Parallel Pairwise Epistasis Detection on Heterogeneous Computing Architectures

Jorge González-Domínguez, Sabela Ramos, Juan Touriño, *Senior Member, IEEE*, Bertil Schmidt, *Senior Member, IEEE*

Abstract—Development of new methods to detect pairwise epistasis, such as SNP-SNP interactions, in Genome-Wide Association Studies is an important task in bioinformatics as they can help to explain genetic influences on diseases. As these studies are time consuming operations, some tools exploit the characteristics of different hardware accelerators (such as GPUs and Xeon Phi coprocessors) to reduce the runtime. Nevertheless, all these approaches are not able to efficiently exploit the whole computational capacity of modern clusters that contain both GPUs and Xeon Phi coprocessors. In this paper we investigate approaches to map pairwise epistasic detection on heterogeneous clusters using both types of accelerators. The runtimes to analyze the well-known WTCCC dataset consisting of about 500K SNPs and 5K samples on one and two NVIDIA K20m are reduced by 27% thanks to the use of a hybrid approach with one additional single Xeon Phi coprocessor.

Index Terms—high performance computing, epistasis, pairwise gene-gene interaction, Xeon Phi, GPU

1 Introduction

The latest developments in high-throughput genotyping technologies allow several thousands of individual DNA samples to be genotyped at hundreds of thousands to a few million genetic markers, such as Single Nucleotide Polymorphisms (SNPs). This collection of genotypes is typically linked to a given phenotype in a Genome Wide Association Study (GWAS). The simplest and most common phenotype classification is a binary trait, i.e. the presence (case) or absence (control) of an associated disease. In classical GWAS each genetic marker is analyzed separately in order to identify markers showing differences in genotype frequencies between cases and controls. Unfortunately, this approach is generally not powerful enough to model complex traits for which the detection of joint genetic effects (epistasis) needs to be considered [1], [2], [3]. For instance, 2-SNPs analyses try to find pairs of SNPs whose joint genotype frequencies show a statistically significant difference between cases and controls which potentially explains the effect of the genetic variation leading to disease. Previous efforts have already addressed this problem in high-throughput GWAS datasets [4], [5].

Computing epistasis is highly time-consuming due to the large number of pairwise tests to be calculated. Modern high-throughput technologies are able to gather information of millions of SNPs from thousands of individuals. For instance, even for a moderately-sized dataset consisting of 500,000 SNPs there are about 125 billion pairwise interaction tests to be performed. This leads to prohibitive runtimes on conventional architectures. Targeting this problem with High Performance Computing (HPC) architectures can help to speed up the process to become acceptable in a typical biologist's workflow. Among them, systems with accelerators are popular. Parallel codes exist to detect epistasis on Xeon Phi coprocessors [6], FPGAs [7], GPUs [8], [9], [10], [11], [12], [13] and even clusters of GPUs [14], [15]. However, existing Xeon Phi implementations are still not fully optimized, and one of the contributions of this paper is a new parallel implementation to detect epistatic interactions on Xeon Phi coprocessors. It uses an approach based on regression models that has already been proved accurate [16] and highly efficient on GPUs [13].

Previous efforts have only focused on comparing the performance of two types of accelerators (GPU and Xeon Phi, or GPUs and FPGAs) for the same problem [6], [17]. Due to their different characteristics and programming models, they might be suitable for different types of parallel problems. Nowadays, there exist heterogeneous HPC systems that integrate two types of accelerators. For instance, Stampede [18] and Shepard [19] include nodes with GPUs and other nodes with Xeon Phi coprocessors. Therefore, hybrid GPU/Xeon Phi codes could exploit the characteristics of these systems by using each architecture for the most suitable part of the algorithm. It means, GPUs and Xeon Phi coprocessors can collaborate in order to accelerate the same task. The main contribution

- *J.González-Domínguez and B. Schmidt are with the Parallel and Distributed Architectures Group, Institute of Computer Science, Johannes Gutenberg University Mainz, Germany.*
E-mail: {j.gonzalez,bertil.schmidt}@uni-mainz.de
- *S. Ramos and J. Touriño are with the Computer Architecture Group, Department of Electronics and Systems, University of A Coruña, Spain. E-mail: {sramos,juan}@udc.es*

of this work consists in the development of two hybrid approaches to detect pairwise epistatic interactions on heterogeneous systems with GPUs and Xeon Phi coprocessors. The workload has been efficiently distributed using UPC++ [20], a Partitioned Global Address Space (PGAS) extension of C++. We evaluate their impact in order to speed up the GWAS analyses.

The rest of the paper is organized as follows. Section 2 reviews some related work. Necessary background information is provided in Section 3. Important features of our Xeon Phi-only parallelization and the hybrid GPU/Xeon Phi approaches are described in Sections 4 and 5, respectively. Performance is evaluated in Section 6. Finally, Section 7 concludes the paper.

2 Related Work

Since both the availability and size of GWAS datasets are increasing rapidly, finding faster solutions is of high importance to the research performed in this area. In order to reduce runtimes, some tools such as SNPHarvester [21], TEAM [22], Screen and Clean [23] and SIXPAC [24] apply prefiltering techniques that allow them to reduce the number of analyzed SNP-pairs and perform a selective test only on a subset of all pairwise combinations. Nevertheless, they can potentially lose some significant SNP combinations [25].

A different approach consists of performing massively parallel exhaustive analyses using hardware accelerators. For instance, GWIS [8] is CUDA-enabled code to execute a test based on ROC curves for all SNP-pair combinations. A method based on dependency differences has also been adapted for GPU computation [9]. Regression models are one of the most popular statistical methods to find epistasis, and their accuracy has been extensively proven. BOOST [16], and its CUDA counterpart GBOOST [10], are frequently employed by biologists (see for instance [26], [27], [28]), thanks to their speed and accuracy. GLIDE [11] also uses regression models on GPUs and enables the use of non-discretized genotypes and phenotypes. EpistSearch [13] provides the same accuracy than GBOOST but with shorter runtimes. It relies on the novel KSASA filter, which is less complex and faster to compute than the filters applied in (G)BOOST. Furthermore, the CUDA kernel is optimized to obtain speedups between 1.5 and 2 over GBOOST on a single GPU.

The present work uses the same filters as EpistSearch and thus all the parallel implementations provide the same results as this tool. Our work focuses on the speedup of GWAS codes using hardware accelerators. Consequently, an analysis of the advantages and drawbacks of different filters is out of the scope of this paper. Regression models are used due to their high accuracy providing SNP-pairs with real epistatic interactions (see [29] for an evaluation of different methods). However, our implementations are flexible enough to support other filters with minimal modifications.

