*Manuscript
Click here to view linked References

Performance Analysis of HPC Applications in the Cloud

Roberto R. Expósito*, Guillermo L. Taboada, Sabela Ramos, Juan Touriño, Ramón Doallo

Computer Architecture Group, University of A Coruña, A Coruña, Spain

Abstract

The scalability of High Performance Computing (HPC) applications depends heavily on the efficient support of network communications in virtualized environments. However, Infrastructure as a Service (IaaS) providers are more focused on deploying systems with higher computational power interconnected via high-speed networks rather than improving the scalability of the communication middleware. This paper analyzes the main performance bottlenecks in HPC applications scalability on Amazon EC2 Cluster Compute platform: (1) evaluating the communication performance on shared memory and a virtualized 10 Gigabit Ethernet network; (2) assessing the scalability of representative HPC codes, the NAS Parallel Benchmarks, using an important number of cores, up to 512; (3) analyzing the new cluster instances (CC2), both in terms of single instance performance, scalability and cost-efficiency of its use; (4) suggesting techniques for reducing the impact of the virtualization overhead in the scalability of communication-intensive HPC codes, such as the direct access of the Virtual Machine to the network and reducing the number of processes per instance; and (5) proposing the combination of message-passing with multithreading as the most scalable and cost-effective option for running HPC applications on Amazon EC2 Cluster Compute platform.

Keywords: Cloud Computing, High Performance Computing, Amazon EC2 Cluster Compute platform, MPI, OpenMP

*Corresponding author. Computer Architecture Group, University of A Coruña, Campus de Elviña s/n, 15071 A Coruña, Spain. Tel.: +34 981167000; Fax: +34 981167160

Email addresses: rreye@udc.es (Roberto R. Expósito), taboada@udc.es (Guillermo L. Taboada), sramos@udc.es (Sabela Ramos), juan@udc.es (Juan Touriño), doallo@udc.es (Ramón Doallo)

1. Introduction

Cloud computing [1, 2] is an Internet-based computing model which has gained significant popularity in the past several years as it provides on-demand network access to a shared pool of configurable and often virtualized computing resources typically billed on a pay-as-you-use basis.

Virtualization is a mechanism to abstract the hardware and system resources from a given operating system, and it is one of the most important technologies that make the cloud computing paradigm possible. Infrastructure-as-a-Service (IaaS) is a type of cloud service which easily enables users to set up a virtual cluster providing cost-effective solutions. Many research efforts have been done in the last years to reduce the overhead imposed by virtualized environments, and due to this, cloud computing is becoming an attractive option for High Performance Computing (HPC).

Amazon Web Services (AWS) is an IaaS provider whose Elastic Compute Cloud (EC2) [3] is nowadays among the most used and largest public clouds platforms. Some early studies [4, 5, 6] have evaluated public clouds for HPC since 2008 and the main conclusion was that clouds at that time were not designed for running tightly coupled HPC applications. The main reasons were the poor network performance caused by the virtualization I/O overhead, the use of commodity interconnection technologies (e.g., Gigabit Ethernet) and processor sharing, that limit severely the scalability of HPC applications in public clouds. To overcome these constraints Amazon released the Cluster Compute Quadruple Extra Large instance (cc1.4xlarge, abbreviated as CC1) in July 2010, a resource that provides powerful CPU resources (two quad-core processors) and dedicated physical node allocation, together with a full-bisection high-speed network (10 Gigabit Ethernet). The availability of a high-speed network is key for the scalability of HPC applications. However, virtualized network resources lack efficient communication support, which prevents HPC codes to take advantage of these high performance networks. More recently, in November 2011, Amazon released a new type of instance suitable for HPC, the Cluster Compute Eight Extra Large instance (cc2.8xlarge, abbreviated as CC2), with improved CPU power as it has two octa-core processors. Both resources (CC1 and CC2 instances) are intended to be well suited for HPC applications and other demanding network-bound applications [7].

This paper evaluates CC1 and CC2 instances for High Performance Cloud Computing on Amazon EC2 cloud infrastructure, the largest public cloud in production, using up to 512 cores, hence 64 CC1 instances and 32 CC2 instances were used. In this evaluation, the scalability of representative parallel HPC codes using the NAS Parallel Benchmarks (NPB) suite [8] is analyzed. Moreover, as the NPB original suite only contains pure MPI kernels, the NPB Multi-Zone (NPB-MZ) suite [9], which contains hybrid MPI+OpenMP codes, have also been assessed since the use of threads can alleviate the network bottleneck in the cloud.

The structure of this paper is as follows: Section 2 presents the related work. Section 3 describes the I/O support in virtualized environments, and particularly explains the network support in Amazon EC2 CC platform. Section 4 introduces the experimental configuration, both hardware and software, and the methodology of the evaluation conducted in this work. Section 5 analyzes the performance results of the selected message-passing middleware on Amazon EC2 CC instances. These results have been obtained from a micro-benchmarking of point-to-point primitives, as well as an application benchmarking using representative HPC codes in order to analyze the scalability of HPC applications. Section 6 summarizes our concluding remarks.

2. Related Work

There are several feasibility studies of the use of public clouds for HPC, all concluding that the network communication overhead is the main performance bottleneck, limiting severely applications scalability. Many are also the works that tackle the reduction of this network overhead.

Currently virtualization of CPU and memory resources presents very low overhead, close to native (bare-metal) performance on x86 architectures [10]. The key for reducing the virtualization overhead imposed by hypervisors, also known as Virtual Machine Monitors (VMM), are software techniques such as ParaVirtualization (PV) [11] or Hardware-assisted Virtualization (HVM) [12, 13]. However, the efficient virtualization support of I/O devices is still work in progress, especially for network I/O, which turns out to be a major performance penalty [14].

2.1. I/O Virtualization

Previous works on I/O virtualization can be classified as either software- or hardware-based approaches. In software-based approaches devices cannot

be accessed directly by the guest VMs and every I/O operation is virtualized. A representative project is the driver domain model [15], where only an Isolated Device Domain (IDD) has access to the hardware and runs native device drivers. The rest of guest VMs pass the I/O requests to the IDD through special VMM-aware or paravirtual drivers. This technique, implemented by all modern hypervisors, has two main drawbacks: (1) the still poor performance, despite its continuous improvements [16, 17], and (2) the need for guest VM modification, unfeasible in some systems.

Hardware-based I/O virtualization achieves higher performance supporting the direct device access (also known as "PCI passthrough access" or "direct device assignment") from a guest VM. Initially the use of self-virtualized adapters (smart network devices) improved I/O performance by offloading some virtualization functionality onto the device [18, 19]. However, the requirements of non-standard hardware support and custom drivers hindered the adoption of this early ad-hoc hardware-approach. A more recent work by Yassour et al. [20] provides almost native network performance in Linux/KVM environments, but its implementation of the PCI passthrough technique does not support live migration and requires that the VM has exclusive access to a device.

Another hardware-based approach is SR/MR-IOV [21], a standard that allows a physical device to present itself to the system as multiple virtual devices, exporting to VMs part of the capabilities of smart network adapters. Thus, VMs have a direct network path bypassing the hypervisor and the privileged guest. However, the access from multiple VMs to the physical device has to be multiplexed by the network interface firmware. Although this approach achieves reasonably good performance [22], a new hardware support has to be incorporated to PCI devices. Nevertheless, the direct device access cannot provide bare-metal performance, according to Gordon et al. [23], due to the host involvement, as it intercepts all interrupts inducing multiple unwarranted guest/host context switches. In order to eliminate the overhead caused by these unwarranted exits Gordon et al. proposed ELI, a software-only approach for handling interrupts directly and securely within guest VMs.

2.2. HPC in the Cloud

The interest in the use of public clouds for HPC increases as their availability, computational power and performance improves, which has motivated lately multiple works about adopting cloud computing for HPC applications.

