 5GHz.

En la Tabla III se muestra a modo resumen aquellas combinaciones driver-chipset que cumplen con todos los requisitos definidos anteriormente de cara a hacer viable un despliegue en drones. Como se puede constatar, las opciones son reducidas. No obstante,

Tabla II: Chipsets compatibles con sistemas operativos Linux agrupados por driver (de M a Z)

Driver	Chipsets
mt76	MediaTek MT76x0U, MT76x2U
mt7601u	MediaTek MT7601U
mwifiex	Marvell SD8786, SD8787, SD8797, 8766, 8897, USB8797
mwl8k	Marvell 88W8366, 88W8863, 88W8687, 88W8764
orinoco	Lucent Hermes, Intersil PRISM-II, PRISM-2.5, Symbol Spectrum24
p54	ISL3877, ISL3880, ISL3886, ISL3887, ISL3890
rt61pci	RT2561, RT2561S, RT2661
rt73usb	RT2571W, RT2573, RT2671
rt2400pci	MediaTek (Ralink) RT2460
rt2500pci	RT2560
rt2500usb	RT2571, RT2572
rt2800pci	RT2760, RT2790, RT2860, RT2880, RT2890, RT3052, RT3090, RT3091, RT3092, RT3390, RT3060, RT3062, RT3562, RT3592, RT5390, RT3290
rt2800usb	RT2770, RT2870, RT3070, RT3071, RT3072, RT3370, RT3572, RT5370, RT5572
rtl819x	RTL8723AU, RTL8723BU, RTL8188CUS, RTL8192CU, RTL8191EU, RTL8192EU, RTL8188EU, RTL8188RU, RTL8188FU, RTL8192FU
rtl8180	RTL8180, RTL8185, RTL8187SE
rtl8187	RTL8187, RTL8187B
rtl8188ee	RTL8188EE
rtl8188eu	RTL8188EU
rtl8821ae	RTL8812AE, RTL8821AE
rtw88_8723de	RTL8723DE
rtw88_8723du	RTL8723DU
rtw88_8821ce	RTL8821CE
rtw88_8821cu	RTL8821CU
rtw88_8822be	RTL8822BE
rtw88_8822bu	RTL8822BU
rtw88_8822ce	RTL8822CE
rtw88_8822cu	RTL8822CU
rtw89_8852ae	RTL8852AE
rtw89_8852be	RTL8852BE
rtw89_8852ce	RTL8852CE
wil6210	Wilocity wil6210
wl12xx	Texas Instruments wl1271, wl1273, wl1281, wl1283
wl18xx	Texas Instruments WiLink 8
wl1251	Texas Instruments wl1251
zd1211rw	ZyDAS ZD1211/ZD1211B

cabe matizar que, con cada actualización del kernel de Linux, se van añadiendo nuevos drivers compatibles. Es recomendable utilizarlos siempre que sea posible, aunque suele ser más difícil encontrar un adaptador de red que utilice un chipset compatible. En nuestro caso, hemos utilizado el chipset RT3572 para las pruebas experimentales.

III. Rendimiento en la banda de 5 GHz: Resultados experimentales

A diferencia de una red en modo infraestructura, la arquitectura de una red ad hoc no tiene un punto de acceso centralizado; en cambio, los nodos de la red se comunican entre ellos directamente.

Aunque como norma general una red en modo infraestructura proporciona más estabilidad y rendimiento, en función de la aplicación hay casos en los que una red ad hoc puede ser una alternativa interesante.

Tabla III: Combinaciones driver-chipset que cumplen todos los requisitos para el despliegue en drones.

Driver	Chipsets
ath6kl	Atheros AR6004
brcmfmac	Broadcom 43235, 43236, 43238, 43143, 43242, 43566, 43569
carl9170	Atheros AR9170
mt76	MediaTek MT7610U, MT7612, MT7602, MT7662, MT7663, MT7921
mwifiex	Marvell USB8797
p54	ISL3886, ISL 3887
rt73usb	RT2571W, RT2573, RT2671
rt2500usb	RT2571, RT2572
rt2800usb	RT2770, RT2870, RT3070, RT3071, RT3072, RT3370, RT3572, RT5370, RT5572
rtw88_8723de	RTL8723DE
rtw88_8723du	RTL8723DU
rtw88_8821ce	RTL8821CE
rtw88_8821cu	RTL8821CU
rtw88_8822be	RTL8822BE
rtw88_8822bu	RTL8822BU
rtw88_8822ce	RTL8822CE
rtw88_8822cu	RTL8822CU
zd1211rw	ZyDAS ZD1211/ZD1211B

Por ejemplo, cuando dos nodos de una red ad hoc quieren comunicarse, si la distancia que los separa es inferior al alcance del adaptador de red de ambos nodos, el ancho de banda de la comunicación podría llegar a duplicarse [10], ya que los paquetes no han de pasar por un punto de acceso para luego ser reenviados al nodo destino.

Esto es especialmente útil en una red de enjambre de drones, ya que la mayoría de las comunicaciones se realizan entre nodos cercanos con el objetivo de coordinar las maniobras de vuelo y evitar posibles colisiones, por lo que duplicar el rendimiento de la red a cortas distancias puede mejorar considerablemente la maniobrabilidad y la seguridad del enjambre.

Otra de las ventajas de una red ad hoc frente una red en modo infraestructura aplicada a un enjambre de drones es la distancia a la que cada tipo de red puede operar. En una red en modo infraestructura, el punto de acceso centralizado suele ser una estación de tierra no móvil, por lo que la distancia a la que puede operar está limitada a la potencia a la que puede emitir la señal. Para aumentar el rango de operación, sería necesario tomar medidas que incrementarían drásticamente el coste económico de la red, como utilizar antenas de larga distancia, incrementar el número de puntos de acceso, disponer de un punto de acceso móvil, etc. Por lo contrario, dado que no existe un punto de acceso centralizado en entornos de FANETs, el rango de operación no está limitado por el alcance de la señal de los adaptadores de red.

Estos factores demuestran que las redes ad hoc son una buena opción para las redes aplicadas a enjambres de drones. Sin embargo, el actual estándar del control por radiofrecuencia de los drones usa la banda de 2.4 GHz, e imposibilita la correcta comunicación entre los nodos de la red [11].

Para evitar las interferencias de la controladora, la solución más sencilla es utilizar la banda de 5GHz para las comunicaciones. Aumentar la frecuencia per-

mite incrementar el ancho de banda, pero al reducir la longitud de onda la señal es más susceptible a la atenuación.

En una red terrestre bastaría con aumentar la potencia de la antena; sin embargo, en FANETs, el incremento del consumo eléctrico al aumentar la potencia de la señal puede reducir significativamente la autonomía del vehículo, por lo que no sería la mejor opción.

Para averiguar como afecta la atenuación de la señal al rendimiento de la red, y determinar si es necesario ajustarla para cumplir los requisitos de aplicación, hemos realizado las siguientes pruebas experimentales: se han colocado dos antenas omnidireccionales de 5dBi de potencia, una fija y una móvil, a tres metros de altura, y con línea de visión directa. Tras mover la antena móvil cierta distancia, se inicia un intercambio de paquetes de 64 bytes usando el comando *ping* y se mide el rendimiento de la red. Una vez finalizada la medición, se aleja la antena móvil y se repite el proceso.

La línea continua roja de la Figura 1 muestra el máximo retardo sufrido en intercambio de paquetes a la distancia indicada, mientras que la línea continua azul muestra la media. Las líneas discontinuas muestran el ajuste por regresión lineal correspondiente.

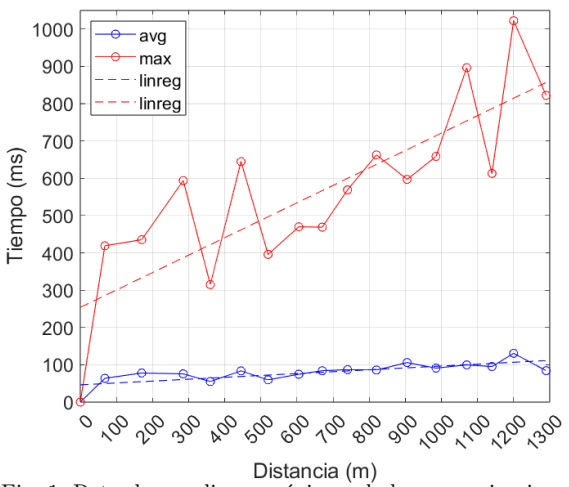


Fig. 1: Retardos medios y máximos de las comunicaciones en función de la distancia

En el mejor de los casos el máximo retardo siempre es mayor a 300 milisegundos, pudiendo llegar al segundo de duración. En función de la distancia y la velocidad del dron, un segundo podría ser crítico.

Actualmente, en España, la velocidad máxima a la que puede circular un dron de categoría máxima (C6) es de 50 metros por segundo. Si tenemos esta información en cuenta a la hora de diseñar los Algoritmos de navegación del enjambre de drones, y ya que el alcance de la red ad hoc lo permite, podemos mantener una distancia de seguridad que evite colisiones causadas por retrasos en la comunicación.

El tiempo medio de retardo en las comunicaciones no supera los 150 milisegundos. Consideramos que es una cifra que cumple los requisitos de responsividad para aplicaciones en drones no tripulados.

Como se puede ver en la Figura 2 la línea continua muestra como la desviación típica en el retardo de las comunicaciones aumenta en función de la distancia, llegando a sumar casi 150 milisegundos por encima de la media, mientras que la línea discontinua muestra el ajuste por regresión lineal de la misma.

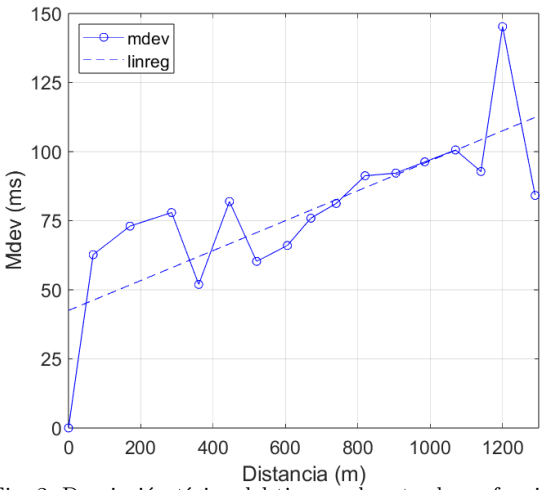


Fig. 2: Desviación típica del tiempo de retardo en función de la distancia

Es importante mantener una conexión estable que nos permita predecir cuanto tiempo va a costar completar la transmisión de una cantidad concreta de datos, por lo que reducir la dispersión de los retardos es un punto a tener en cuenta. No obstante consideramos que las cifras experimentales cumplen los requisitos de la mayoría de aplicaciones de FANETs.

Como muestra la Figura 3, a medida que aumenta la distancia incrementa la pérdida de paquetes, especialmente a partir de los 600 metros, llegando al 1.8 % de pérdida. La línea discontinua muestra el ajuste por regresión lineal. En función de la aplicación y el tamaño del paquete, el rendimiento actual de la red puede ser apropiado.

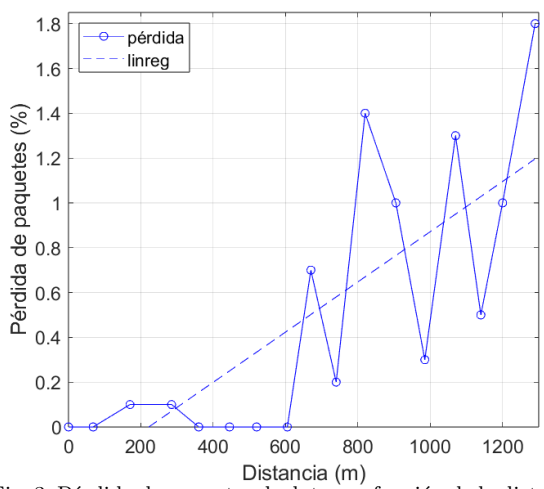


Fig. 3: Pérdida de paquetes de datos en función de la distancia.

En definitiva, si el uso de FANETs en la banda de los 5GHz es una opción adecuada va a depender de los requisitos de la aplicación. Como se ha visto en

las Figuras 2 y 3, la red experimental sufre una variación en el retardo de la señal (jitter) mayor a 30 milisegundos, y una pérdida de paquetes superior al 1 %, por lo que no sería apta para aplicaciones donde se requiera un control preciso del movimiento del dron o servicios VoIP. En FANETs donde la mayor parte de los mensajes de la red solo contienen información acerca de la ubicación, altura o velocidad los nodos, el uso de la banda de los 5GHz es suficiente para asegurar un correcto funcionamiento.

IV. Propuesta de un Algoritmo para la definición de BSSID únicos

Cuando se crea una red ad hoc, el dispositivo que la crea genera un identificador de servicio básico (BSSID) que todos los paquetes de la red utilizan para identificarse como miembros de la misma.

El nodo que crea la red genera el BSSID de forma aleatoria. Cuando un nodo configurado para conectarse a una red ad hoc detecta que existe otro nodo con el mismo identificador de set de servicio (SSID), se adhiere a ella utilizando el BSSID establecido por el primer nodo.

Si el dispositivo no se encuentra dentro del alcance de la red existente, creará una red con un nuevo BSSID, dividiendo la red e impidiendo la comunicación entre los nodos con distintos BSSID, obligando a iniciar los dispositivos de la red dentro del alcance de un nodo en previo funcionamiento.

Esta limitación complica el despliegue de redes de enjambre de drones, ya que obliga a todos los vehículos a despegar desde el mismo punto. Además, cualquier situación que requiera de un reinicio del sistema debe hacerse dentro del alcance de otro nodo en funcionamiento, como por ejemplo un recambio de las baterías.

Otros estudios en redes vehiculares [12] han optado por el uso del estándar 802.11p OCB (Outside the Context of a BSS) que descarta la fase de autenticación del protocolo 802.11 y permite una comunicación directa, evitando los problemas derivados de tener un BSSID, tal y como se ha explicado anteriormente. No obstante, en este caso, todos los nodos cercanos van a estar en una misma red, lo cual no siempre es lo deseable, ya que deberían estar segregados por tipo de servicio.

Para solucionar el problema, se propone un Algoritmo de generación de BSSIDs únicos que permite, por una parte, evitar el problema de los BSSIDs aleatorios y, por otra parte, segregar nodos según servicio. Esto se logra utilizando un SSID común para todos los dispositivos de la red y un prefijo reservado. Mediante una operación hash se puede conseguir un BSSID no aleatorio, único y reproducible por cualquier nodo que conozca el nombre y prefijo de la red.

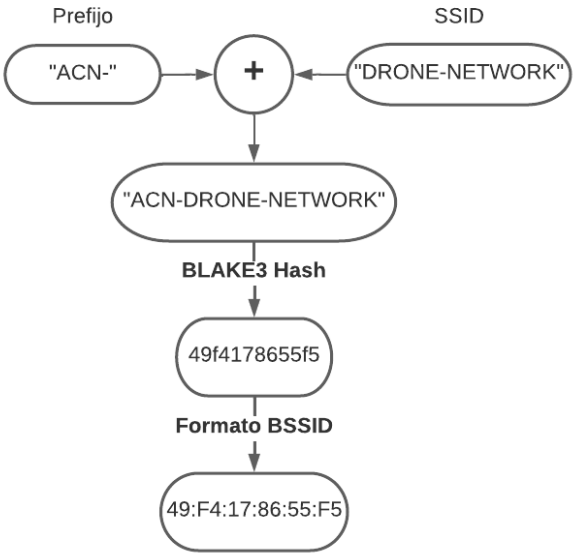


Fig. 4: Generación de BSSID único mediante BLAKE3

La Figura 4 muestra el proceso de generación de un BSSID único a partir de un prefijo de red reservado y un nombre de red arbitrario. Tras combinar ambos elementos, se aplica una función hash al resultado. La longitud de un BSSID es de 48 bits, por lo que es necesario que la salida de la función tenga dicho tamaño. Aunque es posible truncar la salida de una función hash, esto incrementa el riesgo de colisiones, por lo que se ha optado por utilizar la función de longitud variable (BLAKE3). Tras obtener el hash se dividen los 48 bits en bytes aplicando el formato de un BSSID, es decir, el formato de una dirección MAC como el que se muestra en la Figura 5.

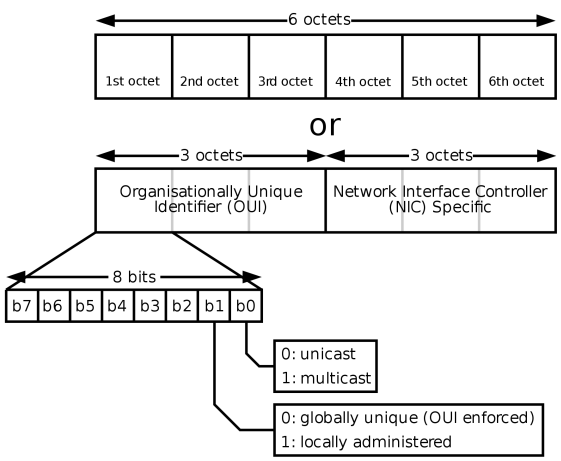


Fig. 5: Estructura de una dirección MAC de 48 bits

Por último, se asegura que el bit *b1* del primer octeto del BSSID valga 1, indicando que la dirección está administrada localmente, por lo que no es única a nivel global.

Aplicando el BSSID resultante a la configuración de red del dispositivo, podemos garantizar que será capaz de comunicarse con el resto de nodos de la red independientemente del lugar o momento en el que se haya iniciado.

V. Conclusiones y trabajo futuro

Este artículo estudia la viabilidad de las redes ad hoc en la banda de los 5GHz para las comunicaciones entre drones, con el objetivo de encontrar una solución a las interferencias presentes en la banda de los 2.4GHz. Se ha medido el rendimiento de la red en una serie de pruebas experimentales. Aunque el uso de la banda de los 5GHz introduce problemas como el incremento de la atenuación de la señal o la escasez de chipsets que soporten esta frecuencia junto con el modo ad hoc, se ha llegado a la conclusión de que es una opción que cumple los requisitos para el despliegue de redes de enjambre de drones.

A nivel técnico, el actual mecanismo de generación de BSSID (aleatorio) impide un uso generalizado de la tecnología IEEE 802.11 estándar para comunicación entre drones de manera general. Por esa razón, se ha propuesto un mecanismo alternativo en el cual se generan BSSIDs fijos para cada servicio que se pretenda implementar en la red de drones, lo cual permite lograr la segregación adecuada de dispositivos, y su comunicación en todo momento.

Como trabajo futuro, se propone la evaluación del Algoritmo de generación de BSSIDs únicos mostrado en este artículo, así como la repetición de las pruebas experimentales utilizando drones, lo que nos permitirá aumentar la distancia y la precisión de las mismas, gracias a la falta de obstáculos, la estabilización y el control de altitud de la controladora de vuelo.

Agradecimientos

El presente trabajo ha sido realizado en el marco del proyecto PID2021-122580NB-I00, financiado por MCIN/AEI/10.13039/501100011033 y "Fondo Europeo de Desarrollo Regional (FEDER)".

Referencias

[1] Logan Kugler, "Real-world applications for drones," *Commun. ACM*, vol. 62, no. 11, pp. 19–21, oct 2019.

[2] S Sharmila and T Shanthi, "A survey on wireless ad hoc network: Issues and implementation," in *2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS)*, 2016, pp. 1–6.

[3] Farhan Mohammed, Imad Jawhar, Nader Mohamed, and Ahmed Idries, "Towards trusted and efficient uav-based communication," in *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, 2016, pp. 388–393.

[4] Giovanni Geraci, Adrian Garcia-Rodriguez, Lorenzo Galati Giordano, David Lopez-Perez, and Emil Björnson, "Understanding uav cellular communications: From existing networks to massive mimo," *IEEE Access*, vol. 6, pp. 67853–67865, 11 2018.

[5] "Ieee standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, 2016.

[6] J. Gozalvez, M. Sepulcre, and R. Bauza, "Ieee 802.11p vehicle to infrastructure communications in urban environments," *IEEE Communications Magazine*, vol. 50, no. 5, pp. 176–183, 2012.

[7] Yong Zeng, Jiangbin Lyu, and Rui Zhang, "Cellular-connected uav: Potential, challenges, and promising technologies," *IEEE Wireless Communications*, vol. 26, no. 1, pp. 120–127, 2019.

[8] İlker Bekmezci, Ozgur Koray Sahingoz, and samil Temel, "Flying ad-hoc networks (fanets): A survey," *Ad Hoc Networks*, vol. 11, no. 3, pp. 1254–1270, 2013.

[9] Francisco Fabra, Carlos T. Calafate, Juan Carlos Cano, and Pietro Manzoni, "ArduSim: Accurate and real-time multicopter simulation," *Simulation Modelling Practice and Theory*, vol. 87, pp. 170–190, Sept. 2018.

[10] Shanna Li, "Comparative analysis of infrastructure and ad-hoc wireless networks," *ITM Web of Conferences*, vol. 25, pp. 01009, 2019.

[11] Francisco Fabra, Carlos T. Calafate, Juan-Carlos Cano, and Pietro Manzoni, "On the impact of inter-UAV communications interference in the 2.4 GHz band," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*. June 2017, IEEE.

[12] Carlo Augusto Grazia, "On the performance of IEEE 802.11p outside the context of a BSS networks," in *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. Sept. 2018, IEEE.

Generación de planes de vuelo para UAVs recolectores de datos en WSN

Adil Elidrissi[1,2], Rafael Casado[2], Abdelkrim Haqiq[1] y Luis Orozco-Barbosa[2]

Resumen— **El objetivo de este trabajo es el desarrollo de un algoritmo para optimizar la trayectoria de un vehículo aéreo no tripulado en un escenario de recogida de datos procedentes de una red inalámbrica de sensores desplegada en cierto área. En primer lugar, se define la ruta a seguir por el vehículo garantizando que pueda acceder a todos los sensores. Posteriormente, se propone un algoritmo eficiente de distribución de los sensores en clusters. Los resultados numéricos muestran que este algoritmo mejora los resultados del algoritmo k-means en cuanto al número de waypoints requeridos. A continuación, se optimiza la trayectoria original del UAV, mediante un algoritmo de traslación sucesiva de dichos waypoints. Los resultados numéricos muestran que el algoritmo propuesto reduce la longitud de la trayectoria recorrida, lo que repercute en el tiempo de vuelo necesario para la recogida de datos.**

Palabras clave— **Vehículos aéreos no tripulados, redes inalámbricas de sensores, adquisición de datos.**

I. Introducción

LOS vehículos aéreos no tripulados (UAV) son aeronaves que pueden funcionar de forma autónoma sin piloto humano. Con sus ventajas de bajo coste, diseño compacto, movilidad y facilidad para llevar cámaras y una amplia gama de sensores, los UAVs tienen cada vez más aplicaciones. La inspección aérea, la vigilancia meteorológica, la fotografía, la agricultura de precisión, la gestión del tráfico, la entrega de paquetes y las telecomunicaciones son sólo algunos de los muchos usos modernos de los UAVs [1]. Los vehículos aéreos no tripulados se han promovido recientemente como una tecnología prometedora para una variedad de aplicaciones inalámbricas, incluyendo la expansión de la cobertura y la gestión de escenarios de desastre/emergencia [2].

Hoy en día, existen muchos tipos diferentes de UAVs siendo utilizados en innumerables aplicaciones. Aunque no existe una única forma de clasificar a los UAVs, se pueden agrupar en diferentes categorías basándose en criterios como sus funciones, peso/carga útil, tamaño, resistencia, configuración de las alas, métodos de control, autonomía de crucero, altitud de vuelo y velocidad [1]. En lo que respecta a altitud, se clasifican en plataformas de gran altitud (HAPs) y baja altitud (LAPs). Las HAPs suelen ser cuasiestacionarias a alturas superiores a 17 km [3]. Por su parte, las LAPs son ágiles, pueden desplazarse con rapidez y volar a altitudes comprendidas entre unos pocos metros y unos pocos kilómetros [4]. Con respecto a la configuración de sus alas, los UAVs se clasifican en UAV de ala fija y UAV de ala rotatoria. Los primeros no pueden flotar en un solo lugar y necesitan una pista o lanzadera para despegar y aterrizar. Por el contrario, los UAV de ala rotatoria pueden permanecer estacionarios sobre un área determinada. El lector puede encontrar una clasificación detallada de los distintos tipos de UAV en [5].

Las redes inalámbricas de sensores (WSN) son redes de sensores interconectados mediante el uso de comunicaciones inalámbricas, que comúnmente se despliegan en un área determinada para recoger y transmitir datos, por ejemplo sobre las condiciones ambientales (niveles de contaminación, temperatura, presión, etc.) [6]. Hoy en día, las WSNs se emplean en multitud de ámbitos de aplicación, como control de riego en campos agrícolas, vigilancia del campo de batalla, transporte inteligente, prevención de desastres naturales, o control de la calidad del agua [2]. La conectividad de los sensores de baja potencia puede lograrse de forma inalámbrica mediante el uso de protocolos y soluciones como ZigBee, Z-wave, y IEEE 802.15.4. Sin embargo, las WSNs se enfrentan a limitaciones considerables, como la baja capacidad de procesamiento, las limitaciones de suministro de energía y la capacidad de almacenamiento de los nodos [7].

En general, hay tres casos típicos de uso de las comunicaciones inalámbricas asistidas por UAVs, que son: cobertura ubicua asistida por UAVs, retransmisión asistida por UAVs [8], difusión de información y recopilación de datos asistida por UAVs [9].

Es posible abordar los problemas antes mencionados en las WSN mediante la integración de vehículos aéreos no tripulados con dispositivos inalámbricos embarcados. De este modo, en lugar de mantener una conexión continua con los centros de datos, los UAV permiten a los sensores establecer conexiones de forma temporal e intermitente. Así, al recopilar y difundir datos periódicamente, los UAV contribuyen a mejorar el rendimiento de la red y a prolongar su vida útil [7]. Por ejemplo, en [10], se ha desarrollado una WSN de bajo coste, que utiliza un UAV para la recogida de datos en la agricultura y la detección de cambios en nodos terrestres muy dispersos que no son directamente conectables. En [11], se proporciona un estudio exhaustivo sobre los sistemas inteligentes de colaboración UAV-WSN.

Sin embargo, todavía hay cuestiones que deben solventarse para poner en práctica las WSN con la asistencia de UAV. Por ejemplo, determinar la velocidad idónea del UAV para obtener el mejor rendimiento

[1]Hassan First University of Settat, Faculty of Sciences and Techniques, Computer, Networks, Mobility and Modeling Laboratory: IR2M, 26000 - Settat, Morocco, e-mail: abdelkrim.haqiq@uhp.ac.ma.

[2]Instituto de Investigación en Informática de Albacete, Universidad de Castilla-La Mancha, 02071 Albacete, Spain, e-mail: adil.elidrissi@alu.uclm.es {rafael.casado,luis.orozco}@uclm.es.

de la red, determinar la mejor ubicación de los sensores desplegados, o optimizar la gestión de conflictos cuando muchos sensores intentan enviar datos al UAV al mismo tiempo. Para aumentar la eficacia de la red, es necesario diseñar un método de recogida de datos que permita su procesamiento autónomo. Encontrar la ubicación adecuada de los sensores en un entorno a gran escala para garantizar que los datos de los sensores se reciben y transfieren al ritmo deseado y de forma fiable supone otro reto con las WSN basadas en UAV. Es necesario construir una planificación rápida de rutas para los UAV que permita recolectar los datos de la red en el menor tiempo posible, y consumiendo la menor cantidad de energía posible [7].

II. Trabajo relacionado

En los últimos años, ha habido cada vez más interés en el uso de vehículos aéreos no tripulados para recopilar datos de redes de sensores inalámbricas. Yawei et al. [12] utilizaron UAVs para recopilar datos de una WSN. Sin embargo, esta investigación se limitó a redes aisladas y a un único tipo de entorno. Además, no se ha estudiado el despliegue óptimo de UAV. El objetivo de Safwan et al. [13] fue reducir la duración media necesaria para la recopilación de datos; minimizando, al mismo tiempo, los costes dentro del sistema en el que participan los UAVs. El objetivo de Binol et al. [14] fue reducir el tiempo y distancia total de viaje de los vehículos aéreos no tripulados, mediante la identificación de la ruta más corta para que los vehículos aéreos no tripulados recopilaran datos de varios sensores de carretera. Por otra parte, Saxena et al. [15] se centraron en abordar el problema de selección de vuelo como una ruta Hamiltoniana que requiere visitas a todas las cabezas de clúster en una WSN dada. En [16], los autores ajustaron la velocidad de vuelo del UAV para aumentar la cantidad de datos recogidos. La investigación presentada en [17] optimizó el programa de recogida de datos, la trayectoria del UAV, el tiempo dedicado a la recogida de datos y la capacidad de transmisión de los nodos de tierra para maximizar la cantidad mínima de datos recogidos. Li et al. [18] investigaron la recopilación total o parcial de datos de dispositivos IoT utilizando el UAV. Esto se logró mediante la determinación de un recorrido cerrado con el fin de optimizar la cantidad de datos adquiridos. Además, Wang et al. [19] mejoraron la ruta de vuelo del UAV, así como la potencia de transmisión de los sensores durante el proceso de carga de datos, teniendo en cuenta las limitaciones impuestas por la potencia y la energía de los sensores.

Por otra parte, gran cantidad de esfuerzos de investigación se ha centrado en WSN con sensores de alto rango de comunicación con una conexión continua con las cabezas de clúster; lo cual no es apropiado, ya que el enlace continuo sensores-cabezas de clúster requiere más energía, reduciendo la vida útil de la red.

En este trabajo proponemos un escenario de recogida periódica de datos utilizando un UAV en una WSN con bajo rango de comunicación entre sensores. Adoptamos un nuevo enfoque de clustering con el fin de minimizar la cantidad de waypoints que requiere la trayectoria del UAV para servir a toda la red.

III. Modelo del sistema

Consideramos una red WSN con una distribución arbitraria de N sensores. Los sensores captan datos que deben transmitir a la red. En algunos casos, la distribución de los sensores impide que puedan interactuar, de modo que los datos adquiridos en cada sensor no pueden enviarse a la estación base a través de relés multisalto. En otras ocasiones existen limitaciones (como edificios) que impiden la comunicación. Además, tienen restricciones energéticas, ya que la transmisión de datos consume un porcentaje considerable de su energía.

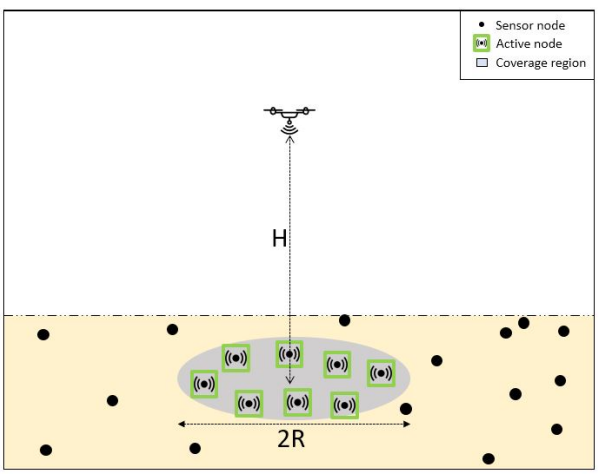


Fig. 1: Modelo del sistema.

Para prolongar la vida útil de cada nodo sensor, se utiliza un UAV para recoger los datos datos captados. Sea $S = \{1, ..., N\}$ es el conjunto de sensores desplegados en ubicaciones conocidas $\{w_i\}_{i \in S}$, donde $w_i = (x_i, y_i, 0)$ son las coordenadas del sensor i^{th}. Sea $q_u = (x_{ut}, y_{ut}, z_u)$ la ubicación 3D del UAV en tiempo t. El UAV se desplaza sobre el área de interés a una altitud fija $z_u = H$, como se muestra en la fig. 1, donde H es la altitud que corresponde al radio R de máxima cobertur del UAV. El UAV debe recoger todos los datos de los sensores y volver a la estación base para descargarlos. La duración de un recorrido del UAV se define por la longitud del recorrido y la cantidad de datos registrados en los sensores cubiertos por el UAV en cada lugar en el que planea durante el recorrido. Se asume que cada sensor i envía un paquete de volumen v_i. Se asume que la comunicación es ortogonal, es decir, que a cada sensor se le asigna una banda de frecuencias o canal distinto para la comunicación inalámbrica que no se solapa con el de ningún otro sensor. La relación señal/ruido (SNR) recibida puede darse como:

$$\gamma_i = \frac{P_i d_i^{-\alpha}}{\alpha^2}, \forall i = 1, ..., N, \quad (1)$$

donde P_i es la potencia de transmisión del sensor

i, d_i es la distancia entre el UAV y el sensor i, α es el exponente de pérdida de trayectoria, y γ^2 es la potencia de ruido. Si $\gamma_i \geq \gamma_0$, el UAV prestará el servicio de recogida de datos, donde γ_0 es la SNR mínima. La pérdida de trayectoria es:

$$P_L = 20 \times log(d_i) + 20 \times log(\frac{4\pi f}{c}) + \eta \quad (2)$$

donde f es la frecuencia de la portadora, c es la velocidad de la luz, y η es el exponente de pérdida de trayectoria excesiva. El servicio se presta sólo si $\eta \geq \eta_0$.

IV. Formulación del problema

En un escenario de recogida de datos por UAV en una red de sensores inalámbrica, se asume que el UAV se aproxima a cada sensor para obtener sus datos almacenados, lo que no resulta práctico en el caso de una WSN de gran tamaño, debido a la limitada energía del UAV. En este artículo, proponemos un algoritmo de agrupación que organiza los sensores en clusters. Primero minimizamos el número de clusters para que cada sensor pertenezca como máximo a un cluster. A continuación, refinamos la ubicación de vuelo estacionario de la UAV, de modo que cada sensor de todos los clústeres se sirve, garantizando al mismo tiempo una trayectoria óptima recta. El problema puede formularse como sigue:

$$\begin{aligned} \min_{x} \quad & \sum_{i=0}^{N} \frac{\|q_i - q_{i+1}\|}{|W|+1} \\ \text{sujeto a} \quad & \gamma_i \geq \gamma_0, \forall i = 1, ..., N \end{aligned} \quad (3)$$

donde $w_0 = w_{N+1}$ es la ubicación de la estación base, $\|.\|$ y $|.|$ son la norma euclidiana y los operadores de cardinalidad, respectivamente. La función de utilidad minimiza la distancia media de recogida de datos. Además, la restricción (3b) asegura que todos los sensores transmiten sus datos de almacenamiento con éxito durante el viaje.

El problema formulado es difícil de resolver directamente, por lo tanto, proponemos un enfoque de tres pasos. En primer lugar, dado el conjunto de sensores W, encontramos las ubicaciones de vuelo estacionario $\{c_i\}_{j=1,...,|C|}$, donde $|C|$ es el número de ubicaciones de vuelo estacionario. En segundo lugar, generamos el camino más corto. El último paso consiste en refinar las posiciones de vuelo estacionario para mantener la ruta más corta del UAV.

V. Solución propuesta

A. Localización de waypoints

Para minimizar los costes, nuestro objetivo es reducir al mínimo el número de ubicaciones de vuelo estacionario. En primer lugar, agrupamos los sensores. La agrupación de K-means es un método popular para agrupar sensores porque es relativamente sencillo y eficaz. Consiste en asignar sensores a un número predeterminado de grupos en función de su proximidad a los centros de los grupos. Los centros de los clusters se actualizan iterativamente hasta que se alcanza la convergencia. Cada sensor pertenece a un cluster de radio R, que corresponde a la altitud óptima que maximiza la zona de cobertura. por ahora, se supone que el UAV planea sobre el centro de cada cluster. El problema puede formularse como sigue:

$$\text{mín} \quad |C| \quad (4a)$$

$$\text{sujeto a} \quad \sum_{i=1}^{|C|} y_{ij} = 1, j = 1, ..., N. \quad (4b)$$

donde $C = \{C_1, C_2, ..., C_M\}$ es el conjunto de clusters, $y_{ij} = \begin{cases} 1 & \text{si } \|c_i - w_j\| \leq R \\ 0 & \text{en otro caso.} \end{cases}$

y c_i, w_j son las coordenadas del centro del cluster C_i y del sensor j, respectivamente. La norma euclidiana $\|c_i - w_j\|$ es la distancia entre el centro del cluster C_i y el sensor j. La restricción $4b$ asegura que cada sensor pertenece sólo a un cluster. Este problema es NP-complejo, sin embargo, un algoritmo adaptativo K-means resuelve 4 iterativamente reuniendo sensores cercanos en clusters de radio R. Nuestro algoritmo propuesto, presentado en el Algoritmo 1, supera al algoritmo de agrupación k-means adaptativo, obteniendo los mejores resultados. La idea media de nuestro algoritmo propuesto es particionar todos los sensores en clusters de radio R que se corresponde con la máxima cobertura del UAV, implicando el mínimo número de clusters. El algoritmo consiste en elegir los dos sensores más cercanos $\{s_a, s_b\}$ y aquellos sensores a una distancia inferior a $2 \times R$ de la media de s_a y s_b y eliminar de la consideración el resto de sensores, ya que no pueden ser atendidos con s_a y s_b al mismo tiempo. El centro del conglomerado (la ubicación de vuelo estacionario) se coloca de modo que s_a, s_b y tantos sensores sin servicio pertenezcan a un conglomerado, dando mayor prioridad a s_a y s_b.

Usamos el ejemplo de la figura 2 para ilustrar los pasos principales de nuestro algoritmo de clustering propuesto. La figura muestra la construcción de los cuatro primeros clusters, damos prioridad a los dos sensores más cercanos. Para localizar el primer cluster, buscamos los dos sensores más cercanos a una distancia menor o igual a $2 \times R$, colocamos el centro inicial del primer cluster en el punto medio del segmento $[s_a, s_b]$ (denotado por los dos puntos rojos en la figura)(paso 2 en el Algoritmo 1). Luego clasificamos otros sensores de acuerdo a sus distancias al centro inicial del cluster, eliminamos de consideración los sensores a una distancia de más de $2R$ (paso 3 en Algoritmo 1). Entonces refinamos la ubicación del cluster para cubrir s_a y s_b y tantos sensores en C_1, dada la prioridad a los sensores más cercanos al centro inicial utilizando el procedimiento **clustering** en el Algoritmo 2.

B. Planificación de ruta

Después de obtener el número mínimo de grupos y las ubicaciones de vuelo estacionario, el problema

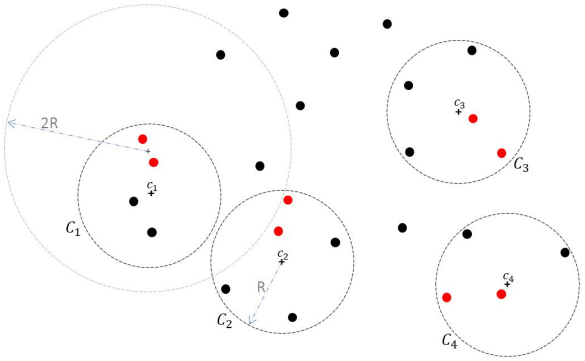


Fig. 2: Ilustración del algoritmo de clustering.

Algorithm 1 Clustering Wds.

Input: set W, with known locations $\{w_j\}_{j\in W}$
Output: C, $\{c_i\}_{i\in C}$, $\{N_{C_i}\}_{i\in C}$
Initialization: $C=\emptyset$, $i=1$, $W_u \leftarrow W$
while $W_u \neq \emptyset$ **do**
find the two nearest Wds (p_1, p_2),
c_i=mean(p_1, p_2)
$W_{F_i} = \{\text{Wd } j / \|c_i - w_j\| > 2R\}$,
$C_i \leftarrow W_u \backslash W_{F_i}$
$[c_i, N_{C_i}] =$ **Clustering**(p_1, p_2, C_i)
$W_u \leftarrow W_u \backslash N_{C_i}$, $C = C \cup \{C_i\}$, $i \leftarrow i+1$
end while

Algorithm 2 Clusterización de nodos.

Function $[c_i, N_{C_i}] =$ **Clustering**(p_1, p_2, C_i).
Initialization: $C_t \leftarrow C_i$.
while $C_t \neq \emptyset$ **do**
Find vertices $a_1, a_2, a_3 \in C_t \cup \{p_1, p_2\}$,
that define the biggest triangle.
$[R_i, c_i] =$ **circlecenter**(a_1, a_2, a_3)
if $R_i > R$
$C_t \leftarrow C_t \backslash \{\text{a vertex } \{a_i\}_{i=1,2,3} \notin \{p_1, p_2\}\}$
else
$N_{C_i} \leftarrow C_t$, $C_t \leftarrow \emptyset$
end
end while

3 se limita a encontrar la mejor trayectoria para recoger datos, el UAV parte de la estación de atraque a las ubicaciones de vuelo estacionario, y vuelve a la estación de atraque después de visitar todas las ubicaciones de vuelo estacionario. Obsérvese que el tiempo de vuelo estacionario es suficiente para que los sensores transmitan los paquetes almacenados. El problema puede formularse como sigue:

$$\text{mín} \quad \sum_{i=0}^{|C|} \sum_{j=0, j\neq i}^{|C|} d(C_i, C_j) x_{ij} \tag{5a}$$

$$\text{sujeto a} \quad \sum_{i=0, i\neq j}^{|C|} x_{ij} = 1, \forall j = 0, 1, ..., |C|. \tag{5b}$$

$$\sum_{j=0, j\neq i}^{|C|} x_{ij} = 1, \forall j = 0, 1, ..., |C|. \tag{5c}$$

$$u_i - u_j + |C|(x_{ij} - 1) \leq -1, . \tag{5d}$$

$$x_{ij} \in \{0, 1\}. \tag{5e}$$

Donde $d(C_i, C_j) = \|c_i - c_j\|$ es la distancia entre los HL i y j, $x_{ij} = 1$ si el UAV viaja desde el HL i al HL j, en caso contrario, $x_{ij} = 0$, u_i y u_j son enteros no negativos. Las restricciones (5b) y (5c) garantizan que el UAV debe llegar y salir de la ubicación de vuelo estacionario una sola vez, respectivamente, y la restricción (5d) es la restricción de eliminación de subtour. Este problema se conoce como problema del viajante de comercio (TSP). El TSP busca el camino más corto posible que un vendedor puede tomar para visitar un conjunto dado de ciudades exactamente una vez y volver al punto de partida. El reto consiste en encontrar una solución óptima entre un número exponencialmente grande de caminos posibles, lo que convierte al TSP en un problema NP-difícil. No obstante, en la literatura se han presentado numerosos algoritmos para abordarlo [20]. De forma similar, el UAV y los HL podrían representarse como el vendedor y las ciudades, respectivamente.
Para resolver el problema 5, empleamos programación entera binaria (BIP). Este problema consiste en encontrar el camino cerrado más corto a través de un conjunto de Hls. El BIP es un tipo de problema de optimización en el que las variables de decisión están restringidas a ser binarias (es decir, o 0 o 1). El objetivo de un BIP es encontrar el conjunto de valores de variables binarias que minimicen la función objetivo sujeta al conjunto de restricciones.

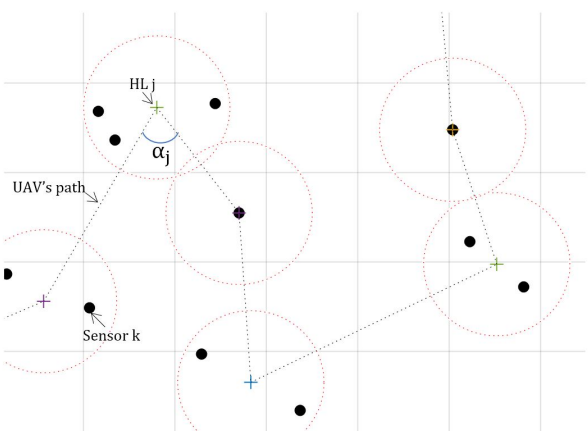


Fig. 3: Ilustración del movimiento del dron.

Algoritmo 3 Ubicación definitiva de waypoints.

Input:
initial locations $\{c_i\}_{i\in C}$,
$\{N_{C_i}\}_{i\in C}$,
simulation step δ
Output: $\{c_{f_i}\}_{i\in C}$
Initialization: $c_{f_i} \leftarrow c_i$, $i = 1...|C|$
while
$\alpha_i \leq 180$ and $\|c_i - w_j\| \leq R$,
$\forall j \in N_{C_i}, i = 1, ..., |C|$
do
$d = \overrightarrow{c_i c_{i-1}} + \overrightarrow{c_i c_{i+1}}$
$u = d/|d|$
$c_t \leftarrow c_{f_i} + \delta u$
if $\|c_t - w_j\| \leq R, \forall j \in N_{C_i}$
$c_{f_i} \leftarrow c_t$
end
end while

C. Algoritmo de ajuste de waypoints

En esta sección, proponemos un algoritmo eficiente para obtener resultados óptimos para resolver el problema 1, basado en el refinamiento de los HLs. La idea principal es la traslación de los HLs en una dirección determinada, garantizando que todos los sensores pertenecen a sus clusters asociados, con el fin de acortar la trayectoria óptima obtenida resolviendo el problema 5.
La trayectoria del UAV puede mejorarse más maximizando los ángulos de desviación $\{\alpha_i\}_{i=1,...,|C|}$, ya que los ángulos agudos en la trayectoria de un UAV requieren más energía. El problema puede formularse como sigue:

$$\text{máx} \quad \alpha \tag{6a}$$

$$\text{sujeto a} \quad \alpha_i \leq 180, \forall i = 1, ..., |C| \tag{6b}$$

$$\|c_i - w_j\| \leq R, \ \forall j \in N_{C_i}, \forall i = 1, ..., |C|, \tag{6c}$$

donde $\alpha = (\alpha_1, ..., \alpha_{|C|})$, and N_{C_i} denota el conjunto de sensores asociados al cluster i.
El problema 6 también es NP-difícil. Proponemos un algoritmo iterativo para resolver el problema 6, basado en refinar los HL hacia un vector direccional d que depende del HL anterior y siguiente a visitar, y luego repetir el proceso (Algoritmo 3) hasta la convergencia.

VI. Evaluación y resultados

Hemos procedido a evaluar nuestra propuesta mediante simulación, empleando MATLAB R2023a. Asumimos un área de 10×10 km^2, con $N = 200$ sensores. Fijamos el radio de cobertura del UAV en $R = 800$ m, que corresponde a la altitud máxima del UAV H.

En la figura 4, comparamos el rendimiento del algoritmo de agrupación k-means adaptativo y el algoritmo de agrupación propuesto, en términos del número de ubicaciones de vuelo estacionario frente

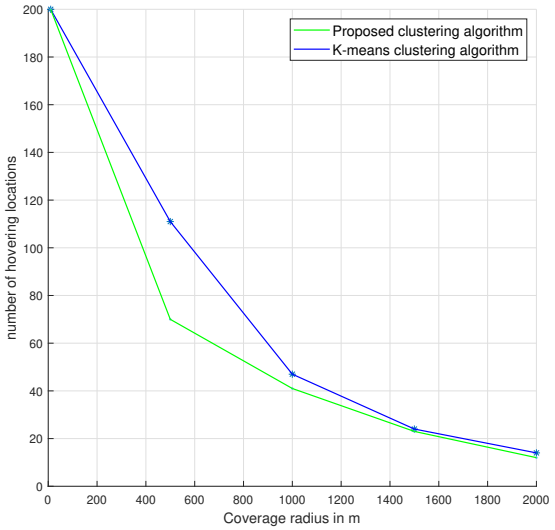


Fig. 4: Número de waypoints para cubrir 200 sensores, en función del radio de cobertura.

al radio de cobertura del UAV. A medida que R aumenta, el número de posiciones de vuelo estacionario disminuye para los dos enfoques. Esto se debe a que los UAV con un radio de cobertura mayor necesitarán un menor número de posiciones de vuelo estacionario para cubrir el área de interés, en comparación con aquellos con un radio de cobertura menor. Además, nuestro algoritmo de agrupación propuesto consigue el mínimo número de posiciones de vuelo estacionario y, por tanto, un menor consumo de energía.

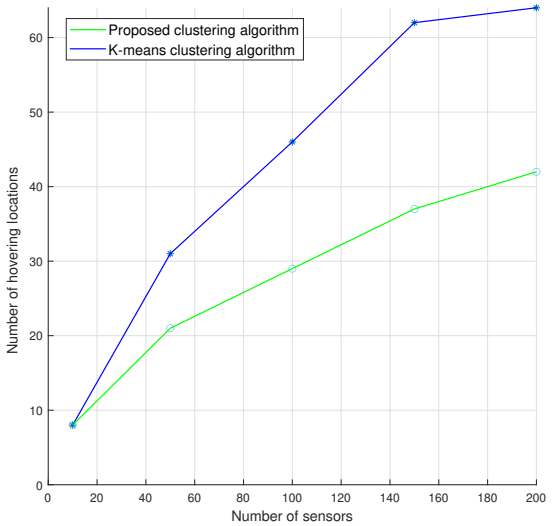


Fig. 5: Número de waypoints en función del número de sensores, dado un radio de cobertura de 800 metros.

Para evaluar el rendimiento del algoritmo 1, fijamos el radio de cobertura del UAV en $R = 800$ m, ejecutamos el algoritmo para varios números de sensores $N \in [10, 200]$. Los resultados se muestran en la figura 5. A medida que N crece, se necesita un mayor número de posiciones de vuelo estacionario para ambos enfoques. Además, el algoritmo de agrupación

propuesto supera al algoritmo k-means en cuanto al número necesario de posiciones de rastreo, especialmente en el caso de un gran número de sensores.

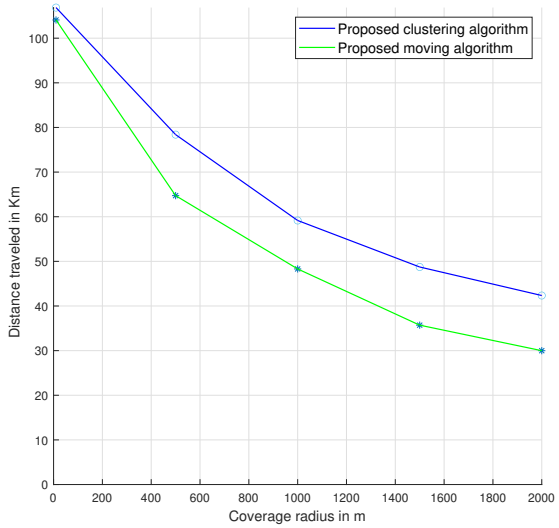


Fig. 6: Distancia recorrida en función del radio de cobertura.

Para investigar el rendimiento del algoritmo 3, comparamos el rendimiento del algoritmo de agrupación propuesto y el algoritmo de movimiento propuesto para determinar las ubicaciones de vuelo estacionario del UAV, en términos de la distancia recorrida frente al radio de cobertura con $N = 200$ sensores. Como se muestra en la figura 6, a medida que R crece, la distancia recorrida por el UAV disminuye para ambos enfoques. Por otra parte, el algoritmo de movimiento propuesto alcanza un rendimiento óptimo en términos de la distancia recorrida durante un viaje para servir a una WSN, en comparación con el algoritmo de agrupación.

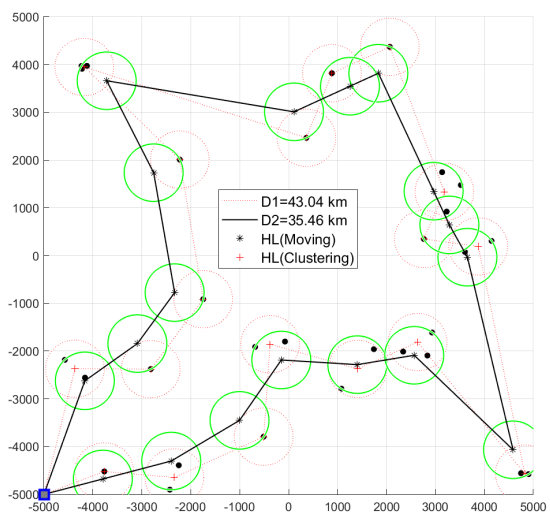


Fig. 7: Trayectoria del UAV para cubrir 20 sensores, con una cobertura de 600 metros.

En la figura 7, comparamos la distancia recorrida por el UAV antes y después de refinar los HL (Algoritmo 3). Suponiendo que $D1$ y $D2$ son las longitudes de trayectoria del UAV para los dos enfoques respectivamente. Como se muestra en la figura, el enfoque móvil logra una longitud de trayectoria reducida, garantizando al mismo tiempo la recogida de datos de 20 sensores dentro de un radio de cobertura de $R = 600$ m. La distancia de ganancia recorrida se determina por $G = (D1 - D2) \times \frac{1}{D1}.$, que es en nuestro escenario $G = 15,9\,\%$. Esto implica que el perfeccionamiento de los HL con las restricciones mencionadas consigue los mejores resultados en términos de distancia recorrida y, por tanto, de duración de la batería, ya que el UAV es un dispositivo de energía limitada. Por lo tanto, es más beneficioso que adoptar HL sin refinar.

VII. Conclusiones

En este artículo se propone un marco de trabajo para las WSN asistidas por UAV en un escenario de recogida de datos, teniendo en cuenta las limitaciones presupuestarias y energéticas. En concreto, se propone una nueva solución de agrupación de sensores para optimizar el número de posiciones de vuelo estacionario, garantizando la recogida de datos de todos los sensores. El algoritmo propuesto se compara favorablemente con el algoritmo K-means en cuanto al número mínimo de ubicaciones de rastreo requeridas. A continuación, explotamos soluciones con BIP para planificar la trayectoria del UAV asegurando el camino más corto. Además, proponemos una solución de HLs móviles para reducir más la longitud de la trayectoria y minimizar así el tiempo de la misión de recogida de datos.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Ciencia e Innovación, la Junta de Comunidades de Castilla-La Mancha, y la Unión Europea (Fondo Europeo de Desarrollo Regional), a través de los proyectos PID2021-123627OB-C52 y SBPLY/19/180501/000159, y por la Universidad de Castilla-La Mancha a través de la ayuda 2023-GRIN-34056.

Referencias

[1] Y. Zeng, Q. Wu, and R. Zhang, "Accessing from the sky: A tutorial on uav communications for 5g and beyond," *Proceedings of the IEEE*, vol. 107, no. 12, pp. 2327–2375, Dec 2019.

[2] R. I. Bor-Yaliniz, A. El-Keyi, and H. Yanikomeroglu, "Efficient 3-d placement of an aerial base station in next generation cellular networks," in *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1–5.

[3] Y. Zeng, R. Zhang, and T. J. Lim, "Wireless communications with unmanned aerial vehicles: opportunities and challenges," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 36–42, May 2016.

[4] A. Al-Hourani, S. Kandeepan, and A. Jamalipour, "Modeling air-to-ground path loss for low altitude platforms in urban environments," in *2014 IEEE Global Communications Conference*, 2014, pp. 2898–2904.

[5] K. P. Valavanis and G. J. Vachtsevanos, *Handbook of Unmanned Aerial Vehicles*, Springer, Amsterdam, The Netherlands, 2015.

[6] S. L. Ullo and G. R. Sinha, "Advances in smart environment monitoring systems using iot and sensors," *Sensors*, vol. 20, no. 11, pp. 3113, May 2020.

[7] M. E. Mkiramweni, C. Yang, J. Li, and W. Zhang, "A survey of game theory in unmanned aerial vehicles communications," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3386–3416, Fourthquarter 2019.
[8] W. Chen, S. Zhao, R. Zhang, Y. Chen, and L. Yang, "Uav-assisted data collection with nonorthogonal multiple access," *IEEE Internet of Things Journal*, vol. 8, no. 1, pp. 501–511, Jan 2021.
[9] W. Chen, S. Zhao, and Q. Shi, "Improve stability in uav relay networks by jointly optimizing communication, trajectory and power," in *2018 IEEE International Conference on Communication Systems (ICCS)*, 2018, pp. 180–185.
[10] J. Polo, G. Hornero, C. Duijneveld, A. García, and O. Casas, "Design of a low-cost wireless sensor network with uav mobile node for agricultural applications," *Computers and Electronics in Agriculture*, vol. 119, pp. 19–32, 2015.
[11] D. Popescu, F. Stoican, G. Stamatescu, O. Chenaru, and L. Ichim, "A survey of collaborative uav–wsn systems for efficient monitoring," *Sensors*, vol. 19, no. 21, pp. 4690, 2019.
[12] Y. Pang, Y. Zhang, Y. Gu, M. Pan, Z. Han, and P. Li, "Efficient data collection for wireless rechargeable sensor clusters in harsh terrains using uavs," in *2014 IEEE Global Communications Conference*, 2014, pp. 234–239.
[13] S. Alfattani, W. Jaafar, H. Yanikomeroglu, and A. Yongacoglu, "Multi-uav data collection framework for wireless sensor networks," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.
[14] Harun Binol, Ersin Bulut, Kemal Akkaya, and Ismail Guvenc, "Time optimal multi-uav path planning for gathering its data from roadside units," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. IEEE, 2018, pp. 1–5.
[15] Kshitij Saxena, Nidhi Gupta, Jitender Gupta, et al., "Trajectory optimization for the uav assisted data collection in wireless sensor networks," *Wireless Networks*, vol. 28, no. 4, pp. 1785–1796, 2022.
[16] Xingjian Li, Jie Tan, Anfeng Liu, Pradeep Vijayakumar, Naveen Kumar, and Moutaz Alazab, "A novel uav-enabled data collection scheme for intelligent transportation system through uav speed control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2100–2110, 2021.
[17] Kun Wang et al., "Uav-based and energy-constrained data collection system with trajectory, time, and collection scheduling optimization," in *2021 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, 2021, pp. 893–898.
[18] Ying Li et al., "Data collection maximization in iot-sensor networks via an energy-constrained uav," *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 159–174, 2023.
[19] Yuting Wang, Min Chen, Chunxiao Pan, Kezhi Wang, and Yi Pan, "Joint optimization of uav trajectory and sensor uploading powers for uav-assisted data collection in wireless sensor networks," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 11214–11226, 2022.
[20] Rajesh Matai, Surya Prakash Singh, and Murari Lal Mittal, "Traveling salesman problem: an overview of applications, formulations, and solution approaches," *Traveling Salesman Problem, Theory and Applications*, vol. 1, 2010.
[21] D. Gettinger and A. H. Michel, "Drone sightings and close encounters: An analysis," Tech. Rep., Center Study Drone, Bard College, Annandale-on-Hudson, NY, USA, 2015.

Arquitecturas heterogéneas

Security improvement in social networking by scalable crypto hardware architecture in FPGA

Jaime David Ríos Arrañaga[1], Juan José Raygoza Panduro[1], Edwin Christian Becerra Alvarez[1], Sandra Eloísa Balderas-Mata[1], José Luis González Vidal[2]

Resumen— **Because social networks have become high impact as a means of communication, having a security system that protects user information is imperative. Here, we present a proposal that allows the services of the Telegram social network to be strengthened by implementing a public key cryptography in real time. The hardware architecture design of the Paillier scheme is presented, implemented in reconfigurable hardware on FPGA *Artix 7 XC7A100TCSG324-1* and the *Spartan 6 FPGA SP601 evaluation kit* for different key lengths, using the Hardware Description Language VHDL and the design and simulate Xilinx tools: *Vivado Design Suite v2021.2, ISE 14.6 and ISim 14.6*. The experimental results show the encryption process in real time of the data sent through *Telegram* social network.**

Palabras clave— **Field Programmable Gate Array, Paillier Scheme, Public Key, Security Networking**

I. Introduction

UNTIL recently, people considered social networks to be a space that fostered freedom of expression and the free discussion of different points of view. Moreover, people believed that social network gave them the freedom to organize themselves, that they served as the new public square, and that they appeared to be the new rulers of online discourse [1]. However, a series of rules imposed by companies that offer social network services and instant messaging are regulating people's freedom of expression. What can be said and shared has now been limited, and in the search for new spaces for expression, people have migrated from one network to another, seeking more freedom to express themselves. In this scenario, the social network of Telegram [2], which offers different services such as instant messaging, sending files, payment platforms, games, education, and mass communication, among others [3], has seen a growing demand of new users who perceive this application a space with fewer restrictions. In this work, a proposal is presented that allows strengthening the services of the Telegram social network through the implementation of a public-key cryptography system in real time.

Whitfield Diffie and Martin Hellman defined Public-Key Cryptography in 1976 [4], as a cryptographic system that calculates encryption and decryption using different keys (public key K_E and private key K_D) in such a way that calculating K_D from K_E is computationally challenging and impracticable. Therefore, the cipher key K_E can be publicly shared or revealed, without compromising the decryption key K_D. Public-Key cryptography, also known as asymmetric cryptography or asymmetric key due to its two different keys used for encryption and decryption, is a set of tree mathematical algorithms of one-way and trap-door functions: The key generation, the encryption algorithm, and the decryption algorithm. Because of its mathematical complexity, public-key is limited to encrypting only small blocks of data, for example, the privet key of symmetric cryptography (privet key or one-key cryptography). Typically, public-key is used to solve problems of key distribution [4], confidentiality, authentication, and non-repudiation in communication systems and digital signatures. Also, it is used in Public Key Infrastructures (PKI) [5], [6], and Asymmetric-Symmetric Hybrid cryptosystems [7], [8].

Currently, the most known public-key cryptographic systems are The RSA cryptosystem, proposes by Ron L. Rivest, Adi Shamir and Leonard Adleman in 1978 [9]; ElGamal by Tahir Elgamal in 1985 [10]; An analogue of the Diffie-Hellmann key exchange using Elliptic Curve by V. Miller in 1985 [11]; The Elliptic Curve Cryptosystems by Near Koblitz in 1985 [12]; and the Public-Key Cryptosystems Based on Composite Degree Residuosity Classes by Pascal Paillier in 1999 [13].

The Paillier Cryptosystem is a Homomorphic public-key probabilistic encryption scheme. It is based on composite degree residues classes, and the problem of factoring the product of two prime numbers in the generation of the keys [13].

This homomorphic property allows specific types of calculations to be carried out on ciphertext to generate an encrypted result, which then matches the operation results performed on plaintext [14]. These properties are used in the design of voting protocols [15], in threshold cryptosystems [16], watermarking, secret sharing schemes [13] and secure computation on the cloud [17].

All cryptographic algorithms can be implemented in software and hardware; however, in some cases, hardware is the best option considering security and efficiency, since it is not easily modified by an attacker or computer viruses. Also, this kind of implementation is faster in general, offering at the same time more intrinsic security [18].

Field Programmable Gate Array (FPGA) is an integrated circuit that the final user can be reconfigure by using some Hardware Description Language

[1]Department of Electro-photonic Engineering, University of Guadalajara, e-mail: jaime.rios@alumnos.udg.mx, {juan.rpanduro, edwin.becerra}@academicos.udg.mx.

[2]Department of Computer Science, Autonomous University of the State of Hidalgo

(HDL) like VHDL or Verilog. Their reconfigurability feature allows the FPGA to behave like a new digital circuit, so almost any kind of digital circuits can be implemented using FPGA [19]. Cryptography is a field of constant update of the algorithms used, or even of their substitution, and it is always seeking greater security and efficiency in the communication and storage of data. Consequently, FPGA's reconfigurability is a great way to develop and test cryptosystems on hardware [20], [21], [22], [23], [24], [25], [26].

This work describes a scalable architecture design of the Paillier public-key scheme on FPGA and its implementation. The mentioned architecture allows expanding the system without the need for a modification; this is possible because of the reconfigurability of the FPGA.

The implementations were tested using the FPGA *Artix 7 XC7A100T* and the *Spartan 6 FPGA SP601 evaluation kit*, the design and simulate tools *ISE 14.6* and *ISim 14.6* from *Xilinx*, and the *Telegram* social network for the experimental tests.

II. Paillier scheme

The Paillier scheme is based on properties of Carmichael function $\lambda(n)$, equal to the least common multiple (lcm) of the $p-1$ and $q-1$, as shown in Equation 1, where p and q are prime numbers in $Z^*_{n^2}$ and its security on the intractability of computing discrete logarithms in Z_{n^2} without the Carmichael number [13], [27].

$$\lambda(n) = lcm(p-1, q-1) \tag{1}$$

The encryption can be performed by anyone who knows the public-key K_E, while decryption can only be done with the knowledge of the private-key K_D. Because it is probabilistic, it is impossible for an adversary to tell whether two cipher-texts are encryptions of the same plaintext or not [17].

The encryption, decryption and key generation algorithms used in the Paillier scheme are calculated as follow [13].

A. Key Generation

- Choose two large prime numbers p and q randomly and independently of each other.
- Set $n = (p) \cdot (q)$ and its Carmichael's function defined in Equation 1.
- Select random integer g such that $g \in Z_{n^2}$ and g non-zero multiple of n. This can be done efficiently by checking whether $gcd(L_n(g^{\lambda(n)} \bmod n^2),\ n) = 1$, where $L_n(u)$ is the Lagrange function defined as $L_n(u) = (u-1)/n$.
- Make $\mu = 1/L_n(g^{\lambda(n)} \bmod n^2)$.
- Then, the public key K_E is the pair (n, g) and the private key K_D the pair $(\lambda(n), \mu)$.

B. Encryption

- Consider the plaintext $m < n$.
- Select a random number $r < n$.
- The ciphertext C is calculated by Equation 2.

$$C = g^m \cdot r^n \ mod\ n^2 \tag{2}$$

C. Decryption

- Consider the ciphertext $C < n^2$.
- The message is retrieve by Equation 3.

$$m = L_n(C^{\lambda(n)} \ mod\ n^2) \cdot \mu \ mod\ n \tag{3}$$

III. Mathematical algorithms

Scalability, interoperability, and security are one of the most important requirements of cryptographic systems [18]. Using the concept of Alexandre Tenca and Çetin KoÇ, an arithmetic module is considerably scalable if "the unit can be reused or replicated in order to generate long-precision results independently of the data path precision for which the unit was originally designed" [28].

When implementing a cryptosystem into a communication system, it is critical to take those requirements into account. Limiting the total of the logical occupation and even trying to reduce it can also be important. Consequently, the system can incorporate as a part, or a module, of a full data processing system, even if the processing time might be affected.

In this section, we present the mathematical algorithms that were used for the encryption and decryption. These algorithms were selected, considering the scalability and low use of resources.

A. Modular Multiplication

Modular multiplication, shown in Equation 4, is an essential arithmetic operation, commonly used in public-key schemes, either as a direct operation or as part of modular exponentiation.

$$C = A \cdot B \ mod\ M \tag{4}$$

Montgomery multiplication is one of the most suitable methods for the calculation of modular multiplication because it is fast and efficient in hardware implementations [29], The main advantage of the Montgomery multiplication is the substitution of divisions and modular reductions by additions, subtraction, and shift operations [30], [31]. This method is commonly used to solve the modular multiplications involved in the modular multiplication exponentiation [31].

To calculate the modular multiplication, the operands A and B are transferred to the so-called Montgomery domain, using Equation 5 where $r = 2^n \ mod\ M$; n is the number of bits of the binary representation of M, and x' is the Montgomery representation of x.

$$x' = x \cdot r \ mod\ M \tag{5}$$

After that, it continues to compute the multiplication within the Montgomery domain and finally changes the result to the real domain using Equation 6, as shown in the algorithm in figure 1 [30], [31], [32].

$$x = x' \cdot r^{-1} \ mod\ M \tag{6}$$

```
let be:
    A' = (A)(r) mod M
    B' = (B)(r) mod M
we will get:
    C = (A)(B) mod M
doing:
    t' = (A')(B')
    q' = (t' mod r) ( M') mod r
    C' = (t' + (q)(M) ) / r
    IF (C > M)
       C' = C' - M
    END IF
    C = (C') (r^(-1)) mod M
    RETURN C
```

Fig. 1: Montgomery Modular Multiplication.

```
let be:
    A, B and M with A, B < M
we will get
    C = (A)(B)(r^(-1)) mod M
doing:
    C = 0
    FOR (i = 0 to i = n-1)
        C = C + (a[i]) (B)
        IF (C[0] = 0)
            C = C/2
        ELSE
                C = (C + M) / 2
        END IF
    END FOR
    RETURN C
```

Fig. 2: Iterative Montgomery Multiplication MMM(A, B, M).

There is another way to develop multiplication, using the iterative version of the Montgomery method and the binary expression of the operands present in Equation 7, where n is the number of the binary digits of E and $b_i \in \{1, 0\}$.

$$E = (b_{n-1}b_{n-2}\dots b_2b_1b_0)_2 \tag{7}$$

The iterative Montgomery modular multiplication algorithm is shown in Figure 2 [33].

The resulting value is the Montgomery value C'. In order to produce the desire value C, we need to calculate an extra modular multiplication by the constant $r = 2^n \ mod \ M$ [33]; however, if the same modular multiplication algorithm is used again, then we need to calculate the Montgomery modular multiplication by r^2, meaning that C would be equal to $MMM(C', R2, M)$.

B. Modular Exponentiation

The Modular exponentiation, Equation 8, is the main operation not only of the Paillier Scheme, but also of many systems, public-key schemes such as the RSA encryption [9], ElGamal [10], Diffie-Hellman key exchange [4] and data scrambling [34].

$$P = B^E \ mod \ M \tag{8}$$

Due to modular exponentiation is calculated by a series of modular multiplications, the Montgomery algorithm is usually used in hardware implementation, to avoid the trial division [35]. Nevertheless, to perform this operation by the straightforward method requires a high cost in computational time.

The R-L algorithm calculates the modular exponentiation scanning the bits of the binary exponent, one to one. Squaring happens at each iteration. Multiplication is then performed only if the scanned bit is equal to $1'$, just as algorithm in Figure 3 shows [23], [36].

```
Inputs:
    B,E,M
we will get
    P = B^E mod m
doing:
    P[0] = 1
        S[0] = b
        FOR (i = 0 to i= n-1)
            S[i+1] = (S[i]) (S[1]) mod
                M
            IF (E[i] = 1)
                    P[i+1] = (P[i]) (S[
                        i]) mod M
            ELSE
                    P[i+1] = P[i$]
            END IF
    END FOR
    RETURN P
\end{algorithmic}
```

Fig. 3: R-L Binary Algorithm.

C. Pseudo-random Number Generation

A Random Number Generator (RNG) is a device or algorithm capable of producing sequences of random numbers, with good statistical properties, such as low-value correlation and non-repeatability [37]. This kind of algorithms are of great importance in modern cryptography, where they are used in the encryption process or in the calculation of encryption keys.

Generally, almost any Pseudorandom number algorithm could be used, but for cryptographic applications, it is important to generate Pseudorandom numbers which will be unpredictable to the adversary even at the exposure of partial information. It means that a Cryptographic Secure Pseudo Random Number Generator (CSPRNG) is needed, an algorithm that can ensure a certain degree of security and one that can pass certain tests like the next-bit test and statistical test such as the NIST (National Institute of Standards and Technology) statistical test suit.

The Pseudo Random Bit Generation (PRBG) proposed by Divyanjali in [38], was used in the architecture of the cryptosystem, and as reported in the article [38], it shows good results in the statistical tests specified by the NIST Special Publication 800-22. Thus all indicate that the PRBG is acceptable for cryptographic purposes.

IV. ARCHITECTURE

The design of the cryptosystem architecture is presented in two separated modules: the encryption module, called Encryptor; and the decryption module, called Decryptor. For a reduced use of logical resources, we used the pre-calculated values of the Keys, the public keys (n, g) and the private keys $(\lambda(n), \mu)$; and a constant value for the modular multiplication, the Montgomery constants r for n and n^2, which we call r_n and r_{n^2} respectively.

subsectionEncryptor Module The proposed architecture of the Encryptor module is shown in the diagram of Figure 4. It is based on the expanded repre-

sentation of Equation 2, presented below in Equation 9, and it has the input signals: *Plaintext*, where the text entries to be encrypted; *Enable*, activation signal that initiates encryption; *CLK*, the clock signal and *Reset*. The Output signals are: *Ciphertext*, the result of encryption; and *SigEND*, indicates the end of the encryption process.

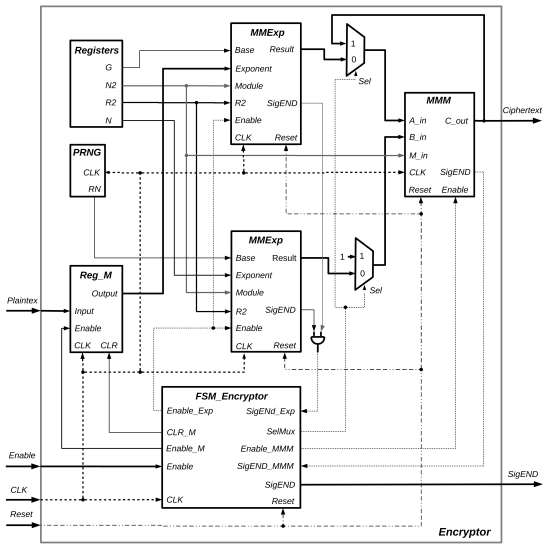


Fig. 4: Architecture of the Encryptor Module.

$$C = (g^m \ mod \ n^2) \cdot (r^n \ mod \ n^2) \ mod \ n^2 \qquad (9)$$

Internally the encryption system is composed of two binary modular exponentiation modules, here called *MMExp*, for the calculation of the modular exponent $g^m \ mod \ n^2$ and $r^n \ mod \ n^2$; a Montgomery modular multiplication *MMM* for the multiplication of the product of both exponentiations and for the calculation of its transformation Montgomery-to-Real; four registers to store the constants values: g, n, n^2, rm^2; a pseudo-random number generator *PRNG*; a finite state machine *FSM_Encryptor*, for process control; a *PIPO* (parallel input - parallel output) register called *Reg_M* to store the plaintext; and two multiplexers for data routing.

The *FSM_Encryptor* is a finite state machine whose output values $\Lambda\{$*enable_Exp, enable_MMM, enable_M, clr_M, SelMux, SigEND*$\}$ determine both, its current state and the current inputs $\Sigma\{$*Enable, SigEND_Exp, SigEND_MMM*$\}$. Its function is to control the other modules to carry out the encryption process. *The FSM_Encryptor* has eight states $\delta\{$*S0, S1, S2, S3, S4, S5, S6, S7*$\}$, the diagram of Figure 5 shows the transition from one state to other.

Each state has a specified task. The initial state is $S0$. It is responsible for restarting the Encryptor module, the mathematical modules, and the data path, as well as for cleaning registers. $S0$ is the default state when the system is turned on or restarted, and it has a duration of one clock cycle. $S1$ stores the input value of the plaintext and prepares the way of the data for the calculation process. While the Enable signal is not activated the system remains in this state. The encryption starts in the $S2$ state. It controls the calculation of the modular exponentiation, while the process of Modular Exponentiation is not ended the system remains in this state; it means while *SigEND_Exp* $= 0$. The state $S3$ is responsible for storing the results of the previous calculation. $S4$ controls the modular multiplication, while the process is not ended the system remains in this state; it means while *SigEND_MMM* $= 0$. The $S5$ state is like $S3$. It stores the results of the previous calculation, but it is also responsible for routing the results back to the multiplier for the next operation. $S6$ controls the Montgomery-to-real transformation process of the calculated value. $S7$ presents the result to the output *Ciphertext* and activates *SigEND* to indicates the end of the encryption process.

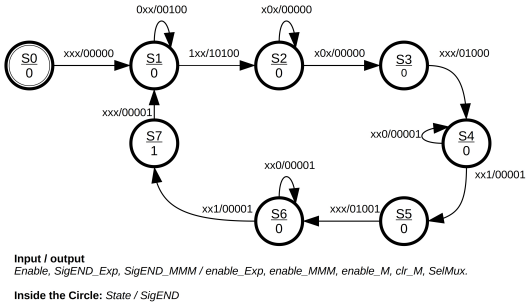


Fig. 5: Finite State Machine of the Encryption architecture.

A. Decryptor module

The proposed architecture of the *Decryptor Module* is shown in the diagram of Figure 6. It is based on the decryption Equation 3, and has the input signals: *Ciphertext*, where the encrypted text entries to be decrypted; *Enable*, activation signal that initiates decryption; *CLK*, the clock signal, and *Reset*. Its Output signals are: *Plaintext*, the result of decryption; and *SigEND*, indicates the end of the process. The decryption system is composed of a binary exponentiation module *MMExp*, for the calculation of the modular exponent $C^{\lambda(n)} \ mod \ n^2$; a *Ln(x) module* for the LaGrange function $L_n(x)$; a Montgomery modular multiplication called *MMM* for the product of $L_n(x)$ function result and the constant μ, and the Montgomery-Real transformation of the result; six registers to store the keys λ, μ, n and the square of n; the constant r^2 of Montgomery for n and n^2, here called r_n^2 and $r_{n^2}^2$ respectively and defined in Equations 10 and 11; a PIPO register called *Reg_M*, to store the decryption result; three multiplexers; and a finite state machine *FSM_Decryptor*, for process control and data flow. The ciphertext is saved internally in a register in the *MMExp module*.

$$r_n^2 = r^2 \ mod \ n \qquad (10)$$

$$r_{n^2}^2 = r^2 \ mod \ n^2 \qquad (11)$$

The *FSM_Decryptor* is a finite state machine whose output values $\Lambda\{$*enable_Exp, enable_MMM, enable_L, enable_M, clr_M, SelMux_M, SelMux_AB,*

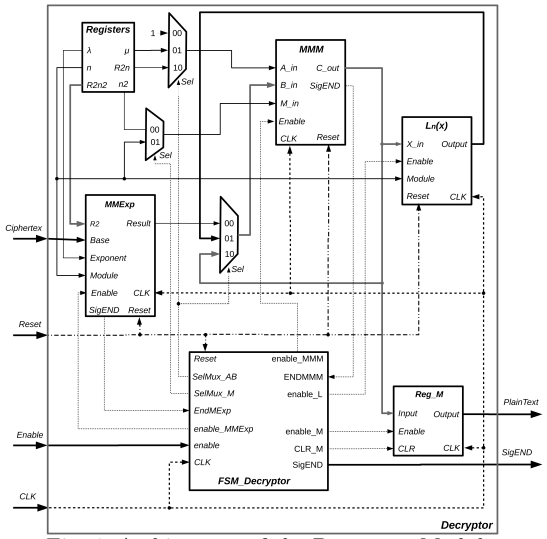


Fig. 6: Architecture of the Decryptor Module.

SigEND} determine both, its current state and the current inputs Σ{*enable, CLK, Reset, SigEND_Exp, SigEND_MMM, SigEND_Ln*}. Its function is to control the other modules to carry out the decryption process. The *FSM_Decryptor* has eight states δ{*S0, S1, S2, S3, S4, S5, S6, S7*}, the diagram of Figure 7 shows the transition from one state to other.

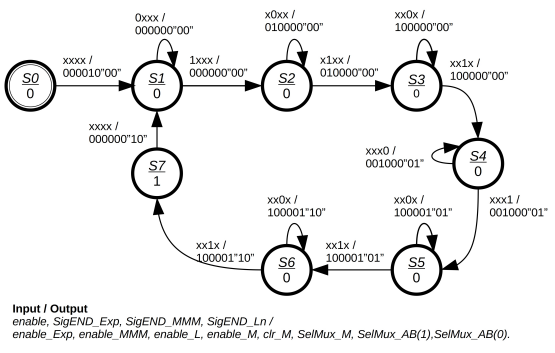


Fig. 7: Finite State Machine diagram of the Decryptor Architecture.

The initial state $S0$ is responsible for restarting the Decryptor module, the mathematical module, and the data path, as well as for cleaning registers. $S0$ is the default state when the system is turned on or restarted, and it has a duration of one clock cycle. $S1$ stores the input value of the *Ciphertext* and prepares the way of the data for the calculation process. While the *Enable* signal is not activated the system remains in this state. The decryption starts in the $S2$ state it controls the calculation of the modular exponentiation $C^{\lambda(n)}\ mod\ n^2$, while the process of Modular Exponentiation is not ended, the system remains in this state meaning while *SigEND_Exp = 0*. The state $S3$ is responsible for the transformation Montgomery-to-real of the previously calculated value. This transformation is necessary because the *Ln(x) module* only works with real domain values, and the result of the modular multiplication is a Montgomery value. $S4$ puts the *Ln(x) module* into operation. During the state $S5$ the multiplication between the result of the module *Ln(x)* and μ is developed. The $S6$ state is responsible for the transformation between the Montgomery domains and the real one because the multiplication in the *MMM module* results in a value in the Montgomery domain. The $S7$ presents the result to the output *Plaintext* and activates *SigEND* to indicates the end of the decryption process.

V. Implementation results

In this section the behavioral simulations resulting of the Encryptor and Decryptor modules using the FPGA Artix 7 XC7A100TCSG324-1 are presented. This implementation was described Using VHDL and Vivado Design Suite v2021.2. The percentage of logical resources occupation and timing processing when both modules are implemented and tested using the pair values {keywords, Plaintext} of (16-bits, 32 bits), (128-bits, 256 bits), (256-bits, 512 bits) (512-bits, 1024 bits).

A. Encryption

To test the Encryption module, the results obtained in the simulation were compared with the results calculated using the Equation 2 and a fixed value of the random value r. For example, in Table I we present the handmade calculation of one encryption using as a plaintext the value 85 and the *public-keys* $g = 2$ and $n = 10403$. The encryption is shown with the same values in Figure 8, and we obtained that number in both cases the result was $44'044,677$. The encryption module has a calculation time of 11,890 μs (microseconds) or $1,189$ clock cycles with a clock frequency of 100 MHz.

Tabla I: Encryption handmade calculation.

Plaintext $(m) = 85$
Public keys: $g = 2, n = 10403$
Montgomery base $rm^2 = 31127067$
$n^2 = 108222409$
random r (established for the test) $= 2$
$C = (g^m \bmod n^2)(r^n \bmod n^2) \bmod n^2$
$C = (285 \bmod 10403^2)(210403 \bmod 10403^2) \bmod 10403^2$
$C = (57040519)(39398) \bmod 108222409$
$C = 44044677$

In Figure 9 a simulation is shown using random values of r for the same values of plaintext and publickey.

Tabla II: Results of encryptor module to different key-lengths in Artix 7 XC7A100T

Key-Length (bits)		**Logical resources occupation**		
n	n^2	*Slicer Register*	*Slice LUTs*	$LUT - FF$
16	32	1,667 (1 %)	1,119 (1 %)	1,341 (6 %)
128	256	7,011 (5 %)	5,921 (9 %)	4,409 (22 %)
256	512	13,720 (10 %)	12,835 (20 %)	7,198 (37 %)
512	1,024	27,062 (21 %)	2,181 (42 %)	14,137 (73 %)

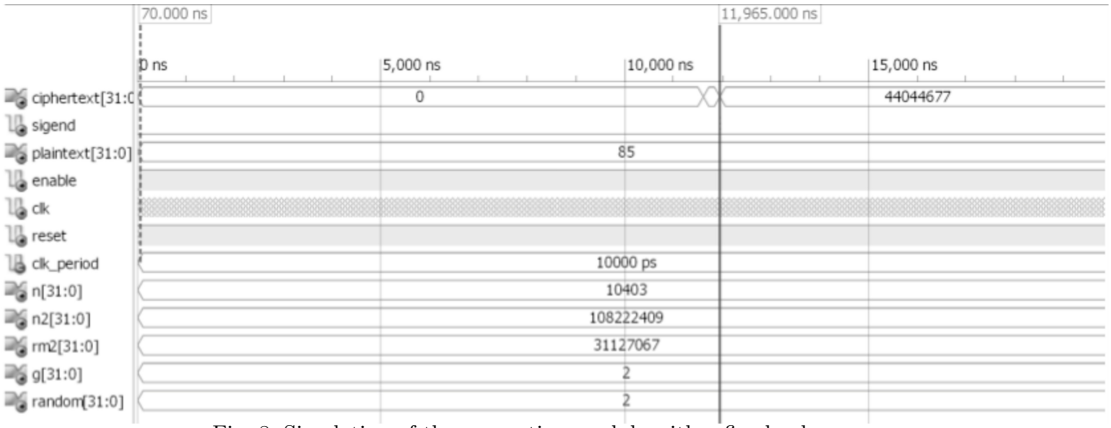


Fig. 8: Simulation of the encryption module with a fixed value or r.