SNPsyn, a tool that detects pairwise epistatic interactions and that can be executed on GPUs and Xeon Phi coprocessors, was presented in [6]. It uses the information gain statistic to measure the epistasis effects. The authors propose to split the dataset in chunks that can be analyzed by the CPU or the accelerators, so each one performs the complete analysis of different SNP-pairs. In this paper we have also implemented an approach where different accelerators collaborate on the analysis of every SNP-pair, performing only the part of the computation more suitable to their hardware characteristics. Up to our knowledge, there are no previous efforts on implementing collaborative hybrid GPU/Xeon Phi applications. As it will be shown in Section 6, our parallel implementations are much more efficient even for only one type of accelerator.

3 Background

The goal of our implementation is the exploitation of HPC facilities in order to accelerate the detection of epistasis. We work with datasets containing information about a large number of biallelic genetic markers from many individuals. For each SNP there are three genotypes: homozygous wild (w), heterozygous (h) and homozygous variant (v). They are numerically represented as {0,1,2}. Each individual is characterized as case or control, depending on the presence or absence of an associated disease. Two SNPs present epistasis or interaction if their combination discriminates between cases and controls significantly better than discrimination using each SNP individually.

3.1 Contingency Tables

The number of SNPs and individuals is denoted as M and N, respectively. The individuals are categorized as cases (value 0) and controls (value 1). The filters that identify which SNP-pairs present interaction use a 3x3x2 contingency table per pair. As shown in the example of Table 1, each cell ijk stores the number of individuals categorized as k (case or control) with the value of the first SNP as i, and the second SNP as j. We can also fill the contingency table with probabilities: $\pi_{ijk} = n_{ijk}/N$.

3.2 Filters

Our implementations apply the same three statistical tests as EpistSearch [13] and its multi-GPU version multiEpistSearch [15]. They are based on the definition of epistasis in terms of log-linear models presented in [16]: epistatic interaction is measured as the information contained in the joint distribution but not in its lower-order factorization. This definition

TABLE 1
2-SNP contingency table

Cases	SNP2=0	SNP2=1	SNP2=2
SNP1=0	n_{000}	n_{010}	n_{020}
SNP1=1	n_{100}	n_{110}	n_{120}
SNP1=2	n_{200}	n_{210}	n_{220}
Controls	**SNP2=0**	**SNP2=1**	**SNP2=2**
SNP1=0	n_{001}	n_{011}	n_{021}
SNP1=1	n_{101}	n_{111}	n_{121}
SNP1=2	n_{201}	n_{211}	n_{221}

leads to measure interaction as $\hat{L}_S - \hat{L}_H$, where $\hat{L}_S$ and $\hat{L}_H$ represent the maximum log-likelihood of the saturated and the homogeneous association models, respectively. It can be calculated from the values of the contingency table. The authors establish that all pairs with log-linear measure higher than a certain threshold $THRES$ show epistasis. Although this log-linear model is affordable, it still requires a lot of computation as it has to be computed by iterative methods. This is the reason why a simpler filter based on the Kirkwood Superposition Approximation (KSA) was designed: $\hat{L}_{\text{S}} - \hat{L}_{\text{KSA}}$. The authors proved that this KSA filter is an upper bound of $\hat{L}_S$ and $\hat{L}_H$ and it can be directly calculated from the cells of the contingency table without iterative methods. The detection of epistasis was further optimized with a novel and simpler filter called the KSA's Superposition Approximation (KSASA). Let E and O denote the counts of expected (control) and observed (case) samples, then EpistSearch and multiEpistSearch use the discrete Kullback-Leibler divergence as measure:

$$D_{\text{KL}}(E, O) = \sum_{ij} \pi_{ij1} \log\left(\frac{\pi_{ij1}}{\pi_{ij0}}\right)$$

It can be shown that it is an upper bound of the KSA test and can be used as first prefilter; i.e. it holds:

$$\hat{L}_{\text{S}} - \hat{L}_{\text{H}} \leq \hat{L}_{\text{S}} - \hat{L}_{\text{KSA}} \leq N \cdot D_{\text{KL}}(E, O)$$

Therefore, these tools apply the KSASA filter to all SNP-pairs, discarding those that have a value below the threshold, and calculating the KSA and log-linear tests only for the remaining pairs. In the remainder of this paper, we call the value of $N \cdot D_{\text{KL}}(E, O)$ and $\hat{L}_S - \hat{L}_{KSA}$ for a specific SNP-pair its "KSASA value" and "KSA value", respectively. The algorithmic workflow is summarized in Figure 1.

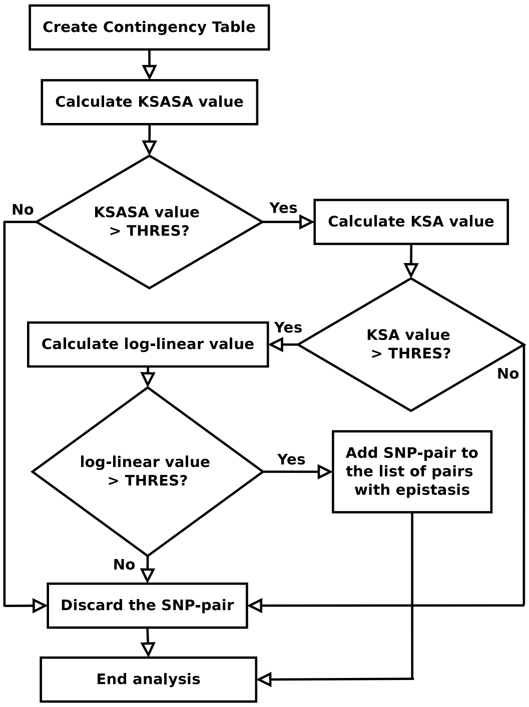

Fig. 1. Workflow of the tests applied to each SNP-pair in EpistSearch and multiEpistSearch

3.3 Xeon Phi Architecture

The Intel Xeon Phi coprocessor [30] is the first commercial manycore system based on the Intel MIC (Many Integrated Core) architecture. Its main advantage over other accelerators or coprocessors is that it is x86-based, enabling the use of general purpose languages like C, C++ or Fortran. Hence, the programming effort focuses on performance exploitation through vectorization and libraries like OpenMP, MPI or Pthreads. Moreover, the use of x86 cores enables the acceleration of codes that are not suitable for GPUs or that require major design and programming efforts, like kernels with branch divergence or several double precision operations.