Among them, the feasibility of HPC on Amazon EC2 public cloud infrastructure is the most common approach. Regarding applications, the execution of scientific workflows [24, 25, 26, 27] has obtained significant success, whereas many early studies [4, 5, 6, 28, 29] have assessed that public clouds have not been designed for running tightly coupled MPI applications, primarily due to their poor network performance, processor sharing and the use of commodity interconnection technologies. In order to overcome this performance bottleneck, Amazon EC2 introduced their Cluster Compute (CC) instances.

Amazon EC2 CC1 instances, the first version of CC instance type, have been evaluated in some recent related work. Thus, Carlyle et al. [30] compared, from an economic point of view, the benefits in academia of operating a community cluster program versus the provision of Amazon EC2 CC1 instances. Regola and Ducom [31] analyzed the suitability of several virtualization technologies for HPC, among them 4 CC1 instances. Nevertheless, their work is more focused on the overall performance of the evaluated hypervisors instead of the virtualized network performance and the scalability of communications. Ramakrishnan et al. [32] stated that virtualized network is the main performance bottleneck, after analyzing the communication overhead of a number of different interconnect technologies, including 10 Gigabit Ethernet. Furthermore, Zhai et al. [33] conducted a comprehensive evaluation of MPI applications on 16 CC1 instances, revealing a significant performance increase compared to previous evaluations on standard and High-CPU EC2 instances. Finally, Mauch et al. [34] give an overview on the current state of HPC IaaS offerings and present an approach to use InfiniBand in a private virtualized environment. They present HPL benchmark results for both CC1 and CC2 instance types, but using only one instance of each type. In addition, they do not study CC1/CC2 performance and scalability using representative HPC applications and a high number of cores.

The review of the related works on evaluating Amazon EC2 for HPC has revealed the lack of suitable assessments of the performance of the new CC2 instances. This paper addresses this lack evaluating thoroughly the performance of CC2 instances, as well as comparing these instances against the previous CC1 instances. Moreover, previous works were limited to the evaluation of MPI codes using up to 16 CC1 instances (128 cores). Additional contributions of this paper are the performance evaluation using a significantly higher number of cores (up to 512) both in terms of single instance performance, scalability and cost-efficiency of its use as well as taking into account hybrid programming models such as MPI+OpenMP.

3. Virtualization Support on Amazon EC2

Xen [10] is a high performance hypervisor or VMM quite popular among cloud providers, and it is used by all current Amazon EC2 instances. Xen architecture (Figure 1) has the hypervisor as the lowest and most privileged layer and above it comes one or more guest operating systems, which the hypervisor schedules across the physical CPUs. The first guest OS, called domain 0 (dom0), boots automatically when the hypervisor boots and receives special management privileges and exclusive direct access to all physical hardware. This dom0 OS is used to manage any further guest OS, called domain U (domU), and the virtualization technologies supported for creating these domU guests are full virtualization assisted with hardware support (HVM) and ParaVirtualization (PV). I/O virtualization is usually implemented in one of three ways (Figure 2): device emulation, using paravirtualized I/O drivers and giving a VM a direct device access ("PCI passthrough access" or "direct device assignment").

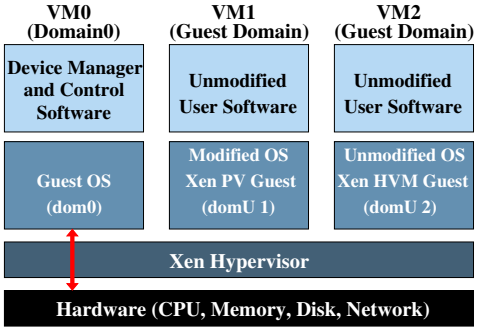
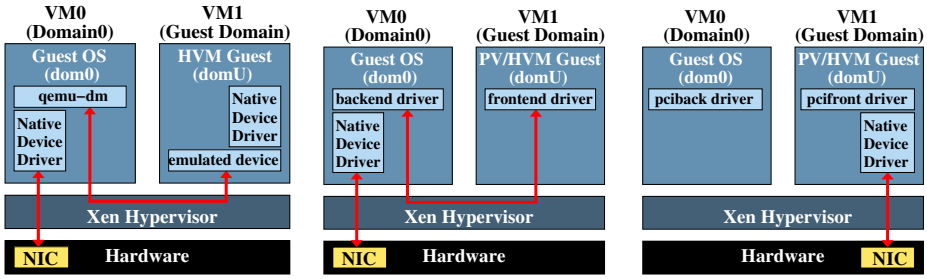


Figure 1: Xen architecture overview

In Xen VMM, device emulation is available for domU HVM guests by default, where dom0 emulates a real I/O device for which the guest already has a driver. The dom0 has to trap all device accesses from domU and converts them to operations on a real and possibly different device (Figure 2(a)), requiring many context switches between domains and therefore offering low performance. In the PV approach (Figure 2(b)), available for PV guests as well as HVM guests, domU guests use special VMM-aware drivers where the I/O code is optimized for VM execution. This code is manually pre-processed or re-coded to remove privileged instructions that are substituted by VM application programming interfaces, or "hypercalls", in their place. Therefore,



(a) Device Emulation (b) Paravirtual Drivers (c) Direct Access

Figure 2: Network virtualization support in Xen

performance is improved but it is still far from native, besides the fact that it requires changes in the device drivers used by domU guests.

With direct device access (Figure 2(c)), also available for both types of guests, the domU guest "sees" a real device and interacts with it directly, without software intermediaries, improving the performance since no dom0 involvement is required. Moreover, domU guests can use any device for which they have a driver, as no modifications are necessary in the native device drivers used by them. For PV guests, Xen does not require any special hardware support, but domU kernel must support the PCI frontend driver (pcifront) in order to work. Hiding the devices from the dom0 is also required, which can be done using the pciback driver. However, an I/O memory management unit (IOMMU) in hardware is required for HVM guests as well as a pciback driver on dom0 kernel. Examples of IOMMUs are Intel's VT-d [35] or AMD's IOMMU [36]. This technique, the only one which can provide near bare-metal performance, also has limitations: it is not fully compatible with live migration and it requires dedication of a device to a domU guest (the latter can be solved with PCI standard SR/MR-IOV [21] devices).

3.1. Amazon EC2 Cluster Compute Platform

The Amazon EC2 Cluster Compute Quadruple Extra Large instances (abbreviated as CC1) and Cluster Compute Eight Extra Large instances (CC2) are resources with 23 and 60.5 Gbytes of memory and 33.5 and 88 EC2 Compute Units (ECUs) for CC1 and CC2, respectively. According to Amazon one ECU provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

	Quadruple Extra Large	**Eight Extra Large**
CPU	2 × Intel Xeon X5570 Nehalem @2.93 GHz	2 × Intel Xeon E5-2670 Sandy Bridge @2.60GHz
ECUs	33.5	88
#Cores	8 (16 with HT)	16 (32 with HT)
Memory	23 GB DDR3	60.5 GB DDR3
Storage	1690 GB	3370 GB
API name	cc1.4xlarge	cc2.8xlarge
Price (Linux)	$1.30 per hour	$2.40 per hour
Interconnect	10 Gigabit Ethernet (Full-bisection bandwidth with Placement Groups)	
Virtualization	Xen HVM 64-bit platform (PV drivers for I/O)	

Table 1: Description of the Amazon EC2 Cluster Compute Quadruple and Eight Extra Large Instances

For these instances, the provider details the specific processor architecture: two Intel Xeon X5570 quad-core Nehalem processors for CC1, hence 8 cores per CC1 instance, and two Intel Xeon E5-2670 octa-core Sandy Bridge processors for CC2, hence 16 cores per CC2 instance. These systems are interconnected via a high-speed network (10 Gigabit Ethernet), which is the differential characteristic of these resources. In fact, these EC2 instance types have been specifically designed for HPC applications and other demanding latency-bound applications.