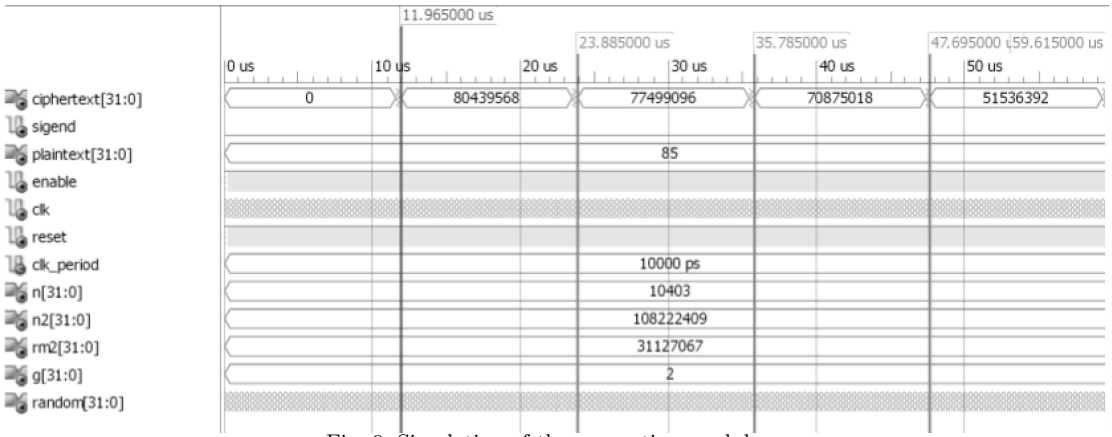


Fig. 9: Simulation of the encryption module.

Tabla III: Time report of the encryptor module in Artix 7 XC7A100TCSG324-1

Key-Length (bits)		**Min. period**	**Max. clock**
n	n^2	(ns)	**freq. (MHz)**
16	32	5.216	191.710
128	256	8.635	115.814
256	512	14.522	68.859
512	1,024	26.299	38.025

B. Decryption

To test the decryption, we used the resulting ciphertexts of the encryption. We, then, compared the results obtained in the simulation with the results calculated used in Equation 3. In Figure 10, a decryption example is shown using the ciphertext equal to $44'044,677$ and obtained to encrypt the plaintext 85; a result obtained from the *Encryptor module* and shown as an example of its functioning in Figure 8. Also, the handmade calculation of the decryption is shown in Table IV.

As a result, we obtained that the handmade calculation in Table IV and the resulting plaintext from the simulation in Figure 10 that are matching with the original plaintext encrypted 85.

The decryption module has a calculation time of 13.230 μs or 1,323 clock cycles with a clock frequency of 100 *MHz*. The results in logical resources utilization for the Decryption module reported by Vivado using an FPGA Artix 7 XC7A100TCSG324-1 for a key-length of 32, 256, 512 and 1,024 bits and a clock frequency of 100 MHz, is shown in Table V.

Tabla IV: Decryption handmade calculation

Ciphertext(C) = $44'044,677$
Public key(n) = 10,403
Private key:$\mu = 9,000$, $\lambda(n) = 10,200$)
Montgomery base $rm^2_2 = 31'127,067$
Montgomery base $rm^2_n = 1,291$
$n^2 = 108'222,409$
$m = L_n(C^{\lambda(n)} \bmod n^2)(\mu) \bmod n$
$U = C^{\lambda(n)} \bmod n^2$
$U = 44'044,677^{10,200} \bmod 108'222,409$
$U = 63'791,197$
$m = L_n(63'791,197)(9,000) \bmod 10,403$
$m = (6,132)(9,000) \bmod 10,403$
$m = 85$

Tabla V: Results of decryptor module to different key-lengths in Artix 7 XC7A100T

Key-Length (bits)		**Logical resources occupation**		
n	n^2	*Slicer Register*	*Slice LUTs*	$LUT - FF$
16	32	137 (1 %)	181 (1 %)	211 (2 %)
128	256	4,558 (3 %)	4,431 (6 %)	2,954 (14 %)
256	512	8,910 (7 %)	12,835 (20 %)	5,642 (29 %)
512	1,024	17,661 (13 %)	18,776 (29 %)	26,950 (35 %)

The minimum period and maximum clock frequency for the decryptor module are presented in the Table VI.

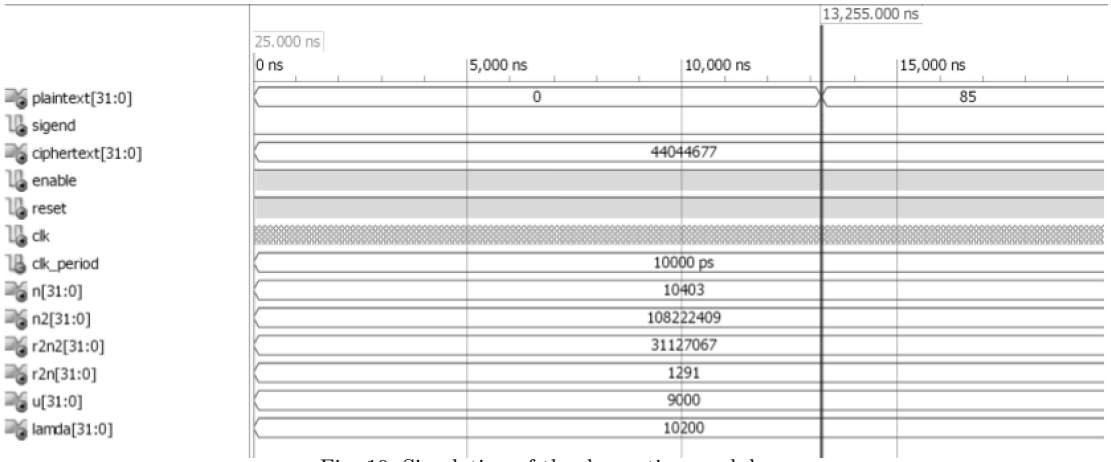


Fig. 10: Simulation of the decryption module.

Tabla VI: Time report of the decryptor module in Artix 7 XC7A100TCSG324-1

Key-Length (bits) n	n^2	**Min. period** (ns)	**Max. clock freq. (MHz)**
16	32	3.483	287.142
128	256	8.635	115.814
256	512	14.522	68.859
512	1,024	26.299	38.025

VI. Message encryption implementation and experimental results

The flow diagram in Figure 11, shows the connection of the Paillier encryption system with the Telegram social network. The first step is to configure and connect to the Telegram network. Then a message is expected that will be sent to the cryptographic system and returned with its equivalent encrypted by the encryption hardware. This process is repeated supporting multiple users through a FIFO system.

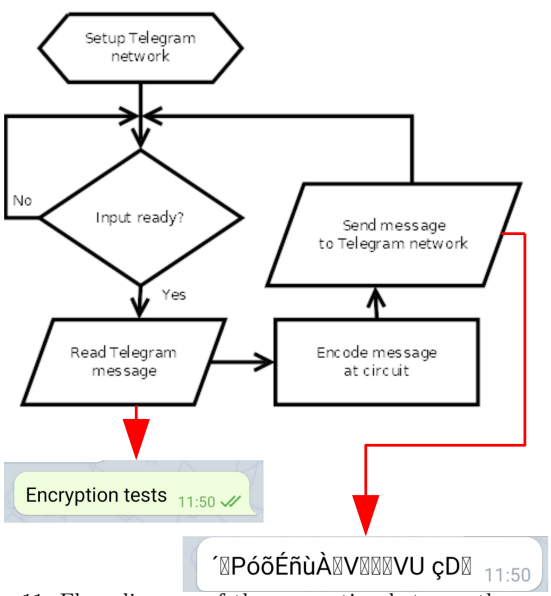


Fig. 11: Flow diagram of the connection between the encryption system and the Telegram social network.

Figure 12 shows a prototype of the complete system implementation integrated by the computer executing connections with the Telegram network and the FPGA *Spartan 6 SP601 evaluation kit* that performs the encryption through Paillier Cryptosystem configured on it and the smartphone that runs the *Telegram App* to test the message encryption system.

The Tables VII and VIII show the resource utilization percentage of the FPGA *Spartan 6 SP601 evaluation kit*. Meanwhile, the experimental test used a key length n = 128. Table VII shows that the use of a higher key length, such as 256 bits, exceeds the necessary resources of the encryptor module and is therefore not allowed in this FPGA. Implementation and simulation were made using ISE 14.6 and ISim 14.6 because the uncompatibility of the board in Vivado.

Tabla VII: Results of encryptor module to different key-lengths in SPARTAN 6 SP601 Evaluation kit.

Key-Length (bits) n	n^2	**Logical resources occupation** *Slicer Register*	*Slice LUTs*	$LUT - FF$
128	256	7,031 (38 %)	5,928 (65 %)	4,409 (51 %)
256	512	13,740 (75 %)	12,840 (140 %)	7,198 (84 %)

Tabla VIII: Results of decryptor module to different key-lengths in SPARTAN 6 SP601 Evaluation kit.

Key-Length (bits) n	n^2	**Logical resources occupation** *Slicer Register*	*Slice LUTs*	$LUT - FF$
128	256	4,561 (25 %)	4,431 (48 %)	2,964 (25 %)
256	512	8,913 (48 %)	8,527 (93 %)	5,652 (47 %)

VII. Conclusions

This work describes the implementation of a scalable crypto hardware system in FPGA as a proposal to improve the security of social networks. Experimental tests showed that the system maintained good performance using the proposed architecture, which allowed it to encrypt information in real-time. Encryption and decryption are described for implementation, individually, presenting the architecture

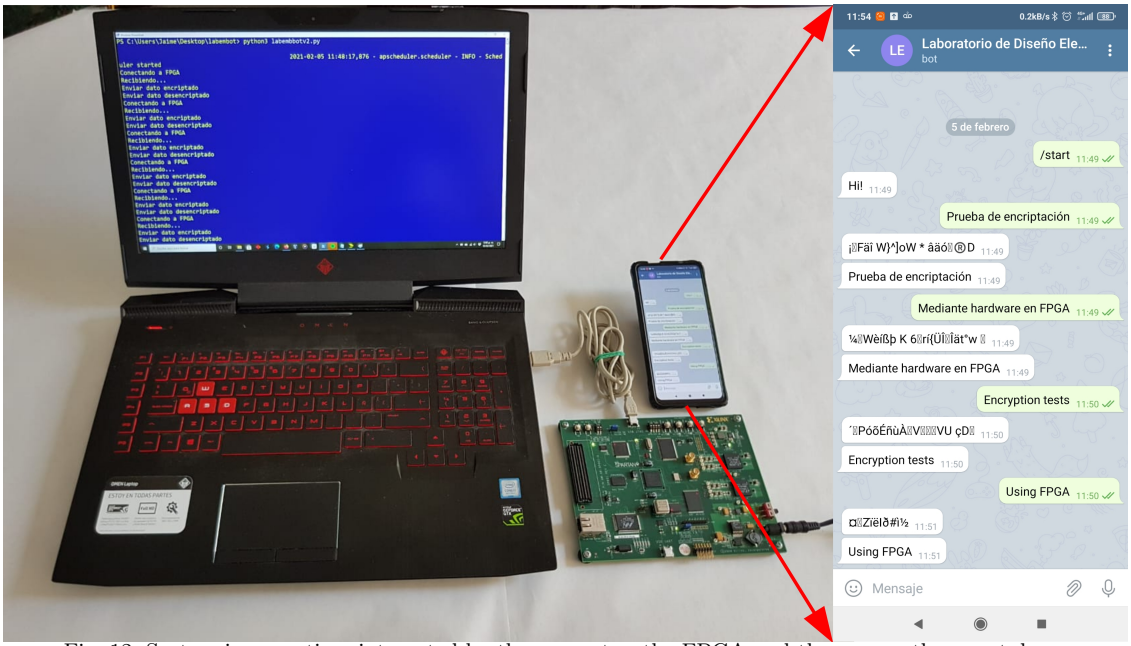


Fig. 12: System in operation, integrated by the computer, the FPGA and the app on the smartphone

diagrams and the corresponding state machines for data and operations management. The use of mathematical algorithms for a hardware implementation was also analyzed.

To verify flexibility and operation, two implementations of the same architecture were made using two different FPGAs: FPGA *Artix 7 XC7A100TCSG324-1* and the *Spartan 6 FPGA SP601 evaluation kit*. The hardware block system was proposed used a parallel design methodology with an approximate resource utilization of 22 % by *prototype A* for the encryptor and 14 % for the decryptor in FPGA *Artix 7*. For *prototype B*, we have an approximate resource utilization percentage of 51 % for the encryptor and 25 % for the decryptor in *Spartan 6 FPGA SP601 evaluation kit* both for a key $n = 128$ bits. As one can see from tables VII and VIII, using a longer key length n is not feasible for *spartan 6*; however, *Artix 7* would be a better option if a longer key length is required.

Acknowledgment

The authors would like to thank the National Council of Science and Technology of Mexico (CONACYT) and the University of Guadalajara for their support in the development of this work.

Referencias

[1] Kate Klonick, "The new governors: The people, rules, and processes governing online speech," *Harvard Law Review*, vol. 131, no. 6, pp. 1599–1670, Jan. 2018.

[2] Soporte Telegram, "Telegram faq," url:https://web.telegram.org/faq, 2021.

[3] Telegram, "Telegram apis," url:https://core.telegram.org/api, 2021.

[4] Whitfield Diffie and Martin Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, November 1976.

[5] Hou Liping and Shi Lei, "Research on trust model of pki," in *2011 Fourth International Conference on Intelligent Computation Technology and Automation*, March 2011, vol. 1, pp. 232–235.

[6] Xi-Jun Lin, Lin Sun, and Haipeng Qu, "An efficient rsa-based certificateless public key encryption scheme," *Discrete Applied Mathematics*, vol. 241, pp. 39–47, 03 2017.

[7] Anane Nadjia and Anane Mohamed, "Aes ip for hybrid cryptosystem rsa-aes," in *2015 IEEE 12th International Multi-Conference on Systems, Signals Devices (SSD15)*, March 2015, pp. 1–6.

[8] Ashraf Darwish and Maged M El-Gendy, "New hybrid cryptosystem for internet applications," *International Journal of Swarm Intelligence and Evolutionary Computation*, vol. 05, 01 2016.

[9] Ron L. Rivest, Adi Shamir, and Leonard Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[10] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theor.*, vol. 31, no. 4, pp. 469–472, Sept. 2006.

[11] Victor S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology — CRYPTO '85 Proceedings*, Hugh C. Williams, Ed., Berlin, Heidelberg, 1986, pp. 417–426, Springer Berlin Heidelberg.

[12] Neal Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, Jan. 1987.

[13] Pascal Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT '99*, Jacques Stern, Ed., Berlin, Heidelberg, 1999, pp. 223–238, Springer Berlin Heidelberg.

[14] Xun Yi, Russell Paulet, and Elisa Bertino, *Homomorphic Encryption and Applications*, Springer International Publishing, 2014.

[15] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen, "A generalization of paillier's public-key system with applications to electronic voting," *International Journal of Information Security*, vol. 9, no. 6, pp. 371–385, 2010.

[16] Minghu Zheng, Yongquan Cui, and Liang Chen, "Security analysis of a paillier-based threshold proxy signature scheme," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, July 2013, pp. 683–687.

[17] Mohamed Nassar, Abdelkarim Erradi, and Qutaibah M. Malluhi, "Paillier's encryption: Implementation and cloud applications," in *2015 International Conference on Applied Research in Computer Science and Engineering (ICAR)*, Oct 2015, pp. 1–5.

[18] Lejla Batina, Siddika Berna Örs, Bart Preneel, and Joos Vandewalle, "Hardware architectures for public key cryptography," *Integr. VLSI J.*, vol. 34, no. 1-2, pp. 1–64, May 2003.

[19] Francisco Rodriguez-Henriquez, N.A. Saqib, Arturo Díaz

Pérez, and Cetin Kaya Koc, *Cryptographic Algorithms on Reconfigurable Hardware*, Springer US, 2007.

[20] J. Deepakumara, H. M. Heys, and R. Venkatesan, "Fpga implementation of md5 hash algorithm," in *Canadian Conference on Electrical and Computer Engineering 2001. Conference Proceedings (Cat. No.01TH8555)*, May 2001, vol. 2, pp. 919–924 vol.2.

[21] Paweł Chodowiec and Kris Gaj, "Very compact fpga implementation of the aes algorithm," in *Cryptographic Hardware and Embedded Systems - CHES 2003*, Colin D. Walter, Çetin K. Koç, and Christof Paar, Eds., Berlin, Heidelberg, 2003, pp. 319–333, Springer Berlin Heidelberg.

[22] William. N. Chelton and Mohammed Benaissa, "Fast elliptic curve cryptography on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 198–205, Feb 2008.

[23] Anane Nadjia and Anane Mohamed, "High throughput parallel montgomery modular exponentiation on fpga," in *2014 9th International Design and Test Symposium (IDT)*, Dec 2014, pp. 225–230.

[24] Octávio de Souza Martins Gomes and Robson Luiz Moreno, "A compact 128-bits symmetric cryptography hardware module," in *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, Oct 2016, pp. 1–5.

[25] Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari Kermani, "A high-performance and scalable hardware architecture for isogeny-based cryptography," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1594–1609, Nov 2018.

[26] P. Ahuja, H. Soni, and K. Bhavsar, "High performance vedic approach for data security using elliptic curve cryptography on fpga," in *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, 2018, pp. 187–192.

[27] Tanyaporn Sridokmai and Somchai Prakancharoen, "The homomorphic other property of paillier cryptosystem," in *2015 International Conference on Science and Technology (TICST)*, Nov 2015, pp. 356–359.

[28] Alexandre F. Tenca and Çetin K. Koç, "A scalable architecture for montgomery nultiplication," in *Cryptographic Hardware and Embedded Systems*, Çetin K. Koç and Christof Paar, Eds., Berlin, Heidelberg, 1999, pp. 94–108, Springer Berlin Heidelberg.

[29] Guilherme Perin, Daniel Gomes Mesquita, and João Baptista Martins, "Montgomery modular multiplication on reconfigurable hardware: Systolic versus multiplexed implementation," *Int. J. Reconfig. Comput.*, vol. 2011, pp. 6:1–6:10, Jan. 2011.

[30] Peter L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, abril 1985.

[31] Sadiel De la Fe and Carles Ferrer, "A secure algorithm for inversion modulo 2k," *Cryptography*, vol. 2, no. 3, 2018.

[32] S. Kavyashree and B. V. Uma, "Design and implementation of different architectures of montgomery modular multiplication," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, May 2017, pp. 1101–1105.

[33] Antonius P. Renardy, Nur Ahmadi, Ashbir A. Fadila, Naufal Shidqi, and Trio Adiono, "Hardware implementation of montgomery modular multiplication algorithm using iterative architecture," in *2015 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, May 2015, pp. 99–102.

[34] Nadia Nedjah and Luiza de Mourelle, "Efficient hardware for modular exponentiation using the sliding-window method with variable-length partitioning," in *2008 9th International Conference for Young Computer Scientists(ICYCS)*, 11 2008, vol. 00, pp. 1980–1985.

[35] Satyanarayana Vollala, B. Shameedha Begum, Amit D. Joshi, and N. Ramasubramanian, "High-radix modular exponentiation for hardware implementation of public-key cryptography," in *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, Dec 2016, pp. 346–350.

[36] M. Ridwan Apriansyah Budikafa and Reza Pulungan, "Parallelization of modular exponentiations of polynomials," in *2017 3rd International Conference on Science and Technology - Computer (ICST)*, July 2017, pp. 116–121.

[37] JinPing Li, Jiong Shan, Lu Wang, and Min Chen, "Hardware implementation of a secure random number generator," *2014 21st IEEE International Conference on Electronics, Circuits and Systems, ICECS 2014*, pp. 17–20, 02 2015.

[38] Divyanjali, Ankur, and Trishansh Bhardwaj, "Pseudo random bit generation using arithematic progression," in *2015 Fifth International Conference on Advanced Computing Communication Technologies*, Feb 2015, pp. 361–366.

[39] K. Klonick, "The new governors: The people, rules and processes governing online speech," *Harvard Law Review*, vol. 131, pp. 1598–1670, 2018.

[40] S. Y. Wang and D. R. Jones, "Multi-user decision support system," *Information Science and Engineering*, vol. 3, pp. 135–150, 1987.

[41] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[42] Vijaya Ramachandran, "Linear-time algorithm for race detection in transistor switch-level circuits.," in *Unknown Host Publication Title*. 12 1983, pp. 345–348, IEEE.

[43] H. T. Kung, "Memory requirements for balanced computer architectures," Tech. Rep. CMU-CS-85-158, Department of Computer Science, Carnegie-Mellon University, 1985.

[44] C. A. O'Reilly and L. R. Pondy, "Organizational communication," in *Organizational Behavior*, S. Kern, Ed., pp. 111–150. Grid Publishing Inc., Columbus, Ohio, 1979.

Compresión de imágenes hiperespectrales con distorsión adaptable en lógica reconfigurale

Julián Caba[1], Dirk Stroobandt[2], María Díaz[3], Jesús Barba[1], Fernando Rincón[1], José Antonio de la Torre[1], Soledad Escolar[1], Sebastián López[3] y Juan Carlos López[1]

Resumen— **Las soluciones de compresión con pérdidas han crecido durante las últimas décadas debido al incremento de la tasa de datos de los sensores hiperespectrales de nueva generación, sin embargo las técnicas de compresión lineal incluyen información poco útil en regiones de poco interés para la aplicación final y al mismo tiempo escasa información en áreas de interés. En este trabajo, el compresor con pérdida HyperLCA ha sido extendido para incluir la característica de distorsión adaptativa en tiempo de ejecución, aportando varios ratios de compresión en un mismo escenario. La solución ha sido diseñada para mantener la posibilidad de desplegar la solución en dispositivos hardware reconfigurables (FPGAs). Los experimentos demuestran que la nueva versión del compresor es capaz de procesar 1024x1024 imágenes hiperespectrales y 180 bandas espectrales (377,5MB) en 0,935 segundos con un consumo de 1,145 vatios. Además, los resultados experimentales también revelan que nuestra arquitectura presenta un alto rendimiento (MSamples/s) y una notable eficiencia energética (MB/s por vatio), 10 y 6 veces superior a la mejor solución del estado del arte, respectivamente.**

Palabras clave— **FPGAs, hyperspectral imaging, anomaly detection, line-by-line performance, real-time, High-Level Synthesis, low-power.**

I. Introducción

La información recogida por sensores hiperespectrales proporciona una gran riqueza de información espectral, convirtiéndola en una de las principales candidatas para el análisis de áreas terrestres y, por ello, ha adquirido una importante relevancia, siendo ampliamente utilizada para una gran variedad de aplicaciones de teledetección como la agricultura de precisión, la cartografía geológica o la exploración minera. Sin embargo, la gran cantidad de datos recogidos por estos sensores requiere enormes recursos de almacenamiento a bordo o comunicaciones de gran ancho de banda, pero ambos son limitados. Además, los avances tecnológicos promueven la comercialización de sensores con mayores resoluciones espectrales y espaciales que hacen que el procesado de datos a bordo sea más desafiante [1], [2].

Tradicionalmente, la información es captada por sensores hiperespectrales montados sobre un vehículo aéreo no tripulado (UAV), donde el procesado de dicha información no suele realizarse debido principalmente por el bajo rendimiento y la limitada capacidad de energía de la plataforma que integra recursos de cómputo y de vuelo. Normalmente, los dispositivos que integran esta plataforma son dispositivos de bajo consumo y coste optimizado para procesar los datos captados, pero estos no cuentan con un alto rendimiento [3]. Existen varías alternativas que se han adoptado en los últimos años; 1) Las imágenes se descargan en la superficie terrestre para ser procesadas fuera de línea por sistemas de alto rendimiento; 2) En los UAVs las imágenes suelen almacenarse a bordo y procesarse cuando finaliza la misión de vuelo [4]. Recientemente, se han hecho algunos esfuerzos para transmitir las imágenes a tierra tan pronto como se capturan, pero se requiere una conexión punto a punto con un gran ancho de banda para reducir los grandes retrasos [5]. En cualquier caso, la transferencia de grandes volúmenes de datos es uno de los principales problemas que puede afectar al rendimiento global, así como el consumo energético consumido por la transmisión [6].

Dado que el ancho de banda es limitado, así como la memoria disponible en los dispositivos y los recursos de hardware dedicados, como los DSP, la forma de abordar este reto es utilizar técnicas de compresión de imágenes hiperespectrales a bordo. Aunque existe una gran variedad de algoritmos de compresión en la literatura, los algoritmos de compresión sin pérdidas son la opción preferente porque preservan la mayor parte de la información hiperespectral [2]. Las técnicas sin pérdidas producen datos no distorsionados después del proceso de descompresión, pero la relación de compresión no es lo suficientemente grande, sin embargo los métodos casi sin pérdidas permiten obtener valores más grandes de relación de compresión, introduciendo distorsiones que pueden ser controladas. Además, los sensores de última generación aumentan la velocidad de transmisión de datos, lo que requiere una mayor relación de compresión y/o reducir el tiempo de compresión para evitar la acumulación de datos sin comprimir y, por tanto, una transmisión eficiente [7]. Por lo tanto, los anchos de banda de comunicación limitados y los volúmenes de datos cada vez mayores obligan a pasar de las técnicas de compresión (casi) sin pérdidas a las técnicas de compresión con pérdidas, en las que se ha realizado un gran esfuerzo de investigación en los últimos años [2], [8].

En escenarios de sistemas móviles embebidos, los dispositivos de procesamiento paralelo, como las FPGAs, son adecuados para implementar algoritmos de compresión de imágenes hiperespectrales, debido al

[1]Dpto. Tecnologías y Sistemas de Información, UCLM, 13071 Ciudad Real, España. e-mail: {julian.caba, jesus.barba, fernando.rincon, joseantonio.torre, soledad.escolar, juancarlos.lopez}@uclm.es

[2]Ghent University, 9000 Ghent, Belgium. e-mail: dirk.stroobandt@ugent.be

[3]Instituto Universitario de Microelectrónica Aplicada (IUMA), 35017 Las Palmas de Gran Canaria, España. e-mail: {mdmartin, seblopez}@iuma.ulpgc.es

equilibrio entre el grado de paralelismo y el coste-energía, además del ahorro de costes que supone una solución basada en FPGAs con pocos recursos. Desafortunadamente, los limitados recursos computacionales de estos dispositivos suponen un nuevo reto cuando una solución se basa en este tipo de tecnologías, por lo que los esquemas de compresión de baja complejidad se erigen como la solución más práctica para escenarios tan restringidos [9], [10]. Sin embargo, la mayoría de los compresores con pérdida del estado del arte se basan en algoritmos existentes de compresión de imágenes 2D o vídeo, que se consideran de alta carga computacional, intensivos en requerimientos de memoria y de naturaleza no paralela [11]. Este hecho provoca que su uso sea limitado en entornos con recursos limitados, como es el caso de la compresión embarcada [12]. En este contexto, el Algoritmo de Compresión con Pérdidas para Sistemas de Imágenes Hiperespectrales (HyperLCA) [13] ha sido desarrollado como un compresor con pérdidas amigable con el hardware para imágenes hiperespectrales, que proporciona buenos ratios de compresión con una carga computacional razonable, ya que el proceso de compresión se basa en operaciones de transformación. Además, este algoritmo se ha diseñado para cumplir las limitaciones impuestas por los escáneres pushbroom/whiskbroom, considerando cada bloque de forma independiente, lo que se traduce en un menor uso de recursos de hardware [14]. Su idoneidad para aplicaciones de rendimiento en tiempo real ha sido analizada previamente en [15].

En este trabajo, se ha ampliado el algoritmo HyperLCA para incluir la función de distorsión adaptativa sin añadir una sobrecarga respecto a la implementación original, convirtiéndolo en un compresor inteligente al seleccionar los bloques relevantes cercanos a un patrón de firma predefinido. Por lo tanto, el algoritmo se ha adaptado para ejecutarse de forma más flexible con diferentes ratios de compresión sin modificar el conjunto de operaciones básicas realizadas en su versión original. Además, se ha llevado a cabo un análisis del HyperLCA basado en FPGA con función de distorsión adaptativa estableciendo diferentes reglas de la distorsión aplicada sobre el mismo escenario, lo que da lugar a una multiplicidad de relaciones de compresión dentro de la misma imagen y a una amplia gama de prestaciones de compresión de calidad. Esto significa que en la imagen comprimida, que contiene distorsiones de alta y baja tasa, se producen múltiples relaciones tasa-distorsión, de modo que no sólo se conservan perfectamente los píxeles hiperespectrales más diferentes; es decir, estos píxeles se encuentran dentro de la información extraída junto con píxeles extra relevantes de bloques interesantes, lo que se traduce en pérdidas insignificantes, ya que algunos bloques comprimidos tienen menos distorsión. Este hecho beneficia a muchas aplicaciones de imágenes hiperespectrales en las que la resolución espectral es decisiva; los bloques que contienen un gran número de similitudes pueden comprimirse con un criterio diferente al de los bloques con menores

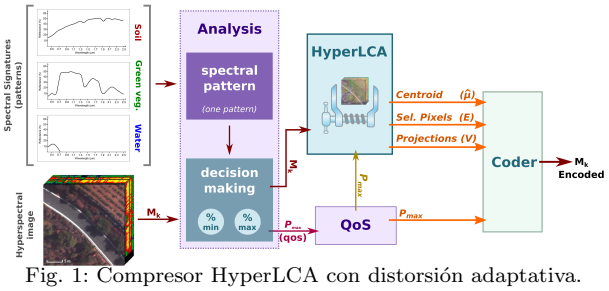


Fig. 1: Compresor HyperLCA con distorsión adaptativa.

similitudes con el patrón. Los bloques relevantes se seleccionan en tiempo de ejecución comparando píxel a píxel con un patrón de firma buscando similitudes en cada una de las bandas del espectro. Por último, la arquitectura se ha comparado con la versión anterior implementada en [16] y otros compresores del estado del arte en términos de rendimiento y utilización de recursos hardware. En este contexto, la mayor motivación de este trabajo es demostrar las afirmaciones anteriormente mencionadas sobre el HyperLCA con control de calidad, y por lo tanto, el objetivo es contribuir a la comunidad científica con un compresor inteligente con pérdida para imágenes hiperespectrales, que enriquezca aquellos bloques de mayor interés en un escenario inesperado/desconocido, mediante el uso de una solución de coste optimizado y energéticamente eficiente basada en tecnología FPGA.

II. Algoritmo HyperLCA

El algoritmo HyperLCA es un compresor con pérdidas basado en transformadas para imágenes hiperespectrales, cuya versión original se ha modificado para convertirla en una versión apta para hardware. Así, se ha rediseñado el algoritmo para conseguir altas relaciones tasa de compresión-distorsión, así como se ha disminuido la carga computacional debido al alto nivel de paralelismo implementado para aplicaciones basadas en sensores pushbroom/whiskbroom. Todos los cambios realizados permiten el uso de arquitecturas de procesamiento paralelo, como GPUs o FPGAs, sobre las que se ha implementado el algoritmo, [15] y [16], respectivamente. Además, el algoritmo HyperLCA ha sido especialmente diseñado para trabajar dentro de rangos numéricos específicos y utilizar aritmética entera, especificando la precisión necesaria para que las operaciones sean adecuadas en arquitecturas paralelas, tal y como se ha comentado previamente en [16], [14], [15].

La solución propuesta puede procesar independientemente bloques de la imagen con independencia de la alineación espacial de los mismos, lo que facilita la paralelización del proceso de compresión. Esto significa que los píxels hiperespectrales pueden procesarse sin dependencias. Sobre esta base, este trabajo incorpora la novedad de analizar cada píxel en tiempo de ejecución para determinar la tasa de distorsión que debe aplicarse al bloque que se está procesando, en lugar de establecer una tasa de compresión mínima deseada para todos los bloques capturados por el sensor, como hacen las versiones anteriores del algoritmo. La figura 1 muestra una visión

general de la nueva versión del algoritmo que representa las tres principales etapas de cálculo implicadas en el compresor HyperLCA con característica de distorsión adaptativa, compuestas por una una nueva etapa de *preprocesado o análisis de bloques*, que determinará el ratio de compresión a aplicar a cada bloque, una etapa de *transformación espectral*, cuyo core de operaciones es heredado de anteriores versiones y una etapa de *codificación*, que se ha adaptado de versiones anteriores para indicar el factor de distorsión aplicado a cada uno de los bloques.

A. Etapa 1: Preprocesamiento o análisis de bloques

En primer lugar, la imagen hiperespectral se divide en bloques compuestos por BS píxeles horizontales consecutivos ($\mathbf{M}_k$), también conocidos como líneas o bloques, que se procesarán. A continuación, se calculan los p_{max} píxeles más representativos dentro de un bloque ($\mathbf{M}_k$). Las implementaciones anteriores del algoritmo HyperLCA inicializan el parámetro p_{max} a partir de tres parámetros de entrada: CR (relación de compresión mínima deseada), N_{bits} (número de bits) y BS (tamaño del bloque) para determinar el número de transformaciones realizadas en el bloque hiperespectral. Sin embargo, en este trabajo, p_{max} no se fija en el momento del diseño, sino que se recalcula para cada bloque en función del número de píxeles similares con una firma de patrón hiperespectral. El equilibrio entre las métricas de calidad SNR, RMSE y MAD ha sido evaluado previamente en publicaciones anteriores combinando diferentes configuraciones de los parámetros de entrada (CR, N_{bits} y BS) [15]. En concreto, una de las mejores configuraciones es establecer los parámetros N_{bits} y BS en 12 y 1024, respectivamente. Estos valores se mantienen para calcular p_{max} en tiempo de ejecución, mientras que CR dependerá del número de píxeles hiperespectrales cercanos a la firma del patrón.

Por lo tanto, el compresor HyperLCA determina el número de transformaciones realizadas en el bloque que se está procesando, $\mathbf{M}_k$, como se muestra en la ecuación (1), donde DR se refiere al número de bits por píxel por banda; nb representa el número de bandas; BS es el número de píxeles horizontales consecutivos en un mismo bloque; CR se refiere a la relación de compresión deseada, definida como la relación entre el número de bits de la imagen original y los de los datos comprimidos; y N_{bits} es el número de bits que determina la precisión y el rango dinámico a utilizar para representar los valores de los datos comprimidos. Así, el número de transformaciones realizadas, p_{max}, determina directamente la máxima relación de compresión que se puede alcanzar con la configuración seleccionada, en la que valores más altos de p_{max} dan lugar a mejores imágenes reconstruidas, pero menores relaciones de compresión.

$$p_{\max} \leq \frac{DR \cdot nb \cdot (BS - CR)}{CR \cdot (DR \cdot nb + N_{\text{bits}} \cdot BS)} \qquad (1)$$

La relación de compresión (CR) utilizada en el cálculo de p_{max} para un bloque debe determinarse

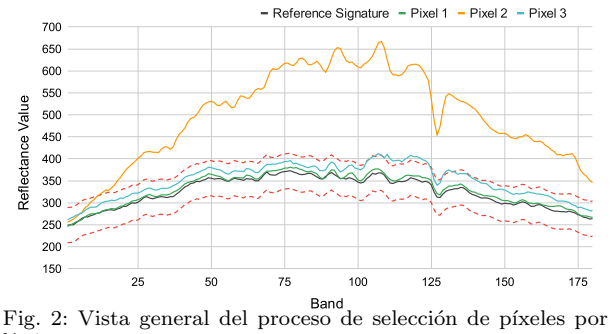


Fig. 2: Vista general del proceso de selección de píxeles por límites.

analizando cada uno de los píxeles de dicho bloque ($\mathbf{M}_k$), comparándolos uno a uno con una firma hiperespectral de referencia (patrón de referencia). Este proceso puede realizarse utilizando la distancia euclidiana con la firma de referencia y los píxeles hiperespectrales dentro de un bloque para determinar si el bloque, que está siendo procesado, contiene un alto porcentaje de píxeles cercanos a la referencia o, por el contrario, contiene pocas coincidencias. Sin embargo, el cálculo de la distancia euclídea es costoso en dispositivos reconfigurables y hace necesario buscar alternativas. En este sentido, este trabajo propone una solución amigable con el hardware mediante el uso de valores de band-limit, donde se define un valor *delta* para establecer los límites superior e inferior que dibujan la misma trama que la firma de referencia. Los resultados de este método son similares a los obtenidos por el método de la distancia euclídea.

La figura 2 muestra gráficamente el enfoque de límite de banda adoptado, donde se trazan tres píxeles hiperespectrales y una firma de referencia (línea negra). Además, las líneas rojas discontinuas definen los límites superior e inferior cuya tendencia es idéntica a la trazada por la firma de referencia. Por lo tanto, las firmas hiperespectrales que se encuentran dentro de ambos límites se consideran próximas a la referencia (línea verde de la figura 2), así como aquellos píxeles cuyo número de valores de reflectancia está fuera de los límites (línea azul de la figura 2); el número de valores fuera de los límites es configurable. Este hecho significa que las firmas hiperespectrales cuya reflectancia es muy similar a la de referencia, excepto en un pequeño número de bandas, también son consideradas próximas a la firma de referencia. Mientras tanto, los píxeles hiperespectrales cuyos valores de reflectancia más están fuera de los límites no se consideran (línea naranja de la figura 2).

Una vez que todos los píxeles hiperespectrales dentro de un bloque se definen como dentro o fuera de los límites, el CR se establece analizando el número de píxeles dentro de los límites. La figura 3 muestra tres ejemplos con diferentes ratios de compresión sobre el mismo escenario. En primer lugar, la figura 3a es una figura RGB original de un escenario de viñedos, mientras que la figura 3e muestra un patrón en el que los colores blanco y negro representan las firmas vegetales verdes y otros materiales, respectivamente. Desafortunadamente, el sensor utilizado en este trabajo captura la información hiperespectral

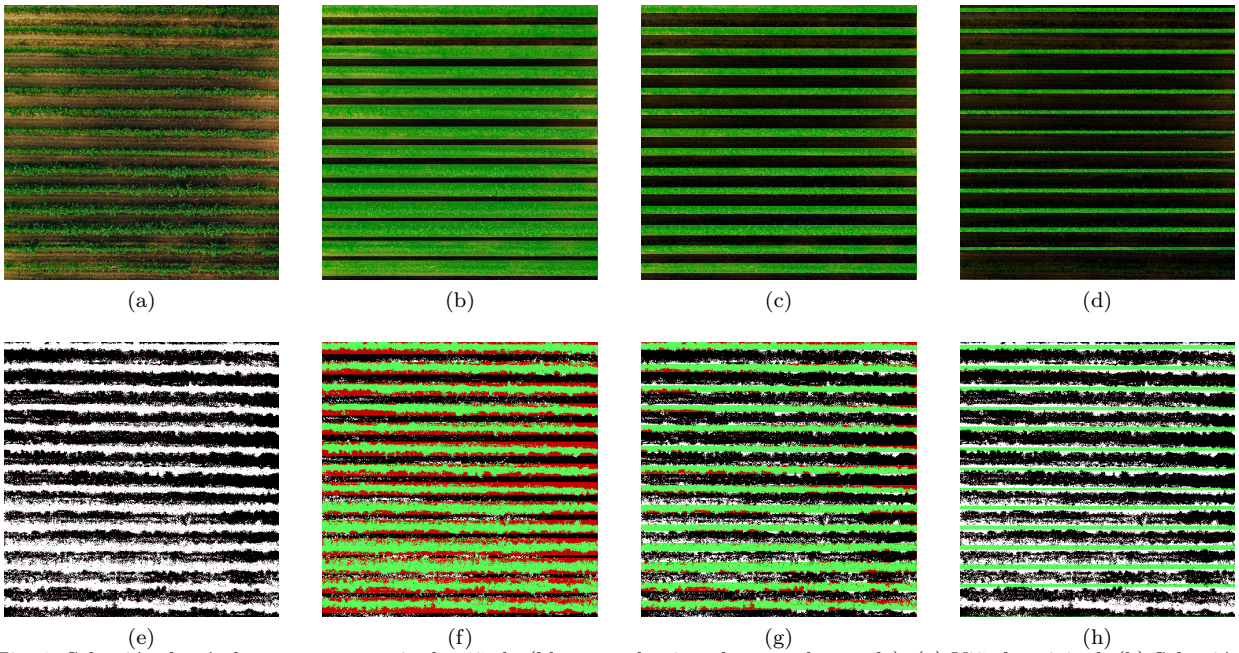


Fig. 3: Selección de píxeles en un escenario de viñedo (bloques seleccionados en color verde). (a) Viñedo original. (b) Selección de líneas con relación de compresión baja. (c) Selección de líneas con relación de compresión media. (d) Selección de líneas con una relación de compresión alta. (e) Patrón de selección de píxeles. (f) Diferencias entre los píxeles seleccionados en (b) y el patrón. (g) Diferencias entre los píxeles seleccionados en (c) y el patrón. (h) Diferencias entre los píxeles seleccionados en (d) y el patrón.

línea a línea, esto significa que no se obtiene inmediatamente toda la información hiperespectral de un escenario, por lo que las operaciones deben realizarse sobre una línea ($\mathbf{M}_k$) con BS píxeles hiperespectrales. Así, cuando se requiere una baja distorsión pero sólo en los bloques que contienen información útil para la aplicación final, el número de píxeles hiperespectrales próximos a la referencia debe ser pequeño. En este sentido, las figuras 3b, 3c y 3d resaltan en verde los bloques cuyo CR es bajo, es decir, estas líneas contienen información útil para la aplicación final, mientras que los bloques cuya información hiperespectral no es demasiado relevante se resaltan en negro. Por su parte, las figuras 3f, 3g y 3h destacan la pérdida, acierto y exceso de información hiperespectral respecto a la extracción de información ideal (figura 3e) en blanco, verde y rojo, respectivamente. El proceso de selección de bloques relevantes en los diferentes escenarios cumple los siguientes criterios: al menos 10 píxeles cercanos a la referencia para el escenario representado en la figura 3b, al menos 300 píxeles cercanos en el caso de la figura 3c y al menos 800 píxeles en el último caso (figura 3d).

B. Etapa 2: Transformación espectral

El compresor HyperLCA se encuentra entre los algoritmos basados en transformaciones que emplean una versión modificada del conocido método de ortogonalización Gram-Schmidt, ampliamente utilizado en compresión con pérdidas debido a su moderada complejidad en la que no se requiere reordenación de bandas. La idea básica de esta clase de algoritmos es mapear el dominio espacial de una imagen hiperespectral en su dominio de transformación [2]. En este sentido, la etapa *transformación espectral* selecciona

Algorithm 1 Transformación HyperLCA.

Inputs:
$\mathbf{M}_k = [\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_{BS}]$, p_{max}
Outputs:
$\hat{\boldsymbol{\mu}}$; $\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_{p_{max}}]$; $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_{p_{max}}]$
Algorithm:
Average pixel: $\hat{\boldsymbol{\mu}}$;
Centralized image: $\mathbf{C} = \mathbf{M}_k - \hat{\boldsymbol{\mu}} = [\mathbf{c}_1, \mathbf{c}_2, ...\mathbf{c}_{BS}]$;
for $n = 1$ **to** p_{max} **do**
  for $j = 1$ **to** BS **do**
    Brightness Calculation: $\mathbf{b}_j = \mathbf{c}'_j \cdot \mathbf{c}_j$;
  end for
  Maximum Brightness: $j_{max} = \text{argmax}(\mathbf{b}_j)$;
  Extracted pixels: $\mathbf{e}_n = \mathbf{r}_{j_{max}}$;
  $\mathbf{q}_n = \mathbf{c}_{j_{max}}$;
  $\mathbf{u}_n = \mathbf{q}_n / b_{j_{max}}$;
  Projection vector: $\mathbf{v}_n = \mathbf{u}'_n \cdot \mathbf{C}$;
  Information Subtraction: $\mathbf{C} = \mathbf{C} - \mathbf{q}_n \cdot \mathbf{v}_n$;
end for

los píxeles más diferentes de una imagen utilizando técnicas de proyección ortogonal. Así, los píxeles seleccionados se utilizan para proyectar la imagen con el fin de eliminar redundancias y obtener así una imagen espectral descorrelacionada y comprimida.

La *transformación espectral* se describe detalladamente en el algoritmo 1 y es la única etapa donde el core de operaciones se mantiene respecto a anteriores versiones. Las entradas de esta etapa son el bloque hiperespectral a comprimir ($\mathbf{M}_k$), que es el mismo que está siendo analizado por la etapa *Preproceso*, y el número de píxeles más diferentes a extraer ($p_{\max}$), que es la salida de la etapa *Preproceso*, es decir, se en-

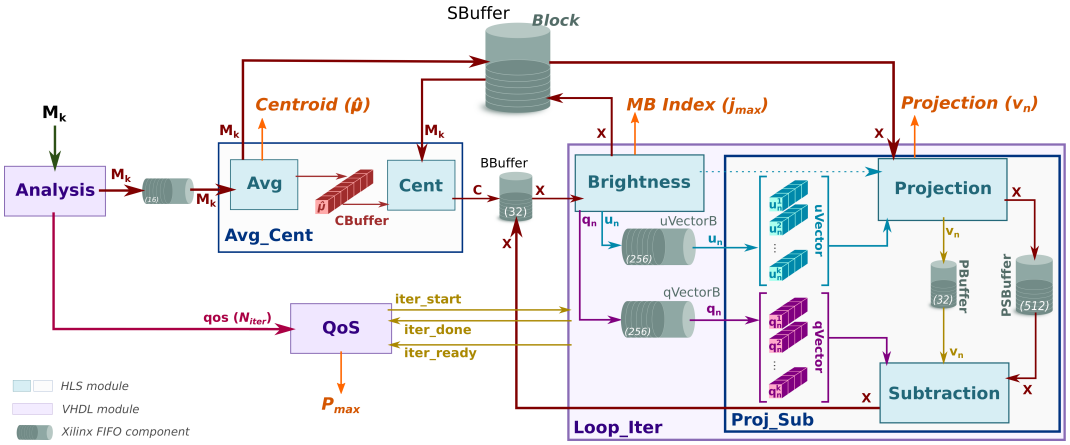


Fig. 4: Implementación hardware del algoritmo de compresión HyperLCA con distorsión adaptativa.

carga de determinar el número de transformaciones o proyecciones ortogonales a realizar sobre el bloque hiperespectral. Mientras que las salidas de esta etapa son el píxel medio ($\hat{\boldsymbol{\mu}}$) del bloque hiperespectral de entrada ($\mathbf{M}_k$), el conjunto de índices relacionados con los p_{rmmax} píxeles más diferentes ($\mathbf{E}$) y sus vectores de proyección ($\mathbf{V}$).

C. Etapa 3: Codificación

La última etapa del compresor HyperLCA realiza la entropía-codificación de los vectores recibidos de la etapa *Transformación Espectral*, donde el uso del algoritmo de Golomb-Rice [17] permite considerar cada vector independientemente del resto, es decir, las salidas de esta etapa pueden ser consumidas a medida que se reciben, por lo que ambas etapas se realizan en paralelo. Para ello, se calcula el parámetro de compresión (N) como valor medio del vector que se está procesando. Posteriormente, se dividen los elementos del vector por N y se obtienen el cociente (q) y el resto (r) de dicha operación. En segundo lugar, se calcula la menor potencia de 2 mayor que N como $b = log_2(N) + 1$. El cociente (q) se codifica utilizando código unario, mientras que el resto (r) se codifica como binario plano utilizando $b-1$ bits para valores de r inferiores a $2^b - N$, de lo contrario se codifica como $r + 2^b - N$ utilizando b bits.

D. Etapa 4: Generación del bitstream

Por último, se realiza la etapa *empaquetado de datos* para generar un único flujo de bits que contiene las salidas de las etapas de compresión anteriormente mencionadas. La figura 5 muestra gráficamente la estructura generada para el flujo de bits comprimido, que se divide en dos bloques jerárquicos.

En primer lugar, tiene lugar una cabecera global al principio de la cadena donde se recoge la información relativa a la imagen hiperespectral, incluidos los parámetros utilizados en el proceso de compresión; información espacial (número de columnas y filas) y espectral (número de bandas) de la imagen procesada, el tamaño de bloque utilizado (BS), el número de bits por píxel por banda (DR) y el número de bits utilizados para representar los valores de los datos comprimidos (N_{bits}).

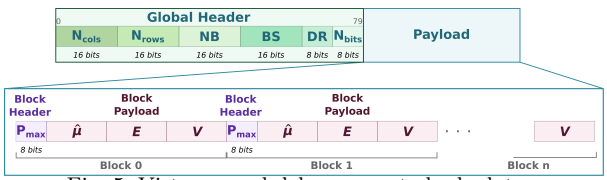


Fig. 5: Vista general del empaquetado de datos.

La carga útil se coloca después de esa cabecera global, que se divide en bloques que contienen dos campos: *cabecera de bloque* y *carga útil*. Debido a que los bloques de una imagen hiperespectral se comprimen con un número variado de transformaciones, es obligatoria una cabecera de 8 bits para determinar el p_{max} aplicado en cada bloque. Tras la *cabecera de bloque*, se encuentra la *carga útil* del mismo, que está compuesto por un único centroide independientemente de la distorsión aplicada sobre el bloque y a continuación los vectores $\mathbf{E}$ (píxeles más diferentes) y $\mathbf{V}$ (proyecciones), cuyo número viene denotado por la mencionada cabecera de 8 bits.

III. Implementación hardware (FPGA)

En este trabajo se han reutilizado aceleradores hardware especializados de [16] para construir la nueva arquitectura de la transformación HyperLCA, añadiendo el comportamiento dinámico necesario para incluir múltiples ratios de compresión. La figura 4 muestra una visión general de la arquitectura hardware implementada para las etapas de *preprocesamiento* y *transformación espectral*, donde los recuadros azules corresponden a los módulos descritos en HLS, y por tanto heredados del desarrollo anterior. Estos módulos están conectados a través de buffers de memoria y lógica a medida que permite orquestarlos para obtener el comportamiento deseado. Para mayor claridad, la correspondencia de las cajas azules con las operaciones realizadas en el algoritmo 1 es la siguiente: *Avg_Cent* corresponde a las líneas 1 y 2, mientras que *loop_iter* es el cuerpo del bucle principal que comprende desde la línea 3 a la 13, donde *brightness* se calcula en el bucle interior y en la línea 7 del Algoritmo 1, obteniéndose además vectores ortogonales ($\mathbf{q}$ y $\mathbf{u}$) una vez seleccionado el píxel más brillante. Mientras tanto, *proyection* y

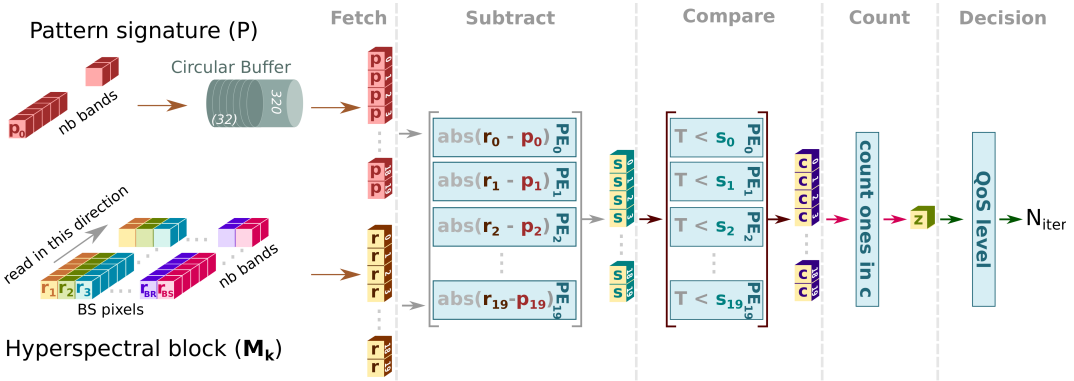


Fig. 6: Arquitectura basada en pipelines para calcular el número de transformaciones (PEs = 20).

subtraction coinciden con las líneas 11 y 12 del algoritmo 1, respectivamente. En busca de mejorar el rendimiento, la arquitectura introduce una mejora a través de la partición de los vectores ortogonales, **q** y **u**, utilizando VHDL en lugar de aplicar una directiva en la descripción HLS.

Hasta aquí, la arquitectura descrita se encarga de realizar el proceso de transformación sobre los bloques hiperespectrales ($\mathbf{M}_k$). Sin embargo, las operaciones de transformación realizadas, es decir, el número de ejecuciones del módulo *loop_iter*, se calcula ahora en tiempo de ejecución. El número de iteraciones está relacionado con la relación de compresión aplicada a un bloque, lo que también significa la cantidad de datos hiperespectrales que se representan para cada bloque. Así, el módulo *QoS* gestiona este proceso, además de notificar el valor p_{max} utilizado para codificar el bloque hiperespectral actual ($\mathbf{M}_k$). Por su parte, el módulo *Analysis* se encarga de calcular el número de iteraciones a realizar en función de la similitud entre los píxeles que componen un bloque hiperespectral y la firma del patrón.

Subtract. Una vez que los datos hiperespectrales están listos, la etapa *subtract* sustrae de forma absoluta del bloque hiperespectral actual ($\mathbf{M}_k$) la firma del patrón por bandas en paralelo, de forma que a la primera banda del píxel hiperespectral perteneciente al bloque procesado se le resta la primera banda del patrón almacenado, luego la segunda banda de ambos y así sucesivamente. Los resultados de esta etapa se almacenan en los registros s_i, que se representan como cuadros amarillos y verdes en la figura 6.

Compare. Los resultados obtenidos de cada PE en la etapa *subtract* se comparan con un valor constante, T, que es el valor *delta* predefinido. Así, si la constante T es menor que el resultado de la resta absoluta obtenida en la etapa anterior, significa que la banda correspondiente está dentro de los límites. Por el contrario, si la comparación da como resultado que T es mayor, implica que el valor de la banda está fuera de los límites. La salida de esta etapa es un cero o un uno cuando la banda está fuera o dentro de los límites, respectivamente. De forma similar a la etapa anterior, los resultados de las comparaciones también se almacenan en los registros c_i (cuadros amarillos y morados en la figura 6).

Count. En esta etapa, los registros que contienen un uno, es decir, la banda está dentro de los límites, se cuentan y se suman a una suma parcial, que se almacena en el registro z (ver figura 6).

Por último, la etapa *decision* se realiza después de comparar todo el píxel hiperespectral de un bloque con la firma del patrón. Determina si un píxel está próximo al patrón a partir del valor de la suma almacenado en el registro z y calculado por la etapa *count*. En este sentido, no es necesario que todos los valores de banda estén dentro de los límites, la solución permite que varias bandas estén fuera de ellos (ver línea azul en la figura 2). Además, la etapa *decision* establece el número de operaciones de transformación realizadas por la etapa *transformación espectral*. Se calcula contando el número de píxeles próximos a la firma del patrón (T_c) y, a continuación, se compara dicha suma con las reglas predefinidas por el usuario, que determina el grado de distorsión. Para ello, el usuario debe proporcionar dos constantes R_{min} y R_{max} para aplicar los siguientes casos, donde QoS_{max}, QoS_{min} y QoS_{med} son el número de iteraciones que también están predefinidas; un número alto significa mayor información espectral extraída.

$$\begin{cases} N_{iter} = QoS_{max}, & \text{if } T_c \geq R_{max} \\ N_{iter} = QoS_{min}, & \text{if } T_c \leq R_{min} \\ N_{iter} = QoS_{med}, & \text{otherwise} \end{cases}$$

IV. Resultados experimentales

En esta sección se evalúa el compresor con distorsión adaptativa mediante el análisis de la arquitectura basada en FPGA implementada en una ZC7Z020-CLG484 y el rendimiento alcanzado por la misma, comparando el rendimiento y el consumo de energía cuando varios PE trabajan en paralelo. En esta sección también se evalúa la precisión de la selección de píxeles cercanos mediante límites frente al uso de la distancia euclídea.

A. Dataset

El rendimiento de la arquitectura presentada ha sido evaluado mediante un conjunto de imágenes hiperespectrales, que fueron captadas por una plataforma aérea personalizada sobre diferentes zonas de cultivo en la isla de Gran Canaria (España). El conjunto de datos contiene 4 imágenes hiperespectrales recogidas sobre dos zonas de viñedos diferentes, cuyas

Tabla I: Comparación de los resultados obtenidos entre la distancia euclidiana y el método propuesto.

Limites	Errores	Verdaderos Positivos ($\mathbf{E} \cap \mathbf{L}$)	Falsos Negativos ($\mathbf{E} - \mathbf{L}$)	Falsos Positivos ($\mathbf{L} - \mathbf{E}$)	Verdaderos Negativos ($\mathbf{U} - (\mathbf{E} \cup \mathbf{L})$)	$\mathbf{E} \subseteq \mathbf{L}$	$\mathbf{L} \subseteq \mathbf{E}$
±15	20	49.093 %	50.906 %	0.0000 %	100.00 %	0.6835 %	100.00 %
	25	54.467 %	45.532 %	0.0000 %	100.00 %	0.7812 %	100.00 %
	30	60.003 %	39.996 %	0.0077 %	99.999 %	1.0745 %	99.804 %
±20	20	82.158 %	17.841 %	0.2145 %	99.992 %	6.3476 %	93.261 %
	25	89.024 %	10.975 %	0.9618 %	99.963 %	14.160 %	73.339 %
	30	94.686 %	5.3133 %	2.8799 %	99.879 %	28.711 %	41.601 %
±23	20	96.843 %	3.1564 %	7.6101 %	99.658 %	42.187 %	22.949 %
	25	99.256 %	0.7437 %	12.146 %	99.413 %	76.172 %	8.0078 %
	30	99.958 %	0.0418 %	17.789 %	99.074 %	98.339 %	2.6369 %
±24	20	98.703 %	1.2969 %	12.198 %	99.413 %	65.332 %	11.230 %
	25	99.823 %	0.1766 %	17.296 %	99.106 %	93.652 %	3.0273 %
	30	99.995 %	0.0046 %	22.758 %	98.739 %	99.804 %	1.3671 %
±25	20	99.526 %	0.4741 %	16.796 %	99.140 %	85.156 %	4.0039 %
	25	99.944 %	0.0557 %	22.292 %	98.773 %	97.949 %	1.5625 %
	30	100.00 %	0.0000 %	27.498 %	98.377 %	100.00 %	0.7812 %
±26	20	99.842 %	0.1580 %	21.478 %	98.831 %	94.443 %	2.2461 %
	25	99.991 %	0.0093 %	26.839 %	98.430 %	99.609 %	0.8789 %
	30	100.00 %	0.0000 %	31.697 %	98.014 %	100.00 %	0.6836 %
±27	20	99.939 %	0.0604 %	25.837 %	98.510 %	97.754 %	1.0742 %
	25	100.00 %	0.0000 %	30.963 %	98.081 %	100.00 %	0.6836 %
	30	100.00 %	0.0000 %	35.542 %	97.641 %	100.00 %	0.6836 %
±30	20	100.00 %	0.0000 %	37.044 %	97.482 %	100.00 %	0.4883 %
	25	100.00 %	0.0000 %	41.147 %	97.008 %	100.00 %	0.3906 %
	30	100.00 %	0.0000 %	44.861 %	96.519 %	100.00 %	0.2929 %

coordenadas exactas son 27°59′35,6″N 15°36′25,6″W y 27°59′15,2″N 15°35′51,9″W.

La plataforma de adquisición se compone de una cámara hiperespectral *Specim FX10* pushbroom en un dron DJI Matrice 600 [18]. El sensor de imagen captura 1024 píxeles espaciales por pista y hasta 224 bandas espectrales en el rango entre 400 y 1000 nm. Sin embargo, en los experimentos realizados sólo se trabaja con 180 bandas espectrales; se descartan las 10 primeras bandas espectrales así como las 34 últimas, debido a que la respuesta en los límites del espectro electromagnético es bajo.

B. Precisión del método de selección de píxeles

Se ha evaluado el método de selección de píxeles mediante límites superior e inferior propuesto en este trabajo, comparando los resultados con los obtenidos mediante la distancia euclídea. Para ello, el análisis se ha realizado utilizando las operaciones de la Teoría de Conjuntos, aplicando seis métricas diferentes para analizar el conjunto de píxeles seleccionados. Cabe destacar que el resultado obtenido, independientemente del método utilizado, es un conjunto de píxeles similares al patrón de referencia, por lo que el análisis debe guiarse por las similitudes entre conjuntos de resultados.

En este contexto, el conjunto universal, $\mathbf{U}$, está compuesto por el índice de píxeles dentro de un bloque hiperespectral ($\mathbf{M}_k$). Así, $\mathbf{U}$ se define como $\mathbf{U} = \{0, ..., 1023\}$. El conjunto de píxeles seleccionados por el algoritmo de distancia euclidiana se denota como $\mathbf{E}$ ($\mathbf{E} \subseteq \mathbf{U}$), mientras que los extraídos por el método de límites se representa como $\mathbf{L}$ ($\mathbf{L} \subseteq \mathbf{U}$). En las líneas siguientes se describen las seis métricas utilizadas para determinar el grado de similitud de los dos enfoques.

Verdaderos Positivos. Esta métrica analiza el número de píxeles comunes seleccionados por los dos algoritmos, es decir, la operación de intersección se aplica a los conjuntos $\mathbf{E}$ y $\mathbf{L}$, que se obtienen mediante la distancia euclídea y el método del límite, respectivamente. Un porcentaje alto significa que la mayoría de los píxeles seleccionados por la distancia euclídea también son elegidos por el método propuesto, pero no implica que sean muy similares; el conjunto $\mathbf{L}$ puede contener más píxeles que $\mathbf{E}$.

Falsos Negativos. En este caso, representa los píxeles dentro de $\mathbf{E}$ pero no en $\mathbf{L}$, o en otras palabras, denota los píxeles que el método propuesto debería haber incluido en su conjunto de soluciones pero que en realidad no están incluidos. Esta métrica y la métrica de verdaderos positivos proporcionan el grado de similitud entre conjuntos; un bajo porcentaje de falsos negativos y un alto porcentaje de verdaderos positivos significa que dos conjuntos son muy similares.

Falsos Positivos. Esta métrica mide el grado de píxeles extra considerados por el método del límite. También se calcula mediante la operación diferencia, pero ahora los píxeles comunes de $\mathbf{E}$ y $\mathbf{L}$ se extraen de $\mathbf{L}$ en lugar de $\mathbf{E}$. Un valor porcentual pequeño de esta métrica implica que la solución es ajustada, siempre que la métrica de verdaderos positivos tenga un valor alto y la métrica de falsos negativos tenga un valor bajo.

Verdaderos Negativos. En este caso, la métrica denota los píxeles no incluidos en las soluciones de ambos métodos, es decir, los píxeles que no están en $\mathbf{E}$ ni en $\mathbf{L}$. Se calcula mediante la unión de los conjuntos $\mathbf{E}$ y $\mathbf{L}$ y luego el resultado se resta (operación de diferencia) del conjunto universal ($\mathbf{U}$).

E es un subconjunto de L. Esta métrica determina si el conjunto obtenido a partir de la distancia euclídea es un subconjunto del conjunto obtenido por el método propuesto. Indica que la solución contiene todos los píxeles que el algoritmo de referencia.

L es un subconjunto de E. En este caso, denota la falta de píxeles en la solución porque el conjunto obtenido por el algoritmo de la distancia euclídea es mayor que el obtenido por el método del límite.

En la tabla I se indica el porcentaje de similitud entre la distancia euclídea y el método pruopuesto. Para ello, el conjunto obtenido por la distancia

Tabla II: Recursos hardware del algoritmo HyperLCA con función de distorsión adaptativa (Dispositivo: Xilinx ZynQ-7020).

PEs	BRAM18K	DSP48E	FlipFlops	LUTs
1	211 *(75.3 %)*	9 *(4.0 %)*	6,146 *(5.7 %)*	7,502 *(14.1 %)*
2	208 *(74.2 %)*	16 *(7.2 %)*	6,186 *(5.8 %)*	8,329 *(15.6 %)*
4	216 *(77.1 %)*	30 *(13.6 %)*	7,048 *(6.6 %)*	9,384 *(17.6 %)*
6	223 *(79.6 %)*	62 *(28.1 %)*	8,134 *(7.6 %)*	10,886 *(20.4 %)*
10	232 *(82.8 %)*	102 *(46.3 %)*	9,684 *(9.1 %)*	12,733 *(23.9 %)*
12	217 *(77.5 %)*	122 *(55.4 %)*	10,573 *(9.9 %)*	14,458 *(27.1 %)*
20	218 *(77.8 %)*	202 *(91.8 %)*	13,834 *(13.0 %)*	19,178 *(36.0 %)*

Tabla III: MSamples/s alcanzadas por las versiones del acelerador HyperLCA con función de distorsión adaptativa.

PEs	1	2	4	6	10	12	20
100 MHz	58	120	236	332	494	562	777
143 MHz	87	180	352	495	734	835	1149

euclídea debe cumplir que ninguno de los miembros de dicho conjunto supere el valor de 200 con respecto a la distancia a la referencia. Por su parte, el método de límites define los límites superior e inferior en función de los valores espectrales de la firma patrón con una única variable (columna *limites*), también define el número máximo de bandas espectrales cuyos valores quedan fuera de los límites (columna *errores*). Estos parámetros son configurables y, para el análisis actual, los valores seleccionados se enumeran en las dos primeras columnas de la tabla I. Las mejores configuraciones son aquellas con un alto porcentaje de verdaderos positivos y un bajo porcentaje de falsos positivos y negativos, por lo que la mejor configuración es aquella cuyos parámetros *Límites* y *Errores* están configurados con ±25 y 30, respectivamente.

C. Análisis hardware

La arquitectura propuesta se ha implementado en un dispositivo de coste optimizado (FPGA ZC7Z020-CLG484) en comparación con otras arquitecturas del mismo fabricante, como Kintex o UltraScale/UltraScale+. En este sentido, el dispositivo seleccionado ofrece un buen equilibrio en tres aspectos: coste, consumo energético y rendimiento. Sin embargo, para lograr un buen equilibrio de ratio en rendimiento por vatio, es necesario invertir esfuerzos de ingeniería en la parte arquitectónica como consecuencia de los limitados recursos del SoC ZynQ.

La tabla II lista los recursos lógicos programables de acuerdo con el número de elementos de procesamiento (PE) instanciados en cada configuración tras la fase de post-implementación. El tamaño del bloque hiperespectral (BS) se fija en 1024 píxeles hiperespectrales, mientras que el tamaño espacial es de 180 bandas. El número de PEs que se pueden instanciar es de 20 PEs, ya que el número de DSPs (Digital Signal Processing units) disponibles es de 220 en el dispositivo FPGA ZC7Z020-CLG484 y dicha configuración requiere 202 de estas unidades. Así, los $DSPs$ son el recurso limitante de la arquitectura propuesta, ya que la siguiente configuración posible instanciará 30 PEs, por lo que demandará 257 $DSPs$, es decir, se solicita un dispositivo con más recursos hardware. Cabe mencionar que el número de PEs debe ser divisor del número de bandas para aplicar correctamente las optimizaciones. Mientras tanto, el resto de recursos hardware no son críticos para la escalabilidad de la arquitectura, aunque las BRAMs están entre 74 % y 83 %, realmente depende del parámetro de tamaño de bloque (BS), cuyo valor se establece en tiempo de diseño.

El rendimiento de cada configuración del acelerador hardware se representa en MSamples por segundo (MSamples/s), que depende del número de PEs instanciados y de la configuración de la frecuencia de reloj. En este sentido, los modelos RTL para los módulos HLS se generaron fijando la frecuencia de reloj objetivo en 100MHz. A continuación, se sintetizaron dos versiones del flujo de bits para dos configuraciones de reloj diferentes: 100 MHz y 143 MHz. En la tabla III se muestra el promedio de MSamples/s alcanzado por el acelerador hardware en función del número de PEs y la configuración de la frecuencia de reloj. Cabe mencionar que la métrica MSamples/s depende principalmente de dos parámetros de configuración del acelerador hardware: la firma del patrón y las reglas que determinan el ratio de compresión aplicado al bloque hiperespectral actual. Así, los valores de MSamples/s de la tabla III son una media de los resultados obtenidos de las cuatro imágenes sensadas por la plataforma UAV aplicando tres reglas, donde las versiones rápidas (143MHz) consiguen un 33 % mejor rendimiento que las lentas (100MHz).

Aunque las herramientas de análisis de energía de Vivado proporcionan resultados precisos sobre el consumo de energía después de la fase de post-implementación, hay otros componentes de hardware fuera del dispositivo FPGA que están involucrados en el proceso de compresión hiperespectral, como la memoria externa (RAM), que no se tienen en cuenta en los informes de energía. Desafortunadamente, la plataforma ZedBoard no tiene múltiples fuentes de alimentación para obtener información en tiempo real sobre el consumo de energía, por lo que no se puede realizar una medición de grano fino como hace la herramienta de estimación de energía AMD-Xilinx. Por otro lado, la herramienta de gestión de hardware Vivado supervisa la tensión de alimentación de los componentes del sistema de procesamiento (PS) y la lógica programable (PL), pero la frecuencia de actualización es demasiado baja. Por lo tanto, el consumo de energía de la arquitectura presentada en este documento se ha medido en la plataforma ZedBoard a través del puerto de detección de corriente (conector J21), utilizando un multímetro con resolución suficiente para mostrar la diferencia en el consumo de energía cuando se realiza un proceso de compresión [19]. La tabla IV enumera el consumo de potencia en vatios y la eficiencia energética en MMuestras/s por vatio (MSs/W) en las diferentes versiones del HyperLCA con función de distorsión.

La figura 7 muestra gráficamente una comparación del rendimiento y la compensación del consumo de energía entre las versiones de 100MHz (barras azules y línea morada) y 143MHz (barras rojas y línea na-

Tabla IV: Consumo de energía del diseño en ZedBoard medido mediante el conector J21.

	PEs	1	2	4	6	10	12	20
100MHz	Vatios (W)	3.624	3.612	3.636	3.642	3.654	3.660	3.720
	MSs/W	16	33	64	91	135	153	208
143MHz	Vatios (W)	3.660	3.648	3.660	3.696	3.714	3.720	3.768
	MSs/W	23	49	96	133	197	224	304

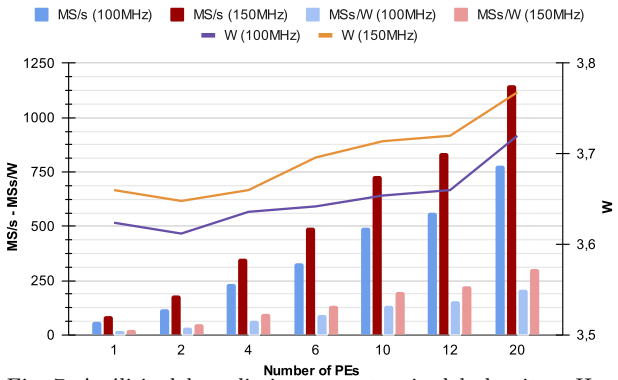


Fig. 7: Análisis del rendimiento y potencia del algoritmo HyperLCA con función de distorsión adaptativa en sus versiones de 100 MHz y 143 MHz a través del conector J21.

ranja) del algoritmo HyperLCA con función de distorsión adaptativa. El consumo de energía tiene un comportamiento similar en ambas versiones, cuyas variaciones dependen del número de PEs, mientras que la eficiencia energética en la versión de 143MHz es un 30 % superior a la versión de 100MHz.

D. Comparativa con otros compresores

La tabla V muestra un resumen detallado del rendimiento alcanzado por las arquitecturas del estado del arte y la presentada en este trabajo, utilizando los valores obtenidos tras la fase de post-implementación para analizar las arquitecturas en el mismo punto. Además, la tabla también muestra en la penúltima columna la potencia en chip representada en vatios. Lamentablemente, algunas propuestas del estado del arte no proporcionan dicha información, por lo que se ha estimado con la herramienta Xilinx Power Estimator (XPE). El uso de recursos embebidos especializados de la FPGA no tiene un gran impacto en el consumo de potencia, pero depende del número de regiones de reloj que estén activas, la cantidad de recursos y la tecnología FPGA utilizada. En este sentido, la arquitectura presentada por D. Keymulen en [22] requiere más potencia que la presentada por D. Báscones et al. en [25], donde ambas propuestas utilizan el mismo dispositivo FPGA y VHDL para describir la arquitectura; la diferencia radica en el número de recursos FPGA utilizados para cada propuesta. Podemos concluir que la arquitectura presentada en este trabajo supera al resto de propuestas del estado del arte en el balance MB/s por vatio (MBS/W), donde la versión más rápida del compresor HyperLCA con función de distorsión, es decir, la versión que contiene 20 PEs trabajando en paralelo, es entre 4,9× y 5,8× mejor que la arquitectura presentada por D. Báscones et al. en [25], cuando la frecuencia de reloj se configura a 100MHz y 143MHz, respectivamente.

Cabe mencionar que la implementación anterior del compresor HyperLCA es entre 3,6× y 4,1× más lenta que la versión con función de distorsión adaptativa. Además, la configuración de la frecuencia de reloj tiene poca influencia en la tasa de MB/s por vatio, se incrementa 54,66 MBS/W cuando la frecuencia de reloj se establece en 143MHz. Por tanto, la arquitectura presentada en este trabajo es capaz de comprimir la citada imagen hiperespectral de 1024x1024 de tamaño espacial y 180 bandas espectrales en 0,935s con un consumo de 1,145W.

V. Conclusiones

En este trabajo se ha abordado la inclusión de la característica de distorsión adaptativa en el algoritmo HyperLCA con el fin de aumentar la información espectral en aquellas regiones de interés, cuya información espectral contenga una cantidad de píxeles cercana a una firma de patrón hiperespectral predefinida. De este modo, la propuesta puede adaptarse al escenario que se esté procesando, por ejemplo, podría recogerse más información espectral de la vegetación que del suelo en un viñedo. Además, el compresor puede configurarse para aumentar la información en las líneas que contienen píxeles anómalos, por ejemplo, barcos en medio del mar.

El conjunto de operaciones hiperespectrales centrales se hereda de trabajos anteriores [16], [14] donde se comprobó la idoneidad de la tecnología FPGA para este tipo de aplicaciones, por lo que se ahorra una cantidad significativa de tiempo. Sin embargo, la nueva arquitectura incluye nuevas optimizaciones que permiten instanciar más PEs trabajando en paralelo, que a su ver permite aumentar el rendimiento, mientras que la función de distorsión adaptativa introduce una pequeña sobrecarga en términos de tamaño debido a la necesidad de incluir una cabecera de 8 bits por cada muestra que se comprime. Además, la comparación con otras arquitecturas basadas en FPGA revela que la arquitectura propuesta es una solución eficiente en coste-energía sin reducir la calidad de compresión, cuya pérdida de información espectral puede reducirse incrementando el parámetro p_{max}, con lo que se consigue una mejor calidad de compresión.

Agradecimientos

Esta investigación está parcialmente financiada por el Ministerio de Economía y Competitividad (MINECO) del Gobierno de España a través de los proyectos TALENT (PID2020-116417RB-C4, subproyectos 1 y 4) y MIRATAR (TED2021-132149B-C41) y por el programa Europeo Horizonte 2020 bajo el proyecto SHAPES (GA Nº 857159).

Referencias

[1] Antonio Plaza, Jon Atli Benediktsson, Joseph W Boardman, Jason Brazile, Lorenzo Bruzzone, Gustavo Camps-Valls, Jocelyn Chanussot, Mathieu Fauvel, Paolo Gamba,

Tabla V: Comparación de rendimiento con otras implementaciones de compresores hiperespectrales basadas en FPGA.

Propuesta	Imagen Hiperespectral	Muestras	Lineas	Bandas	Rendimiento (MSamples/s)	MB/s	Vatios (W)	MBS/W
L. Santos et al.[20]	-	16	16	256	30.25	0.236	2.029*	0.116
A. García et al.[21]	AVIRIS (Jasper Ridge)	614	512	224	27.7	5.449	2.022*	2.695
D. Keymulen (1 core)[22]	AVIRIS (Jasper Ridge)	614	512	224	9	4.722	7.84	0.602
D. Keymulen (15 cores)[22]	AVIRIS (Jasper Ridge)	614	512	224	95	49.842	11.4	4.372
D. Fernández et al.[23]	AVIRIS (Jasper Ridge)	614	512	224	80.29	9.004	2.46*	3.661
D. Báscones et al.[24]	AVIRIS	512	512	256	119.96	29.99	2.732	10.977
D. Báscones et al.[25]	AVIRIS	512	512	256	162.3	40.575	0.714	56.828
Y. Barrios et al. (1 core) [26]	AVIRIS	512	512	256	0.4	0.1	0.6*	0.167
Y. Barrios et al. (8 cores)[26]	AVIRIS	512	512	256	1.7	0.425	0.9*	0.472
J. Caba et al. (12 PEs) [16]	Own *(Specim FX10)*	1024	1024	180	342	120.234	1.6	75.146
J. Caba et al. (12 PEs) [16]	Own *(Specim FX10)*	1024	1024	180	511	179.648	2.22	80.923
Our (20 PEs - 100MHz)	Own *(Specim FX10)*	1024	1024	180	777	273.164	0.993	275.089
Our (20 PEs - 143MHz)	Own *(Specim FX10)*	1024	1024	180	1,149	403.945	1.225	329.751

*Obtenido mediante la herramienta Xilinx Power Estimator (XPE).

Anthony Gualtieri, et al., "Recent advances in techniques for hyperspectral image processing," *Remote sensing of environment*, vol. 113, pp. S110–S122, 2009.

[2] Amal Altamimi and Belgacem Ben Youssef, "A systematic review of hardware-accelerated compression of remotely sensed hyperspectral images," *Sensors*, vol. 22, no. 1, 2022.

[3] Miloš Radosavljević, Branko Brkljač, Predrag Lugonja, Vladimir Crnojević, Željen Trpovski, Zixiang Xiong, and Dejan Vukobratović, "Lossy compression of multispectral satellite images with application to crop thematic mapping: A hevc comparative study," *Remote Sensing*, vol. 12, no. 10, pp. 1590, 2020.

[4] Fred Ortenberg, PS Thenkabail, JG Lyon, and A Huete, "Hyperspectral sensor characteristics: airborne, spaceborne, hand-held, and truck-mounted; integration of hyperspectral data with lidar," *Hyperspectral Remote sensing of vegetation*, pp. 39–68, 2011.

[5] José M. Melián, Adán Jiménez, María Díaz, Alejandro Morales, Pablo Horstrand, Raúl Guerra, Sebastián López, and José F. López, "Real-Time Hyperspectral Data Transmission for UAV-Based Acquisition Platforms," *Remote Sensing*, vol. 13, no. 5, 2021.

[6] Elodie Morin, Mickael Maman, Roberto Guizzetti, and Andrzej Duda, "Comparison of the Device Lifetime in Wireless Networks for the Internet of Things," *IEEE Access*, vol. 5, pp. 7097–7114, 2017.

[7] Bormin Huang, *Satellite data compression*, Springer Science & Business Media, 2011.

[8] Yaman Dua, Vinod Kumar, and Ravi Shankar Singh, "Comprehensive review of hyperspectral image compression algorithms," *Optical Engineering*, vol. 59, no. 9, pp. 1 – 39, 2020.

[9] Aaron B Kiely, Matthew Klimesh, Ian Blanes, Jonathan Ligo, Enrico Magli, Nazeeh Aranki, Michael Burl, Roberto Camarero, Michael Cheng, Sam Dolinar, et al., "The new ccsds standard for low-complexity lossless and near-lossless multispectral and hyperspectral image compression," 2018.

[10] Estanislau Augé, Jose Enrique Sánchez, Aaron B Kiely, Ian Blanes, and Joan Serra-Sagrista, "Performance impact of parameter tuning on the CCSDS-123 lossless multi-and hyperspectral image compression standard," *Journal of Applied Remote Sensing*, vol. 7, no. 1, pp. 074594, 2013.

[11] Lucana Santos, Enrico Magli, Raffaele Vitulli, José F López, and Roberto Sarmiento, "Highly-parallel GPU architecture for lossy hyperspectral image compression," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 6, no. 2, pp. 670–681, 2013.

[12] Yubal Barrios, Antonio J Sánchez, Lucana Santos, and Roberto Sarmiento, "SHyLoC 2.0: A Versatile Hardware Solution for On-Board Data and Hyperspectral Image Compression on Future Space Missions," *Ieee Access*, vol. 8, pp. 54269–54287, 2020.

[13] Raúl Guerra, Yubal Barrios, María Díaz, Lucana Santos, Sebastián López, and Roberto Sarmiento, "A new algorithm for the on-board compression of hyperspectral images," *Remote Sensing*, vol. 10, no. 3, pp. 428, 2018.

[14] Raúl Guerra, Yubal Barrios, María Díaz, Abelardo Baez, Sebastián López, and Roberto Sarmiento, "A hardware-friendly hyperspectral lossy compressor for next-generation space-grade field programmable gate arrays," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 12, pp. 4813–4828, 2019.

[15] María Díaz, Raúl Guerra, Pablo Horstrand, Ernestina Martel, Sebastián López, José F. López, and Sarmiento Roberto, "Real-Time Hyperspectral Image Compression Onto Embedded GPUs," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, pp. 1–18, 2019.

[16] Julián Caba, María Díaz, Jesús Barba, Raúl Guerra, and Jose A. de la Torre and Sebastián López, "FPGA-Based On-Board Hyperspectral Imaging Compression: Benchmarking Performance and Energy Efficiency against GPU Implementations," *Remote Sensing*, vol. 12, no. 22, 2020.

[17] Paul G Howard and Jeffrey Scott Vitter, "Fast and efficient lossless image compression," in *Data Compression Conference, 1993. DCC'93.* IEEE, 1993, pp. 351–360.

[18] Pablo Horstrand, Raúl Guerra, Aythami Rodríguez, María Díaz, Sebastián López, and José Fco López, "A UAV platform based on a hyperspectral sensor for image capturing and on-board processing," *IEEE Access*, vol. 7, pp. 66919–66938, 2019.

[19] Velleman, "User Manual: DVM1200 - Multimeter with USB interface," Available Online: `https://www.velleman.eu/downloads/1/dvm1200gbnlfresdplit.pdf`, (Accessed on 26 May 2023).

[20] Lucana Santos, José Fco. López, Roberto Sarmiento, and Raffaele Vitulli, "FPGA implementation of a lossy compression algorithm for hyperspectral images with a high-level synthesis tool," in *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)*, 2013, pp. 107–114.

[21] Aday García, Lucana Santos, Sebastián López, Gustavo Marrero, José Fco. López, and Roberto Sarmiento, "High level modular implementation of a lossy hyperspectral image compression algorithm on a FPGA," in *2013 5th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, 2013, pp. 1–4.

[22] Didier Keymeulen, "FPGA implementation of lossless and lossy compression of space-based multispectral and hyperspectral imagery," 2016.

[23] Daniel Fernández, Carlos González, Daniel Mozos, and Sebastián Lopez, "FPGA implementation of the principal component analysis algorithm for dimensionality reduction of hyperspectral images," *Journal of Real-Time Image Processing*, vol. 16, 2019.

[24] Daniel Báscones, Carlos González, and Daniel Mozos, "An Extremely Pipelined FPGA Implementation of a Lossy Hyperspectral Image Compression Algorithm," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 10, pp. 7435–7447, 2020.