Figure 2 represents the basic architecture of the Xeon Phi. The current commercial device (code name Knights Corner, KNC), has between 57 and 61 simplified Intel CPU cores running at between 1053 and 1238 MHz, and supports 4 threads per core. The cores have a vector unit with 64 byte registers featuring the vector instruction set known as Initial Many Core Instructions (IMCI) [31]. Each core has a 32 KB L1 data cache, 32 KB L1 instruction cache, and a private 512 KB L2 unified cache which is kept coherent by a distributed tag directory system (DTDs). The cores and the DTDs are arranged on a bidirectional ring bus with the PCIe bus and the memory controllers, that provide access to the GDDR5 memory (between 6 and 16 GB of global memory). The address-mapping to the tag directories is based on hash functions over the memory addresses, leading to an even distribution around the ring. The coprocessor runs a simplified Linux-based OS in one of the cores.

The Xeon Phi can be used in three different manners [32]:

1) *Native model*: or coprocessor-only model. The application is launched on the Xeon Phi and it

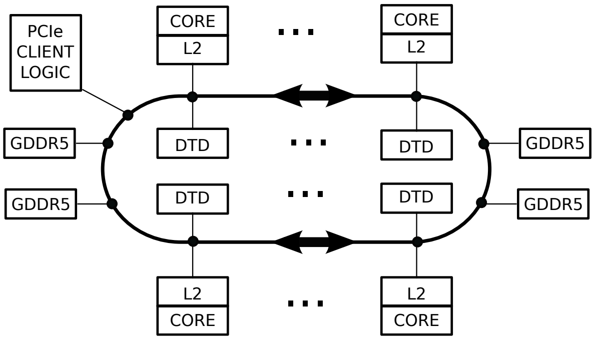

Fig. 2. Architecture of the Intel Xeon Phi coprocessor

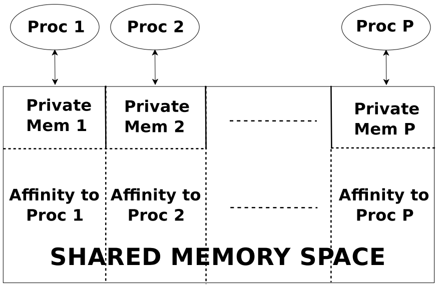

Fig. 3. Memory model in UPC++

runs independently.

2) *Symmetric model*: the coprocessor communicates with the host in a symmetric manner using MPI.
3) *Offload model*: the host offloads blocks of code to be accelerated.
   a) *Automatic offload*: when using the Intel MKL library, some functions can be automatically offloaded to the coprocessor.
   b) *Compiler-assisted offload*: the programmer inserts directives to indicate which blocks of code must run in the coprocessor.

The native model is usually preferred for small applications that run on one Xeon Phi and with almost no I/O instructions. In the symmetric model, the computation is split in MPI tasks that run both in the host(s) and the coprocessor(s). Finally, the offload model enables the use of a Xeon Phi as a mere coprocessor. This mode can be combined with any distributed computing model in which nodes with coprocessors can leverage blocks of code to be accelerated by the Xeon Phis.

3.4 Unified Parallel C++

We have used Unified Parallel C++ (UPC++) [20] in order to implement hybrid GPU/Xeon Phi parallel codes where accelerators are placed on different nodes of a system. This language is an extension of C++ which has evolved from Unified Parallel C (UPC) [33] and follows the PGAS programming model. The execution model of UPC++ is Single Program Multiple Data (SPMD). As this language is able to work on both shared-memory and distributed-memory systems, each independent execution unit (from now on, UPC++ process) can be implemented as an OS process or a pthread. UPC++ takes advantage of unique C++ language features, such as templates, object-oriented design, operator overloading, and lambda functions (in C++ 11) to provide advanced PGAS features. Hence, UPC++ gathers the advantages of this PGAS model and the object-oriented paradigm.

As all PGAS languages, UPC++ exposes to the user a globally shared address space which is logically divided among processes, so each process is associated or presents affinity to a different part of the shared memory. Moreover, UPC++ also provides a private memory space per process for local computations, as shown in Figure 3. Therefore, each process has access to both its private memory and to the whole global space (even the parts that do not present affinity to it) with read/write functions. This memory specification combines the advantages of both the shared and distributed programming models. On the one hand, the global shared-memory space facilitates the development of parallel codes, allowing all processes to directly read and write from remote memory (shared-memory space with affinity to another process) without explicitly notifying the owner. On the other hand, the performance of the codes can be increased by taking data affinity into account. Typically the accesses to remote data will be much more expensive than the accesses to local data (i.e. accesses to private memory and to shared memory with affinity to the process).

Other parallel programming paradigms, such as MPI, could have been employed to implement the hybrid GPU/Xeon Phi approaches. UPC++ has been selected because it gathers the following necessary features:

- UPC++ is able to work on distributed-memory systems. Therefore, our approaches can be used on clusters, and GPUs and Xeon Phi coprocessors are not required to be in the same node.
- C++ codes can be integrated into the application (the Xeon Phi filters are implemented using this language).
- Locks in shared memory are available in order to synchronize the work of the processes, as needed for the inter-pair parallelization (see Section 5.1). This feature is not present in MPI, where the only alternative would be a more expensive barrier.
- One-sided communications available in UPC++ can provide better performance than two-sided communications [34], [35].
- UPC++ provides asynchronous copies that were used to overlap computation and communication.

4 Parallel Implementation on the Xeon Phi Coprocessor

The three fundamental techniques to achieve high performance on the Xeon Phi are: scaling, vectorization and memory usage [36]. The Xeon Phi provides a large number of threads and 512-bit SIMD instructions at the cost of using simplified Intel CPU cores. Hence, in order to benefit from the Xeon Phi architecture, workloads have to be able to exploit high degrees of parallelism. Moreover, given its low core frequency, the use of vector instructions is key to accelerate the code run by each core.

Regarding memory usage, on the one hand the Xeon Phi benefits from local memory accesses to its GDDR5 modules instead of accessing the host memory through PCIe. On the other hand, although all cores can access all L2 slices, the access to remote cache slices is costly due to the directory-based cache coherence protocol [37].

The workflow of EpistSearch complies with the requirements to exploit the Xeon Phi manycore architecture. First of all, it presents an embarrassingly parallel workflow in which SNP-pairs can be analysed independently, as shown in Figure 1. Second, if the SNPs are uploaded into the coprocessor memory, each core only accesses the information of the pair that is being processed, and it only has to write to the output array when it finds a pair with epistasis. Third, the information of the SNPs is arranged in arrays that are manipulated to generate the contingency tables, enabling the use of vector instructions. Note that we have to align the memory addresses of these arrays in order to better exploit vectorization.