Both versions of CC instances, whose main characteristics are presented in Table 1, use Xen HVM virtualization technology, whereas the rest of Amazon EC2 instance types are Xen PV guests. Moreover, instead of using an I/O device emulation for the Network Interface Card (NIC) which is configured by default in HVM guests, these cluster instances have installed paravirtual drivers for improving network and disk performance. Therefore, the access to the NIC in Amazon EC2 instances is paravirtualized, so a direct access is not available which causes a significant performance penalty as will be shown later.

4. Experimental Configuration and Evaluation Methodology

The performance evaluation has been conducted on 64 CC1 and 32 CC2 instances of the Amazon EC2 cloud [3]. These resources have been allocated simultaneously and in the same placement group in order to obtain nearby

instances, and thus obtaining the benefits from the full-bisection high bandwidth network provided by Amazon for these instance types.

Regarding the software, two widely extended HPC messaging middleware, OpenMPI [37] 1.4.4 and MPICH2 [38] 1.4.1, were selected for the performance evaluation of native codes (C/C++ and Fortran). In addition, FastMPJ [39] was also selected as representative Message-Passing in Java (MPJ) middleware [40]. In all cases, the most efficient communication mechanism available for network transfers and shared memory scenarios was selected.

The evaluation consists of a micro-benchmarking of point-to-point data transfers, both inter-VM (through 10 Gigabit Ethernet) and intra-VM (shared memory), at the message-passing library level. The point-to-point micro-benchmarking results have been obtained with the Intel MPI Benchmarks suite (IMB) and its MPJ counterpart communicating byte arrays (hence, with no serialization overhead). Then, the impact of paravirtualized network on the scalability of representative parallel codes, NAS Parallel Benchmarks (NPB) kernels [8], has been assessed using the official NPB-MPI version 3.3 and the NPB-MPJ implementation [41]. In order to determine whether a hybrid parallel programming model could overcome the current Amazon EC2 network limitations, the NPB Multi-Zone (NPB-MZ) suite [9] version 3.3.1 for hybrid MPI+OpenMP codes has also been analyzed. The metrics considered for the evaluation of the NPB kernels are MOPS (Millions of Operations Per Second), which measures the operations performed in the benchmark (that differ from the CPU operations issued), and their corresponding speedups. Moreover, NPB Class C workloads have been selected because their performance is highly influenced by the efficiency of the communication middleware and the support of the underlying network, as well as they are the largest workloads that can be executed in a single CC1 instance.

Both the GNU 4.4.4 and the Intel 12.1 compilers have been considered for the NPB kernels, and the reported results in the next graphs have been obtained from binaries compiled with the best compiler in each case. Regarding the Java Virtual Machine (JVM), the version used was the OpenJDK Runtime Environment version 1.6.0_20. Finally, the performance results presented in this paper are the mean of several measurements, generally 10,000 iterations in ping-pong benchmarks and 5 measurements for NPB kernels.

5. Assessment of Amazon EC2 Cluster Compute Platform for HPC

This section presents an analysis of the performance of point-to-point communications and the scalability of HPC codes on the Amazon EC2 Cluster Compute platform designed for HPC, using the selected micro-benchmarks and representative kernels described in the previous section.

5.1. Inter-VM Point-to-point Micro-benchmarking

Figure 3 shows point-to-point latencies (for short messages) and bandwidths (for long messages) of message-passing transfers using the selected message-passing middleware, MPICH2, OpenMPI and FastMPJ, on Amazon EC2 CC1 (left graph) and CC2 (right graph) instances in an inter-VM scenario, where communications are performed through a 10 Gigabit Ethernet network link. The results shown are the half of the round-trip time of a ping-pong test and its corresponding bandwidth.

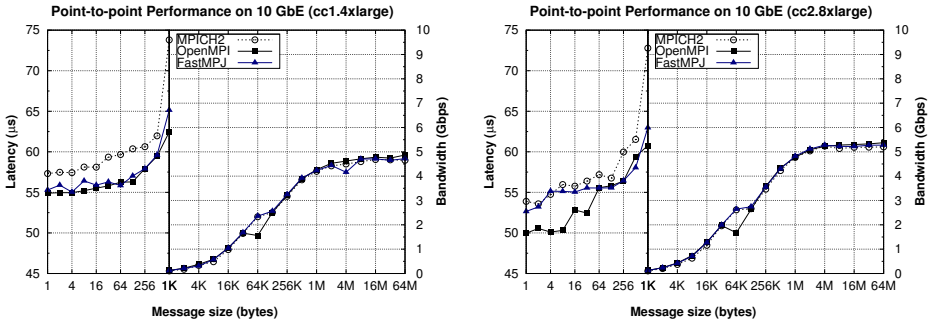


Figure 3: Point-to-point communication performance on Amazon EC2 CC instances over 10 Gigabit Ethernet

On CC1 instances (left graph) OpenMPI and FastMPJ obtains the lowest start-up latency (55 μs), slightly better than MPICH2 latencies (around 57 μs), but quite high compared to the usual 10 Gigabit Ethernet start-up latencies on bare-metal, which can be as low as 10 μs. Regarding bandwidth the three libraries show similar performance on CC1 instances, up to 4.9 Gbps bandwidth, quite far from the theoretical 10 Gbps bandwidth provided by the interconnection technology and the 9 Gbps that message-passing libraries can achieve without virtualization overhead.

The results on CC2 instances (right graph) show slightly better performance (10% in the best case -OpenMPI-) than using CC1 instances, which

can be motivated by the higher performance (about 31%) of the CC2 processor core (5.5 ECUs) compared to the computational power of the CC1 processor core (4.2 ECUs). Thus, these libraries obtain start-up latencies around 50 - 54 μs, whereas observed bandwidths are up to 5.4 Gbps, also suffering from poor network virtualization support. Despite these minimum improvements the overhead in the paravirtualized access of VMs to the 10 Gigabit Ethernet NIC still represents the main performance bottleneck.

5.2. Intra-VM Point-to-point Micro-benchmarking

Figure 4 shows point-to-point performance of message-passing transfers in the intra-VM scenario, where data transfers are implemented on shared memory (hence, without accessing the network hardware). Thus, the observed shared memory performance results are significantly better than the inter-VM scenario.

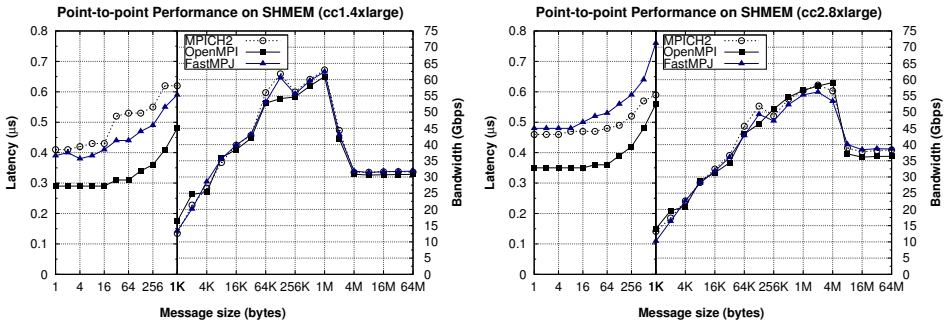


Figure 4: Point-to-point shared memory communication performance on Amazon EC2 CC instances

Performance results on CC1 instances (left graph) present very low start-up latencies, 0.3 μs for OpenMPI and 0.4 μs for MPICH2 and FastMPJ, and similar high bandwidths, up to 60.9 Gbps for OpenMPI and 62.5 Gbps for MPICH2 and FastMPJ, which confirms the efficient virtualization support in the Xen hypervisor for CPU- and memory-intensive operations when no I/O network activity is involved.