[25] Daniel Báscones, Carlos González, and Daniel Mozos, "An FPGA Accelerator for Real-Time Lossy Compression of Hyperspectral Images," *Remote Sensing*, vol. 12, no. 16, 2020.

[26] Yubal Barrios, Alfonso Rodríguez, Antonio Sánchez, Arturo Pérez, Sebastián López, Andrés Otero, Eduardo de la Torre, and Roberto Sarmiento, "Lossy Hyperspectral Image Compression on a Reconfigurable and Fault-Tolerant FPGA-Based Adaptive Computing Platform," *Electronics*, vol. 9, no. 10, 2020.

Motor de convolución pulsante para redes neuronales de convolución pulsantes

Dagnier A. Curra-Sosa[1], Ricardo Tapiador-Morales[2], Francisco Gómez-Rodríguez, Alejandro Linares Barranco[3]

Resumen— **Una alternativa prometedora en tareas de visión artificial y que reduce considerablemente el costo computacional, es el procesamiento neuromórfico basado en eventos. Los sensores visuales neuromórficos, emulan la retina humana y generan pulsos de señales asíncronos y binarios en respuesta a cambios de iluminación, generando una cantidad mínima de información. En este contexto, utilizamos un sistema multiconvolución sobre una FPGA[1] basado en eventos, inspirado en la neurona Leaky Integrate-and-Fire (LIF) para demostrar su viabilidad en la implementación de una red neuronal convolucional pulsante (SCNN). La principal innovación de este sistema es la combinación de un arbitrador de memoria para un acceso eficiente a la memoria, lo que permite el procesamiento fila por fila de los filtros de convolución. Con esta técnica, es posible reducir significativamente el número de operaciones necesarias para procesar un flujo de eventos visuales, lo que hace que el sistema sea más eficiente en términos de recursos computacionales. En este trabajo demostramos que las capas de convolución de la red LeNet-5, entrenada con MNIST, puede implementarse de forma pulsante y discutimos sobre las modificaciones necesarias en la arquitectura para ofrecer esta solución en tiempo real.**

Palabras clave— **Visión Computacional, Redes Neuronales Convolucionales, Ingeniería Neuromórfica, Sensor de Visión Dinámica, FPGA, Representación basada en eventos**

I. Introducción

Las CNN son ampliamente utilizadas en tareas de visión artificial por su eficacia en la extracción de características de una escena [1]. Sin embargo, el entrenamiento de estas redes requiere de aceleradores de hardware costosos como las unidades gráficas de procesamiento o servidores de gama alta, debido al gran número de operaciones que requieren las convoluciones basadas en fotogramas, en el orden de los billones por cada imagen. A pesar de que existen implementaciones ASIC[2] de aceleradores de CNN que reducen el consumo de energía, el número de operaciones sigue siendo elevado. Sin embargo, existen técnicas neuromórficas basadas en eventos que pueden reducir el número de operaciones y el consumo de energía.

La ingeniería neuromórfica es un campo prometedor que busca emular el funcionamiento del cerebro humano mediante el diseño y desarrollo de sistemas de hardware y software inspirados en la neurobiología. Esta inspiración se basa en la analogía entre el comportamiento de los transistores polarizados en la región sub-umbral y la física de las neuronas biológicas [2], por lo cual la información se codifica en pulsos (eventos) que son procesados en paralelo por masivas capas de neuronas interconectadas mediante sinapsis [3].

Las cámaras basadas en eventos, también conocidas como retinas de silicio, se diferencian de las cámaras convencionales en que requieren una alta resolución temporal, del orden de cientos de nanosegundos, y un consumo de energía extremadamente bajo. Estas cámaras ofrecen el potencial de combinar redes neuronales y sensores de visión basados en eventos para lograr redes ópticas altamente eficientes y de gran ancho de banda [4]. En este sentido, se han desarrollado diversos sensores de visión dinámica (DVS) [5] y cócleas artificiales [6].

El DVS captura una escena de manera visual, donde cada píxel funciona como una neurona generando un flujo de eventos en función de los cambios de luminosidad. A diferencia de las cámaras convencionales basadas en fotogramas, que registra todos los valores de píxel de una escena, incluso aquellos que no han cambiado; el flujo de eventos del DVS representa solo la realidad en movimiento, evitando cargar los píxeles estáticos de una imagen. Esta característica reduce significativamente la cantidad total de píxeles que se deben procesar; pues, tan pronto como un píxel experimenta un cambio, se genera un evento y se envía desde el sensor. Esto hace que el procesamiento basado en eventos sea asíncrono y continuo, sin la necesidad de seguir un ritmo de muestreo preestablecido [7].

El creciente interés en los DVS ha impulsado el desarrollo de las SCNN. Las redes neuronales convolucionales tradicionales no pueden aprovechar plenamente las ventajas de las entradas provenientes de cámaras DVS, ya que las neuronas ReLU[3] no pueden capturar información temporal. Sin embargo, las SCNN son ideales para procesar datos provenientes de estos sensores, gracias a su capacidad para manejar la dinámica temporal de las neuronas. La combinación de entradas de cámara DVS con las SCNN permite lograr una eficiencia energética significativa y un mejor rendimiento en tareas de clasificación de imágenes [8].

Estas técnicas basadas en eventos tiene como ventaja principal la pseudo-simultaneidad [9], que permite iniciar el procesamiento con la llegada del primer evento desde el sensor, y por tanto, obtener una

[1]Dpto. de Arquitectura y Tecnología de Computadores, EPS-ETSII, Universidad de Sevilla, e-mail: dcurra@us.es
[2]Sony, Zurich, Suiza, e-mail: ricardotapiador@gmail.com
[3]Grupo de Ingeniería Neuromórfica, Unidad de excelencia SCORE, Universidad de Sevilla.

[1]Field-Programmable Gate Array
[2]Application-Specific Integrated Circuit
[3]Rectified Lineal Unit

salida mientras el sensor continúa produciendo eventos. La principal desventaja de las SCNNs es la memoria RAM pues, mientras que una CNN solo la necesita para los cálculos de una capa, la SCNN requiere almacenar el estado del potencial de membrana de toda la red para actualizarlo cuando llega un evento.

En el artículo, se parte de un convolucionador basado en eventos sobre el modelo neuronal LIF. El mismo aplica filtros sobre las neuronas, almacena el potencial de membrana en memoria empotrada y genera eventos cuando se alcanzan los umbrales establecidos. Su diseño incluye propiedades de las neuronas LIF, como la pérdida y el período refractario. Su aporte principal consiste en la lectura/escritura fila por fila de la memoria, que reduce la latencia en el procesamiento de un evento de entrada. El sistema, implementado en una FPGA (Xilinx Zynq), se ha probado con la aplicación de filtros de convolución de una SCNN, visualizando y estudiando la salida.

La estructura de este artículo se organiza del modo siguiente: la Sección II presenta el estado del arte de la temática que se aborda, mientras que la Sección III explica los métodos utilizados sobre los cuales se apoya el funcionamiento del hardware. La Sección IV describe el caso de estudio analizado en la experimentación y los resultados derivados de la misma. Finalmente, la Sección V presenta las conclusiones y las líneas de trabajo futuro.

II. Estado del Arte

Del mismo modo que en el ámbito de los aceleradores de chip basados en fotogramas, se han creado numerosas soluciones para FPGA y ASIC. Linares-Barranco et al. [10] presentan dos implementaciones FPGA de convolucionadores basados en el protocolo para la representación de direcciones de eventos (AER) en FPGA Xilinx relativamente pequeñas (Spartan-II 200 y Spartan3-400), que procesan imágenes de 64×64 con filtros de convolución de 11×11, cuyos rendimientos se comparan con procesadores de convolución basados en fotogramas. Camuñas-Mesa et al. [11] implementa un módulo de convolución multinúcleo controlado por eventos para calcular convoluciones 2D en flujos de direcciones de eventos sin restricciones de fotogramas, el cual se combina con un DVS para reconocimiento de alta velocidad. Además, Zamarreño-Ramos et al. [12] presenta un enfoque modular y escalable para ensamblar sistemas AER neuromórficos estructurados jerárquicamente, donde se analiza el rendimiento de dos modos de enrutamiento de eventos sobre una FPGA Virtex-6 que procesa datos sensoriales reales de una retina DVS.

Un desafío común en los aceleradores de convolución es el acceso a la memoria para los valores de los filtros, lo cual puede aumentar la latencia y generar cuellos de botella. No obstante, es posible mejorar esta situación al modificar la forma en que se accede a los datos. Por ejemplo, el acceso por filas ha sido implementado antes en chips [13][14] para la realización de convoluciones basadas en eventos en tiempo real, pero no en antiguas FPGA debido a que sus recursos de memoria eran insuficientes para implementar de modo eficiente varios convolucionadores independientes en el procesamiento pixel a pixel.

En la presente decada se ha intensificado el diseño de aceleradores multinúcleos basados en eventos para implementaciones de redes neuronales pulsantes en hardware empotrado. Fang et al. [15] propone un método para acelerar la codificación de pulsos en FPGA y una implementación de una red neuronal pulsante (SNN) de bajo consumo en un sistema en tiempo real para el reconocimiento digital utilizando múltiples núcleos en paralelo en el chip Xilinx XCZU9EG. Irmak et al. [16] presenta una arquitectura reconfigurable dinámica para aceleradores de CNN y SNN en FPGA, con el propósito de optimizar el uso y rendimiento de los recursos de hardware mediante combinaciones de arquitecturas híbridas de redes neuronales en la clasificación de imágenes. Veeravalli y Fong [17] han desarrollado un coprocesador neuromórfico basado en FPGA, capaz de emular SNN con aprendizaje basado en STDP para tareas de reconocimiento de objetos.

En cuanto al diseño e implementación de arquitecturas de procesadores de convolución pulsantes, también se tienen resultados significativos. Zhang et al. [18] presenta una arquitectura de convolución pulsante multinúcleo para procesar flujos AER en chips de DVS. Su diseño incluye el modelo neuronal LIF y el procesamiento fila por fila de los 64 filtros que procesa, de tamaño variable entre 1x1 y 32x32, en un prototipo FPGA de una placa Xilinx Zynq. Huang et al. [19] detalla la metodología de implementación de SCNN en FPGA para la identificación de radioisótopos mediante datos de alta resolución sobre la plataforma neuromórfica SpiNNaker. Zhang et al. [20] propone una arquitectura de convolución pulsante multinúcleo y multicapa configurable implementada en una Kintex-7 FPGA. Esta admite múltiples esquemas de codificación en adaptación a diferentes modelos SCNN para los cuales obtiene buenos resultados en la clasificación y un bajo consumo energético. Feng et al. [21] presentan un eficiente procesador multicapa de SCNN para el reconocimiento de objetos con baja precisión de bits y paralelismo a nivel de canal, el cual tiene la primicia de implementar la interconexión entre capas pulsantes.

Estas investigaciones representan avances significativos en el diseño y la implementación de SNN en FPGA, abriendo nuevas posibilidades para aplicaciones de procesamiento de información en tiempo real y eficientes en energía.

III. El acelerador de SCNNs

Los modelos de CNN suelen estar formados por tres capas: (1) una capa de convolución, en la que se convolucionan las imágenes; (2) una capa de agrupación, en la que se submuestrea una imagen para reducir su tamaño con el fin de disminuir el cómputo en capas futuras, y (3) una capa de no linealidad, por ejemplo ReLU [22].

En el procesamiento de imágenes basado en fotogramas, la operación de convolución consiste en aplicar una matriz de filtro a un valor de píxel, multiplicar los valores del filtro por la intensidad en la escala de los píxeles correspondientes y sumar estos resultados como un único valor. La convolución se realiza deslizando el filtro sobre todos los píxeles de la imagen, empezando normalmente por la esquina superior izquierda. En la convolución, se suele utilizar la técnica de relleno cero para procesar los píxeles de los límites; otras implementaciones de la convolución simplifican el cálculo saltándose los límites. Matemáticamente, se define como se muestra en la Ecuación 1, donde K es el filtro de la convolución NxM, X es la imagen de entrada e Y es la imagen convolucionada.

$$Y\left(i,j\right) = \sum_{a=-\frac{N}{2}}^{\frac{N}{2}} \sum_{b=-\frac{M}{2}}^{\frac{M}{2}} K\left(a,b\right) \cdot X\left(a+i, b+j\right) \quad (1)$$

$Y(i,j)$ se define por el píxel de entrada correspondiente $X(i,j)$ y los píxeles adyacentes ponderados, escalados por valores K [23].

Sin embargo, en un sistema de procesamiento basado en eventos, no siempre se procesan todos los píxeles porque los sensores neuromórficos obtienen los cambios de luminosidad de la escena real, y estos cambios se transmiten como eventos. Un evento se representa con una tupla (x, y, p) que corresponde a la dirección del píxel (x, y) de una imagen que está cambiando, y un bit de polaridad (p) que indica si el píxel está ON u OFF. La polaridad ON significa que la luminosidad detectada por el píxel en el estado actual es mayor que la luminosidad detectada en el estado anterior, mientras que la polaridad OFF significa lo contrario: la luminosidad detectada en el estado actual es menor que la luminosidad detectada en el estado anterior [24].

En una convolución pulsante una imagen de entrada X se codifica de tal manera que cada píxel $X(i,j)$ se representa por un número de eventos de una fuente visual neuromórfica (`DVS` retina). Los resultados de las operaciones de convolución se almacenan en una matriz Y (condensadores para los circuitos analógicos o registros o celdas `RAM` para los circuitos digitales). Cuando llega un evento de entrada, el píxel correspondiente y sus vecinos se modifican en Y, añadiendo el filtro de convolución. La siguiente Ecuación (2) muestra la operación para computar cada evento entrante con dirección (i,j):

$$\begin{aligned} &Y\left(i+a, j+b\right) = Y\left(i+a, j+b\right) + K\left(a,b\right); \forall a, b \\ &a \,\epsilon \left[-\frac{N}{2}, \frac{N}{2}\right], b \,\epsilon \left[-\frac{M}{2}, \frac{M}{2}\right]; N, M = dim\left(K\right) \end{aligned} \quad (2)$$

Una vez recibidos y calculados todos los eventos del píxel $X(i,j)$, el valor integrador de la dirección correspondiente $Y(i,j)$ acumula $X(i+a, j+b)\forall(a,b)$, veces el valor del kernel, como establece la ecuación (1). En otras palabras, estamos añadiendo el valor del filtro a un vecino de salidas tantas veces como número de eventos de entrada. Esta adición continua es equivalente a multiplicar la intensidad de un píxel por un valor de filtro en convoluciones basadas en fotogramas. La salida de la operación de convolución, en este punto, se almacena en una matriz de integradores Y que puede enviarse de varias maneras. En este trabajo, se inspira en el modelo de neuronas `LIF` [25], donde la suma continua de los valores del filtro sobre una neurona, aumenta o disminuye su potencial de membrana en función de coeficientes positivos o negativos, respectivamente. Cuando el potencial de membrana de una neurona alcanza un umbral positivo (**PTH**), se genera un pulso con polaridad positiva en la neurona de dirección (x, y), reseteando su potencial de membrana, como se muestra en la Fig 1.

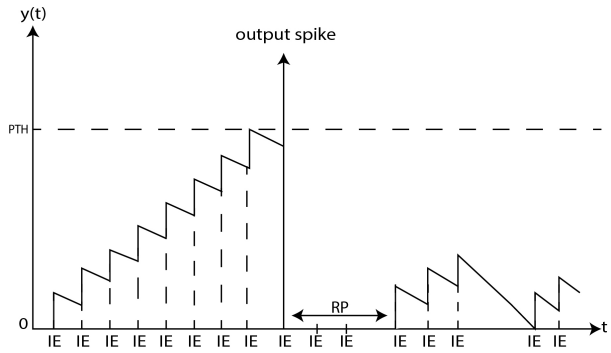


Fig. 1: Cambios del potencial de membrana de una neurona en el tiempo por eventos de entrada (IE).

Una neurona biológica disminuye su potencial de membrana por pérdida cuando no recibe ninguna excitación. Hemos imitado esta propiedad de las neuronas biológicas, porque una neurona `LIF` que no ha recibido ninguna excitación significa que no está dando información sobre la escena. El tiempo de decaimiento por pérdida de una neurona y el valor en que la neurona disminuye su potencial son parámetros que permiten controlar la tasa de eventos de salida. Esto implica que, con un tiempo de decaimiento pequeño, las neuronas se reiniciarían más a menudo. Por otro lado, un tiempo de decaimiento más largo hace que las neuronas restablezcan su potencial raramente, aumentando el número de pulsos de salida.

Aunque la pérdida disminuye la tasa de eventos en la salida, la tasa de eventos sería mayor al implementar `SCNN`, ya que una capa comprende varias convoluciones en paralelo. Una solución para estabilizar la tasa de salida es implementar el periodo refractario (**RP**) de la neurona `LIF`. Así, si una neurona se dispara y genera un pulso, debe esperar un periodo de tiempo antes de recibir cualquier tipo de excitación.

En las `CNN`, una capa de convolución suele ir seguida de una capa de agrupación [26]. Para `CNN` basadas en fotogramas, existen varios tipos de agrupamientos (pooling); por ejemplo, el max-pooling aplica el filtro de máximos a la imagen de entrada, filtrando

los píxeles con el valor máximo. En SCNN, el submuestreo suele consistir en dividir por 2 la dirección (x, y) del evento de salida de una etapa de convolución, reduciendo el tamaño de la imagen (espacio de direcciones) [27]. En hardware, este paso es fácil de implementar con una operación de desplazamiento a la dirección x,y una posición a la derecha, como se muestra en la Fig 2.

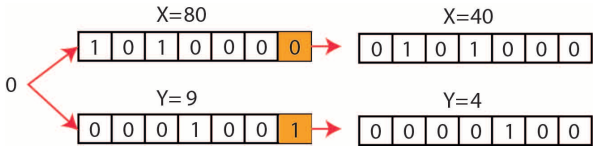


Fig. 2: Ejemplo de submuestreo de evento. Los valores de la dirección se dividen por 2 con el desplazamiento a la derecha

A. Arquitectura

El acelerador de SCNN utilizado en este trabajo [28] es un sistema de convolución digital totalmente programable inspirado en el funcionamiento de la neurona LIF. La arquitectura tiene tres interfaces: dos interfaces de tipo **AER** para el protocolo asíncrono de handshaking de cuatro pasos, utilizado para enviar y recibir señales entre sistemas neuromórficos; además de una interfaz digital de 32 bits utilizada por un microcontrolador empotrado para configurar el acelerador. El sistema es capaz de computar un número máximo de 64 operaciones de convolución en paralelo con tamaños de kernel de 1x1 a 7x7. Estas convoluciones se calculan fila a fila de forma totalmente aleatoria, donde se calcula una fila completa por ciclo de reloj. El convolucionador es capaz de realizar la operación de agrupación, disminuyendo el tamaño espacial de la representación para reducir el cómputo en una SCNN.

Para permitir el procesamiento de una fila completa, cuando llega un evento de entrada, cada convolucionador lee de la memoria el potencial de membrana, las marcas de tiempo refractario y las marcas de tiempo de pérdida para toda la fila. Esos valores corresponden a neuronas alrededor de la dirección del evento recibido y al tamaño del filtro programado. El convolucionador compara las marcas de tiempo con dos contadores globales, uno para el periodo refractario y otro para la pérdida, con el fin de verificar si se ha cumplido el periodo refractario y si debe aplicarse la pérdida. A continuación, el convolucionador opera una fila de potencial de membrana con una fila de filtro. Aquellas neuronas que pueden disparar y alcanzan su umbral, producen un evento con dirección (x, y). Este proceso se repite hasta que todas las filas son convolucionadas por todas las filas del filtro.

B. Estructura de almacenamiento

Para este acelerador, la BRAM[4] se divide en cuatro bancos de memoria diferentes: potencial de membrana (**MP**), marcas de tiempo de pérdida (**LT**), marcas de tiempo refractario (**RT**) y valores del filtro (**KV**).

Los bancos de memoria para MP, LT y RT están organizados en múltiples bloques que almacenan valores en filas de 8 píxeles. Cuando un convolucionador accede a los datos de memoria, transmite tres direcciones x,y de la fila correspondiente a convolucionar y el ID de convolución (**CID**). La dirección **X** selecciona los bloques de memoria a los que se va a acceder, utilizando un decodificador para habilitar o deshabilitar los bancos.

Las memorias almacenan valores para un tamaño máximo de imagen de 128x128 para todos los convolucionadores, y dado que los bancos de memoria son compartidos por estos, el CID especifica la región de memoria para el convolucionador correspondiente y la dirección **Y** selecciona qué fila de píxeles se lee/escribe. Cada fila de memoria BRAM almacena 8 píxeles con una resolución de 8 bits para MP y 7 bits para marcas de tiempo; por lo tanto, para 128 filas de píxeles se necesitan 16 bloques BRAM. La profundidad de esta memoria está relacionada con el número de convolucionadores (**N**) multiplicado por el número de filas de una imagen, que es 128.

La Fig 3 muestra un ejemplo de cómo se leen/escriben MP, RT y LT. Durante el procesamiento de un evento, el convolucionador siempre lee una fila de dos BRAM consecutivas, aquella a la que pertenece el evento de entrada y la fila vecina. La razón de esta lectura múltiple es que los píxeles vecinos del evento de entrada pueden estar implicados en la operación de convolución, pero pueden estar almacenados en un banco diferente.

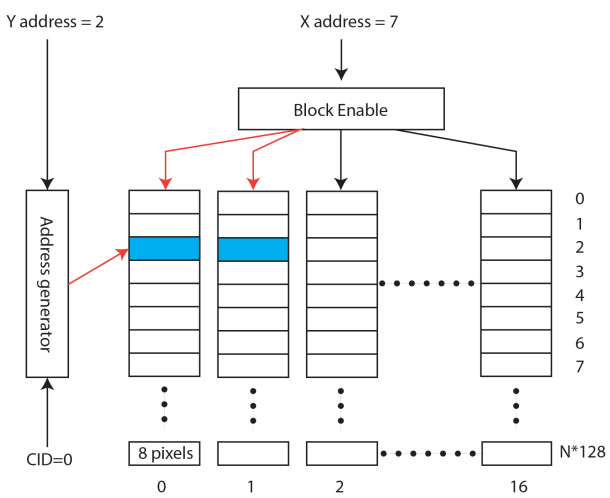


Fig. 3: Ejemplo de acceso a la BRAM para un evento de entrada con dirección x,y 7,2, respectivamente. El píxel con dirección x 7 está en el bloque 0; este bloque y su bloque vecino están habilitados para lectura/escritura. La dirección Y y el CID generan la dirección de memoria, que en ese caso es la dirección 2 del motor de convolución 0.

Siguiendo el mismo concepto, la memoria del filtro se organiza en un bloque compartido, donde las KV se almacenan fila por fila.

[4]Block Random Access Memory

C. Sistema de pérdida

La pérdida de una neurona se aplica durante la convolución, disminuyendo el valor del potencial de membrana de la neurona por un valor de decaimiento configurable (DV). La pérdida se implementó utilizando un contador de 32 bits y debido a que almacenar el contenido de este contador para cada neurona necesitaría una gran cantidad de recursos, solo se almacenan 7 bits de ellos en BRAM. Estos 7 bits se seleccionan mediante una ventana deslizante. que configura la resolución de temporización en función de la tasa de eventos de entrada. Durante la convolución, cuando una fila de entrada va a ser procesada, todos los LT de las neuronas involucradas son comparados con el valor actual del contador de pérdoda (LC), para luego ser actualizados con este. Así, una vez que la diferencia entre LT y LC es mayor que un tiempo configurable para la pérdida, se aplica DV durante la convolución.

D. Sistema de período refractario

El período refractario es una propiedad de las neuronas biológicas que garantiza un tiempo de separación entre dos pulsos generados por la misma neurona. El período refractario utiliza otro contador con el mismo mecanismo de ventana deslizante que el contador de la pérdida. Durante la convolución, RT se compara con el tiempo del contador refractario (RCT). Si RT es mayor que RCT, se ha cumplido el período refractario y la neurona puede disparar si el umbral es alcanzado. De lo contrario, esta neurona debe esperar hasta que se cumpla dicho período. Además, si durante la convolución, una neurona dispara, entonces actualiza su RT por el resultado de la suma del RCT y un periodo refractario programable (RPV). La marca de tiempo resultante indica la próxima vez esa neurona podrá disparar. De lo contrario, su RT es 0 y la neurona puede disparar la próxima vez que se acceda a ella.

Los bancos de memoria del período refractario y la pérdida se encuentran en cada convolucionador y son leidos/escritos fila por fila. Cuando LC produce un desborde, los convolucionadores no pueden actualizar MP antes de LT puesto que el potencial de membrana puede ser reiniciado. Sin embargo, los convolucionadores pueden leer de la memoria durante la actualización, reduciendo así los tiempos de espera.

E. Convolucionador

El módulo del convolucionador consiste en una máquina de estados que se comunica con los bancos de memoria mediante las direcciones de los eventos. Para ello, solicita permisos de acceso de lecura y escritura al arbitrador de memoria, para procesar fila por fila los valores del potencial de membrana y los filtros hasta que este último se aplique completamente, reduciendo de modo considerable la cantidad de accesos a memoria.

Aunque la operación de convolución de toda una fila puede realizarse en un ciclo de reloj, se necesitan muchos recursos puesto que varias operaciones son requeridas, como el chequeo de las marcas de tiempo refractario y de pérdida. Así, para reducir estos recursos, la operación de convolución se divide en dos fases por cada fila: aplicar máscaras y realizar los cálculos.

En el paso de aplicar máscaras, se verifican las marcas de tiempo para aplicar la pérdida (o decaimiento) y se determina si se ha cumplido el periodo refractario.

Inspirado en cómo los procesadores SIMD[5] trabajan sobre matrices de datos [29][30], el paso de aplicar máscaras genera dos máscaras binarias, una para la pérdida y la otra para el período refractario. La máscara de pérdida (**LM**) indica a cada neurona si debe aplicarse decaimiento durante la convolución (1 lógico) o no (0 lógico). Por otro lado, la máscara refractaria (**RM**) indica si el periodo refractario de una neurona se ha cumplido o no.

El **paso de convolución** actualiza las marcas de tiempo, las memorias de pérdida y las memorias de periodo refractario. Antes de la convolución, se comprueban los valores de fila correspondientes de la memoria de pérdidas, como se ha mencionado anteriormente. Si el valor de pérdida se ha desbordado, se restablece el potencial de membrana, de lo contrario se realiza la operación de convolución para esa neurona.

Durante la operación de convolución, la máscara de pérdida (**LM**) se multiplica por DV para aplicar la pérdida a las neuronas correspondientes. Por lo tanto, la operación de convolución consiste en sumar MP a KV y restar DV, si debe aplicarse. Sin embargo, la neurona LIF no puede dispararse si no se cumple el periodo refractario; por lo tanto, el resultado de la operación de convolución se multiplica por la máscara de periodo refractario. La máscara de periodo refractario es una máscara binaria, lo que implica que, si no se cumple el periodo refractario, el resultado de la operación de convolución es 0, ya que la neurona no puede recibir ningún tipo de excitación. En la Fig. 4 se muestra cómo se realiza la operación de convolución con las máscara de pérdida y la refractaria.

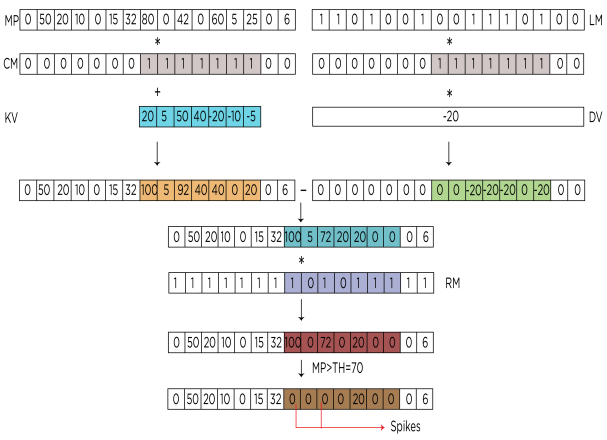


Fig. 4: Estructura de memoria de los filtros y generación de filas en la convolución

[5]Single-Instruction Multiple Data

En la operación de convolución, al aplicar un valor del filtro a una neurona, su potencial de membrana puede alcanzar el umbral. En ese caso, esta neurona debe dispararse, reseteando su valor de potencial de membrana y almacenando su dirección en una FIFO[6] de salida. La ventaja de esta implementación es que disminuye el número de accesos a memoria, ya que los datos se leen y convolucionan fila a fila. Fig. 5 describe el proceso de convolución. En este ejemplo se observa que los valores de MP se mantienen positivos y por debajo del umbral.

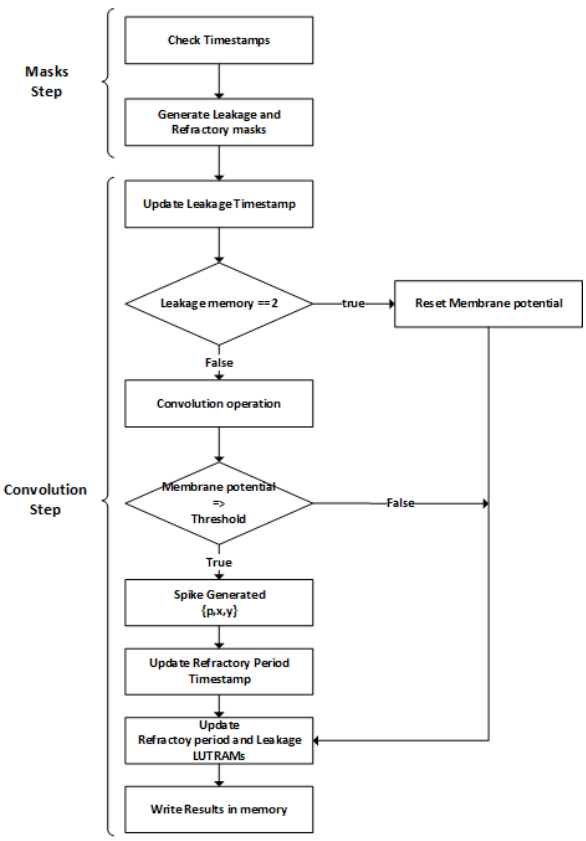


Fig. 5: Fases de la operación de convolución.

F. Implementación en hardware

El diseño fue descrito como RTL[7] con el lenguaje System Verilog y sintetizado para una plataforma Zynq-7100 MMP usando Vivado 2019.2. Esta plataforma contiene un microcontrolador PSoC con un Dual ARM Cortex-A9 como sistema de procesamiento (PS), y una Kintex-7 FPGA como lógica programable (PL), que contiene 444 K celdas lógicas en el mismo chip.

En esta implementación, la plataforma Zynq ejecuta un sistema operativo (OS) empotrado llamado Petalinux en el PS, que permite a los desarrolladores configurar el sistema comodamente. La configuración consiste en un programa C++ que lee un archivo de texto con los valores de los parámetros y los envía a la PL, el cual se comunica con el PS mediante un bus AXI [8]. Una vez configurado el sistema, recibe y envía eventos mediante las interfaces AER.

Las interfaces del acelerador están divididas en dos buses diferentes: un bus esclavo AXI [31], que configura los parámetros del sistema (tiempo de pérdida, decaimiento, umbral de potencial de membrana y valores de filtros), y dos buses AER [32][33][34], que reciben eventos de un sensor y envían eventos salida. El diagrama de la arquitectura se muestra en la Fig.6.

Los recursos necesarios de la parte FPGA (PL) han sido $\tilde{2}$12k LUTs (76 %), 50k LUTRAMs (46 %), $\tilde{1}$70k Flip-Flops (30 %) and 708 BRAM (94 %). Y se require un consumo estimado dinámico de 573mW para los relojes, 639mW para las señales, 479mW para las puertas lógicas y LUTRAM, 11mW para los BRAM, sumando un total de 1708mW dinámicos de la FPGA.

IV. Resultados experimentales

El diseño experimental se muestra en la Fig. 7. Consiste en una placa USBAERmini2 [32] que reproduce eventos, previamente grabados de una DVS128, desde un ordenador portátil mediante USB y los envía mediante una interfaz AER a la FPGA donde los eventos son procesados. Luego, los eventos de salida de la FPGA son recogidos por la misma placa USBAERmini2 (puerto de monitorización) y visualizados por el software jAER [35], que permite grabarlos en un archivo .aedat en el ordenador para su posterior uso.

El experimento propuesto consiste en procesar segmentos de escenas visuales en tiempo real con el dataset MNIST-DVS [36], utilizando los pesos de la red LeNet-5 entrenada con MNIST. Para poder evaluar la salida del procesador convolucional, se utilizaron los 6 filtros de la primera capa y primer filtro de la segunda capa del modelo Letnet-5 [37]. Estos se obtienen a partir de un entrenamiento básico de 10 iteraciones, con una precisión del 99.86 % para el dataset MNIST basado en fotogramas. Luego se codifican en conjunto con los parámetros del modelo neuronal LIF mediante un parser de Python que genera el archivo de texto de configuración. En la Fig. 8 se tienen los mapas para los filtros de la primera capa de la red $(K1, K2, ..., K6)$, en los cuales, las tonalidades azules representan valores negativos y las tonalidades amarillas representan valores positivas.

La FPGA se configura para utilizar un convolucionador con tamaño de filtro de 5x5 píxeles, el cual, se codifica en un archivo de texto en conjunto con los restantes parámetros de configuración del procesador de convolución (umbral de potencial de membrana, dacaimiento, pérdida, período refractario, etc.). Luego, mediante el visualizador de la plataforma neuromórfica jAER, que integra las propiedades de múltiples sensores visuales y auditivos, per-

[6]First In First Out
[7]RRegister-Transfer Level

[8]Advanced eXtensible Interface. Es una interfaz de bus estandar ARM incluida en la tercera generación de estandares libres de la Arquitectura de Buses para Microcontroladores Avanzados (AMBA). Se utiliza en Zynq para la interfaz de comunicación de alto rendimiento entre PS y PL

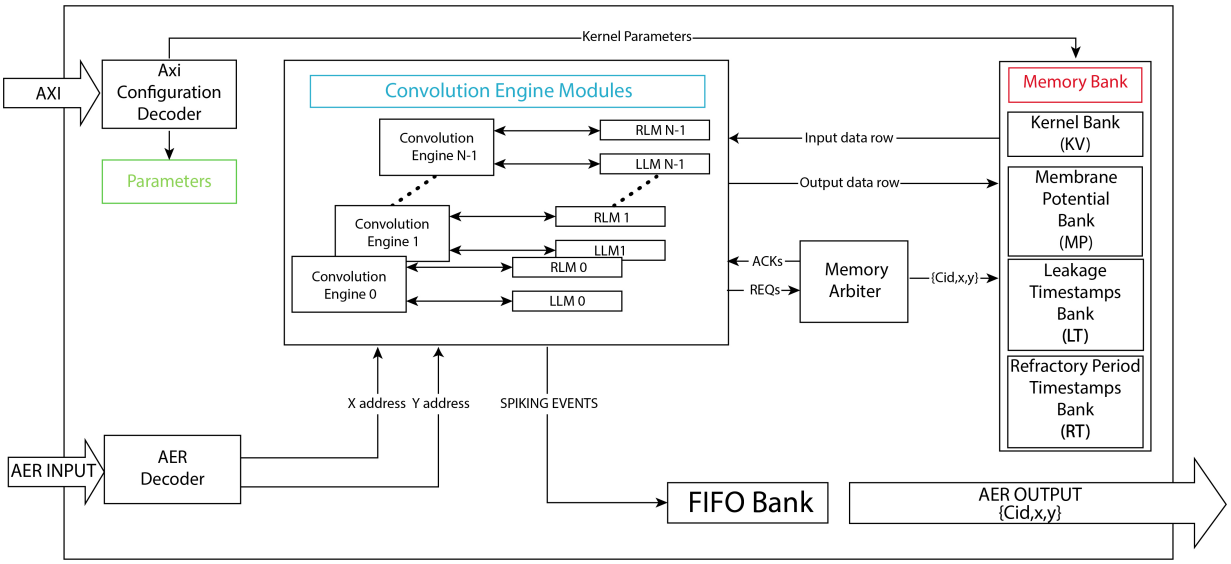


Fig. 6: Diagrama de bloques de la arquitectura del sistema

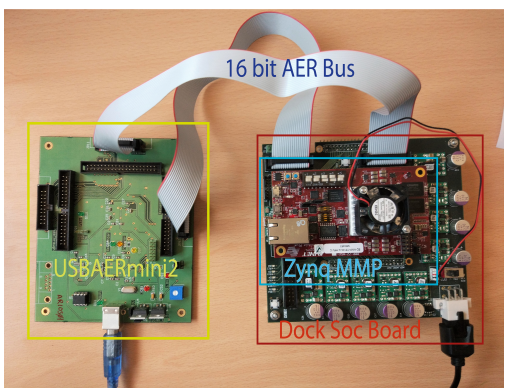


Fig. 7: Configuración experimental

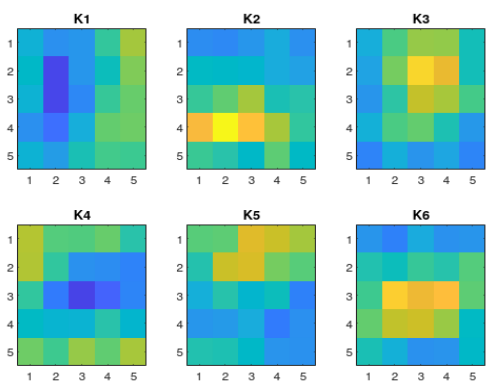


Fig. 8: Mapas de los filtros de la primera capa de convolución

mite la captura del flujo de eventos en tiempo real durante la secuenciación. Esta secuenciación se realiza igualmete con jAER y mediante el uso de la tarjeta USBAERmini2.

A partir de los numerosos archivos del dataset, que consisten en grabaciones realizadas por una retina DVS de 128x128 pixeles y representan movimientos de los dígitos manuscritos, se tomaron observaciones del dígito "8" como caso de ejemplo. En la emulación del modelo CNN para la variante pulsante con el procesador antes descrito, se definieron etapas de procesamiento similares a las CNN tradicionales, como puede apreciarse en la Fig. 9.

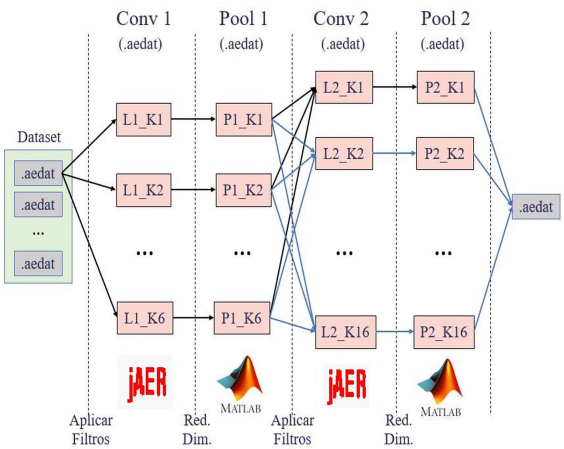


Fig. 9: Etapas de procesamiento SCNN

En la figura se distinguen las capas de convolución, la primera con 6 filtros y la segunda con 16 filtros según el modelo Lenet-5, y las capas de agrupamiento o submestreo asociadas a ellas. La salida de cada una de las capas será un archivo de extensión ".aedat" correspondiente a la grabación de una escena visual con un sensor DVS. En este caso, el archivo se obtiene a partir de la secuenciación con el jAER para las capas convolucionales donde se aplica el filtro correspondiente $(L1\_K1, L1\_K2, ..., L1\_K6)$, y para la capas de agrupamiento se reduce la dimensionalidad mediante scripts de MatLab. De este modo, es posible extraer características del archivo original en función de una futura clasificación del mismo.

A partir de las grabaciones de salida para la primera capa, se reitera la secuenciación para aplicar los filtros de la segunda capa de convolución. En esta última se le aplica cada uno de los 16 filtros que posee a los 6 mapas de salida de la primera capa de agrupamiento, para luego integrarlos en un solo archivo que sería la salida para cada uno de los fil-

tros de la segunda capa de convolución. En la Fig. 9, se aprecia mediante las conexiones entre bloques de color negro, el experimento que fue realizado para obtener la salida del primer filtro de la segunda capa de convolución.

En la Fig. 10 se muestra en las columnas primera y tercera, las salidas para los 6 filtros de la primera capa de convolución antes de reducir la dimensión (previo al pooling), al considerar una acumulación de eventos durante un breve período de tiempo. En la columna central, se tiene en la parte superior un histograma del archivo de entrada, en el cual son distinguibles las polaridades de los eventos generados en las diferentes secciones del digito de estudio (píxeles blancos y negros). En la posición central de la figura, se aprecia la integración de los 6 mapas de características luego de aplicarles el submuestreo posterior a la primera capa de la CNN y de aplicar el primer filtro de la segunda capa de convolución. Finalmente, en la parte inferior central se ofrece la tasa para la generación de eventos en el tiempo en cada capa de convolución.

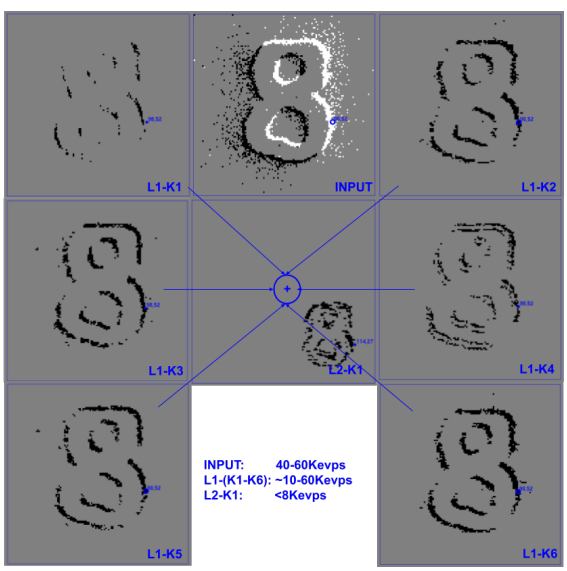


Fig. 10: Entrada al modelo y mapas de características generados por la SCNN

Con este experimento se valida la viabilidad de usar este sistema de convoluciones pulsantes para implementar una CNN pulsante. Actualmente estamos trabajando en una modificación que permita configurar los 64 convolucionadores para trabajar por conjuntos con las diferentes capas de una CNN pulsante, de forma que los eventos producidos a la salida de un convolucionador, puedan ser rutados al correspondiente convolucionador de la siguiente capa de forma sucesiva. Con ello se persigue que, únicamente la salida de la última capa de la CNN pulsante, salga por el puerto AER de salida del sistema. Esto requiere de enrutador de eventos a la entrada y otro a la salida y que sean configurables según la SCNN a desplegar.

V. Conclusiones

En artículo se ha presentado un procesador de convolución pulsante para FPGA y los resultados obtenidos en la ejecución del modelo CNN Lenet-5 con el dataset MNIST-DVS. El sistema se caracteriza por la lectura y escritura fila por fila e implementa propiedades de la neurona LIF como la pérdida y el período refractario. El tamaño de filtro utilizado (5x5 pixeles) se corresponde con la estructura definida por los bancos de memoria BRAM, el ancho del bus de datos y la frecuencia de operaciones en la plataforma Zynq.

En la literatura consultada, se constata la actualidad e importancia en la implementación de las SCNN para la reducción considerable del tiempo de procesamiento y el consumo energético. En el caso que nos ocupa, la propagación por las diferentes capas de los eventos de entrada, provenientes de las grabaciones hechas por la actividad de un sensor DVS-128, transcurre sin retrasos para una recepción de los eventos de salida en un orden temporal de milisegundos.

El diseño experimental considerado, aunque guiado en las diferentes etapas en el procesamiento de los datos mediante el uso de varias herramientas, se ajustó al procedimiento seguido por las CNN tradicionales. A partir del mismo, se concretaron salidas coherentes en la aplicación de filtros y la reducción de dimensionalidad.

Los resultados obtenidos de la experimentación confirman la viabilidad de uso del acelerador para modelos CNN basados en fotogramas y datasets compuestos por escenas visuales grabadas por retinas DVS, cuyos requerimientos de memoria y ancho de banda en el tráfico de datos, se correpondan con las bondades de la plataforma neuromórfica utilizada. Para trabajos futuros, se pretende lograr la ejecutación del procedimiento de evaluación de la SCNN mostrado, con una mínima o ninguna intervención humana, de modo que se gane en su automatización.

Agradecimientos

Este trabajo ha sido parcialmente soportado por el proyecto MINDROB (PID2019-105556GB-C33) financiado por MCIN/AEI/10.13039/501100011033 y el proyecto CHIST-ERA H2020 grant SMALL (PCI2019-111841-2) financiado por MCIN/AEI/10.13039/501100011033. Además, el trabajo de D.A. Curra-Sosa está soportado por la beca de formación del profesorado universitario (FPU) de la Junta de Andalucía.

Referencias

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, , no. 521, pp. 433–44, 2015.

[2] C. Mead and M. Ismail, *Analog VLSI implementation of neural systems*, vol. 80, Springer Science Business Media, 1989.

[3] P. Sterling and S. Laughlin, *Principles of neural design*, MIT press, 2015.

[4] F. Paredes-Vallés, K.Y.W. Scheper, and G.C.H.E. De Croon, "Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion," *IEEE Transactions on Pattern Analy-*

sis and Machine Intelligence, vol. 42, no. 8, pp. 2051–64, 2018.

[5] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128 × 128 1.5 % contrast sensitivity 0.9 % fpn 3 s latency 4 mw asynchronous framefree dynamic vision sensor using transimpedance preamplifiers," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 3, pp. 827–38, 2013.

[6] A. Jimenez-Fernández et al., "A binaural neuromorphic auditory sensor for fpga: A spike signal processing approach," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 4, pp. 804–18, 2017.

[7] Y. Tsividis, "Event-driven data acquisition and digital signal processing—a tutorial," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 8, pp. 577–81, 2010.

[8] G. Gallego et al., "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 154–80, 2010.

[9] C. Farabet et al., "Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convnets for visual processing," *Frontiers in neuroscience*, vol. 6, pp. 32, 2012.

[10] A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, A. Jiménez, M. Rivas, G. Jiménez, and A. Civit, "On the aer convolution processors for fpga," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 4237–4240.

[11] L. Camuñas-Mesa, C. Zamarreño-Ramos, A. Linares-Barranco, A.J. Acosta-Jiménez, T. Serrano-Gotarredona, and B. Linares-Barranco, "An event-driven multi-kernel convolution processor module for event-driven vision sensors," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 2, pp. 504–517, 2012.

[12] C. Zamarreño-Ramos, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco, "Multicasting Mesh AER: A Scalable Assembly Approach for Reconfigurable Neuromorphic Structured AER Systems. Application to ConvNets," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 7, no. 1, pp. 82–102, feb 2013.

[13] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jiménez, C. Serrano-Gotarredona, J.A. Pérez-Carrasco, B. Linares-Barranco, A. Linares-Barranco, G. Jiménez-Moreno, and A. Civit-Ballcels, "On real-time aer 2-d convolutions hardware for neuromorphic spike-based cortical processing," *IEEE Transactions on Neural Networks*, vol. 19, no. 7, pp. 1196–1219, July 2008.

[14] L. Camuñas-Mesa, A. Acosta-Jiménez, C. Zamarreño-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 32×32 pixel convolution processor chip for address event vision sensors with 155 ns event latency and 20 meps throughput," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 4, pp. 777–790, April 2011.

[15] B. Fang, Y. Zhang, R. Yan, and H. Tang, "Spike trains encoding optimization for spiking neural networks implementation in fpga," in *2020 12th International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, August 2020, pp. 412–418.

[16] H. Irmak, F. Corradi, P. Detterer, N. Alachiotis, and D. Ziener, "A dynamic reconfigurable architecture for hybrid spiking and convolutional fpga-based neural network designs," *Journal of Low Power Electronics and Applications*, vol. 11, no. 3, pp. 32, 2021.

[17] B. Veeravalli and X. Fong, "An fpga-based co-processor for spiking neural networks with on-chip stdp-based learning," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, May 2022, pp. 2157–2161.

[18] J. Zhang, L. Feng, T. Wang, W. Shi, Y. Wang, and G. Zhang, "Fpga-based implementation of an event-driven spiking multi-kernel convolution architecture," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1682–1686, 2021.

[19] X. Huang, E. Jones, S. Zhang, S. Xie, S. Furber, J. Goulermas, and A. Hamilton, "An fpga implementation of convolutional spiking neural networks for radioisotope identification," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, May 2021, pp. 1–5.

[20] J. Zhang, R. Wang, T. Wang, J. Liu, S. Dang, and G. Zhang, "A configurable spiking convolution architecture supporting multiple coding schemes on fpga," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 12, pp. 5089–5093, 2022.

[21] L. Feng, Y. Zhang, and Z. Zhu, "An efficient multilayer spiking convolutional neural network processor for object recognition with low bitwidth and channel-level parallelism," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 12, pp. 5129–5133, 2022.

[22] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[23] S. C. Wong, M. Jasiunas, and D. Kearney, "Fast 2d convolution using reconfigurable computing," in *Proceedings of the Eighth International Symposium on Signal Processing and Its Applications, 2005.*, August 2005, vol. 2, pp. 791–794.

[24] C. Bartolozzi, R. Benosman, K. Boahen, G. Cauwenberghs, T. DelbrÃ¼ck, G. Indiveri, S. C. Liu, S. Furber, N. Imam, B. Linares-Barranco, T. Serrano-Gotarredona, K. Meier, C. Posch, and M. Valle, "Neuromorphic systems," 11 2016.

[25] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biological cybernetics*, vol. 95, no. 1, pp. 1–19, 2006.

[26] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010.

[27] J. A. Pérez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco, "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing - Application to feedforward convnets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2706–2719, 2013.

[28] R. Tapiador-Morales, A. Linares-Barranco, A. Jimenez-Fernandez, and G. Jimenez-Moreno, "Neuromorphic lif row-by-row multiconvolution processor for fpga," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 1, pp. 159–169, 2019.

[29] Kai Hwang, Shun Piao Su, and Lionel M. Ni, "Vector Computer Architecture and Processing Techniques," *Advances in Computers*, pp. 115 – 197, 1981.

[30] Z. Chen and D. Kaeli, "Balancing scalar and vector execution on gpu architectures," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 973–982.

[31] A R M AMBA, "Axi4-stream protocol specification," 4.

[32] R. Berner, T. Delbruck, A. Civit-Balcells, and A. Linares-Barranco, "A 5 meps 100 usb2.0 address-event monitor-sequencer interface," in *2007 IEEE International Symposium on Circuits and Systems*, 2007, pp. 2451–2454.

[33] Taras Iakymchuk, Alfredo Rosado, Teresa Serrano-Gotarredona, Bernabé Linares-Barranco, Angel Jiménez-Fernández, Alejandro Linares-Barranco, and Gabriel Jiménez-Moreno, "An AER handshake-less modular infrastructure PCB with x8 2.5 Gbps LVDS serial links," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 1556–1559.

[34] A. Ríos-Navarro, J. P. Domínguez-Morales, R. Tapiador-Morales, D. Gutierrez-Galan, A. Jiménez-Fernández, and A. Linares-Barranco, "A 20mevps/32mev event-based usb framework for neuromorphic systems debugging," in *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, June 2016, pp. 1–6.

[35] T Delbrück, "jAER open source project (2007)," .

[36] A. Yousefzadeh, T. Serrano-Gotarredona, and B. Linares-Barranco, "MNIST-DVS mnist-dvs and flash-mnist-dvs databases," 2015.

[37] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

Aceleración del Análisis de Componentes Conectadas en Imágenes para Procesamiento de Altas Prestaciones en el Borde

José L. Mira, Jesús Barba, Julián Caba, José A. de la Torre, Fernando Rincón, Soledad Escolar y Juan Carlos López[1],

Resumen— **En el procesamiento de imágenes, el algoritmo de Componentes Conectadas es un método utilizado para identificar y etiquetar los distintos objetos o regiones presentes en una imagen digital. Este algoritmo puede ser útil para diversas tareas de procesamiento de imágenes, como el reconocimiento de objetos, la segmentación de imágenes y la extracción de características. Este trabajo presenta la implementación de un algoritmo de paso único en un dispositivo basado en FPGA adecuado para aplicaciones de visión por computador de alto rendimiento en el edge, la tarjeta de computación Ultra96-V2. El diseño y la implementación del núcleo IP se han enfrentado a retos utilizando el flujo de trabajo y las herramientas HLS de AMD-Xilinx, que requieren un uso eficiente y optimizado de los recursos, así como la reingeniería del algoritmo para cumplir con los requisitos impuestos por el marco de desarrollo. El rendimiento del acelerador propuesto se ha analizado a fondo utilizando el marco de evaluación comparativa YACCLAB frente a una CPU de gama alta y otra de gama baja. Los resultados muestran una pérdida de rendimiento esperada debido a las limitaciones de memoria y frecuencia de reloj. Sin embargo, en lo que respecta a la eficiencia energética, la arquitectura multinúcleo de hardware supera a las alternativas de software con una mejora de entre dos y cinco veces, dependiendo del tamaño y la complejidad de las imágenes.**

Palabras clave— **Análisis de componentes conectadas, computación de altas prestaciones, procesamiento en el borde, visión por computador, FPGA**

I. Introducción

El análisis de componentes conectadas (CC) es un paso crucial en muchas aplicaciones de visión por computador, que consiste en asignar etiquetas únicas a cada región de una imagen que ha sido previamente segmentada aplicando un proceso de umbralización para diferenciar los objetos del fondo. A continuación, se extraen las características de cada región, como el área, el centro de gravedad, el cuadro delimitador y el valor del píxel, basándose en sus etiquetas con el objetivo de clasificar cada región en una de las múltiples clases.

La complejidad temporal y el uso intensivo de memoria son dos de los principales problemas a los que hay que hacer frente cuando se trata de desarrollar una solución válida para dispositivos integrados con una cantidad de recursos limitados o un consumo de energía restringido. De hecho, este es el contexto para el procesamiento de imágenes en el edge, un paradigma informático que realiza las tareas de procesamiento de imágenes localmente, cerca de la fuente de los datos, en lugar de enviar todos los datos a una nube o centro de datos para su procesamiento. El procesamiento de imágenes en el edge también puede permitir el análisis en tiempo real o casi real de datos de imágenes, lo que puede ser crítico en aplicaciones como vehículos autónomos, sistemas de vigilancia o imágenes médicas.

A pesar de que ha habido múltiples algoritmos de CC a lo largo del tiempo, sólo los algoritmos de paso único son realmente adecuados para implementaciones basadas en FPGA debido a la reducción del ancho de banda de memoria, que representa el principal cuello de botella para esta clase de dispositivos [1]. Esta clase de algoritmos CC también son adecuados para el procesamiento de imágenes en streaming.

En este trabajo, se revisa el modelado, diseño e implementación del algoritmo de análisis CC singlepass propuesto originalmente por Bailey et al. en [2] y posteriormente optimizado en [3] y adaptado al flujo de diseño Xilinx-AMD Vitis. El objetivo principal es evaluar el rendimiento y la eficiencia energética de la implementación FPGA frente a las alternativas de software más avanzadas que se ejecutan en plataformas de procesadores de gama alta y baja. Queda fuera del alcance de este trabajo comparar nuestra solución hardware en términos de uso de recursos (principalmente memoria BRAM) con otras propuestas debido al uso de diferentes arquitecturas FPGA y herramientas de síntesis. Además, el modelo HLS propuesto para el núcleo se ha generalizado para permitir su evaluación comparativa mediante el framework YACCLAB [4]. La arquitectura permite configuraciones multinúcleo, contador de etiquetas en el peor de los casos, así como resoluciones más allá de la resolución común de 640x640 reportada en la mayoría de la literatura [5][3], lo que aumenta la demanda de recursos.

II. El acelerador HW-CC

Aunque el algoritmo propuesto por Bailey et al. [2] está diseñado teniendo en cuenta los dispositivos de computación basados en FPGAs evitando, por ejemplo, la necesidad de almacenar toda la imagen en memorias BRAM, sigue presentando retos a la hora de empaquetarlo como un núcleo completamente funcional para arquitecturas FPA-SoCs como la Xilinx ZynQ-UltraScale+. Además, se ha realizado un análisis del rendimiento y los recursos, limitando la validación a un tamaño específico de imágenes de entrada y complejidad. Por lo tanto, falta una visión

[1]Escuela Superior de Informática. Dpto. de Tecnologías y Sistemas de Información. Universidad de Castilla-La Mancha, Ciudad Real. e-mail: joseluis.mira@uclm.es

completa del desarrollo real de las soluciones basadas en FPGA, lo que impide una comparación justa con las soluciones de software. Además, enfrentarse al modelado de la arquitectura y la lógica del algoritmo utilizando la tecnología de Síntesis de Alto Nivel (HLS) exige la reingeniería del diseño original para hacerlo compatible con las restricciones semánticas y sintácticas de las herramientas HLS.

La principal aportación de este trabajo es el desarrollo del acelerador HW-CC, un núcleo IP totalmente parametrizado y listo para su uso en la plataforma FPGA-SoC de Xilinx. La parametrización permite su uso para un amplio rango de imágenes de entrada y escenarios (ver III) haciendo que nuestra solución sea flexible para adaptarse a diferentes requisitos de aplicaciones y plataformas de destino (disponibilidad de memoria, latencia, etc.). Esta flexibilidad tiene un coste, principalmente debido a las grandes necesidades de memoria para almacenar datos intermedios cuando se trabaja con imágenes grandes; por ejemplo, la *data table*, una estructura de datos que contiene estadísticas de las regiones de etiquetas, depende de la anchura (W) y la altura (H) siendo $W/2 \times H/2$ el máximo teórico. Por este motivo, en la solución propuesta, se han tenido que desarrollar mecanismos para superar este hándicap, haciendo que una plataforma de gama baja basada en FPGA sea adecuada para los algoritmos CC.

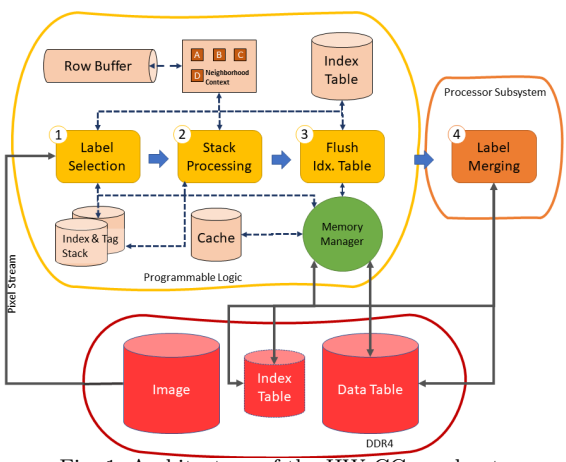


Fig. 1: Architecture of the HW-CC accelerator

Fig. 1 esboza una vista de alto nivel de la arquitectura HW-CC que se ha modelado en Vitis HLS C/C++. La interfaz de memoria utiliza puertos AXI-Stream y AXI-Memory para que el núcleo IP pueda integrarse fácilmente con el subsistema del procesador. Este subsistema es el encargado en ejecutar el firmware para la configuración de la plataforma y los periféricos, la gestión de la transferencia DMA y la ejecución del paso *Label Merging*.

El modelo HLS define el control y los elementos de memoria, junto con su mapeo a los recursos de la FPGA, que realiza el algoritmo de la forma más óptima mediante el uso de los pragmas y parámetros de modelo adecuados.

A. Elementos de la memoria

- **Neigbourhood Context + Row Buffer**: Estos dos elementos se encargan de almacenar temporalmente la información necesaria para determinar la creación y actualización de etiquetas asociadas a grupos de píxeles. El tamaño del Row Buffer es un parámetro del modelo.
- **Index Table**: Matriz unidimensional que contiene el valor de la etiqueta a la que apunta un índice determinado. El tamaño de esta matriz puede parametrizarse en el modelo para ajustarse a los requisitos típicos de la imagen de entrada. Otro parámetro del modelo es el número máximo de etiquetas, que determina el tamaño de la palabra para la tabla de índices.
- **Index & Tag Stack**: Se trata de una estructura de datos utilizada para fusionar dos regiones en función de los valores de sus etiquetas. Esta estrucura es accedida una vez al final de cada fila de la imagen. Para aprovechar que el acceso a la información de índices y etiquetas es independiente, se ha decidido aplicar el pragma *ARRAY PARTITION* para permitir el acceso paralelo a ambos valores. El tamaño de la pila también es un parámetro del modelo.
- **Memory Manager + Data Table + Cache**: La *data table* es la estructura en la que se almacenan las estadísticas relacionadas con las etiquetas. En la aplicación propuesta se proporcionan las coordenadas de la esquina superior izquierda, la anchura, la altura y el area de cada región. Los requisitos de memoria de la tabla de datos agotarían rápidamente las BRAM disponibles en la estructura de la FPGA, especialmente para altas resoluciones. Para que el acelerador HW-CC se adapte a una gran variedad de tamaños y complejidades, se ha decidido implementar un sencillo sistema de caché que permita al núcleo manejar grandes tamaños de imagen, sin sacrificar la velocidad de acceso a los datos. Este módulo también se encarga de comprobar, leer y escribir los bloques de la memoria caché con respecto a la RAM de a bordo cuando es necesario. Dado que durante el funcionamiento del algoritmo, éste va asignando nuevas etiquetas a las nuevas regiones que atraviesa. En este proceso, se ocupa espacio en la *data table* correspondiente a estas nuevas etiquetas. Al mismo tiempo que tiene lugar este proceso, también se produce la fusión de varias etiquetas asignadas anteriormente. De esta forma, el acceso a las entradas de la tabla se suele realizar de forma alterna. A la hora de implementar la caché, este patrón de acceso a la memoria supone un problema, ya que hay zonas de la imagen en las que se accede de forma consecutiva a entradas de la *data table* pertenecientes a distintos bloques de datos. Esta situación se traduce en un elevado número de fallos de caché, con el consiguiente tiempo de ejecución empleado en tener que reemplazar el bloque de datos varias veces.

Basándose en estas características de los patrones de acceso a la memoria, se decidió implementar un sistema de caché de 2 vías para evitar la sustitución innecesaria de bloques de datos.

B. Etapas del algoritmo

B.1 Label Selection (HW)

Esta etapa se realiza una vez para cada píxel de la imagen, donde se evalúa el Neighbourhood Context para determinar qué etiqueta se generará en un píxel dado. En este proceso pueden darse las siguientes situaciones:

- Generar una nueva etiqueta, para lo cual hay que crear un nuevo registro en la tabla de datos.
- Asignar una etiqueta existente, lo que requiere acceder a la tabla de datos para actualizar las estadísticas relacionadas con esa etiqueta.
- Hay dos posibles etiquetas en el Contexto de Vecindad por lo que se almacenan en el *Index & Tag Stack* para ser resueltas más tarde, la menor de las dos se asigna al píxel y las estadísticas se actualizan en la *Data Table*.

B.2 Stack Processing (HW)

Esta etapa se realiza para cada fila de la imagen que ha sido procesada por el algoritmo. Accede al *Index & Tag Stack* para realizar el proceso de fusión entre etiquetas, que accede a la tabla de índices para actualizar los índices correspondientes.

C. Flush Index Table (HW)

Una vez que el algoritmo ha terminado de recorrer la imagen, los bloques de datos almacenados en caché se actualizan en la DDR.

C.1 Label Merging (SW)

Una vez finalizado el componente FPGA, el procesador recorre la Tabla de Índices y la Tabla de Datos contenidas en la memoria RAM para realizar el proceso final de fusión entre las diferentes regiones con el fin de obtener el número final de etiquetas y sus estadísticas. Este paso se trasladó al ámbito del software debido a que se trata de una tarea intensiva de control con acceso irregular a patrones de memoria, un escenario en el que el procesador rinde mejor que la FPGA.

III. Resultados

En esta sección, se presentaran los resultados de nuestra evaluación de rendimiento de varios algoritmos de etiquetado de regiones utilizando el marco de referencia YACCLAB [4]. Nuestra evaluación tenía como objetivo comparar la velocidad y la eficiencia energética del acelerador HW-CC IP propuesto frente a diferentes algoritmos en un conjunto de imágenes de prueba proporcionadas por YACCLAB. Las pruebas de software se ejecutaron en: (1) un Intel(R) Core(TM) i7-12700K de 12ª generación (3,6GHz, 64GB DDR5) ejecutando una distribución Ubuntu 20.4 GNU/Linux ; y (2) un Intel(R) Celeron(R) N5095 (2,0GHz, 8GB DDR4) ejecutando una distribución Ubuntu 22.4 GNU/Linux. En ambos casos, el conjunto de pruebas se compiló con GCC 11.3. Al incorporar tanto un procesador de estación de trabajo de gama alta como una CPU de gama baja comúnmente presente en las soluciones comerciales de computación en el edge, la validación experimental puede proporcionar una mejor comprensión de las capacidades del acelerador propuesto.

El conjunto de pruebas incluyó los algoritmos de CC listos para usar que vienen con YACCLAB, a saber: Scan Array-based with Union Find [6] (SAUF), Block-Based with Decision Tree [7] (BBDT), Pixel Prediction [8] (PRED), Directed Rooted Acyclic Graph [9] (DRAG) y Spaghetti Labelling [10].

El conjunto de datos utilizado para la validación experimental comprendía una selección del subconjunto YACCLAB 2D, que consiste en imágenes binarias con conected components etiquetados. El conjunto de datos incluye tanto imágenes sintéticas como imágenes del mundo real procedentes de diversas fuentes. Las imágenes reales se utilizan para probar el rendimiento de los algoritmos en imágenes con variaciones naturales de forma e intensidad. En este trabajo, probamos el acelerador HW-CC para el procesamiento de imágenes en el edge sobre MIR-flickr (25K imágenes, tamaño medio y componentes conectados: 0,17 megapíxeles y 495, respectivamente), Hamlet (104 imágenes), Tobacco800 (1290 documentos, 150-300 PPP y resoluciones de 1200x1600 a 2500x3200), 3DPeS y subconjuntos de imágenes de ruido aleatorio sintético. Cabe destacar que este último permite comprobar objetivamente la escalabilidad y eficacia de distintos algoritmos. Este subconjunto proporciona diez imágenes para cada combinación de tamaño (32x32 hasta 4096x4096 en pasos de x2) y densidad (10 % a 90 %), lo que da un total de 720 imágenes.

La plataforma de prototipado Ultra96-v2 se ha utilizado para desplegar y probar el núcleo del acelerador de procesamiento. La placa Ultra96-v2 se basa en el MPSoC Xilinx Zynq UltraScale+, que integra lógica programable y un procesador Arm Cortex-A53 de 64 bits y cuatro núcleos. El chip ZU3EG integrado en la plataforma es una FPGA de gama baja con capacidades modestas y bajo consumo de energía, lo que la hace adecuada para aplicaciones de computación en el edge. Se ha utilizado la versión 2021.1 de la suite de herramientas de Xilinx (Vitis HLS, Vivado y Vitis Unified Software Platform) para modelar el núcleo IP HW-CC, diseñar y sintetizar la plataforma y desplegar toda la solución.

En primer lugar, evaluamos el rendimiento de la solución propuesta frente a los subconjuntos del mundo real seleccionados. Aunque la arquitectura del acelerador HW-CC es flexible y permite configurar múltiples parámetros, para este experimento se propuso una estrategia agresiva con el fin de obtener el máximo rendimiento. Por lo tanto, el tamaño del búfer de fila se estableció en 4K palabras, la memoria

de datos de etiquetas en 64K y la memoria de bloque de caché en 1K. El número de núcleos aceleradores instanciados en la FPGA es de cuatro, lo que eleva la utilización de los recursos BRAM al 83 % del total disponible en el dispositivo. Los resultados posteriores a la síntesis mostraron que era posible configurar un reloj de 175 MHz para controlar la lógica. Sin embargo, establecimos un enfoque conservador a este respecto para evitar comportamientos inesperados en la placa. Así, se estableció una configuración de reloj de 150MHz.

Tabla I: Tiempo medio de ejecución (ms) para el subconjunto de imágenes reales

	MIRFlickr	3DPes	Hamlet	Tobacco	Avg. Gain
HW-CC-150	1.71	2.73	19.77	36.92	-
Intel i7-12700K					
SAUF	0.44	0.61	4.96	7.65	**-4.29x**
BBDT	0.36	0.35	3.44	4.93	**-6.45x**
PRED	0.54	0.72	6.92	10.52	**-6.33x**
DRAG	0.36	0.35	3.44	4.91	**-6.46x**
Spaguetti	0.2	0.18	1.89	2.68	**-11.99x**
Intel Celeron N5095					
SAUF	1.46	2.76	22.19	37.33	**-1.01x**
BBDT	0.88	1.33	13.86	22.96	**-1.76x**
PRED	1.49	2.87	22.96	38.69	**+1.02x**
DRAG	0.87	1.33	13.86	23.03	**-1.76x**
Spaguetti	0.41	0.53	5.16	7.89	**-4.46x**

La Tabla I presenta los tiempos medios de ejecución para todos los subconjuntos de imágenes. Los resultados muestran una pérdida significativa de rendimiento en comparación con el procesador i7, y una disminución moderada del rendimiento en el caso del procesador Celeron.

La característica más atractiva de la arquitectura Zynq UltraScale+ para aplicaciones de computación en el edge es su eficiencia en términos de consumo de energía. El consumo medio de energía de nuestro HW-CC se ha medido utilizando la interfaz proporcionada por la *Platform Management Unit* integrada, lo que ha dado como resultado 3,74 W durante la ejecución de la prueba. En el caso del procesador i7, se ha utilizado la herramienta de línea de comandos *powerstat* para monitorizar el incremento en el consumo de energía a través de la interfaz RAPL (Running Average Power Limit); se observó una media de 31,4W. Sin embargo, en la plataforma informática basada en Celeron se utilizó un medidor de consumo de energía externo que registró un incremento sostenido de 10,3 W durante la ejecución de la prueba.

Tabla II: Ganancia en la eficiencia energética media (J) para el subconjunto real de imágenes. HW-CC vs Intel i7 & Celeron.

	SAUF	BBDT	PRED	DRAG	Spaghetti
Intel i7	1.87	1.24	2.56	1.24	0.67
Intel Celeron	2.87	1.75	2.97	1.76	0.63

La Tabla II muestra la comparación de la eficiencia, expresada como *Julios* consumidos, desglosada por subconjunto y algoritmo. Los resultados medios exponen una ganancia consistente entre plataformas en todos los casos excepto en el algoritmo Spaghetti, que supera al resto debido a su bajo tiempo de ejecución.

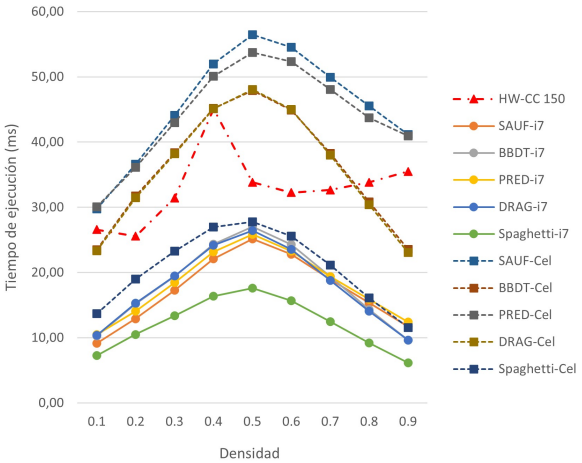


Fig. 2: Resultados por densidad de test

Para validar el rendimiento y la escalabilidad de una solución, YACCLAB incluye un conjunto de imágenes generadas aleatoriamente que permite la ejecución de pruebas de densidad y tamaño. Las Figuras 2 y 3 muestran una comparación de los tiempos de ejecución de cada algoritmo de software y la versión de 4 núcleos a 150 MHz del acelerador HW-CC propuesto.

El tiempo medio de ejecución es estable para HW-CC con valores ligeramente superiores a medida que aumenta la densidad, con un pico en las imágenes de densidad de primer plano de 0,4, como puede verse en la Fig. La comparación con el procesador i7 muestra una degradación del rendimiento de $\approx -2x$ para todos los algoritmos excepto Spaghetti que alcanza $-3,03x$ con una pérdida máxima de $5,76x$ para imágenes de densidad 0,9. Sin embargo, HW-CC es más rápido que el procesador Celeron en todos los casos - SAUF (35,9 %), BBDT (4,2 %), PRED (32,6 %), DRAG (3,4 %) - pero, de nuevo, Spaghetti ($1,72x$ más lento de media con una pérdida máxima de $3,07x$ para imágenes de densidad 0,9).

En cuanto a las pruebas de tamaño, el rendimiento del acelerador propuesto mantiene el mismo patrón en todos los procesadores y algoritmos. Como puede verse en la Fig. 3, el núcleo HW-CC mantiene la misma dependencia lineal del tiempo de ejecución que sus homólogos de software. La pérdida media de rendimiento frente al procesador i7 es de $-2,14x$. Teniendo en cuenta que el consumo de energía del procesador Intel i7 es aproximadamente diez veces la energía que necesita la FPGA, se obtiene una mejora de $\approx 5x$ en términos de eficiencia energética, al tiempo que se mantiene una capacidad de computación justa para un dispositivo que se va a desplegar en edge de la infraestructura informática. En cuanto a la CPU Celeron, la pérdida media de rendimiento es de $1,09$, muy penalizada por la eficiencia del algoritmo Spaghetti ($1,82$ mejor). En los demás casos, HW-CC es un 10 % más rápido. En conjunto, nuestra solución basada en FPGA es 2,5 veces más eficiente que la plataforma informática Celeron.

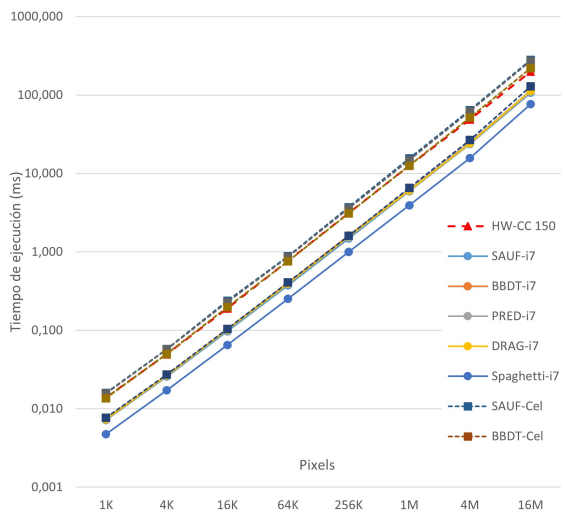


Fig. 3: Resultados por tamaño de test.

IV. Conclusiones

En este trabajo se ha realizado el diseño y desarrollo de la arquitectura de un acelerador hardware para el análisis y etiquetado de componentes en imágenes de una manera eficiente en términos de la relación potencia de cómputo versus consumo. Debido a que su despliegue tiene como objetivo plataformas de cómputo empotradas que estarán desplegadas en el borde de la infraestructura de adquisición, procesamiento y transmisión de la información (p.ej. en una plataforma IoT), el análisis de los resultados obtenidos ha tenido en cuenta, no sólo el rendimiento sino también las necesidades energéticas de la solución. En última instancia, el tipo de plataformas consideradas para la aplicación final consistirá en un dispositivo de procesamiento basado en un SoC-FPGA y estará alimentado por batería; de ahí la importancia de la dimensión energética.

La comparación con aportaciones previas en el estado del arte es bastante difícil debido a que los trabajos realizados en muchas ocasiones no proporcionan la información necesaria, tienen como objetivo dispositivos FPGA de generaciones previas y tampoco existe una estandarización respecto a las pruebas realizadas e imágenes utilizadas en las mismas (con impacto directo en la latencia de procesamiento que depende de la complejidad de la misma y su tamaño, por ejemplo). Como siguiente paso, trabajaremos en el desarrollo de un marco comparativo y recopilación de información que permita evaluar lo más objetivamente posible nuestra solución respecto a otros trabajos en el estado del arte.

Agradecimientos

Esta investigación ha sido financiada por el programa H2020 de la Unión Europea bajo el acuerdo de subvención nº 857159 (proyecto SHAPES) y por el Ministerio de Economía y Competitividad (MINECO) del Gobierno de España bajo el proyecto TALENT (PID2020-116417RB-C4, subproyecto 1) y el proyecto proyecto MIRATAR (TED2021-132149B-C41).

Referencias

[1] Robert Walczyk, Alistair Armitage, and T. David Binnie, "Comparative study on connected component labeling algorithms for embedded video processing systems," in *International Conference on Image Processing, Computer Vision, & Pattern Recognition*, 2010.

[2] Donald Bailey and Christopher Johnston, "Single pass connected components analysis," *Proceedings of Image and Vision Computing*, 01 2007.

[3] Ni Ma, Donald G. Bailey, and Christopher T. Johnston, "Optimised single pass connected components analysis," in *2008 International Conference on Field-Programmable Technology*, 2008, pp. 185–192.

[4] Costantino Grana, Federico Bolelli, Lorenzo Baraldi, and Roberto Vezzani, "Yacclab - yet another connected components labeling benchmark," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2016, pp. 3109–3114.

[5] Fanny Spagnolo, Fabio Frustaci, Stefania Perri, and Pasquale Corsonello, "An efficient connected component labeling architecture for embedded systems," *Journal of Low Power Electronics and Applications*, vol. 8, no. 1, 2018.

[6] Kesheng Wu, Ekow J. Otoo, and Kenji Suzuki, "Optimizing two-pass connected-component labeling algorithms," *Pattern Analysis and Applications*, vol. 12, pp. 117–135, 2009.

[7] Costantino Grana, Daniele Borghesani, and Rita Cucchiara, "Optimized block-based connected components labeling with decision trees," *IEEE Transactions on Image Processing*, vol. 19, no. 6, pp. 1596–1609, 2010.

[8] Costantino Grana, Lorenzo Baraldi, and Federico Bolelli, "Optimized connected components labeling with pixel prediction," in *Advanced Concepts for Intelligent Vision Systems*, Jacques Blanc-Talon, Cosimo Distante, Wilfried Philips, Dan Popescu, and Paul Scheunders, Eds., Cham, 2016, pp. 431–440, Springer International Publishing.

[9] Federico Bolelli, Lorenzo Baraldi, Michele Cancilla, and Costantino Grana, "Connected components labeling on drags," in *2018 24th International Conference on Pattern Recognition (ICPR)*, 2018, pp. 121–126.

[10] Federico Bolelli, Stefano Allegretti, Lorenzo Baraldi, and Costantino Grana, "Spaghetti labeling: Directed acyclic graphs for block-based connected components labeling," *IEEE Transactions on Image Processing*, vol. 29, pp. 1999–2012, 2020.

Docencia y sistemas ciberfísicos

Monitorización de estado, detección de anomalías de funcionamiento y predicción de fallos en motores eléctricos

Luis Magadán, Francisco J. Suárez, Juan C. Granda[1]

Resumen—
El mantenimiento predictivo de los equipos industriales se ha convertido en un aspecto crítico en la Industria 4.0. Este artículo aborda el diseño, implementación, despliegue y prueba de un sistema de Internet de las Cosas Industrial (IIoT) de bajo costo diseñado para monitorizar el estado de motores eléctricos en tiempo real. El sistema se puede utilizar para detectar anomalías de funcionamiento y dar soporte a la diagnosis y predicción de fallos. El sistema se construye utilizando componentes de hardware de bajo costo (módulos inalámbricos multisensor y computadoras de placa única como gateways), software de código abierto y servicios en la nube abiertos, donde se almacenan las señales de vibración de los motores eléctricos. Un prototipo del sistema de monitorización ha sido desplegado y se encuentra en funcionamiento en una planta industrial cementera. También se presenta una nueva técnica que permite identificar fallos en los rodamientos de los motores y predecir su tiempo restante de vida útil (RUL) bajo diferentes condiciones utilizando modelos de aprendizaje automático basados en redes neuronales recurrentes.

Palabras clave— **IoT Industrial, Redes inalámbricas de sensores, Detección y diagnosis de anomalías, Predicción de fallos, Tiempo restante de vida útil, Aprendizaje automático.**

I. Introducción

El mantenimiento predictivo de equipos es una opción cada vez más utilizada como alternativa al simple mantenimiento reactivo (cuando los equipos fallan) o al mantenimiento preventivo (en períodos planificados), ya que supone un ahorro significativo en los costes de mantenimiento al optimizar la planificación de las operaciones necesarias. Un sistema de mantenimiento predictivo necesita disponer de una gran cantidad de datos obtenidos a partir de la continua monitorización del funcionamiento de los equipos. Con estos datos se puede realizar directamente un análisis en tiempo real para detectar automáticamente anomalías en el funcionamiento de los equipos y alertar a los técnicos de mantenimiento. Una vez detectadas las anomalías pueden ser diagnosticadas en base a la información disponible y también, a más largo plazo, se pueden alimentar modelos de predicción de fallo (construidos a partir de todo el histórico de datos de los equipos y de los posibles fallos registrados) para poder así estimar el tiempo de vida útil restante de los equipos.

En el contexto de este trabajo se ha colaborado con una importante planta industrial de producción de cemento ubicada en Asturias. En este tipo de industria se utiliza una gran cantidad de motores eléctricos como base del funcionamiento de trituradoras, molinos, cintas transportadoras, hornos rotatorios, etc. La fábrica funciona en régimen 24x7 y el fallo imprevisto de alguno de estos motores puede suponer grandes pérdidas económicas, debido por ejemplo a una disminución de la productividad. Se ha desplegado un prototipo de sistema de monitorización en tiempo real de los niveles de vibración de dos motores eléctricos de la fábrica. Se han analizado los datos capturados tanto en el dominio del tiempo como de la frecuencia con objeto de detectar y diagnosticar anomalías de funcionamiento y reducir de esta forma los costes de mantenimiento gracias a una mejor planificación de dichas labores. Se ha conseguido generar un histórico de datos en la nube de varios meses que ha servido de base para la detección y diagnosis de anomalías y servirá en un futuro para alimentar modelos predictivos de fallo.

Los resultados obtenidos son prometedores. En el dominio del tiempo se ha conseguido detectar anomalías de funcionamiento directamente a partir de la visualización de las magnitudes de vibración. En el dominio de la frecuencia el análisis de la tendencia de amplitudes de frecuencias características ha permitido detectar también anomalías de funcionamiento a partir del histórico de datos almacenado y abordar la diagnosis de tales anomalías.

Debido a la falta de suficientes datos de monitorización de los motores y con la intención de poder establecer comparativas con otros trabajos similares, se ha realizado un análsis de técnicas de predicción de de tiempo restante de vida útil de los rodamientos de los motores eléctricos haciendo uso de redes neuronales recurrentes y conjuntos de datos de dominio público relevantes sobre fallos en motores.

En las secciones que siguen se dan detalles sobre trabajos precedentes en el ámbito de la investigación llevada a cabo, el sistema de monitorización, el plan experimental diseñado y los resultados obtenidos hasta el momento para finalizar con las conclusiones y la orientación actual de la investigación.