Regarding the Xeon Phi use models, we have selected the offload model because it allows us to leverage the computation of the contingency tables and the filters (see Figure 1) into the coprocessor while the host performs the I/O tasks (loading SNP information and writing the output file). Moreover, it simplifies the use of multiple coprocessors, being the host in charge of splitting tasks (sets of SNP-pairs) among them.

Within the compiler-assisted offload, there are two ways of managing data transfers between the host and the coprocessor: shared and non-shared memory models [38]. The shared memory model (or implicit memory copy model) uses Cilk to share complex data structures (pointer-based structures, classes, etc.) between the host and the coprocessor, suffering from coherence overhead. The non-shared memory model (or explicit memory copy model) relies on directives that indicate explicitly which data has to be transferred to and from the coprocessor and when. The non-shared memory model enables a finer-grained control over data transfers but it is limited to primitive data types. The data required to process SNP-pairs is stored in arrays of primitive types and hence we use the non-shared memory model with explicit data transfers.

Since the host is not performing extra computation, we use synchronous transfers (i.e., the host offloads a block of code and waits until the coprocessor finishes). Once the data has been transferred, the Xeon Phi operates with the SNP-pairs to obtain the contingency tables and applies the filters to select those pairs with epistasis. We have used OpenMP within the Xeon Phi to exploit the inter-pair parallelism. The pairs are generated using a nested loop that iterates over two lists of SNPs, covering all the combinations. We split the outer loop among threads using OpenMP so that each thread maintains one SNP in cache and iterates over the second set of SNPs. Since the computation of each pair is independent, we do not need to assign tasks taking into account thread locality. We have experimentally verified that two threads is usually the optimum load per core, since it enables to interleave memory accesses. Moreover, the core running the OS and in charge of the offload daemon must be avoided when distributing the computation.

One of the most costly operations is the creation of the contingency tables, whose bottleneck is the Hamming weight function (`popcount`) to count the activated bits in the arrays that represent the SNPs. We use a `popcount` function developed by the authors of MICA [39] that takes advantage of prefetching and vectorization. This function is optimized to operate with full cache lines. In our implementation, each SNP is represented by four arrays (two for the cases and two for the controls) in which each bit represents one individual. Thus, one cache line (64 bytes) comprises information for 512 individuals. This means that the `popcount` function will provide better performance when the number of cases and the number of controls are both multiple of 512.

Finally, once the Xeon Phi has selected those SNP-pairs with epistasis, the results are transferred back to the host to be written to an output file.

5 Hybrid GPU/Xeon Phi Parallelization

Two approaches have been explored in order to implement a hybrid GPU/Xeon Phi parallel code where the two types of accelerators collaborate to analyze the same dataset: inter-pair and intra-pair. Both are flexible enough to work with several GPUs and Xeon Phi coprocessors and are based on UPC++.

5.1 Inter-Pair Parallelism

In this first approach each accelerator performs the whole analysis (calculation of the contingency tables and whole filtering) of different SNP-pairs. The UPC++ processes call either GPU or Xeon Phi functions to perform the analysis of a block of SNP-pairs. The biallelic information of the SNPs is distributed

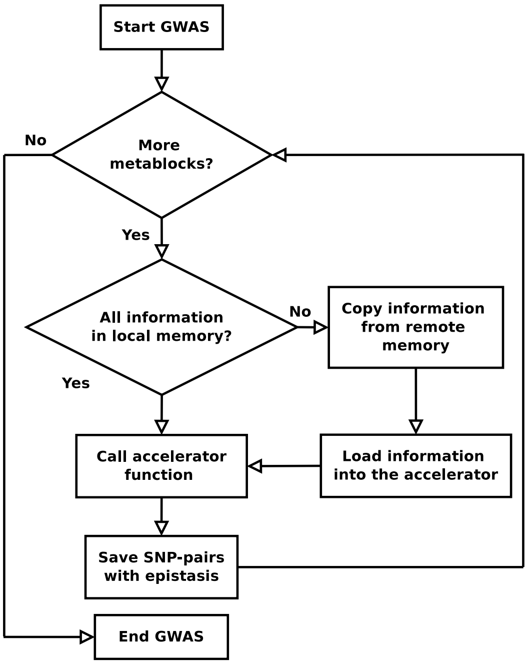
Start GWAS
More metablocks?
No
Yes
All information in local memory?
No
Copy information from remote memory
Yes
Call accelerator function
Load information into the accelerator
Save SNP-pairs with epistasis
End GWAS

Fig. 4. Procedure within each UPC++ process in the inter-pair approach

in blocks in a round-robin way, where the user can specify the number of blocks per process. This information is stored in shared memory so it can be directly accessed by all processes.

The flowchart in Figure 4 summarizes the behavior of each UPC++ process. The term "metablock" describes each block of SNP-pairs that must be computed, i.e., all possible combinations with the blocks of SNPs. In an example with 3 processes where the biallelic data is distributed using 2 blocks per process (6 blocks in total) there would exist 36 metablocks. However, as the results of the filters are the same for symmetric pairs (i.e., same result for pair (x, y) and pair (y, x)), only 21 metablocks of SNP-pairs must be analyzed. In general, if the biallelic information is split in n blocks, $\frac{n \cdot (n+1)}{2}$ metablocks are analyzed. For each associated metablock, if the biallelic information is not in the part of shared memory with affinity, the process initially reads it from remote memory and loads it into the accelerator memory. Next, it calls the device to search for epistasis in all the SNP-pairs within the metablock (the whole analysis illustrated in Figure 1). Thanks to the shared-memory space and the one-sided communications available in UPC++, remote copies can be performed without synchronization with the owner. Moreover, the execution on the accelerator and the copy of the information needed to compute the next metablock are overlapped using asynchronous communication.

The metablocks are not initially assigned to certain UPC++ processes. A table of UPC++ locks, with one open lock per metablock, is created in shared memory (accessible by all processes). Every time one accelerator finishes the analysis of one metablock, the process accesses this shared table to know which metablocks have not been or are not being computed at the same time by other devices, i.e., those metablocks whose lock is still open. Once one idle process finds a metablock to compute, it closes the associated lock.

The main advantage of the inter-pair approach is its minimum data transfer requirements among accelerators. Only the access to the table in shared memory with the information of the remaining metablocks needs to be synchronized using locks. Internal executions are completely independent as the computations on each device are related to different SNP-pairs, i.e., accelerators do not need to wait for data generated on other devices. Moreover, as the workload schedule is dynamic and flexible (the first device to finish, the first to analyze the next metablock), faster accelerators perform more work.