Regarding CC2 instances (right graph), start-up latencies are slightly higher than on CC1: OpenMPI also obtains the lowest values, 0.35 μs, whereas MPICH2 and FastMPJ obtain around 0.47 μs, probably due to the lower clock frequency of the processor in CC2 instances (2.60 GHz) than

in CC1 (2.93 GHz). The three evaluated libraries show again similar long message performance results, which suggests that their communication protocols are close to the maximum effective memory bandwidth. The measured shared memory bandwidths are slightly higher in CC1, particularly in the range 1 KByte-1 MByte, due to its higher clock frequency. However, from 2 MBytes CC2 shared memory performance is higher as its L3 cache size (20 MBytes shared by 8 cores) is larger than CC1 L3 cache size (8 MBytes shared by 4 cores). In fact, CC1 performance results fall from 2 MBytes on as the messages and the intermediate shared memory buffer (used internally by message-passing libraries) exceed the L3 cache region that can be addressed by a single process (4 MBytes). Moreover, as the message size increases the percentage of the message that fits in L3 cache reduces and as a direct consequence its performance falls. It has to be taken into account that the OS generally schedules the two communicating processes on the same processor, thus reducing the L3 cache region available for each one. However, this scheduling strategy improves the overall communication performance, especially the start-up latency.

This significant influence of the cache hierarchy on the performance of shared memory communications on both CC1 and CC2 instances confirms the low virtualization overhead experienced, thus providing highly efficient shared memory message-passing communication on virtualized cloud environments.

5.3. HPC Kernels Performance Analysis

The impact of the paravirtualized network overhead on the scalability of representative HPC codes has been analyzed using the MPI and MPJ implementations of the NPB, selected as it is probably the benchmarking suite most commonly used in the evaluation of languages, libraries and middleware for HPC. Four representative NPB kernels, the most communication-intensive codes of the suite, have been evaluated: CG (Conjugate Gradient), FT (Fourier Transform), IS (Integer Sort) and MG (Multi-Grid). As mentioned in Section 4, NPB Class C workloads have been used.

The native (C/Fortran) implementations of the kernels have been compiled using the GNU and Intel compilers (both with -O3 flag), showing their performance for the serial kernels, including also Java results, in Figure 5. As it can be seen, the impact of the compiler on the performance of these codes is almost negligible. Thus, for clarity purposes next graphs only show results obtained with the best performer compiler for each kernel.

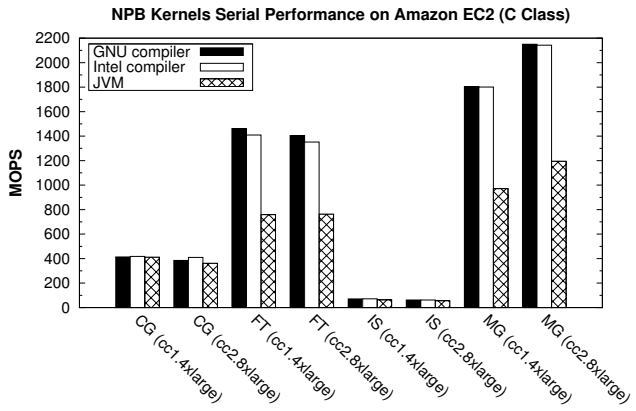


Figure 5: NPB kernels serial performance

Figures 6 and 7 present the performance of CG, FT, IS and MG using up to 512 cores on Amazon EC2 CC platform (hence, using 64 CC1 and 32 CC2 instances). The performance metrics reported are MOPS (left graphs), and their corresponding speedups (right graphs). The number of CC instances used in the performance evaluation is the number of cores used divided by the number of cores per instance type (8 cores for CC1 and 16 cores for CC2).

The analysis of the NPB kernels performance shows that the evaluated libraries obtain good results when running entirely on shared memory (on a single VM) using up to 8 and 16 cores in CC1 and CC2 instances, respectively, due to the higher performance and scalability of intra-VM shared memory communications. However, when using more than one VM, the evaluated kernels scale poorly, experiencing important performance penalties due to the network virtualization overhead. The poorest scalability has been obtained by FT kernel on CC2 instances, CG on CC1, and IS both on CC1 and CC2.

CG kernel, characterized by multiple point-to-point data movements, achieves on CC1 its highest speedup value of 22 using 64 cores, dropping dramatically its performance from that point on as it has to rely on inter-VM communications, where the network virtualization overhead represents the main performance bottleneck. CG obtains higher performance on CC2 instances, a speedup of 40 on 256 cores, although its parallel efficiency is very poor in this case, below 16%. FT kernel achieves a limited scalability on CC1 whereas it does not scale at all on CC2. In fact, FT shows similar