II. Trabajo previo

La monitorización del estado de equipos es una de las bases de la Industria 4.0 [1]. Se han desarrollado muchos sistemas para monitorizar corriente, presión, temperatura y otras variables en plantas industriales. Con los avances en los sistemas microelectromecánicos, es posible desplegar una gran cantidad de sensores de bajo costo capaces de detectar, procesar y comunicarse de forma inalámbrica para recopilar in-

[1]Dpto. de Informática, Universidad de Oviedo, e-mail: magadanluis,fjsuarez,jcgranda@uniovi.es

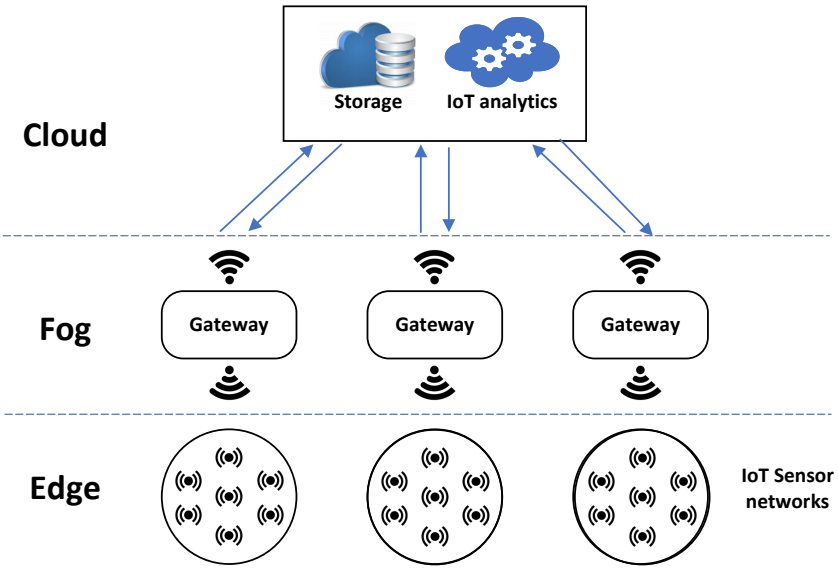


Fig. 1: Arquitectura del sistema

formación sobre el entorno y el equipo monitorizados [2]. Estos sensores se conectan mediante redes inalámbricas de sensores y envían datos a la nube para su almacenamiento o procesamiento adicional utilizando protocolos y tecnologías de IoT [3,4]. Muchos proveedores de servicios de nube pública ofrecen servicios de IoT utilizando protocolos estándar para el almacenamiento en tiempo real y el desarrollo de análisis a partir de los datos. Esto permite utilizar datos históricos para predecir fallos futuros en el equipo.

En ocasiones, la cantidad de datos que se deben enviar a la nube o la latencia de envío de datos a la nube y de regreso a los sensores/actuadores es excesiva. En estos casos, trasladar parte del procesamiento cerca de los sensores puede aliviar los recursos consumidos en la red y en la nube. El paradigma de cómputo en la niebla (fog computing) promueve el uso de recursos de sensores inteligentes y gateways como complemento de los recursos en la nube [5]. Las implementaciones fog requieren definir la topología para interconectar los sensores y los gateways que proporcionan acceso a la nube. Los sensores suelen generar flujos de datos que pueden ser preprocesados, agregados o filtrados antes de llegar a la nube [6]. De manera similar, algunas de las analíticas de datos pueden llevarse a cabo por parte de los gateways. Así pues, el paradigma de computación fog es fundamental para equilibrar la carga de cómputo y el consumo de recursos de red con el fin de ahorrar costos de la nube pública y reducir la latencia.

Los motores eléctricos de inducción son actores principales en la mayoría de las fábricas industriales, por lo que el mantenimiento predictivo de los motores eléctricos es de especial importancia. Los fallos mecánicos y eléctricos producen vibración en los motores eléctricos con diferentes amplitudes y frecuencias, razón por la que las soluciones de monitorización del estado de los motores se centran principalmente en medir las vibraciones en cada uno de los ejes de los motores [7]. Por otra parte, la mayoría de los fallos en los motores eléctricos se deben a rodamientos y los fallos en la carrera exterior de los rodamientos son además uno de los casos más comunes [8].

La detección de anomalías de funcionamiento se puede llevar a cabo incluso cuando no se dispone de datos de fallos anteriores en el equipo [9]. Para la detección y diagnosis de anomalías de funcionamiento en motores es recomendable también tener en cuenta las normas internacionales como la ISO 20816 [10], la ISO 13373 [11–14] y la ISO 17359 [15].

Para una planificación eficiente de reparaciones o reemplazo de equipos se utilizan modelos de predicción de fallos en un futuro cercano basados en aprendizaje automático [16]. Entre los diferentes tipos de modelos de predicción de fallos han cobrado bastante interés en los últimos años los modelos conducidos por datos basados en redes neuronales recurrentes [17]. Estos modelos pueden ser efectivos sin requerir un conocimiento experto en casos en los que hay una gran cantidad de datos disponibles, aunque no proporcionan una comprensión detallada de los procesos físicos subyacentes. Recopilar datos *run-to-failure*, es decir, hasta que el motor falle, con el fin de entrenar los modelos resulta inviable, ya que mantener averías que lleguen hasta el fallo puede causar la destrucción del motor. Como alternativa, existen conjuntos de datos públicos que contienen datos adquiridos a partir de pruebas de degradación acelerada de motores eléctricos. Entre los conjuntos de datos públicos que incluyen datos de fallos en rodamientos destacan IMS [18], FEMTO [19] y más recientemente XJTU-SY [20].

III. Sistema de monitorización

A. *Arquitectura del sistema*

Como se puede observar en la figura 1, la arquitectura del sistema está compuesta por tres capas en las cuales la información puede ser procesada. La

primera capa es la capa de borde (edge), la cual está compuesta por todos los sensores de IoT. La segunda capa es la capa fog, que contiene los gateways o pasarelas. La última capa es la nube (cloud), donde todos los datos relevantes son almacenados, visualizados y analizados.

Todas las capas tienen capacidad de cómputo. En la capa edge, el filtrado, la agregación y la transformación de datos son llevados a cabo directamente por los sensores. La capa fog permite a los gateways recopilar datos de múltiples sensores utilizando tecnologías de comunicación inalámbrica tales como el Bluetooth de Baja Energía (BLE) y continuar procesándolos. Tanto la capa edge como la capa fog ayudan a distribuir el procesamiento de información entre los sensores y la nube, mejorando la latencia y reduciendo la cantidad de datos a transferir a la nube.

B. Componentes del sistema

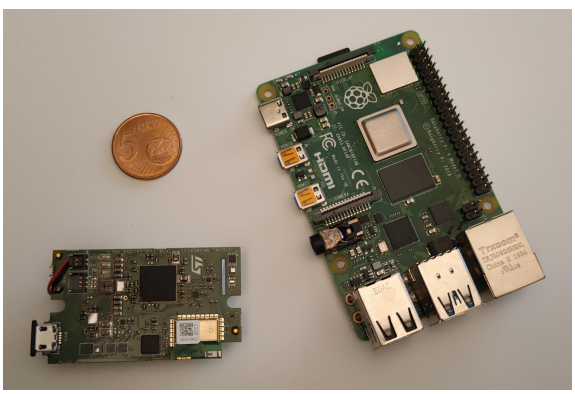


Fig. 2: Multisensor (izquierda) y Gateway (derecha)

En la capa edge se utiliza el módulo multisensor de bajo coste SensorTile.box de ST Microelectronics (Fig. 2), que cuenta con un procesador ARM Cortex-M4 con DSP y FPU, 2 MiB de memoria flash programable y ocho sensores integrados, incluyendo sensores de movimiento y humedad. LM6DSOX es el sensor de movimiento. Tiene un acelerómetro, un magnetómetro y un giroscopio que miden vibraciones con una frecuencia de salida de datos de hasta 5 kHz. El sensor de humedad es el HTS221. Mide la humedad relativa y la temperatura del entorno. El módulo admite comunicación inalámbrica con el protocolo BLE. La naturaleza inalámbrica del módulo permite una implantación muy rápida y económica en el entorno industrial.

El gateway utilizado en la capa fog es la computadora de placa única de bajo coste Raspberry Pi 4 Model B, mostrada en la Fig. 2. Cuenta con 4 GiB de RAM, dos puertos micro-HDMI, dos puertos USB 2.0, dos puertos USB 3.0, así como un puerto CSI y un puerto DSI para conectar una cámara y una pantalla táctil. La interfaz Ethernet admite velocidades de datos de hasta 1 Gbps. También incluye interfaces WiFi, Bluetooth 5.0 y BLE. Incorpora una CPU Broadcom BCM2711 (4C Cortex-A72 ARM v8 64-bit SoC @1.5 GHz) con GPU integrada. El gateway ha sido instalado con una carcasa de aluminio con un ventilador para mantener estable la temperatura del dispositivo.

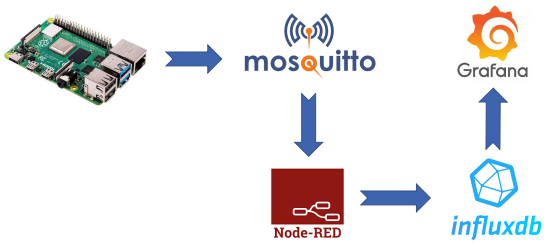


Fig. 3: Nube

Finalmente, la capa cloud está compuesta por un broker MQTT, Node-RED, InfluxDB y Grafana (Fig. 3). El gateway está conectado al broker MQTT y publica mensajes con las amplitudes de las componentes frecuenciales de la señal de vibración. Estos mensajes son recibidos, procesados y almacenados en una base de datos InfluxDB mediante diversos flujos de Node-RED. Finalmente, estos datos se muestran en un panel de control utilizando Grafana. Es una nube de código abierto que puede ser contenerizada, lo que facilita su despliegue en cualquier infraestructura, pública o privada.

C. Software

Se ha desarrollado un firmware personalizado para el módulo multisensor, que permite que el acelerómetro muestree la vibración a 5 kHz con un rango de 8 G. La información que se puede extraer de la señal de vibración del motor eléctrico en el dominio del tiempo para identificar condiciones anómalas es limitada. Se deben seguir dos pasos. Primero, los datos de aceleración se filtran e integran para obtener la velocidad. Segundo, los datos de velocidad sin procesar se transforman al dominio de frecuencia utilizando la Transformada Rápida de Fourier (FFT). Los datos se dividen en ventanas utilizando una ventana de Hanning para evitar el aliasing. La salida de la FFT es la amplitud de la velocidad media cuadrática (RMS) en función de la frecuencia. Dentro del gateway se utiliza la función FFT de la biblioteca Python SciPy, utilizando una matriz de entre 8192 y 65536 muestras. Estos intervalos de muestras, equivalentes a 1.6 y 13 segundos de operación del motor, monitorizan suficientes revoluciones del motor para cubrir todo su comportamiento dinámico, al tiempo que permiten una buena precisión (número de bins) en el espectro de frecuencia.

Los módulos multisensor y los gateways se comunican utilizando el protocolo BLE, que se utiliza para transmitir paquetes pequeños de datos leídos por los módulos multisensor, al tiempo que consume menos energía de la batería que otros protocolos. El principal inconveniente de este protocolo es su alcance, ya que a partir de unos diez metros entre dos dispositivos BLE en áreas interiores la comunicación deja de ser fiable. Finalmente, los datos se transfieren desde el gateway a la capa cloud utilizando el protocolo MQTT.

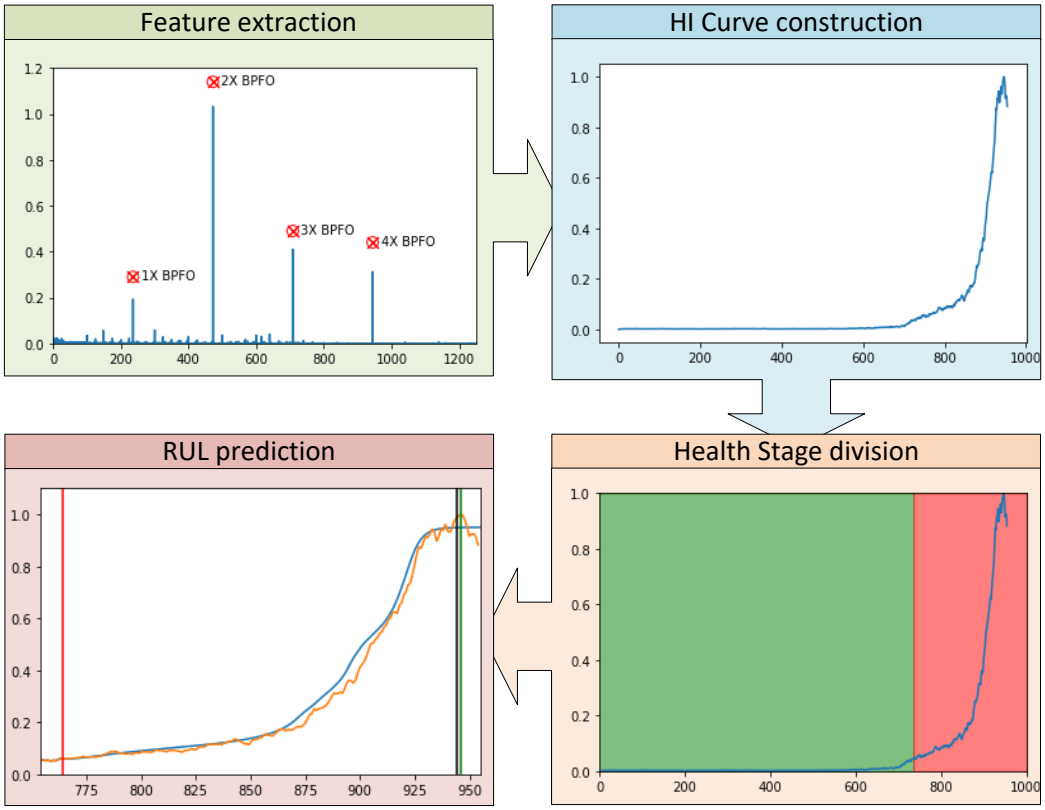


Fig. 4: Metodología para la predicción del RUL

IV. Plan experimental

A. Objetivos

1. Pruebas que permitan diseñar reglas de dimensionamiento del sistema de monitorización en función de las capacidades tanto de los sensores como los gateways.
2. Despliegue de un prototipo del sistema de monitorización de motores eléctricos en una planta industrial.
3. Visualización en tiempo real de los datos monitorizados y construcción de un histórico de datos de monitorización en la nube.
4. Detección de anomalías de funcionamiento y generación de las correspondientes alertas como ayuda a la toma de decisiones por parte de los técnicos de mantenimiento.
5. Diagnosis de las anomalías de funcionamiento detectadas.
6. Predicción del tiempo restante de vida útil (Remaining Useful Lifetime, RUL) para diferentes tipos de fallos de los rodamientos de los motores.

B. Metodología

A continuación se indica el enfoque seguido para llevar a cabo cada uno de los objetivos propuestos.

1. Pruebas del sistema de monitorización: Se analizó la distancia máxima admisible entre sensores y gateways para un nivel de señal de comunicación (RSSI) adecuado, así como la capacidad de los gateways para soportar el procesamiento concurrente de los datos proporcionados por múltiples sensores.
2. Despliegue del prototipo del sistema de monitorización: Se ha seleccionado una planta industrial de producción de cemento para el despliegue. Los datos a monitorizar son principalmente magnitudes de vibración y componentes del espectro de frecuencia de dichas vibraciones (frecuencias relevantes y sus amplitudes correspondientes).
3. Visualización de datos en tiempo real: Se enviaron los datos relevantes desde los gateways hacia la nube y se almacenaron allí como histórico. Para la visualización de datos se ha desarrollado un dashboard web mediante la herramienta Grafana.
4. Detección de anomalías de funcionamiento: La detección de anomalías se ha hecho en base al análisis de series temporales de los datos monitorizados mediante scripts en Python. Para ello se han tenido en cuenta las recomendaciones de la norma internacional ISO 20816.
5. Diagnosis de anomalías de funcionamiento: La diagnosis se ha llevado a cabo a partir de la literatura científica y las normas internacionales ISO 13373 e ISO 17359.
6. Predicción del tiempo restante de vida útil (RUL): Se ha desarrollado una técnica de predicción de fallos en motores que permite identificar fallos en las carreras exterior e interior de sus rodamientos y predecir el RUL de los mis-

mos. La técnica se divide en los cuatro pasos principales mostrados en la figura 4: extracción de características, construcción de la curva de índice de salud (HI), división en zonas de salud (sin riesgo y con riesgo) y predicción de RUL de los rodamientos. Esta se realiza mediante la predicción de la curva HI mediante redes neuronales recurrentes (RNN), que son muy efectivas para el modelado de secuencias. Se han desarrollado modelos de memoria a corto y largo plazo bidireccional (BiLSTM) para predecir el RUL. BiLSTM es una forma de RNN que se utiliza típicamente para extraer la dependencia a largo plazo de los datos de muestra de entrada. Está compuesto por dos capas LSTM, una utilizada para la propagación hacia adelante y la otra utilizada para la propagación hacia atrás, permitiendo el flujo en ambas direcciones. Para el entrenamiento y validación de los modelos se han empleado conjuntos de datos de dominio público relevantes sobre fallos en motores, tales como IMS, FEMTO y XJTU-SY.

V. Resultados

A continuación se resumen los resultados obtenidos en relación a los objetivos planteados:

A. Pruebas del sistema de monitorización

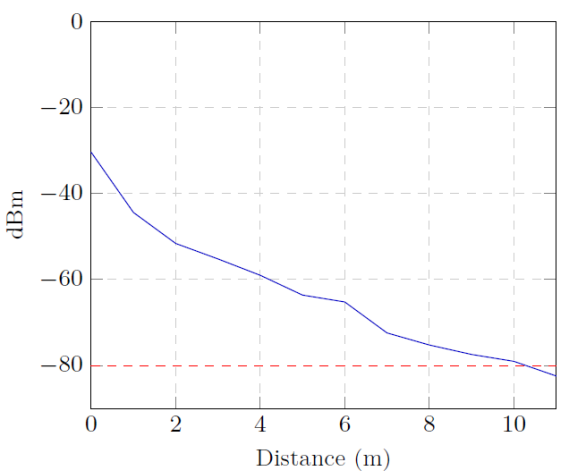


Fig. 5: RSSI según la distancia multisensor-gateway

La velocidad de transmisión entre sensores y gateways depende principalmente de la intensidad de la conexión de red (BLE). Se analizó la distancia máxima admisible entre un módulo multisensor y un gateway. Para ello se posicionó un gateway en un punto fijo y se alejó progresivamente el módulo del gateway, sin obstáculos entre ellos, registrando el nivel de RSSI entre ambos para cada posición. El nivel de RSSI se calculó como valor medio de varias mediciones para aumentar su fiabilidad. Para que una señal se considere adecuada y proporcione un reenvío de paquetes confiable debe tener un nivel de RSSI por encima de -80 dBm. Según los resultados mostrados en la figura 5, el nivel de RSSI está por encima de -80 dBm para distancias inferiores a 11 metros en un entorno sin interferencias.

Además de una distancia máxima, el gateway utilizado impone algunas restricciones en la cantidad de sensores que puede manejar un solo gateway. Según pruebas empíricas, el número máximo de módulos de múltiples sensores que puede gestionar simultáneamente un gateway es de 15. Los módulos adicionales no logran conectarse al gateway. Por lo tanto, un gateway puede manejar hasta 15 módulos multisensor simultáneamente, siempre que estén dentro de un rango de 11 metros.

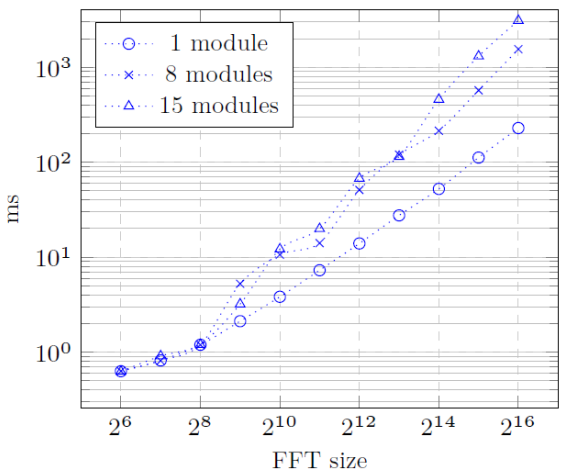


Fig. 6: Tiempo de computación de la FFT

Por último se analizó la capacidad de los gateway para soportar el procesamiento concurrente de los datos recibidos desde varios sensores. La figura 6 muestra el tiempo promedio necesario para computar la FFT en el caso más defavorable, cuando el gateway está procesando de forma simultanea los datos recibidos desde todos los sensores. El tiempo de computación de la FFT crece con el tamaño de la propia FFT y con el número de sensores.

B. Despliegue del prototipo del sistema de monitorización

En la figura 7 se muestra el despliegue realizado en las dos zonas de la planta industrial donde operan los dos motores seleccionados, motores que impulsan ventiladores de rechazo encargados de retornar a los molinos aquellos trozos de material que superan un tamaño mínimo. El prototipo desplegado consta de un sensor en cada uno de los motores, dos gateways y una interfaz 4G que permite la comunicación de uno de los gateways con el exterior. Se montó un sensor de vibración encima del rodamiento delantero de cada motor, cada uno de los cuales se comunica de forma inalámbrica con el gateway situado en su proximidad. Los gateways, que proporcionan comunicación con la nube, están integrados en una red mallada con protocolo BATMAN sobre WiFi, de modo que solo uno de ellos necesita establecer comunicación con el exterior mediante una interfaz 4G.

C. Visualización de datos en tiempo real

En la figura 8 se muestra el monitor de vibración de uno de los motores durante una semana. Se indica

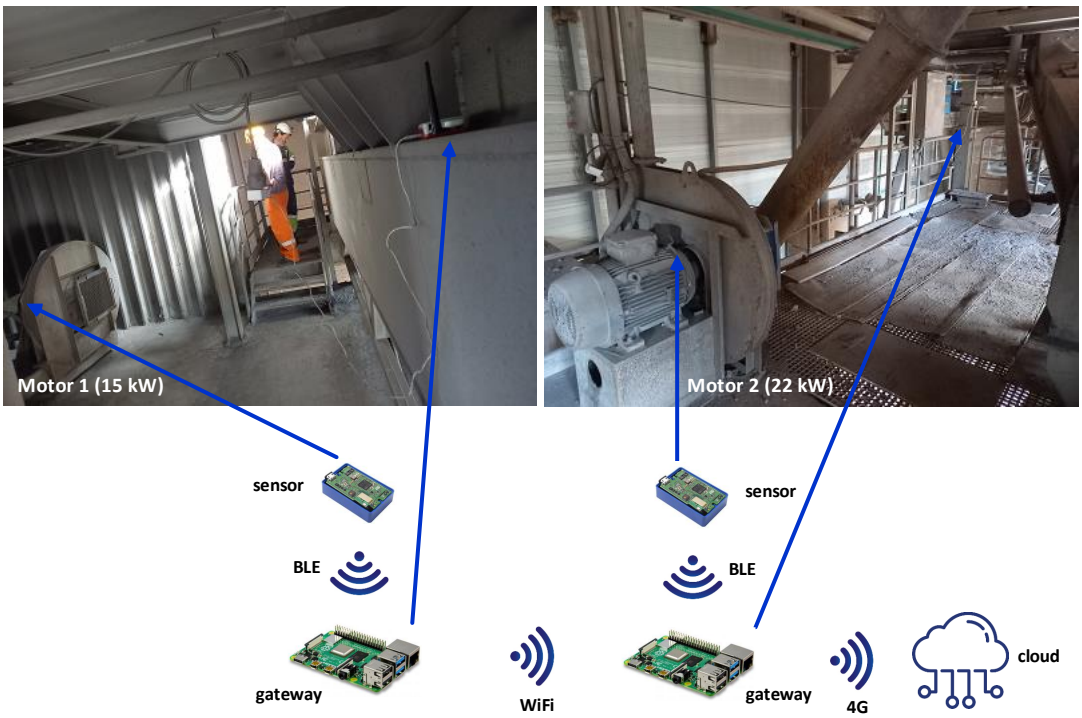


Fig. 7: Caso de estudio en una planta industrial cementera

el nivel de vibración en velocidad (mm/s) para cada uno de los 3 ejes (axial, radial horizontal y radial vertical) junto con los umbrales de seguridad definidos en la noma ISO 20816 [1].

D. Detección de anomalías de funcionamiento

La detección de anomalías de funcionamiento se realiza tanto en el dominio del tiempo como de la frecuencia, a partir del análisis de tendencia en las series temporales de la variables monitorizadas.

En la figura 8 se aprecia un nivel de vibración en el eje X (radial) bastante elevado en el motor 2, superando el umbral de seguridad indicado por la norma ISO 20816. Esta anomalía de funcionamiento detectada por el sistema generó la correspondiente alerta para los técnicos de mantenimiento de la planta. De hecho, este motor se encuentra en una fase más avanzada dentro de su ciclo de mantenimiento preventivo de 40000 horas (90 % del ciclo) que el otro motor (40 % del ciclo).

En el caso del dominio de la frecuencia las variables utilizadas son las amplitudes de las frecuencias resultantes de aplicar la transformada rápida de Fourier (FFT) a conjuntos de muestras de vibraciones correspondientes a varios segundos de operación del motor. En la figura 9 se muestra un espectro correspondiente al eje X del motor 2.

Generando espectros de forma continua se puede analizar la tendencia de las amplitudes de ciertas frecuencias características y así detectar la aparición temprana de fallos de funcionamiento asociados a dichas frecuencias. En la figura 10 se muestra la evolución creciente (durante 4 meses) de la amplitud de la frecuencia de rotación (RF) del motor 2 en el eje X (radial), debida a que es el motor que se encuentra en la fase más avanzada dentro de su ciclo de mantenimiento preventivo. Actualmente se está tratando de determinar el umbral de amplitud a partir del cual generar la correspondiente alerta de anomalía de funcionamiento.

E. Diagnosis de anomalías de funcionamiento

Se han generado históricos de datos de monitorización de 10 meses para cada uno de los motores en los que se incluyen las aceleraciones de los 3 ejes del motor, realizando capturas cada 8 minutos.

El análisis de tendencias de las amplitudes correspondientes a frecuencias características asociadas a posibles fallos de funcionamiento es la base de la diagnosis que se está llevando a cabo. Se están analizando las tendencias de las amplitudes correspondientes a frecuencias características y sus primeros múltiplos (2X/3X/4X) de los motores, del ventilador que actúa como carga y de los rodamientos, al ser estos una de la principales causas de fallos en los motores. A continuación se indican dichas frecuencias características:

- Rotación del motor (RF)
- Rotación de las aspas (8) del ventilador (VPF)
- Carrera interna del rodamiento (BPFI)
- Carrera externa del rodamiento (BPFO)
- Bola del rodamiento (BSF)
- Jaula del rodamiento (FTF)

F. Predicción del tiempo restante de vida útil (RUL)

Los modelos de predicción de RUL propuestos se han entrenado con el conjunto de datos IMS y después validado con los conjuntos IMS, FEMTO y

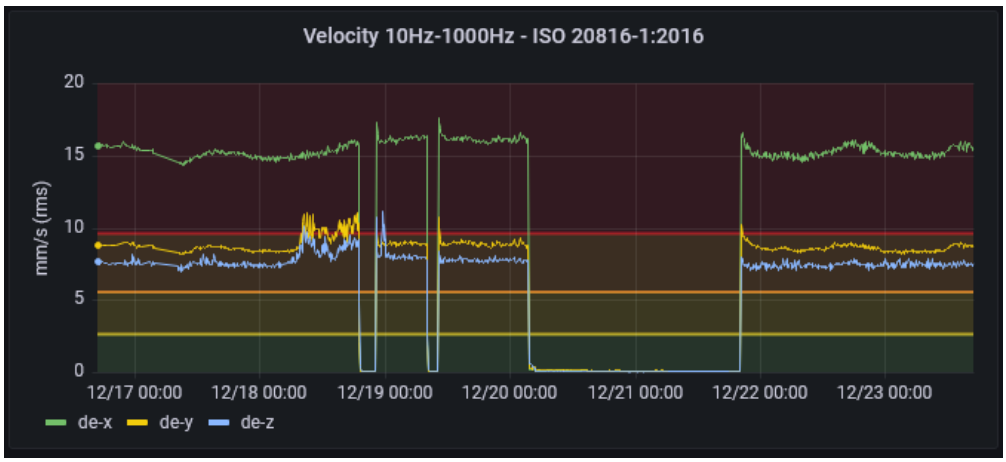


Fig. 8: Monitor del motor 2

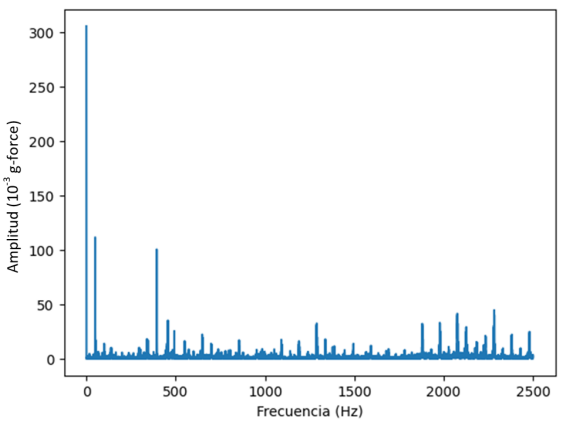


Fig. 9: Espectro de frecuencia

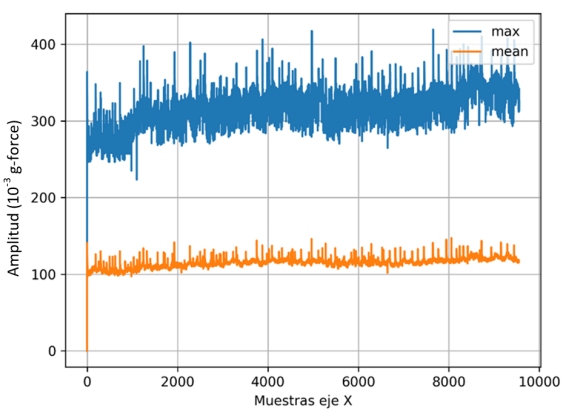


Fig. 10: Tendencia de amplitud de frecuencia característica.

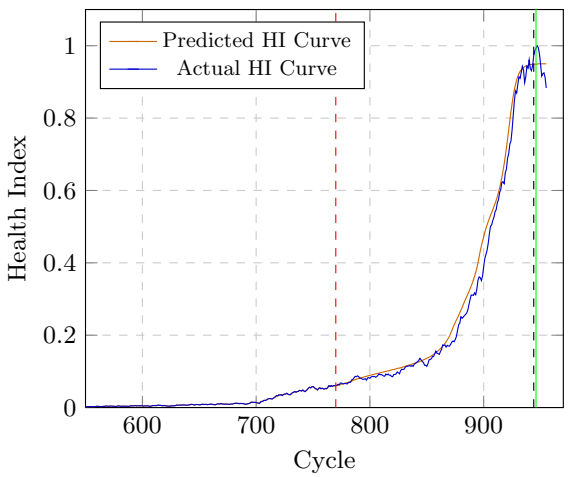


Fig. 11: Predicción de RUL durante el entrenamiento con IMS

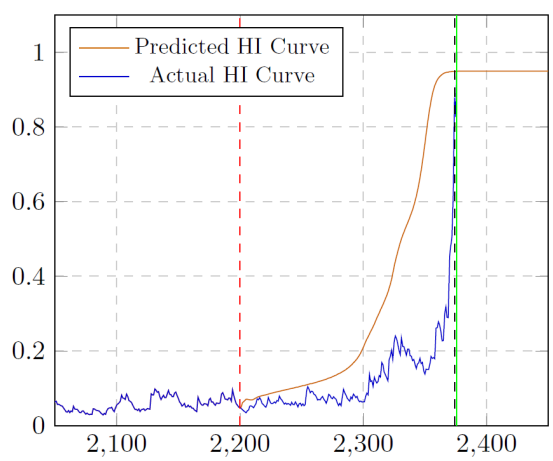


Fig. 12: Predicción de RUL para FEMTO

XJTU-SY sin necesidad de volver a entrenarlos, obteniendo resultados precisos en todos ellos y demostrando su robustez con diferentes motores y condiciones de trabajo. En particular, los resultados obtenidos con los conjuntos de datos FEMTO y XJTU-SY son más precisos que los de trabajos anteriores entrenados y validados con el mismo conjunto de datos.

En las figuras 11, 12 y 13 se muestran las predicciones de RUL para fallos en la carrera exterior de los rodamientos con los conjuntos de datos de entrenamiento (IMS) y de validación (FEMTO y XJTU-SY) respectivamente. La línea roja discontinua vertical corresponde al instante en el que se realiza la predicción, la línea verde corresponde al RUL real y la línea negra discontinua corresponde a la predicción de RUL.

VI. Conclusiones y trabajo futuro

Se ha presentado el diseño, implementación, despliegue y prueba de un sistema IIoT de bajo costo para la monitorización en tiempo real del estado de los motores eléctricos. El sistema está diseñado utilizando componentes de hardware de bajo costo,

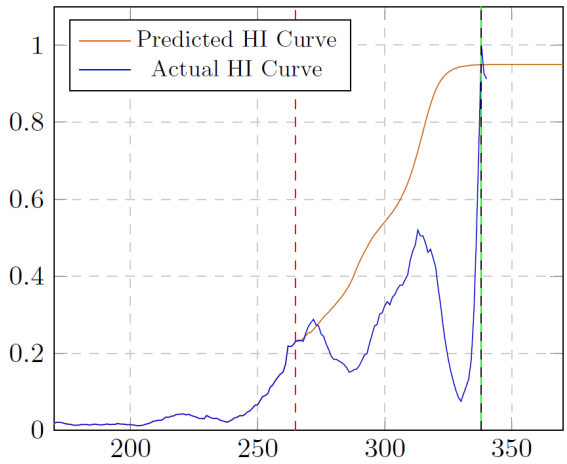


Fig. 13: Predicción de RUL para XJTU-SY

como módulos inalámbricos de múltiples sensores y computadoras de placa única como gateways, junto con software de código abierto y servicios abiertos en la nube. Los módulos recopilan datos de vibración en tiempo real de los motores eléctricos. Se han realizado análisis de vibración en los dominios de tiempo y de frecuencia para detectar y diagnosticar anomalías de funcionamiento, facilitando así el mantenimiento predictivo de los motores eléctricos.

Se ha diseñado, implementado, entrenado y probado una nueva técnica de predicción de fallos en motores que identifica fallos en la carrera exterior e interior de los rodamientos y predice su tiempo restante de vida útil (RUL) utilizando modelos de aprendizaje automático basados en redes neuronales recurrentes. Se ha demostrado la robustez de los modelos propuestos al predecir el RUL de los rodamientos en diferentes motores y condiciones de operación (frecuencia del eje, carga, tipo de rodamiento) sin necesidad de reentrenar o ajustar los modelos.

La investigación actual está orientada hacia el uso de TabPFN, un modelo Transformer preentrenado de muy reciente aparición (2022) que puede realizar clasificación supervisada tabular para pequeños conjuntos de datos en menos de un segundo, no requiere ajuste de hiperparámetros y es competitivo con los métodos de clasificación de última generación. Los resultados preliminares obtenidos en la aplicación de TabPFN a la clasificación de fallos en máquinas rotatorias con un bajo número de muestras ya mejoran los resultados de otros algoritmos de clasificación tradicionales como Random Forest, Label Propagation o XGBoost.

Agradecimientos

La investigación llevada a cabo en este trabajo ha sido financiada por el Plan Nacional de Investigación, Desarrollo e Innovación de España, la Universidad de Oviedo, el Instituto Universitario de Tecnología Industrial de Asturias (IUTA) y el programa de ayudas predoctorales Severo Ochoa para la formación en investigación y docencia de Asturias.

Referencias

[1] L. Magadán, F.J. Suárez, J.C. Granda, and D.F. García, "Low-cost industrial iot system for wireless monitoring of electric motors condition," *Mobile Networks and Applications*, pp. 1–10, 2022.

[2] V.C. Gongora and G.P. Hancke, "Industrial wireless sensor networks: Challenges, design principles and technical approaches," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 4258–4265, 2009.

[3] L.D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[4] L. Magadán, F.J. Suárez, J.C. Granda, and D.F. García, "Clustered wsn for building energy management applications," *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Science and Technologies for Smart Cities. SmartCity 360 2021.*, vol. 442, pp. 673–687, 2021.

[5] R.K. Naha et al., "Fog computing: Survey of trends, architectures, requirements and research directions," *IEEE Access*, vol. 6, pp. 47980–48009, 2018.

[6] S. Yang, "Iot stream processing and analytics in the fog," *IEEE Communications Magazine*, vol. 51, no. 8, pp. 21–27, 2017.

[7] K. Ágoston, "Fault detection of the electrical motors based on vibration analysis," in *8th International Conference Interdisciplinarity in Engineering*, 2014, pp.–.

[8] Y. Merizalde, L. Hernández-Callejo, and O. Duque-Perez, "State of the art and trends in the monitoring, detection and diagnosis of failures in electric induction motors," *Energies*, vol. 10, no. 7, pp. 1056, 2017.

[9] J. Wang et al., "Sensor data based system-level anomaly prediction for smart manufacturing," in *IEEE International Congress on Big Data*, 2018, pp.–.

[10] International Organization for Standardization, *ISO 20816-1:2016, Mechanical vibration — Measurement and evaluation of machine vibration — Part 1: General guidelines*, 2016.

[11] International Organization for Standardization, *ISO 13373-1:2002. Condition monitoring and diagnostics of machines — Vibration condition monitoring. Part 1: General Procedures*, 2002.

[12] International Organization for Standardization, *ISO 13373-3:2015. Condition monitoring and diagnostics of machines — Vibration condition monitoring. Part 3: Guidelines for vibration diagnosis*, 2015.

[13] International Organization for Standardization, *ISO 13373-2:2016. Condition monitoring and diagnostics of machines — Vibration condition monitoring. Part 2: Processing, analysis and presentation of vibration data*, 2016.

[14] International Organization for Standardization, *ISO 13373-9:2017. Condition monitoring and diagnostics of machines — Vibration condition monitoring. Part 9: Diagnostic techniques for electric motors*, 2017.

[15] International Organization for Standardization, *ISO 17359:2018. Condition monitoring and diagnostics of machines — General guidelines*, 2018.

[16] M. Paolanti et al., "Machine learning approach for predictive maintenance in industry 4.0," in *14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, 2018, pp.–.

[17] L. Magadán, F.J. Suárez, J.C. Granda, F.J. delaCalle, and D.F. García, "A robust health prognostics technique for failure diagnosis and the remaining useful lifetime predictions of bearings in electric motors," *Applied Sciences*, vol. 13, no. 4, pp. 2220, 2023.

[18] H. Qiu, J. Lee, J. Lin, and G. Yu, "Wavelet filter-based weak signature detection method and its application on rolling element bearing prognostics," *Journal of sound and vibration*, vol. 289, no. 4-5, pp. 1066–1090, 2006.

[19] P. Nectoux, R. Gouriveau, K. Medjaher, E. Ramasso, Brigitte B. Chebel-Morello, N. Zerhouni, and C. Varnier, "Pronostia: An experimental platform for bearings accelerated degradation tests," in *IEEE International Conference on Prognostics and Health Management, PHM'12.* IEEE Catalog Number: CPF12PHM-CDR, 2012, pp. 1–8.

[20] B. Wang, Y. Lei, N. Li, and N. Li, "A hybrid prognostics approach for estimating remaining useful life of rolling element bearings," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 401–412, 2018.

Optimization strategies for energy-aware computation offloading in the Extreme Edge of Internet of Things

Paula González[1], Gabriel Mujica[1], and Jorge Portilla[1]

Abstract— **The proliferation of Internet of Things (IoT) devices has led to an exponential growth in data generation, placing significant computational demands on resource-constrained IoT devices at the extreme edge. To address these challenges, computation offloading has emerged as a promising technique for enabling energy-efficient and resource-efficient computing in IoT networks. This paper focuses on the optimization of computation offloading in the extreme edge of IoT, specifically addressing the bandwidth allocation aspect using iterative and genetic algorithms. Through simulations and comparative analysis, we evaluate the performance of the proposed algorithms in terms of energy efficiency, and overall system performance. The experimental tests demonstrate that the iterative and genetic algorithms yield improved bandwidth allocation schemes, effectively reducing energy consumption. The results highlight the potential of genetic algorithms over iterative algorithms to improve energy efficiency and computational performance in resource-constrained IoT environments.**

Key words— **Computation Offloading, Extreme Edge of IoT, Wireless Sensor Networks, Edge Computing.**

I. Introduction

The Internet of Things (IoT) paradigm consists of a set of devices that collect and exchange data, connected to each other via the Internet. The IoT encompasses the collection, processing and analysis of the collected data for its subsequent use in a wide variety of applications such as smart cities or healthcare monitoring. Advances in sensor electronics and embed systems have enabled the size, prize and power consumption of wireless devices to be reduced and scalable, which has led to the proliferation of IoT networks [1].

However, these devices still impose constraints in terms of battery life, processor performance or storage capacity. Computation offloading has emerged as an attempt to alleviate these restrictions. This paradigm is based on the presence of servers with high-processing capacity close to the network, which share computing resources with the rest of devices [2]. For mobile systems, offloading has the potential to save energy and improve performance. However, this usually depends on a number of parameters, such as the bandwidth of the network or the amount of data exchanged over the network, that need to be considered when implementing offloading.

With regard to the infrastructure on which computation offloading can be carried out, there are several alternatives, including Cloud Computing, Fog Computing and Mobile Edge Computing. Due to problems of latency, power consumption and storage and processing limitations, the use of Cloud Computing (CC) for such processing is not an effective solution. Instead, the trend in recent years has been to move data processing closer to the edge of the network. This has given rise to the fog computing and Mobile Edge Computing (MEC) paradigms.

Fog computing consists on bringing CC to the edge of the network using fog nodes or cloudlets [3]. In Mobile Edge Computing (MEC), serves or devices at the edge of the network share processing capacities, bypassing the connection to the cloud and focusing on device-device connections. The proximity between data sources and Base Stations in MEC has several advantages over the traditional cloud-based computing paradigm in terms of response time and energy consumption [4].

In this article, we will focus on the study of optimisation algorithms for bandwidth allocation on computational offloading in networks with MEC infrastructure, but particularly on the Extreme Edge of IoT [5][6] where resource-constrained sensor devices participate in the wireless sensor network. The main objective for the proposed algorithms is to reduce energy consumption on dynamic mobile networks when varying the number of devices and bandwidth shared by the base station of the network with the participant sensor nodes. We propose five different algorithms that can be used to select the best computational offloading strategy at any point in the life of a network to reduce the battery consumption of the nodes in the network.

The rest of the article is organized as follows. Section II overviews the problem of optimizing the resource and bandwidth distribution in IoT sensor devices from the point of view of network lifetime, by applying the concept of computation offloading. Section III details the proposed offloading strategies to tackle the optimization problem set, while Section IV outlines the main simulation results and discussion of comparing the proposed solutions. Finally, Section V presents the conclusions and future lines of research for the proposed work.

II. Problem description

This paper compares various task offloading algorithms in the Mobile-Edge Computing (MEC) infrastructure. The main goal is to minimise battery con-

[1]Centro de Electrónica Industrial, Universidad Politécnica de Madrid, Spain, e-mail: {paula.gonzalezdedu, gabriel.mujica, jorge.portilla}@upm.es.

sumption while considering the network bandwidth when the number of nodes of the network increases. An additional goal is to make the battery consumption of the nodes as homogeneous as possible, favouring greater offloading in the nodes with lower batteries. In this way, if we consider that the network lasts until the first node runs out of battery, we will maximise the lifetime of the network. The way to achieve this goal varies depending on whether the algorithm is iterative or metaheuristic and is explained in more detail in Section III.

The problem definition starts with the description of the network architecture and its parameters, followed by the formulation of the optimisation problem.

A. Mobile Edge Computing Network architecture

We consider a local network of N geographically distributed IoT edge nodes $I = \{ I_1, ..., I_N \}$ and a Base Station (BS), which are connected through a local network with a star topology. A similar system model is considered in [7]. Nodes are described by static and dynamic parameters. During the initialisation the nodes are set with the following invariant parameters:

$$I_d = (Q_i, F, T_R, DC, T_C, P_C, Bat),\ d = 1, ..., N$$

- Q_i denotes the number of different *offloading levels* that the IoT sensor node can handle. In our study, it is assumed that all of the nodes are identical and perform the same task. This task can be partitioned into several parts, each of which can be processed individually. This allows these parts to be sent to the BS for processing. Each level corresponds to a different partitioning of the task. The parameters of the levels used in the simulations carried out for the purpose of this work are given in Table III.
- F denotes the node's clock frequency.
- T_R is the data transfer rate between the node and the BS, which needs to be set in accordance with the network characteristics.
- DC represents the duty cycle of the node.
- T_C denotes the energy consumption during the data transmission from the node to the BS.
- P_C denotes the processing battery consumption of the nodes.
- Bat represents the initial battery of the nodes.

In each iteration, the following node parameters are updated:

$$N_d = (b_{d_i}, E_d), \quad d = 1, ..., N$$

- b_{d_i} represents the the number of bandwidth bytes occupied by node d working at level i.
- E_d is the remaining battery of sensor node d.

The Base Station processes the sent data and sets the offloading strategy of the network. The available network bandwidth is quantified by the maximum number of bytes the BS can share at each iteration. This architecture is also used in [8][9].

B. Optimization Problem

The problem considered in this work belongs to the class of Generalised Assignment Problems (GAP). GAP consist in assigning a set of tasks to a set of agents with limited resources, taking into account a minimum total cost [10]. In this work, the resource is the bandwidth of the Base Station, which has to be allocated to the IoT nodes to minimise the total energy cost. All of this applied to a network of IoT nodes sharing computational resources with a Base Station is called task offloading [11].

The task offloading optimization problem can be formulated in many ways, as surveyed in [12]. Equations 1 and 2 represent, respectively, the objective and the constraint of the optimisation problem to be formulated. We will base on them to obtain the offloading level i of each IoT device at run-time.

$$Optim.goal : min(battery\,consumption) \tag{1}$$

$$Constraint : \sum_{d=1}^{N} b_d \leq B_M \tag{2}$$

We need to define the following parameters for our considered problem:

- p_{d_i} is the overall power consumption of device d operating at offloading level i. It encompasses the on-board processing consumption p_{ob} and the data transmission consumption p_{tr} (at the rate TR). These are obtained as follows:

$$p_{ob} = P_C \frac{Cycles}{F} \tag{3}$$

$$p_{tr} = T_C \frac{b_{d_i}}{TR} \tag{4}$$

$$p_d = p_{ob} + p_{tr} \tag{5}$$

- T_R has been set in accordance with the IEEE Standard for Low-Rate Wireless Networks (IEEE 802.15.4) in the 2.4GHz band.

III. Proposed Algorithms

Once the problem has been defined, algorithms are generated to find the most optimal bandwidth allocation and offloading management routines. These algorithms share the parameter initialisation routine and are scalable, allowing the number of nodes in the network and the bandwidth to be varied. In addition, all level-related parameters, such as the number of levels or the number of processing cycles needed in level, can be easily modified.

The solution can be divided into two parts: 1) finding the best bandwidth allocation and, 2) implementing the solutions to achieve balanced consumption. The generated algorithms can therefore be divided into two groups, the Iterative Bandwidth Allocation (IBA) algorithms and the methaeuristic algorithms, depending on their bandwidth allocation routines.

A. Iterative Bandwidth Allocation algorithms

IBA algorithms use iterative strategies to distribute the available bandwidth among the nodes in such a way that the maximum possible bandwidth is used, thereby minimising the battery consumption of the IoT node. In order to carry out this distribution, two different methods have been taken into consideration, as described in the IBA1 and IBA2 algorithms. The input to each of them is a list of the bandwidth and battery consumption of each level, ordered from highest to lowest, such that the last level has the highest bandwidth and lowest consumption at the node.

- *IBA1*: The way IBA1 finds an optimal solution is as follows: First, it identifies the lowest consumption level. It then generates a solution in which all nodes are set at that level. If this solution exceeds the bandwidth, it cycles through the nodes, downgrading one level at a time until it is satisfied. This process is represented in Algorithm 1.
- *IBA2*. This algorithm uses a different bandwidth allocation mechanism to find the solution, based on finding the maximum number of nodes that can be assigned to each level. It first calculates the maximum number of nodes that can be at the lowest consumption level and then distributes the remaining bandwidth to the subsequent levels in the same way. The process followed by this algorithm is similar than the IBA1.

Once both algorithms have found the optimal solution, they are implemented in such a way that the levels of each node are switched cyclically to achieve homogeneous network consumption.

Algorithm 1 Bandwidth Allocation IBA1

Ensure: *sol*: List of the bandwidth allocation with the lowest consumption

```
sol ← []
min ← None
min_idx ← None
lev_cons ← get_consumption(levels)
for idx, cons in ENUMERATE(lev_cons) do
    if (min is None or cons < min) then
        min_idx ← idx
    end if
end for
sol ← n * [levels[min_idx]]
if SUM(sol) > BW then
    nd ← 0
    while SUM(sol) > BW do
        if nd == n then
            nd ← 0
        end if
        sol ← DOWNGRADE_LEVEL(sol, nd)
        nd ← nd + 1
    end while
end if
return sol
```

B. Metaheuristic Algorithms

Metaheuristic algorithms apply computational intelligence methods with advanced problem-solving capabilities to solve complex optimisation problems [13].

In this paper, we will focus on a type of metaheuristic algorithm that belongs to the family of nature-inspired algorithms: the Genetic Algorithms (GA), first introduced by Goldberg [14] and Holland [15]. There are a variety of applications for which genetic algorithms are useful in MEC other than bandwidth allocation, as seen in [16] where they use a GA similar to those presented in this paper for sequential task scheduling.

Genetic algorithms are inspired by Darwin's theory of biological evolution to solve problems. The first step is creating an initial population and selecting the fittest chromosomes to reproduce and mutate. The result is a new generation of better-adapted chromosomes. The chromosomes represent a set of parameters (genes) which define a proposed solution. In our case, the chromosomes are a list of the bandwidths to be occupied by each node of the network, in other words, each chromosome consists of an offloading strategy and its size matches the number of nodes in the network. Table I shows the structure of the chromosomes.

Table I: Chromosome Description

b_{1_i}	b_{2_i}	...	b_{N_i}

The fitness function must be defined in advance along with the chromosomes. It determines the fitness of an individual based on a fitness score. This fitness score sets the probability that an individual will reproduce. The effectiveness of the different fitness functions proposed in this work will be evaluated in the following section.

The initial population must be randomly generated to achieve maximum diversity. However, in our work, even though the initial population is generated at random, all the chromosomes are forced to satisfy the constraint described in Eq. 2. The size of the population must be large enough to ensure said diversity and the choice of this parameter is discussed in the simulation section.

The next step is the selection of the best individuals for mating. As shown in [17], there are several ways to implement the selection of the individuals. After considering the various options, we decided that selection by tournament was best suited to the problem at hand. The first step in Tournament Selection is to randomly select a set of individuals. These individuals are ranked according to their calculated fitness, and then the fittest one is selected for mating. The number of individuals selected for each tournament is set to two. Since the descendants come from the crossing of two parents, this process is repeated twice for each time a new chromosome is to be generated.

Once the progenitors have been selected, the mating process will begin. The mating is done by single-

point crossover. The crossover point is chosen to be in the middle of the parents so that the two children generated each have one-half of each parent. For the problem formulated, none of the solutions must exceed the bandwidth. Thus, after the generation of the children, they are checked for compliance with this constraint and the offspring that do not satisfy it are discarded. The process of selecting and mating the parents continues until a new generation of the given population size is obtained.

The last step is mutation. The mutation is used to maintain diversity in the population, preventing the solution from stagnating at a local optimum. As in Darwin's theory mutation is done to a random number of individuals. The mutation rate parameter is set beforehand and is usually around 10%. In our algorithms, the probability of an individual being selected for mutation is an adjustable parameter. Once an individual has been selected for mutation, the mutation routine selects a random node and changes its bandwidth to that of another random level. The mutated individual may exceed the bandwidth. Given this case, the mutated individual is discarded and the original individual is reintroduced into the population. This implies a reduction in the previously set mutation rate. It has been experimentally determined that the number of mutations rejected due to non-compliance with this constraint does not exceed 30%. This is taken into account in the selection of the GA simulation parameters, where it has been decided to set a higher value for the mutation rate to compensate this effect.

The proposed algorithms incorporate elitism. Elitism is an operational feature of GAs that provides a means of reducing genetic drift by ensuring that the best chromosome is passed without modification to the next generation [18]. In this way, the best individual of the current generation called the elite, is passed on to the next generation. There are several ways in which the elite can be incorporated. In this paper, we will consider two of them. GA1 and GA2 add the elite to the next generation when the mutation of its offspring is complete. GA3 replaces the worst individual in the next generation with the elite.

The new generation is completed after elitism is carried out. It must be large enough to allow the GAs to reach the best solution and can be determined empirically. Another option is to add a termination condition that stops the creation of new generations when a high percentage of the population satisfies it. To ensure that the number of generations is sufficient, all three proposed algorithms have a termination condition that stops the creation of new generations when more than 80% of the population has the same fitness.

The solution given by the GAs is the chromosome with the best fitness in the last generation. In the majority of cases, there will be more than one individual with the best fitness score. The selection of the final solution from these individuals will be discussed below.

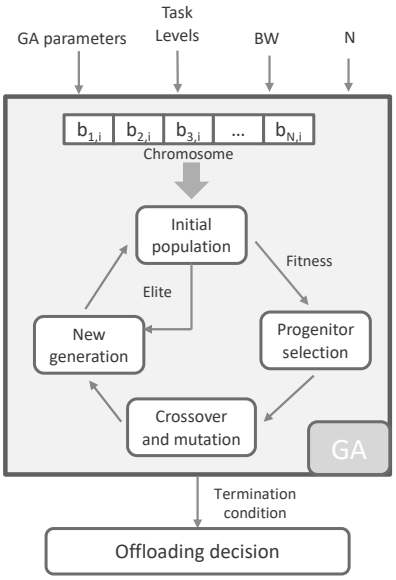


Fig. 1: Outline of the process followed by the GAs.

Three different genetic algorithms have been generated: GA1, GA2 and GA3. Figure 1 represents a general outline of the process followed by the GAs. As mentioned above, GA1 and GA2 share the same elitism method: the first individual in the list with the highest fitness is selected as the elite and is added directly to the next generation. They differ in their fitness function. The objective of GA1 is to minimise the average consumption of the network. Therefore, the fitness of each chromosome is the average of the batteries that the nodes would have if this solution were chosen. In GA2, a more homogeneous consumption of the network is prioritised. The fitness of the chromosomes has been set as the minimum battery that the network would have if this solution were chosen. GA3 is an update to GA1 that purports to achieve a lower standard deviation between node batteries by modifying the elitism method. Instead of selecting the elite as the first chromosome in the list with the highest fitness, the elite is selected as the one with the highest minimum network battery among all individuals with the highest fitness. In addition, to get to the best solution faster, the elite replaces the worst individual in the next generation.

Genetic algorithms GA1 and GA2 are executed through the two functions represented in Algorithm 2, and GA3 in Algorithm 3. The simulation of genetic algorithms is different from the simulation of iterative algorithms. Instead of finding a single solution, each step of the simulation executes the GAs to find the best solution with the network as it is. This is because GAs are designed for dynamic networks, where nodes do not start with the same battery and the number of nodes can increase.

IV. Simulation and results comparison

Several simulations were run to compare the effectiveness of the algorithms in terms of average battery consumption, standard deviation and execution

Algorithm 2 Genetic Algorithm for GA1 and GA2

```
procedure NEXTGENERATION(currentGen, mutation_rate)
    fitness_list ←GET_ALL_FITNESS(currentGen)
    best_idx ←GET_BESTFIT_IDX(fitness_list)
    elite ←currentGen[best_idx]
    children ←MATING(currentGen, fitness_list)
    nextGen ←MUTATE_POPULATION(children, mutation_rate, elite)
    return nextGen
end procedure

procedure GENETICALGORITHM(mutation_rate, popSize, generations)
    pop ← GENESIS(popSize)
    for i ← 1 to generations do
        pop ← NEXTGENERATION(pop, mutation_rate)
        fitness_list ←GET_ALL_FITNESS(pop)
        max ← None
        for num in fitness_list do
            if max is None or max < num then
                max ← num
            end if
        end for
        indice ← FIND_INDEX(fitness_list, max)
        if len(indice) ≥ popSize * cond_fin then
            break
        end if
    end for
    best_sol ← []
    for i ← 1 to len(indice) do
        best_sol.append(pop[indice[i]])
    end for
    return best_sol[0], len(indice)
end procedure
```

Algorithm 3 Genetic Algorithm for GA3

```
procedure NEXTGENERATION(currentGen,fitness_list, mutation_rate, elite, elite_fit)
    children ←MATING(currentGen, fitness_list)
    nextGen ←MUTATE_POPULATION(children,mutation_rate)
    next_fitness_list ← get_all_fitness(nextGen)
    worst_idx ← get_worstfit(next_fitness_list)
    nextGen[worst_idx] ← elite
    next_fitness_list[worst_idx] ← elite_fit
    return nextGen, next_fitness_list
end procedure

procedure GENETICALGORITHM(mutation_rate, popSize, generations)
    pop ← GENESIS(popSize)
    fitness_list ← GET_ALL_FITNESS(pop)
    for i ← 1 to generations do
        elite, elite_fit, repetidas ←FIND_ELITE(pop, fitness_list)
        pop, fitness_list ←NEXTGENERATION(pop, fitness_list, mutation_rate, elite, elite_fit)
        if len(indice) ≥ popSize * cond_fin then
            break
        end if
    end for
    return nextGen, next_fitness_list
end procedure
```

time.

The network parameters used to perform the simulations are shown in the Table II.

T_R has been set in accordance with the IEEE Standard for Low-Rate Wireless Networks (IEEE 802.15.4) in the 2.4GHz band.

Table II: Simulation Parameters

Parameter	Value
Frequency	8 MHz
Transmission rate (TR)	250 Kbps
Duty cycle (DC)	50 ms
Transmission consumption (T_C)	5 mA
Processing consumption (P_C)	40 mA
Battery	12000 mAh

As for the offloading levels, it was decided to divide the task into 5 different ways, so that the corresponding parameters do not vary in a linear way. The reason for this is to approach the simulation from a realistic point of view. The parameters of the different levels are set out in Table III.

Table III: Levels Description

Level	Processed Bytes	Cycles	Bandwidth
1	300	150000	0
2	210	70000	90
3	100	50000	200
4	40	40000	260
5	0	0	300

The simulation method for iterative algorithms is not the same as for genetic algorithms. The reason for this is to test the ability of the genetic algorithms to adapt when the network nodes change. The simulation of the genetic algorithms is developed by searching for the most optimal solution through the GA at each duty cycle of the simulation, rather than cyclically switching levels of a single solution between nodes as in the IBAs simulation. This allows us to demonstrate the effectiveness of the GAs when competing with the best case scenario for IBAs. This is when the initial battery of the nodes is the same for each node in the network. In addition, we will be able to observe if the genetic algorithms are able to provide homogeneous consumption solutions on their own. It should also be noted that when the GAs are simulated with the same simulation method as the IBAs, identical average battery consumption are obtained and they differ from the simulations shown in this work only in the standard deviation of the batteries, which is zero for all the algorithms.

To this end, the following assumptions are made:

- We start with a 5-node network, which is extended to 10 and 20 nodes in the following simulations.
- All nodes start with the maximum battery, that of Table II.
- The variation of the bandwidth to be simulated is set to be between 50 and 1600 bytes.
- The simulation time is experimentally set to 5 seconds.
- The simulation results are displayed in 4 different plots:
  - Mean battery consumption
  - Best to worst percentage difference in battery consumption
  - Standard deviation between the batteries of the network after the simulation for each

bandwidth, to measure the homogeneity of the battery consumption solution.
– Execution time of the algorithms.

Calibration of the GA parameters is of paramount importance for good performance. Gibbs et al [19] propose a number of methods that attempt to provide insight into how to set these values. The method used in this paper is trial and error. The assumptions made for setting the GAs parameters are as follows:

- **Population size**. If the population size is too small, the GA may have a quick converging time; if it is too large, the GA may take longer time and computational resources to achieve a solution. The compromise between the two for the problem at hand, has been empirically found to be a population size of **200** chromosomes.
- **Mutation rate**. The mutation rate on GAs is usually set to low values, such as 0.01. However, it has been shown experimentally that setting the mutation rate to **0.1** significantly improves both the average consumption and the standard deviation.
- **Number of generations**.The criticality of the number of generations decreases when a termination condition is included. The termination condition is set for all GAs so that the algorithms stop generating new generations when more than **80%** of the chromosomes have the same fitness.

A. Mean Consumption Results

The average consumption results of the five algorithms after simulating different bandwidths on networks with 5, 10 and 20 nodes are shown in Figure 2. Note that it is difficult to distinguish between GA1 and GA3 in the figure, as they have almost the same average consumption. This is due to the fact that the fitness function of both algorithms is the same.

In most cases, it can be seen that IBA2 gives a higher battery consumption than the other algorithms. This is a result of the bandwidth allocation method used by IBA2. In the solutions proposed by this algorithm, there is a higher probability of having nodes at level 5, which is the one with the highest consumption. For example, for a 5-node network with a bandwidth of 600 bytes, IBA2 proposes the solution [0, 0, 0, 300, 300], where in each simulation step two nodes are at level 1 and three are at level 5, giving an average consumption of 46.92 mAh. On the other hand, IBA1 gives the solution [90, 90, 90, 90, 90, 200] with an average consumption in the 5 seconds of simulation of 34.792 mAh. It should also be noted that although IBA1 does not occupy the entire bandwidth, it achieves an average consumption 25.8% lower than IBA2. A fair distribution of the bandwidth is therefore more important than prioritising full bandwidth usage.

The algorithms with the lowest consumption are IBA1, GA1 and GA3. To facilitate discussion of the results, from now on when we refer to the GA3 in terms of average battery consumption, we are referring to both the GA3 and the GA1.

The graphs in Figure 2 show that for bandwidths of less than 500 bytes, IBA1 and GA3 give practically identical average consumption in the three cases considered. GA3 gives the lowest consumption for all bandwidths on 5-node networks. However, IBA1 improves consumption as the number of nodes increase. To better see the difference between the average consumption, we will focus on the graphs in Figure 3. This figure shows the division between the average consumption of IBA1 and GA3 for each of the simulated bandwidths. If this value is greater than 1, it means that IBA1 is consuming more than GA3, and the higher it is, the better GA3 is performing compared to IBA1.

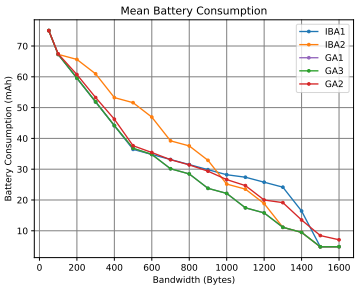
Figure 3a shows that GA3 has a lower average consumption than IBA1 for the vast majority of bandwidths. This difference increases with the bandwidth until they are equal again when the bandwidth allows all nodes to be at the highest level of computation offloading.

In the graph shown in 3c, IBA1 improves mean consumption over GA3. This is due to the fact that the bandwidth to be distributed is significantly low for the number of nodes in the network. For example, with a bandwidth of 1600 bytes, only 5 nodes could be at the highest level of computational offloading, which represents 25% of the network. Furthermore, the difference between the average consumption of IBA1 and GA3 does not exceed 12%. We can see that the improvement in consumption of GA3 over IBA1 reaches a difference of almost 40% if we change the bandwidth variation according to the number of nodes in the network.

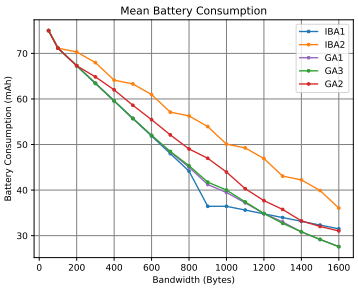
This can be seen in Figures 4 and 5, where the average consumption and the comparison between GA3 and IBA1 have been plotted for a 20-node network with a bandwidth variation between 2000 and 6100 bytes. As Figure 5 shows GA3, from a bandwidth of 2700 bytes, GA3 reduces the average network consumption, reaching a value 1.85 times lower than IBA1 (45% difference), while the lowest value in the graph is 0.86, indicating that GA3's consumption is only 1.16 times higher than IBA1 or, in other words, a difference of 14%. This leads to the conclusion that genetic algorithms far outperform iterative algorithms when bandwidth is adjusted according to the number of nodes in the network.

The same can be concluded for a 10-node network. As shown in Figure 3b IBA1 outperforms GA3 in a certain bandwidth range, but from 900 bytes the average consumption of GA3 starts a downward trend and overtakes IBA1.

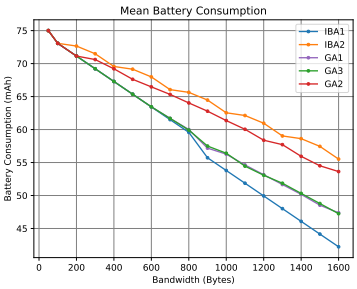
With respect to GA2, the graphs show that it presents intermediate results in terms of consumption. As it will be seen in the next section, its most interesting contribution is in its results in terms of standard deviation.



(a) 5 Nodes

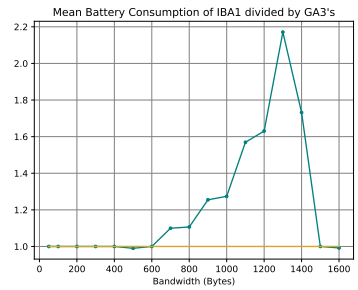


(b) 10 Nodes

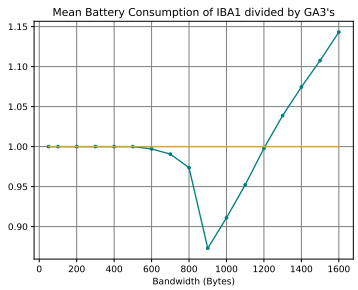


(c) 20 Nodes

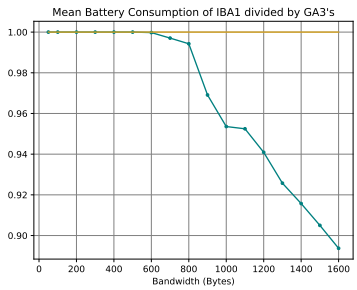
Fig. 2: Average consumption.



(a) 5 Nodes



(b) 10 Nodes



(c) 20 Nodes

Fig. 3: Mean Battery Consumption of IBA1 divided by GA3's.

B. Standard Deviation Results

The standard deviation is set as an additional objective for the algorithms as a measure of the homogeneity of the battery consumption throughout the network. Its simulation for the three cases under consideration is shown in the Figure 6.

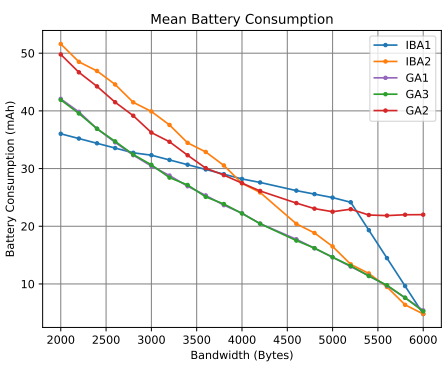


Fig. 4: 20-node network with higher bandwidths.

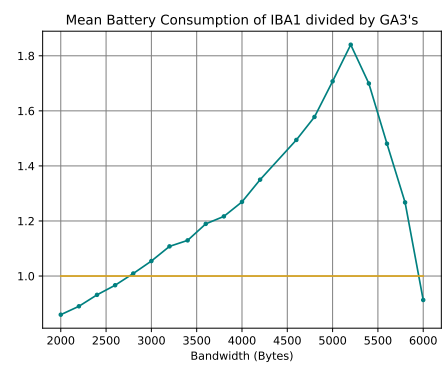


Fig. 5: Mean Battery Consumption of IBA1 divided by GA3's 20-node network with higher bandwidths.

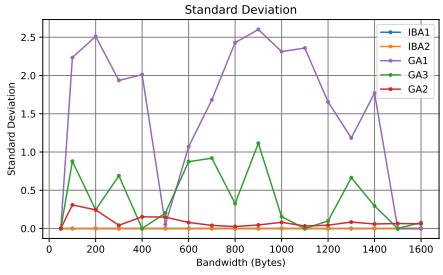
IBA1 and IBA2 have zero standard deviation because the implementation of the two iterative algorithms simulation forces the consumption to be homogeneous by cyclically switching the solution between the nodes.

It is not so trivial to achieve homogeneous consumption in genetic algorithms if the simulation method does not impose it. A first approach was to create GA2 specifically for this purpose by making its fitness function the minimum battery of the network. Although, as it can be seen in Figure 2, the average consumption of GA2 is negatively affected, the results in terms of standard deviation show a tendency to converge with that of the IBAs, taking values very close to zero, as the bandwidth increases.

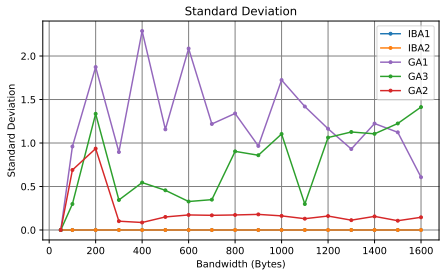
As far as GA1 and GA3 are concerned, changing the elite selection method from GA1 to GA3 improves the homogeneity of the consumption by placing it in an intermediate band between the standard deviation of GA1 and GA3.

However, as it can be seen in Figure 6, the standard deviations of the genetic algorithms do not exceed three points. This is a relatively low value and suggests a low variability in battery consumption.

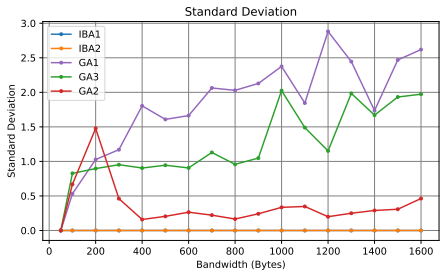
Looking at the results obtained in terms of Average Battery Consumption and Standard Deviation, it can be seen that the selection of the ideal algorithm varies greatly depending on the bandwidth available and the importance given to homogeneous battery consumption. These simulations could therefore be used to create a new algorithm that selects the most appropriate algorithm for the network in which computation offloading is to be applied, according to the characteristics and needs of the network.



(a) 5 Nodes



(b) 10 Nodes



(c) 20 Nodes
Fig. 6: Standard Deviation.

C. Execution time of the algorithms

Since the simulation process is different for the two types of algorithms in this work, their execution time will be studied separately.

When comparing the execution times of the GAs, the best approach is to present the average solution time for each simulation iteration. The values obtained by simulating the execution time show that the time taken by genetic algorithms to find the optimal solution, as well as the battery consumption, varies greatly with the bandwidth and the number of nodes in the network. Nevertheless, it is possible to draw some general conclusions. Firstly, GA2 is the fastest in all cases. Its execution time varies from a minimum of 5.2 ms to 47 ms for a 5-node network, compared to the slowest, GA1, whose maximum and minimum times are 9.8 ms and 374 ms. The execution time of GA3 in the vast majority of cases remains between the values of GA1 and GA2. For networks of 10 and 20 nodes, similar variations are obtained, with longer times, leading to a second conclusion. This is that as the number of nodes increases and the bandwidth to be distributed is maintained, the genetic algorithms take longer to find the solution.

The value of these times must be taken into account when selecting one of these algorithms for application in a real network. This will depend on the importance of speed in finding solutions.

In the context of IBAS, the execution time to find the best solution does not exceed tens of milliseconds. However, because the instructions used by iterative processes are less complex than those used by metaheuristics, these data are not comparable with those obtained from genetic processes. Furthermore, iterative algorithms only search for the offloading decision at the beginning of the simulation and the only change between successive simulation cycles is the rotation of the values of the solution chain, which considerably reduces the simulation time.

V. Conclusions and further research

A. Conclusions

Looking at the results in Figure 2, it can be concluded that, with the given simulation parameters, the best iterative algorithm in terms of mean consumption is IBA1, and that in most cases it is matched or improved by GA1 and GA3. It has also been proved (Fig. 4) that as bandwidth increases in critical situations, genetic algorithms respond better than iterative algorithms in terms of average consumption.

In terms of achieving balanced consumption between nodes for genetic algorithms, it has been shown that the standard deviation is greatly improved by choosing the selection logic of the best individual according to it. However, if it is considered critical, it is necessary to include the standard deviation in the fitness function to obtain values close to zero.

As mentioned above, genetic algorithms were designed with dynamic networks in mind, i.e. networks where nodes do not necessarily start with the same battery and where the number of nodes can vary. Under these conditions, it is expected that the IBA algorithms will perform worse in terms of network lifetime, as they do not take into account the current state of the nodes' batteries. Demonstrating this through simulations is planned as a continuation of the research.

In conclusion, genetic algorithms have been proven to be a powerful and useful tool for finding the best offloading strategy at any point in the network's lifetime. Among the three different GAs proposed, GA3 solves the NP-hard optimisation problem posed in the most efficient way, minimising battery consumption and also fulfilling the additional objective of ensuring that the grid batteries are balanced.

B. Further research

Two possible lines of future research are considered as a result of this work: the search for new genetic algorithms that can include more than one target and the use of artificial intelligence.

After investigating possible variants of genetic algorithms to better fit the desired results, we thought that multi-objective genetic algorithms could be a very promising line of research. Furthermore, we

found that [20] introduces a novel algorithm that we believe could be adapted to our problem. It consists of a multi-objective co-evolutionary co-generative genetic algorithm (MOCCGA), which combines two existing genetic algorithms, the multi-objective genetic algorithm (MOGA) and the cooperative co-evolutionary genetic algorithm (CCGA). The aim is to improve the performance of MOGA by incorporating the cooperative co-evolutionary effect into its search mechanisms.

As [21] surveys, Artificial Intelligence (AI) and Machine Learning (ML) methods could also be powerful tools for decision making in computational offloading environments, as they introduce the capability of reinforcement learning.

Acknowledgement

This work has been funded by project TALENT-HIPSTER, reference PID2020-116417RB-C41 funded by MCIN/AEI/10.13039/501100011033/ and by FEDER "Una manera de hacer Europa".

References

[1] Farzad Samie, Lars Bauer, and Jörg Henkel, "Iot technologies for embedded computing: A survey," in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2016, pp. 1–10.

[2] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava, "A survey of computation offloading for mobile systems," *Mobile networks and Applications*, vol. 18, pp. 129–140, 2013.

[3] H Sabireen and VJIE Neelanarayanan, "A review on fog computing: architecture, fog with iot, algorithms and research challenges," *Ict Express*, vol. 7, no. 2, pp. 162–176, 2021.

[4] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[5] Pablo Merino, Gabriel Mujica, Jaime Señor, and Jorge Portilla, "A modular iot hardware platform for distributed and secured extreme edge computing," *Electronics*, vol. 9, no. 3, pp. 538, 2020.

[6] Jorge Portilla, Gabriel Mujica, Jin-Shyan Lee, and Teresa Riesgo, "The extreme edge at the bottom of the internet of things: A review," *IEEE Sensors Journal*, vol. 19, no. 9, pp. 3179–3190, 2019.

[7] Farzad Samie, Vasileios Tsoutsouras, Lars Bauer, Sotirios Xydis, Dimitrios Soudris, and Jörg Henkel, "Computation offloading and resource allocation for low-power iot edge devices," in *2016 IEEE 3rd world forum on internet of things (WF-IoT)*. IEEE, 2016, pp. 7–12.

[8] Zhi Li and Qi Zhu, "Genetic algorithm-based optimization of offloading and resource allocation in mobile-edge computing," *Information*, vol. 11, no. 2, pp. 83, 2020.

[9] Shuang Fu, Chenyang Ding, and Peng Jiang, "Computational offloading of service workflow in mobile edge computing," *Information*, vol. 13, no. 7, pp. 348, 2022.

[10] Helena Ramalhinho Lourenco and Daniel Serra, "Adaptive search heuristics for the generalized assignment problem," *Mathware & soft computing. 2002 Vol. 9 Núm. 2 [-3]*, 2002.

[11] Bo Wang, Changhai Wang, Wanwei Huang, Ying Song, and Xiaoyun Qin, "A survey and taxonomy on task offloading for edge-cloud computing," *IEEE Access*, vol. 8, pp. 186080–186101, 2020.

[12] Akhirul Islam, Arindam Debnath, Manojit Ghose, and Suchetana Chakraborty, "A survey on task offloading in multi-access edge computing," *Journal of Systems Architecture*, vol. 118, pp. 102225, 2021.

[13] Mohamed Abdel-Basset, Laila Abdel-Fatah, and Arun Kumar Sangaiah, "Metaheuristic algorithms: A comprehensive review," *Computational intelligence for multimedia big data on the cloud with engineering applications*, pp. 185–231, 2018.

[14] David E Goldberg, *Genetic algorithms*, pearson education India, 2013.

[15] Holland Jh, "Adaptation in natural and artificial systems," *Ann Arbor*, 1975.

[16] Ahmed A Al-Habob, Octavia A Dobre, and Ana Garcia Armada, "Sequential task scheduling for mobile edge computing using genetic algorithm," in *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2019, pp. 1–6.

[17] Khalid Jebari and Mohammed Madiafi, "Selection methods for genetic algorithms," *International Journal of Emerging Sciences*, vol. 3, no. 4, pp. 333–344, 2013.

[18] Chang Wook Ahn and Rudrapatna S Ramakrishna, "Elitism-based compact genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 4, pp. 367–385, 2003.

[19] Matthew S Gibbs, Holger R Maier, Graeme C Dandy, and John B Nixon, "Minimum number of generations required for convergence of genetic algorithms," in *2006 IEEE International Conference on Evolutionary Computation*. IEEE, 2006, pp. 565–572.

[20] Nattavut Keerativuttitumrong, Nachol Chaiyaratana, and Vara Varavithya, "Multi-objective co-operative co-evolutionary genetic algorithm," in *Parallel Problem Solving from Nature—PPSN VII: 7th International Conference Granada, Spain, September 7–11, 2002 Proceedings 7*. Springer, 2002, pp. 288–297.

[21] Sundas Iftikhar, Sukhpal Singh Gill, Chenghao Song, Minxian Xu, Mohammad Sadegh Aslanpour, Adel N Toosi, Junhui Du, Huaming Wu, Shreya Ghosh, Deepraj Chowdhury, et al., "Ai-based fog and edge computing: A systematic review, taxonomy and future directions," *Internet of Things*, p. 100674, 2022.

Desarrollo de una plataforma remota para prácticas de sistemas electrónicos digitales

Rubén Gil, Sandra I. Serrano, José L. Lázaro, Alfredo Gardel, Ignacio Bravo [1]

Resumen— **El objetivo de este trabajo es presentar un sistema de acceso remoto a un equipamiento de laboratorio de sistemas electrónicos digitales disponible para realizar distintas prácticas. Se ha desarrollado una plataforma accesible de forma remota como parte de la oferta de equipamiento de los laboratorios del Departamento de Electrónica de la Universidad de Alcalá. De esta manera, el sistema ofrece a los alumnos la posibilidad de generar y controlar distintos tipos de señales a través de una interfaz web local con el objetivo de evaluar la ejecución de sus prácticas de laboratorio. En particular se ha realizado una plataforma para prácticas docentes de sistemas electrónicos digitales. Ahora bien el campo de aplicación es muy amplio y el desarrollo realizado puede ser ampliado a otro tipo de laboratorios remotos de forma que se contribuya a una mejora de la calidad de la docencia y de la experiencia educativa de los estudiantes. En este caso se necesita reconfigurar las tecnologías, recursos y herramientas que se han aplicado tradicionalmente en un entorno presencial, con el propósito de que los estudiantes puedan desarrollar las mismas competencias de manera remota, personalizando y adaptando su formación a las condiciones espacio-temporales disponibles.**

Palabras clave— **Prácticas remotas, aprendizaje a distancia, laboratorio remoto, sistemas electrónicos digitales.**

I. Introducción

La promoción de una educación universitaria de alta calidad siempre ha sido una prioridad para los responsables de las distintas instituciones y organismos internacionales -como lo demuestra la definición del cuarto objetivo de la Agenda 2030 para el Desarrollo Sostenible establecida por las Naciones Unidas [1].

Asimismo, la excepcional circunstancia sanitaria de pandemia ha catalizado distintos procesos de enseñanza-aprendizaje que se habían estado desarrollando a un ritmo más pausado, obligando a centrar los esfuerzos en el despliegue de métodos de enseñanza no presencial e híbrida [2].

En esta dirección, si bien el desarrollo de clases teóricas se ha integrado de forma muy adecuada con las tecnologías de enseñanza a distancia y recursos online, la implementación de la enseñanza práctica aunque se puede ver ayudada con simuladores [3], a veces, se ve obstaculizada por la necesidad de acceder a equipos físicos de los que los estudiantes no disponen en sus hogares [4].

En respuesta a este desafío, una solución viable para mejorar la adquisición de habilidades prácticas sin la necesidad de presencia física en los laboratorios es permitir el acceso remoto a los equipos físicos. De este modo, se podrían generar señales de entrada, recopilar señales de salida y visualizar los datos luminosos o mecanismos que se estén utilizando en las prácticas respectivas. Esto equivaldría a disponer de los mismos recursos instrumentales requeridos, pero incrementando la disponibilidad de la instrumentación y los equipos de laboratorio de manera remota.

La implementación de dicha infraestructura, además de facilitar el desarrollo de prácticas a distancia, serviría como un soporte complementario que permitiría a los estudiantes tener un mayor número de horas de laboratorio. Incluso permitiría a estudiantes internacionales hacer uso de dicho equipamiento para llevar a cabo prácticas remotas, en horarios que habitualmente los sistemas están sin funcionamiento [5].

Existen precedentes de instituciones universitarias que han trabajado en el desarrollo e implantación de laboratorios remotos aplicados al estudio de ingenierías tecnológicas. Algunos significativos se citan a continuación.

En el Instituto Tecnológico de Karlsruhe (Alemania), por ejemplo, han desarrollado el "Robot Learning Lab", un laboratorio de robótica accesible a distancia que permite a los estudiantes acceder a los robots del laboratorio de forma remota y ejecutar sus propios proyectos en ellos. El laboratorio consta de diez robots desplegados en serie, cada uno equipado con un brazo manipulador con capacidades de detección de fuerza/par y visión artificial, sensor 3D y una cámara web para la visualización remota del experimento. Los alumnos también obtienen registros y datos recogidos durante los experimentos; el objetivo es proporcionar la misma experiencia de desarrollo y acceso a los datos como si los usuarios estuvieran junto a los robots. El laboratorio también participa en la impartición de cursos masivos abiertos en línea (MOOC) con cientos de participantes. Para los estudiantes, poder manejar el laboratorio más horas es una forma única de aprender los fundamentos de la robótica y tener más experiencia con el hardware robótico [6].

La Escuela Politécnica Federal de Lausana pone a disposición de los estudiantes, a través de su "Remote Lab of the Automatic Control Laboratory", equipamiento para enseñanza remota de sistemas de control que consta de los siguientes elementos: veintidós estaciones para prácticas online, diez servo-drivers, diez sistemas de control de temperatura y un sistema para el control de un péndulo invertido [7].

La Universidad de Nueva Gales del Sur (Sidney) pone a disposición de los estudiantes de la Escuela de Ingeniería Eléctrica y de Telecomunicaciones la po-

[1]Dpto. de Electrónica, Universidad de Alcalá, e-mail: {ruben.gil, josel.lazaro, alfredo.gardel, sandraisabel.serrano, ignacio.bravo}@uah.es.

sibilidad de trabajar remotamente en el laboratorio desde sus casas. Lo hacen a través de una interfaz de software de acceso remoto, para generar y medir formas de onda de tensión y corriente variables en el tiempo, y trabajando con software y con hardware reconfigurables para construir circuitos. Además, se utiliza una cámara web y Microsoft Teams para mayor realimentación y sensación de interactuación con el sistema [8].

Además, el equipo proponente de este proyecto ha trabajado con anterioridad en el desarrollo, de sistemas de reserva y acceso a recursos compartidos y remotos, tanto en convocatorias de innovación como en proyectos de internacionalización de la Universidad.