5.1.1 Considerations for Xeon Phi

This implementation enables the use of multiple Xeon Phis regardless of whether they are in the same or in different hosts. We also provide an optimization for coprocessors that reside in the same host with only one process per host and using asynchronous offload primitives to distribute the metablocks among the multiple Xeon Phis. Unlike synchronous offload primitives, asynchronous ones do not block the host until the computation of the offloaded code has finished (unless a `wait` directive is found). This enables to transfer metablocks to all coprocessors using only one host process without serializing the computation on the different coprocessors. We use `wait` directives to ensure that the computation on one Xeon Phi has finished before the host tries to offload the next metablock to the same card. Data transfers are optimized so that the coprocessors are able to reuse data that do not change throughout consecutive offloads. The results are copied back to the host only after each coprocessor finishes its last part of code.

5.2 Intra-Pair Parallelism

A problem of the inter-pair approach is that its CUDA kernel is affected by thread divergence because one thread could need to perform the KSA and log-linear filters for one SNP-pair whereas the other threads of the warp discarded their pairs in the KSASA filter and thus are idle. The effect of the divergence depends on the number of SNP-pairs discarded by each filter but it is always harmful for performance. We have developed an intra-pair version that removes from the kernel the computation that is not always performed by all threads. This version divides the workflow illustrated in Figure 1 into two parts and assigns each of them to a different type of accelerator. Concretely, the GPUs calculate the contingency tables and the KSASA filter of all SNP-pairs while the Xeon Phi coprocessors

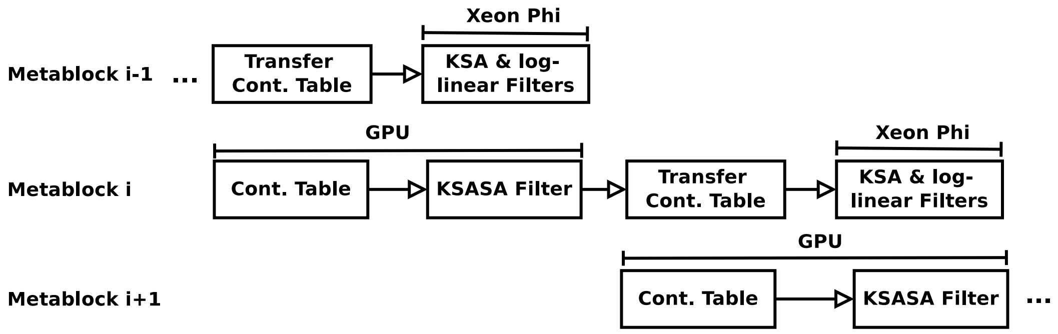

Fig. 5. Workflow of the hybrid GPU/Xeon Phi intra-pair approach with overlapping

compute the KSA and log-linear filters of the SNP pairs when necessary. Therefore, in this version the SNP-pairs are not distributed among the accelerators, but both GPUs and Xeon Phi coprocessors collaborate to analyze the same SNP-pair. Similarly to the inter-pair approach, the computation is split in metablocks. One UPC++ process is created for each available accelerator. As an example, the workflow for each metablock in an execution with one GPU and one Xeon Phi coprocessor would be:

1) Calculation of the contingency tables and application of the KSASA filter on the GPU for all the SNP-pairs within the metablock. The GPU kernel has been simplified compared to the inter-pair approach by removing the KSA and log-linear filters. This step finishes with the contingency tables of the SNP-pairs that passed the KSASA filter copied into a buffer of shared memory. This buffer has affinity to the UPC++ process associated to the GPU so that the contingency tables can be directly copied from GPU memory to the buffer and, at the same time, they are accessible to other processes.
2) Copy of the contingency tables of the SNP-pairs that passed the KSASA filter to another buffer in the part of the shared memory with affinity to the UPC++ process associated to the Xeon Phi coprocessor. This buffer will be used as input for the computation on the Xeon Phi.
3) Computation of the KSA filter using the input contingency tables on the Xeon Phi. Additionally, the log-linear filter is performed for those SNP-pairs that passed the two previous filters.
4) Write the information of the SNP-pairs that passed the three filters in the output file. This is performed by the CPU.

Copies in PGAS languages can be performed following a push or a pull approach, i.e., the process responsible of transferring data is either the source (push) or the destination (pull). In this case the copy of step 2 is performed in a pull way. It means that the host process associated to the Xeon Phi directly copies the remote data to local memory because this process has less computational workload.

Our intra-pair implementation is flexible enough to work with several GPUs and/or several Xeon Phi coprocessors. In this case, the SNP-pairs and contingency tables would be distributed among the GPUs and Xeon Phi coprocessors in steps 1 and 3, respectively. Furthermore, although this workflow is inherently sequential, we overlap computation with communication and other computation by simultaneously working with different metablocks on different accelerators, as illustrated in Figure 5: the UPC++ processes associated to the Xeon Phi coprocessors perform steps 2 and 3 for metablock i at the same time that the processes associated to GPUs complete step 1 for the next metablock ($i+1$). The writing of the results (step 4) by the CPU is also overlapped with the previous steps.

The main advantage of this approach is the reduction of the CUDA thread divergence. As it will be seen in Section 6.1 the computational power of modern GPUs is higher than that of Xeon Phi coprocessors. Therefore, optimizing the CUDA kernel is key in order to improve performance. The new kernel implemented for the intra-pair version only includes the calculation of the contingency tables and the KSASA filter, which are performed for all SNP-pairs. Therefore, the computation of each thread is similar, the divergence is reduced and the performance significantly improves. On the other hand, the necessary copies of the contingency tables between different parts of the shared memory (step 2) are the main source of overhead of this intra-node approach. Moreover, the sequential nature of the workflow (the four steps must be computed sequentially for the same metablock) could have been an additional drawback but it has been significantly alleviated with the overlapping of the computation of different metablocks.