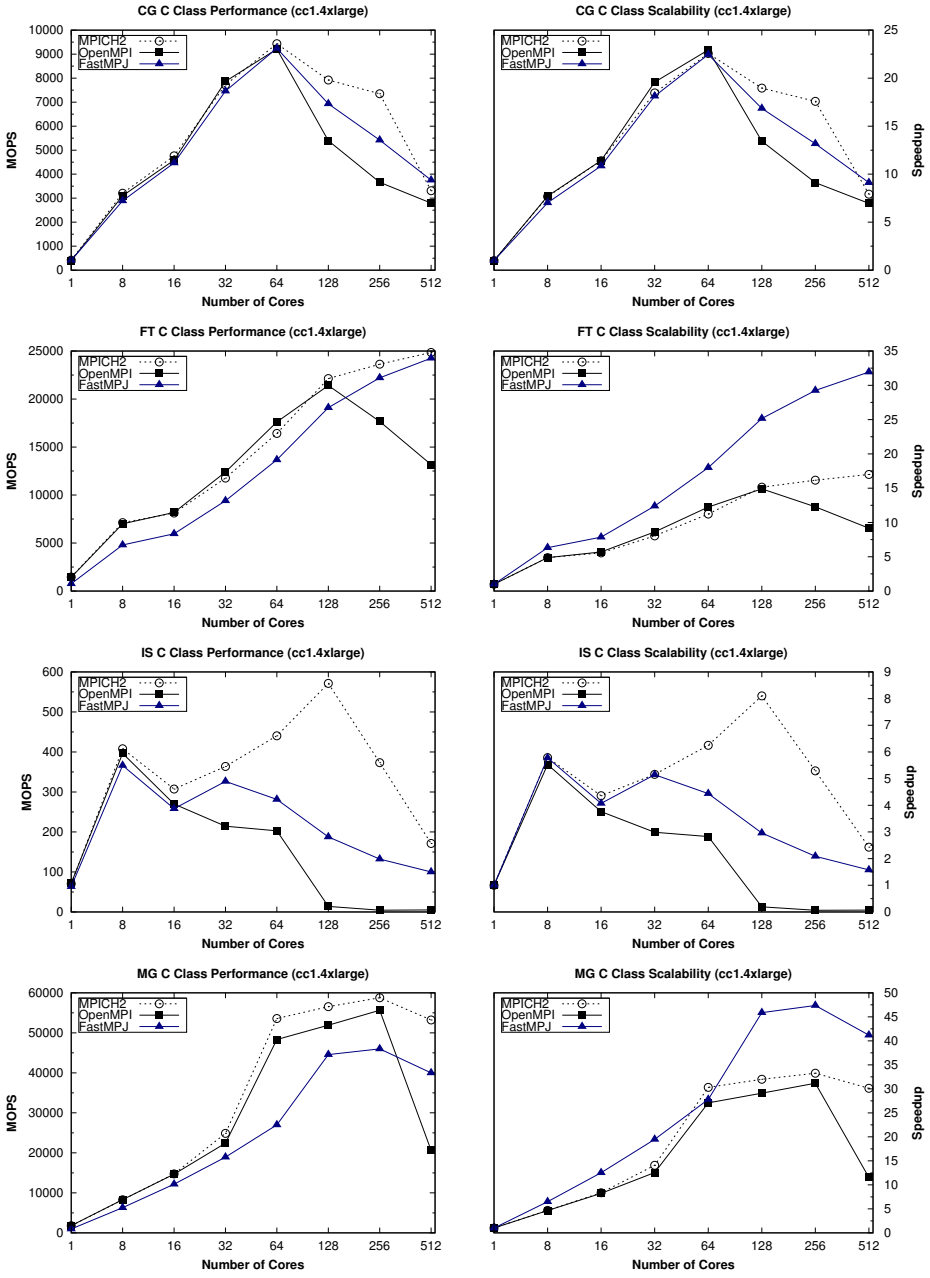


Figure 6: NPB kernels performance and scalability on Amazon EC2 CC1 instances

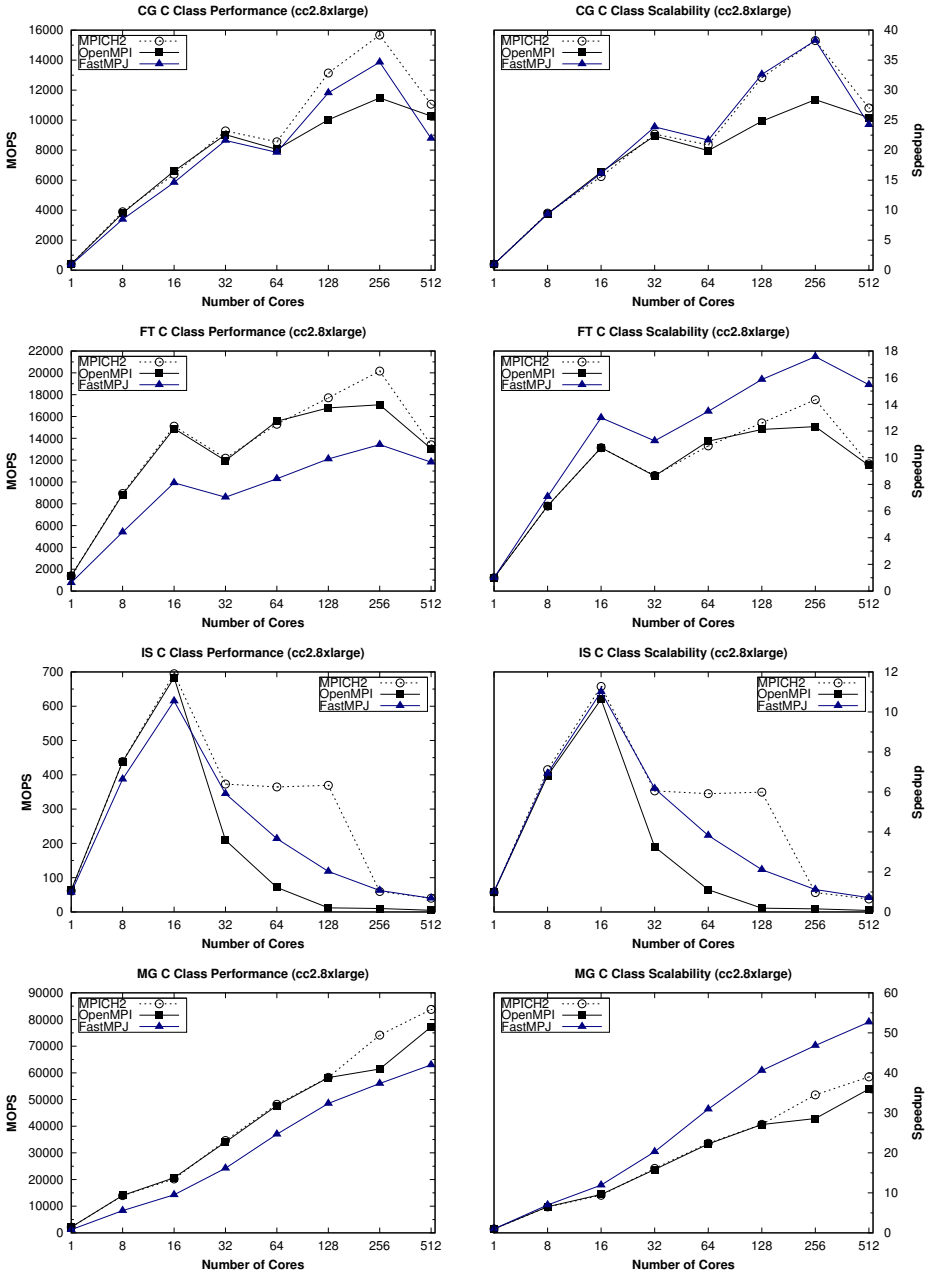


Figure 7: NPB kernels performance and scalability on Amazon EC2 CC2 instances

results on CC2 using a single instance (16 cores) and 32 instances (512 cores). The reason for this behavior is that FT uses extensively Alltoall primitives which access massively the interconnection network, increasing significantly the network overhead imposed by the hypervisor by having 8/16 processes on each instance accessing the paravirtualized NIC.

IS kernel is a communication-intensive code whose scalability greatly depends on the performance of the Allreduce primitive and point-to-point communication start-up latency. Thus, this code only scales when using a single VM thanks to the high performance of intra-VM transfers, whereas it suffers a significant slowdown when using more than one VM, both for CC1 and CC2 instance types. Finally, MG kernel is the less communication-intensive code under evaluation and for this reason it presents the highest scalability on Amazon EC2, especially on CC2 instances, achieving a maximum speedup value of 53 for FastMPJ.

The analysis of scalability has revealed an important issue: the high start-up latencies and limited bandwidths imposed by the paravirtualized access to the NIC limit severely the scalability of communication-intensive codes. The presence of processors with higher computational power in the new CC2 instances partly alleviates this issue for some codes but also aggravates the situation when communication-intensive collective operations (e.g., Alltoall or Allreduce) are involved. Thus, a more efficient I/O network virtualization support, such as a direct access to the NIC in virtualized environments, is needed together with techniques that reduce the number of communicating processes accessing simultaneously through the paravirtualized NIC.

5.4. HPC Kernels Cost Analysis

The use of a public cloud infrastructure like Amazon involves a series of associated costs that have to be taken into account. In order to ease the analysis of the cost of Amazon EC2 resources for HPC we present in Figure 8 the productivity in terms of USD per GOPS (Giga Operations Per Second) of the already evaluated NPB codes. This proposed metric is independent of the number of cores being used. Moreover, these kernels ran on-demand CC1 and CC2 instances. The use of spot instances from the spot market [42] enables bidding for unused Amazon EC2 capacity, which can significantly reduce costs as they provide the same performance as on-demand instances but at lower cost. However, the spot price fluctuates periodically depending on the supply and the demand for spot instance capacity. Moreover, the provision of the requested number of instances, especially when it is a high

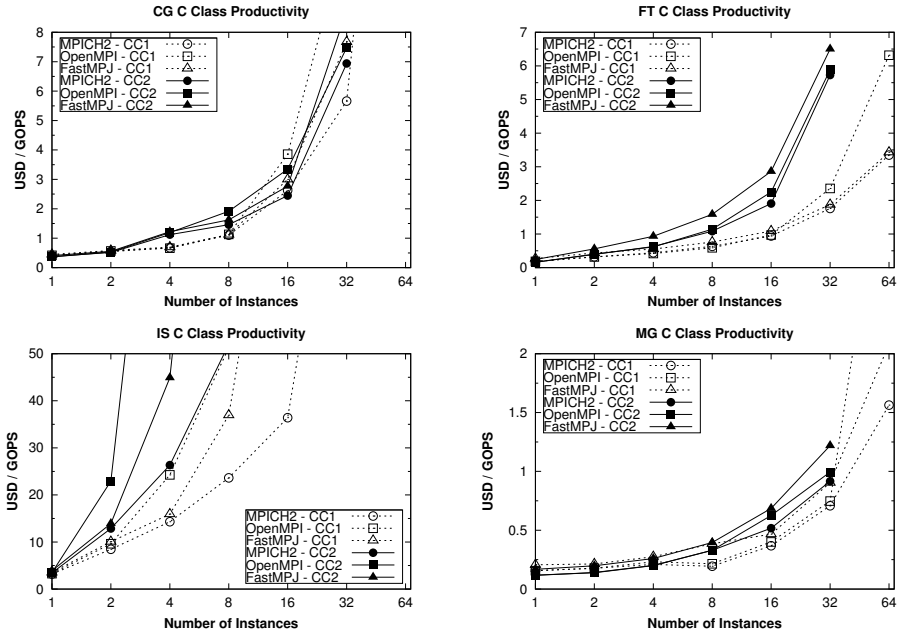


Figure 8: NPB kernels productivity on Amazon EC2 CC instances

number, something that it can be expected for an HPC application, is not fully guaranteed unless the resources are invoked as on-demand instances.