En este trabajo se plantea el desarrollo de herramientas y materiales para la realización de un puesto de práctica completo de Sistemas Electrónicos Digitales que permita la impartición de docencia en remoto, así como para fomentar prácticas colaborativas entre docentes y estudiantes de diferentes centros o entre alumnos en sus domicilios y en el centro.

El artículo se ha organizado de la siguiente manera. En la sección II se realiza una descripción de la plataforma de prácticas con los distintos módulos que permiten su uso a distancia. Posteriormente, la sección III se muestra el funcionamiento del sistema para el desarrollo de una práctica real de sistemas electrónicos digitales. Finalmente, la sección IV resume las principales conclusiones y contribuciones del desarrollo.

II. Descripción de la plataforma de prácticas

El diagrama de bloques de la plataforma de prácticas de acceso remoto es el mostrado en la figura 1.

Como punto de partida se debe tomar el ordenador del usuario, que se conectará al equipo de laboratorio (plataforma de prácticas) a través de una aplicación de escritorio remoto. La plataforma de prácticas está compuesta por el ordenador de laboratorio, una tarjeta encargada de la generación de señales y una tarjeta de desarrollo miniDK2, aunque puede utilizarse cualquier otra, o incluso tarjetas de desarrollo con CPLD (Complex Programmable Logic Device),FPGA (Field Programmable Gate Array), etc. En esta tarjeta el usuario realizará el volcado de sus programas haciendo uso de diferentes elementos adicionales y configurados al efecto como elementos de las prácticas.

Los mecanismos de seguridad que permite el acceso sólo a estudiantes matriculados en la asignatura está fuera del ámbito de este trabajo. Tampoco se comenta el sistema de reservas ni el procedimiento utilizado para distribuir de forma adecuada el recurso entre los distintos usuarios.

A continuación, en la subsección siguiente se describe la tarjeta de desarrollo utilizada. Es importante resaltar que cada plataforma de acceso remoto dispone de 2 de estas tarjetas. Una es programada por el estudiante para la realización de la práctica.

A. Tarjeta de desarrollo utilizada

Para llevar a cabo la práctica de Sistemas Electrónicos Digitales se ha empleado la tarjeta de desarrollo mini-DK2 basada en el microcontrolador LPC1768, con procesador ARM Cortex-M3. Este es un procesador de 32 bits diseñado idóneo para aplicaciones de bajo costo, en donde la eficiencia de procesado es importante, con baja latencia de interrupción y facilidad de uso. Las principales razones por los que se ha decidido utilizar esta tarjeta son su facilidad de programación y depuración a través del entorno Keil con librerías de acceso a nivel de registro interno, utilizando su versión educacional gratuita y cesión de licencia profesional para los laboratorios. Dispone de múltiples periféricos, de los que cabe destacar los siguientes, dado que serán utilizados en la práctica:

1. Puerto Ethernet
2. Puerto USB para descarga
3. ADC de 12 bits, con 8 canales de entrada.
4. DAC de 10 bits.
5. 4 temporizadores de propósito general.
6. 6 salidas de PWM.
7. 70 GPIOs - pines de entrada/salida de propósito general.
8. Más puertos como I2C o UART, entre otros.

B. Interfaz web de la tarjeta de control

En esta sección se muestra la interfaz web de la tarjeta de control que ha sido previamente programada para que el usuario pueda generar señales de entrada y controlar los equipos de medida de la práctica de sistemas electrónicos digitales. A partir del hardware integrado en la plataforma y mediante el uso de esta interfaz se permite generar y controlar señales analógicas y digitales que se conectan a la tarjeta de desarrollo pudiendo visualizar los resultados a partir de la realimentación visual con cámaras.

Una vez el estudiante accede al ordenador del laboratorio puede ejecutar la aplicación web fácilmente, a partir de un icono ubicado en la pantalla del escritorio del ordenador remoto. La capa de aplicación web se basa en el protocolo CGI (Common Gateway Interface), que facilita el intercambio de datos estandarizado entre servidores y aplicaciones externas, permitiendo que el contenido de una página web se genere dinámicamente en respuesta a las solicitudes del usuario. En la figura 2 se muestra cómo la información ingresada por el cliente del servidor TCP/IP se transmite a la aplicación de software como una variable de entorno a través de CGI, que también admite el envío de información en la dirección opuesta desde la aplicación al cliente. De esta sencilla manera se ha implementado el manejo web de la tarjeta software-hardware de control.

A partir de archivos CGI se han ido desarrollando los distintos elementos de control necesarios para la realización de prácticas en el ámbito de los sistemas electrónicos digitales.

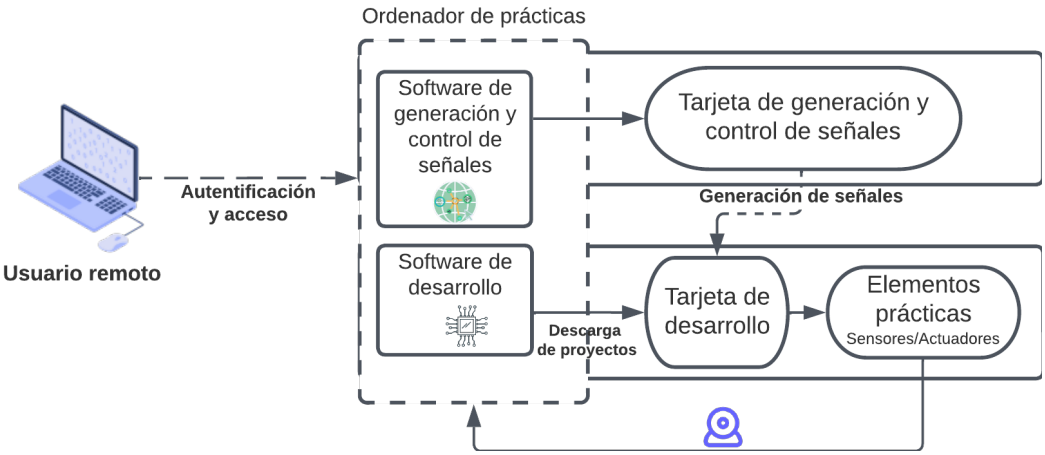


Fig. 1: Diagrama de bloques general de la plataforma de desarrollo

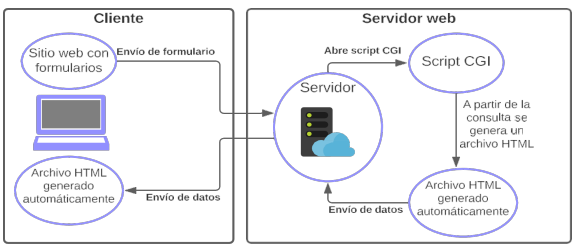


Fig. 2: Protocolo web CGI implementado

B.1 Control de señales digitales

En la figura 3 se muestra la interfaz web a través del cual el estudiante puede generar diversas señales digitales, secuencias, el paso a nivel alto/bajo de una señal, etc. El usuario de la plataforma remota puede controlar directamente su valor digital a partir de los botones de los que dispone la interfaz. Además, se pueden generar secuencias de 16 bits pudiendo controlar el tiempo de emisión de cada símbolo y seleccionar emisión periódica o una sola vez. Al igual que en la señales de cambio de nivel, las secuencias también cuentan con botones para su control, pudiendo comenzar su emisión. pausarlas o pararlas.

B.2 Generación de formas de onda

Dentro del mismo entorno el usuario puede configurar señales temporales analógicas, PWM y seleccionar el canal a capturar desde el osciloscopio. La interfaz para la generación y control de todas estas señales se muestra en la figura 4. La interfaz permite por un lado configurar los valores de generación de la señal analógica (frecuencia y amplitud) y por otro arrancar/parar la generación de la señal que puede ser de tipo sinusoidal, diente de sierra o triangular. Los pasos de configuración son los siguientes:

1. Seleccionar la frecuencia de la señal analógica a generar. El rango disponible va de 1 Hz a 1 MHz.
2. Configurar la amplitud de la señal, entre 0 y 3300 mV.
3. Generar o parar la salida de señal, seleccionando una forma de onda.

B.3 Generación de señales PWM

Por otro lado, el sistema de generación permite establecer hasta cuatro señales PWM compartiendo todas ellas la misma referencia de frecuencia.

Se dispone de hasta dos señales PWM de tipo *single edge* y dos de tipo *double edge*, pudiendo configurar en estas últimas el momento en el que la señal cambia de valor en el flanco de subida y en el de bajada. Se debe configurar cada instante de cambio de la señal como porcentaje del ciclo de trabajo de la señal PWM. La interfaz cuenta con dos botones de *Start* y *Stop* que permite dar comienzo o detener la emisión de las señales PWM.

B.4 Selección de los canales de captura del osciloscopio

La plataforma de prácticas cuenta con un osciloscopio y un multiplexor analógico de 8 canales que permite la selección del canal a mostrar en el osciloscopio que desee el usuario. En la figura 5 se tienen listados los pines de la tarjeta conectados a cada uno de los canales del multiplexor.

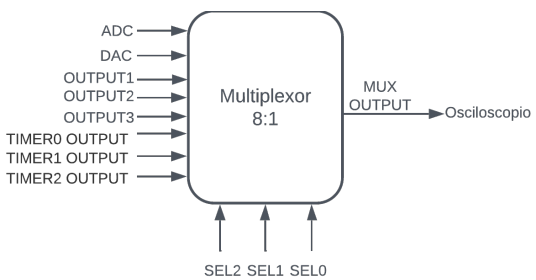


Fig. 5: Conexiones de los canales del multiplexor

Una vez seleccionado el pin de la tarjeta a monitorizar, el entorno permite controlar remotamente el osciloscopio con 3 botones. El botón principal, "Sel", permite navegar entre los elementos mostrados en la pantalla del osciloscopio. Cuando se tiene seleccionado la escala de tiempos del osciloscopio, con los botones "+" y "-", se permite ampliar o disminuir la división temporal de la cuadrícula horizontal del osciloscopio.

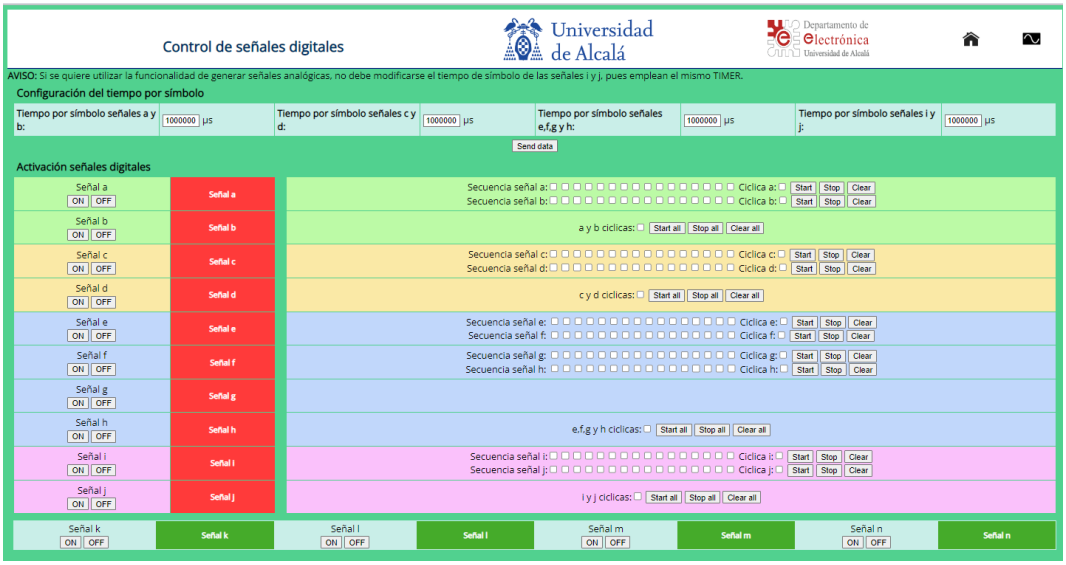


Fig. 3: Interfaz de control de señales digitales

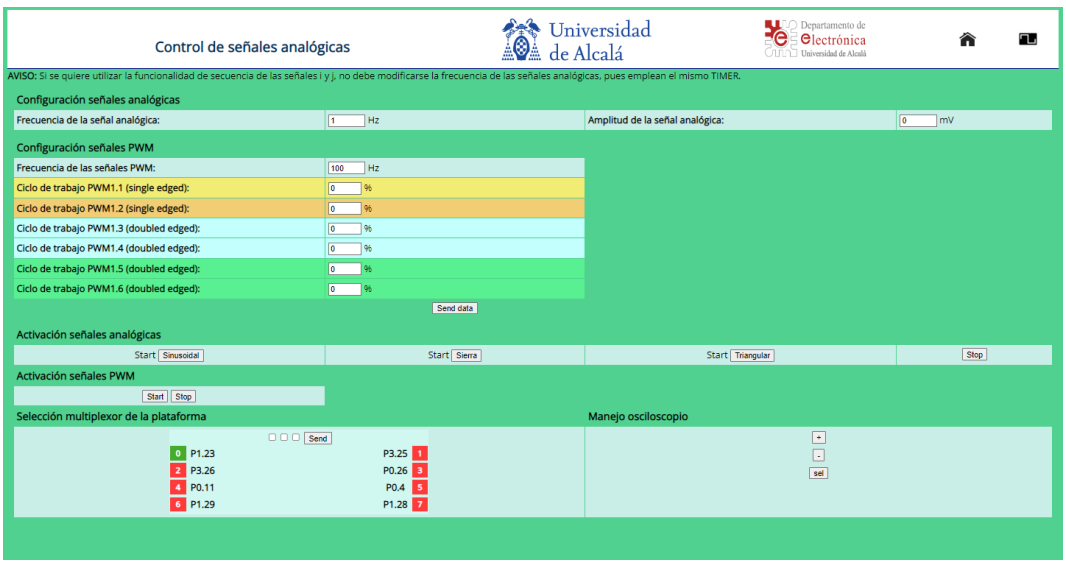


Fig. 4: Interfaz de control y generación de señales PWM y analógicas

Para comprobar su funcionamiento, se ha generado con la interfaz web de señales PWM, una señal cuadrada de 1 Hz por el pin P1.29 con un ciclo de trabajo configurable entre 20, 40, 60 y 80 % según el valor de los pines P0.0 y P0.1, conectados a las señales a y b. En la figura 6 se muestra un ejemplo real de la señal generada con un ciclo de trabajo del 50 %. La escala seleccionada con la botonera web del osciloscopio es de 0,2 s/div en el eje horizontal y de 1V/div en el eje vertical.

III. Resultados experimentales

Con el fin de realizar las pruebas bajo un escenario realista, se ha probado la plataforma utilizado las prácticas de laboratorio que se están desarrollando de manera habitual en las asignaturas de Sistemas Electrónicos Digitales impartidas en la Escuela Politécnica Superior de la Universidad de Alcalá [9].

En este apartado se muestra el desarrollo de una práctica compleja donde se verifica y validan muchas de las opciones que brinda la plataforma remota.

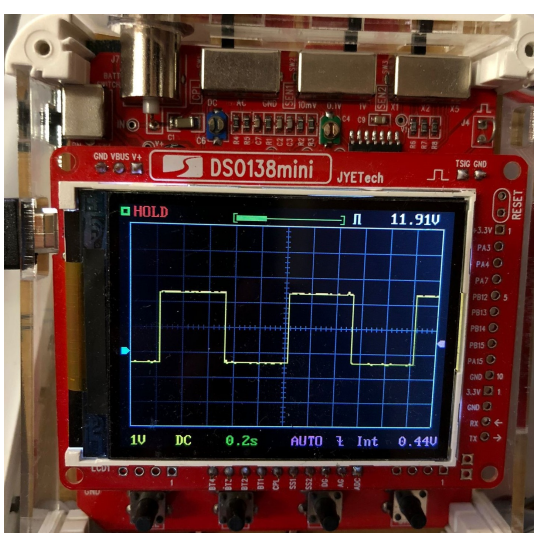


Fig. 6: Señal cuadrada de 0.5 Hz con ciclo de trabajo del 50 %

En la figura 7 se muestra la plataforma con los distintos componentes básicos (ordenador y tarjeta

generadora de señales, tarjeta de desarrollo y osciloscopio), además de un conjunto de elementos ya conectados (protoboard con leds, display, servo y sonar de ultrasonidos). Como se aprecia, el sistema es flexible y permite incorporar diferentes elementos. En este caso se ha incluido un modor DC y puente en H, y aunque no ha sido utilizado en la práctica de sistemas electrónicos digitales permite utilizar la plataforma para el aprendizaje de sistemas de control reales.

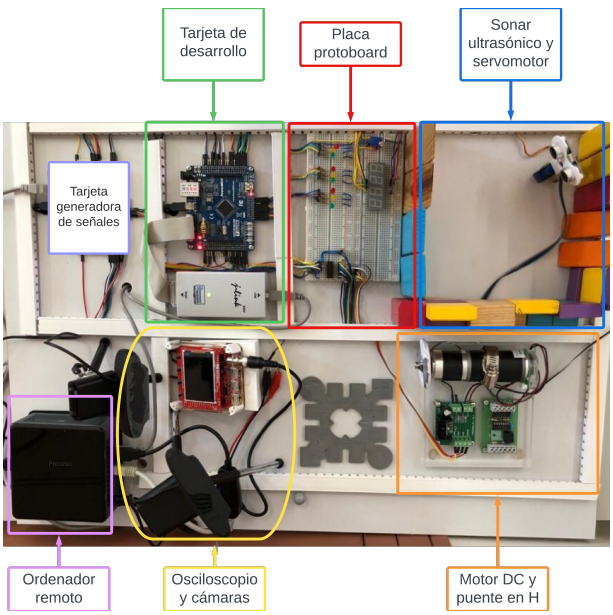


Fig. 7: Plataforma de prácticas remotas

Se disponen de múltiples prácticas de laboratorio que abarcan el uso de la gran mayoría de periféricos de la tarjeta de desarrollo, actuadores y sensores de la plataforma. Concretamente, para este artículo se ha seleccionado un proyecto donde el objetivo es realizar un sistema electrónico digital que implemente un sonar ultrasónico capaz de extraer las medidas de distancia a los objetos más próximos en un barrido de 180º. El movimiento del sensor de ultrasonidos es controlado mediante un servomotor. El sistema ante la presencia de un obstáculo genera de salida una señal de frecuencia proporcional a la distancia de detección, de manera que se tiene una señal acústica de detección de obstáculos.

Las especificaciones de funcionamiento del sistema son las siguientes:

1. Se controlará la posición del servo mediante 2 entradas, de nombre “l” y “m” (pines P2.11 y P2.12), de manera que a través de su activación (pulsación) haga girar la posición del sensor 10º hacia la izquierda o hacia la derecha, respectivamente.
2. Si la medida de la distancia a los objetos cercanos está entre 3 y 15 cm se debe generar una señal sinusoidal proporcional con una frecuencia de valor en kHz igual a los centímetros medidos. Por ejemplo, si se mide un valor de 4.5 cm, se debe generar una señal de 4.5 kHz.
3. Si la medida de la distancia es inferior a 3 cm o superior a 15 cm, se debe generar una señal sinusoidal de baja frecuencia, 500 Hz, indicando que no hay medida (bien porque está muy cercano o porque el objeto de reflexión está muy alejado).

El entorno de piezas configurado, es conocido por el usuario y permite realizar distintas medidas de prueba. Para conocer si los valores de medida son correctos o existe algún error en la configuración, control o medida desde el sistema digital.

Cabe mencionar, que el usuario dispone en todo momento de la herramienta de depuración Keil μVision para conocer los valores que proporciona el sensor de ultrasonidos, pudiendo comprobar la medida de distancia.

En la figura 8 se posiciona el servomotor del sensor frente a un objeto del entorno que se encuentra a 6.6 cm del sensor.

Fig. 8: Sensor ultrasonidos direccionado a 6.6 cm

Si el estudiante realiza correctamente la práctica, en la figura 9, podrá comprobar que la medida obtenida por el sonar de 6.6 cm se visualiza en forma de señal sinusoidal con una frecuencia proporcional en kHz, esto es 6.6 kHz. La escala de los ejes del osciloscopio se adapta desde la interfaz web a 50 μs/div en el eje abscisas es y 1 voltio/div en el eje de ordenadas.

Finalmente, en la figura 10 se muestra un diagrama esquemático de las conexiones de la práctica de ultrasonidos desarrollada. Cabe mencionar que el conexionado de la maqueta es fijo, permitiendo muchas opciones y gran flexibilidad para generar diferentes tipos de prácticas de sistemas digitales.

Ahora bien, dado que el cableado está predeterminado, la confección de las prácticas se deben orientar de una manera particular para que sea factible realizarlas con ese conexionado. En el caso particular de la práctica de sensor de ultrasonidos, el sistema digital realiza la generación de la señal de envío del ultrasonido con un temporizador periódico (MAT3.1). Cada instante de emisión se captura a través de un pin configurado en modo captura/contador (CAP2.0). El bloque hardware se configura para que esté a la espera del eco de la señal (CAP2.1), que se captura

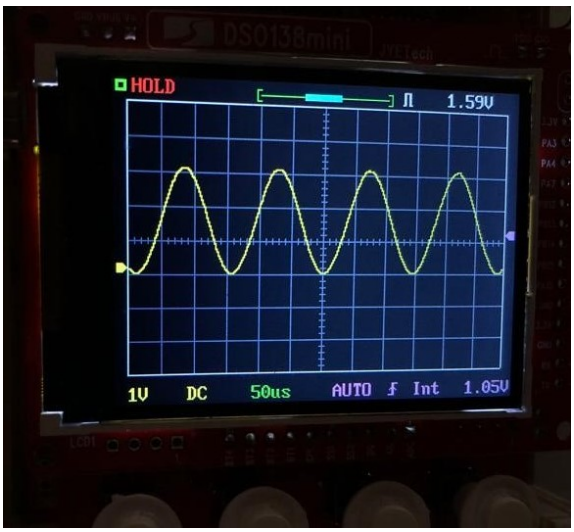


Fig. 9: Señal de 6.6 kHz proporcional a la medida de distancia para un obstáculo a 6.6cm

en otro contador y genere la interrupción al sistema y se obtenga la medición de distancia a partir de la diferencia de los valores de ambos contadores.

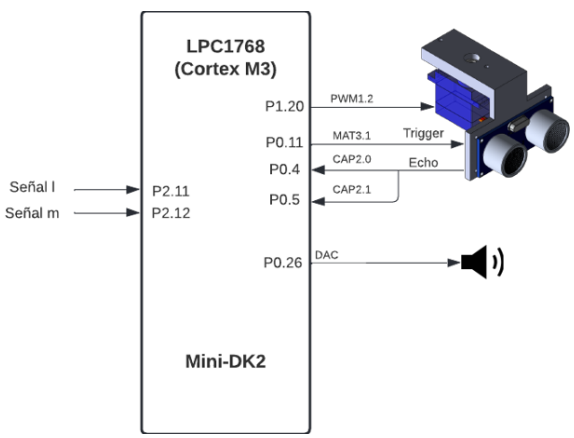


Fig. 10: Diagrama de la práctica

En la figura 11 la maqueta en funcionamiento con la captura de la pantalla del ordenador donde se puede visualizar la configuración de las señales de entrada, canal del osciloscopio, etc. También se muestra la captura de la pantalla del osciloscopio con la salida de la medida en la señal de frecuencia y la posición actual del sensor en el entorno real.

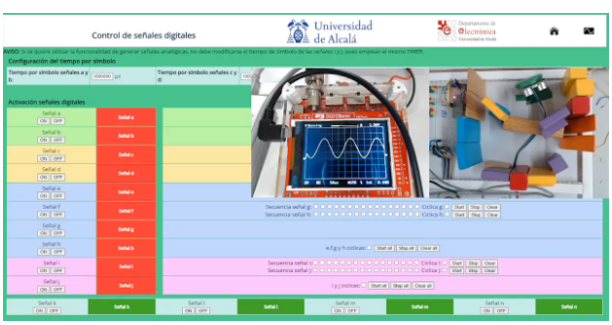


Fig. 11: Maqueta en funcionamiento con los elementos necesarios para desarrollar la práctica de ultrasonidos

IV. Conclusiones

A continuación, se proporciona un resumen de los logros clave derivados del trabajo realizado:

1. Acceder y controlar remotamente a un equipo de laboratorio, de forma que se pueda utilizar todas las herramientas instaladas en el ordenador que controla la plataforma de desarrollo del software del usuario.
2. Sistema de generación y control de señales de entrada digitales, valores simples o secuencias complejas, señales periódicas o de una única emisión, emisión síncrona o asíncrona de señales.
3. Sistema de generación y control de señales de entrada analógicas con forma de onda cuadrada, triangular o sierra y señales PWM de frecuencia y ciclo de trabajo seleccionable.
4. Sistema de realimentación visual mediante cámaras que permite obtener en tiempo real los resultados de las pruebas realizadas desde distintas cámaras instaladas en la plataforma.
5. Sistema de visualización de señales y sensado a través de un osciloscopio de bajo coste junto con un multiplexor analógico que permite visualizar tanto las señales de entrada desde la tarjeta generadora como las salidas del software desarrollado por el usuario.
6. Desarrollo de una plataforma de prácticas adaptable y escalable a gran variedad de prácticas con un mismo hardware o pudiendo ampliar el hardware instalado sin necesidad de modificar el software de generación de señales.
7. Gran usabilidad para usuarios que están comenzando en el estudio de sistemas electrónicos digitales.
8. Desarrollo a distancia de una práctica real de programación de sistemas electrónicos digitales permitiendo así a los usuarios disponer de un mayor número de horas de laboratorio.

El sistema propuesto optimiza la utilización de hardware específico en la realización de distintas prácticas académicas por parte delos estudiantes. Sin embargo, conviene observar que la plataforma, al tener una configuración predeterminada, implica ciertas limitaciones que pueden afectar a la flexibilidad que de partida tiene el diseño de un sistema digital. Específicamente, la solución propuesta conlleva restricciones en el diseño de las interconexiones entre el microcontrolador y los sensores o actuadores ya cableados. Este condicionamiento reduce en cierta medida las opciones del estudiante para explorar y experimentar con diversas arquitecturas y topologías de sistemas digitales. Por ello se aconseja su utilización hasta niveles intermedios de aprendizaje de sistemas digitales.

Agradecimientos

El presente trabajo ha sido cofinanciado a través del proyecto Erasmus+ DECEL - Digital Electronics Collaborative Enhanced Learning, Ref. 2021-1-ES01-KA220-HED-000032189.

Referencias

[1] Daniel De la Rosa Ruiz, Pilar Giménez Armentia, Carmen De la Calle Maldonado, et al., "Educación para el desarrollo sostenible: el papel de la universidad en la agenda 2030.," *revista prisma social*, 2019.

[2] Kelum AA Gamage, Kerlin Jeyachandran, Shyama CP Dehideniya, Chris G Lambert, and Allan EW Rennie, "Online and hybrid teaching effects on graduate attributes: Opportunity or cause for concern?," *Education Sciences*, vol. 13, no. 2, pp. 221, 2023.

[3] Michael E Auer and Danilo G Zutin, *Online Engineering & Internet of Things: Proceedings of the 14th International Conference on Remote Engineering and Virtual Instrumentation REV 2017, Held 15-17 March 2017, Columbia University, New York, USA*, vol. 22, Springer, 2017.

[4] Ingvar Gustavsson, "Remote laboratory experiments in electrical engineering education," in *Proceedings of the fourth IEEE international caracas conference on devices, circuits and systems (Cat. No. 02TH8611)*. IEEE, 2002, pp. I025–I025.

[5] Martyn Cooper and Jose MM Ferreira, "Remote laboratories extending access to science and engineering curricular," *IEEE Transactions on learning technologies*, vol. 2, no. 4, pp. 342–353, 2009.

[6] Karlsruhe Institute of Technology, "Kuka robot learning lab at kit," .

[7] Escuela Politécnica Federal de Lausana, "Remote lab of the automatic control labora- tory," .

[8] Universidad de Nueva Gales del Sur, "A new paradigm for the delivery of remote laboratory education," .

[9] A. Gardel Vicente, C. Mataix Gómez, C. Luna Vázquez, J. L. Lázaro Galilea, J. J. García Domínguez, and J. M. Miguel Jiménez, *Prácticas de laboratorio de Sistemas Electrónicos Digitales*, Universidad de Alcalá, 2018.

[10] James M. Kieley, "Cgi scripts: Gateways to world-wide web power," *Springer*, vol. 28, 1996.

[11] Ryan Levering and Michal Cutler, "The portrait of a common html web page," in *Proceedings of the 2006 ACM Symposium on Document Engineering*, New York, NY, USA, 2006, DocEng '06, p. 198–204, Association for Computing Machinery.

[12] Ricardo A Diogo, Neri dos Santos, and Eduardo FR Loures, "Digital transformation of engineering education for smart education: A systematic literature review," *Reliability Modeling in Industry 4.0*, pp. 407–438, 2023.

Integración de software libre en todas las etapas de diseño de sistemas electrónicos: aplicación en las asignaturas de *Diseño de Circuitos y Sistemas Integrados* y *Sistemas Empotrados*

Antonio Martínez-Álvarez[1], Sergio Cuenca-Asensi[1], Jose Luis Sánchez-Romero[1], Luis Lucas-Ibáñez[1], Daniel Gutiérrez-Castro[1], Jose Manuel Palomares-Muñoz[2] y Alejandro Serrano-Cases[3]

Resumen—**El trabajo presentado en este artículo, de corte docente, quiere compartir una experiencia de integración de distintas soluciones de software libre en asignaturas relacionadas con el diseño de sistemas electrónicos en cualquier nivel de abstracción. El empleo de software libre se justifica y defiende presentando los casos de uso y problemáticas, en el ámbito de asignaturas técnicas relacionadas, para las que lo estimamos plenamente recomendable. Se han tomado dos asignaturas relevantes del Máster en Telecomunicación e Ingeniería Robótica, junto con las aplicaciones *Electric VLSI Design System*, *ngspice*, *QEMU* y las herramientas del *toolchain* para ARM de GNU y el entorno para hardware reconfigurable *APIO* para compartir algunos de los resultados que hemos obtenido.**

Palabras clave— **Software libre, laboratorio ubicuo, diseño electrónico.**

I. Introducción

LAS disciplinas de corte tecnológico asociadas al diseño de sistemas electrónicos necesitan desarrollar en el laboratorio resultados de aprendizaje que suelen hacer uso de aplicaciones informáticas de diseño electrónico con una alta tasa de especialización. El estudiantado encuentra generalmente estas aplicaciones instaladas y plenamente operativas en los equipos de los distintos laboratorios, sin embargo, este tipo de software presenta habitualmente grandes contrapartidas para que el estudiantado pueda trabajar con estas herramientas desde casa. Destacamos entre otras las siguientes: elevados costes de las licencias software y dificultad para su obtención, imposibilidad de ejecución remota, difícil instalación, aplicaciones informáticas sólo disponibles en un sistema operativo concreto y dificultad para acceder a manuales. Por tanto, se genera la necesidad imperiosa de dar solución a cada uno de los anteriores inconvenientes. La respuesta viene dada por el uso y el estudio de soluciones de software libre que ofrezcan la misma funcionalidad que sus equivalentes privativos, y posibiliten al estudiantado poder aprender desde cualquier ubicación (ubicuidad) y en cualquier horario. La relevancia de esta propuesta se potencia aún más en el actual entorno post-pandemia, por el hecho de que estamos habilitando la posibilidad de que se pueda enseñar-aprender con herramientas accesibles, gratuitas y bien documentadas, que se pueden instalar en distintos sistemas operativos (Linux, Windows y Mac) y, por tanto, no están sujetas a un laboratorio concreto. Con respecto a la viabilidad, hemos realizado pruebas de concepto en los dos últimos cursos académicos con resultados satisfactorios, y estamos listos para desplegar todas las soluciones en nuestras asignaturas.

Este trabajo presenta la experiencia docente definida por el uso de herramientas de software libre en las asignaturas relacionadas con el diseño de sistemas electrónicos en todas las fases de especificación/diseño de éste. Hemos tomado como ejemplo las siguientes dos asignaturas de la Universidad de Alicante: *Diseño de Circuitos y Sistemas Integrados* (DCSI) del Máster en Telecomunicación y *Sistemas Empotrados* (SE) del Grado en Ingeniería Robótica. DCSI tiene de una media de 15 a 20 estudiantes por curso, se imparte por la tarde y casi la mitad del estudiantado tiene algún trabajo en activo relacionado con la asignatura. Por su parte SE, asignatura de tercero, tiene una media de 60 estudiantes y 3 grupos de prácticas (20 personas/grupo) altamente motivados.

Pretendemos que el estudiantado pueda instalarse en casa y sin coste alguno estas aplicaciones para poder trabajar en la asignatura donde y cuando quiera. El trabajo está basado en otra experiencia similar en la Universidad de Granada llevada a cabo por uno de los autores de este artículo [1]. En el citado trabajo se exploraban distintas soluciones informáticas *top-down* dentro del ámbito del EDA (*Electronic Design Automation*) para realizar diseños electrónicos enfocados principalmente en la lógica reconfigurable.

La propuesta que aquí nos ocupa pretende:

- motivar la ubicuidad del estudiantado en el proceso de trabajar desde casa o cualquier otro sitio.
- flexibilizar al máximo el acceso (multiplataforma), en cualquier momento, al software que vamos a usar sin incurrir en ningún gasto de licencias.
- maximizar las posibilidades de encontrar documentación actualizada y de calidad del software que se va a usar.

[1]Dpto. de Tecnología Informática y Computación, Universidad de Alicante, e-mail: {amartinez, sergio, sanchez, luis.lucas, daniel.castro}@dtic.ua.es

[2]Dpto. de Ingeniería Electrónica y de Computadores, Universidad de Córdoba, e-mail: jmpalomares@uco.es

[3]A. Serrano Cases trabaja en el Barcelona Supercomputing Center (BSC) (e-mail: alejandro.serrano@bsc.es)

II. Metodología de trabajo

Con respecto a la metodología que se ha seguido para adecuar correctamente el proceso de aprendizaje del nuevo software a los requisitos y exigencias de cada asignatura, hemos seguido los siguientes hitos:

- Evaluación soluciones: Se ha procedido a preevaluar las aplicaciones de software libre más importantes en el diseño de sistemas electrónicos.
- Implantación: El estudiantado debe encontrar instalado en el laboratorio de la universidad exactamente el mismo software (libre) para su asignatura que puede instalar, si lo desea, en casa.
- Elaboración de material: El estudiantado encontrará en el campus virtual de la Universidad de Alicante (sitio web Moodle) tutoriales, manuales y prácticas guiadas para facilitar el acceso a cada una de las aplicaciones software empleadas.
- Soporte online: Hemos seleccionado las herramientas que mejor soporte tienen en Internet por la comunidad de usuarios en aquellos medios en los que el estudiantado busca información: YouTube, GitHub, páginas web de la comunidad "*maker*", etc.
- Impacto solución: Durante el curso se efectúan varias encuestas para estudiar el impacto de las nuevas herramientas en el proceso de enseñanza-aprendizaje.

Esta metodología requiere la realización de algunas tareas, por parte del profesorado, para adecuar y controlar perfectamente la adopción del nuevo software a las prácticas de las distintas asignaturas. Sistemas Empotrados (SE) es de primer cuatrimestre , y Diseño de Circuitos y Sistemas Integrados (DCSI) es de segundo. Las tareas se realizan a lo largo de todo el curso académico, y básicamente consisten en:

- Elaboración de tutoriales/prácticas guiadas para la instalación y uso de las herramientas ⇝ Partimos de tutoriales de cursos anteriores, elaborados por profesorado y estudiantes, que se pueden modificar/mejorar durante todo el cuatrimestre. Es un material, por tanto, colaborativo.
- Impartición de tutoriales/prácticas guiadas ⇝ Al inicio de cada práctica durante todo el cuatrimestre.
- Moderación de foros para la resolución de dudas relacionadas con las herramientas y materiales durante todo el cuatrimestre.
- Elaboración y ejecución de encuestas docentes al final del cuatrimestre y análisis de resultados (realimentación).

A. Metodología específica de DCSI

Las prácticas de la asignatura *Diseño de Circuitos y Sistemas Integrados* se vertebran en torno a la idea de que, al final del cuatrimestre, tenemos que ser capaces de diseñar y comprobar un componente electrónico digital completo acercándonos al mismo desde diferentes granularidades y niveles de abstracción. El resultado final es una hoja de especificaciones (*datasheet*) con todas las características calculadas (incluidas las típicas gráficas). En este sentido, el estudiantado aprenderá de forma paulatina los siguientes ítems:

- Diseño mediante *ngspice* de un componente digital.
  - ↪ Se describe en *netlist* de *SPICE/ngspice*.
  - ↪ Simulación con *ngspice* (simulación *transient*, DC, AC, OP, derivas de temperatura, etc.) para calcular las características dinámicas del componente: máxima frecuencia de funcionamiento, potencia disipada, retardo de las distintas señales, posibles problemas estructurales, variabilidad del componente con la temperatura, etc.
- Modelo del componente en HDL (*Hardware Description Language*) en *Verilog* o *VHDL*
  - ↪ Modelo de síntesis
  - ↪ Modelo de simulación + *test_bench* que integre todas las características dinámicas calculadas en la simulación *ngspice* y permita integrar el modelo en un simulador de *Verilog* o *VHDL*.

III. Herramientas libres y estrategias de aprendizaje empleadas

Herramientas de software libre que se usan: ngspice (simulador de circuitos analógicos basado en SPICE), Electric VLSI Design System (herramienta de diseño físico de semiconductores y componentes/sistemas electrónicos asistida por computadora), QEMU (simulador varias arquitecturas de microprocesador), APIO (entorno para el diseño de sistemas basados en FPGA).

A. Herramienta VLSI de diseño físico (layout)

El diseño físico VLSI o diseño de *layouts* de circuitos integrados es una competencia que demanda varias habilidades complejas al estudiantado: capacidad visoespacial (2D y 3D), visión jerárquica de un diseño electrónico, abstracción de modelos (rectángulos ⇝ transistores ⇝ diseño lógico ⇝ esquemáticos ⇝ lenguajes de descripción de hardware), diseño e interrelación de componentes electrónicos en distintos niveles de descripción/-detalle y granularidad, etc. Es por tanto deseable el empleo de una herramienta EDA (Electronic Design Automation) que aglutine todas estas *vistas* de un componente electrónico y permita la comprobación de su funcionalidad y características eléctricas mediante la integración de distintos motores de simulación.

Entre las distintas herramientas disponibles que cumplen en mayor o menor grado los condicionantes anteriores encontramos *Tanner EDA suite* [2], como referente fundamental dentro el ámbito académico en diseño y verificación de circuitos integrados en VLSI. Además, se integra bien con herramientas de simulación de circuitos analógico/digitales como *ngspice* y algunos de sus *front-ends* como *LTSpice*. Aun cuando no se trata de una herramienta de software libre multiplataforma, su penetración en el ámbito universitario la convierte en modelo o paradigma a seguir en la búsqueda de una solución no propietaria. Dentro del ámbito del tipo de software que no ocupa en este artículo encontramos soluciones como *Alliance/-Coriolis VLSI CAD Tools* [3], que aun cuando cumplen la premisa de ser software libre, su enfoque (menos integrado de *Tanner*) y complejidad nos han hecho descartarla.

La herramienta EDA elegida es *Electric VLSI Design System* [4], que tiene las siguientes características:

- Multiplataforma: Es una aplicación Java suministrada como un único archivo ejecutable empaquetado como `.jar`.
- Fácil instalación + ejecución: Al ser una aplicación java autocontenida en un contenedor `.jar`

B. Simulación de circuitos compatible con SPICE

SPICE (*Simulation Program with Integrated Circuit Emphasis* [5] es el programa estándar *de facto* para modelar y analizar el comportamiento de circuitos electrónicos analógicos, digitales y mixtos. Sus propios creadores/mantenedores lo definen como un programa de simulación de circuitos no lineal DC y transitorios, y de análisis AC lineal. Su núcleo de funcionamiento e interfaz de texto primaria está publicada como software libre, y muchas herramientas (mayoritariamente propietarias) lo integran y enriquecen con *front-ends* dedicados (capturas de modelos de circuitos esquemáticos, facilidades para definir y ejecutar varios tipos de análisis, capacidad de representación de la salida eléctrica de circuitos, etc.).

Aún cuando ciertos *front-ends* de SPICE como LTspice [6] tienen la capacidad de integrar el diseño de circuitos mediante esquemáticos y el acceso a la simulación SPICE-compatible dentro del mismo programa, también es cierto que es difícil encontrar software libre en este tipo de programas, o se trata de un software sólo compatible con algunos sistemas operativos, como Windows.

Por motivos didácticos, hemos optado por usar el programa de simulación *ngspice*, cuya entrada de diseño viene dada por un archivo de texto donde se especifica la lista de componentes interconectados y el tipo de simulación según las directrices de *SPICE*, que puede verse como un estándar para la definición de circuitos y análisis a efectuar. *Ngspice* [7] es, por tanto, un programa de simulación de circuitos electrónicos de código abierto basado en el estándar *SPICE* (*Simulation Program with Integrated Circuit Emphasis*). De hecho, el propio *ngspice* representa el núcleo de simulación pricipal de algunas herramientas comerciales dentro del ámbito EDA.

Las principales características de *ngspice* son las siguientes:

- Código abierto, gratuito y de calidad: Se distribuye bajo la licencia GNU GPL, lo que significa que es completamente gratuito y los usuarios tienen acceso al código fuente. Aunque no es necesario, no es difícil compilar el código fuente para particularizar nuestra propia copia del programa.
- Supone una mejora con el *SPICE* original en: rendimiento y precisión de la simulación, soporte para circuitos complejos, interfaz de usuario y nuevas alternativas de simulación.
- El acceso al código fuente lo hace atractivo a la academia, habiendo multitud de proyectos que lo utilizan como base o plataforma de investigación, o simplemente participan en su mejora para acelerar su procesamiento en computadoras modernas.
- Al hilo del anterior punto, su código fuente es sumamente extensible, ya que ofrece la posibilidad de agregar extensiones y modelos de dispositivos personalizados, lo que permite a los usuarios adaptar el programa a sus necesidades específicas y utilizar componentes especializados en sus simulaciones.
- Aplicación multiplataforma: Disponible para una amplia gama de plataformas, incluyendo Linux, Windows y macOS. Esto asegura que el estudiantado pueda utilizar el programa en su sistema operativo preferido.
- Amplia compatibilidad: compatible con los modelos de dispositivos estándar utilizados en la industria de semiconductores, lo que permite una simulación precisa de una amplia gama de componentes electrónicos, como transistores, diodos, circuitos integrados, entre otros.

Estrategias usadas con ngspice

- Se alienta al estudiantado a conocer un programa cuya entrada (descripción circuital dada en formato SPICE) viene dada por un archivo de texto que ellos mismos aprenden a diseñar e interpretar. Queremos subrayar que, aunque está permitido, las prácticas se hacen directamente transcribiendo los esquemas circuitales de sus libretas a un fichero SPICE sin que intermedie ningún programa tipo captura de esquemáticos (como tiene *PSpice o LTspice entre otros*).
- Los distintos análisis que se requieran (transitorios, DC, AC, temperatura, etc.) se añaden como órdenes al propio circuito, o bien, se utiliza *ngspice* en modo texto interactivo.
- *ngspice* genera gráficas para representar gráficamente los vectores de puntos calculados en cada análisis. Sin embargo, estas gráficas, aunque plenamente eficaces y resolutivas, no son muy interactivas y adolecen de cierta parametrización. Para sacar partido a esta situación, y dado que *ngspice* permite guardar en un archivo los vectores calculados para su posterior procesamiento, se da puntuación extra si se mejora la presentación de las gráficas usando otras herramientas como: la biblioteca *matplotlib + seaborn* de Python, Matlab, o cualquier otro programa.

En general, el estudiantado ha perdido el contacto con las aplicaciones cuya interfaz se define mediante la interacción con un intérprete de órdenes en modo texto, como *bash* y similares en Linux/MacOSX o *cmd* o *PowerShell* en Windows. Sin embargo, el forzar a usar este programa en modo texto, algo que supone un importante esfuerzo en un principio, hace que, a medida que pasan las sesiones de prácticas, se refuerce la confianza y se entienda mejor lo que se está diseñando. La contrapartida es un *front-end* con captura de esquemático enormemente recargado (iconos, menús, parámetros, etc.) que suele requerir entrenamiento y del que, a la larga, uno no sabe qué parámetros están activos y afectando a un cierto análisis que se pide. En las práctica pues, reforzamos la idea de: "*lo que tienes en el fichero es lo que describes y simulas*".

C. Herramientas basadas en HDL para el diseño y simulación de circuitos digitales

El proceso de enseñanza-aprendizaje de diseño de circuitos digitales dados en algún lenguaje de descripción de hardware (HDL) es un resultado de aprendizaje que

requiere mucho esfuerzo por profesorado y estudiantado. El inherente planteamiento paralelo que requieren estos lenguajes, como VHDL o Verilog, cuya proyección de un algoritmo en una FPGA es radicalmente distinto a cómo se plantearía en lenguajes asociados a un microprocesador (y por tanto con una repertorio de instrucciones como C/C++/C#/Java, Rust, Python, ...) es una tarea que requiere no poco esfuerzo. Además, las herramienta asociadas de desarrollo (o *toolchain*) suelen ser propietarias, grandes consumidoras de espacio en disco, lentas en una máquina relativamente moderna y monoplataforma.

En un intento de solventar cada uno de los problemas anteriores, hemos optado por el *toolchain Apio*. Este conjunto de herramientas se instalan como *plugins* de Python, son multiplataforma, software libre y funcionan mediante una sencilla interfaz de órdenes en modo texto para verificar, sintetizar, simular y cargar sus diseños Verilog. La instalación de *Apio* en Linux o Windows dura pocos minutos y deja un entorno de trabajo plenamente operativo. La contrapartida es que sólo es compatible con las FPGA que soporta *yosys* [8], su motor de síntesis. Además *Apio* está muy unido al movimiento *maker* [9] y al hardware libre [10], lo que incentiva enormemente al aula si se presenta de forma apropiada estos dos conceptos.

Estrategias docentes usadas con Apio

- Se realiza un tutorial en clase para la instalación de la herramienta y la correcta síntesis, simulación y configuración en la FPGA de un proyecto tipo *¡Hola mundo!*.
- Se emplea la *FPGA* libre *Alhambra* 1.1 cuyo diseño se muestra en clase mediante el programa de diseño de PCB *KiCAD*. Hemos comprobado que el acercamiento a estas nuevas herramientas y el hardware libre incentiva mucho al estudiantado y eleva el interés por la asignatura.
- Las prácticas son semiguiadas y cada curso se va realizando un proyecto distinto que involucre algún tipo de procesamiento por parte de la FPGA.
- Aunque el tamaño de la FPGA (en puertas lógicas) es reducido, la rapidez de síntesis y configuración del dispositivo contribuye muy positivamente al dinamismo de las prácticas, sobretodo si comparamos tiempos con soluciones propietarias.

La posibilidad de, por menos de 70€, tener en casa la misma FPGA que usamos en la asignatura, junto con todo el software necesario para el diseño y la simulación de circuitos digitales en Verilog, es un hecho que todos los años llevan a la práctica algunos/as estudiantes interesados.

D. *Herramienta de simulación de programas en distintas arquitecturas*

En la asignatura Sistemas Empotrados (SE) se estudian varias arquitecturas de procesador, y dos en concreto, con mayor profundidad: ARM y RISC-V [11]. Para generar y estudiar los distintos programas se emplea el *toolchain* correspondiente de GNU para una arquitectura concreta. Los estudiantes aprenden a desensamblar, generar código ensamblador a partir del C/C++, localizar y cambiar datos en los archivos binarios generados (en formato ELF), describir las dependencias dinámicas que tiene un ejecutable, etc. Para simular estos programas en distintas arquitecturas hemos escogido a *QEMU* [12]. *QEMU* (*Quick Emulator*) es un software de código abierto que proporciona una plataforma de emulación y virtualización de sistemas. Fue creado originalmente por Fabrice Bellard en 2003 y actualmente es mantenido por una comunidad activa de desarrolladores.

QEMU permite emular una amplia gama de arquitecturas de CPU, incluyendo x86, ARM, PowerPC, MIPS, RISC-V y más. Es posible ejecutar sistemas operativos completos e incluso aplicaciones diseñadas para una arquitectura específica en un entorno diferente, lo que lo hace muy útil para pruebas, desarrollo de software y virtualización. Además de la emulación a nivel de sistema, *QEMU* también ofrece capacidades de virtualización, lo que significa que puede funcionar como un hipervisor para crear máquinas virtuales. En este modo, *QEMU* trabaja junto con KVM (Kernel-based Virtual Machine) en sistemas Linux para proporcionar un rendimiento mejorado y una mayor eficiencia en la virtualización. *QEMU* se utiliza ampliamente en diversas áreas, como desarrollo de sistemas operativos, portabilidad de software, pruebas de seguridad, desarrollo de hardware y emulación histórica. También es una herramienta popular en la comunidad de código abierto debido a su flexibilidad, rendimiento y soporte para múltiples plataformas.

Cabe mencionar que *QEMU* tiene una interfaz de línea de órdenes muy potente que habilita la posibilidad de depuración del programa simulado mediante GNU GDB, característica ésta que usamos en la asignatura para realizar depuraciones *paso a paso* de, por ejemplo, un programa compilado para ARM en un anfitrión x86-64.

IV. Resultados experimentales

A. *Resultados de* layouts

Las figuras 1 y 2 representan el *layout* en 2 y 3D de un inversor CMOS y una simulación de su salida versus su entrada, respectivamente, realizadas por un estudiante el presente curso. Se trata de una primera práctica de aproximación al diseño VLSI con *Electric*.

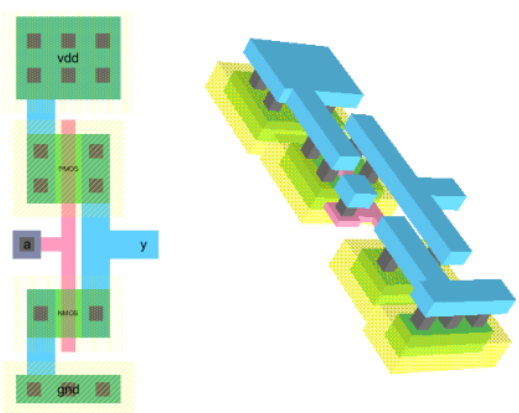


Fig. 1: Vista 2D y 3D de un inversor CMOS.

Como puede observarse, la vista en 3D, que es interactiva, ya que se puede girar, acercar y alejar en tiempo real, da es una utilidad de meridiana importancia para entender el diseño 2D del inversor. Una vez que los distintos

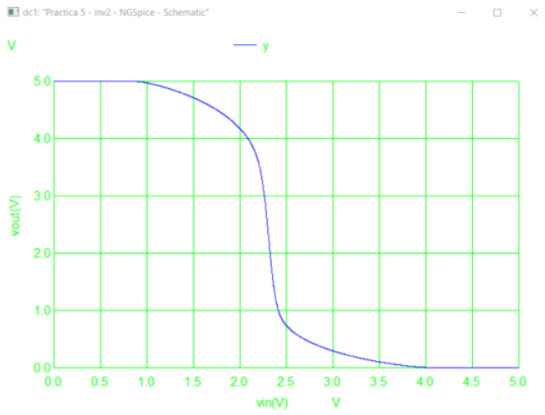


Fig. 2: Simulación con *ngspice* del circuito de la fig. 1. Se representa $V_{salida}(V_{entrada})$ entre 0 y 5 voltios.

grupos de prácticas han desarrollado el inversor CMOS, se les pide un diseño más complejo, como el multiplexador 4 a 1 que se muestra en la figuras 3, 4 y 5 (vista 2D, vista 3D y simulación *ngspice* mediante un transitorio de la salida frente a todas las entradas lógicas posibles).

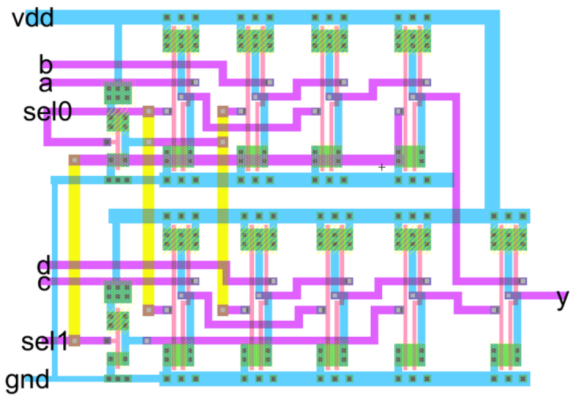


Fig. 3: Vista 2D de un multiplexador 4 a 1 en tecnología CMOS con *Electric*.

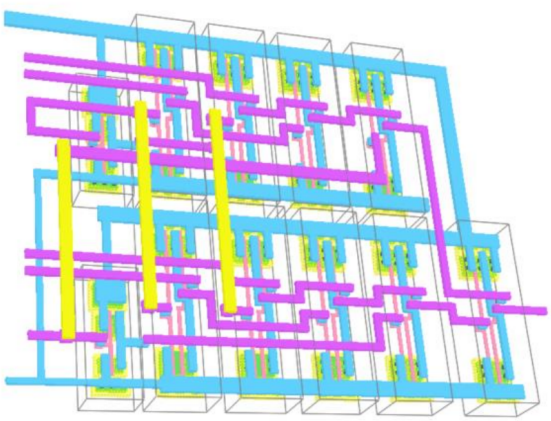


Fig. 4: Vista 3D de un multiplexador 4 a 1 en tecnología CMOS con *Electric*.

Para la realización de este último ejercicio de prácticas han tenido 4 horas (2 sesiones) completas. Dentro de este tiempo se incluye la comprobación mediante *ngspice* del correcto funcionamiento del circuito que se muestra en la citada figura. Aunque no se trata del mejor *layout* posible, hemos incluido este ejemplo para evidenciar cómo el estudiantado es capaz de realizar diseños relativamente complejos con *Electric* con un entrenamiento (prácticas)

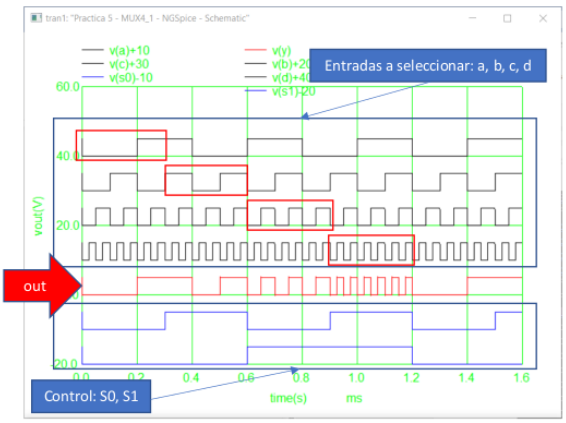


Fig. 5: Mux 4a1, comprobación funcionamiento con *ngspice*

de un tercio del cuatrimestre (5 sesiones de práctica de 2 horas cada una). Es particularmente interesante el diseño original que propone el grupo de prácticas del trazado de los dos primeros niveles de metalización (*meta1* y *meta2*).

B. *Resultados de* ngspice

La figura 6 muestra una entrega real de una práctica en la que se pedía diseñar una puerta XNOR2 en *SPICE/ngspice* de forma jerárquica y reutilizando componentes realizados en ejercicios previos. Nótese cómo se incluye el archivo `CMOSinverter.cir` (segunda línea), que define el componente `CMOSinverter` y se usa en el diseño (`XnotA`, `XnotB`, `Xxnor`) como subcircuitos (en terminología de *SPICE*.

```
.title XNOR 2 input

.include CMOSinverter.cir

* Power sources
VDD Vs GND DC 5
V_signalA A GND pulse(0 5 0 10ms 10ms 1 2)
V_signalB B GND pulse(0 5 0 10ms 10ms 0.5 1)

* Signal inversion
XnotA Vs GND A notA CMOSinverter
XnotB Vs GND B notB CMOSinverter
Xxnor Vs GND XOR XNOR CMOSinverter

* Active components
M1 XOR A AB AB cmosn
M2 AB B GND GND cmosn
M3 XOR notA notAB notAB cmosn
M4 notAB notB GND GND cmosn

.model cmosn nmos level=1 VTO=0.77 KP=7.7e-5 GAMMA=0.71
+ PHI=0.73 LAMBDA=0.0625 TOX=3e-8 LD=2e-7 U0=670
+ NSUB=2e16 CGDO=1.6908E-10 CGSO=1.6908E-10 CGBO=5.0932E-10
+ CJ=2.8901E-04 MJ=5.3532E-01 CJSW=1.4790E-10 MJSW=1.0000E-01
+ PB=9.9000E-01

M5 com notA Vs Vs cmosp
M6 XOR A com com cmosp
M7 com notB Vs Vs cmosp
M8 XOR B com com cmosp

.model cmosp pmos level=1 VTO=-1.1 KP=2.1e-5 GAMMA=0.355
+ PHI=0.66 LAMBDA=0.053 TOX=3e-8 LD=5e-8 U0=180
+ NSUB=5e15 CGDO=1.6260E-10 CGSO=1.6260E-10 CGBO=4.2445E-10
+ CJ=2.8670E-04 MJ=4.2120E-01 CJSW=1.6584E-10 MJSW=1.2657E-01
+ PB=7.3408E-01

* Analisis
.control
      tran 1ms 2
      set color0 = rgb:255/255/255
      plot v(A) v(notA)+6 v(B)+12 v(notB)+18
      plot v(A) v(B)+6 v(XNOR)+12
.endc
.end
```

Fig. 6: Diseño de una puerta XNOR2 con *SPICE/ngspice*.

V. Conclusiones

Este artículo ha compartido parte de una experiencia docente consistente en utilizar software libre en todas las etapas de diseño de sistemas electrónicos en las asignaturas *Diseño de Circuitos y Sistemas Integrados* (Máster en

Telecomunicación) y *Sistemas Empotrados* (Grado en Ingeniería Robótica). El estudiantado ha podido instalarse sin coste estas aplicaciones en casa y ha podido trabajar en la asignatura donde y cuando ha querido. El impacto de esta solución y su validación, que llevamos varios cursos potenciando, es totalmente satisfactorio, toda vez que comparamos resultados en otros centros con asignatura propietarias.

Agradecimientos

El presente trabajo ha sido financiado mediante el Programa de Redes de Investigación en Docencia Universitaria de la Universidad de Alicante. Convocatoria 2022-2024. Título de la red: *Integración de soluciones de software libre en todas las etapas del proceso de diseño de sistemas electrónicos: aplicación en Diseño de Circuitos y Sistemas Integrados y Sistemas Empotrados*.

Referencias

[1] Antonio Martínez-Álvarez, "Herramientas libres para trabajar con lenguajes de descripción de hardware: simulación y síntesis," in *V Jornadas de Computación Reconfigurable y Aplicaciones (JCRA 2005) dentro del I Congreso Español de Informática (CEDI 2005)*, CEDI, Ed., 2005, pp. 401–406.

[2] George P. Patsis, "Educational introduction to vlsi circuit design and simulation with tanner-eda tools," *IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)*, vol. 11, no. 1, pp. 9–21, 2021.

[3] "Alliance/Coriolis VLSI CAD Tools," `http://coriolis.lip6.fr`, Última visita: 6 de junio, 2023, Alicante, Spain.

[4] S.M. Rubin, "A general-purpose framework for cad algorithms," *IEEE Communications Magazine*, vol. 29, no. 5, pp. 56–62, 1991.

[5] "Spice - simulation program with integrated circuit emphasis," `http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/`, Último acceso: 7 de junio de 2023.

[6] Linear Technology, "Ltspice," `https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html`, Último acceso: 7 de junio de 2023.

[7] "Ngspice: Open source spice simulator," `http://ngspice.sourceforge.net/`, Último acceso: 7 de junio de 2023.

[8] "Yosys - open synthesis suite," `http://www.clifford.at/yosys/`, Accessed: Junio 7, 2023.

[9] Erica R Halverson and Kimberly Sheridan, "The maker movement in education," *Harvard educational review*, vol. 84, no. 4, pp. 495–504, 2014.

[10] Richard Stallman, *Free Software, Free Society: Selected Essays*, GNU Press, Boston, MA, 2002.

[11] David A. Patterson and Andrew Waterman, "The risc-v instruction set manual," Technical Report, 2019, Accessed: fecha de acceso.

[12] Fabrice Bellard, "Qemu: Fast processor emulator," *ACM Transactions on Modeling and Computer Simulation*, vol. 17, no. 4, pp. 41:1–41:17, 2007.

[13] ARM Holdings, "Arm architecture reference manual," Technical reference manual, ARM Limited, 2023, Accessed: 7 de junio, 2023.

IoT, tiempo real y control de errores

Towards Linux for safety-critical systems: System call execution time variability analysis

Markel Galarraga[1,2], Charles-Alexis Lefebvre[1], Jon Perez-Cerrolaza[1], Jose A. Pascual[2]

Abstract— **Multiple next-generation transportation and industrial domain applications target the development of safety-critical heteronomous or autonomous systems such as autonomous vehicles and collaborative robots. These systems exhibit a combination of escalating software complexity and the need to integrate diverse software stacks and Machine Learning algorithms that require high-performance computing devices. In this context, several research initiatives are paving the way toward the usage of Linux for developing such complex safety-related systems. However, the Linux kernel was not designed for safety-critical applications and has a significant execution path and execution time variability. This research presents a novel statistical approach to analyze the Linux system call execution time variability in combination with the execution path variability of a Linux-based safety-critical system. The proposed method is applied to a representative use case that implements an Autonomous Emergency Brake in a NVIDIA Jetson Nano board connected to the CARLA autonomous driving simulator.**

Keywords— **Linux, system calls, real-time, safety.**

I. Introduction

Safety-critical embedded systems are programmable electronic systems composed of electronics and software. Their failure can lead to catastrophic consequences such as human casualties (e.g., autonomous car accidents). Therefore, it is crucial to adhere to strict safety certification standards, like generic IEC 61508 or automotive ISO 26262, when developing and certifying such systems. During the last decades, diverse industry sectors have made substantial investments in the advancement of groundbreaking and highly intricate safety-oriented systems, such as autonomous vehicles. These systems have the potential to bring about revolutionary changes across multiple market domains, particularly within the realm of functional safety. Many of these pioneering safety-related systems exhibit notable characteristics, including stringent performance requirements, escalating software complexity, and the incorporation of Free/-Libre and Open Source Software (FLOSS) components, along with Machine Learning (ML) algorithms and associated software stacks. Consequently, a pressing need arises for an Operating System (OS) that can effectively accommodate the integration of these complex applications with the required software stacks, while simultaneously offering comprehensive support for functional safety, facilitating the compliance of functional safety standard requirements.

[1]Ikerlan Technology Research Centre, Basque Research and Technology Alliance (BRTA), e-mail: {mgalarraga,calefebvre,jmperez}@ikerlan.es.

[2]Faculty of Informatics, University of the Basque Country UPV/EHU, e-mail: joseantonio.pascual@ehu.eus.

Multiple works have been carried out aiming to join safety certification requirements and technological progress, mainly focusing on Linux [1], [2], [3], [4]. Linux is the leading OS from embedded systems to supercomputers and almost everywhere in between [5]. In addition, Linux has already been deployed in critical applications (e.g., telecommunication, banking) and in dependable systems such as spacecrafts (e.g., SpaceX Falcon 9, Dragon) [6]. Therefore, there is great interest in making its usage in safety-critical systems possible, since it would greatly help reduce development effort and costs. Moreover, Linux's open-source development model and its strict requirements have been argued by the SIL2LinuxMP project as possibly valid for IEC 61508 certification via Route 3_S, also known as "compliant non-compliant development" [2].

Likewise, the project Statistical Path Coverage (SPC) [3], [6], [7] proposes statistical methods to overcome the impossibility of achieving 100 % test coverage with Linux, i.e., testing all the possible software application paths, or, in the context of the work, testing every possible path the Linux kernel can take. Since 100 % test coverage can not be obtained in the Linux kernel, SPC focuses on extracting system call execution paths (traces) in a testing process, and statistically estimating the test coverage and the risk of untested paths appearing in operational time. The objective is to show how an argument for certification could be made if the probability of detecting a new path is extremely low (equivalent to the safety integrity level probability of failure).

However, SPC obviates timing and solely focuses on spatial determinism. Timing analysis is also essential for safety-critical systems, because they must meet hard real-time requirements. For example, an Autonomous Emergency Brake (AEB) must detect a possible collision in time to activate the brakes and slow down the vehicle, avoiding or minimizing the impact. Hence, timing constraints are as crucial as functional constraints. To the best of our knowledge, neither SPC nor other work has analyzed Linux system call time execution variability and combined it with the system call path analysis in the context of safety-critical systems. Our work aims to complement SPC by finding a relationship between system call execution paths and their execution times; and modifying the methodology to give it the ability to also analyze the real-time performance of a system. In this paper, we present the preliminary results of our work, which exhibit how execution traces and execution times are linked and how we aim to exploit them in future work. It is worth noting that the com-

plete real-time behavior and Worst-Case Execution Time (WCET) analysis of the safety-critical system must also take into consideration the temporal interferences due to the underlying computing device (e.g., multicore devices [4], GPUs [8]) and system software specific characteristics.

Finally, it is worth noting that the methodology we propose in this work does not aim to state whether Linux as a whole is or is not suitable for safety-critical systems. The objective is to propose a methodology that every use case could use. The obtained results only apply to the particular use case that has been tested.

The rest of this paper is organized as follows: Section II presents other works that have also focused on the usage of Linux in safety-critical systems. In Section III the methodology followed in this work is explained. Section IV presents our use case and the experimental set-up. Section V introduces the results and the discussion. Finally, Section VI concludes the paper and presents some lines of future work.

II. Related work

Multiple works have studied the usage of Linux in safety-critical systems. Statistical Path Coverage (SPC) [3], [6], [7] statistically analyzes the Linux kernel execution paths, and presents a methodology to estimate the obtained test coverage in a testing process and the risk of executing untested paths. The SIL2LinuxMP project [2] "aims at the certification of the base components of an embedded GNU/Linux RTOS[1] running on a single-core or multi-core industrial COTS[2] computer board." [9]. Various works serve as preliminary and as the base for the SIL2LinuxMP project [10], [11], [12], [13], [14] by analyzing the Linux kernel's spatial- and temporal-determinism. Previous to SIL2LinuxMP, the SIL4Linux project [15] implemented an architecture based on formal methods, tracing and profiling to study the use of Linux in SIL4 systems. In [16], authors test the differences in observed traces if different filesystems are used and if the system is under stress. Enabling Linux In Safety Applications (ELISA) [1] and Automotive Grade Linux (AGL) [17] are two other projects that aim to use Linux in safety-critical environments. The former is focused on helping companies include Linux in their safety-critical applications, while the latter's objective is building an open software base for the future development of automotive technology.

In the work [18], an argument is given to make projects such as SIL2LinuxMP focus not only on safety but also on security. The work [19] creates a modified and minimal Linux kernel for its integration in systems that must comply with the aeronautical ARINC 653 specification. In [20], the authors show a methodology to find sources of *Software Aging* in the Linux kernel, which refers to the tendency of systems to show degrading performance and eventually fail due to error conditions that accumulate over time. Finally, in the publication [21] the authors implement real-time containers over a Linux kernel patched with PREEMPT_RT.

To the best of our knowledge, the relationship between Linux kernel traces and system call execution time has not been studied by any other work. The work [13] presents some results showing the time distribution of a system call, but does neither analyze it nor give any conclusions in that regard. The publication [22] does analyze system call timings and justifies the usage of Linux for millisecond order deadlines. However, it is done in a single-core system with a pre-2.6 Linux kernel, so results no longer hold —the 2.6 version made the kernel preemptible and added multicore support—.

III. Methodology

The work presented in this project is based on Statistical Path Coverage (SPC). SPC proposes a methodology to statistically analyze the Linux kernel system call execution path variability and paves the way towards the testing of Linux-based safety-critical systems. Classic static analysis requested by the safety standards such as IEC 61508 [23] (e.g., 100 % test coverage), is considered unfeasible for a system as complex as Linux [3]. However, as technology progresses, so does the safety-critical applications' complexity. Autonomous Emergency Brake (AEB) systems are a prime example of these advances: they combine high-performance computing devices such as multi-core devices [4] and GPUs [8], operating systems, and machine learning (ML) models for braking cars autonomously in an emergency. Therefore, new methods must be defined beyond the currently described in safety standards. SPC states that obtaining 100 % test coverage in a system using Linux is unfeasible [6], and proposes a statistical method to analyze the Linux kernel execution paths (traces) and estimate the test coverage achieved in the testing process, i.e., the percentage of possible paths observed in testing. It also presents a way to estimate the risk of untested traces.

Throughout our work, we use the terminology found in work related to SPC. We summarize here the most used concepts in our work.

- Statistical Path Coverage (SPC): A methodology that proposes a statistical analysis of the Linux kernel to test its validity for safety-critical scenarios. The idea is to estimate the test coverage and the risk statistically.
- System call (or syscall): A function of the Linux kernel that serves as the interface between the user and kernel space.
- Trace: The path or chain of functions a system call follows from beginning to end. System calls can take different paths; thus, multiple different traces are followed by the same system call.
- Unique trace: A specific path a system call has taken. For example, a system call executed ten times can follow unique trace A eight times and

[1] Real-Time Operating System.
[2] Commercial-Off-The-Shelf.

unique trace B two times. It's important to note that the execution time of the system call is not considered to calculate path uniqueness, only the path taken is.

- Most common trace (MCT): The unique trace that happens the most for a certain system call. Normally, system calls tend to follow certain paths most of the time while following other paths much less. The formers can be considered the most common traces of the syscall, while the latters can be considered rare traces.
- Test campaign: An iteration of the testing process where the tested application is executed N times.

SPC uses DB4SIL2, a tool that works with the Linux function tracer (ftrace) and extracts and post processes system call traces from its output. DB4SIL2 is part of the SIL2LinuxMP project. Besides DB4SIL2, SPC also uses the isolation architecture proposed by SIL2LinuxMP [9], which is based on containerization to create isolated partitions over the same Linux kernel. We also use both the DB4SIL2 tool —with modifications to also capture execution times— and the isolation architecture in the work we present in this paper.

Taking SPC as the base for our work, we propose to extend the method by adding the execution time of traces. With this goal, we have modified DB4SIL2 to give it the ability to extract system call execution times from the ftrace output. Ftrace does output execution times together with traces, but the original DB4SIL2 ignores them when extracting data from the trace file. Our modifications make DB4SIL2 not remove timing information from the trace, but rather read it and save it with the trace. As stated in the definition of unique traces, we do not consider execution times when calculating the uniqueness of traces, i.e., unique traces are unique because they take a certain path, not regarding the time they need to execute it. This allows us to link every execution of unique traces with the time it took to complete them. Therefore, when all data has been processed, we have, for each execution of each system call, what path it followed and the time it took to complete it.

To check the validity of our results, we follow the validation methodology proposed by SPC, namely, using Extreme Value Theory (EVT), which is used to estimate if extending the testing process would give any more information regarding unique traces. If the estimation shows that more campaigns would not output more information, or the probability is extremely low, the length of the testing process is considered sufficient. However, this does not mean we can not obtain more information regarding system call execution times, so we must also check their validity. In their case, the value of untested execution times is as essential as the number of untested execution times we can expect with more campaigns, so using EVT with them as we do with unique traces may not be enough. For instance, if EVT results show that only one new value would be found if the testing process was extended, we do not know whether that value would be similar to other obtained values or very different and, thus, possibly problematic. For example, in a system where execution times generally take around 100 μs, an execution that took 101 μs would probably not be problematic, but one that took 1,000,000 μs probably would. Despite being aware of this, we chose for now to use EVT and focus on the number of new execution times we would obtain with more campaigns, leaving the optimization of the method as future work.

With the results validated, we analyze them by separating each system call and unique trace. The main idea of our analysis is to find if each unique trace has a particular timing behavior and whether we can facilitate timing analysis by focusing on each unique trace individually instead of focusing on system calls as a whole. We present our findings and conclusions in Section V.

IV. Use case

The use case we have developed extends the Automatic Emergency Brake system used in the SPC project [6] with a connection to the CARLA autonomous driving simulator. The AEB is executed in a Jetson Nano running Linux that communicates with a CARLA server via Ethernet. The application consists of two vehicles that are about to collide at an intersection. To give one of the cars the ability to brake and avoid the collision, we add our AEB system: CARLA communicates with the Jetson Nano and continuously sends frames captured by a camera in the car's front bumper. The Nano makes inferences using YOLO and detects the other vehicle, as seen in Figure 1. The information is passed to the motion control module, which calculates the distance to the car, and, depending on the current speed, decides whether the emergency braking should act. If so, it sends the braking order to CARLA, which makes the car brake, thus avoiding the collision. Once the other car has left the intersection, the braking order is canceled, and the car continues on its way. The SIL2LinuxMP architecture is implemented in the Jetson Nano to isolate the AEB, which allows its traces to be unaffected by the rest of the system [2].

The Linux kernel we use in the Jetson Nano is the Civil Infrastructure Platform SLTS v4.19, and we use YOLOv3 [24] for inference. The CARLA version we use is 0.9.9, and the virtual camera we add to the car's front bumper has a resolution of 640 x 360 pixels and a FOV of 90º.

The idea behind the application is to encapsulate the situation in which the AEB system needs to be activated and repeat it multiple times until we obtain enough results to use the statistical tools on them.

With the AEB, DB4SIL2 is executed in the Nano, with modifications that give it the ability to extract the execution time of a system call together with its trace. This application uses ftrace to obtain the kernel traces of the AEB and then post-processes them

Table I: For each system call, total number of executions, unique traces found, and approximate number of unique traces found per campaign.

System call	No. of executions	No. of observed unique paths	Approximate no. of observed unique paths per campaign
SyS_read	400,038	102,154	1,300
SyS_ioctl	32,033,482	227,147	8,000
SyS_sendto	210,869	3,096	175

Fig. 1: Frame from the camera in the front bumper of the car simulated in CARLA. On the left side of the image, the other car is shown, surrounded by a box that indicates that YOLO identifies the object as a car.

to extract the system calls and to give a unique MD5 hash identificator to each unique trace, i.e., to each unique chain of functions that system calls follow from beginning to end.

The scenario is run continuously throughout the testing process and is distributed in 100 campaigns of 20 executions each. The SPC methodology states that "the rule of thumb, currently used for the data set size, is that the sample size of a test-campaign needs to be significantly larger than the overall expected number of unique traces. We have currently chosen a factor of 1:10" [3]. For our use case, the rule is mostly followed, as we show with the results in Section V.

Regarding the number of campaigns, we chose to execute 100, due to the results of the data validation method proposed by SPC: in our use case, EVT provides stable results for the selected system calls with 100 campaigns. The technique allows for incrementally validating data and knowing whether more data is necessary, so the testing process can be tuned as needed.

V. Results and discussion

As stated in the previous section, 100 campaigns of 20 application runs have been executed. Due to the complexity of the application, certain system calls are executed thousands of times in each execution (`SyS_ioctl`, for example), while others are executed only once (`SyS_clone`, for example). Therefore, the analysis is focused on the main system calls of the application, i.e., those that are used for the main functionality of the application and are executed many times in a single run.

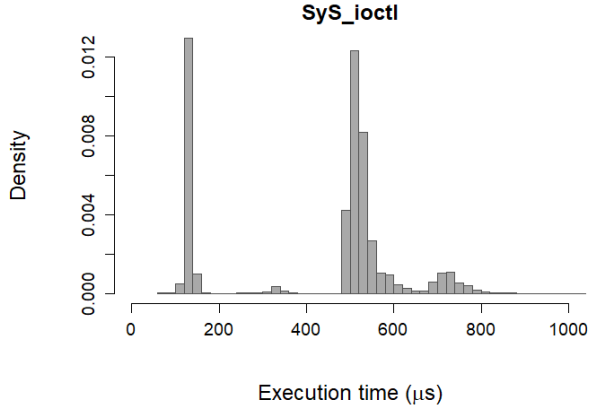
The system calls we have chosen are `SyS_read`, `SyS_ioctl`, and `SyS_sendto`. These three form the main part of the application because they are used, respectively, to receive the camera image from a socket, to communicate with the GPU when executing the inference, and to control the car via a socket depending on the inference and the motion control algorithm. Table I shows the total number of executions of each system call for the entire testing process, the number of unique traces found, and the number of unique traces seen per campaign. Note that the unique traces found in a campaign can be the same as those found in another, so the total number of unique paths is not the sum of the number of paths found in each campaign. As mentioned previously, SPC proposes that data should follow a 1:10 rule of expected unique traces to executions per campaign. For our use case, `SyS_ioctl` and `SyS_sendto` follow that, as we show in Table I. On the contrary, `SyS_read` only follows a factor of 1:4. However, since the analysis yields similar results for the three system calls, we believe the conclusions can be extended to `SyS_read`.

To estimate whether we have enough results to use statistical tools on them, we follow SPC's data validation methodology. As explained in Section III, SPC proposes using Extreme Value Theory (EVT) to verify that executing more test campaigns would not give more information regarding unique traces. Table II shows that our results for `SyS_ioctl` are stable with larger numbers of campaigns, so increasing the duration of the testing process would not yield more information. The other two system calls show the same behavior. As stated in SPC, in order to use EVT, data must be independent and identically distributed. The SPC qualitative analysis shows positive results for the three system calls. It is important to note that SPC removes the most common traces from the analysis and focuses on the rare traces. In our case, we intend to focus our analysis in the most common traces, so we can not remove them. Therefore, independence results are slightly worse for us.

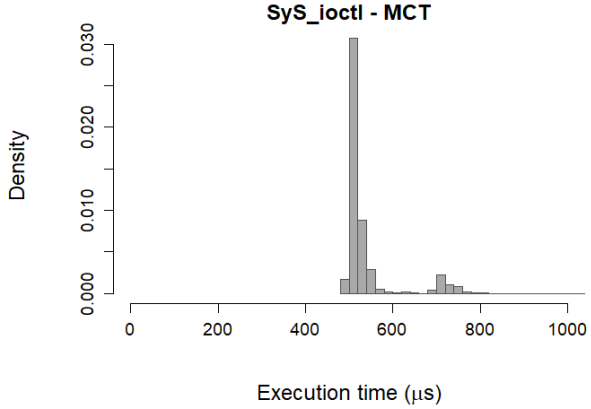
Table II: EVT results for `SyS_ioctl` unique traces, with quantile confidence intervals.

Tests	2.5 %	Estimate	97.5 %
500-Test	8313.228	8416.475	8559.685
1000-Test	8321.687	8433.677	8599.754
10000-Test	8334.674	8471.151	8702.498

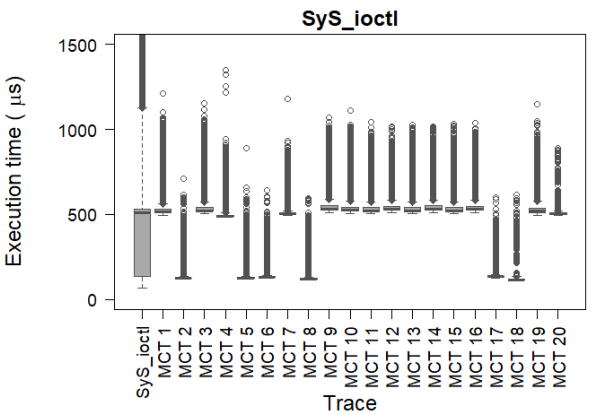
This validation only considers unique traces, not their execution time. Having enough data to study the possible paths does not mean there are sufficient results to also analyze execution times, they must be validated separately. In addition, Extreme Value Theory estimates the possibility of finding new information with more campaigns but does not tell us



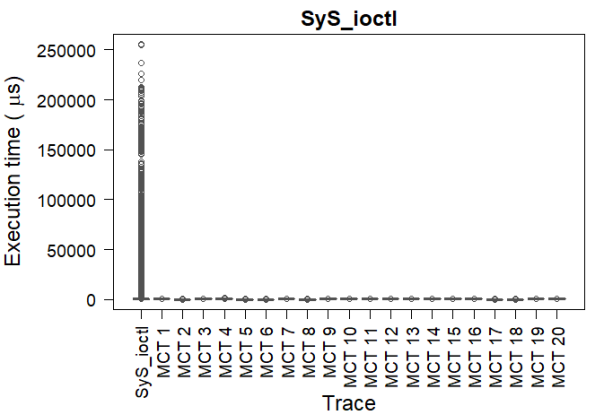
(a) Histogram of the `SyS_ioctl` system call, cut at 1000 μs.



(b) Histogram of the most common trace of `SyS_ioctl`.



(c) Distribution of the `SyS_ioctl` system call and its 20 most common traces, cut at 1500 μs.



(d) Distribution of the `SyS_ioctl` system call and its 20 most common traces.

Fig. 2: Timing results for the `SyS_ioctl` system call.

what that information is. This is enough in the unique traces case, because we care about finding new paths, but knowing the exact path is not essential. However, with execution times, we are interested in finding untested cases —not observed in the testing process—, and we also care about their actual value: we could find ourselves in a situation where EVT estimates that more campaigns would only give a couple of new execution times, but if those are, say, 1000 times bigger than the others, they should be considered. We intend to tackle this problem in the future, but we choose to use EVT for now because it is a useful tool to validate results in this state of the work. Since time is a continuous variable, we discretize our results into microseconds to carry out this analysis. If time was left as continuous, or discretized to too small of a unit, every result found would be unique, and the analysis would prove useless. We show in Table III the EVT results for the `SyS_ioctl` system call's execution times, where we can see that more campaigns would barely output new execution time values, proving the testing length is adequate. The other two system calls show similar behavior. Qualitative analysis (independent and identically distributed data) yields positive results for the three system calls.

Timing results are shown in Figure 2, which is structured as follows:

Table III: EVT results for `SyS_ioctl` execution times with quantile confidence intervals.

Tests	**2.5 %**	**Estimate**	**97.5 %**
500-Test	2857.382	2924.086	3011.874
1000-Test	2863.553	2938.435	3041.652
10000-Test	2876.595	2974.729	3132.263

(a) The histogram of the system call `SyS_ioctl`. Despite being cut at 1000 μs for comprehensibility, there are some values outside of that range (outliers).
(b) The histogram of only its most common trace (MCT) —the path that the system call follows the most—.
(c) Boxplots that show, first, the distribution of the entire system call (all of its paths) and then, its twenty most common paths, from more to less frequent, cut at 1500 μs.
(d) The same boxplots as in (c), but without being cut at 1500 μs, showing how the outliers of the system call as a whole are much further from the general values than the outliers of the most common traces.

What these figures show us is, first, in Figure 2a, that the system call as a whole does not seem to follow a normal distribution but instead a combination of multiple normal distributions. If we analyze uni-

que traces independently, as done in Figure 2b, they do seem to follow a normal distribution, but a different one for each unique trace, which explains the results of the entire syscall. The figure only presents the most common trace, but we can see in Figure 2c that this is true for all of the most common traces: they all show a distribution with the most values on the shortest times but with multiple outliers —a tail— on longer execution times. Furthermore, as seen in Figure 2d, the outliers of the most common traces are much smaller than the outliers of the system call as a whole, which means that the outliers furthest from the most common times occur only when a rare trace occurs, never with a common trace. This fact opens the possibility of a combined unique trace and execution time analysis, and we intend to exploit it in future work. Although we do not display them here, `SyS_read` and `SyS_openat` also show similar results, so conclusions are also valid for them.

Finally, we must note that we use a vanilla Linux kernel with no real-time patch. We intend to apply the PREEMPT_RT patch of the Real-Time Linux (RTL) project [25] in the future, and we expect to obtain execution time distributions with fewer and less disperse outliers, which will enable to better model results and estimations.