6 Experimental Evaluation

Twelve synthetic datasets, consisting of different numbers of SNPs (50K and 100K) and individuals (1,600; 2,048; 3,200; 4,096; 6,400; 8,192), have been used in our experiments. They have been generated with the genomeSIMLA tool [40] and contain the same number of cases and controls. Regarding the hardware, two

TABLE 2
Specifications of the hardware accelerators used for the experimental evaluation

	NVIDIA K20m	Xeon Phi 5110P
Cores	2496	60
Clock frequency	0.706 GHz	1.053 GHz
Memory size	5 GB	8 GB
Memory bandwidth	208 GB/s	320 GB/s

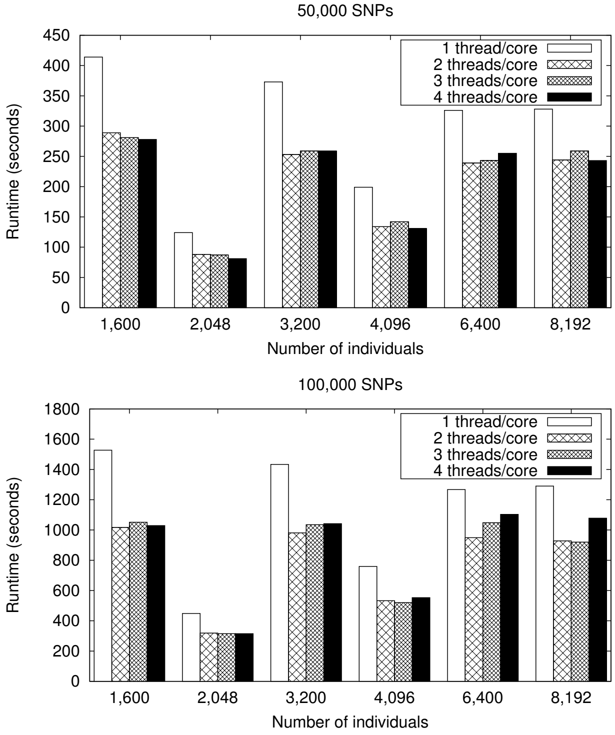

Fig. 6. Evaluation of the single Xeon Phi implementation according to the number of threads per core

NVIDIA K20m GPUs and two Intel Xeon Phi 5110P coprocessors have been employed. Their characteristics are summarized in Table 2. Note that although the Xeon Phi architecture provides 60 cores, one of them is used for the operating system and the offload daemon issues, and thus only 59 cores are employed for computation. Figure 6 shows the runtime (in seconds) of the single Xeon Phi implementation presented in Section 4 depending on the number of threads per core. The use of only one thread per core always leads to the worst performance but there is no significant difference among two, three or four threads per core. Similar conclusions are obtained for multiple coprocessors. This behavior has been previously observed for other Xeon Phi codes [41], [42]. From now on we will use two threads per core (118 threads in total) for all the executions on the Xeon Phi coprocessors as more threads do not imply better performance.

TABLE 3
Runtimes (in seconds) of multiEpistSearch (on one K20m GPU) and the Xeon Phi implementations

Num SNPs	Num Inds.	1 GPU	1 Phi	2 Phi
50,000	1,600	8.64	289.41	146.18
	2,048	9.18	88.14	47.20
	3,200	12.21	253.05	136.93
	4,096	13.52	133.72	75.57
	6,400	16.76	238.99	129.37
	8,192	20.49	244.49	137.85
100,000	1,600	37.75	1017.07	562.81
	2,048	38.63	318.57	177.07
	3,200	43.77	980.51	552.91
	4,096	50.41	532.72	305.27
	6,400	68.74	948.63	513.38
	8,192	83.15	928.47	531.50

6.1 GPU vs Xeon Phi Comparison

Table 3 compares the runtimes (in seconds) of the Xeon Phi implementation described in Section 4, the optimized multi-Phi version explained in Section 5.1.1, and our GPU counterpart. The GPU times have been obtained executing multiEpistSearch on one NVIDIA K20m as the CUDA kernel of this tool is more optimized than EpistSearch thanks to the exploitation of the GPU shared memory (see [15] for more details).

The first conclusion that can be drawn is that the evolution of runtimes according to the number of individuals is different on the GPU and the Xeon Phi coprocessors. The GPU presents the expected runtime behavior: linear increase with the number of individuals and quadratic with the number of SNPs. This behavior is also reproduced on the Xeon Phi if we consider only the series with number of individuals power of two (2,048; 4,096; 8,192). For instance, this runtime series for 50K SNPs on one coprocessor is (88.14s, 133.72s, 244.49s), and for 100K SNPs is (318.57s, 532.72s, 928.47s). However, runtimes on one Xeon Phi coprocessor for other amount of samples are much higher than expected. For example, the time needed to analyze the dataset with 50K SNPs and 1,600 individuals (289.41s) is 3.28 times higher than for the same number of SNPs but 2,048 samples (88.14s). Similar trends are observed for two Xeon Phi coprocessors. The reason for this erratic behavior is the implementation of the `popcount` function (see Section 4), which is the core part of the calculation of the contingency tables. Our `popcount` function is optimized to work with full lines which, in our approach, are filled with the information of 512 individuals. Therefore, the use of a number of samples where cases and controls are multiple of 512 enables to benefit from vectorization and memory alignment. Padding could be used to alleviate this effect at the expense of increasing the memory usage (twice the memory would be necessary in the worst case).

Regarding the comparison between the GPU and the Xeon Phi, runtimes are significantly lower for

GPUs: between 8.25 and 33.50 times faster than one Xeon Phi and between 5.14 and 16.92 compared to two coprocessors. The maximum speedup is reduced if we only take into account datasets containing a number of cases and controls multiple of 512: 11.93 and 6.75 times faster over one and two coprocessors, respectively. The high speedups of the GPU over the Xeon Phi are due to the higher real computational power of the NVIDIA K20m compared to the Xeon Phi 5110P. The theoretical peak performance of the Xeon Phi [30] is close to the NVIDIA K20m because it considers that all floating point instructions are vectorized. However, real codes rarely have all instructions vectorized and real performance is far from the theoretical peak. A new metric has been used to show the level of hardware exploitation of each implementation. Let M be the number of SNPs and T the runtime, we define speed (S) as the thousands of SNP-pairs analyzed per second:

$$S = \frac{M\,(M-1)}{T \cdot 2 \cdot 10^3}$$

Let C and F denote the number of cores of the accelerator and the clock frequency per core, respectively. The parallel exploitation metric (*Exp*) is calculated as the speed divided by the maximum amount of GHz provided by each core of the accelerator:

$$Exp = \frac{S}{C \cdot F} = \frac{M\,(M-1)}{C \cdot F \cdot T \cdot 2 \cdot 10^3}$$

Figure 7 shows the level of hardware exploitation of multiEpistSearch on one NVIDIA K20m and of our implementations for one and two Xeon Phi coprocessors for the same 12 datasets presented before. The values for C and F were obtained from Table 2, and T from Table 3. Note that we use $C = 118$ (two threads per core) to calculate the metric of the Xeon Phi versions as it has been proved as the best configuration. These graphs show that the difference of performance among the three implementations is not high when we take into account the computational power of the accelerator. The exploitation level is on average only 1.50 and 1.65 times higher on the GPU than on one and two Xeon Phi coprocessors, respectively. Furthermore, the exploitation metric of the Xeon Phi versions is higher than that of the GPU for the datasets containing 50K SNPs and number of cases and controls multiple of 512. This is caused by the exploitation of vectorization in the `popcount` function. Figure 7 is also useful to graphically illustrate the trends of the performance for a varying number of individuals: the hardware exploitation on the GPU decreases linearly, but the lines for the Xeon Phi present several peaks (since the Xeon Phi implementations are more efficient when the number of cases and controls is a multiple of 512).