CC2 instances provide twice the number of physical cores and 2.6 times more memory than CC1. Additionally, CC2 instances are based on Sandy Bridge, a more modern microarchitecture than Nehalem, the technology behind CC1, which results in more performance per core. However, CC2 does not provide communication-intensive HPC applications with a performance increase proportional to its higher features due to the network overhead which severely limits the scalability of these applications. If we take into account the cost of each instance (as of May 2012), $1.30 for CC1 and $2.40 for CC2, it can ben easily concluded with the aid of Figure 8 that the use of CC2 instances is generally more expensive than the use of CC1 instances and therefore it is worth recommending CC2 only for applications with high memory requirements that cannot be executed on CC1.

5.5. Impact of Process Allocation Strategies

It seems intuitive that after allocating a certain number of EC2 instances the best option would be running as many processes per instance as cores has the instance, thus fully using the paid resources. However, in terms of performance, this option is not going to be always the best.

Figure 9 presents performance results of the NPB kernels with CC1 instances (left graphs) and CC2 ones (right graphs) using only one process per instance (labeled as "1 ppi") until it reaches the maximum number of available instances, and posteriorly two, four, and finally eight processes per instance. This scheduling strategy, which is significantly much more expensive than the previous one, is able to obtain higher performance in the evaluated kernels, except for CG where this strategy seems to perform similarly (for CC1) or even slightly worse (for CC2). For the other kernels this scheduling achieves up to 7 times better results for IS on CC2 instances (OpenMPI), 3.3 times on CC1 instances (MPICH2) or 2.3 for FT on CC1 instances (OpenMPI). Here FT and IS take full advantage of this approach as the use of less processes per instance alleviates the overhead introduced by the hypervisor in collective communications, frequently used in these kernels. This strategy can be very useful when there is a strong constraint about the execution time of a particular application, such as meeting a deadline, in exchange for much higher costs.

5.6. Using Several Levels of Parallelism

The popularity of hybrid shared/distributed memory architectures, such as clusters of multi-core processors, is currently leading to the support of several levels of parallelism in many prominent scientific problems. The NPB suite is not an exception, existing an NPB Multi-Zone (NPB-MZ) implementation that takes advantage of two-level parallelism through hybrid MPI+OpenMP codes. The NPB-MZ contains three applications, the Lower-Upper Symmetric Gauss-Seidel (LU, but limited to 16 MPI processes in the MZ version), Scalar Penta-diagonal (SP), and Block Tri-diagonal (BT) applications, whose serial performance is shown in Figure 10. FastMPJ results are not shown due to the lack of an NPB-MZ implementation in Java.

Figure 11 shows performance results, in terms of speedups, of the hybrid NPB-MZ Class C workloads, together with their NPB counterparts. The NPB-MZ codes have been executed with the following two configurations: (1) a single MPI process per instance with as many OpenMP threads as cores of the instance (8 and 16 for CC1 and CC2, respectively); and (2) two

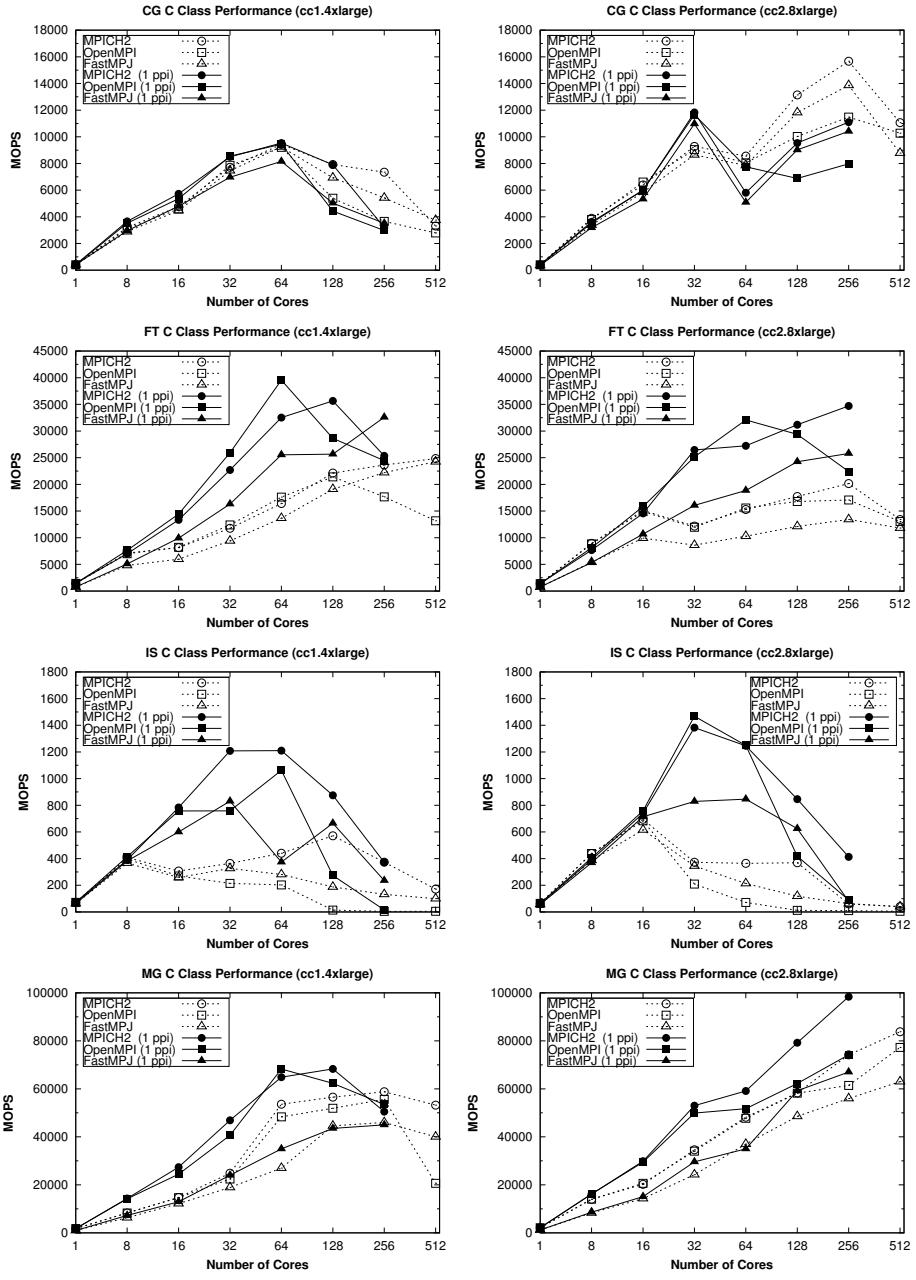


Figure 9: NPB kernels performance on Amazon EC2 CC instances (ppi = processes per instance)

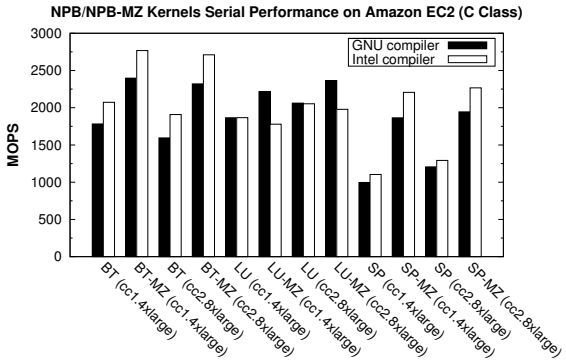


Figure 10: NPB/NPB-MZ kernels serial performance

MPI processes per instance with as many OpenMP threads as cores of each processor (4 and 8 for CC1 and CC2, respectively). Additional configurations have also been evaluated, such as the use of 4 MPI processes per instance and 2 and 4 threads for CC1 and CC2, respectively, but the results obtained were worse.