VI. Conclusions and future work

In this work, we add the Linux kernel system call execution time analysis to Statistical Path Coverage's (SPC) trace analysis methodology. Throughout the work, we show that system call paths and their execution times have a relationship we can exploit, which enables us to enhance the SPC methodology by adding the time variable, enhancing its applicability in real-time safety-critical Linux-based systems.

With an Autonomous Emergency Brake use case implemented in CARLA, in which an autonomous car detects an incoming vehicle with a camera and YOLO, and brakes before colliding with it, we have shown that the system calls executed in Linux show different timing behavior depending on the path followed, and that the most common paths show execution time distributions that seem easier to analyze if they are considered one by one, and not globally.

However, the behavior we see in this work does not mean every use case will behave the same. The obtained results only apply to the analyzed use case, and the conclusions can not be extrapolated to every Linux application. The goal of the methodology is not to say Linux is suitable or unsuitable for every safety-critical scenario, but instead to give a tool that can be used in every use case.

Finally, it is important to note again that results are preliminary, and work must be done in the future to follow the ideas presented in this paper. First, the data collection process must be tuned and optimized, and a way must be found to validate the execution time results in both quantity and quality, together with a method to model those results and combine them with the SPC methodology. In addition, the PREEMPT_RT patch must be added to the system to bring the use case even closer to reality. While we keep working on a methodology that includes the abovementioned capabilities, we believe the presented results give promising information and serve as a basis to statistically analyze Linux system call execution times and traces together.

Acknowledgments

This work is supported by the Spanish Ministry of Economy and Competitiveness MINECO (PID2019-104966GB-I00) and by the Economic Development and Infrastructure Department of the Basque Government (Emaitek program and projects ELKARTEK21/89 and IT1504-22).

References

[1] "ELISA - Advancing Open Source Safety-Critical Systems," https://elisa.tech/.

[2] Andreas Platschek, Nicholas Mc Guire, and Lukas Bulwahn, "Certifying Linux: Lessons Learned in Three Years of SIL2LinuxMP," in *Embedded World*, 2018.

[3] Imanol Allende, Nicholas Mc Guire, Jon Perez, Lisandro G. Monsalve, and Roman Obermaisser, "Towards Linux based safety systems—A statistical approach for software execution path coverage," *Journal of Systems Architecture*, vol. 116, June 2021.

[4] Jon Perez-Cerrolaza, Roman Obermaisser, Jaume Abella, Francisco J. Cazorla, Kim Grüttner, Irune Agirre, Hamidreza Ahmadian, and Imanol Allende, "Multi-core Devices for Safety-critical Systems: A Survey," *ACM Computing Surveys*, vol. 53, no. 4, Sept. 2020.

[5] The Linux Foundation, "2021 Linux foundation annual report. New Horizons for Open Source," https://www.linuxfoundation.org/resources/publications/linux-foundation-annual-report-2021.

[6] Imanol Allende, Nicholas M. McGuire, Jon Perez-Cerrolaza, Lisandro G. Monsalve, Jens Petersohn, and Roman Obermaisser, "Statistical Test Coverage for Linux-Based Next-Generation Autonomous Safety-Related Systems," *IEEE Access*, vol. 9, pp. 106065–106078, 2021.

[7] Imanol Allende, Nicholas Mc Guire, Jon Perez, Lisandro G. Monsalve, Javier Fernandez, and Roman Obermaisser, "Estimation of Linux Kernel Execution Path Uncertainty for Safety Software Test Coverage," in *Proceedings -Design, Automation and Test in Europe, DATE*. Feb. 2021, vol. 2021-February, pp. 1446–1451, Institute of Electrical and Electronics Engineers Inc.

[8] Jon Perez-Cerrolaza, Jaume Abella, Leonidas Kosmidis, Alejandro J. Calderon, Francisco Cazorla, and Jose Luis Flores, "GPU Devices for Safety-Critical Systems: A Survey," *ACM Computing Surveys*, vol. 55, no. 7, pp. 147:1–147:37, 2022.

[9] OSADL, "SIL2LinuxMP: OSADL - Open Source Automation Development Lab eG," https://www.osadl.org/SIL2LinuxMP.sil2-linux-project.0.html.

[10] Nicholas Mc Guire, Peter Okech, and Georg Schiesser, "Analysis of inherent randomness of the Linux kernel," in *Proceedings of the 11th Real-Time Linux Workshop*, Dresden, 2009, OSADL.

[11] Peter Okech and Nicholas Mc Guire, "Analysis of Statistical Properties of Inherent Randomness," in *Proceedings of the 12th Real Time Linux Workshop*. 2010, OSADL.

[12] Peter Okech, Nicholas Mc Guire, and William Okelo-Odongo, "Inherent Diversity in Replicated Architectures," in *Proceedings of Student Forum*, Paris, France, 2015.

[13] Peter Okech, Nicholas Mc Guire, Christof Fetzer, and William Okelo-Odongo, "Investigating Execution Path Non-determinism in the Linux Kernel," in *Proceedings of the 15th Real Time Linux Workshop*, Lugano-Manno, Switzerland, 2013, OSADL.

[14] Peter Okech, Nicholas Mc Guire, and Christof Fetzer, "Utilizing Inherent Diversity in Complex Software Systems," in *Proceedings of the Australian System Safety Conference (ASSC2014)*, 2014.

[15] Lijuan Wang, Chuande Zhang, Zhangjin Wu, Nicolas Mc Guire, and Qingguo Zhou, "SIL4Linux: An attempt to explore Linux satisfying SIL4 in some restrictive conditions," in *Proceedings of the 11th Real-Time Linux Workshop*, Dresden, 2009, OSADL.

[16] Yucong Chen, Xianzhi Tang, Shuaixin Xu, Fangfang Zhu, Qingguo Zhou, and Tien-Hsiung Weng, "Analyzing execution path non-determinism of the Linux kernel in different scenarios," *Connection Science*, vol. 35, 2023.

[17] "AGL - Automotive Grade Linux," https://www.automotivelinux.org/.

[18] Giuseppe Procopio, "Safety and Security in GNU/Linux Real Time Operating System Domain," in *Advances in Intelligent Systems and Computing*. 2020, vol. 925, pp. 245–254, Springer Verlag.

[19] João Craveiro, José Rufino, Carlos Almeida, Rui Covelo, and Pedro Venda, "Embedded Linux in a partitioned architecture for aerospace applications," in *IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*, 2009, pp. 132–138.

[20] Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo, "Software aging analysis of the linux operating system," in *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, 2010, pp. 71–80.

[21] Marcello Cinque and Domenico Cotroneo, "Towards Lightweight Temporal and Fault Isolation in Mixed-Criticality Systems with Real-Time Containers," in *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN-W 2018*. July 2018, pp. 59–60, Institute of Electrical and Electronics Engineers Inc.

[22] Steven A. Finney, "Real-time data collection in Linux: A case study," *Behavior Research Methods, Instruments, & Computers*, vol. 33, no. 2, pp. 167–173, May 2001.

[23] International Electrotechnical Commission, *IEC 61508 (1-7): Functional safety of Electrical/Electronic/Programmable Electronic safety-related systems (Second edition)*, 2010.

[24] Joseph Redmon and Ali Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[25] "Realtime:rtl:start [Wiki]," https://wiki.linuxfoundation.org/realtime/rtl/start.

Solución IoT de bajo costo para gestionar la contaminación del aire en zonas urbanas

Willian Zamora[2], [1] Mike Machuca, Johnny Larrea, José Arteaga-Vera, Edison Almeida [2]

Resumen— **La combustión de diversos materiales, procesos industriales, vehículos automotores, entre otros, produce contaminantes en el aire que perjudica la salud humana. Es así que, el monitoreo de la contaminación del aire se ha convertido en un requisito esencial para las ciudades de todo el mundo. Sin embargo, las estaciones de monitoreo fijas tradicionales presentan desafíos en términos de costo y dificultad de instalación. En este contexto, soluciones de monitoreo basadas en tecnologías emergentes como IoT y sistemas embebidos han presentado un creciente interés. En este artículo, se describe, una solución móvil de bajo costo para el monitoreo de la contaminación del aire. La solución combina el diseño de una tarjeta electrónica con un Raspberry Pi. La tarjeta permite conectar hasta 6 sensores de gases electroquímicos de la marca Alphasense. Los datos recopilados son almacenados en una base de datos local y enviado directamente al cloud. Para validar nuestra propuesta se ha instalado la solución en dos ubicaciones distintas dentro la ciudad de Manta, Ecuador. La finalidad es analizar el comportamiento de la solución en zonas de alta circulación vehicular. En particular se ha analizado el gas monóxido de carbono (CO), dióxido de azufre (SO_2), y óxidos de nitrógeno (NO_2). Resultados experimentales muestran que los valores obtenidos a las del gas NO_2 superan los umbrales permitidos según la Organización Mundial de la Salud (OMS).**

Palabras clave— **IoT, Alphasense, contaminación, aire.**

I. Introducción

La contaminación del aire es uno de los problemas ambientales más apremiantes. La emisión de contaminantes atmosféricos ha aumentado debido a la mayor demanda de energía y la industrialización. Estos contaminantes, que incluyen gases tóxicos y partículas en suspensión, tienen graves consecuencias tanto para el ser humano como para el medio ambiente [1], [2]. Las actividades industriales, la generación de energía, el transporte y la quema de combustibles fósiles son algunas de las principales fuentes de contaminación. Estos contaminantes incluyen dióxido de carbono (CO_2), monóxido de carbono (CO), óxidos de nitrógeno (NO_2), dióxido de azufre (SO_2), compuestos orgánicos volátiles (COV) y partículas finas ($PM_2$5). La contaminación del aire es dañina. Las enfermedades respiratorias, cardiovasculares y neurológicas pueden ser causadas por la exposición prolongadas a altos niveles de contaminantes. Las ciudades con aire irrespirable son indicativas de la magnitud del problema de la contaminación del aire. En consecuencia, los gobiernos, las organizaciones internacionales y las comunidades reconocen la importancia de abordar este problema juntos. Antes esto, es importante consultar la legislación y regulaciones ambientales locales y de las organizaciones internacionales para obtener información precisa sobre índices permisibles de los contaminantes del aire. Los organismos gubernamentales encargados de la calidad del aire suelen proporcionar información actualizada sobre los estándares aplicables y los límites permitidos. En este contexto, la ciudad de Manta, ubicada en la costa del Pacífico sur de Ecuador, no está exenta de estos contaminantes. Es así que en la actualidad la ciudad no cuenta con estudios oficiales o informes locales del monitoreo de la calidad del aire. Esta ciudad, con una población aproximada de 250 mil habitantes, posee una zona urbana caracterizada por una alta concentración vehicular. En consecuencia, resulta de vital importancia realizar un análisis exhaustivo de los niveles de concentración de contaminantes atmosféricos en dicha localidad. En respuesta a esta creciente preocupación, soluciones basadas en tecnologías emergentes como el Internet de las cosas (IoT) representan una respuesta innovadora y prometedora para abordar estos desafíos. En este sentido los sistemas embebidos, compuestos por dispositivos electrónicos compactos y de bajo consumo, desempeñan un papel fundamental en el desarrollo de soluciones de monitorización de la calidad del aire. Estos sistemas permiten integrar sensores personalizados en dispositivos portátiles y fáciles de implementar. Al combinar estos sensores, dispositivos inteligentes y la conectividad inalámbrica, el IoT permite la recopilación en tiempo real de datos relacionados, como la concentración de contaminantes, entre otros. En este articulo proponemos el desarrollo de un dispositivo que permite la monitorización ambiental del aire basado en IoT. Nuestra solución combina el diseño de una placa electrónica que combina un Raspberry Pi y sensores de marca Alphasense. La solución permite almacenar y mostrar datos en tiempo real a través de los sensores de: dióxido de carbono (CO_2), monóxido de carbono (CO), óxidos de nitrógeno (NO_2), dióxido de azufre (SO_2), ozono (O_3) y material partículado ($PM_2$5 - $PM_1$0). Para evaluar la propuesta se ha analizado 3 sensores en dos ubicaciones distintas en la ciudad de Manta, Ecuador. Este documento está organizado como sigue: En la sección II se presentan los trabajos relacionados. La sección III describe la arquitectura propuesta para la monitorización de la calidad del aire usando sensores de bajo coste. Además, se describen los sensores analizados para este estudio. En la sección IV se muestra la evaluación y validación de la propuesta, destacando los resultados obtenidos mediante gráficos estadísticos en dos ubicaciones

[1]FCVT, Universidad Laica Eloy Alfaro de Manabí, Email: {willian.zamora, mike.machuca, johnny.larrea, jose.arteaga, edison.almeida} @uleam.edu.ec

[2]Department of Computer Engineering, Universitat Politècnica de València, Email: wilzame@posgrado.upv.es

particulares. Por último, la sección V presenta las conclusiones y trabajos futuros.

II. Estado del arte

Soluciones para la monitorización de la calidad del aire de bajo coste han ganado popularidad, en consecuencia existen numerosos trabajos de investigación relacionadas a esta temática. Recientemente, se han realizado revisiones sistemáticas que evalúa los sensores de calidad del aire de bajo costo, aspectos como: arquitectura, selección de sensores, calibración entre otros son estudiados [3], [4], [5]. A continuación se describen algunas soluciones relacionadas.

En [6], implementa una red de nodos de sensores autónomo en la ciudad de Cambridge (Reino Unido). Para esto, los autores utilizan sensores electroquímicos a nivel de ppb (partes por mil millones) y analizan los gases de NO, NO_2 y CO en zonas de alta densidad. Sus resultados muestran la potencialidad de estos enfoques obteniendo valores de calidad del aire con una alta granularidad.

En trabajo previos[7] se presentó una solución que permite recoger la contaminación del aire mediante sensores móviles. La solución combina una arquitectura que usa un dispositivo comercial de la marca Libelium, Raspberry pi, teléfonos inteligentes y un servidor web. En general, se analizó el impacto del gas O_3 y su mejor estrategia de muestreo en la ciudad de Valencia, España. En particular, se buscó determinar una buena estrategia de estimación de la distribución del contaminante de O_3 en una área especifica. En [8] se propuso una solución de bajo coste para medir la calidad del aire en entornos urbanos.

Mas adelante, autores [9] presenta el desarrollo de un dispositivo sensor inalámbrico de bajo costo para el monitoreo en tiempo real de la calidad del aire. Los autores utilizan sensores de gas electroquímicos de baja potencia (NO_2, y O_3) y además, agrega un sensor para obtener los valores de material partículado (PM_25). Su solución propuesta usa un Raspberry Pi para adquirir, almacenar y cargar los datos al servidor web.

En [10] implementaron seis dispositivos de Internet de las cosas (IoT) para monitorizar la calidad del aire en dos ubicaciones dentro de la ciudad de Southampton, Reino Unido. En particular, cada uno equipado con cuatro sensores PM (material partículado) de bajo costo diferentes y transceptores de red inalámbrica LoRaWAN para probar la cobertura a escala de la ciudad. Sus resultados demuestran que no todos los sensores PM son iguales y que es posible lograr una correlación razonable con las estaciones de referencia. Recientemente, en [11] autores muestran una solución para monitorear la calidad del aire en una planta donde opera un horno de fusión. El diseño permite el uso de sensores comerciales de bajo costo. Su arquitectura utiliza hardware y software no propietario. Los datos recopilados son transmitidos por los nodos sensores a través del canal LoRa y transmitidos a un servidor LoRaWAN remoto.

Respecto con la marca de los sensores de gases utilizados, en [12] autores evaluaron los sensores electroquímicos de bajo costo de Alphasense Ltd., para la monitorización en tiempo real de los gases CO, NO_2 y O_3. Se encontró que las respuestas de los sensores eran altamente lineales en relación a las concentraciones de gas medidas con instrumentos de referencia. De igual manera, los autores [13], indican el potencial de las mediciones de contaminación del aire usando esta marca y en particular, se enfocan en la validación de mediciones obtenidas de los gases de CO, NO, NO_2 y O_3.

Respecto con trabajos previos nuestra propuesta integra hasta 6 sensores de gases, material partículado, sensores de temperatura, humedad y presión, en una tarjeta electrónica propia. Además, nuestra solución busca enfocarse en zonas urbana de la ciudad de Manta, Ecuador, que a mayor de nuestro conocimiento no existen estudios previos de los niveles de contaminación del aire.

III. Materiales y métodos

En esta sección se describe la propuesta desarrollada así como los gases analizados para su evaluación y su método de comparación con la normativas vigente local e internacional. A continuación se indican mas detalles.

A. Arquitectura propuesta

La arquitectura de nuestro sistema de monitorización propuesto cuenta con 3 componentes principales. El primero corresponde a la placa electrónica el mismo que permite conectar los diferentes sensores de bajo costo. El segundo corresponde al dispositivo que permite la captura y almacenamiento de los datos obtenidos por los sensores y el tercer componente es el servidor web. En la figura 1 se muestra la arquitectura del sistema de medición de la calidad del aire. A continuación más detalle.

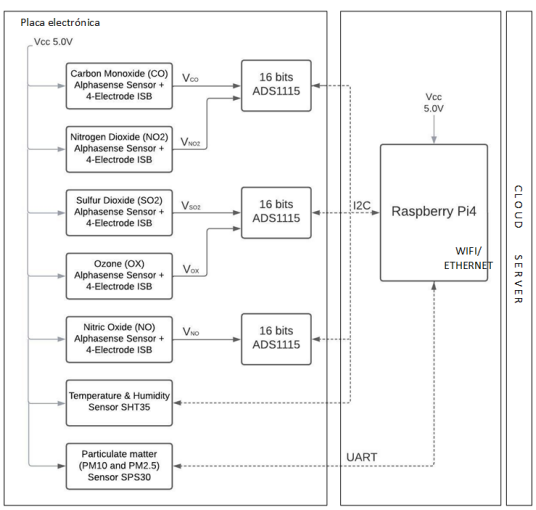


Fig. 1: Análisis de sensores de gases en la ubicación A

A.1 Placa electrónica

Nuestra placa electrónica permite conectar hasta 6 sensores de gases electroquímicos de la marca Alphasense. En particular, en hemos utilizado los gases de

(CO [14] , NO_2 [15], SO_2 [16], O_x [17] y NO [18]). Cada uno con un circuito adaptador de señal de la misma marca (4-Electrode ISB). En la figura 2, se muestra la tarjeta electrónica con los gases utilizados. El uso del circuito Alphasense ISB permite reducir aún más el ruido. Además, posee un sensor de temperatura y humedad (SHT35) y un sensor de material partículado (SPS30 [19]) que mide PM_10 y PM_25 . Para convertir las señales analógicas provenientes de los sensores de electroquímicos se utiliza un conversor analógico-digital de 16 bits (ADS115). Todos los sensores, a excepción del sensor de material partículado (utiliza protocolo serial UART). Este componente se comunican con el de adquisición de datos mediante el protocolo de comunicación serial I2C. Todo el sistema se alimenta a 5V.

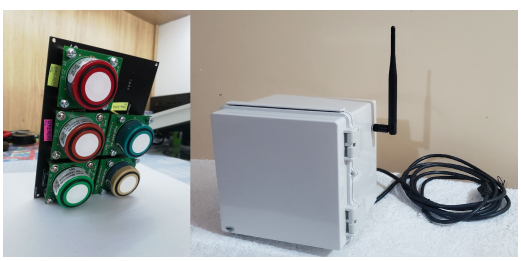


Fig. 2: Tarjeta electrónica y la caja para exteriores.

A.2 Adquisición de datos

Para el proceso de adquisición de los datos se ha usado una Raspberry Pi 4. El dispositivo se conecta mediante protocolo de comunicación I2C permitiendo obtener los datos de los diferentes sensores conectados a la tarjeta electrónica. El Raspberry Pi 4 además, permite procesar, almacenar los datos en la memoria microSD interna. Para el almacenamiento local se uso SQL lite, y mediante un servicio permite enviar los datos a la nube y transmitir en tiempo real la información capturada. La comunicación ha sido mediante la red WiFi y Ethernet.

A.3 Cloud Server

El Cloud Server permite visualizar los datos en tiempo real. Para esto, primero los datos se almacenan en una base de datos en el servidor. En este caso se ha usado PostgreSQL v12. Posteriormente, se puede acceder a ellos con una interfaz web basada en Grafana v9.2.18. Esto permite observar los datos obtenidos basadas en plataforma de tablero (dashboard) de código abierto.

B. Método

Una vez descrita la propuesta, a continuación se describen los gases analizados para nuestras pruebas y describimos normativas de valores máximos permitidos de algunos contaminantes del aire. El objetivo que se busca es comparar los valores obtenidos de nuestra propuesta por los solicitados por organismo nacional e internacional.

(a) Ubicación A (b) Ubicación B

Fig. 3: Puntos de análisis, Manta, Ecuador

Tabla I: Valores máximos permitidos en 1 hora y 24 horas según organismos A.M97 y AQG 2021

Gas	A.M. 97 (1 hora) ug/m3	AQG Nivel (24 horas) ug/m3
CO	30000	4000
SO_2	500	40
NO_2	200	25

B.1 Gases analizados

En este articulo nos referimos al análisis de 3 de los 5 gases instalados en nuestra propuesta. En particular, se analiza el CO, NO_2, y SO_2 que principalmente se forman como resultado de la combustión de combustibles fósiles. Además, que estos gases al estar en una exposición continua puede causar enfermedades especialmente en las vías respiratorias. En general, la solución propuesta mide los valores de estos gases en parte por billón (PPB), y para llevarlos a ($\mu g/m^3$) para su posterior análisis se ha utilizado la ecuación 1[20].

$$Concentración(ug/m3) = 0{,}0409x(ppb)xMH \quad (1)$$

En donde MH [21] es el peso molecular de cada uno de los gases estudiado, y el ppb es el valor obtenido por el sensor de gas.

B.2 Directrices de valores permitidos

Para los diferentes contaminantes se han establecidos regulaciones y estándares ambientales en cada país o región, donde se indican los limites máximos permisibles para cada uno de estos gases contaminantes. Al conocer este indicador esto nos permite comparar los valores obtenidos por nuestra propuesta. Para esto se ha tomado como referencia internacional las directrices mundiales de calidad del aire de la OMS [1] y a nivel nacional el acuerdo ministerial relacionada a la norma de calidad ambiental [22]. En la tabla I se muestran los valores limites promedios permitidos para los gases en micro-gramos por metro cúbico ($\mu g/m^3$) para 1 hora y 24 horas respectivamente.

IV. Resultados experimentales

En esta sección se describen los resultados obtenidos. En especial se han tomado como referencia dos

Tabla II: Análisis de gases ubicación A y B en 7 días.

Lugar/gas sensor		Media (ppb)	Max (ppb)	Min (ppb)	Std (ppb)
A	CO	52.19	347.79	8.30	46.46
	NO_2	9.89	65.90	1.57	8.80
	SO_2	0.97	6.11	0.06	0.83
B	CO	50.06	281.55	7.81	34.15
	NO_2	9.48	53.35	1.48	6.47
	SO_2	1.05	6.13	0.14	0.67

ubicaciones una frente al mar y otra distante a la misma. Ambas ubicaciones son lugares de alta demanda de circulación vehicular. En la figura 3 se muestra los puntos de referencia. Se ha tomado 7 días consecutivos de lunes a domingo para cada ubicación. Las evaluaciones que se hicieron son i) Análisis del comportamiento de los gases CO, NO_2, y SO_2 por semana y en cada ubicación. ii) Evaluación del día y hora de mayor concentración y iii) se compararon los resultados obtenidos del análisis con los límites establecidos por la normativa correspondiente. Esto permitirá determinar si las concentraciones de gases se encuentran dentro de los umbrales permitidos o si superaban los límites recomendados para garantizar la calidad del aire. A continuación más detalles de los resultados.

A. Análisis de sensores de gases semanal

En esta sección, se analiza el impacto de la lectura obtenida de los sensores de gases disponibles en la propuesta. En particular, se ha evaluado los gases de: CO, SO_2, y NO_2 en dos ubicaciones distintas indicada previamente. El objetivo que se persigue es determinar el impacto de cada uno de estos gases en el sitio analizado. Además, se busca elegir un gas para tomar las siguientes decisiones. Para este propósito, se ha tomado la lectura de 7 días consecutivos de lunes a domingo para cada ubicación. La frecuencia de muestreo establecida fue de 1 lectura cada 5 minutos. En general, se obtuvo un aproximado de 2020 registros por ubicación.

En las figuras 4-5 se muestran los valores obtenidos en ppb por cada uno de los sensores de gases analizados. En general, se observa que en las ubicaciones A, y B, el monóxido de carbono (CO) presenta mayores niveles de concentración de ppb respecto con el gas de SO_2 y el NO_2 respectivamente. Por el contrario, para ambas ubicaciones los niveles de concentración de ppb bajos se observan para el dióxido de azufre. En particular, en la figura 5, se observa una mayor uniformidad de concentración del gas NO_2 durante los 7 días analizados.

En la tabla II, se observa los valores estadísticos obtenidos para este primer análisis en las 2 ubicaciones. En particular, se puede observar que los valores de desviación estándar para cada uno de los sensores de la ubicación B es inferior respecto con la ubicación A. Adicionalmente, se observa que la temperatura media para la ubicación B es mayor que la ubicación A, esto se debe por que la información procesada es del mes de noviembre y la de la ubicación A es del mes de octubre, ambas del año 2022.

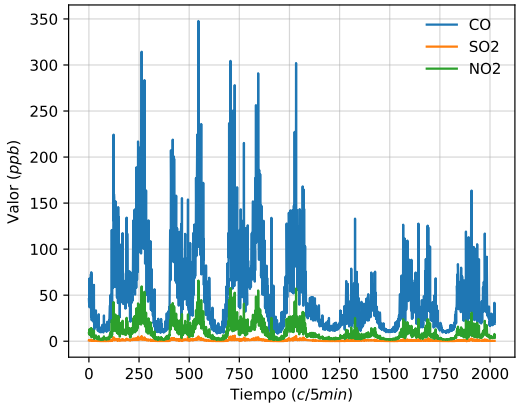


Fig. 4: Análisis de sensores de gases en la ubicación A

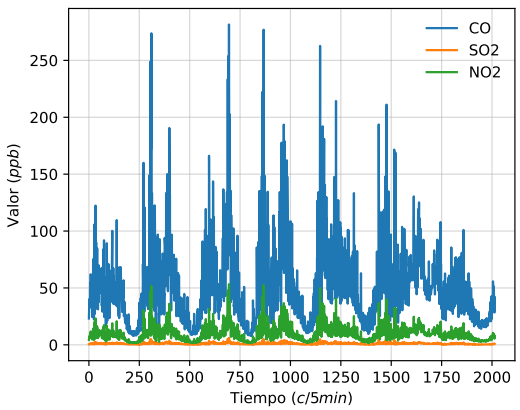


Fig. 5: Análisis de sensores de gases en la ubicación B

B. Análisis de sensores de gases por día y hora

En la sección anterior se hizo un análisis general del comportamiento de los gases en dos ubicaciones distintas (A-B). En esta sección, se busca identificar el día y hora de la semana de mayor concentración media de ppb. Para este propósito, en primer lugar se ha procesado la información para clasificar cada registro, identificando el día y hora de la semana que le corresponde. En segundo lugar, se ha seleccionado al sensor de gas SO_2, la principal razón de esto es que, en ambas ubicaciones A y B presenta la menor desviación estándar evidenciado en la tabla II. Finalmente, se uso diagrama de caja y bigote agrupado por día y posteriormente por hora para representar la información obtenida. La cantidad de registros procesado es la misma que en el primer análisis.

En la figura 6, se muestra los resultados obtenidos del gas So2 clasificados por días de la semana. En general, se observa que la ubicación B tiene una concentración media de material partículado, (ppb) uniforme durante los días laborales (lunes a viernes), mientras que en la ubicación A, se muestra valores medios similares durante los primeros 3 días de la semana, y disminuye ligeramente entre el jueves y viernes. Además, se observa que en el día domingo los valores disminuyen para ambas ubicaciones, esto se debe por que el domingo es un día de menor concentración laboral. En particular, y tomando como referencia solamente los días laborales, se obser-

va que para el día miércoles se encuentra los valores máximos obtenidos en 15.99 ppb y 16.05 ppb para la ubicación A, y B respectivamente.

Una vez, establecido el día de la semana de mayor concentración de ppb, se procedió a clasificar la información por franjas horarias. El fin de este propósito es identificar el rango hora de mayor concentración. Para este propósito se tomo como referencia el gas que se ha estado evaluando el SO_2. En la figura 7, se muestra el comportamiento del gas SO_2 por hora. En general, la figura 7 muestra que a las 10 de la mañana se produce la mayor concentración del gas SO_2 en dicha ubicación. Además, se evidencia un comportamiento no lineal según cae el atardecer siendo el punto más bajo de concentración ppb a las 18 hora de la tarde.

C. Análisis de gases según normativas

Una vez identificado el día de la semana y la hora de mayor concentración de ppb se procedió a comparar dichos valores con la normativa Ecuatoriana y las directrices mundiales de calidad del aire de la OMS. Para este propósito, primero se procedió a convertir los valores obtenidos en ($\mu g/m^3$) unidad de medida estándar según las normativas evaluadas. Seguidamente, se tomó como hora de referencia a las 10 de la mañana, para posteriormente comparar los valores obtenidos con la normativa Ecuatoriana. Respecto con la directrices mundial de calidad del aire se estableció como referencia el día de mayor concentración de ppb (miércoles), en este contexto, los datos obtenidos son de las 24 horas de dicho día. En ambos casos los valores calculados son los valores límites promedio respectivamente.

En las figuras 9 y 8, se observa los valores obtenidos en ($\mu g/m^3$) de los gases de CO, NO_2, y SO_2 en 1 hora y 24 horas. En las figuras se observa que los valores de mayor concentración de partículas se encuentra para el monoxido de carbono, mientras que el de menor concentración es para el SO_2.

En la tabla III muestra los valores límite promedio obtenido de los gases caso de estudio en 1 y 24 horas respectivamente. Estos valores permiten comparar las concentraciones de gases en cada ubicación con los límites establecidos por la normativa ecuatoriana y la OMS (ver tabla I). En general, en la tabla III se observa que en ambas ubicaciones A y B, los valores límite promedio de los gases en 24 horas son inferiores a los valores obtenidos en 1 hora. Por lo contrario, según la normativa de la OMS los valores en porcentaje de los gases estudiados son superiores respecto con la normativa ecuatoriana.

En particular, se observa que el gas NO_2 para las dos ubicaciones se encuentra entre un 48 % a 53 % de concentración aceptable según la normativa ecuatoriana. En cuanto al mismo gas pero, basado a las recomendaciones de la OMS los niveles encontrados dentro de las 24 horas superaron los límites normativos en un 271.64 % para la ubicación A y en un 202.04 % para la B.

Estos hallazgos resaltan la necesidad de abordar la calidad del aire en ambas ubicaciones para proteger la salud pública y aplicar medidas adecuadas de control y mitigación de la contaminación atmosférica.

Tabla III: Porcentaje de concentración máxima promedio según normativas en 1 hora y 24 horas por ubicación.

Lugar	Gas	Valor máximo promedio		Normativa	
		1 hora ($\mu g/m^3$)	24 horas ($\mu g/m^3$)	Decreto 97 (1hora)	AQG (24 horas)
A	CO	339.3	218.18	1.13 %	5.45 %
	SO_2	13.82	8.7	6.91 %	21.75 %
	NO_2	105.6	67.91	52.80 %	271.64 %
B	CO	312.33	162.3	1.04 %	4.06 %
	SO_2	13.9	7.46	6.95 %	18.65 %
	NO_2	97.21	50.51	48.61 %	202.04 %

V. Conclusiones

En este artículo se ha descrito una solución que permite monitorizar la calidad del aire usando sensores de bajos costo. La arquitectura propuesta combina tres componentes: i) una placa electrónica que interconectan sensores, ii) el componente de adquisición de datos formado por una Raspberry Pi 4, y iii) el servidor de la nube. Sobre el análisis de datos obtenidos a través de los sensores de gases semanal se establece como una buena práctica referencial medir el impacto de las lecturas obtenidas en zonas diferentes en períodos de tiempos consecutivos. Resultados obtenidos demuestran que existen altos niveles de concentración en las ubicaciones estudiadas, en particular se evidencio que el gas NO_2 excede los valores limites permitidos según, la regulación de la OMS. A corto plazo, se plantea analizar los otros gases y se busca implementar un calibrado inteligente.

Agradecimientos

El presente trabajo ha sido financiado según el proyecto de Ïnfraestructura tecnológica aplicando dispositivos IoT y Redes de sensores móviles"de la Universidad Laica Eloy Alfaro de Manabí, Manta, Ecuador.

Referencias

[1] "WHO global air quality guidelines," *Coastal And Estuarine Processes*, pp. 1–360, 2021.

[2] Committee of the Environmental and Occupational Health Assembly of the American Thoracic Society, "Health effects of outdoor air pollution," *Am J Respir Crit Care Med*, vol. 153, no. 1, pp. 3–50, 1996.

[3] Ye Kang, Lu Aye, Tuan Duc Ngo, and Jin Zhou, "Performance evaluation of low-cost air quality sensors: A review," *Science of The Total Environment*, vol. 818, pp. 151769, 2022.

[4] Mannam Veera Narayana, Devendra Jalihal, and S. M. Shiva Nagendra, "Establishing a sustainable low-cost air quality monitoring setup: A survey of the state-of-the-art," *Sensors*, vol. 22, no. 394, 2022.

[5] Prashant Kumar, Lidia Morawska, Claudio Martani, George Biskos, Marina Neophytou, Silvana Di Sabatino, Margaret Bell, Leslie Norford, and Rex Britter, "The rise of low-cost sensing for managing air pollution in cities," *Environment international*, vol. 75, pp. 199–205, 2015.

[6] M.I. Mead, O.A.M. Popoola, G.B. Stewart, P. Landshoff, M. Calleja, M. Hayes, J.J. Baldovi, M.W. McLeod, T.F. Hodgson, J. Dicks, A. Lewis, J. Cohen, R. Baron, J.R. Saffell, and R.L. Jones, "The use of electrochemical sensors for monitoring urban air quality in low-cost, high-density networks," *Atmospheric Environment*, vol. 70, pp. 186–203, 2013.

[7] Óscar Alvear, Willian Zamora, Carlos T. Calafate, Juan-Carlos Cano, and Pietro Manzoni, "Ecosensor: Monitoring environmental pollution using mobile sensors," in *2016 IEEE 17th International Symposium on A World*

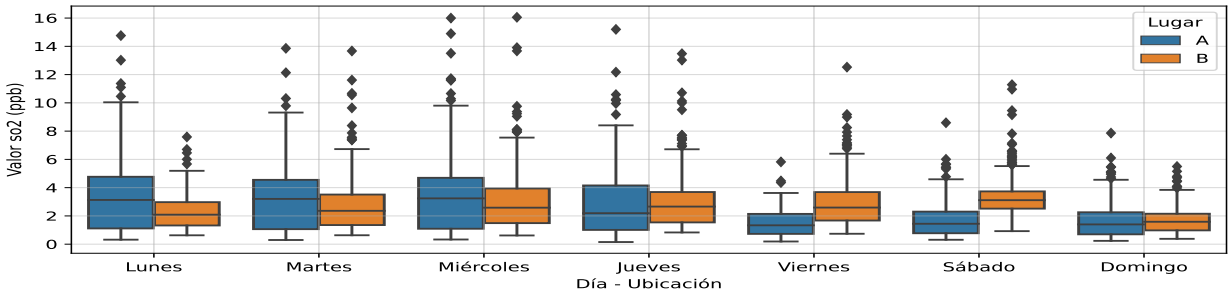


Fig. 6: Análisis del gas SO2 por día en ambas ubicaciones.

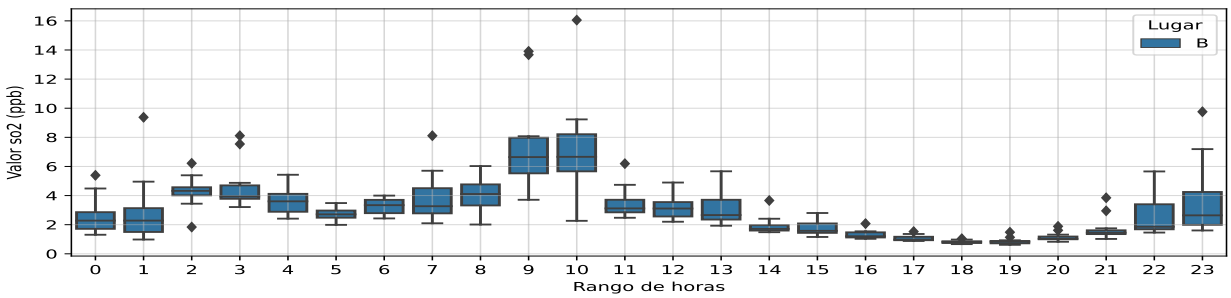
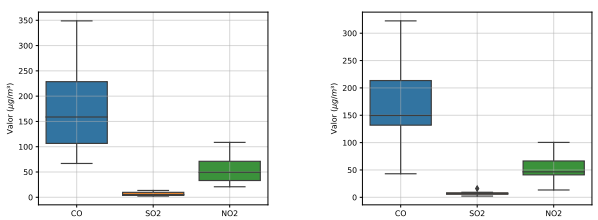
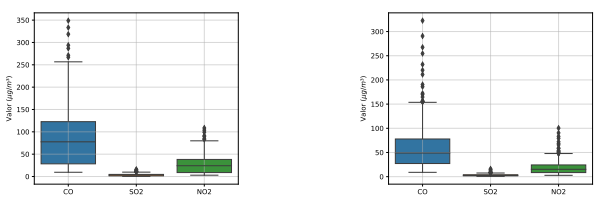


Fig. 7: Análisis del gas SO2 por hora en la ubicación B.



(a) Ubicación A (b) Ubicación B

Fig. 8: Análisis CO,NO_2, y SO_2 en una hora.



(a) Ubicación A (b) Ubicación B

Fig. 9: Análisis CO, NO_2, y SO_2 en 24 hora.

of Wireless, Mobile and Multimedia Networks (WoWMoM), 2016, pp. 1–6.

[8] Jose Bazurto, Willian Zamora, Johnny Larrea, Dolores Muñoz, and Dahiana Alvia, "System for monitoring air quality in urban environments applyng low-cost solutions," in *2020 15th Iberian conference on information systems and technologies (CISTI)*. IEEE, 2020, pp. 1–6.

[9] Anamika Sharma, Brijesh Mishra, Ronak Sutaria, and Rajesh Zele, "Design and development of low-cost wireless sensor device for air quality networks," in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, 2019, pp. 2345–2350.

[10] Steven J. Johnston, Philip J. Basford, Florentin M. J. Bulot, Mihaela Apetroaie-Cristea, Natasha H. C. Easton, Charlie Davenport, Gavin L. Foster, Matthew Loxham, Andrew K. R. Morris, and Simon J. Cox, "City scale particulate matter monitoring using lorawan based air quality iot devices," *Sensors*, vol. 19, no. 1, 2019.

[11] Lorenzo Parri, Marco Tani, David Baldo, Stefano Parrino, Elia Landi, Marco Mugnaini, and Ada Fort, "A distributed iot air quality measurement system for high-risk workplace safety enhancement," *Sensors*, vol. 23, no. 11, 2023.

[12] Nima Afshar-Mohajer, Christopher Zuidema, Sinan Sousan, Laura Hallett, Marcus Tatum, Ana M Rule, Geb Thomas, Thomas M Peters, and Kirsten Koehler, "Evaluation of low-cost electro-chemical sensors for environmental monitoring of ozone, nitrogen dioxide, and carbon monoxide," *Journal of occupational and environmental hygiene*, vol. 15, no. 2, pp. 87–98, 2018.

[13] E. S. Cross, L. R. Williams, D. K. Lewis, G. R. Magoon, T. B. Onasch, M. L. Kaminsky, D. R. Worsnop, and J. T. Jayne, "Use of electrochemical sensors for measurement of air pollution: correcting interference response and validating measurements," *Atmospheric Measurement Techniques*, vol. 10, no. 9, pp. 3575–3588, 2017.

[14] Alphasense Ltd, "Co-b4," `https://www.alphasense.com/wp-content/uploads/2019/09/CO-B4.pdf`, (Accessed on 01/01/2023).

[15] Alphasense Ltd, "No2-b43f," `https://www.alphasense.com/wp-content/uploads/2019/09/NO2-B43F.pdf`, 01 2022, (Accessed on 01/01/2023).

[16] Alphasense Ltd, "So2-b4," `https://www.alphasense.com/wp-content/uploads/2019/09/SO2-B4.pdf`, 01 2022, (Accessed on 01/01/2023).

[17] Alphasense Ltd, "Ox-b431," `https://www.alphasense.com/wp-content/uploads/2019/09/OX-B431.pdf`, 01 2022, (Accessed on 01/01/2023).

[18] Alphasense Ltd, "No-b4," `https://www.alphasense.com/wp-content/uploads/2019/09/NO-B4.pdf`, 01 2022, (Accessed on 01/01/2023).

[19] Sensirion, "Sps30 - pm2.5 sensor for hvac and air quality applications sps30," `https://sensirion.com/products/catalog/SPS30`, (Accessed on 01/01/2023).

[20] Terrie K. and P.E Boguski, "Understanding Units of Measurement," *October*, , no. October, pp. 1–3, 2008.

[21] Breeze Technologies, "Air pollution - how to convert between mg/m3, µg/m3 and ppm, ppb - breeze technologies," `https://www.breeze-technologies.de/blog/air-pollution-how-to-convert-between-mgm3-%C2%B5gm3-ppm-ppb/`, (Accessed on 01/01/2023).

[22] MAE, "REFORMA TEXTO UNIFICADO LEGISLACION SECUNDARIA, MEDIO AMBIENTE, LIBRO VI, Decreto Ejecutivo 3516," *REFORMA TEXTO UNIFICADO LEGISLACION SECUNDARIA, MEDIO AMBIENTE, LIBRO VI, Decreto Ejecutivo 3516*, vol. 0, pp. 18–34, 2015.

Ecovana: una arquitectura para detectar CO_2 en interiores

Anayeli Cedeño, Willian Zamora[2], Johnny Larrea, Robert Moreira, Patricia Quiroz-Palma
1 2

Resumen— **En la actualidad la contaminación del aire se ha transformado en una amenaza para la salud en el mundo, sobre todo en los lugares más frecuentados y con poca ventilación, la monitorización se ha convertido en un requisito esencial para las ciudades. Para resolver el problema, hemos desarrollado Ecovana, una solución para medir la calidad del aire mediante el uso de dispositivos de bajo costo y el desarrollo de una aplicación móvil. En particular, se propone Ecovana una solución basada en el Internet de las Cosas para medir los niveles de CO_2 en espacios cerrados en la Ciudad de Manta. Ecovana, recoge los niveles de CO_2 del aire y mediante su aplicación móvil lo muestra a los usuarios en tiempo real. Además, para validar la solución Ecovana, muestra un análisis comparativo con un dispositivo comercial. Resultados experimentales demuestran que Ecovana presenta un porcentaje de error del 1.36 % respecto con un equipo comercial en ambientes cerrados.**

Palabras clave— **IoT, contaminación, aire, Co2.**

I. Introducción

EN la actualidad, la calidad del aire en la mayoría de las ciudades del mundo que monitorean la contaminación no alcanza los niveles de seguridad, por lo que se establece que pone en riesgo a las personas antes un peligro de enfermedades respiratorias y otros problemas de salud. En particular, Ecuador y en especial, la ciudad de Manta no está libre de estos problemas.

Manta, es una de las ciudades con una población que bordea los 500 mil habitantes y ubicada en la provincia de Manabí. Es llamada "La Puerta del Pacífico" por ser uno de los principales puertos para la economía ecuatoriana. A pesar de la importancia que tiene Manta a nivel nacional presenta problemas de contaminación. Uno de ellos es la contaminación del aire producida principalmente por los siguientes factores: la laguna de oxidación, y las emisiones de gases ocasionadas de manera especial por los vehículos y fabricas.

En general, se conoce que la contaminación del aire puede darse por muchos tipos de gases emitidos al aire como puede ser: monóxido de carbono, dióxido de carbono (CO2), ozono, entre otros. Además, la Organización mundial de la salud señala que a alta exposición de estos gases puede ser perjudicial para la salud humana [1].

En concreto, en este articulo se evalúa la presencia de los índices de contaminación del aire del Co2, en diferentes ambientes (cerrados y abiertos). Las altas concentraciones de CO2 causan efectos adversos para la salud especialmente en espacios cerrados, tales como aulas, restaurantes, oficinas de trabajo, etc. Todo esto como consecuencia de un sistema de ventilación deficiente que no elimina el CO2 generado por los ocupantes a través de la respiración. Además, que existen espacios con alto grado de contaminación, las personas se expone a una alta probabilidad de contagios de enfermedades respiratorias como lo es el Covid-19.

A esta causa se le suma los costos elevados que se presentan en el mercado para adquirir un dispositivo de medición del CO2. Ante esto, hoy el Internet de las Cosas (IoT) [2, 3] es una tecnología emergente, que permite integrar dispositivos dentro de la infraestructura de internet existente y a bajos coste. En este contexto, este artículo propone Ecovana, una arquitectura basada en el internet de las cosas para medir los niveles de concentración del Co2 en espacios abiertos y cerrados. En particular se compara nuestra solución con un dispositivo comercial.

Este documento está organizado como sigue: En el la siguiente sección II se presentan los trabajos relacionados. La sección III, se describe la solución propuesta ECOVANA. En la sección IV se muestran los resultados obtenidos de la evaluación de la propuesta en comparación con un equipo comercial. Por último, la sección V presenta las conclusiones y trabajos futuros.

II. Estado del arte

En la literatura se encuentran varias propuestas que permiten el monitorio de la calidad del aire. Algunas de estas, proponen su arquitecturas de monitoreo, pero, diferenciándose en los gases y en la tecnología usada. A continuación algunos temas.

En [4] los autores proponen un sistema inalámbrico de monitoreo para la calidad del aire en tiempo real. En particular, mide las concentraciones de monóxido de carbono (CO) y dióxido de carbono (CO_2). La solución propuesta utiliza un placa basada en Arduino UNO [5], y lo sensores de gases (CO_2) Mg811 [6] y (CO) MQ-7 [7]. En [8], además de los gases anteriores en [4] agrega el SO2 y NO2 en la estación de medición propuesta. En particular, los autores utilizan un Raspberry Pi 3 [9] y su arquitectura propuesta se basa en el Internet de las Cosas (IoT) [9].

En [10] propone EcoSensor una arquitectura para el monitoreo ambiental del aire usando sensores móviles. En particular, esta propuesta analiza la lectura del ozono en tiempo real dentro de la ciudad de Valencia, España. Además, compara su propues-

[1]FCVT, Universidad Laica Eloy Alfaro de Manabí, Email: anayeli4373@gmail.com {willian.zamora, johnny.larrea, robert.moreira, patricia.quiroz} @uleam.edu.ec
[2]Department of Computer Engineering, Universitat Politècnica de València, Email: wilzame@posgrado.upv.es

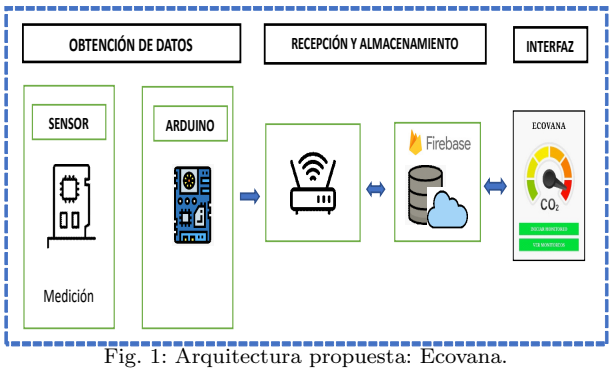


Fig. 1: Arquitectura propuesta: Ecovana.

ta con una central de monitoreo ambiental de altos costos y valida su propuesta con técnicas de interpolación de Kriggin analizando una estrategia óptima de lectura. Más adelante los autores [11] propusieron una arquitectura genérica para analizar la calidad del aire usando sensores de gases de bajos costos.

En [12] propone una solución de monitoreo basado en la arquitectura de IoT de Gartner, los autores miden los niveles de concentración de los gases contaminantes como Co2 y CH4. La solución utiliza placa Arduino y protocolo MQTT [13] para la transmisión de los datos. Mas adelante, en [14] proponen un prototipo de bajo costo para el monitoreo de la calidad del aire en ambientes interiores. La propuesta utiliza un microcontrolador ESP8266 [15] junto a sensores de gases (MQ) como el monóxido de Carbono y dióxido de carbono entre otros.

En [16] los autores proponen un solución para determinar la concentraciones de sulfuro de hidrógeno(H2S) en un complejo industrial de logística y distribución. Específicamente, detectan la emisión de los gases que emanan las baterías eléctricas de los montacargas. Los autores han integrado un sensor MQ136 [17] y una tarjeta de desarrollo ESP32 [18] y tecnología LoRaWAN [19].

Como se puede apreciar existe abundante soluciones relacionadas para obtener niveles de contaminantes del aire. A diferencia de las soluciones mostradas en estos artículos, nuestra propuesta compara ECOVANA con un dispositivo comercial.

III. Arquitectura Ecovana

Una vez analizado el estado de arte, a continuación se describe Ecovana. Se denominó Ecovana por las abreviatura de .ecológicoz "VaNa"por los autores del estudio. Ecovana se caracteriza por ser un sistema para el monitoreo de los niveles de dióxido de carbono en el ambiente. En la figura 1 se muestra la arquitectura propuesta. En general, la arquitectura tiene tres componentes: el primero se denomina obtención de los datos y se encarga de la captura del dióxido de carbono. El segundo corresponde a la recepción y almacenamiento de los datos. Y finalmente el tercer componente se lo ha denominado Ïnterfazz es el encargado de mostrar la los datos en tiempo real mediante una aplicación móvil. A continuación se describen más detalles de cada uno de los componentes.

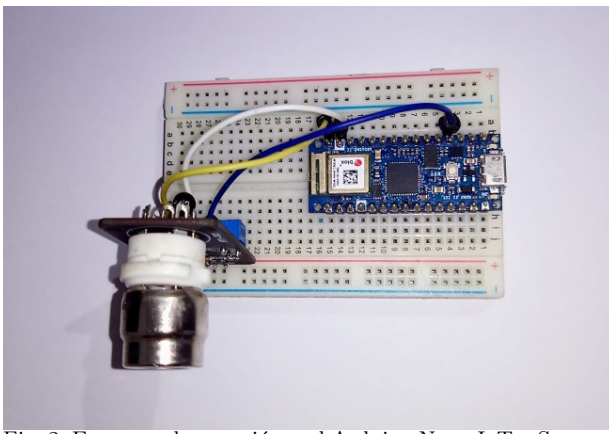


Fig. 2: Esquema de conexión real Arduino Nano IoT y Sensor.

A. Obtención de los datos

El componente obtención de los datos esta constituido por 3 partes: el primero corresponde al microcontrolador Arduino Nano 33 IoT [20], el segundo es el Sensor de dióxido de carbono (CO_2) modelo Mg811 [6] y el tercero la placa de prueba. El funcionamiento de este componente consiste en obtener la lectura de los niveles de CO_2 en el ambiente y luego mediante el uso de la red inalámbrica WiFi sea enviado para su respectivo almacenamiento. Cabe indicar que el microcontrolador empieza a enviar niveles de CO_2 siempre que el componente de la interfaz active la lectura del sensor. En la figura 2, se muestra el esquema de conexión de la solución propuesta mediante el uso de una placa de prueba.

B. Recepción y almacenamiento

El componente de recepción y almacenamiento actúa como intermediario entre la interfaz (aplicación móvil) y el dispositivo sensor. En concreto, recibe los datos del microcontrolador conectado a la red WiFi local para ser directamente almacenado en la base de datos Firebase.

C. Interfaz Ecovana

En general la App Ecovana, ha sido desarrollada en Android Studio y permite mostrar los niveles de CO_2 en tiempo real. La características principales de la solución son la siguiente:

- Permite autentificar.
- Permite registrar automáticamente la localización geográfica del lugar que se desea monitorizar. Además, se puede describir la ubicación y circunstancias de dicha lectura.
- Permite iniciar y detener la captura de datos en cualquier momento.
- Muestra los datos en tiempo real según la frecuencia de tiempo programada.
- Una vez detenida la captura muestra un resumen de dicha lectura.

En la figura 3 se muestra la interfaz de la solución propuesta Ecovana.

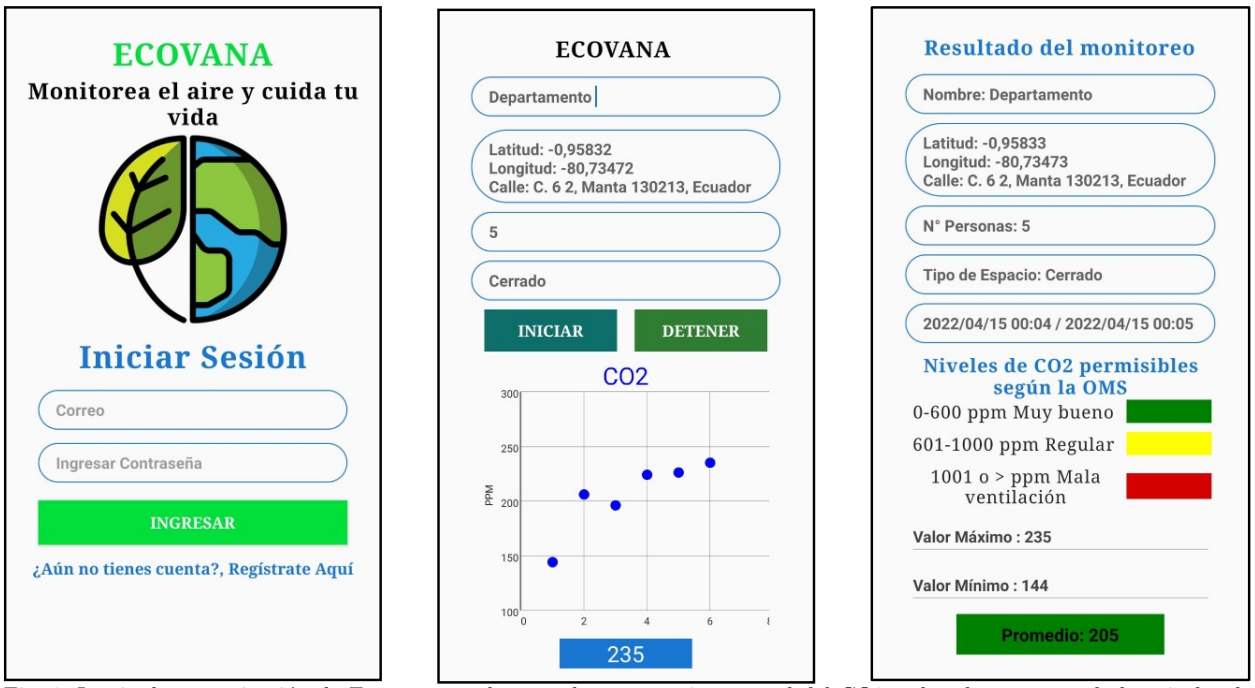


Fig. 3: Izquierda autenticación de Ecovana, en el centro lectura en tiempo real del CO2 y derecha resumen de los niveles de CO2.

IV. Validación de Ecovana

Una vez analizada la arquitectura propuesta se procedió a realizar la evaluación de la solución Ecovana. Para dicho propósito la metodología a utilizar es el análisis comparativo. Específicamente, se analiza el error porcentual entre la lectura obtenida por Ecovana y un dispositivo comercial. Para nuestro análisis se dispone de un Sensor BLATN BR-Smart [21], que es un dispositivo que permite obtener datos en tiempo real de varios sensores incluyendo el CO_2. Este equipo permite obtener datos en interiores y exteriores. Además, permite almacenar los datos obtenidos mediante una tarjeta de memoria interna.

Teniendo la estrategia de evaluación se plantearon 3 escenario de pruebas. El primer escenario corresponde a una evaluación preliminar en un escenario de interior no controlado (las personas entran y salen frecuentemente). La segunda evaluación se la realizó en un escenario de clases controlado. La última evaluación se la hizo en un ambiente abierto, específicamente, en una sala de estudiantes de la Facultad de ciencias Informáticas. En la figura 4, se muestra un ejemplo de comparación entre la solución propuesta y la solución comercial. Para cada evaluación la solución propuesta Ecovana almacena el valor del promedio de 6 muestras por minuto. A continuación, se darán más detalles de cada evaluación.

Fig. 4: Escenario de comparación

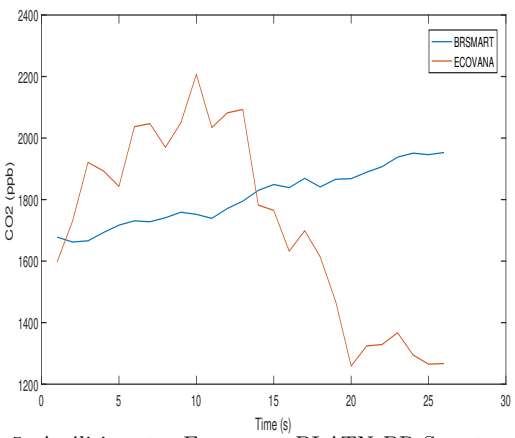


Fig. 5: Análisis entre Ecovana y BLATN BR-Smart en interiores - no controlado.

A. Comparación en interiores no controlado

En este primer escenario se lo realizo dentro de las instalaciones de la asociación de estudiantes de la Facultad de Ciencias Informáticas. Además, el escenario de la evaluación no fue controlado ya que dentro del proceso de toma de datos se permitió el libre acceso de estudiantes. También, cabe indicar que esta primera evaluación se la hizo sin que el equipo BLATN BR-Smart no se haya calibrado. El tiempo de duración de la muestra fue de una hora aproximadamente. Respecto con los datos capturados, se eliminaron muestras de precalentamiento que se debía esperar hasta que ambos sensores se estabilicen por tal razón se tomaron 26 registros en total.

En la figura 5, se observa los resultados obteni-

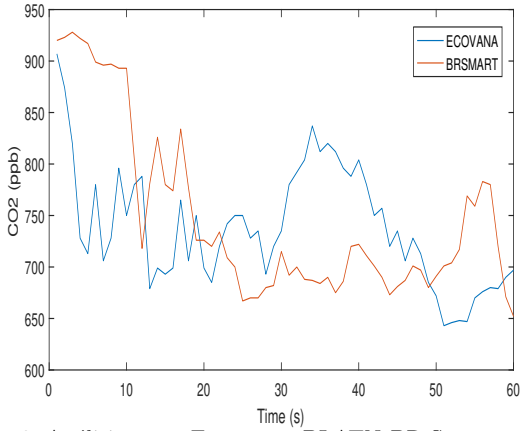


Fig. 6: Análisis entre Ecovana y BLATN BR-Smart en interiores controlado

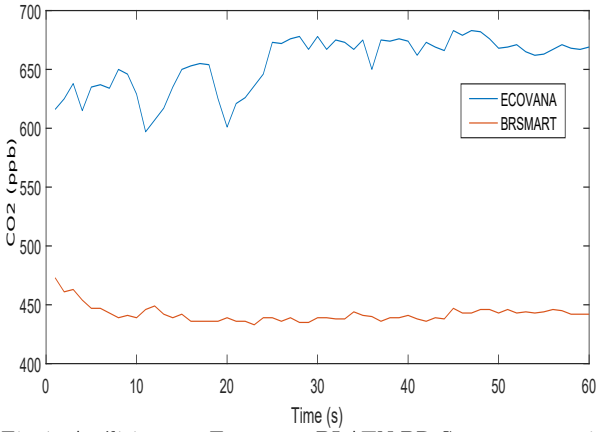


Fig. 7: Análisis entre Ecovana y BLATN BR-Smart en espacio abierto.

Tabla I: Análisis estadísticos de evaluaciones

	Exteriores		Interior - No controlado		Interior - Laboratorio	
	Ecovana	Br-Smart	Ecovana	Br-Smart	Ecovana	Br-Smart
Min (ppb)	597	433	1259	1662	643	652
Max (ppb)	683	473	2207	1953	907	928
Avg (ppb)	531.14	442.10	1714.27	1806.85	738.08	748.23
% Error	20.14		5.12		1.36	

dos. En general, se puede muestra que en los espacios cerrados con varias personas hay una cantidad de CO_2 alta, lo que nos indica que el estado de ventilación del aire interior no presenta una buena circulación. Además, se observa que el equipo comercial BR-Smart [21] se encuentra mucho más estable, por lo contrario a la propuesta Ecovana esta presenta distorsiones en sus lecturas. Es notable que Ecovana se elevan y luego empieza a descender en el segundo 14.

Estos resultados se debieron ya que en primera instancia el equipo patrón de lectura y el nuestro no se había calibrado.

B. Comparación en interiores controlado

Una vez ajustada la solución Ecovana y calibrado el equipo BR-Smart [21] según los requerimientos de fabrica se procedió al segundo experimento en otro escenario. En este segundo escenario se lo realizo dentro de un laboratorio de clases de estudiantes de la Facultad de Ciencias Informáticas, dicho espacio tiene un aire acondicionado por lo que su climatización está controlada. Además, el escenario no se permitió que los estudiantes se levanten y salgan del laboratorio de clases. El tiempo de duración del experimento fue de 60 minutos aproximadamente y de igual manera se tomo el promedio por minuto. En esta ocasión se tomaron todos los registros.

En la figura 6, se muestra la representación de los datos obtenidos. Es notable que estos valores se encuentran mucho más próximo respecto con la evaluación anterior. En ambos instrumentos Ecovana y BR-Smart [21] las soluciones se encuentran asimétricas. En particular, Ecovana muestra una línea ligeramente inferior a la solución patrón (BR-Smart). Además, se puede observar que en los espacios cerrados hay una cantidad de CO_2 regular, lo que nos indica que el estado de circulación del aire interior para este ambiente se debería controlar.

C. Comparación en espacio abierto

Finalmente, se procedió a evaluar nuestra propuesta en un escenario al aire libre. Esto se lo realizo en la sala de estudiantes de la Facultad de Ciencias Informáticas. De igual manera que el anterior experimento, se hizo una evaluación de una hora. En esta ocasión se tomo los datos entre las 11:30 AM a 12:30 PM. Hay que destacar que este escenario no se encontraba controlado por lo que existía libre circulación de estudiantes y se encuentra con una circulación de aire natural.

En la figura 7, se muestra un comportamiento distinto a las otras dos evaluaciones, en este caso se observa que los valores en un escenario al aire libre de la propuesta Ecovana son superiores al instrumento guía (BR-Smart). Además, se observa que Ecovana presenta ciertos picos de lectura cambiante, esto se debe a la sensibilidad de los primeros 25 minutos y a cierta cantidad de estudiantes que circularon en ese tiempo. En general, se puede decir que los niveles de concentración de CO_2 se encuentran aceptable en espacios abiertos.

D. Análisis del error porcentual

Una vez evaluado los 3 escenario indicados previamente, a continuación se describe el error porcentual de la propuesta Ecovana respecto con el equipo comercial. En este sentido para cada evaluación se tomo el valor medio de cada lectura y posteriormente se cálculo el error porcentual. En la tabla I se observa el porcentaje de error de cada uno de los escenarios evaluados así como los valores promedio, máximo y mínimos obtenidos. En general, es notable que la solución propuesta tiene un margen de error del 1.36 % respecto al instrumento referencial. Por otro lado, se observa que ambiente interiores no controlados el error suele subir al 5.12 %. Finalmente se observa que el error es de 20.14 % en lugares abiertos.

V. Conclusiones y trabajos futuros

En este artículo se propuso Ecovana, una solución desarrollada para el monitoreo de los niveles del dióxido del carbono en el aire. La solución propuesta combina la utilización de dispositivos de bajo costo,

smartphones y servicios Cloud. Ecovana, mediante su aplicación móvil permite mostrar los niveles del dióxido de carbono en tiempo real. Además, permite visualizar históricos de lecturas y resúmenes en un periodo determinado.

Para validar la propuesta se hizo un análisis comparativo con un equipo de carácter comercial. En este sentido, se hicieron varios análisis para determinar la diferencia de error porcentual existente, entre nuestra propuesta y el equipo comercial. En particular, se tomaron pruebas en espacios interiores controlados y no controlados, así como pruebas en espacios de libre circulación. Resultados experimentales demuestran que en espacios interiores controlados los valores obtenidos en Ecovana son muy aceptables teniendo un rango de error menor al 1.36 % respecto con el sensor comercial.

Como trabajo futuro se plantea mejorar el proceso de calibración usando alguna estrategia de regresión. Además, se proyecta que esta solución se unifique a una estación móvil de monitoreo que incluya más de un sensor.

Agradecimientos

El presente trabajo ha sido financiado según el proyecto de Ïnfraestructura tecnológica aplicando dispositivos IoT y Redes de sensores móviles"de la Universidad Laica Eloy Alfaro de Manabí, Manta, Ecuador.

Referencias

[1] "New who global air quality guidelines aim to save millions of lives from air pollution," https://www.who.int/news/item/22-09-2021-new-who-global-air-quality-guidelines-aim-to-save-millions-of-lives-from-air-pollution, 01 2021, (Accessed on 10/01/2021).

[2] Mike O. Ojo, Stefano Giordano, Gregorio Procissi, and Ilias N. Seitanidis, "A review of low-end, middle-end, and high-end iot devices," *IEEE Access*, vol. 6, pp. 70528–70554, 2018.

[3] Mohsen Marjani, Fariza Nasaruddin, Abdullah Gani, Ahmad Karim, Ibrahim Abaker Targio Hashem, Aisha Siddiqa, and Ibrar Yaqoob, "Big iot data analytics: Architecture, opportunities, and open research challenges," *IEEE Access*, vol. 5, pp. 5247–5261, 2017.

[4] Walter Fuertes, Diego Carrera, César Villacís, Theofilos Toulkeridis, Fernando Galárraga, Edgar Torres, and Hernán Aules, "Distributed system as internet of things for a new low-cost, air pollution wireless monitoring on real time," in *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2015, pp. 58–67.

[5] ARDUINO, "Arduino uno wifi rev2 — arduino official store," https://store.arduino.cc/products/arduino-uno-wifi-rev2, (Accessed on 06/01/2022).

[6] Cytron Marketplace, "Mg811 carbon dioxide co2 sensor module," https://www.cytron.io/c-sensor/c-gas-sensor/p-mg811-carbon-dioxide-co2-sensor-module, (Accessed on 01/01/2022).

[7] Mechatronics, "Sensor mq-7 gas co monóxido de carbono," https://naylampmechatronics.com/sensores-gas/74-sensor-mq-7-gas-monoxido-de-carbono-co.html, (Accessed on 01/24/2022).

[8] Gagan Parmar, Sagar Lakhani, and Manju K. Chattopadhyay, "An iot based low cost air pollution monitoring system," in *2017 International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE)*, 2017, pp. 524–528.

[9] RaspberryPi, "Buy a raspberry pi 3 model b – raspberry pi," https://www.raspberrypi.com/products/raspberry-pi-3-model-b/, (Accessed on 05/17/2022).

[10] Óscar Alvear, Willian Zamora, Carlos T. Calafate, Juan-Carlos Cano, and Pietro Manzoni, "Ecosensor: Monitoring environmental pollution using mobile sensors," in *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2016, pp. 1–6.

[11] Jose Bazurto, Willian Zamora, Johnny Larrea, Dolores Muñoz, and Dahiana Alvia, "System for monitoring air quality in urban environments applyng low-cost solutions," in *2020 15th Iberian conference on information systems and technologies (CISTI)*. IEEE, 2020, pp. 1–6.

[12] Felipe Diaz Sanchez Cesar Villate Barrera, "Sistema para la medición de gases de efecto invernadero mediante los principios del internet de las cosas, alineado al cumplimiento de los compromisos de colombia ante las naciones unidas," https://1library.co/document/6zk0dp1y-sistema-medicion-invernadero-mediante-principios-cumplimiento-compromisos-colombia.html, (Accessed on 01/15/2022).

[13] MQTT, "Mqtt - the standard for iot messaging," https://mqtt.org/, (Accessed on 01/16/2022).

[14] Nicolás Castiblanco Avendaño, Cristian Cañón Alfonso, "Prototipo de bajo costo para monitoreo de calidad del aire en ambientes interiores," http://repository.unipiloto.edu.co/handle/20.500.12277/4880, (Accessed on 04/27/2022).

[15] ESPRESSIF, "Esp32 wi-fi & bluetooth mcu i espressif systems," https://www.espressif.com/en/products/socs/esp32, (Accessed on 01/27/2022).

[16] Henrry Francisco Guerrero Espinoza, "Repositorio institucional de la universidad politécnica salesiana: Plataforma basada en internet de las cosas (iot) para la medición y monitoreo remoto del nivel de gas sulfuro de hidrógeno generado por baterías de montacargas eléctricos en entornos logísticos," https://dspace.ups.edu.ec/handle/123456789/20311, (Accessed on 02/25/2022).

[17] sensorica, "Mq-136e.pdf," http://www.sensorica.ru/pdf/MQ-136.pdf, (Accessed on 02/11/2022).

[18] ESPRESSIF, "Esp8266 wi-fi mcu i espressif systems," https://www.espressif.com/en/products/socs/esp8266, (Accessed on 01/27/2022).

[19] LoraWan, "What is lorawan® specification - lora alliance®," https://lora-alliance.org/about-lorawan/, (Accessed on 06/20/2022).

[20] ARDUINO, "Arduino nano 33 iot — arduino online shop," https://store-usa.arduino.cc/products/arduino-nano-33-iot, (Accessed on 06/01/2022).

[21] Blatn, "Indoor air quality monitor br-smart-128s co2 pm2.5 tvoc hcho," https://blatn.com/es/products/indoor-air-quality-monitor-br-smart-128s-co2-pm2-5-tvoc-hcho, (Accessed on 02/01/2022).

Evaluación bajo radiación de técnicas de mitigación de fallos basadas en el cálculo aproximado

Antonio Martínez-Álvarez[1], Darío González-Montesoro[1], Alejandro Serrano-Cases[2], Alexander Aponte-Moreno[3], Felipe Restrepo-Calle[3], Yolanda Morilla, [4] Pedro Martín-Holgado[4], Sergio Cuenca-Asensi[1]

Resumen— **Este artículo evalúa una técnica de mitigación de fallos tipo *soft errors* inducidos por radiación en microprocesadores COTS que se fundamenta en el uso de computación aproximada. Los resultados experimentales muestran que puede detectar y corregir fallos de evento simple (SEUs) manteniendo la precisión bajo control y sin comprometer el rendimiento del sistema.**

Palabras clave— **Computación aproximada, ARM, irradiación con protones, tolerancia a fallos.**

I. Introducción

LOS dispositivos de hardware comerciales disponibles en el mercado (COTS, emphComercial Off-The-Shelf) siguen ganando popularidad allí donde la tolerancia a fallos es una necesidad. Dada la naturaleza de estos sistemas, las técnicas de *software* definen la fuente más evidente de mejora en cuanto a fiabilidad. En este contexto, las técnicas tradicionales de TMR (*Triple Modular Redundancy*) son el método más común para proporcionar protección. Sin embargo, las técnicas de tolerancia a fallos basadas en software, sean del tipo que sean, siempre involucran un coste. En efecto, implican una serie de sobrecargas no deseadas, en el tiempo de ejecución y el uso de memoria, que deben limitarse y mantenerse bajo control, especialmente cuando se aplican a sistemas embebidos.

En aquellos casos en los que los requisitos de precisión se puedan relajar, es posible utilizar técnicas de computación aproximada, en adelante referidas como CA, para reducir al mínimo los costes de tiempo. De hecho, recientemente, las técnicas de CA se han empleado para diseñar sistemas tolerantes a fallos con una reducida sobrecarga de tiempo. [1, 2]. El paradigma CA mejora el rendimiento (y en consecuencia, también la sobrecarga de tiempo) y la eficiencia energética de un software, a costa de introducir cierta inexactitud en la salida [3]. Sin embargo, es esencial garantizar que la salida aproximada siga siendo compatible con los resultados precisos del software, manteniendo al mismo tiempo la precisión bajo control.

En este trabajo, proponemos y evaluamos bajo radiación de protones, una versión modificada de la técnica de tolerancia a fallos basada en CA introducida en [4]. Nuestra técnica combina la redundancia y la CA para minimizar en mayor medida el sobrecoste temporal y está diseñada para proteger microprocesadores comerciales contra *soft errors* inducidos por radiación ionizante. Se presenta un amplio caso de estudio en el que el cálculo crítico, definido por las conocidas transformaciones empleadas en robótica `inverse2kj` y `forward2kj`, se endurece utilizando una combinación de redundancia temporal y diferentes niveles de CA. El dispositivo sometido a test, también llamado DUT (Device Under Test) es un SoC (*System on a Chip*) Xilinx Zynq-7010 que integra un microprocesador ARM Cortex-A9 de 28 nm de proceso fotolitográfico. El sistema se irradió con protones de $15,3MeV$ de energía. Los resultados muestran que la técnica es perfectamente adecuada para detectar y corregir fallos que generan *SEUs*, a la vez que se mantiene bajo control la precisión sin incurrir en sobrecostes de rendimiento.

II. Mitigación de fallos con cálculo aproximado

La técnica de mitigación de fallos que emplea CA propuesta en [4] consiste en aproximar el resultado de un programa dado, aproximando sus cálculos antes de aplicar las estrategias de endurecimiento TMR, para compensar los sobrecostes asociados a la aplicación de esta técnica de tolerancia a fallos. Suponemos en este artículo que los algoritmos que se quieren proteger contienen bucles en su implementación. Estos bucles, en los que se realizan cálculos intermedios que se emplearán en la salida final del programa, serán el objetivo de la aplicación de nuestra técnica de mitigación de fallos. Para minimizar los sobrecostes temporales en que se incurre, los autores proponen una nueva técnica de CA llamada *Iteraciones simplificadas*, que es una variación derivada de la técnica de *perforación de bucles*, consistente en omi-

[1]Antonio Martínez-Álvarez(✉), Darío González-Montesoro y Sergio Cuenca-Asensi están en el Departamento de Tecnología Informática, Universidad de Alicante, Carretera San Vicente del Raspeig s/n, 03690 Alicante, España (e-mail: {amartinez, dgonzalez, sergio}@dtic.ua.es; Autor de contacto: amartinez@dtic.ua.es).

[2]Alejandro Serrano-Cases trabaja en el Barcelona Supercomputing Center (BSC) (e-mail: alejandro.serrano@bsc.es)

[3]Alexander Aponte-Moreno y Felipe Restrepo-Calle trabajan en el Departamento de Ingeniería Industrial y de Sistemas de la Universidad Nacional de Colombia, Bogotá D.C., Colombia. (correo electrónico: jaapontem@unal.edu.co, ferestrepoca@unal.edu.co).

[4]Yolanda Morilla y Pedro Martín-Holgado trabajan en el Centro Nacional de Aceleradores (CNA), CSIC, JA, Universidad de Sevilla, E-41092 Sevilla, España, ESPAÑA (e-mail: ymorilla@us.es, pmartinholgado@us.es).

tir la ejecución de algunas iteraciones de los bucles. La técnica *iteraciones simplificadas* omite también algunas iteraciones de forma controlada y parametrizada y realiza cálculos aproximados en esas dichas iteraciones. De esta forma, esta nueva técnica de CA se adapta bien a casos en los que la *perforacion de bucles* presentaría elevados errores de aproximación, como programas que operan sobre un gran conjunto de datos de entrada, como en este trabajo.

El funcionamiento de la técnica de *iteraciones simplificadas* se ilustra en la Figura 1. La sección superior muestra una estructura de código típico con bucle *for*, mientras que en la sección inferior se ha aplicado la técnica de *iteraciones simplificadas* en el mismo bucle. La función `select_iteration()` determina qué iteraciones harán los cálculos precisos utilizando `original_calculations()` como en el código original, mientras que la función `approximated_calculations()` se invoca para realizar los cálculos aproximados.

```
1  //Buble For
2  for (i = 0; i < max_value; i++) {
3      original_calculations();
4  }
5  ...
6  //Buble For aproximado
7  for (i = 0; i < max_value; i++) {
8      if (select_iteration(i))
9          original_calculations();
10     else
11         approximated_calculations();
12 }
```

Fig. 1: Fragmento de código en C que ilustra la aplicación de la técnica de *iteraciones simplificadas*

En este artículo, proponemos utilizar la técnica de mitigación de fallos de Duplicación con Comparación (*DWCF* [5]) junto con la técnica CA de iteraciones simplificadas, en lugar de con la tradicional *TMR*. *DWCF* es una versión avanzada de la *DWC* convencional que implementa la capacidad de corregir fallos. En *DWCF* el programa se ejecuta dos veces y se comparan los resultados, en caso de que se haya generado un error, se activa una tercera ejecución del programa y, por último, se realiza una segunda comparación para corregir cualquier fallo utilizando un votador mayoritario con los resultados de las tres ejecuciones del programa. La figura 2 muestra el funcionamiento de DWCF en su versión original y aproximada.

El uso de módulos de CA reduce notablemente el tiempo de ejecución. Además, el tiempo se reduce aún más al solo realizar la tercera ejecución en caso de fallo. La reducción del tiempo de ejecución depende del grado de aproximación alcanzado por la técnica de CA utilizada. En este caso, mediante las iteraciones simplificadas, el grado de aproximación está relacionado con el número de iteraciones del bucle en el que se simplifican los cálculos, y cuánto se pueden aproximar las operaciones en la iteración. Por lo tanto, aumentando el número de iteraciones en las que los cálculos precisos se sustituyen por cálculos aproximados, es posible obtener distintos niveles de aproximación.

III. Experimentación

El dispositivo sometido a test (*DUT*) seleccionado para los experimentos bajo radiación fue la placa *Zynq*, equipada con un sistema en chip (*SoC*) [6] de 28 nm CMOS Xilinx ZYNQ XC7Z010. Este *SoC* integra un microprocesador ARM Cortex A9 de doble núcleo a 667 MHz y 32 bits con un pipeline de instrucciones de 13 etapas, predictor de saltos y soporte para dos niveles de caché. También dispone de 256 KiB de memoria integrada en chip (OCM) y 512 MiB de memoria DDR externa. En este trabajo sólo se utilizó un procesador con su caché de datos L1 desactivada.

El banco de pruebas empleado en los experimentos de radiación está compuesto por los siguientes dos programas: `Inversek2j (IN)` y `Forwardk2j (FW)`, que se obtuvieron de la *suite* de referencia *AxBench* para CA [7]. Ambos algoritmos se utilizan para calcular la cinemática, directa e inversa, de un brazo robótico de dos grados de libertad. Estas pruebas son esenciales, ya que los brazos robóticos no sólo se limitan al uso terrestre, sino que también tienen potencial para operar en el espacio [8], donde hay una cantidad significativa de exposición a la radiación. `Forwardk2j` (cinemática directa) calcula la posición del extremo del brazo en el espacio basándose en los ángulos de las articulaciones. `Inversek2j` (cinemática inversa) calcula los ángulos necesarios en las articulaciones del robot para alcanzar una posición deseada del extremo del brazo en el espacio cartesiano. En este trabajo, la entrada para ambos programas fue una matriz de 500 coordenadas en tipo coma flotante de 64 bits, es decir, 500×2.

Siguiendo el método de *iteraciones simplificadas*, sustituimos cálculos complejos por otros más sencillos en iteraciones específicas de la ejecución. El grado de aproximación varía en función del análisis funcional de cada algoritmo. Ambos algoritmos emplean funciones trigonométricas en sus ecuaciones cinemáticas y los cálculos de éstas conllevan una gran cantidad de recursos computacionales. Por tanto, sustituirlos por operaciones más sencillas supone una reducción significativa del tiempo de ejecución. Más concretamente, aproximamos las funciones *seno*, *coseno*, *arcseno* y *arcocoseno* sustituyéndolas por los segmentos de recta que se aproximan a los valores de las funciones trigonométricas originales.

Para las pruebas de radiación de este trabajo, seleccionamos dos versiones aproximadas (A2 y A5), además de la precisa (P). En la versión A2, las operaciones precisas se realizan cada 3 iteraciones, mientras que en la versión A5, se utilizaron cálculos precisos en una de cada seis, y las cinco iteraciones restantes utilizaron cálculos aproximadas. Con este método de CA, se obtuvo una aceleración de $2,81\times$ y $5,06\times$ para las versiones A2 y A5 en `Forwardk2j`. En el caso de `Inversek2j`, el aumento de velocidad obtenido fue de $2,84\times$ y $5,22\times$ para A2 y A5, respectivamente.

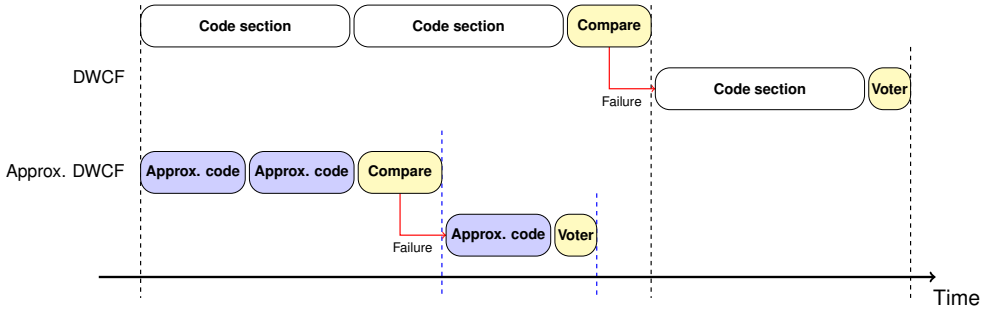


Fig. 2: *DWCF* con aproximación de iteraciones simplificadas.

La mejora del rendimiento de las versiones aproximadas se refleja en la reducción de la sobrecarga en el tiempo de ejecución debida al endurecimiento del programa. En el peor de los casos, en el que las tres ejecuciones del algoritmo `Forwardk2j` se realizan en *DWCF*, la sobrecarga para la versión precisa y endurecida es de $3,02\times$. Sin embargo, el uso de la versión A2 reduce la sobrecarga a $1,09\times$, y la versión A5 la reduce aún más, a $0,60\times$. Se obtienen resultados similares para el algoritmo `Inversek2j`, con una sobrecarga de $3,01\times$ para la versión precisa y $1,07\times$ y $0,59\times$ para las versiones A2 y A5, respectivamente.

Se realizó un experimento de radiación con el banco de pruebas mencionado. Ambos algoritmos se probaron utilizando versiones no endurecidas (`_UH_`) y endurecidas (`_H_`), y tres niveles de aproximación: cálculo preciso o exacto completo (`_P`) y las dos versiones aproximadas antes mencionadas: `_A2` y `_A5`. Como ejemplo de la notación empleada en este trabajo, `FW_UH_P` significa que hemos irradiado una versión no endurecida del algoritmo de cinemática directa `forwardk2j` a precisión completa. Todas las compilaciones se realizaron utilizando el compilador GCC del proyecto Linaro (versión `arm-eabi-gcc v7.2-2017.11`).

La campaña de radiación se realizó en la línea del haz externo del ciclotrón compacto 18/9 *Ion Beam Applications* situado en el Centro Nacional de Aceleradores (CNA) [9] en Sevilla, España, en marzo de 2023. El DUT fue irradiado sin decapar al aire libre. La energía del haz en la superficie del DUT fue de $15,3 MeV$, con una dispersión estimada del orden de $400 keV$. El flujo medio de protones se mantuvo en el rango de $5,0 - 6,7 \cdot 10^8 protones/(cm^2 \cdot s)$ y la fluencia total para cada ejecución estuvo dentro de los $4,2 - 8,8 \cdot 10^{11} protones/cm^2$. El punto de haz homogéneo de $16mm$ de diámetro cubrió el área de interés. La energía de los protones en la zona activa del silicio se considera suficiente para producir *SEUs* en un dispositivo de tecnología de 28nm sin decapar.