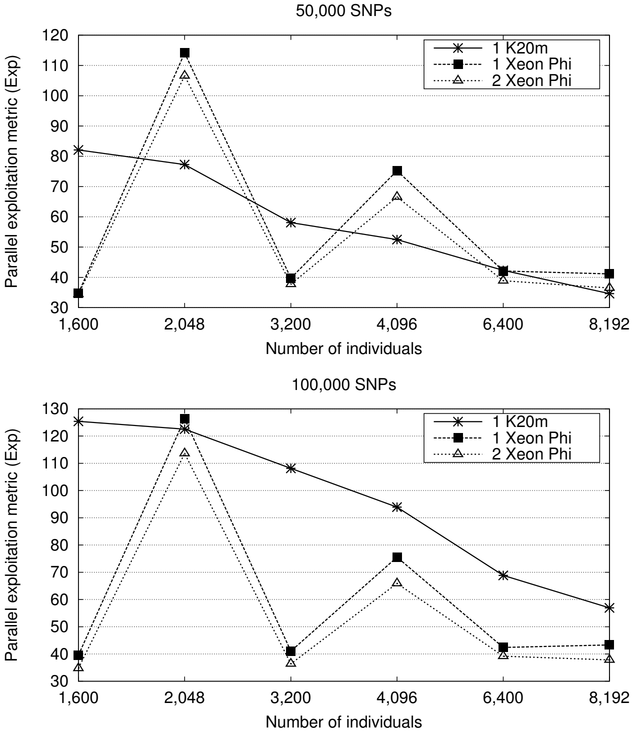

Fig. 7. Parallel exploitation of multiEpistSearch (on one K20m GPU) and the Xeon Phi implementations

TABLE 4
Runtimes (in seconds) of SNPsyn

Num SNPs	Num Inds.	1 GPU	1 Phi	2 Phi
50,000	1,600	878.24	980.38	494.79
	2,048	1114.74	1216.69	614.48

We have also evaluated the performance of SNPsyn [6] on the same system in order to compare our implementations to the previous work. We have tested the two different methods available in SNPsyn (*Laplace* and *Relative*) for both types of accelerators, with and without additional heuristics to filter SNPs before applying the detection method. Although the use of the different detection methods causes minor variations in performance, the use of filters has significant impact on the results. In order to provide a fair comparison, only the runtime for the best approach on each scenario is included in Table 4 (*Laplace* and the *main* heuristic on GPU; *Relative* without heuristic on Xeon Phi). Only the two smallest datasets have been employed for this comparison due to the large runtime of SNPsyn. The runtime of our implementations for the same datasets can be seen in the first two rows of Table 3. These results show that our GPU implementation is more efficient than SNPsyn, with speedups over 100 for both datasets. The Xeon Phi implementation of SNPsyn is more competitive, but still between 3.38 and 13.80 times slower.

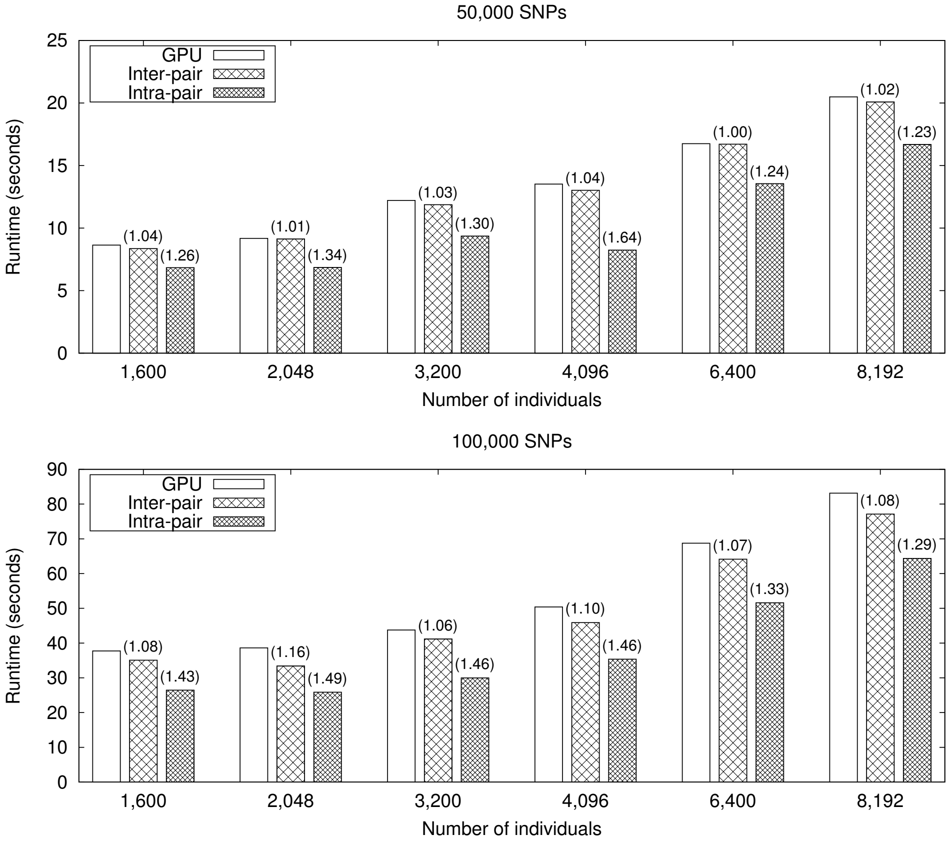

Fig. 8. Performance comparison of multiEpistSearch on one GPU and the hybrid inter- and intra-pair approaches using one additional Xeon Phi coprocessor. In parentheses, the speedups of the hybrid versions over the GPU-only version

6.2 Evaluation of the Hybrid Approaches

The hybrid GPU/Xeon Phi implementations presented in Section 5 have been evaluated on a system with two nodes connected through an InfiniBand FDR network (maximum bandwidth of 56 Gbps and latencies around 1-2 microseconds). One node contains two NVIDIA K20m GPUs and the other node one Intel Xeon Phi 5110P coprocessor. Figure 8 shows the performance of multiEpistSearch on one NVIDIA K20m and the inter-pair and intra-pair approaches when using an additional Xeon Phi coprocessor. Speedups of the hybrid implementations over the GPU-only one are specified in parentheses above the boxes. In all the experiments the intra-pair version is significantly faster than the inter-pair one. It obtains on average speedups of 1.37 and 1.20 over the GPU-only and the inter-pair approaches, respectively. As explained in Section 5.2, the thread divergence of the CUDA kernel in the intra-pair version is removed, which optimizes the part of the algorithm executed on the GPU (creation of the contingency tables and KSASA filter). Although the computational power of the Xeon Phi is lower, it is sufficient to perform the KSA and log-linear filters of the selected SNP-pairs within one block simultaneously to the GPU computation of the next block. Furthermore, thanks to overlapping computation and communication, the transfer of the contingency tables information between GPU and Xeon Phi does not involve performance overhead.