These results have revealed some important facts: (1) although these codes (BT, LU and SP) are not as communication-intensive as the previously evaluated kernels (CG, FT, IS and MG), the pure MPI versions present poor scalability on CC instances (the maximum speedup achieved is 60 for BT on CC2 using 512 cores); (2) NPB-MZ kernels can overcome this issue and outperform significantly NPB kernels (the maximum reported speedup is 185 on 512 cores for BT-MZ on CC2). Thus, as the message transfers through the network are reduced to only one (or two) MPI processes communicating per instance, the overhead caused by the paravirtualized access to the NIC is drastically decreased; (3) BT and SP obtain the best performance using two MPI processes per instance, which suggests that the right balance between MPI processes and OpenMP threads has to be found for each particular application; and (4) CC2 significantly outperforms CC1 for BT-MZ and SP-MZ in the best configuration (for LU-MZ similar results are obtained) as their overall performance does not rely heavily on the communication performance. Unlike NPB kernels, BT-MZ and SP-MZ present a better cost/performance ratio on CC2 than on CC1.

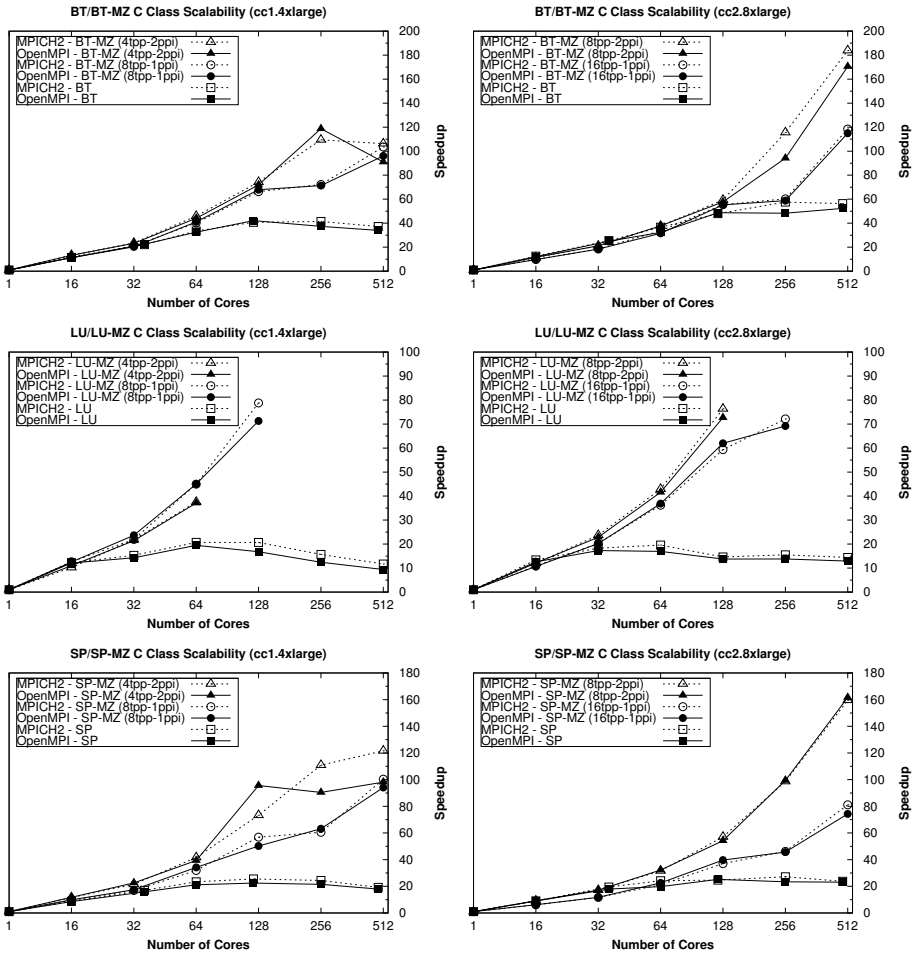


Figure 11: NPB/NPB-MZ kernels scalability (tpp = threads per process − ppi = processes per instance)

6. Conclusions

The scalability of HPC applications on public cloud infrastructures relies heavily on the performance of communications, which depends both on the network fabric and its efficient support in the virtualization layer. Amazon EC2 Cluster Compute (CC) platform provides powerful HPC resources with access to a high-speed network (10 Gigabit Ethernet), although without a proper I/O virtualization support as these resources rely on a paravirtualized access to the NIC.

The contributions of this paper are: (1) it has evaluated the performance of communications on Amazon EC2 CC platform, both 10 Gigabit Ethernet and shared memory transfers for CC1 and CC2 instances; (2) it has assessed the scalability of representative message-passing codes (NPB) using up to 512 cores; (3) it has revealed that the new CC2 instances, despite providing more computational power and slightly better point-to-point communication performance, present poorer scalability than CC1 instances for collective-based communication-intensive applications; (4) the use of CC1 instances is generally more cost-effective than relying on CC2 instances; (5) it is possible to achieve higher scalability running only a single process per instance, thus reducing the communications performance penalty in the access to the network; (6) finally, it has been proposed the use of multiple levels of parallelism, combining message-passing with multithreading, as the most scalable and cost-effective option for running HPC applications on Amazon EC2 CC platform.

Acknowledgment

This work was funded by the Ministry of Science and Innovation of Spain under Project TIN2010-16735 and an FPU Grant AP2010-4348.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility, Future Generation Computer Systems 25 (6) (2009) 599–616.

[2] L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galn, J. Fontn, R. S. Montero, I. M. Llorente, From Infrastructure Delivery to Service Management in Clouds, Future Generation Computer Systems 26 (8) (2010) 1226–1240.

[3] Amazon Web Services LLC, Amazon Elastic Compute Cloud (Amazon EC2), `http://aws.amazon.com/ec2`, Last visited: May 2012.

[4] C. Evangelinos, C. N. Hill, Cloud Computing for parallel Scientific HPC Applications: Feasibility of running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2, in: Proc. 1st Workshop on Cloud Computing and Its Applications (CCA'08), Chicago, IL, USA, 2008, pp. 1–6.

[5] E. Walker, Benchmarking Amazon EC2 for High-Performance Scientific Computing, LOGIN: The USENIX Magazine 33 (5) (2008) 18–23.

[6] J. Ekanayake, G. C. Fox, High Performance Parallel Computing with Clouds and Cloud Technologies, in: Proc. 1st Intl. Conference on Cloud Computing (CLOUDCOMP'09), Munich, Germany, 2009, pp. 20–38.

[7] Amazon Web Services LLC, High Performance Computing Using Amazon EC2, `http://aws.amazon.com/ec2/hpc-applications/`, Last visited: May 2012.

[8] D. H. Bailey et al., The NAS Parallel Benchmarks, International Journal of High Performance Computing Applications 5 (3) (1991) 63–73.