El *DUT* fue controlado desde un ordenador externo, en este caso una *RaspberryPi 3 Modelo B* que actúa como ordenador de control (*CC*), cuya tarea principal es la de supervisar el estado del DUT, recibir y registrar toda la información generada en el *DUT* y reiniciarlo cuando sea necesario. El *DUT* fue configurado para enviar un mensaje de estado cada 5-10s en ausencia de errores y el *CC* está programado para recibirlos. En caso de que no haya mensajes entrantes durante el doble del tiempo del mensaje, entonces el *CC* reinicia automáticamente y reprograma el *DUT*.

Hemos observado las siguientes categorías de error:

- Corrupción silenciosa de datos (*SDC*, *Silent Data Corruption*). La ejecución del *benchmark* ha terminado con errores en los datos de salida resultantes.
- *Hangs*: El flujo de ejecución del procesador ha sido interrumpido bruscamente por una excepción inesperada, probablemente causada por un acceso prohibido a memoria. Si no se manejan, este tipo de errores se convertirían en errores de tiempo de espera. El resultado es que con este error el DUT deja de responder.
- Error de comunicación: La comunicación serie con el procesador ha sido corrompida y los registros se vuelven ilegibles, por lo que es imposible identificar otros errores.
- Comunicación atascada e incongruente: El puerto serie entra en un bucle infinito, enviando el mismo mensaje de forma indefinida.

La tabla I representa los resultados del experimento de radiación. Mostramos, para cada punto de referencia, la fluencia total acumulada (Φ) en la segunda columna, el número total de eventos observados ($\#SDC$), detectados/recuperados ($\#Det/Rec$) y ($\#Hang/Invalid\ status$), junto con sus correspondientes secciones eficaces (σ) en las columnas 3 a 9. Obsérvese que, en aras de la simplicidad, los sucesos que provocan un estado no válido, tal como se ha indicado anteriormente (error de comunicación, fallos de comunicación, etc.), se recogen juntos y se etiquetan como ($\#Hang/Invalid\ status$) en la tabla. Los márgenes de error superior e inferior pueden verse encima y debajo de las distintas mediciones de *cross-sections*. Se calcularon utilizando la distribución chi-cuadrado inversa ($inv - \chi^2$) tal y como se describe en [10], con un nivel de confianza del 95 % considerando una incertidumbre de fluencia de $\pm 10\,\%$. Las versiones (`P`, `A2` y `A5`) de cada punto de referencia se enfrentan en filas separadas en la tabla para mostrar mejor la evidencia de la mejora de esta técnica de endurecimiento. Por último, para poder comparar las distintas versiones de los puntos de referencia teniendo en cuenta no sólo la tasa de error, sino también los gastos generales de tiempo inherentes, se utilizó la métrica *MWTF* (*Mean Work to Failure*).

Tabla I: Resultados del experimento con haz de protones

Bench	$\Phi \cdot 10^{11}(n/cm^2)$	#eventos			$\sigma \cdot 10^{-10}(cm^2)$				$MWTF \cdot 10^{12}$	
		#SDC	#Det/Rec	#Hang/ Invalid status	SDC	Det/Rec	Hang	Total	SDC	Hang
FW_UH_P	2,07	99	–	16 / 1	$4{,}79^{5{,}94}_{3{,}78}$	–	$0{,}78^{1{,}26}_{0{,}43}$	$5{,}62^{6{,}87}_{4{,}49}$	0.745	4.607
FW_H_P	3,13	0	3 / 75	13 / 4	0,03	$0{,}10^{0{,}28}_{0{,}02}$ / $2{,}40^{3{,}05}_{1{,}83}$	$0{,}42^{0{,}71}_{0{,}22}$	$0{,}64^{0{,}99}_{0{,}38}$	66.797	5.138
FW_UH_A2	3,56	139	–	12 / 4	$3{,}90^{4{,}71}_{3{,}17}$	–	$0{,}34^{0{,}59}_{0{,}17}$	$4{,}35^{5{,}21}_{3{,}56}$	1.199	13.889
FW_H_A2	3,76	0	6 / 63	18 / 11	0,03	$0{,}16^{0{,}35}_{0{,}06}$ / $1{,}68^{2{,}18}_{1{,}25}$	$0{,}48^{0{,}76}_{0{,}28}$	$0{,}93^{1{,}31}_{0{,}63}$	102.053	5.669
FW_UH_A5	3,19	105	–	17 / 4	$3{,}29^{4{,}06}_{2{,}61}$	–	$0{,}53^{0{,}86}_{0{,}30}$	$3{,}95^{4{,}80}_{3{,}18}$	1.547	9.552
FW_H_A5	2,57	0	8 / 52	19 / 5	0,04	$0{,}31^{0{,}62}_{0{,}13}$ / $2{,}02^{2{,}69}_{1{,}47}$	$0{,}74^{1{,}16}_{0{,}44}$	$1{,}25^{1{,}77}_{8{,}33}$	72.273	3.803
IN_UH_P	2,61	89	–	13 / 8	$3{,}41^{4{,}27}_{2{,}66}$	–	$0{,}50^{0{,}86}_{0{,}26}$	$4{,}22^{5{,}18}_{3{,}36}$	0.347	2.377
IN_H_P	1,78	0	7 / 34	17 / 11	0,06	$0{,}39^{0{,}81}_{0{,}16}$ / $1{,}91^{2{,}69}_{1{,}29}$	$0{,}96^{1{,}54}_{0{,}55}$	$1{,}97^{2{,}76}_{1{,}34}$	11.297	0.664
IN_UH_A2	2,99	106	–	16 / 3	$3{,}54^{4{,}36}_{2{,}81}$	–	$0{,}53^{0{,}87}_{0{,}30}$	$4{,}18^{5{,}08}_{3{,}36}$	0.695	4.602
IN_H_A2	2,92	0	7 / 61	18 / 5	0,03	$0{,}24^{0{,}50}_{0{,}09}$ / $2{,}09^{2{,}72}_{1{,}55}$	$0{,}62^{0{,}98}_{0{,}36}$	$1{,}03^{1{,}48}_{6{,}77}$	38.879	2.159
IN_UH_A5	2,68	81	–	21 / 3	$3{,}02^{3{,}82}_{2{,}33}$	–	$0{,}78^{1{,}21}_{0{,}48}$	$3{,}92^{4{,}83}_{3{,}11}$	1.092	4.214
IN_H_A5	2,88	0	5 / 52	22 / 9	0,03	$0{,}17^{0{,}41}_{0{,}06}$ / $1{,}81^{2{,}40}_{1{,}31}$	$0{,}76^{1{,}16}_{0{,}47}$	$1{,}25^{1{,}75}_{8{,}55}$	40.938	1.860

** No se observaron errores, por lo que se asume el peor caso para poder realizar comparaciones.

En cuanto al número de eventos *Hang/Invalid* observados, se aprecia cómo las versiones endurecidas siempre generan más eventos de esta categoría respecto a sus homólogos no endurecidos, con la única excepción de la primera fila (`FW_UH_P` vs `FW_H_P`). Este comportamiento era de esperar, debido a que el endurecido del código está pensado para incluir tanto *DWCF* como la técnica de CA. Esto implica una mayor exposición a fallos, ya que tenemos más código que ejecutar y con nuevos puntos críticos añadidos que tienen que ver con el flujo de control del programa.

Cabe mencionar que no se observaron eventos *SDC* en las versiones endurecidas de los puntos de referencia. En este escenario, las secciones eficaces correspondientes se calcularon asumiendo el peor caso de 1 SDC por experimento, evitando así una sección eficaz nula. Esto hace posible comparar las diferentes secciones eficaces teniendo en cuenta la fluencia: cuanto menor sea ésta, mayor será la *cross-section*.

Como era de esperar, las versiones endurecidas de cada algoritmo muestran una mejora en la sección eficaz de SDC. Esta mejora es de un orden de magnitud en cada caso. Por ejemplo, $4,79$ frente a $0,10$ ($10^{-10}cm^2$) en `FW_UH_P` y `FW_H_P`, respectivamente. Las mejoras oscilan entre $\times 160$ (`FW_UH_P` vs `FW_H_P`) y $\times 57$ (`IN_UH_P` vs `IN_H_P`), respectivamente. Centrándonos en las técnicas de CA (A2 y A5), las secciones eficaces de SDC aumentan dentro de lo previsible. El motivo por el que se observan secciones eficaces SDC mayores en las versiones A2 que en las A5 es por la diferencia en el tiempo de computación.

La figura 3 representa las diversas puntuaciones *MWTF SDC* y *Hang*, para cada punto de referencia, como una función monotónicamente creciente. Para ello, se han ordenado los ejes de abscisas correspondientes, de forma que se pueda extraer la información útil de esta disposición. Centrándonos en la salida *SDC*, observamos que, como era de esperar, un primer grupo de versiones no endurecidas, que se ordenan como `UH_P`, `UH_A2` y `UH_A5` muestran una mejora (`UH_A5` duplica a `UH_P`) que es 2 órdenes de magnitud menor que el grupo restante de versiones endurecidas (`H_P`, `H_A2` y `H_A5`). Asimismo, las versiones precisas obtienen peores puntuaciones en cada grupo. Esto se debe al aumento progresivo de la complejidad/duración de cada versión.

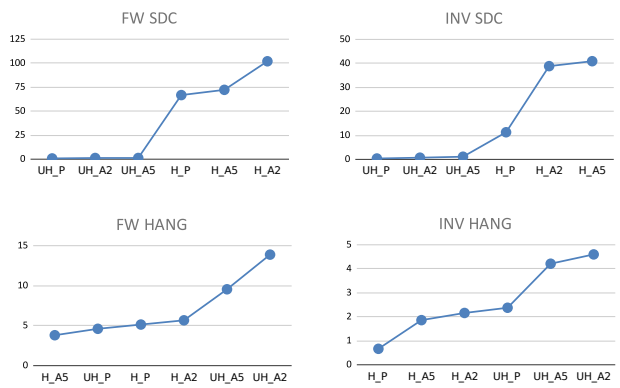


Fig. 3: Representación del comportamiento monotónico del MWTF ($\cdot 10^{12}$).

Recopilando la información relevante de la Tabla I y la Figura 3, vemos que la sola aplicación de las técnicas de CA supone una mejora de la protección como efecto secundario. De hecho, la sección eficaz del *MWTF* y *SDC* se hacen mayores en este caso. Esta mejora es más evidente en la sección transversal SDC. Si se aplica el híbrido CA + *DWCF*, observamos una mejora en ambas métricas. Lógicamente, no todos los algoritmos nos permiten variar la precisión mediante cálculo aproximado, por lo que estas técnicas se limitan a aquellos casos en los que es viable.

IV. Conclusiones

El empleo de computación aproximada, junto con técnicas de mitigación de fallos inducidos por radiación, ha conseguido reducir al mínimo los sobrecostes de tiempo que supone la aplicación de este tipo de estrategias. También se mantiene bajo control, en todo momento, la precisión de los cálculos que exige siempre la aplicación de técnicas de computación aproximada. El presente trabajo ha estudiado y evaluado una variación de una técnica de mitigación de fallos basada en la computación aproximada (que usa *DWCF* en lugar de la *TMR* tradicional) para reducir los *soft errors* inducidos por radiación en microprocesadores COTS.

Se estudiaron varias versiones protegidas y no protegidas con distintos niveles de aproximación y se efectuó una campaña de radiación con protones para la evaluación real de la protección en cada caso.

Los resultados experimentales bajo radiación muestran que la técnica puede detectar y corregir *SEUs* manteniendo la precisión bajo control y sin comprometer el rendimiento. El principal resultado de este trabajo es que el cálculo puede protegerse significativamente aplicando CA con o sin una técnica de endurecimiento. Es decir, CA mejora la protección frente a fallos como efecto secundario, tal y como revelan las dos métricas más relevantes (*SDC cross-section* y *MWFT*).

Agradecimientos

Los autores desean expresar su agradecimiento al Ministerio de Ciencia e Innovación de España por el financiamiento otorgado para llevar a cabo este estudio que está apoyado por el proyecto `PID2019-106455GB-C22` del mismo ministerio.

Referencias

[1] Gennaro S. Rodrigues, Ádria Barros de Oliveira, Fernanda Lima Kastensmidt, Vincent Pouget, and Alberto Bosio, "Assessing the reliability of successive approximate computing algorithms under fault injection," *Journal of Electronic Testing*, vol. 35, no. 3, pp. 367–381, Jun 2019.

[2] Alexander Aponte-Moreno, Alejandro Moncada, Felipe Restrepo-Calle, and Cesar Pedraza, "A review of approximate computing techniques towards fault mitigation in hw/sw systems," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, 2018, pp. 1–6.

[3] Sparsh Mittal, "A survey of techniques for approximate computing.," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, 2016.

[4] Alexander Aponte-Moreno, Felipe Restrepo-Calle, and Cesar Pedraza, "A low-cost fault tolerance method for arm and risc-v microprocessor-based systems using temporal redundancy and approximate computing through simplified iterations," *Journal of Integrated Circuits and Systems*, vol. 16, no. 3, pp. 1–14, 2021.

[5] Heather Marie Quinn, Zachary Kent Baker, Thomas D. Fairbanks, Justin Leonard Tripp, and Melvin George Duran Il, "Robust duplication with comparison methods in microcontrollers," *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, 1 2016.

[6] Xilix, UG585, "Zynq-7000 all programmable SoC: Technical reference manual," 2016.

[7] Amir Yazdanbakhsh, Divya Mahajan, Hadi Esmaeilzadeh, and Pejman Lotfi-Kamran, "AxBench: A Multiplatform Benchmark Suite for Approximate Computing," *IEEE Design & Test*, vol. 34, no. 2, pp. 60–68, apr 2017.

[8] Zhuoman Wen, Yanjie Wang, Jun Luo, Arjan Kuijper, Nan Di, and Minghe Jin, "Robust, fast and accurate vision-based localization of a cooperative target used for space robotic arm," *Acta Astronautica*, vol. 136, pp. 101–114, jul 2017.

[9] Centro Nacional de Aceleradores, "CNA," `http://www.cna.us.es`, Last visited: March 31st, 2023, Seville, Spain.

[10] ESA/ESCC, "Single event effects test method and guidelines. escc basic specification no. 25100," Oct 2014.

Inteligencia Artificial en sistemas empotrados

Despliegue de modelos de detección de objetos basados en aprendizaje profundo sobre dispositivos de edge computing y en la nube

Darío G. Lema [1], Rubén Usamentiaga [1] y Joaquín Entrialgo [1]

Resumen— **En este trabajo, se realiza una comparación cuantitativa para evaluar la viabilidad del despliegue de modelos de aprendizaje profundo en la detección de objetos en imágenes. Se analizan dos opciones de despliegue ampliamente utilizadas en la actualidad: la computación en la nube y el edge computing. Se emplea Azure como plataforma cloud, y se utilizan los dispositivos NVIDIA Jetson AGX Xavier y NVIDIA Jetson Nano. La aplicación seleccionada para evaluar el rendimiento de ambas alternativas es un sistema en tiempo real para verificar el uso de Equipos de Protección Individual (EPIs) en una instalación industrial. Se utiliza YOLOv5-S como detector de objetos y StrongSORT como algoritmo para realizar el seguimiento temporal de los trabajadores. Los resultados concluyen que, para lograr el tiempo real requerido, es necesario utilizar dispositivos de edge computing. Los tiempos de red necesarios para enviar y recibir la información a través de la nube resultan demasiado altos para procesar los datos con la suficiente rapidez. La principal contribución de este trabajo es un análisis cuantitativo sobre infraestructuras locales y en la nube que permita tomar la mejor decisión a la hora de desplegar aplicaciones basadas en modelos de aprendizaje profundo.**

Palabras clave— **Edge computing, cloud computing, tiempo real, detección de objetos.**

I. Introducción

La detección de objetos en imágenes ha sido objeto de investigación durante mucho tiempo y ha experimentado avances significativos en los últimos años, gracias al desarrollo de nuevas tecnologías como el aprendizaje automático, el aprendizaje profundo y las Unidades Gráficas de Procesamiento (GPUs) [1], [2]. Este enfoque se centra en la identificación de patrones en imágenes, utilizando algoritmos para analizar y detectar objetos, incluso cuando no son fácilmente reconocibles para el ojo humano.

El aprendizaje profundo permite entrenar algoritmos con grandes conjuntos de imágenes de ejemplo, lo que capacita al sistema para detectar objetos en imágenes. Las redes neuronales desempeñan un papel fundamental en este proceso, ya que aprenden a identificar objetos a partir de un conjunto de datos de entrenamiento. La detección de objetos se ha vuelto cada vez más común en diversos campos, como vehículos autónomos, robótica, seguridad y aplicaciones de realidad aumentada y virtual [3], [4].

No obstante, el procesamiento de datos en tiempo real y la transmisión de grandes volúmenes de información desde dispositivos periféricos plantean un desafío para la tecnología tradicional de detección de objetos. Es en este punto donde entra en juego dos de las formas de despliegue más extendidas actualmente: el edge computing y la nube. El edge computing [5], una tecnología emergente que permite realizar el procesamiento de datos en la proximidad del lugar donde se generan. El edge computing disminuye la latencia y el ancho de banda necesarios al procesar los datos directamente en el dispositivo, en lugar de enviarlos a una ubicación centralizada para su procesamiento. La computación en la nube [6] es un modelo de acceso a recursos informáticos a través de Internet, donde los servicios y aplicaciones se ejecutan en servidores remotos en lugar de en equipos locales. Este enfoque permite el almacenamiento, procesamiento y análisis de grandes cantidades de datos de manera eficiente y escalable. En relación a la detección de objetos, la computación en la nube proporciona una infraestructura robusta y flexible para ejecutar algoritmos de detección en grandes volúmenes de imágenes, permitiendo un procesamiento rápido y preciso. Además, facilita el acceso a modelos de aprendizaje automático y bases de datos compartidas, lo que impulsa el desarrollo y mejora de los sistemas de detección de objetos.

En este estudio, se analizan opciones de despliegue sobre dispositivos de edge computing y en la nube utilizando YOLOv5-S, uno de los algoritmos de detección de objetos más extendido, con el objetivo de realizar predicciones en tiempo real. En este estudio, se considera como tiempo real un intervalo de 33 ms. Esto se debe a que las cámaras de video vigilancia utilizadas para monitorear la planta industrial capturan imágenes a una velocidad de 30 frames por segundo. Se analizan también distintas opciones de despliegue desde el punto de vista del coste y del rendimiento. El propósito de este modelo es verificar el cumplimiento constante del uso de Equipos de Protección Individual (EPIs) dentro de una instalación industrial. Para que el sistema funcione de manera efectiva, es crucial realizar un análisis de información en todos los frames de las imágenes. Por lo tanto, el procesamiento de imágenes en tiempo real desempeña un papel vital en este proceso.

II. Módelos de aprendizaje profundo para la detección de objetos

Los modelos de aprendizaje profundo han revolucionado la detección de objetos al lograr un alto rendimiento y precisión en esta tarea. Estos modelos se basan en Redes Neuronales Convolucionales (CNN)

[1]Dpto. de Informática, Universidad de Oviedo (Gijón, Asturias), e-mail: gonzalezdario@uniovi.es

que son capaces de aprender automáticamente características y patrones relevantes en las imágenes. Utilizando grandes conjuntos de datos de entrenamiento que contienen imágenes etiquetadas con objetos, los modelos de aprendizaje profundo pueden aprender a reconocer y detectar una amplia variedad de objetos en imágenes.

Una de las ventajas clave de los modelos de aprendizaje profundo es su capacidad para aprender representaciones jerárquicas de las características de los objetos. A través de las múltiples capas de la red neuronal, estos modelos pueden aprender características de bajo nivel, como bordes y texturas, y combinarlas para formar representaciones de alto nivel que son específicas de los objetos que se buscan detectar. Esto permite una detección más precisa y robusta, incluso en situaciones desafiantes con variaciones en iluminación, pose y oclusión.

Además, los modelos de aprendizaje profundo para la detección de objetos son altamente adaptables y transferibles. Una vez entrenados en un conjunto de datos grande y diverso, estos modelos pueden ser utilizados para detectar objetos en nuevas imágenes y dominios sin necesidad de un entrenamiento exhaustivo desde cero. Esto proporciona una gran flexibilidad y escalabilidad en el despliegue de sistemas de detección de objetos en diferentes contextos y aplicaciones.

En los últimos años, se han desarrollado diferentes algoritmos de detección de objetos que ofrecen buenos resultados en conjuntos de datos publicos como Common Objects in Context (COCO) [7]. Estos algoritmos se dividen en dos tipos: detectores de dos etapas y de un estado. Los detectores de dos etapas proponen primero un conjunto de regiones de interés (ROIs) y luego realizan las detecciones correspondientes en estas regiones. Ejemplos claros de detectores de dos etapas son R-CNN, Fast R-CNN y Faster R-CNN [8], [9], [10]. Por otro lado, los detectores de un estado, como SSD y YOLO, realizan la detección tratando toda la imagen en su conjunto.

Uno de los mayores desafíos que presentan estos modelos es su despliegue en entornos de producción, especialmente en términos de velocidad de inferencia. En este caso, se han seleccionado detectores de un estado ya que son muy rápidos. SSD [11] mejora el tiempo de inferencia de R-CNN introduciendo mejoras como la detección a múltiples escalas, cajas predeterminadas y relaciones de aspecto. Por otro lado, YOLOv1 [12] se destaca por su capacidad de inferencia en tiempo real al dividir la imagen en una cuadrícula de tamaño $S \times S$ y detectar los objetos dentro de cada celda. A pesar de las ventajas que ofrece YOLOv1, tiene resultados inferiores a otros detectores.

Para superar las limitaciones de YOLOv1, se desarrollaron versiones posteriores como YOLOv2 [13] y YOLOv3 [14]. YOLOv2 introduce mejoras como los *anchor boxes*, que aprovechan las formas y relaciones de aspecto comunes de los objetos para generar predicciones iniciales más precisas. YOLOv3 aborda el problema de la detección de objetos pequeños y mejora la capacidad de realizar predicciones a tres escalas diferentes. Además, YOLOv4 [15] y YOLOv5 [16] son continuaciones naturales de YOLOv3, y presentan mejoras adicionales en la red neuronal y la generación de predicciones.

Estos detectores, incluyendo YOLOv5, introducen ideas innovadoras como la ampliación de datos mediante la técnica de mosaico, que combina partes de diferentes imágenes para mejorar la capacidad de generalización del modelo. Además, se utilizan cajas ancla automáticas para simplificar el proceso de selección de cajas ancla adecuadas para conjuntos de datos específicos. YOLOv5 presenta varias versiones: YOLOv5-N, YOLOv5-S, YOLOv5-M, YOLOv5-L y YOLOv5-X. La diferencia entre las diferentes versiones es la cantidad de capas convolucionales utilizadas. Generalmente, a más convoluciones mejores resultados aunque a veces esto no es así, debido por ejemplo al sobreajuste. Sin embargo, cuantas más convoluciones se usen más lento será el proceso de detección. Existen versiones más recientes de YOLO, sin embargo debido a su reciente lanzamiento no se analizan en este trabajo.

Uno de los mayores desafíos que presentan este tipo de modelos es su despliegue en entornos de producción. Aunque los modelos de aprendizaje profundo para la detección de objetos pueden lograr un alto rendimiento en etapas de investigación y desarrollo, su implementación en sistemas en tiempo real con recursos limitados puede ser complicada. Estos modelos suelen requerir una gran cantidad de memoria y capacidad de procesamiento, lo que puede ser problemático en dispositivos con recursos limitados [17].

III. Dispositivos de edge computing

La capacidad computacional de las CPU (Unidades de Procesamiento Central) resulta insuficiente para entrenar con grandes volúmenes de datos en tiempos razonables, y así poder crear modelos basados en CNNs. Por suerte, en los últimos años se han popularizado las GPU (Unidades Gráficas de Procesamiento). La importancia de las GPUs en la detección de objetos radica en la necesidad de llevar a cabo numerosas convoluciones para realizar esta tarea. Mientras que las CPUs son capaces de ejecutar una amplia variedad de operaciones, las GPUs se especializan en un conjunto específico de operaciones, pero a una velocidad considerablemente más rápida. Entre estas operaciones se encuentran la multiplicación de matrices, lo que convierte a las GPUs en herramientas ideales para este tipo de tareas. Sin embargo, hay dos desventajas asociadas a las GPUs: su alto consumo de energía y su elevado costo. En muchos casos, la movilidad es un factor importante y es necesario contar con una batería para alimentar el dispositivo encargado de la detección de objetos, por lo que se requiere un bajo consumo de energía. Además, adquirir múltiples GPUs puede resultar costoso. Por esta razón, los dispositivos integrados, conocidos como dispositivos de edge computing, están

ganando aceptación como una alternativa para proporcionar soluciones de procesamiento de imágenes a bajo costo, manteniendo un rendimiento óptimo y un consumo de energía reducido. Estos dispositivos disponen de una GPU integrada que ofrece capacidades de procesamiento suficientes para ejecutar algoritmos basados en CNNs sin la necesidad de utilizar GPUs externas. Esto los convierte en una opción atractiva para aplicaciones de detección de objetos en entornos con limitaciones de recursos. Además, al estar integrados en el dispositivo, brindan mayor flexibilidad y movilidad, lo que los hace adecuados para su uso en sistemas autónomos, cámaras de seguridad, vehículos autónomos y otros dispositivos inteligentes.

Estos dispositivos presentan una serie de ventajas sobre la computación en la nube [18]:

- Latencia: la reducción de la latencia es crucial para implementar servicios en tiempo real. En el caso del reconocimiento de objetos en imágenes, la latencia se compone del tiempo de transmisión de la imagen, el tiempo de inferencia y el tiempo de respuesta. Con la computación en la nube, es necesario enviar la imagen a través de una red WAN, lo que genera una alta latencia. Sin embargo, con el edge computing, no es necesario enviar la información fuera de la red local (LAN), lo que resulta en una menor latencia. Al procesar los datos en dispositivos de edge computing, se reducen los requisitos de transferencia de red y se pueden utilizar tamaños de entrada más grandes. En la computación en la nube, la transmisión de imágenes grandes aumenta la latencia de manera significativa, mientras que el uso de tamaños de entrada más grandes en el edge computing mejora la precisión del reconocimiento de objetos.
- Conectividad: la falta de conectividad es un factor importante a considerar en la implementación de soluciones de edge computing. En situaciones donde los dispositivos se encuentran en ubicaciones remotas y no tienen acceso a la red, el edge computing sigue siendo viable y funcional, ya que no depende de la conectividad externa. Por otro lado, las soluciones basadas en la computación en la nube se verían afectadas en tales escenarios, ya que requieren una conexión constante a la red para funcionar correctamente.
- Privacidad y seguridad: con el edge computing, no es necesario enviar datos fuera de la LAN, lo que significa un aumento en la privacidad. Si, por ejemplo, se recopilan imágenes faciales, enviarlas a través de la red puede implicar una violación de la privacidad. Además, los datos sensibles están más expuestos a posibles ciberataques.

En este trabajo se evaluan diferentes modelos de detección de objetos sobre los dos dispositivos de edge computing más populares a día de hoy: NVIDIA Jetson Nano y NVIDIA Jeson AGX Xavier. En la Figura 1 se muestra una imagen de cada uno de estos dispositivos, mientras que en la Tabla I se encuentran las especificaciones técnicas de ambos dispositivos.

El Kit de Desarrollo NVIDIA Jetson Nano es una pequeña y potente computadora diseñada para desarrollar soluciones de Inteligencia Artificial (IA) [19]. Está compuesto por una CPU Quad-Core ARM Cortex-A57 y una GPU NVIDIA Maxwell de 128 núcleos. Una de sus ventajas es que, al ser una computadora Linux pequeña, puede ejecutar casi cualquier modelo de aprendizaje profundo.

El Kit de Desarrollo NVIDIA Jetson AGX Xavier es un dispositivo integrado robusto que ofrece 32 TOPs (Operaciones Tera por Segundo). Está equipado con una CPU ARM v8.2 de ocho núcleos, una GPU NVIDIA Volta de 512 núcleos y dos motores NVDLA. Al igual que el Jetson Nano, es una computadora basada en Linux. Gracias a su hardware, logra velocidades de inferencia muy altas, aunque a un costo elevado.

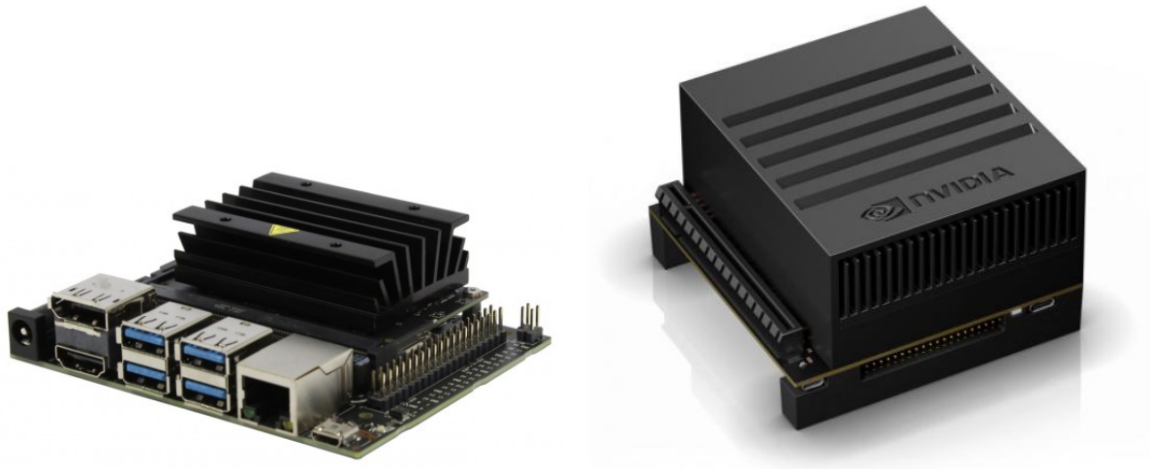
IV. Computación en la nube

La ventaja principal de la nube en el despliegue de aplicaciones de detección de objetos es la capacidad de escalar de manera flexible según las necesidades de la organización. Con Azure, plataforma de computación en la nube utilizada en este trabajo, las empresas pueden ajustar rápidamente la capacidad de procesamiento y almacenamiento en función de la carga de trabajo, lo que les permite manejar grandes volúmenes de datos de manera eficiente. Esto es especialmente importante en aplicaciones de detección de objetos, donde se pueden recibir y procesar grandes cantidades de imágenes en tiempo real.

Otra ventaja de la nube es la facilidad de acceso a recursos y servicios adicionales. Azure ofrece una amplia gama de servicios y herramientas que pueden complementar las aplicaciones de detección de objetos, como Azure Machine Learning, que permite entrenar y mejorar los modelos de detección de objetos con algoritmos de aprendizaje automático avanzados. Además, Azure proporciona servicios de análisis y visualización de datos, como Azure Data Lake Analytics y Power BI, que permiten extraer información valiosa de los datos de detección de objetos y presentarla de manera intuitiva y comprensible.

La nube también ofrece una mayor disponibilidad y fiabilidad en comparación con las soluciones de edge computing. Azure garantiza altos niveles de tiempo de actividad y redundancia de datos, lo que significa que las aplicaciones de detección de objetos estarán disponibles y funcionando de manera continua. Además, Azure ofrece servicios de seguridad avanzados, como Azure Security Center, que ayudan a proteger los datos y las aplicaciones contra amenazas y ataques cibernéticos.

Dentro de la nube existen diferentes paradígmas. Algunos de los más importantes son: Infrastucture as a Service (IaaS), Container as a Service (CaaS) y Function as a Service (FaaS). IaaS es un modelo de computación en la nube en el que se proporciona infraestructura virtualizada, como servidores, alma-



(a) (b)

Fig. 1: (a) NVIDIA Jetson Nano. (b) NVIDIA Jeson AGX Xavier.

Tabla I: Especificaciones técnicas de NVIDIA Jetson Nano y NVIDIA Jetson AGX Xavier

	Nvidia Jetson Nano [20]	**Nvidia Jetson AGX Xavier** [21]
CPU	Quad-Core ARM Cortex-A57	Eight-Core ARM v8.2
GPU	NVIDIA Maxwell 128-Cores	NVIDIA Volta 512-Cores
Memoria	2GB – 4 GB LPDDR4	32 GB LPDDR4
Alamacenamiento	16 GB eMMC, MicroSD slot	32GB eMMC
Conexión a red	Gigabit Ethernet	Gigabit Ethernet
USB	4× USB 3.0, USB 2.0 Micro-B	2× USB-C 3.1
Consumo eléctrico	5 – 10 W	10 – 30 W
S.O.	Linux (Ubuntu 18.04)	Linux (Ubuntu 18.04)
Tamaño (mm)	100 × 80 × 29	105 × 105 × 65
Precio	50 €– 95 €	650 €

cenamiento y redes, a través de Internet. Los proveedores de IaaS ofrecen una plataforma escalable y flexible que permite a los usuarios desplegar y administrar sus propias aplicaciones y sistemas operativos sin tener que preocuparse por la gestión de la infraestructura subyacente. CaaS es un modelo en el que se proporciona un entorno de ejecución basado en contenedores a través de la nube. Los contenedores son unidades de software portables y autocontenidas que encapsulan aplicaciones y sus dependencias, lo que facilita su despliegue y gestión. Con CaaS, los usuarios pueden crear, implementar y escalar contenedores de manera eficiente, sin preocuparse por la infraestructura subyacente. Y FaaS es un modelo de computación en la nube en el que se ejecutan funciones individuales en respuesta a eventos específicos. Los desarrolladores pueden cargar sus funciones en la nube y el proveedor de FaaS se encarga de gestionar automáticamente la infraestructura y la escalabilidad. Esto permite una ejecución rápida y eficiente de funciones, lo que resulta útil en escenarios donde se requiere una respuesta rápida a eventos o se necesita ejecutar pequeñas porciones de código de manera independiente.

V. Resultados experimentales

A. Sistena de verificación de EPIs

El escenario de aplicación se ubica en un almacén de bobinas de acero para la fabricación de productos, como automóviles. La disposición de las bobinas crea pasillos que dificultan la visibilidad, por lo que se instalaron dos cámaras en cada pasillo, una en cada extremo, para garantizar la seguridad de los trabajadores. Con un total de 37 pasillos, se requerirán 74 cámaras en total. Si se utiliza dispositivos de edge computing, se necesitará un dispositivo por cámara.

Antes de poner en funcionamiento el sistema, se realizó el entrenamiento de un modelo de detección de objetos. Se entrenó un modelo utilizando el dataset público CHV [22] ya que permite detectar cascos, chalecos y personas. A continuación, se aplicó transfer learning, ajustándolo con imágenes específicas de la instalación. Después de analizar los resultados con diferentes versiones de YOLOv5, se determinó que la versión S es la más adecuada para este contexto. Las versiones más grandes no mejoraron considerablemente las métricas y presentan tiempos de inferencia más grandes.

Las métricas utilizadas para evaluar el modelo son precision, recall y F1. La precision, mostrada en la Eq 1, sirve para medir cómo de fiables son las predicciones realizadas. La recall, mostrada en la Eq 2,

Tabla II: Resultados de generación de alarmas con el modelo lógico y YOLOv5-S.

	Precision	*Recall*	F_1
Alarma–chaleco	0.7435	0.7277	0.7355
Alarma–casco	0.5263	0.5172	0.5217
No alarma	0.8441	0.7585	0.7990

indica el porcentaje de detecciones realizadas sobre el total de objetos a detectar. La F1, definida en Eq 3, sirve para ponderar precision y recall en un solo valor. La Tabla II muestra los resultados de las alarmas por no usar chaleco (alarma–chaleco), por no usar casco (alarma–casco) y trabajadores que usan ambos (No alarma) utilizando una confianza de 0.6. La conclusión es clara, estos resultados no son lo suficientemente buenos como para implementar un sistema válido.

$$Precision = \frac{TPs}{TPs + FPs} \tag{1}$$

$$Recall = \frac{TPs}{TPs + FNs} \tag{2}$$

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3}$$

Tras analizar manualmente las detecciones realizadas, se observa que el principal problema que afecta a estas métricas es la cantidad de Falsos Positivos (FPs) generados. En este contexto, un FP podría ser que se detecte una persona sin casco, cuando realmente si está haciedo uso de él. Esto se puede solucionar si en vez de analizar frame a frame se toma la decisión de generar la alarma teniendo en cuenta información temporal. En la Figura 2 se muestra un ejemplo de la problemática planteada. En los seis frames mostrados, se puede observar que solo en el tercero no se detecta el chaleco del trabajador. Si se utilizara únicamente la detección de objetos, este ejemplo generaría una alarma, ya que el chaleco no sería detectado. Sin embargo, en los otros cinco frames, el chaleco sí se detectó. Esto plantea la necesidad de utilizar un sistema de seguimiento de objetos para evitar generar falsas alarmas.

Para abordar este desafío, se ha seleccionado el algoritmo StrongSORT [23] como rastreador de objetos. Este algoritmo se basa en el uso de detectores de objetos, y en este caso, se ha utilizado el modelo YOLOv5-S. El objetivo de utilizar StrongSORT es seguir la trayectoria de los objetos detectados a lo largo del tiempo, lo que permite tener un contexto más completo y tomar decisiones más precisas.

El enfoque combinado de detección de objetos y seguimiento de objetos es fundamental para prevenir falsas alarmas en situaciones como la no detección de chalecos de trabajadores. A través de la utilización de algoritmos como StrongSORT y modelos de detección como YOLOv5-S, se logra una mayor robustez y precisión en el sistema, evitando situaciones en las que un solo frame pueda generar alarmas innecesarias debido a falsos positivos.

Tabla III: Valores de N_{Init} y Max_{Age} para obtener los mejores resultados

	MOTA	N_{Init}	Max_{Age}
Alarma–chaleco	76 %	5	5
Alarma–casco	63 %	5	4

Para utilizar un rastreador de objetos, es necesario ajustar tres parámetros clave:

- Max_{Age}: Este parámetro define el número máximo de frames en los que un objeto no detectado se descarta. Por ejemplo, si Max_{Age} se establece en 5 y un objeto no es detectado durante 6 frames consecutivos, la próxima vez que se detecte se considerará como un objeto completamente nuevo.
- N_{Init}: Este parámetro establece el número mínimo de frames consecutivos en los que se debe detectar un objeto para ser considerado válido. Si un objeto no se detecta en al menos N_{Init} frames seguidos, se descarta y se trata como un objeto nuevo cuando se detecta nuevamente.
- NN_{Budget}: Para realizar el seguimiento de un objeto en diferentes frames, es necesario calcular la distancia entre la ubicación del objeto en el frame actual y el frame anterior. NN_{Budget} determina el número de frames utilizados para calcular esta distancia y, por lo tanto, afecta la precisión y la velocidad del seguimiento.

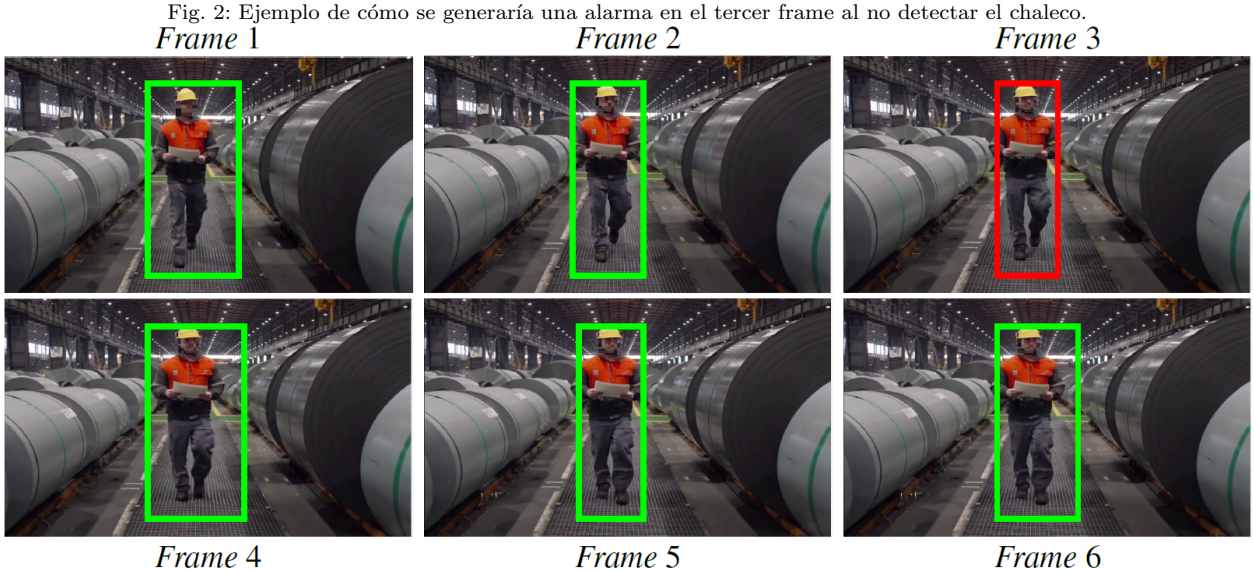
Ajustar adecuadamente estos parámetros es crucial para lograr un seguimiento preciso y confiable de los objetos en un sistema de rastreo. Para evaluar los resultados obtenidos con el rastredor de objetos se utiliza la métrica MOTA [24]. Como la velocidad a la que se mueven los trabajadores no es muy alta, el parámetro NN_{Budget} permanece constante con un valor de 50. En la Tabla III se muestra la mejor configuración para alarmas por falta falta de chaleco y casco.

B. Despliegue del sistema

En esta sección se analizan los resultados tras evaluar YOLOv5-S con un tamaño de 416 × 416 en los dos dispositivos de edge computing, y sobre diferentes infraestructuras en Azure. Para seleccionar la infraestructura adecuada, es fundamental considerar dos aspectos principales: la velocidad de procesamiento del sistema y su coste. Para analizar la velocidad del sistema, se deben tener en cuenta dos tiempos:

- Tiempo de red: dado que el cliente y el servidor se encuentran en ubicaciones diferentes, existe un tiempo de comunicación entre ellos conocido como tiempo de red. Este tiempo se refiere al tiempo necesario para enviar las imágenes, procesarlas y recibir los resultados.
- Tiempo de inferencia: es el tiempo requerido para procesar las imágenes utilizando el algoritmo de detección de objetos y obtener los objetos de interés.

Fig. 2: Ejemplo de cómo se generaría una alarma en el tercer frame al no detectar el chaleco.



Una vez establecida la forma de medir el tiempo necesario para realizar el servicio, se procede a alquilar diferentes instancias en Azure utilizando los modelos de servicio IaaS, CaaS y FaaS. Se ha creado un cliente y varios servidores en Azure. Los resultados obtenidos se presentan en la Tabla IV. En la tabla se comparan diferentes instancias: IaaS (con distintos tipos de máquina: NC6, NC12, B8MS, B12MS, D8sv3, A8mv2), CaaS (con diferentes cantidades de CPUs) y FaaS.

Los tipos de instancia NC6 y NC12 tienen una y dos GPU respectivamente, pero el uso de dos GPU en el caso de NC12 no mejora el tiempo de inferencia. El tipo de instancia D8sv3 mejora el rendimiento en comparación con B8MS, B12MS y A8mv2. En CaaS, se observa una mejora significativa al pasar de una CPU a dos, pero no se encuentra una mejora adicional al reemplazar dos CPUs con tres o cuatro.

La instancia FaaS tiene el tiempo total más alto, pero es importante tener en cuenta su naturaleza serverless. El tiempo de transmisión de la red no es constante, sino que varía entre las diferentes instancias. Por lo tanto, se concluye que dependiendo de la instancia contratada, la disponibilidad de red varía, lo que explica las diferencias en los tiempos de red.

En cuanto al coste, las instancias NC6 y NC12, que ofrecen una y dos GPU respectivamente, son las más costosas debido a sus características [25], [26], [27], seguidas por las demás opciones de IaaS analizadas. La opción más económica por hora de uso en la nube es utilizar un CaaS con una CPU. Sin embargo, la opción FaaS tiene un precio similar y es ligeramente más rápida. Además, tiene la ventaja de que se cobra por el uso real en lugar de tener la máquina encendida constantemente.

En lo referente al edge computing se han evaluado dos de los dispositivos más populares. Ambos dipositivos son de NVIDIA por lo que se ha utilizado TensorRT como framework. TensorRT es una biblioteca de inferencia de alto rendimiento. Está diseñada específicamente para optimizar y acelerar la inferencia de modelos de aprendizaje automático en hardware NVIDIA, como GPUs y plataformas de procesamiento en paralelo. Los resultados muestran como estos dispositivos son capaces de procesar modelos de aprendizaje profundo de detección de objetos. La NVIDIA Jetson AGX Xavier procesa una imagen con YOLOv5-S y un tamaño de entrada de 416×416 en 22 ms. La NVIDIA Jetson Nano en 68 ms. Las instacias NC6 y NC12 compiten con la NVIDIA Jetson AGX Xavier en velocidad de inferencia, mientras que la NVIDIA Jetson Nano procesa imágenes más lento. Aún así, es capaz de procesar imágenes más rápido que el resto de instancias cloud.

La clave con los dispositivos de edge computing es que no es necesario transmitir la información por red. Gracias a ello, la NVIDIA Jetson AGX Xavier puede realizar el servicio el doble de rápido que la instancia NC6, ya que esta necesita 21 ms para transmitir los datos. Es más, la NVIDIA Jetson Nano compite con la instancia NC12 debido al alto tiempo de red que presenta esta opción cloud.

VI. Conclusiones

En este estudio, se han analizado dos de las opciones más populares para implementar modelos de aprendizaje profundo: el edge computing y la computación en la nube. Se seleccionó el algoritmo de detección de objetos YOLOv5-S para evaluar el despliegue de un sistema capaz de verificar en tiempo real el uso de Equipos de Protección Individual en un entorno industrial. Esta elección se basa en la combinación de velocidad de inferencia y precisión que ofrece este detector. Además, para controlar el número de FP se decidió utilizar un rastreador de objetos capaz de analizar información temporal.

Tras analizar los resultados, queda claro que se requiere un sistema con GPUs para lograr un tiempo real de procesamiento (30 FPS). En la plataforma cloud Azure, se disponen de dos infraestructuras con GPU: NC6 y NC12. Ambas infraestructuras logran un tiempo de inferencia inferior a 33 ms, cumplien-

Tabla IV: Resultado de evaluar YOLOv5-S con tamaño de entrada de 416×416 sobre múltiples instancias de Azure. Los tiempos se encuentran en ms.

Instancia	Paradigma	Modo	T. Inferencia	T. Red	T.Total	% T. Red	Coste
NC6	IaaS	GPU	24.5	21.0	**45.5**	46 %	0.98 €/h
NC6	IaaS	CPU	78.3	20.4	98.7	21 %	0.98 €/h
NC12	IaaS	GPU	25.4	41.3	66.7	62 %	1.97 €/h
NC12	IaaS	CPU	89.0	24.9	113.9	22 %	1.97 €/h
B8MS	IaaS	CPU	134.1	112.9	247.0	46 %	0.32 €/h
B12MS	IaaS	CPU	129.6	110.4	240.0	46 %	0.49 €/h
D8s v3	IaaS	CPU	90.2	118.7	208.9	57 %	0.40 €/h
A8m v2	IaaS	CPU	197.9	79.9	277.8	29 %	0.46 €/h
CaaS: 1vCPU	CaaS	CPU	242.6	18.3	260.9	7 %	0.11 €/h
CaaS: 2vCPU	CaaS	CPU	128.5	18.9	**147.4**	13 %	0.15 €/h
CaaS: 3vCPU	CaaS	CPU	136.1	18.6	154.7	12 %	0.19 €/h
CaaS: 4vCPU	CaaS	CPU	142.3	18.3	160.6	11 %	0.23 €/h
FaaS	FaaS	CPU	220.2	121.6	341.8	36 %	0.15 €/h
Jetson Nano	Edge	GPU	68.8	0	68.8	0 %	50 €
Jetson Xavier	Edge	GPU	22.4	0	**22.4**	0 %	650 €

do así con el requisito de tiempo real. Sin embargo, es importante tener en cuenta que, al tratarse de la nube, se debe transmitir la información desde la fuente hasta el centro de procesamiento, y luego enviar los resultados de vuelta. Este tiempo, conocido como tiempo de red, oscila entre 20 y 40 ms para estas infraestructuras. La variación en el tiempo de red se debe a que los proveedores de servicios en la nube ofrecen diferentes anchos de banda según la infraestructura seleccionada.

Estos tiempos de red relativamente altos, sumados al tiempo de inferencia, hacen que el procesamiento de imágenes utilizando YOLOv5-S en la nube no cumpla con el requisito de 30 FPS. Sin embargo, existe la alternativa de utilizar el edge computing. En este caso, los datos se procesan en la fuente, lo que significa que no es necesario enviarlos a través de la red hacia un centro de procesamiento. Por lo tanto, los tiempos de red son cercanos a cero. La NVIDIA Jetson Nano, con un tiempo de inferencia de alrededor de 70 ms para YOLOv5-S y un tamaño de entrada de 416×416 píxeles, no cumple con el objetivo de procesar las imágenes en tiempo real. Sin embargo, la NVIDIA Jetson AGX Xavier, con un tiempo de inferencia de 22 ms, sí cumple con el objetivo de procesamiento en tiempo real, siendo una opción viable para esta tarea.

Otro factor a tener en cuenta es el coste. En el caso de la computación en la nube se paga por uso, mientras que con el edge computing se adquiere el dispositivo y no hay ningún otro coste adicional. En este tipo de servicios, donde se está utilizando el capacidad de procesamiento continuamente, el edge computing es mucho más barato. Por ejemplo, en el caso de utilizar el sistema 8 horas al día durante un año con una instancia NC6 es coste sería de 2860 €. Si por el contrario, se opta por adquirir la NVIDIA Jetson AGX Xavier el coste sería de 650 €, es decir el coste de adquisición. Dicho esto, la elección de la computación en la nube o la utilización de dipostivos de edge computing dependerá del escenario de uso, ya que si lo que se buusca es escalabilidad y alta fiabilidad el cloud es una mejor opción.

En el futuro se evaluaran nuevos algortimos detectores de objetos que ofrezcan mejores resultados que YOLOv5, junto con nuevas infraestructuras en la nuebe y dispositivos de edge computing.

Agradecimientos

El presente trabajo ha sido financiado mediante el plan nacional PID2021-124383OB-I00.

Referencias

[1] Sepp Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 1907–1915.

[4] Darío G Lema, Oscar D Pedrayes, Rubén Usamentiaga, Pablo Venegas, and Daniel F García, "Automated detection of subsurface defects using active thermography and deep learning object detectors," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–13, 2022.

[5] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun, "An overview on edge computing research," *IEEE Access*, vol. 8, pp. 85714–85728, 2020.

[6] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo, "Cloud computing: An overview," in *Cloud Computing*, Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong, Eds., Berlin, Heidelberg, 2009, pp. 626–631, Springer Berlin Heidelberg.

[7] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[8] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[10] Ross Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[11] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, Eds., Cham, 2016, pp. 21–37, Springer International Publishing.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[13] Joseph Redmon and Ali Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[14] Joseph Redmon and Ali Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[15] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[16] Glenn Jocher, "Yolov5," Mayo 2020, [Online] Accedido el 15 de Marzo de 2023. Disponible en: `https://github.com/ultralytics/yolov5`.

[17] Daniel Padilla Carrasco, Hatem A. Rashwan, Miguel Ángel García, and Domènec Puig, "T-yolo: Tiny vehicle detection based on yolo and multi-scale convolutional neural networks," *IEEE Access*, vol. 11, pp. 22430–22440, 2023.

[18] Jiasi Chen and Xukan Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[19] Alessio Gagliardi, Francesco de Gioia, and Sergio Saponara, "A real-time video smoke detection algorithm based on kalman filter and cnn," *Journal of Real-Time Image Processing*, vol. 18, no. 6, pp. 2085–2095, Dec 2021.

[20] NVIDIA, "Jetson nano," Oct. 2020, [Online] Accessed on 18 May 2022. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[21] NVIDIA, "Jetson agx xavier," Oct. 2020, [Online] Accessed on 18 May 2022. https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit.

[22] Zijian Wang, Yimin Wu, Lichao Yang, Arjun Thirunavukarasu, Colin Evison, and Yifan Zhao, "Fast personal protective equipment detection for real construction sites using deep learning approaches," *Sensors*, vol. 21, no. 10, 2021.

[23] Yunhao Du, Yang Song, Bo Yang, and Yanyun Zhao, "Strongsort: Make deepsort great again," *arXiv preprint arXiv:2202.13514*, 2022.

[24] Patrick Dendorfer, Hamid Rezatofighi, Anton Milan, Javen Shi, Daniel Cremers, Ian Reid, Stefan Roth, Konrad Schindler, and Laura Leal-Taixé, "Mot20: A benchmark for multi object tracking in crowded scenes," *arXiv preprint arXiv:2003.09003*, 2020.

[25] Microsoft, "Azure virtual machines pricing," Mar. 2021, [Online] Accedido el 26 de Marzo de 2021. Disponible en: `https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/`.

[26] Microsoft, "Azure container instances pricing," Mar. 2021, [Online] Accedido el 26 de Marzo de 2021. Disponible en: `https://azure.microsoft.com/en-us/pricing/details/container-instances/`.

[27] Microsoft, "Azure functions pricing," Mar. 2021, [Online] Accedido el 26 de Marzo de 2021. Disponible en: `https://azure.microsoft.com/en-us/pricing/details/functions/`.

[28] U.S. Bureau of Labor Statistics, "Census of fatal occupational injuries summary, 2020," Accessed on 06 Jul 2022.

[29] Eurostat, "Accidents at work statistics," Accessed on 29 Jun 2022.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, Eds. 2012, vol. 25, Curran Associates, Inc.

[31] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed. 1989, vol. 2, Morgan-Kaufmann.

[32] Stephen Cass, "Taking ai to the edge: Google's tpu now comes in a maker-friendly package," *IEEE Spectrum*, vol. 56, no. 5, pp. 16–17, 2019.

[33] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

Tensor decompositions for neural networks compression on resource-constrained devices

Unai Sainz de la Maza[1,2], Jose A. Pascual[1], Javier Navaridas [1], Txomin Romero [2]

Abstract— As the demand for deploying machine learning models on resource-constrained devices grows, neural network compression has become an important area of research. Tensor decomposition is a promising technique for compressing neural networks, as it enables the representation of the network weights in a lower-dimensional format, while maintaining their accuracy and performance. In this work, we explore the application of tensor decomposition techniques, including Canonical Polyadic (CP) decomposition and Tucker decomposition, for neural network compression. We provide an exhaustive overview of both tensor decomposition methods and compare their performance in terms of compression rates model size and accuracy. We implement and evaluate the different compression methods on the CIFAR-10 benchmark dataset, using popular models such as ResNet and VGG. Our results show that tensor decomposition can significantly reduce the number of parameters of neural networks and the size of the model, while maintaining their accuracy. Finally, we discuss the challenges and opportunities of using tensor decomposition for neural network compression and highlight the open research questions in this field.

Keywords— Neural networks, tensor networks, compression, resource-contrained devices

I. Introduction

In recent years, deep neural networks (DNNs) have become a powerful and valuable tool in many industrial and commercial applications. In this manner, DNNs have been successfully used in various domains such as computer vision, audio processing, speech recognition, etc. In computer vision, Convolutional Neural Networks (CNNs) have achieved the best performance within the state-of-the-art on several tasks like image classification [1], object detection [2], semantic segmentation [3], and human pose estimation [4].

One of the key features behind these methods is over-parametrization, for which there is evidence that helps to find a local minima [5]. However, over-parametrization leads to issues like redundancy, or making generalization harder, because it excessively increases the number of parameters. Also, increasing the number of parameters has an impact in terms of storage and computational requirements. Because of that, the deployment of these over-parametrized models on devices with limited computational resources, such as mobile devices and edge computing, is sometimes unfeasible.

Neural network compression is the process of reducing the size and complexity of a neural network, while maintaining its accuracy and improving its performance. Compressed neural networks have a smaller memory footprint, require less computational resources for training and inference, and can be deployed more efficiently on low-power devices such as mobile phones or embedded systems. Several approaches have been proposed to reduce the redundancy and improve the efficiency of the models such as quantization, network pruning, weight sharing, knowledge distillation and low-rank factorization. For the interested reader, a general survey about neural network compression methods is provided in [6].

Tensor methods are mathematical techniques for representing high-dimensional data using lower-dimensional structures, which can significantly reduce the storage and computational requirements. Tensors are multidimensional arrays and a core mathematical object in multilinear algebra that arise naturally in a wide range of applications such as quantum physics simulations [7], signal processing [8], numerical linear algebra [9], neuroscience [10], graph analysis [9], data mining [11], and more. Tensor methods have been used in recent years for compressing neural networks, taking advantage of the inherent multidimensional structure of neural networks to achieve compression by decomposing the weight tensors of the neural network into a low-dimensional core tensor and a set of matrices, or into a collection of lower dimensional tensors. Indeed, tensor decompositions can be applied to the weights of neural network layers to compress them, and in some cases, speed them up [12].

The Tucker decomposition (or Higher-order singular value decomposition) [13] is one of the earliest methods used for compressing neural networks, where we decompose the high-dimensional tensor into a core tensor multiplied by a matrix along each mode. However, other popular methods such as Canonical Polyadic (CP) decomposition [14], in which we factorize a tensor into a sum of outer products of vectors are also used to reduce the number of parameters in the neural networks while preserving its performance. Other tensor decomposition algorithms have been used in the recent literature for neural network compression, such as Tensor-Train [15], Tensor Ring [16], Hierarquical Tucker [17], and Block-Term decomposition [18].

In this work, we explore the tensorization of different models based on CNNs such as ResNet [19] and VGG [20]. We have focused our work on tensorizing the convolutional layers of these models using Tensor Decompositions, including Canonical Polyadic and Tucker. The goal is to compare both tensorization strategies in terms of compression rate size of

[1]Faculty of Informatics, University of the Basque Country UPV/EHU, e-mail: {joseantonio.pascual, javier.navaridas, txomin.romero}@ehu.eus.

[2]Donostia International Physics Center, e-mail: unai.sainzdelamaza@dipc.org.

the model and accuracy trade-off. Overall, we have found that these techniques can substantially reduce the memory consumption of the models (up to one order of magnitude), without significantly affecting the accuracy of the models (within 1-4%).

II. Related work

In this section, we provide a brief review of the different techniques used for compressing neural networks.

Neural network compression has been an active research area in recent years due to the growing demand for deploying machine learning models on resource-constrained devices. Various techniques have been proposed to compress neural networks, including quantization, pruning, knowledge distillation, and tensor decomposition.

Quantization is concerned with quantizing the weights and/or the features of a neural network [21], in this straightforward way, we can speed up neural network computations and minimize memory requirements. Network pruning is an approach to reduce a heavy network to obtain a light-weight form by removing redundancy in the heavy network, different types of pruning have been proposed, e.g., structured pruning [22], unstructured pruning [23], and many others. Knowledge distillation [24], is the process of transferring knowledge from a large model to a smaller one, the key idea behind this approach is that while large models have higher knowledge capacity than small models, this capacity might not be fully utilized.

Tensor decomposition [9], in particular, has received increasing attention as an effective method for compressing neural networks. Tensor decomposition techniques aim to represent the weights of a neural network in a lower-dimensional format, by decomposing the weight tensor into a set of factor matrices along each mode. This approach can significantly reduce the storage and computational requirements of the network, which is crucial for deployment on edge devices and embedded systems.

Several tensor decomposition techniques have been proposed in the literature, including Canonical Polyadic (CP) decomposition [14], Tucker decomposition [13], Hierarchical Tucker (HT) decomposition [17], and Tensor Train (TT) decomposition [15]. Where each of these techniques has its strengths and weaknesses, and the choice of the decomposition method depends on the specific use case. CP decomposition represents a tensor as a sum of rank-one tensors and is particularly useful for compressing convolutional layers. Tucker decomposition decomposes a tensor into a small core tensor and factor matrices and is suitable for compressing fully connected layers. Hierarchical Tucker decomposition is an extension of Tucker decomposition that allows for a more flexible decomposition of tensors with high modes. TT decomposition decomposes a tensor into a set of TT cores and is particularly useful for compressing very high-dimensional tensors.

The use of tensor decompositions for neural network compression was first investigated in [25], where CP decomposition was proposed to compress a 4-dimensional convolution kernel. Additionally, low-rank matrix factorization was used to speed up the inference time of the CNNs. In a similar vein, [12] treated weight matrices as multi-dimensional tensors and applied the TT decomposition algorithm to compress feed-forward layers. The authors achieved high compression ratios without significant loss of accuracy, and provided theoretical support for the proposed method. Following this approach, in [26], the authors propose a TT decomposition based tensorization, both for the feed-forward and the convolutional layers of different CNN models. More recently, this approach of using TT decomposition based tensorization have been studied from a physics perspective in [27], and extended to compress 3D convolutional neural networks for video classification in [28].

Inspired in [25], other works have tried to compress CNNs using CP decomposition. For example, in [29], the authors improve the CP decomposition algorithm and they propose a Tensor Convolutional Neuro-Network (TCNN) for anomaly detection. In [30], they compress 3DCNNs using CP decomposition with an application to spatio-temporal facial emotion analysis. In [31], after tensorizing the convolutional layers via CP decompostion, they analyze the value of the decomposed kernels to guide feature selection. Also, in works such as [32], the authors use Tucker decomposition with nonlinear response to compress the convolutional layers, [33] propose a hybrid tensor decomposition scheme, where they combine TT decomposition for feed-forward layers and hierarchical tucker for the convolutional layers. In [34] the authors propose T-Net, a fully parametrized CNN with a single high-order low-rank tensor using TT and Tucker decompositions, this allows them to regularize the whole network thanks to the low-rank structure imposed on the weight tensor and drastically reduce the number of parameters.

III. Tensor methods

In this section we introduce briefly the definition and notation used to describe tensors and a class of techniques known as Tensor Networks (TNs) [35], that are used to deal with very large tensors by representing them as collections of small interconnected tensors, also called "blocks", "cores", "factors", or "components". The primary goal of TNs is to approximate the large tensors in a compressed and distributed manner, overcoming the curse of dimensionality associated with these tensors [36].

A. Notation

Tensors [37], also known as multi-way arrays, can be seen as higher-order extensions of vectors (1^{st}-order tensors), or matrices (2^{nd}-order tensors). In the same way as rows and columns in a matrix, an Nth-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$ has N modes (i.e., orders, ways, or indices) whose dimensions (i.e.,

lengths) are represented by $I_1, \ldots, I_N$. An element $(i_1, i_2, \ldots, i_N)$ of tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$ is accessed as: $\mathcal{X}_{i_1,i_2,\ldots,i_N}$ or $\mathcal{X}(i_1, i_2, \ldots, i_N)$. Also, given a set of N matrices (or vectors) that correspond to each mode of $\mathcal{X}$, the n^{th} matrix (or vector) is denoted as $\mathbf{U}^{(n)}$ (or $\mathbf{u}^{(n)}$). Furthermore, *fibers* are the higher-order generalization of the concept of rows and columns of matrices to tensors, obtained by fixing all indices but one, where to represent all the elements of a mode, we use a colon. For instance, if $\mathcal{X}$ is a third order tensor, then its mode-1 (column) fibers can be denoted as $\mathcal{X}_{:,j,k}$ (see Figure 1). *Slices* are two-dimensional sections of a tensor, defined by fixing all but two indices. As shown in Figure 2, the horizontal, lateral and frontal slices of a third-order tensor $\mathcal{X}$ are denoted by $\mathcal{X}_{i_1::}, \mathcal{X}_{:i_2:}$, and $\mathcal{X}_{::i_3}$, respectively.

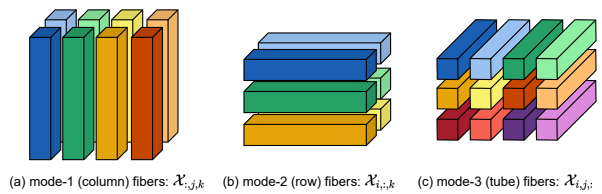


Fig. 1: Fibers of a third order tensor.

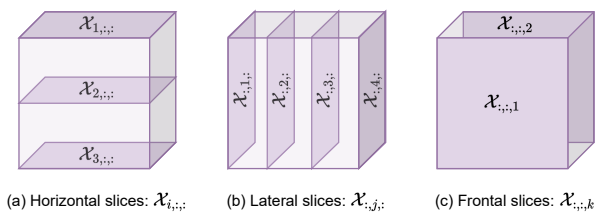


Fig. 2: Slices of a third order tensor of size $3 \times 4 \times 2$.

B. Tensor Diagrams

The intricate structures of advanced TNs are difficult to comprehend using only mathematical notation. To aid in the comprehension of the interconnections between tensors within TNs, Roger Penrose introduced TN diagrams in the early 1970s [38]. These diagrams represent tensors as nodes with edges, providing a simple graphical representation of complex tensors [7]. Therefore, TN diagrams are a practical tool for visually presenting and conveniently representing complex tensors. Furthermore, the potential of tensor diagrams as a versatile tool for network analysis in the field of deep learning is noteworthy [39].

As illustrated in Figure 3, a tensor is denoted as a node with edges. The number of edges denotes the modes of a tensor, and the value of the edge represents the dimension of the corresponding mode. For example, a node with one edge represents a vector $\mathbf{v} \in \mathbb{R}^I$, a node with two edges represents a matrix $\mathbf{M} \in \mathbb{R}^{I \times J}$, and a node with three edges represents a third-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$. In general, a node with N edges represents a Nth-order tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \cdots \times I_N}$.

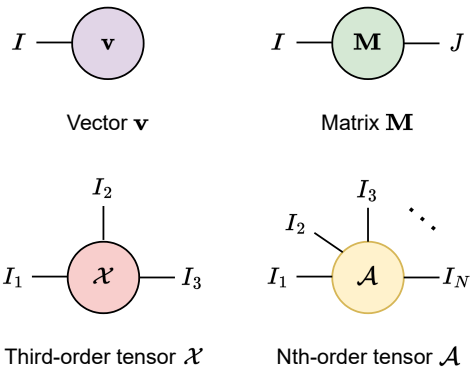


Fig. 3: Basic symbols for TN diagrams.

Tensors are linked by tensor contractions, which involve combining two tensors into one by pairing their corresponding indices. As a consequence, the connected edges vanish, while the dangling edges remain. See Figure 4, where the diagram of a matrix multiplication operation is shown. It is worth mentioning that matrix multiplication is the most common form of tensor contraction. In general, tensor contraction can be formulated as a tensor product. To compute tensor contractions among multiple tensors, such as in complex TNs, it is necessary to perform contractions sequentially between each pair of tensors. However, determining the order of these contractions is crucial to achieve better calculation efficiency.

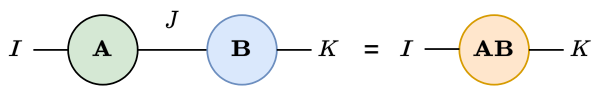


Fig. 4: Diagram of the most common tensor contraction form, i.e., matrix multiplication.

C. Tensor Decompositions

In this work, we adopt a unified approach to the terminology of "tensor decomposition" (TD) and "tensor network" (TN) as they are equivalent. TD models, such as CP [14] and Tucker decomposition [13], are considered basic types of TNs. It should be noted that different terminologies have been used for the same model, as TNs and TDs originated from different research fields. Here, we briefly introduce some of the most significant TDs by employing TN diagrams.

C.1 CANDECOMP/PARAFAC

CP factorization factorizes a higher-order tensor $\mathcal{X}$ into a sum of several rank-1 tensor component. The rank of the tensor $\mathcal{X}$ is the minimum number of rank-1 tensors that sum to $\mathcal{X}$, also known as the CP rank. This generalizes the notion of matrix rank to higher-order tensors. For instance, given a Nth-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$, each of its elements in the CP format can be expressed as

$$\mathcal{X}_{i_1,i_2,\ldots,i_N} \approx \sum_{r=1}^{R} \mathcal{G}_r \prod_{n=1}^{N} \mathcal{A}^{(n)}_{i_n,r}, \tag{1}$$

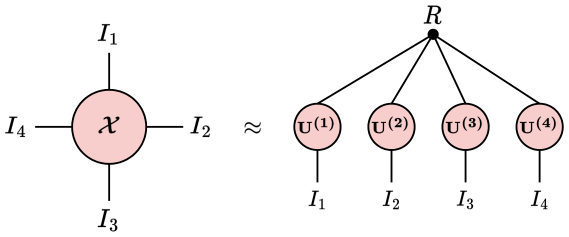


Fig. 5: TN diagram of the CP decomposition.

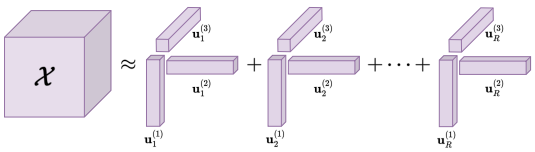


Fig. 6: CP decomposition of a third-order tensor $\mathcal{X}$ into a sum of rank-1 tensors.

where R denotes the CP rank, $\mathcal{G}$ represents the diagonal core tensor (consisting of R non-zero elements on the superdiagonal), and $\mathcal{A}^{(n)} \in \mathbb{R}^{I_n \times R}$ represents a series of factor matrices. See Figure 5 for the TN diagram that describes this decomposition.

One of the issues with this method is that computing the tensor rank, i.e., the number of rank-1 tensor components, is a NP-hard problem [40]. Therefore, a predefined CP rank R should be known in advance, this can be seen as fixing hyperparameters and using them to fit different CP-based models [9]. Other possible ways of computing the rank, such as estimating the rank from the data, e.g., using Bayesian approaches [41], or using deep neural networks [42] are also possible. An illustrative example can be found in Figure 6, where CP decomposition is applied to a third-order tensor $\mathcal{X}$.

C.2 Tucker Decomposition

The Tucker decomposition factorizes a higher-order tensor non-uniquely into a core tensor multiplied by a corresponding factor matrix along each mode. More precisely, given a Nth-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$, the Tucker decomposition can be formulated for each element as

$$\mathcal{X}_{i_1,i_2,\ldots,i_N} \approx \sum_{r_1,\ldots,r_N=1}^{R_1,\ldots,R_N} \mathcal{G}_{r_1,r_2,\ldots,r_N} \prod_{n=1}^{N} \mathcal{A}^{(n)}_{i_n,r_n}, \quad (2)$$

$$\mathcal{X}_{i_1,i_2,\ldots,i_N} \approx \sum_{r_1,\ldots,r_N=1}^{R_1,\ldots,R_N} \mathcal{G}_{r_1,r_2,\ldots,r_N} \prod_{n=1}^{N} \mathcal{A}^{(n)}_{i_n,r_n}, \quad (3)$$

$\mathcal{G}_1$ where $R = \{R_1, R_2, \ldots, R_N\}$ denotes the Tucker ranks, $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \cdots \times R_N}$ denotes the core tensor, and $\mathcal{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ denotes a factor matrix. The core tensor captures interactions between the columns of factor matrices, and if $R_n \ll I_n$, $\forall n$, the core tensor can be viewed as a compressed version of $\mathcal{X}$. It should be noted that in this case, $R_1, R_2, \ldots, R_N$ can take different values, which leads to the non-uniqueness of its decomposition results; see Figure 7 for an illustrative example. Finally, it is worth noting that by imposing the factor matrices to be orthonormal, the Tucker decomposition is known as the higher-order singular value decomposition (HOSVD) [43].

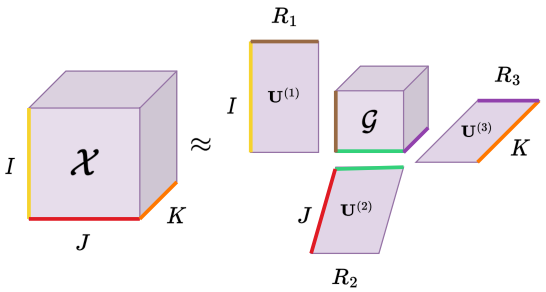


Fig. 7: Tucker decomposition of a third-order tensor $\mathcal{X}$.

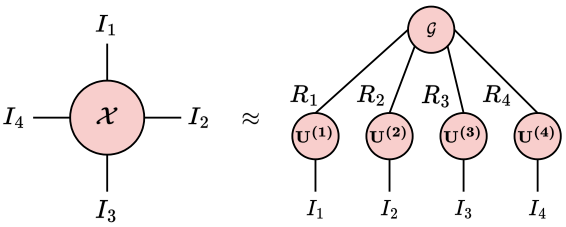


Fig. 8: TN diagram of the Tucker decomposition.

IV. Network compression using TDs

In this section, we will explore the application of tensor decomposition techniques to compress convolutional layers in deep neural networks. More precisely, we will use some of the TDs introduced in Section III such as CP and Tucker decompositions.

A. Tensorization of convolutional layers

Unlike in fully-connected layers, the weights of convolutional layers are naturally represented as tensors. For example, a 2D convolution is directly represented by a 4^{th}-order tensor. Therefore, we can directly apply tensor decompositions to compress convolutional layers.

While the utilization of tensor decomposition in CNNs is a recent and currently relevant development, the concept of representing linear operators through separable representations using tensor decomposition is not new (refer to [36] for more details). In this way, the motivation behind depthwise separable convolutions is to improve the efficiency and effectiveness of CNNs.

B. Depthwise separable convolutions

Traditional convolutional layers in CNNs apply a single filter to all input channels, resulting in a large number of computations. Depthwise separable convolutions aim to reduce this computational cost by decomposing the convolution operation into two separate steps: depthwise convolution and pointwise convolution.

Depthwise convolution applies a separate filter to each input channel independently, capturing

channel-wise spatial correlations. This step significantly reduces the number of parameters and computations compared to traditional convolutions.

Pointwise convolution, also known as 1×1 convolution, which can be viewed as a tensor contraction, is a fundamental example of convolution. It is commonly employed in deep neural networks to introduce data bottlenecks where the channel-wise information obtained from the depthwise convolution is combined, as seen in architectures like MobileNet [44] and Inception [45]. Let's consider a 1×1 convolution operator denoted as Φ. It is defined by a kernel tensor $\mathcal{W} \in \mathbb{R}^{T \times C \times 1 \times 1}$ and is applied to an activation tensor $\mathcal{X} \in \mathbb{R}^{C \times H \times W}$. The squeezed version of the kernel along the first mode is represented as $\mathbf{W} \in \mathbb{R}^{T \times C}$. Thus, we can express this as follows:

$$\Phi(\mathcal{X})_{t,y,x} = \mathcal{X} \star \mathcal{W} = \sum_{k=1}^{C} \mathcal{X}_{t,k,y,x} \mathcal{W}_{k,y,x} = \mathcal{X} \times_1 \mathcal{W}, \tag{4}$$

where $\times_1$ represents the mode-1 tensor contraction, $\star$ denotes the convolution operation, and $x = y = 1$ as we are considering a 1×1 convolution.

The depthwise separable convolution operation can be viewed as a form of tensor decomposition, where the original kernel tensor is decomposed into a set of smaller tensors.

C. Kruskal convolutions

CP decomposition allows separating modes of a convolutional kernel, resulting in a Kruskal form. This was proposed in [25], where the authors start from a pre-trained convolutional kernel and apply CP decomposition to it in order to obtain a separable convolution. In this case, the CP decomposition was achieved by minimizing the reconstruction error between the pretrained weights and the corresponding CP approximation. The authors demonstrated both space savings and computational speedups. Us-

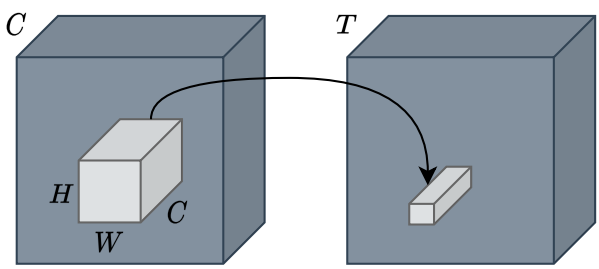


Fig. 9: Standard convolution operation.

ing a CP decomposition to factorize the kernel offers a notable benefit: the resulting factors can serve as parameters for a set of efficient depthwise separable convolutions, effectively replacing the original convolution operation [25]. Let's consider a standard convolution illustrated in Figure 9 and denoted as:

$$\mathcal{F}_{t,y,x} = \sum_{k=1}^{C} \sum_{j=1}^{H} \sum_{i=1}^{W} \mathcal{W}(t,k,j,i) \mathcal{X}(k, j+y, i+x) \tag{5}$$

It's Kruskal convolution is obtained by expanding the kernel $\mathcal{W}$ in the CP form as

$$\mathcal{F}_{t,y,x} = \underbrace{\sum_{r=1}^{R} \mathbf{U}_{t,r}^{(T)} \underbrace{\left[\sum_{i=1}^{W} \mathbf{U}_{i,r}^{(W)} \underbrace{\left(\underbrace{\sum_{j=1}^{H} \mathbf{U}_{j,r}^{(H)} \left[\underbrace{\sum_{k=1}^{C} \mathbf{U}_{k,r}^{(C)} \mathcal{X}(k, j+y, i+x)}_{1\times1 \text{ convolution}} \right]}_{\text{depthwise convolution}} \right)}_{\text{depthwise convolution}} \right]}_{\text{depthwise convolution}}}_{1\times1 \text{ convolution}}, \tag{6}$$

This expression is explained as follows: an initial 1×1 convolution reduces input channels to the rank (blue). Two depthwise convolutions are then applied to the height and width of the activation tensor (red and green). Lastly, a second 1×1 convolution restores the number of channels from the rank of the CP decomposition to the desired output channel count (black) (see Figure 10).

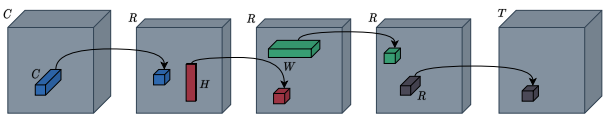


Fig. 10: Convolution operation with a CP-decomposed kernel, i.e., Kruskal form.

D. Tucker convolutions

As previously, we consider the convolution $\mathcal{F} = \mathcal{X} \star \mathcal{W}$ described in Equation 5. However, in this case we consider using Tucker decomposition to compress convolutional layers as proposed in [46].

First, instead of a Kruskal structure, the convolution kernel is assumed to admit Tucker format described as follows:

$$\mathcal{W}(t,s,j,i) = \sum_{r_1=0}^{R_1} \sum_{r_2=0}^{R_2} \sum_{r_3=0}^{R_3} \sum_{r_4=0}^{R_4} \mathcal{G}_{r_1,r_2,r_3,r_4} \mathbf{U}_{t,r_1}^{(T)} \mathbf{U}_{s,r_2}^{(C)} \mathbf{U}_{j,r_3}^{(H)} \mathbf{U}_{i,r_4}^{(W)}, \tag{7}$$

where $\mathbf{U}^{(n)}$, with $n = \{T, C, H, W\}$, are the four factor matrices, and $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times R_3 \times R_4}$ is the core tensor that represents the interactions between the modes.

Using the decomposed kernel tensor, we can create an efficient reformulation [46]: initially, the factors along the spatial dimensions are incorporated into the core tensor by expressing $\mathcal{H} = \mathcal{G} \times_3 \mathbf{U}_{j,r_3}^{(H)} \times_4 \mathbf{U}_{i,r_4}^{(W)}$. Rearranging the terms, we observe that a Tucker convolution is equivalent to sequentially handling the channel count transformation, performing a small convolution, and then restoring the channel dimension from the rank to the desired number of channels:

$$\mathcal{F}_{t,y,x} = \underbrace{\sum_{r_1=1}^{R_1} \mathbf{U}_{t,r_1}^{(T)} \left[\underbrace{\sum_{j=1}^{H} \sum_{i=1}^{W} \sum_{r_2=1}^{R_2} \mathcal{H}_{r_1,r_2,j,i} \left(\underbrace{\sum_{k=1}^{C} \mathbf{U}_{k,r_2}^{(C)} \mathcal{X}(k, j+y, i+x)}_{1\times1 \text{ convolution}} \right)}_{H\times W \text{convolution}} \right]}_{1\times1 \text{ convolution}}. \tag{8}$$

This expression can be described more precisely as follows: after decomposing the full kernel, the

factors corresponding to input and output channels are utilized to parameterize 1 x 1 convolutions, applied to the initial (blue) and final (green) stages, respectively. The remaining two factors are integrated into the core tensor and employed to parameterize a (small) regular 2D convolution (red) (see Figure 11).

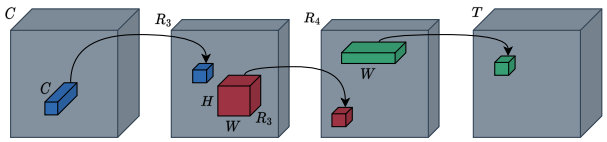


Fig. 11: Convolution operation with a Tucker-decomposed kernel.

V. Experimental setup

This section is devoted to explain the experimental setup used to evaluate the compression techniques. We first explain the dataset that has been used, following with the training parameters and performance metrics used in the evaluation.

A. Dataset

The CIFAR-10 dataset is composed of 60,000 images, with each color image (RGB) having a resolution of 32x32 pixels. These images are evenly distributed across ten object classes, with each class containing 6,000 images. The ten classes are as follows: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

The CIFAR-10 dataset [47] is divided into two subsets: a training set and a test set. The training set consists of 50,000 images, with each class having 5,000 samples, where we randomly select 5000 examples for the validation set. Note that these splits are made using the same random seed to maintain the reproducibility of the experiments. The remaining 10,000 images form the test set, serving as an independent benchmark to evaluate the generalization and performance of the trained models.

B. Training and evaluation framework

All the models are trained for 50 epochs, a batch size of 128, and a fixed learning rate of 1e−3. For the optimizer, we use the well known Adam optimizer [48].

To evaluate the performance of the models, as we are dealing with a balanced multi-class classification problem, we use the accuracy metric defined as

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{9}$$

where TP are true positives, TN true negatives, FP false positives, and FN false negatives. Additionally, as we are interested in measuring the trade-off between accuracy and parameter reduction, we need to define a metric to measure this reduction. Therefore, we define the compression ratio metric, CR, which measures the size reduction achieved dividing the number of parameters in the uncompressed model by the number of parameters in the compressed model.

$$\text{CR} = \frac{\#\text{params uncompressed}}{\#\text{params compressed}} \tag{10}$$

All the models were implemented using Pytorch [49], and tensorized via Tensorly [50]. Regarding the experimental platform, we have used several Nvidia RTX 3090 and RTX A5000 GPUs equipped with 24 GB of VRAM.

VI. Analysis of the results

To test the performance of each decomposition scheme, we apply each of the methods to the convolutional layers of two different architectures like ResNet [19], and VGG [20]. More precisely, we use several size configurations of these two architectures, i.e., ResNet18, ResNet50, VGG11, and VGG19 networks.

VGG11 and VGG19 networks are built sequentially stacking 8 and 16 convolutional layers, respectively, followed by 3 fully-connected layers. In the same way, ResNet18 is built stacking 17 convolutional layers, followed by a single fully-connected layer. On the other hand, ResNet50, which is composed by 49 convolutional layers followed by a fully-connected layer, divides the convolutional layers into several blocks, with each block containing a series of convolutions, i.e., 1×1 convolution, 3×3 convolution and 1×1 convolution, known as data bottlenecks [19].

To tensorize the networks, we replace all the convolutional layers but the first one of the different architectures by the TD-based convolutions, i.e., CP-decomposed (Section IV-C) and Tucker-decomposed (Section IV-D) convolutions. A summary of the different configurations is shown in Table I.