[9] H. Jin, R. F. Van der Wijngaart, Performance Characteristics of the Multi-Zone NAS Parallel Benchmarks, Journal of Parallel and Distributed Computing 66 (2006) 674–685.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the Art of Virtualization, in: Proc. 19th ACM Symposium on Operating Systems Principles (SOSP'03), Bolton Landing, NY, USA, 2003, pp. 164–177.

[11] A. Whitaker, M. Shaw, S. D. Gribble, Denali: Lightweight Virtual Machines for Distributed and Networked Applications, in: Technical Report 02-02-01, University of Washington, USA, 2002.

[12] D. Abramson et al., Intel Virtualization Technology for Directed I/O, Intel Technology Journal 10 (3) (2006) 179–192.

[13] AMD Virtualization Technology (AMD-V), `http://sites.amd.com/us/business/it-solutions/virtualization/Pages/amd-v.aspx`, Last visited: May 2012.

[14] D. Ghoshal, R. S. Canon, L. Ramakrishnan, I/O Performance of Virtualized Cloud Environments, in: Proc. 2nd Intl. Workshop on Data Intensive Computing in the Clouds (DataCloud-SC'11), Seattle, WA, USA, 2011, pp. 71–80.

[15] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, M. Williamson, Safe Hardware Access with the Xen Virtual Machine Monitor, in: Proc. 1st Workshop on Operating System and Architectural Support for the On-Demand IT InfraStructure (OASIS'04), Boston, MA, USA, 2004.

[16] A. Menon, A. L. Cox, W. Zwaenepoel, Optimizing Network Virtualization in Xen, in: Proc. USENIX'06 Annual Technical Conference, Boston, MA, USA, 2006, p. 2.

[17] J. R. Santos, Y. Turner, G. Janakiraman, I. Pratt, Bridging the Gap Between Software and Hardware Techniques for I/O Virtualization, in: Proc. USENIX'08 Annual Technical Conference, Boston, MA, USA, 2008, pp. 29–42.

[18] H. Raj, K. Schwan, High Performance and Scalable I/O Virtualization via Self-Virtualized Devices, in: Proc. 16th IEEE Intl. Symposium on High-Performance Distributed Computing (HPDC'07), Monterey, CA, USA, 2007, pp. 179–188.

[19] K. Mansley, G. Law, D. Riddoch, G. Barzini, N. Turton, S. Pope, Getting 10 Gb/s From Xen: Safe And Fast Device Access From Unprivileged Domains, in: Proc. Workshop on Virtualization/Xen in High-Performance Cluster and Grid Computing (VHPC'07), Rennes, France, 2007, pp. 224–233.

[20] B.-A. Yassour, M. Ben-Yehuda, O. Wasserman, Direct Device Assignment for Untrusted Fully-Virtualized Virtual Machines, Tech. rep. (2008).

[21] PCI SIG. I/O Virtualization, http://www.pcisig.com/specifications/iov/, Last visited: May 2012.

[22] J. Liu, Evaluating Standard-Based Self-Virtualizing Devices: A Performance Study on 10 GbE NICs with SR-IOV Support, in: Proc. 24th IEEE Intl. Parallel and Distributed Processing Symposium (IPDPS'10), Atlanta, GA, USA, 2010, pp. 1–12.

[23] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, D. Tsafrir, A. Schuster, ELI: Bare-Metal Performance for I/O Virtualization, in: Proc. 17th Intl. Conference on Architectural Support for Programming Languages Operating Systems (ASPLOS'12), London, UK, 2012.

[24] C. Vecchiola, S. Pandey, R. Buyya, High-Performance Cloud Computing: A View of Scientific Applications, in: Proc. 10th Intl. Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN'09), Kaoshiung, Taiwan, ROC, 2009, pp. 4–16.

[25] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, P. Maechling, Data Sharig Options for Scientific Workflows on Amazon EC2, in: Proc. 22th ACM/IEEE Conference on Supercomputing (SC'10), New Orleans, LA, USA, 2010, pp. 1–9.

[26] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing, IEEE Transactions on Parallel and Distributed Systems 22 (2011) 931–945.

[27] S. Gogouvitis, K. Konstanteli, S. Waldschmidt, G. Kousiouris, G. Katsaros, A. Menychtas, D. Kyriazis, T. Varvarigou, Workflow management for soft real-time interactive applications in virtualized environments, Future Generation Computer Systems 28 (1) (2012) 193–209.

[28] J. Napper, P. Bientinesi, Can Cloud Computing Reach The TOP500?, in: Proc. Combined Workshops on UnConventional High Performance Computing Workshop Plus Memory Access Workshop (UCHPC-MAW'09), Ischia, Italy, 2009, pp. 17–20.

[29] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, N. J. Wright, Performance Analysis of

High Performance Computing Applications on the Amazon Web Services Cloud, in: Proc. 2nd IEEE Intl. Conference on Cloud Computing Technology and Science (CloudCom'10), Indianapolis, USA, 2010, pp. 159–168.

[30] A. G. Carlyle, S. L. Harrell, P. M. Smith, Cost-Effective HPC: The Community or the Cloud?, in: Proc. 2nd IEEE Intl. Conference on Cloud Computing Technology and Science (CloudCom'10), Indianapolis, USA, 2010, pp. 169–176.

[31] N. Regola, J. C. Ducom, Recommendations for Virtualization Technologies in High Performance Computing, in: Proc. 2nd IEEE Intl. Conference on Cloud Computing Technology and Science (CloudCom'10), Indianapolis, USA, 2010, pp. 409–416.

[32] L. Ramakrishnan, R. S. Canon, K. Muriki, I. Sakrejda, N. J. Wright, Evaluating Interconnect and Virtualization Performance for High Performance Computing, in: Proc. 2nd Intl. Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS'11), Seattle, WA, USA, 2011, pp. 1–2.

[33] Y. Zhai, M. Liu, J. Zhai, X. Ma, W. Chen, Cloud Versus In-House Cluster: Evaluating Amazon Cluster Compute Instances for Running MPI Applications, in: Proc. 23th ACM/IEEE Conference on Supercomputing (SC'11, State of the Practice Reports), Seattle, WA, USA, 2011, pp. 1–10.

[34] V. Mauch, M. Kunze, M. Hillenbrand, High performance cloud computing, Future Generation Computer Systems (In press, http://dx.doi.org/10.1016/j.future.2012.03.011).

[35] G. Neiger, A. Santoni, F. Leung, D. Rodgers, R. Uhlig, Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization, Intel Technology Journal 10 (3) (2006) 167–178.

[36] AMD I/O Virtualization Technology (IOMMU) Specification, `http://support.amd.com/us/Processor_TechDocs/34434-IOMMU-Rev_1.26_2-11-09.pdf`, Last visited: May 2012.

[37] Open MPI Website, `http://www.open-mpi.org/`, Last visited: May 2012.

[38] MPICH2 Website, http://www.mcs.anl.gov/research/projects/mpich2/, Last visited: May 2012.

[39] G. L. Taboada, J. Touriño, R. Doallo, F-MPJ: Scalable Java Message-Passing Communications on Parallel Systems, Journal of Supercomputing 60 (1) (2012) 117–140.

[40] M. Baker, B. Carpenter, MPJ: A Proposed Java Message Passing API and Environment for High Performance Computing, in: Proc. 2nd International Workshop on Java for Parallel and Distributed Computing (JavaPDC'00), Cancun, Mexico, 2000, pp. 552–559.

[41] D. A. Mallón, G. L. Taboada, J. Touriño, R. Doallo, NPB-MPJ: NAS Parallel Benchmarks Implementation for Message-Passing in Java, in: Proc. 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP'09), Weimar, Germany, 2009, pp. 181–190.

[42] W. Voorsluys, S. K. Garg, R. Buyya, Provisioning Spot Market Cloud Resources to Create Cost-Effective Virtual Clusters, in: Proc. 11th Intl. Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'11), Melbourne, Australia, 2011, pp. 395–408.