Network	Baseline configuration	Tensorized configuration
VGG11	8 convolutional layers 3 fully-connected layers	1 convolutional layer 7 TD-convolutional layers 3 fully-connected layers
VGG19	16 convolutional layers 3 fully-connected layers	1 convolutional layer 15 TD-convolutional layers 3 fully-connected layers
ResNet18	17 convolutional layers 1 fully-connected layer	1 convolutional layer 16 TD-convolutional layers 1 fully-connected layer
ResNet50	49 convolutional layers 1 fully-connected layer	1 convolutional layer 48 TD-convolutional layers 1 fully-connected layer

Table I: Networks configuration summary.

Finally, we compare our tensorized networks against the uncompressed models, where to select the ranks of the tensorized networks, we search optimal tensorization shapes and heuristic ranks by TensorLy [50] based on the target compression ratio.

A. Results on ResNet

The results presented in Table II demonstrate the effective compression of the ResNet18 network using both CP and Tucker decompositions. In comparison to the uncompressed baseline, the CP-based

model achieves a compression ratio of 1.965x by halving the number of parameters while only reducing the accuracy by 0.0052. Furthermore, retaining only 10% of the parameters and reducing the model size from 44.695MB to 5.153MB, the model experiences a minor accuracy reduction of 0.0243. Similarly, the Tucker decomposition exhibits similar behavior, with medium-high parameter retention such as 50% and 80% outperforming CP decomposition and maintaining the baseline accuracy. Figure 12 visually presents these results as well.

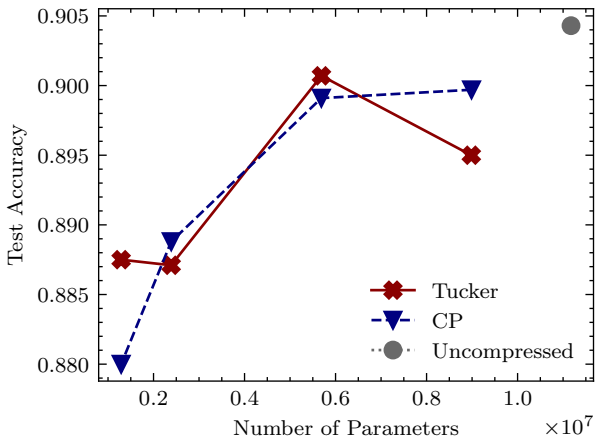


Fig. 12: Performance comparison of uncompressed and tensorized ResNet18 models with several compression ratios.

Method	%Param	#Param	Size (MB)	Comp. Ratio	Accuracy
Uncomp	1.0	11173962	44.695	1x	0.9043
CP	0.1	1288468	5.153	8.672x	0.88
CP	0.2	2389079	9.556	4.677x	0.8888
CP	0.5	5687499	22.749	1.965x	0.8991
CP	0.8	8987725	35.951	1.243x	0.8997
Tucker	0.1	1288513	5.154	8.672x	0.8875
Tucker	0.2	2387722	9.551	4.68x	0.8871
Tucker	0.5	5690166	22.761	1.964x	0.9007
Tucker	0.8	8983194	35.932	1.244x	0.9036

Table II: First results, compressing the convolutional layers of a ResNet18 model.

Moving on to ResNet50, although CP decomposition is typically discouraged due to unstable optimization [51], Table III reveals that it performs slightly better than Tucker decomposition. For instance, with a parameter retention of 10%, the CP-based model achieves a model size of 37.946MB compared to the uncompressed model's size of 94.080MB, resulting in an accuracy reduction of only 0.0092. On the other hand, the best performance for the Tucker-based models is observed at a parameter retention of 80%, yielding an accuracy of 0.9018.

Overall, considering that the ResNet architecture already employs more efficient convolutional layers, such as bottlenecks [19], we observe that TD-based compression significantly reduces the size of the models while minimally impacting the accuracy of the uncompressed baselines.

B. Results on VGG

In both Table IV and Table V, we can observe that the VGG architecture, which solely employs standard convolutional layers, achieves higher compression ratios compared to the ResNet models. This observation is particularly evident when analyzing the CP-based models.

Method	%Param	#Param	Size (MB)	Comp. Ratio	Accuracy
Uncomp	1.0	23520842	94.080	1x	0.9106
CP	0.1	9486467	37.946	2.479x	0.9014
CP	0.2	11050056	44.200	2.129x	0.8936
CP	0.5	15732335	62.929	1.495x	0.9029
CP	0.8	20409706	81.639	1.152x	0.9038
Tucker	0.1	9488912	37.956	2.479x	0.8953
Tucker	0.2	11046237	44.185	2.129x	0.8867
Tucker	0.5	15737527	62.950	1.495x	0.8859
Tucker	0.8	20410051	81.640	1.152x	0.9018

Table III: First results compressing the convolutional layers of a ResNet50 model.

Method	%Param	#Param	Size (MB)	Comp. Ratio	Accuracy
Uncomp	1.0	9231114	36.924	1x	0.8803
CP	0.1	937456	3.750	9.847x	0.8424
CP	0.2	1860182	7.441	4.962x	0.8565
CP	0.5	4627272	18.509	1.995x	0.8777
CP	0.8	7394881	29.580	1.248x	0.8747
Tucker	0.1	938922	3.756	9.832x	0.8437
Tucker	0.2	1858301	7.433	4.968x	0.8556
Tucker	0.5	4631515	18.526	1.993x	0.8691
Tucker	0.8	7395593	29.582	1.248x	0.8671

Table IV: First results compressing the convolutional layers of a VGG11 model.

In the case of VGG11, the CP-based model reduces the size from 36.924MB to 3.75MB, resulting in a significant model reduction. However, this reduction in size comes at the cost of a decrease in accuracy by 0.0379. Similarly, the CP-based VGG19 model demonstrates even higher model reduction, reducing the size from 80.162MB to 8.108MB, with a compression ratio of 9.887x. However, this also leads to a reduction in accuracy by 0.0311.

It is worth noting that compared to the ResNet models, VGG models experience a more considerable decrease in accuracy when using a small parameter retention setting. This suggests that the impact on accuracy is more pronounced in VGG models when employing aggressive compression techniques.

Method	%Param	#Param	Size (MB)	Comp. Ratio	Accuracy
Uncomp	1.0	20040522	80.162	1x	0.8937
CP	0.1	2026948	8.108	9.887x	0.8626
CP	0.2	4031038	16.124	4.972x	0.8687
CP	0.5	10040663	40.163	1.996x	0.8812
CP	0.8	16051973	64.208	1.248x	0.892
Tucker	0.1	2029861	8.119	9.873x	0.8625
Tucker	0.2	4028238	16.113	4.975x	0.8594
Tucker	0.5	10052199	40.209	1.994x	0.8834
Tucker	0.8	16056695	64.227	1.248x	0.8784

Table V: First results compressing the convolutional layers of a VGG19 model.

On the other hand, Tucker decomposition generally achieves worse results overall compared to the CP-based models. It fails to reach the accuracy obtained by the CP-based models even when considering different parameter retention settings.

In summary, the VGG architecture demonstrates higher compression ratios, especially with CP-based models. However, the reduction in accuracy is more substantial in VGG models when using lower parameter retention settings. Additionally, Tucker decomposition does not achieve the same accuracy levels as

the CP-based models across all parameter retention settings.

VII. Conclusions and future work

In this work, we have investigated the effectiveness of Tensor Decompositions (TDs) as a method for compressing convolutional layers in VGG and ResNet architectures. Our results demonstrate that TDs offer a promising approach for neural network compression, as they can significantly reduce the model size while minimally impacting the accuracy.

Specifically, we explored the usage of CP and Tucker decompositions, two widely used TD methods, to compress the convolutional layers. By decomposing the weight tensors of these layers into smaller rank tensors, we achieved substantial model size reduction without sacrificing much in terms of accuracy.

Our experiments revealed that TDs can effectively capture the important features and patterns in the convolutional layers, enabling the compressed models to retain their discriminative power. The minimal reduction in accuracy indicates that the compressed models can still achieve comparable performance to their original counterparts, despite their smaller size.

The reduction in model size achieved through TDs is particularly noteworthy. By compressing the convolutional layers using CP and Tucker decompositions, we were able to significantly decrease the number of parameters and the memory footprint of the models. This has important implications for resource-constrained environments, such as mobile devices or edge computing platforms, where memory and storage limitations often pose challenges.

The combination of compression and minimal accuracy loss makes TDs an attractive method for neural network compression. The compressed models not only require less storage space, but may also help to consume fewer computational resources during inference. This can lead to improved efficiency and faster execution times, properties which are highly desirable in various real-world applications.

The findings of this work contribute to the growing body of research on efficient and compact deep learning models, paving the way for more practical and resource-efficient deployments of neural networks in various domains. One area of focus is the development of customized and optimized kernels for tensor decompositions. This research involves investigating and developing efficient algorithms and computational techniques tailored to tensor decomposition, i.e., designing customized kernels specific to the target hardware architecture. Additionally, parallel computing techniques will be explored to accelerate the execution of tensor decomposition algorithms.

Another promising future direction is the deployment of TD-based CNNs in energy-efficient platforms. By integrating tensor decompositions with CNNs, we can reduce the number of parameters and improve computational efficiency. This research will explore the feasibility of deploying TD-based CNNs on platforms such as RISC-V or FPGAs, which offer energy-efficient computing capabilities. An important aspect of this work will be measuring and analyzing the energy consumption of TD-based CNNs at inference time, comparing their efficiency to traditional CNN architectures.

Acknowledgments

This work is supported by the Basque Government (projects ELKARTEK21/89 and IT1244-19) and by the Spanish Ministry of Economy and Competitiveness MINECO (PID2019-104966GB-I00). Dr. Javier Navaridas is a Ramón y Cajal fellow from the Spanish Ministry of Science, Innovation and Universities (RYC2018-024829-I).

References

[1] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie, "A ConvNet for the 2020s," Tech. Rep.

[2] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu, "Object Detection With Deep Learning: A Review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[3] Yujian Mo, Yan Wu, Xinneng Yang, Feilin Liu, and Yujun Liao, "Review the state-of-the-art technologies of semantic segmentation based on deep learning," *Neurocomputing*, vol. 493, pp. 626–646, 2022.

[4] Qi Dang, Jianqin Yin, Bin Wang, and Wenqing Zheng, "Deep learning based 2D human pose estimation: A survey," *Tsinghua Science and Technology*, vol. 24, no. 6, pp. 663–676, 2019.

[5] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song, "A Convergence Theory for Deep Learning via Over-Parameterization," in *Proceedings of the 36th International Conference on Machine Learning*, Kamalika Chaudhuri and Ruslan Salakhutdinov, Eds. 2 2019, vol. 97 of *Proceedings of Machine Learning Research*, pp. 242–252, PMLR.

[6] Brian R Bartoldson, Bhavya Kailkhura, and Davis Blalock, "Compute-Efficient Deep Learning: Algorithmic Trends and Opportunities," *Journal of Machine Learning Research*, vol. 24, pp. 1–77, 2023.

[7] Román Orús, "Tensor networks for complex quantum systems," *Nature Reviews Physics*, vol. 1, no. 9, pp. 538–550, 2019.

[8] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos, "Tensor Decomposition for Signal Processing and Machine Learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[9] Tamara G Kolda and Brett W Bader, "Tensor Decompositions and Applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 8 2009.

[10] Fengyu Cong, Qiu-Hua Lin, Li-Dan Kuang, Xiaofeng Gong, Piia Astikainen, and Tapani Ristaniemi, "Tensor decomposition of EEG signals: A brief review," *Journal of Neuroscience Methods*, vol. 248, pp. 59–69, 2015.

[11] Kijung Shin and U Kang, "Distributed Methods for High-Dimensional and Large-Scale Tensor Factorization," *2014 IEEE International Conference on Data Mining*, pp. 989–994, 2014.

[12] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov, "Tensorizing Neural Networks," *arXiv:1509.06569 [cs]*, 12 2015.

[13] Ledyard R Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[14] J Douglas Carroll and Jih-Jie Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[15] I V Oseledets, "Tensor-Train Decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 1 2011.

[16] Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki, "Tensor Ring Decomposition," *ArXiv*, vol. abs/1606.05535, 2016.

[17] Daniel Kressner and Christine Tobler, "Algorithm 941: Htucker—A Matlab Toolbox for Tensors in Hierarchical Tucker Format," *ACM Trans. Math. Softw.*, vol. 40, no. 3, 4 2014.

[18] Lieven De Lathauwer, "Decompositions of a Higher-Order Tensor in Block Terms—Part I: Lemmas for Partitioned Matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1022–1032, 2008.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[20] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[21] Song Han, Huizi Mao, and William J Dally, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding," *arXiv: Computer Vision and Pattern Recognition*, 2015.

[22] M Xia, Zexuan Zhong, and Danqi Chen, "Structured Pruning Learns Compact and Accurate Models," in *Annual Meeting of the Association for Computational Linguistics*, 2022.

[23] Trevor Gale, Erich Elsen, and Sara Hooker, "The State of Sparsity in Deep Neural Networks," *ArXiv*, vol. abs/1902.09574, 2019.

[24] Jianping Gou, B Yu, Stephen J Maybank, and Dacheng Tao, "Knowledge Distillation: A Survey," *International Journal of Computer Vision*, vol. 129, pp. 1789 – 1819, 2020.

[25] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, I Oseledets, and Victor S Lempitsky, "Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition," *CoRR*, vol. abs/1412.6553, 2014.

[26] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov, "Ultimate tensorization: compressing convolutional and FC layers alike," 11 2016.

[27] Richik Sengupta, Soumik Adhikary, Ivan Oseledets, and Jacob Biamonte, "Tensor networks in machine learning," 7 2022.

[28] Dingheng Wang, Guangshe Zhao, Guoqi Li, Lei Deng, and Yang Wu, "Compressing 3DCNNs Based on Tensor Train Decomposition," *Neural Networks*, vol. 131, pp. 215–230, 11 2020.

[29] Xiaolong Wu, "TCNN: a Tensor Convolutional Neuro-Network for big data anomaly detection," p. 7.

[30] Jean Kossaifi, Antoine Toisoul, Adrian Bulat, Yannis Panagakis, Timothy Hospedales, and Maja Pantic, "Factorized Higher-Order CNNs with an Application to Spatio-Temporal Emotion Estimation," 3 2020.

[31] Yinan Wang, Weihong "Grace" Guo, and Xiaowei Yue, "Tensor decomposition to Compress Convolutional Layers in Deep Learning," *IISE Transactions*, pp. 1–60, 4 2021.

[32] Ye Liu and Michael K Ng, "Deep neural network compression by Tucker decomposition with nonlinear response," *Knowledge-Based Systems*, vol. 241, pp. 108171, 4 2022.

[33] Bijiao Wu, Dingheng Wang, Guangshe Zhao, Lei Deng, and Guoqi Li, "Hybrid tensor decomposition in neural network compression," *Neural Networks*, vol. 132, pp. 309–320, 12 2020.

[34] Jean Kossaifi, Adrian Bulat, Georgios Tzimiropoulos, and Maja Pantic, "T-Net: Parametrizing Fully Convolutional Nets with a Single High-Order Tensor," 4 2019.

[35] Jacob Biamonte and Ville Bergholm, "Tensor networks in a nutshell," *arXiv preprint arXiv:1708.00006*, 2017.

[36] Andrzej Cichocki, Anh-Huy Phan, Qibin Zhao, Namgil Lee, Ivan Oseledets, Masashi Sugiyama, Danilo P Mandic, and others, "Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives," *Foundations and Trends® in Machine Learning*, vol. 9, no. 6, pp. 431–673, 2017.

[37] Cesar F Caiafa and Andrzej Cichocki, "Generalizing the column–row matrix decomposition to multi-way arrays," *Linear Algebra and its Applications*, vol. 433, no. 3, pp. 557–573, 2010.

[38] Roger Penrose, "Applications of negative dimensional tensors," *Combinatorial mathematics and its applications*, vol. 1, pp. 221–244, 1971.

[39] Yoav Levine, Noam Wies, Or Sharir, Nadav Cohen, and Amnon Shashua, "Chapter 7 - Tensors for deep learning theory: Analyzing deep learning architectures via tensorization," in *Tensors for Data Processing*, Yipeng Liu, Ed., pp. 215–248. Academic Press, 2022.

[40] Johan Håstad, "Tensor rank is np-complete," *J. Algorithms*, vol. 11, pp. 644–654, 1989.

[41] Shinichi Nakajima, Masashi Sugiyama, S Derin Babacan, and Ryota Tomioka, "Global analytic solution of fully-observed variational bayesian matrix factorization," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1–37, 2013.

[42] Zhiyu Cheng, Baopu Li, Yanwen Fan, and Yingze Bao, "A novel rank selection scheme in tensor ring decomposition based on reinforcement learning for deep neural networks," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 3292–3296.

[43] Göran Bergqvist and Erik G Larsson, "The higher-order singular value decomposition: Theory and an application [lecture notes]," *IEEE signal processing magazine*, vol. 27, no. 3, pp. 151–154, 2010.

[44] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[45] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[46] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin, "Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications," 2 2016.

[47] Alex Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[48] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[49] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al., "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[50] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic, "Tensorly: Tensor learning in python," *Journal of Machine Learning Research*, vol. 20, no. 26, pp. 1–6, 2019.

[51] Vin De Silva and Lek-Heng Lim, "Tensor rank and the ill-posedness of the best low-rank approximation problem," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1084–1127, 2008.

Clasificación de alimentos a partir de imágenes con dispositivos móviles

Bernabé Sánchez Sos[1] , Jorge Azorín López[2] y Andrés Fuster Guilló[2]

Resumen— **El presente artículo científico aborda la creación de un clasificador de imágenes, capaz de clasificar hasta 20 platos distintos y su incorporación a una aplicación móvil centrada en la gestión dietética y nutricional.**

Palabras clave— **Deep Learning, Visión por Computador, Clasificación de imágenes, Aplicación móvil.**

I. Introducción

En los últimos años, el avance de la Inteligencia Artificial y la Visión por Computador ha revolucionado numerosos campos de estudio, incluyendo la clasificación de objetos y reconocimiento de imágenes. En particular, la aplicación de técnicas de Aprendizaje Profundo (Deep Learning) ha demostrado un gran potencial en la resolución de tareas complejas relacionadas con la interpretación y comprensión de imágenes. Un campo en el cual el Aprendizaje Profundo ha encontrado una aplicación prometedora es la clasificación de platos en el ámbito de la nutrición.

La clasificación de platos es una tarea desafiante debido a la amplia variedad de formas, colores, texturas y presentaciones que pueden presentar los alimentos. Sin embargo, gracias a los avances en la tecnología y la disponibilidad de grandes conjuntos de datos etiquetados, los sistemas de Visión por Computador basados en Deep Learning han logrado abrirse camino en esta área.

El objetivo de este artículo científico es explorar y analizar diferentes enfoques y técnicas utilizadas para la clasificación de platos utilizando Deep Learning y Visión por Computador. Se compararán y seleccionarán modelos y arquitecturas de redes neuronales profundas ya existentes para abordar este tipo de problemas de clasificación de imágenes. Además, se mostrará la transformación e incorporación del modelo de red diseñado a una aplicación para dispositivos móviles, para facilitar su uso y acceso a los usuarios.

II. Estado del Arte

A. Food-101

El dataset Food-101 [1] es uno de los datasets más utilizados actualmente para la clasificación de platos alimenticios y, consiste en un conjunto de datos de 101 categorías de alimentos, que almacena y emplea 101.000 imágenes etiquetadas con el nombre del plato, 1.000 por plato. Las imágenes están compuestas por unidades de información digital conocidas como píxeles, formadas a su vez por 3 valores entre 0 y 255 (2^8 bits) que representan el modelo de color RGB. El ancho y alto varía, pero tienen en común que ninguno de los dos valores sobrepasa los 512 píxeles.

B. Red Neuronal Convolucional (CNN)

Una CNN es un tipo red neuronal profunda que se centra en la clasificación de información almacenada en imágenes sin tener en cuenta características espaciales. Verbi gracia, en la resolución del problema de clasificación de platos, la red no se centra en localizar la posición en la que se encuentra un determinado plato, sino que detecta el plato independientemente de la posición en la que se encuentre y, lo clasifica.

Como su nombre indica, la operación principal de este tipo de red es la convolución de matrices, un proceso que permite reducir la cantidad de parámetros que necesita aprender la red y resaltar y extraer características descriptivas de las imágenes [2]. Supongamos que el formato de entrada al modelo de red es de 6x6x3, es decir, 6 píxeles de ancho, 6 píxeles de alto y una profundidad de píxel de 3 (RGB). Sobre cada una de las imágenes se va a aplicar un kernel de 3x3x3 donde cada plano de este convolucionará con su plano de la imagen correspondiente, que dependiendo de los valores que contenga, suavizará la imagen, detectará bordes, realzará la imagen o eliminará ruido presente en la imagen.

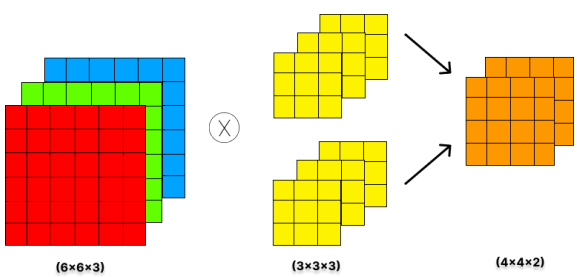


Fig. 1: Operación de convolución

En el caso de la figura 1, se puede apreciar de manera visual como se produce la operación de convolución de una imagen RGB y, como tras aplicar dos kernel 3x3x3, se reduce el número de parámetros de 108 (6x6x3) a 32 (4x4x2). El número de kernels es arbitrario en función de qué características de la imagen se deseen extraer para resolver un problema. Si el número de kernels aumentase, cambiaría la estructura de la información que va a procesar de la red, ya que pasaríamos de un dato con un ancho y alto considerable pero con una profundidad baja (3) a un dato con menor ancho y alto, pero con una profundi-

[1]Ingeniería Multimedia, Universidad de Alicante, e-mail: bss42@gcloud.ua.es.
[2]Dpto. de Tecnología Informática y Computación, Universidad de Alicante, e-mail: {jazorin,fuster}@ua.es.

dad superior que corresponderá al número de kernel aplicados.

B.1 Average Pooling

La operación de agrupación promedio es una técnica utilizada en Deep Learning para reducir las dimensiones de un tensor tridimensional y reducir el sobreajuste durante la fase de entrenamiento [3]. El funcionamiento de este método se basa en agrupar y realizar el promedio matrices en función del hiperparámetro stride. En la figura 2, se realiza la función de agrupación sobre una matriz 4x4x3 utilizando un filtro 2x2 y un stride de valor 2.

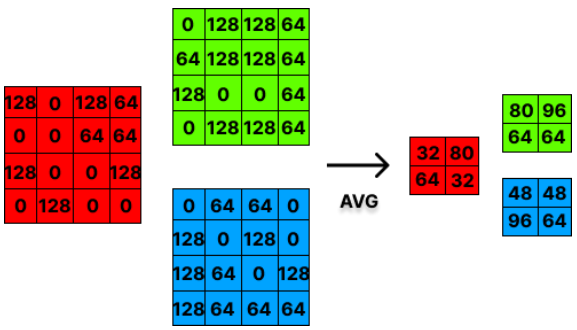


Fig. 2: Agrupación promedio

Este método sumado a las operaciones de convolución de la red acabarán dando como resultado un mapa de características con un ancho y alto con valor 1 y una profundidad que dependerá del número de convoluciones y agrupaciones promedio realizadas. Posteriormente, este resultado será enviado a las últimas capas para poder predecir la clase a la que pertenece la imagen de entrada.

B.2 Dropout

Dropout es un algoritmo relativamente nuevo para entrenar redes neuronales que se basa en abandonar estocásticamente las neuronas durante el entrenamiento, con el fin de evitar la coadaptación en la detección de características [4]. De esta manera, se fuerza a las neuronas a que trabajen de manera independiente, sin tener que depender de la información aportada por las neuronas más próximas. Además, evita el fenómeno conocido como sobreajuste, que hace que la red neuronal se acostumbre a unos datos de entrada determinados sin mejorar durante el entrenamiento o al usar otros datos para entrenamiento, pruebas o predicción.

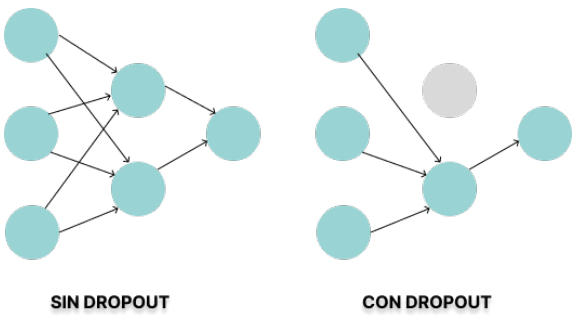


Fig. 3: Dropout

B.3 Dense

La adición de una o varias capas densas a la red hace que mejore el rendimiento si la red no está sobreajustada, ya que intenta reducir la pérdida durante el entrenamiento. Cada capa Dense empleará una función de activación denominada relu, que se añade a las capas ocultas, ésta transforma los valores negativos en cero y es usada para obtener mejores resultados durante la fase de entrenamiento. La última capa de la red también suele ser una capa Dense, conocida como capa totalmente conectada y en los problemas de clasificación en los que intervienen más de dos categorías, se utiliza junto a una función de activación softmax, la cual calcula la probabilidad para cada una de las categorías del problema propuesto [5].

B.4 BatchNormalization

Esta capa de normalización por lotes es capaz de aumentar el rendimiento, haciendo que el tiempo de entrenamiento de la red se reduzca y mejore su precisión. Esto es gracias a la centralización y normalización de cada lote de imágenes a partir de una media y una desviación que se calcula con ese lote, después se vuelve a reescalar y descentrar la información contenida en las imágenes [6].

III. Metodología

A. *Experimentos iniciales*

La primera tarea en el desarrollo de la red neuronal convolucional para clasificar imágenes era encontrar un modelo existente que se pudiese utilizar para resolver el problema. Los modelos de red seleccionados para estudio y comparación fueron: Xception, VGG16, VGG19, ResNet50, InceptionV3 y MobileNetV2. Para ello, se decidió someter a estos modelos a entrenamientos con pocas épocas y datos. Con este objetivo en mente, se escogieron 10.000 imágenes correspondientes a 10 categorías del dataset Food-101. Se realizó un proceso de redimensión de las imágenes a un tamaño de 224x224 con una división de entrenamiento con un 75 % de las imágenes y de validación del 25 %. El tamaño de lote, es decir, el número de imágenes que iba a procesar la red neuronal fue 32. En cada una de estas pruebas no se incluye la última capa del modelo, hay que tener en cuenta que estas arquitecturas de red fueron utilizadas para clasificar 1.000 categorías distintas, por lo que fue necesario cambiar como mínimo la última capa para especificar el nuevo número de categorías a clasificar. Se utilizaron los pesos resultantes de la evaluación de las arquitecturas de red con el conjunto de datos ImageNet y se congelaron todas las capas del modelo, evitando así que pudiesen ser entrenadas.

Con toda la información y técnicas identificas y estudiadas, se diseño un modelo de red con las siguientes características:

1. El conjunto de capas de la red predefinida (Xception, VGG16, ...).
2. Una capa Dropout con rate=0,2, es decir, se desconectan un 20 % de las neuronas.

3. Una capa Dense con 512 unidades y la función de activación relu.
4. Una capa BatchNormalization para normalizar y descentralizar la información de las capas anteriores.
5. Otra capa Dropout con un rate=0,1.
6. Otra capa Dense con 256 unidades junto a la función de activación relu.
7. Otra capa BatchNormalization.
8. Otra capa Dropout con rate=0,1.
9. Una capa Dense con 10 unidades que corresponden al número de categorías posibles junto a la función de activación softmax.

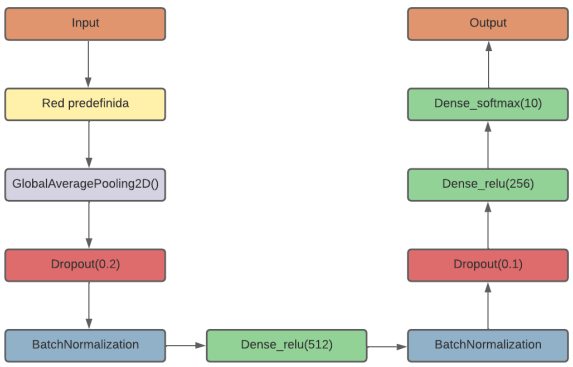


Fig. 4: Arquitectura de red

Cada una de las redes neuronales convolucionales diseñadas a partir de una red ya existente fue entrenada durante 5 épocas, un número relativamente bajo, pero se realizó con el objetivo de descartar rápidamente aquellas arquitecturas de red que no obtuviesen buenos resultados en un corto período de entrenamiento y tiempo, ya que el tiempo medio de entrenamiento de cada red tuvo una duración de 5 minutos aproximadamente. De manera adicional, se utilizó una función de parada temprana que monitorizaba la precisión de validación obtenida durante el entrenamiento con una paciencia de 2, esto se traduce en que si la red no obtenía una precisión de validación superior tras 2 épocas, se detenía el entrenamiento. Además, se incluyó la recuperación de los mejores pesos obtenidos durante la fase de entrenamiento, sin importar la época en la que se alcanzaron.

Un factor determinante para obtener buenos resultados durante el entrenamiento y validación de una red neuronal convolucional, es el uso de un optimizador adecuado en función de las características del conjunto de datos y los parámetros de entrenamiento. En el caso de estos experimentos iniciales, se ha empleado Adam (Adaptative Moment Estimation), el algoritmo de optimización más utilizado en Deep Learning [7], el cual posee las siguientes ventajas:

- Resulta fácil de implementar.
- Es computacionalmente eficiente y requiere de pocos requisitos de memoria.
- Es adecuado para la resolución de problemas compuestos por gran cantidad de datos y/o parámetros.
- Por lo general, no es necesario cambiar los hiperparámetros de la función de optimización porque suelen tener una interpretación intuitiva.

De forma resumida, los parámetros configurados para los entrenamientos corresponden a los de la siguiente tabla:

Tabla I: Parámetros experimentales.

Parámetro	**Valor**
Optimizador	Adam
Tamaño de lote	32
Tasa de aprendizaje	0,001
Número de épocas	5

Los resultados obtenidos tras el entrenamiento de cada red fueron analizados detenidamente con el objetivo de descartar aquellas redes con un aprendizaje lento, estas redes no obtuvieron valores de precisión de validación superiores al 20 %. Estas redes en concreto, son las que utilizaron las arquitecturas de red predefinidas Xception, InceptionV3 y MobilenetV2. Esto no quiere decir que los modelos de red señalados sean poco óptimos para la resolución de problemas de Aprendizaje Profundo, si no que no obtienen buenos resultados con los parámetros y conjuntos de datos propuestos. Por otro lado, las redes neuronales convolucionales desarrolladas a partir de VGG16, VGG19 y ResNet50 obtuvieron mejores resultados.

Tabla II: Resultados de modelos de red.

Red	**Prec. Entr.**	**Prec. Val.**	**Pérd. Val.**
VGG16	93,7 %	75,8 %	1,035
VGG19	95,9 %	74,9 %	0,972
ResNet50	95,8 %	67,1 %	1,571

Los resultados varían en función de la arquitectura de red entrenada. El valor referente a la precisión de entrenamiento indica que no existe una alta diferencia entre los tres modelos de red, esto es debido a las características y parámetros usados. Por otro lado, el valor de precisión de validación sí que presenta diferencias significativas entre las dos redes VGG y ResNet50, llegando a haber una diferencia de un 8,7 % de precisión entre el modelo basado en VGG16 y ResNet50. Respecto a la pérdida de validación ocurre lo mismo, una alta disimilitud entre las arquitecturas de red fundamentadas en cualquiera de las dos versiones VGG y ResNet50. La arquitectura de red elegida para el experimento final de clasificación de imágenes de platos de alimentos fue VGG16, porque a pesar de presentar un número significativamente mayor de parámetros que el modelo justificado en la arquitectura ResNet50, es decir, un mayor coste computacional y temporal de entrenamiento, ha obtenido una precisión de validación notablemente superior. Además, ha obtenido una precisión de validación superior al modelo apoyado en VGG19 que posee un número superior de parámetros.

B. Experimento final

Como ya se ha mencionado anteriormente, la arquitectura de red en la que se fundamentó la red neu-

ronal es en VGG16. En este experimento, el número de categorías es 20 (el doble que en los experimentos anteriores), con el objetivo de aumentar el número de posibilidades y abarcar más tipos de platos. Los platos seleccionados son: apple_pie, caesar_salad, cheesecake, chicken_curry, churros, donuts, escargots, fish_and_chips, french_fries, greek_salad, hamburger, ice_cream, macarons, omelette, paella, pizza, ramen, spring_rolls, sushi y tacos.

Para ello se utilizaron 20.000 imágenes de 20 categorías distintas del dataset Food-101. Las imágenes fueron divididas en tres grupos: entrenamiento, validación y evaluación. El grupo de entrenamiento contenía el 72 % de las imágenes (14.400), el grupo de validación el 18 % (3.600) y el grupo de evaluación un 10 % (2.000). Sobre las imágenes de entrenamiento se aplicó aumento de datos, esta técnica resulta de gran utilidad para conjuntos de datos con pocas muestras o para la simulación de la captura de una imagen en determinadas condiciones. Un caso de uso de esta técnica es el cambio en la rotación de una imagen, ya que es posible que al capturar una imagen de un plato, la cámara este rotada, de esta manera la red se entrena con una mayor cantidad de datos diferentes de una misma categoría, aumentando así la capacidad de predicción de ésta.

Los filtros aplicados para el aumento de datos seleccionados son los siguientes:

- Rotación de la imagen en un rango de -90 y 90 grados.
- Generación de brillo en la imagen entre 0,1 y 0,7.
- Desplazar la imagen hacia la derecha o hacia la izquierda un 50 %.
- Desplazar la imagen hacia arriba o abajo un 50 %.
- Invertir los píxeles de las filas de la imagen.
- Invertir los píxeles de las columnas de la imagen.

En la siguiente figura se puede observar el resultado de aplicar la técnica de aumento de datos con los filtros antes mencionados a una imagen:

Fig. 5: Aumento de datos

El tamaño de las imágenes se estableció en 224x224, con un tamaño de lote de 32 y se mezclaron las imágenes, de forma que el orden cambie para que la red no reciba todas las imágenes de una misma categoría de manera seguida en el conjunto de entrenamiento y validación. Para el desarrollo de este experimento, se parte de los parámetros y resultados obtenidos en otros experimentos que también emplean como base el modelo de red VGG16. Los experimentos en los que se fundamenta la arquitectura de red desarrollada, sus características y parámetros, son los siguientes:

- Experimento 1: en este experimento, se realiza la clasificación de imágenes sobre las 101 categorías de platos del dataset Food-101, con un total de 101.000 imágenes, según una división de entrenamiento y evaluación del 75 % y 25 % respectivamente. Las imágenes fueron redimensionadas a un tamaño de 224x224 píxeles. El tamaño de lote elegido fue de 10, con un valor de 10 épocas durante el entrenamiento. El optimizador escogido para el experimento fue SGDM, con una tasa de aprendizaje del 0,001. Los resultados obtenidos durante este experimento, corresponden a un valor de precisión durante el proceso de evaluación de un 79,86 % [8].
- Experimento 2: en este experimento, se realiza la clasificación de imágenes sobre 10 categorías de platos del dataset Food-101, con un total de 10.000 imágenes según una división de entrenamiento, validación y evaluación del 63,8 %, 1,12 % y 25 % respectivamente. Las imágenes fueron redimensionadas a un tamaño de 224x224 píxeles. El tamaño de lote elegido fue de 64, con un valor de 50 épocas durante el entrenamiento. El optimizador elegido para el experimento fue Adam con una tasa de aprendizaje del 0,001. Además, se aplica el método de Ajuste Fino en el modelo, es decir, se congelan la mayor parte de las capas preentrenadas del modelo VGG16 y solamente se descongelan las últimas capas del modelo, que en este caso fueron 2, para una mejor adaptación del modelo al problema de clasificación de imágenes de platos. Los resultados obtenidos durante este experimento corresponden a un valor de precisión durante el proceso de evaluación de un 81,56 % [9].
- Experimento 3: en este experimento, se realiza la clasificación de imágenes sobre 10 categorías de platos del dataset Food-101, con un total de 5000 imágenes, según una división de entrenamiento y validación del 70 % y 30 % respectivamente. Las imágenes fueron redimensionadas a un tamaño de 224x224 píxeles. El tamaño de lote elegido fue de 64 con un valor de 408 épocas durante el entrenamiento. El optimizador seleccionado para el experimento fue SGDM con una tasa de aprendizaje del 0,001. Los resultados obtenidos durante este experimento corresponden a un valor de precisión durante el proceso de validación de un 85,07 % [10].

En la siguiente tabla, se pueden observar las características principales de cada experimento:

Tabla III: Experimentos de otras fuentes.

Exp.	Optim.	Época	Tam. Lot.	Prec.
1	SGDM	10	10	79,86 %
2	Adam	50	64	81,56 %
3	SGDM	408	64	85,07 %

El conjunto de experimentos formado por experimentos analizados de otras fuentes y los experimentos previos, detallados en el apartado anterior, sirvieron como base para la creación del experimento final para la resolución del problema de clasificación de imágenes. Los parámetros utilizados se obtuvieron a partir de los experimentos anteriores:

- Del primer experimento, se recoge el uso del optimizador SGDM, a diferencia de los experimentos previos en los cuales se había utilizado Adam como función de optimización. SGDM (Sthocastic Gradient Descent with Momentum) es una función de optimización que acelera los vectores de gradientes en las direcciones correctas, lo que se traduce en una convergencia más rápida. El impulso es un promedio móvil de los gradientes que es utilizado para actualizar los pesos de la red. Como resultado, se generan promedios ponderados exponencialmente que facilita la obtención de mejores resultados al proporcionar una estimación alejada de los datos ruidosos durante el entrenamiento [11]. Un estudio que compara las funciones de optimización más utilizadas para la clasificación multicategoría Adam y SGDM, indica que Adam obtiene unos mejores resultados durante el entrenamiento con pocas épocas, pero según éstas se van incrementando, SGDM obtiene mejores resultados de precisión [12].
- Del segundo experimento, se utilizaron los filtros aplicados a la técnica de aumento de datos, ya que el número de filtros era mayor que en los otros dos experimentos. Un número mayor de filtros en la aplicación de aumento de datos favorece la obtención de mejores resultados, pero hay que tener en cuenta que es un proceso costoso computacionalmente que puede alargar la fase de entrenamiento y validación. También se recoge el número de épocas equivalente a 50, en los otros dos experimentos el número de épocas era 10 y 408. Un número de épocas bajo suele ser empleado en conjuntos de datos escasos y con pocas categorías a clasificar, en caso contrario y dependiendo del modelo de red propuesto, los resultados de precisión obtenidos serán bajos comparados con un incremento del número de épocas en el mismo modelo de red. Por otro lado, el incremento del número de épocas es recomendable hasta cierto punto en función de la arquitectura de red, el conjunto de datos y sus características, ya que traspasado ese límite, se estará consumiendo tiempo y recursos sin obtener mejores resultados. Por último, se imita el método de descongelación para las últimas tres capas para adaptar el modelo de red a la resolución del problema de clasificación de imágenes de platos.
- Del tercer experimento, además del uso de la función de optimización SGDM, se recogen las capas de agrupación para reducir la dimensionalidad entre capas y así acelerar el entrenamiento, y la capa dropout para mejorar los resultados del entrenamiento tras cada época.

Tabla IV: Parámetros del experimento final.

Parámetro	Valor
Optimizador	SGDM
Tasa de aprendizaje	0,001
Impulso	0,9
Tamaño de lote	32
Épocas	50
Capas descongeladas	3

Tras determinar todas las características del experimento, se inició el proceso de entrenamiento, validación y evaluación, que posteriormente fueron almacenados y guardados con el fin de analizar y preservar los resultados obtenidos.

IV. Resultados

A. Resultados experimentales

La duración total del experimento fue de 4 horas teniendo en cuenta las características del sistema en el que se realizó el experimento, las cuales son:

- Procesador Intel Core i7 2.20 GHz.
- Tarjeta gráfica NVIDIA GeForce RTX 2060.
- Memoria de acceso aleatorio (RAM) de 16 GB.
- Sistema Operativo Windows 10 con arquitectura de 64 bits.

Los resultados obtenidos son los representados a continuación:

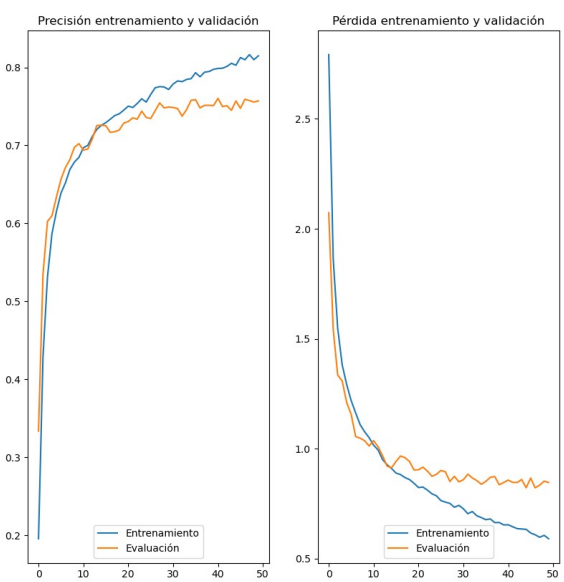


Fig. 6: Resultados del experimento final

Los resultados analizados corresponden al valor de precisión de cada división: entrenamiento, validación

y evaluación. Los resultados se analizan en función de las métricas Top 1 y Top 5. Top 1 es el valor obtenido a partir de la comparación de la respuesta de la red y el valor real. Top 5 corresponde a la comparación de las 5 mejores probabilidades obtenidas y el valor real, si alguna de las categorías cuya probabilidad se encuentra en el Top 5 corresponde al valor real o esperado, se refleja en la precisión Top 5 del modelo.

Tabla V: Resultados Top 1 y Top 5.

Top	Prec. Entr.	Prec. Val.	Prec. Eval.
1	81,63 %	76,02 %	84,38 %
5	97,19 %	94,75 %	97,27 %

Respecto al valor de pérdida, el modelo obtuvo un resultado durante el proceso de validación de 0,8214, un dato situado entre el valor de pérdida del Experimento 2 (0,907) y el Experimento 3 (0,7435).

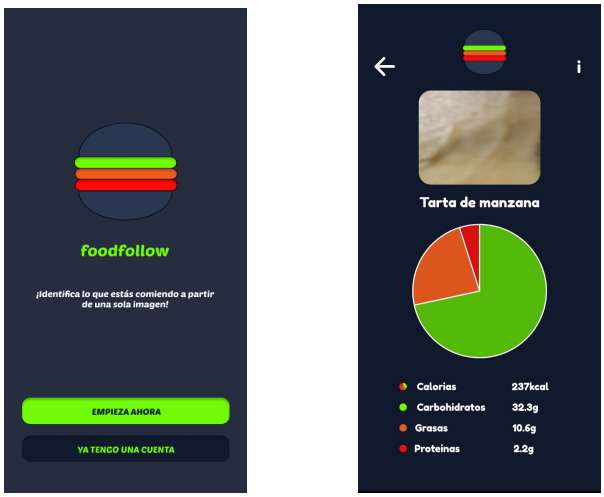
Por último, cabe destacar que se ha obtenido un modelo de red cuyos resultados son capaces de competir con los de los experimentos realizados por otras fuentes. Además, el código generado para realizar el experimento final se encuentra alojado en un repositorio de GitHub para que cualquier persona interesada en la investigación pueda acceder a través del siguiente enlace: `Código del experimento`.

B. Transformación y despliegue de la red

Para que los usuarios pudiesen probar el funcionamiento de la red neuronal convolucional obtenida, se diseñó y creó una aplicación móvil a partir del uso de tecnologías ampliamente conocidas en el desarrollo web. Un paso importante en la implementación de la red en la aplicación, fue el de transformación de formatos.

Para poder utilizar el modelo de predicción, fue preciso convertirlo a un formato compatible con las herramientas que se iban a utilizar. El formato en el que se guardó el modelo desarrollado con el lenguaje de programación Python corresponde a un archivo con extensión .h5, un formato poco amigable con un entorno de desarrollo basado en tecnologías web. El formato elegido para poder utilizar el modelo en la aplicación fue JSON, esta conversión se pudo realizar gracias a la librería tensorlflowjs, que posee una función para convertir un modelo .h5 en un modelo JSON, con las capas que lo componen almacenadas en formato .bin.

De esta forma, el modelo de red convolucional se ejecuta y funciona directamente en el dispositivo móvil del usuario, sin necesidad de un servidor con características hardware elevadas para procesar las imágenes que se capturan y con un tiempo de predicción una vez que el modelo está cargado en memoria de aproximadamente solo 2 segundos. A continuación, se muestran algunas de las interfaces de la aplicación móvil:



(a) Página de Inicio (b) Página de Plato

Fig. 7: Interfaces de la aplicación móvil

V. Conclusiones

En conclusión, este estudio ha proporcionado una visión significativa sobre el uso de redes neuronales convolucionales para la clasificación de imágenes de platos y ha revelado varios hallazgos importantes. A través del análisis exhaustivo de los datos y la aplicación rigurosa de los métodos, se ha llegado a las siguientes conclusiones clave:

1. Las redes neuronales convolucionales son un método excelente para resolver tareas de clasificación de imágenes, aunque resulta complicado dar con el modelo perfecto. También cabe destacar que los resultados de predicción dependen de muchos factores, como funciones de optimización, número de capas, técnicas y/o capas para la mejora del rendimiento y resultados, aumento de datos e incluso, la naturaleza de los datos de entrada.
2. El presente artículo aporta una sólida contribución al problema de clasificación de platos, mostrando distintas técnicas que pueden ser utilizadas para mejorar factores como el rendimiento durante el entrenamiento y la reducción de parámetros de la red para reducir su dimensión. Se espera, que este experimento sirva como base de futuras investigaciones y experimentos, que consigan obtener mejores resultados y una arquitectura de red con cada vez menos parámetros, con el objetivo de que pueda estar presente en el máximo número de sistemas embebidos posibles sin importar sus características hardware.

Agradecimientos

Este trabajo fue apoyado por la Agencia Estatal de Investigación (AEI) de España bajo la subvención PID2020-119144RB-I00 financiada por MCIN/AEI/10.13039/501100011033.

Referencias

[1] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool, "Food-101 – mining discriminative components with random forests," in *European Conference on Computer Vision*, 2014.

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.
[3] Min Lin, Qiang Chen, and Shuicheng Yan, "Network in network," 2014.
[4] Pierre Baldi and Peter Sadowski, "Understanding dropout," 2013.
[5] Alireza M. Javid, Sandipan Das, Mikael Skoglund, and Saikat Chatterjee, "A relu dense layer to improve the performance of neural networks," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2021-June, pp. 2810–2814, 2021.
[6] Vignesh Thakkar, Suman Tewary, and Chandan Chakraborty, "Batch normalization in convolutional neural networks - a comparative study with cifar-10 data," *Proceedings of 5th International Conference on Emerging Applications of Information Technology, EAIT 2018*, 9 2018.
[7] Diederik P. Kingma and Jimmy Lei Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 12 2014.
[8] Abdulkadir Sengur, Yaman Akbulut, and Umit Budak, "Food image classification with deep features," *2019 International Conference on Artificial Intelligence and Data Processing Symposium, IDAP 2019*, 9 2019.
[9] James McDermott, "Hands-on Transfer Learning with Keras and the VGG16 Model - LearnDataSci," 2021.
[10] Sapna Yadav, Alpana, and Satish Chand, "Automated food image classification using deep learning approach," *2021 7th International Conference on Advanced Computing and Communication Systems, ICACCS 2021*, pp. 542–545, 3 2021.
[11] Sebastian Ruder, "An overview of gradient descent optimization algorithms," 9 2016.
[12] Pujo Hari Saputro, Dhina Puspasari Wijaya, Musthofa Galih Pradana, Dyah Listianing Tyas, and Wahyuni Fithratul Zalmi, "Comparison adam-optimizer and sgdm for classification images of rice leaf disease," *Proceedings - 4th International Conference on Informatics, Multimedia, Cyber and Information System, ICIMCIS 2022*, pp. 348–353, 2022.

Reconocimiento facial y de voz en sistemas empotrados de bajo coste mediante el uso de TinyML

José Miguel Moreno[1], José Antonio de la Torre[1], Fernando Rincón[1], Julián Caba[1], José Luis Mira[1], Juan Carlos López[11]

Resumen— **Tradicionalmente los dispositivos embebidos han estado limitados a tareas de recolección y filtrado de datos, mientras que el procesamiento de estos datos se ha delegado en servidores con mayor capacidad de procesamiento en la nube. Sin embargo, con los últimos avances tecnológicos han surgido nuevos paradigmas como la computación en el Edge y en el Fog, acercando la capacidad de cómputo al dispositivo final. En este contexto, surge el concepto de TinyML, que busca implementar algoritmos de aprendizaje por computador directamente en el dispositivo final.**

En este trabajo, se evalúa la viabilidad del uso de técnicas avanzadas de aprendizaje por computador, como redes neuronales, en microcontroladores de bajo consumo. Específicamente, se ha implementado un sistema de reconocimiento facial y de voz para identificar a un usuario y gestionar su acceso a un área restringida. Se presenta la arquitectura de la solución implementada, logrando ejecutar ambos modelos dentro de dispositivo ARM Cortex M4 a 64 MHz, con una precisión del 84.4 % en el reconocimiento facial y del 90 % en el reconocimiento de voz.

Palabras clave— **TinyML, Deep Learning, Neural Networks, MobileNet, MFCC, Arduino, DSP, Low power**

I. Introducción

El término TinyML (Tiny Machine Learning) hace referencia a un concepto relativamente nuevo que involucra los sistemas embebidos y el aprendizaje por computador [1]. El TinyML traslada las técnicas utilizadas en aprendizaje por computador o *machine learning* en dispositivos de bajo consumo (en el orden de magnitud de milivatios) y de bajo coste. Normalmente, estos dispositivos se encuentran limitados en el uso de recursos, tanto a nivel de procesamiento como de memoria RAM y Flash.

Hasta ahora, el uso de machine learning en dispositivos embebidos se ha realizado a través de los despliegues IoT (Internet of Things) aprovechando las capacidades de comunicación de este tipo de dispositivos con el Cloud. Sin embargo, según el número de dispositivos en los despliegues ha incrementado, las técnicas tradicionales para la descarga de trabajo en servidores en el Cloud se han visto comprometidas, principalmente por la sobrecarga impuesta por la comunicación. Para solucionar estos inconvenientes se han propuesto otras técnicas como la computación en el Edge y en el Fog [2]. Este tipo de técnicas consiste en virtualizar las tareas realizadas típicamente en el Cloud y acercarlas a los dispositivos finales, reduciendo la latencia y mejorando la privacidad. Sin embargo, los dispositivos en el Edge, son dispositivos que normalmente están conectados a una fuente de energía y tienen altas capacidades de cómputo en comparación a los dispositivos embebidos finales.

El último paso en esta transición, resumida en la Figura 1, es ejecutar estos algoritmos en los dispositivos finales.

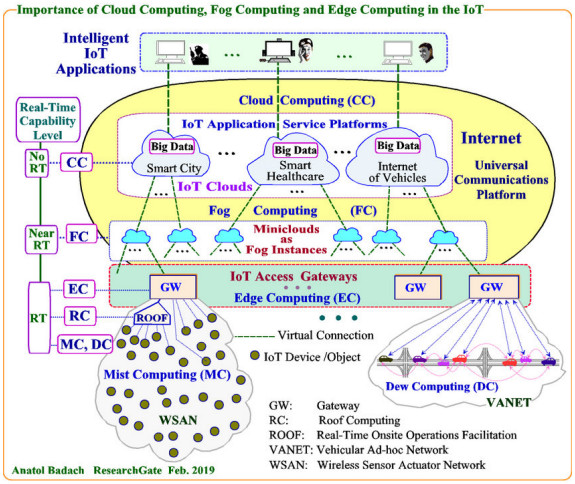


Fig. 1: Arquitectura Computing Continuum [3].

Tal y como se refleja en el estudio sistemático realizado por Hui Han y Julien Siebert [4], en el que se analizan los trabajos más relevantes en el área del TinyML, los casos de uso de TinyML van desde la detección de palabras clave hasta el reconocimiento de actividades deportivas (ver Figura 2). Uno de las primeras aplicaciones y, que más relevancia científica ha acaparado en los últimos años, es el de la detección de palabras claves. Muchos dispositivos inteligentes de grandes compañías como Google, Amazon o Microsoft utilizan este tipo de palabras para activar sus dispositivos y realizar las tareas para las que han sido diseñados. Tradicionalmente este tipo detección se ha realizado mediante una combinación de hardware especializado para la detección de palabras específicas y más tarde, una vez realizada la detección, el reconocimiento final en servidores en el Cloud. En los últimos años estas compañías han ido presentando nuevos modelos que permiten la inferencia total en local sin necesidad de servidores en el Cloud.

En este trabajo se presenta una implementación de un caso de uso para sistemas de seguridad mediante el uso de la voz y el reconocimiento facial. Se

[1]Dpto. de Tecnologías y Sistemas de Información, UCLM, 13071 Ciudad Real, España

pretende validar la viabilidad de este tipo de técnicas de TinyML para escenarios más complejos mediante la combinación de diversos modelos de aprendizaje dentro de un mismo dispositivo. El dispositivo embebido hace uso de las técnicas de TinyML para la detección inicial de personas mediante el reconocimiento facial para más tarde detectar e identificar una frase clave que permita la apertura de una puerta de seguridad. Finalmente, además de reconocer la frase se identifica a la persona usando la voz y su huella vocal.

II. Caso de uso

El objetivo principal del trabajo es validar la viabilidad de las técnicas de TinyML para el análisis directamente en el dispositivo final. El caso de uso plantea un reto debido al proesamiento de flujos de datos de imagen y audio dentro de un dispositivo embebido. En concreto se busca identificar tanto por voz como por imagen a una persona para su autorización de acceso a un área restringida. Este mismo caso de uso se podría usar en una cerradura inteligente.

El trabajo por tanto se ha dividido en tres 4 fases:

- Fase 1: Exploración del espacio de diseño: Se identifican las principales técnicas y herramientas para llevar a cabo el estudio.
- Fase 2: Implementación de la identificación facial
- Fase 3: Implementación de la identificación vocal
- Fase 4: Integración final: Despliegue en el microcontrolador

III. Desarrollo

A. Exploración del espacio de diseño

El campo del TinyML es un campo relativamente nuevo y las herramientas para el diseño e implementación todavía son escasas. En el trabajo [1] se realiza un análisis de las principales plataformas hardware y software utilizadas hasta la fecha. En esta revisión se concluye que la mayoría del hardware está basado en la arquitectura ARM. Esto se debe principalmente a las ventajas que ofrece este conjunto de instrucciones a la hora de realizar cómputo con un bajo coste energético [5]. En los últimos años ha aparecido un nuevo competidor a este conjunto de instrucciones que además de estas capacidades tiene la peculiaridad de ser de código abierto y sin coste por licencia, RISC-V. El uso de este nuevo conjunto de instrucciones todavía se encuentra enfocado en el entorno académico debido en parte a la falta de adopción por las compañías, aunque en los últimos años están apareciendo algunas como SiFive. En [6] se puede ver una implementación de un núcleo de bajo consumo basado en RISC-V aunque la implementación es experimental dentro de una FPGA (Field Programable Gate Array).

Basado en estos datos y la compatibilidad con el software que más adelante detallaremos, en este trabajo se ha usado la arquitectura ARM mediante el SoC (System on Chip) nRF52840 de la compañía Nordic. Este SoC está basado en el ARM Cortex-M4 y posee una gran cantidad de periféricos y sistemas de comunicación que lo hacen ideal para su uso en entornos IoT. Como placa de desarrollo hemos elegido el Arduino Nano 33 BLE Sense ya que incluye varios periféricos como el micrófono y el sensor de proximidad y está oficialmente soportado por la plataforma software que utilizaremos.

A la hora de elegir el hardware resulta indispensable tener en cuenta las tecnologías software que se van a usar. En la revisión realizada en [6] y [7] se realiza un estudio sobre las principales plataformas software centradas en TinyML. La principal librería para la implementación de modelos de aprendizaje en sistemas embebidos es Tensorflow Lite, desarrollada por Google y basada en su conocida librería Tensorflow. Esta librería se enfoca en el despliegue de redes neuronales en dispositivos de bajo cómputo principalmente mediante el uso de técnicas de post-procesado como la cuantización [8]. Este proceso permite reducir el uso de energía y el consumo de memoria. La ventaja principal de esta librería frente a otros es la compatibilidad prácticamente total con topologías de redes neuronales existentes únicamente teniendo que realizar el post-procesado final.

Sin embargo, el procesado de la red neuronal es solo uno de los pasos necesarios en las soluciones de machine learning. También es necesario el preprocesado de los datos, conexión con periféricos y comunicaciones. Para simplificar esta labor la empresa Edge Impulse proporciona una suite compatible con una gran variedad de dispositivos y librerías como Tensorflow Lite. En Edge Impulse los diferentes pasos dentro del flujo típico en los proyectos de TinyML se resumen en los mostrados en la Figura 3

- Adquisición de datos: Mediante un protocolo implementado en el módulo *edge-impulse-data-forwarder* se adquieren los datos directamente del dispositivo final o de cualquier dataset público mediante su interfaz web. La propia herramienta cuenta con un previsualizador puidiendo usar modelos preentrenados o mediante técnicas clásicas de reducción de la dimensionalidad como t-SNE ([9]) o PCA ([10]).
- Diseño del impulso: Dentro de Edge Impulse a la fase de procesamiento de datos e inferencia se le llama impulso. Los impulsos están divididos en 3 grandes bloques: bloque de entrada, bloque de procesamiento y bloque de aprendizaje. El bloque de entrada se encarga de adaptar los datos crudos en un conjunto de características adaptados a las necesidades concretas. Por ejemplo, en el caso de imágenes este bloque típicamente se encarga de hacer el redimensionado y de la imagen. El bloque de procesamiento, también llamado bloque DSP (Digital Signal Processing), es el encargado de generar las características de los datos mediante técnicas tradicionales, como la transformada de Fourier, cálculo de coeficientes de mel (MFCC), espectogramas, etc. Estos bloques son una parte importante de cualquier

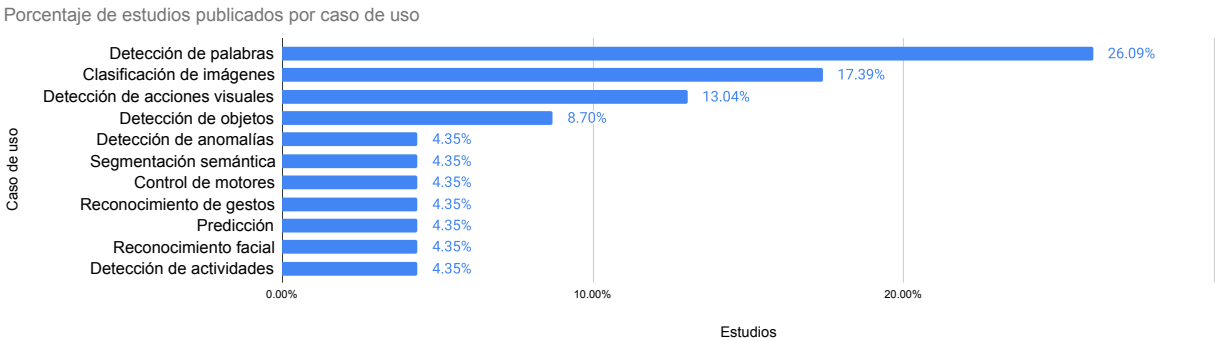


Fig. 2: Porcentaje de casos de uso más repetidos en las técnicas de TinyML [4].

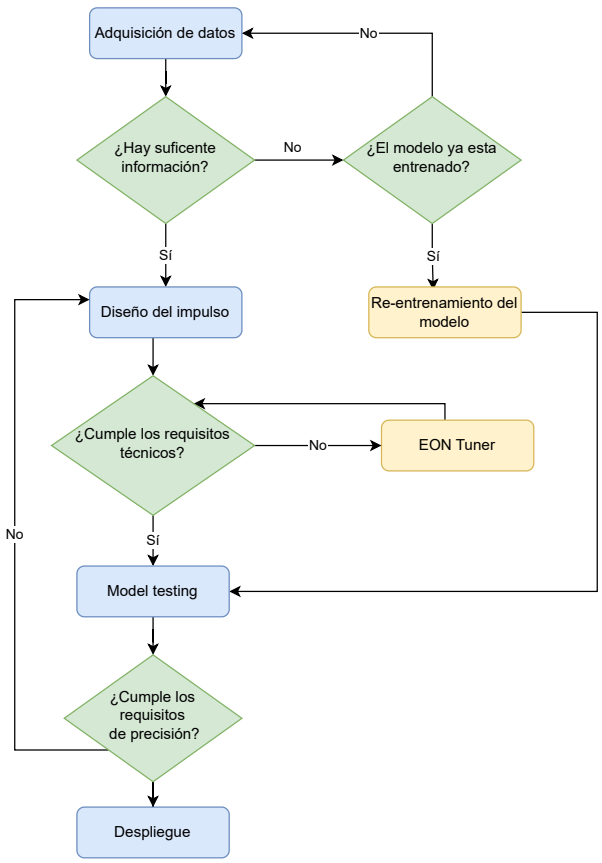


Fig. 3: Diagrama de flujo del proceso de desarrollo en Edge Impulse.

flujo de aprendizaje por computador y, por esta razón, Edge Impulse realiza una implementación adaptada a cada microcontrolador con el objetivo de utilizar las librerías y coprocesadores específicos de cada fabricante. Por último, el bloque de aprendizaje es el encargado de realizar la clasificación, detección, regresión, etc. en función del caso de uso a implementar. Edge Impulse da soporte para bloques de aprendizaje personalizados así como bloques con transferencia de conocimiento para reducir el tiempo de desarrollo.

- Post-Procesado: Tal y como se ha comentado la fase de post-procesado se encarga de cuantizar y adaptar los modelos para los dispositivos embebidos. La herramienta de Edge Impulse, EON Tuner analiza de forma automática los datos de entrada y el impulso diseñado para ofrecerte otras alternativas cambiando alguno de los bloques y cumpliendo con los requisitos de latencia y memoria impuestos por el usuario. Esta herramienta ayuda a reducir el tiempo de desarrollo mediante la reducción del espacio de diseño.
- Despliegue: En la fase de despliegue Edge Impulse proporciona varias alternativas en función del tipo de proyecto que se vaya a realizar. La opción más versátil es la encapsulación del proyecto en una librería C++ para su uso en firmware personalizado.

Debido a la gran versatilidad de la herramienta y su utilización de librerías establecidas dentro del área del machine learning en este proyecto se ha utilizado la herramienta Edge Impulse como software de desarrollo. Tal y como adelantamos, la placa Arduino Nano 33 BLE Sense esta soportada oficialmente por Edge Impulse por lo que se simplifica aun más el desarrollo mediante el uso de las capas HAL (Hardware Abstraction Layers) proporcionadas por Edge Impulse.

B. Implementación de la identificación facial

El reconocimiento facial de la persona autorizada se realiza mediante el pipeline representado en la Figura 4.

En primer lugar se realiza un preprocesado de la imagen a una resolución de 96x96. Con esta reducción conseguimos reducir los requisitos de memoria RAM y el consumo de energía en los bloques DSP posteriores. Además de este preprocesado se ha hecho uso del bloque *Image* que se encarga de normalizar la imagen y adaptar la profundidad de color para el bloque posterior. Este bloque se ha configurado para normalizar los valores RGB en un valor de punto flotante entre 0 y 1 para cada componente del espacio RGB. Esto permite centrar los valores de cada canal para que los bloques posteriores trabajen sobre el mismo espacio.

Tras la configuración del bloque DSP se añade una etapa de aprendizaje mediante el bloque de "transferencia de aprendizaje". La transferencia de aprendizaje permite solucionar problemas de clasificación, en este caso de imágenes, con un dataset pequeño [11]. En este caso Edge Impulse provee de varias redes preentrenadas para diferentes tamaños de imagen. Para

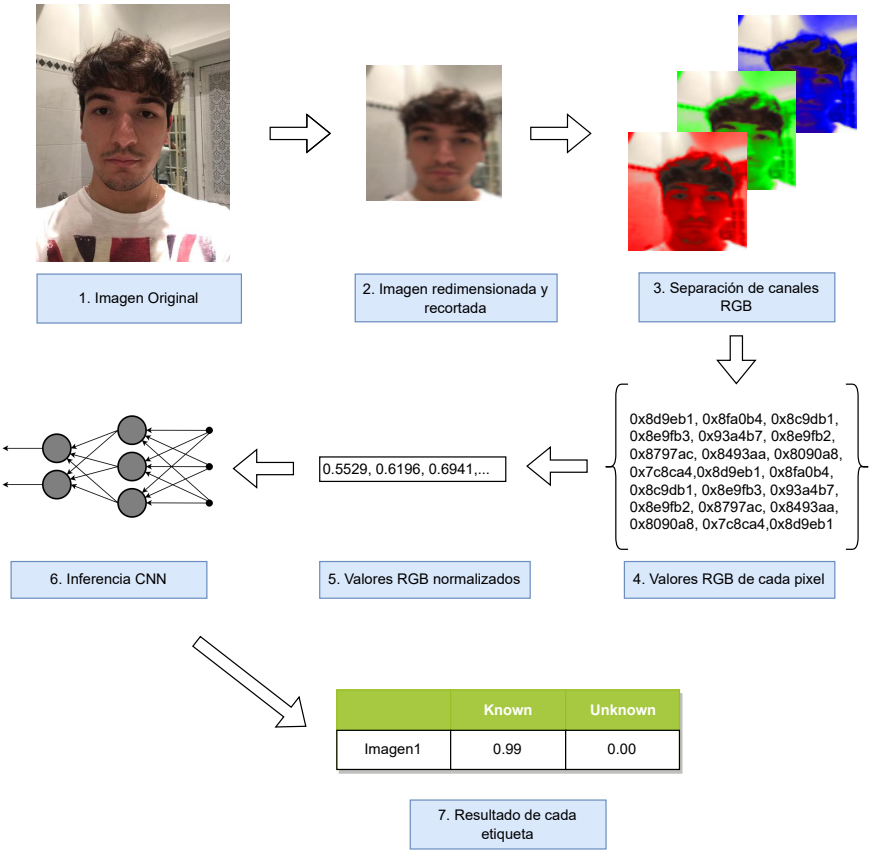


Fig. 4: Pipeline del reconocimiento facial con las diferentes etapas.

este trabajo se ha optado por MobileNetV1 [12]. Sobre esta red se ha añadido varias capas congelando el entrenamiento de las capas intermedias.

Para el entrenamiento se ha utilizado el dataset público de Kaggle [1] de imágenes de tipo *selfie* para las clases de personas desconocidas (no autorizadas). Por otra parte, para la clase de persona autorizada (autorizado) se han utilizado fotografías tomadas en el laboratorio. Finalmente, se han utilizado imágenes de diferentes fondos para proporcionar ejemplos de imágenes sin personas a la red. En la Tabla I se puede ver la matriz de confusión antes de la cuantización.

Finalmente, tras realizar el entrenamiento se ha cuantizado el modelo mediante el uso de enteros de 8 bits usando la herramienta Edge Impulse. Este paso permite reducir los requisitos de memoria y cómputo sin afectar de forma significativa a la precisión. La matriz de confusión tras la cuantización se puede ver en la Tabla II. Por otro lado, en la Tabla III se puede ver como ha variado tanto la precisión como el tiempo de procesamiento y el uso de memoria RAM antes y después de la cuantización. Como se puede observar el ahorro de recursos es sustancial aunque la precisión se ha visto afectada. Este tipo de compromiso entre cómputo y precisión es uno de los puntos fundamentales a la hora de evaluar la viabilidad en el despliegue de modelos de datos en dispositivos embebidos. Edge Impulse mediante EON Tuner simplifica mucho la tarea creando una matriz de configuraciones que resume el espacio de diseño aportando las diferentes estimaciones de métricas (Flash, RAM, CPU) después del postprocesado.

[1] `https://www.kaggle.com/datasets/tapakah68/selfies-id-images-dataset`

Tabla I: Matriz de confusión.

	Fondo	Autorizado	Desconocido
Fondo	100 %	0 %	0 %
Autorizado	0 %	80 %	20 %
Desconocido	3.3 %	5 %	91.7 %
F1	0.99	0.85	0.89

Tabla II: Matriz de confusión tras la cuantización a 8 bits.

	Fondo	Autorizado	Desconocido
Fondo	90.4 %	9.6 %	0 %
Autorizado	0 %	82.5 %	17.5 %
Desconocido	3.3 %	18.3 %	78.3 %
F1	0.94	0.73	0.82

C. *Implementación de la identificación vocal*

Para la identificación vocal se ha implementado el pipeline mostrado en la Figura 5. Como se puede ver, la estructura general es similar a la de la identificación facial, es decir, procesamiento de datos, DSP, aprendizaje y clasificación.

El audio de entrada está preparado para 16000 Hz y se aplica una ventana deslizante de 1 segundo con 1 segundo de salto. Estos valores se pueden adaptar en función del caso de uso y del sonido a detectar, en nuestro caso queremos detectar la palabra

Tabla III: Comparación del consumo de recursos antes y después de la cuantización en el modelo facial.

	Precisión	Tiempo	RAM	Flash
Original	92.5 %	7836 ms	243.3 KB	578.9 KB
Cuantizado	84.4 %	790 ms	100.3 KB	227.4 KB

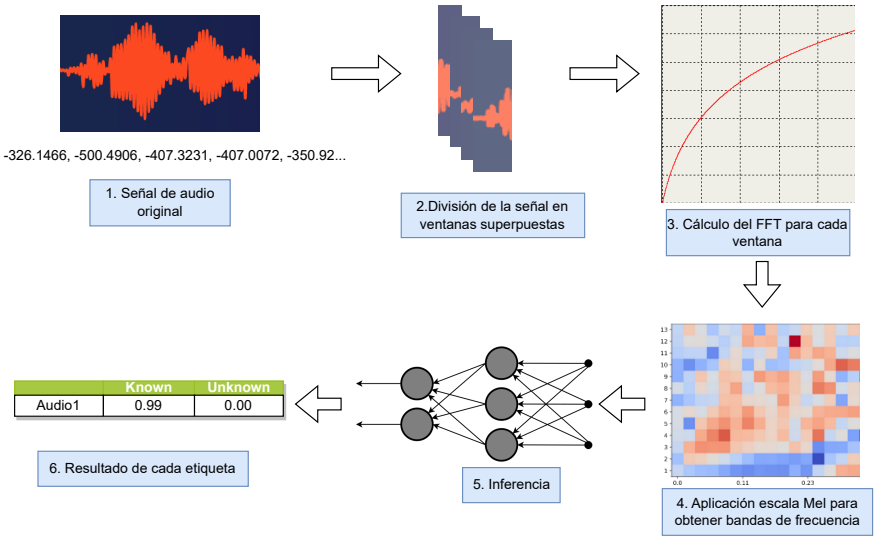


Fig. 5: Pipeline del reconocimiento de voz con las diferentes etapas.

«apertura», de un sujeto autorizado, de otras palabras, otros sujetos o sonido de fondo.

Analizar el audio de manera directa no resulta viable ya que se generan muchas características y el consumo de recursos sería alto sin una mejora en la precisión. Para mejora la clasificación hemos utilizado un bloque DSP llamado MFCC (Mel Frequency Cepstral Coefficients) que consiste en una serie de filtrados y procesamientos que reduce el número de características a clasificar. En la Tabla IV se pueden ver los coeficientes que hemos utilizado para el bloque. Estos coeficientes forman parte de los llamados hiperparámetros que se podrían validar mediante diferentes técnicas [13].

Tabla IV: Coeficientes "Mel Frequency Cepstral".

Coeficientes	13
Longitud	0.02
Stride	0.02
N Filtros	32
Longitud FFT	256
Tamaño ventana	101
Min Freq	0 Hz
Max Freq	8000 Hz
Pre Coef	0.98

Es importante tener en cuenta que en función del número de bloques DSP utilizados el consumo aumentará considerablemente. El bloque MFCC tiene un coste teórico de 294 ms de procesamiento y 13 KB de memoria RAM. La clasificación se ha realizado mediante una red neuronal de 6 capas. El número de entradas a la red es de 650 características que corresponden a los coeficientes calculados por el bloque DSP MFCC. En la Tabla V se puede ver la matriz de confusión una vez entrenado y cuantizado el modelo. Resulta interesante remarcar que el consumo de CPU de ejecutar la red neuronal es de apenas 5 ms y 3.7 KB de RAM en comparación a los 294 ms del bloque DSP. Es por esta razón que Edge Impulse utiliza las librerías del fabricante para el cálculo matemático con el objetivo de optimizar al máximo dichos bloques.

Para el entrenamiento de este pipeline se ha utilizado un dataset generado en el laboratorio con audios de hombres y mujeres. Cada sujeto ha grabado varios audios diciendo diferentes frases algunas con la palabra clave (apertura) y otras sin decir la palabra. Los datos se han dividido en un 78 % para entrenamiento y un 22 % para validación y se ha autobalanceado mediante la opción proporcionada por Edge Impulse. Tras la cuantización se obtiene una precisión del 90.8 %.

D. Despliegue

Una vez entrenados ambos modelos se han desplegado dentro del microcontrolador nRF52840 del Arduino Nano BLE 33 Sense. Para realizar el despliegue Edge Impulse proporciona varios métodos que se pueden dividir en dos grandes categorías.

La primera categoría son los despliegues autocontenidos que Edge Impulse proporciona para sus placas soportadas. En estos casos se proporciona un binario con todo el código listo para ser programado en la placa. Este código viene con un protocolo serie que te permite controlar el impulso y obtener los resultados del mismo. Este tipo de despliegues es útil para realizar medidas o para validar el diseño. Sin embargo, resulta imposible adaptar este firmware a un programa ya existente. Para soportar esta última opción Edge Impulse da la opción de exportar el pro-

Tabla V: Matriz de confusión de la clasificación de audio tras cuantización.

	Autorizado	Ruido	Otro hombre	Otra mujer	Desconocido
Autorizado	90 %	0 %	5 %	0 %	5 %
Ruido	0 %	100 %	0 %	0 %	0 %
Otro hombre	5.6 %	0	83.3 %	5.6 %	5.6 %
Otra mujer	0 %	10 %	0 %	80 %	10 %
Desconocido	0 %	0 %	9.1 %	0 %	90.9
F1	0.92	0.98	0.81	0.84	0.91

yecto como una librería C++ o Arduino en el caso que se desee utilizar dicho framework. En nuestro caso se ha utilizado la librería C++ por la versatilidad.

La librería se divide 3 componentes principales. El primero de los componentes es el SDK de edge-impulse que se mantiene constante entre los diferentes proyectos. Este SDK tiene las APIs de alto nivel y las adaptaciones a cada uno de los microcontroladores. El código es open-source y se puede acceder desde github[2]. Además por cada modelo se definen los metadatos del modelo y los pesos y operaciones de las redes neuronales y bloques DSP. Debido a una limitación a la hora de realizar la autogeneración de las librerías desde la plataforma no se pueden fusionar ambos impulsos en un mismo programa ya que se redefinen símbolos. Esta limitación está impuesta por Edge Impulse y, en caso de usar directamente Tensorflow Lite, evitaríamos este problema.

IV. Resultados

Una de las ventajas principales de las técnicas de TinyML es la reducción en el consumo de energía y reducción en la latencia. En este trabajo nos hemos centrado principalmente en reducir el consumo ya que la latencia no es relevante siempre que se mantenga dentro de un rango razonable.

Para la obtención de resultados y la comparación con las mismas implementaciones mediante soluciones tradicionales se han planteado 3 escenarios diferentes. El primer escenario pretende simular el cómputo en un procesador de alto rendimiento como sería el uso del Cloud mediante la arquitectura x86-64. En el segundo escenario se ha utilizado una placa SBC (Single Board Computer) con las capacidades típicas de un dispositivo en el Edge. Finalmente, el último escenario es el desarrollado en este trabajo que representa el cómputo en el dispositivo final.

Para sistematizar la recogida de resultados se ha alterado ligeramente el firmware para leer los datos de prueba (imágenes y fuentes de audio) desde el almacenamiento externo. También se ha añadido una rutina para monitorizar el tiempo de inferencia y procesamiento. En el caso de los escenarios basados en microprocesador, es decir, el Cloud y el Edge, se ha añadido un monitor que permite monitorizar el porcentaje de uso de CPU y RAM. Para sincronizar las medidas entre el monitor y el proceso de inferencia se hace uso de señales. El conjunto de pruebas consiste en la clasificación de 294 imágenes en diferentes resoluciones y 100 muestras de audio cada una de ellas de un segundo de duración. Tras realizar las pruebas múltiples veces se han obtenido los resultados que se pueden ver en la Tabla VI y en la Figura 6, Figura 7.

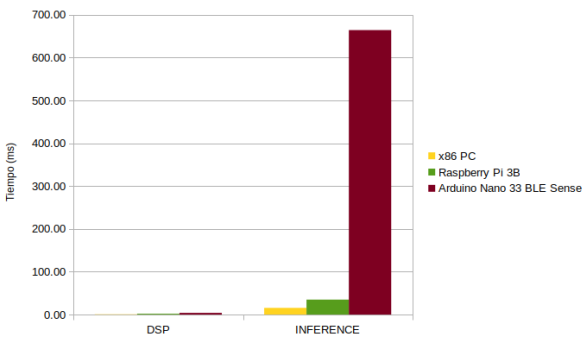


Fig. 6: Tiempo de inferencia y de procesamiento DSP en cada plataforma para el impulso de reconocimiento facial.

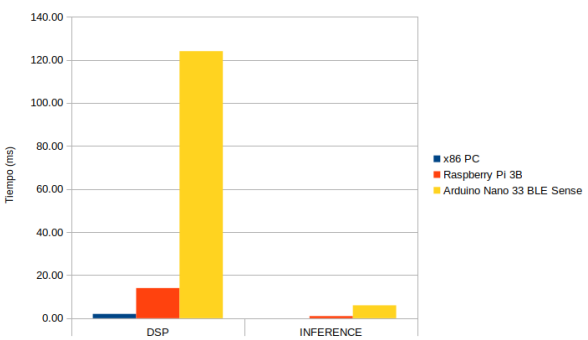


Fig. 7: Tiempo de inferencia y de procesamiento DSP en cada plataforma para el impulso de reconocimiento vocal.

Como se puede observar los tiempos tanto de DSP como sobretodo de inferencia son mucho mayores entre la implementación en x86 y en el microcontrolador. Sin embargo, el uso de RAM es mayor en el primer caso. Además se puede observar un uso de CPU del 90.01 % en el caso del x86 lo cual hace que el consumo energético sea considerablemente superior al caso del microcontrolador.

Por último, en este estudio no se ha tenido en cuenta el consumo de la transferencia de datos en comunicaciones que se debería realizar en el caso de delegar el procesamiento del microcontrolador al Cloud. Este cálculo, supone uno de los factores principales a la hora de valorar la descarga del cómputo entre los diferentes niveles Cloud, Edge, Fog [14]. En el caso de incluirlas se vería favorecido, aun más, el despliegue en el dispositivo final.

[2] `https://github.com/edgeimpulse/inferencing-sdk-cpp`

Tabla VI: Resultados de las pruebas sintéticas.

		x86	Raspberry Pi 4	Arduino
Imagen	DSP	0.67 ms	2.00 ms	4.00 ms
	Inferencia	15.67 ms	35.00 ms	664 ms
	CPU	90.01 %	100 %	100 %
	RAM	20.8 MB	16.02MB	100.3 KB
Audio	DSP	2.00 ms	14.00	124.00 ms
	Inferencia	0 ms	1.00 ms	6.00 ms
	CPU	2.13 %	6.82 %	100 %
	RAM	12.53 %	6.15 MB	13 KB

V. Conclusiones

En este trabajo se ha validado la efectividad de las técnicas de TinyML para desplegar algoritmos complejos de aprendizaje por computador en dispositivos de bajo coste. Se ha obtenido una precisión del 84.4 % en la tarea de reconocimiento facial y un 90 % en el reconocimiento de voz. Además se ha conseguido reducir el consumo de RAM y Flash lo suficiente para incluir ambos modelos en un mismo dispositivo. Se puede concluir por tanto que la tecnología está lo suficientemente madura para realizar modelos los suficientemente precisos dentro de un dispositivo embebido. La limitación principal de estas soluciones está en el uso de RAM y, aunque las técnicas de cuantización han mejorado lo suficiente para que la relación precision/energía sea cada vez mayor los dispositivos embebidos cuentan con una cantidad muy limitada de recursos. En base a los datos obtenidos y los resultados experimentales creemos que la combinación de estas técnicas con el cómputo cercano en el Edge pueden suponer la mejor alternativa. El primer filtrado se puede realizar directamente en el dispositivo final para enviar únicamente los casos en los que no se tenga la suficiente certeza a los dispositivos más cercanos en el Edge.

Agradecimientos

Esta investigación está parcialmente financiada por el Ministerio de Economía y Competitividad (MINECO) del Gobierno de España a través de los proyectos TALENT (PID2020-116417RB-C4, subproyectos 1 y 4) y MIRATAR (TED2021-132149BC41) y por el programa Europeo Horizonte 2020 bajo el proyecto SHAPES (GA Nº 857159).

Referencias

[1] Taiwo Samuel Ajani, Agbotiname Lucky Imoize, and Aderemi A. Atayero, "An overview of machine learning within embedded and mobile devices–optimizations and applications," *Sensors*, vol. 21, no. 13, 2021.

[2] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, New York, NY, USA, 2012, MCC '12, p. 13–16, Association for Computing Machinery.

[3] Anatol Badach, *IIoT – Intelligent IoT*, p. 23, WEKA Media, 03 2019.

[4] Hui Han and Julien Siebert, "TinyML: A Systematic Review and Synthesis of Existing Research," in *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Jeju Island, Korea, Republic of, Feb. 2022, pp. 269–274, IEEE.

[5] Teodor Neagoe, Ernest Karjala, and Logica Banica, "Why arm processors are the best choice for embedded low-power applications?," in *2010 IEEE 16th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 2010, pp. 253–258.

[6] Satyajit Bora and Roy Paily, "A high-performance core micro-architecture based on risc-v isa for low power applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 6, pp. 2132–2136, 2021.

[7] Muhammad Shafique, Theocharis Theocharides, Vijay Janapa Reddy, and Boris Murmann, "Tinyml: Current progress, research challenges, and future roadmap," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1303–1306.

[8] Ioan Lucan Orăşan, Ciprian Seiculescu, and Cătălin Daniel Caleanu, "Benchmarking tensorflow lite quantization algorithms for deep neural networks," in *2022 IEEE 16th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 2022, pp. 000221–000226.

[9] Laurens Van Der Maaten and Geoffrey Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579 – 2625, 2008, Cited by: 23201.

[10] Hervé Abdi and Lynne J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433 – 459, 2010, Cited by: 5880.

[11] Sinno Jialin Pan and Qiang Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345 – 1359, 2010, Cited by: 13539.

[12] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[13] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng, "Hyperparameter optimization for machine learning models based on bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26 – 40, 2019, Cited by: 470.

[14] Carlos Pereira, António Pinto, Duarte Ferreira, and Ana Aguiar, "Experimental characterization of mobile iot application latency," *IEEE Internet of Things Journal*, vol. 4, no. 4, pp. 1082 – 1094, 2017, Cited by: 30.

Indice de autores