However, the speedups of the inter-pair version are much lower. On average it is only 1.06 times faster than using only the GPU. In this case the thread divergence is not removed from the CUDA kernel as the GPU must perform the whole analysis of the corresponding SNP-pairs. Due to its lower computational power, only a small percentage of SNP-pairs (metablocks) are assigned to the Xeon Phi coprocessor. Therefore, the impact of its collaboration is not significant. Moreover, there is a performance overhead due to accessing the table stored in shared memory from different nodes. Similar conclusions are taken from Figure 9 for the experiments with two GPUs and the Xeon Phi coprocessor. In this case the inter-pair version is even slower than using only the two GPUs whereas the intra-pair approach obtains speedups between 1.21 and 1.59.

Table 5 presents a comparison between our inter-pair version and SNPsyn. As explained in Section 6.1, in SNPsyn each card obtains maximum performance with different methods and heuristics. Therefore,

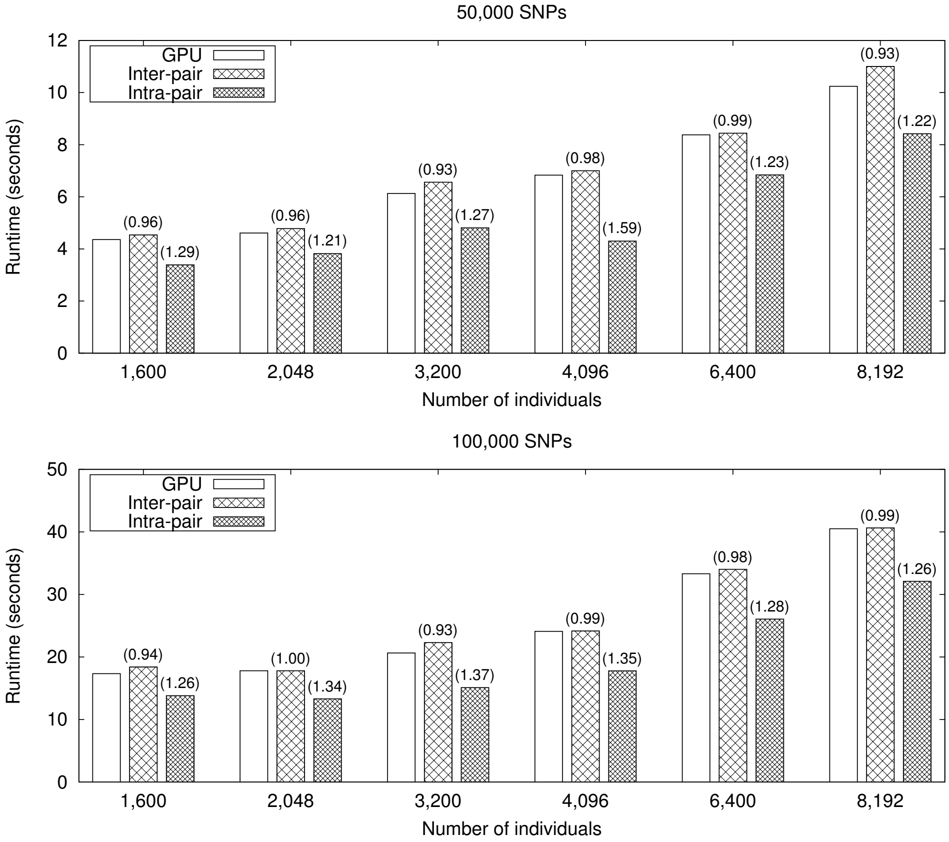

Fig. 9. Performance comparison of multiEpistSearch on two GPUs and the hybrid inter- and intra-pair approaches using one additional Xeon Phi coprocessor. In parentheses, the speedups of the hybrid versions over the GPU-only version

when combining both accelerators, it is not possible to select the best configuration for both cards without causing inconsistencies in the results. In order to select the best parameters for a hybrid scenario, the analysis was performed for all configurations. The task scheduling in SNPsyn depends on the estimation of the relative performance of each accelerator, which depends on the set of parameters used. To avoid bias related to this estimation, we divided the job in 10 chunks and tested different splitting schemes. Although the reported times correspond to the best configuration and splitting scheme, the performance of SNPsyn is not competitive even when compared to our inter-pair version (note that the SNPsyn latency does not include the cost of task scheduling and merging the final results), being the runtimes between 79.15 and 116.98 times slower.

Finally, Table 6 shows the runtime and the speeds *S* (as thousands of SNP-pairs per second) for multiEpistSearch on one and two K20m and the hybrid implementations with an additional Xeon Phi coprocessor, analyzing a real-world dataset obtained from the Wellcome Trust Case-Control Consortium (WTCCC) [43]. It consists of 3,004 controls from the 1958 British Birth Cohort and 2,005 cases with bipolar disorder genotyped at 500,568 SNPs. The runtime reduction of the intra-pair approach is 27% compared to using either only one or two GPUs. Nevertheless, only 6% and 3% of reduction is obtained, respectively, with the inter-pair version.

TABLE 5
Runtimes (in seconds) of SNPsyn and the inter-pair version using 50,000 SNPS

Num Inds.	Library	1 GPU + 1 Phi	2 GPUs + 1 Phi
1,600	inter-pair	8.36	4.54
	SNPsyn	661.69	434.26
2,048	inter-pair	9.13	4.68
	SNPsyn	844.69	547.48

The results of this section have shown that Xeon Phi coprocessors can be effectively used to accelerate codes in combination with GPUs and that the key resides in assigning different tasks that are suitable for the different architectures. Hence, appropriate hybrid implementations can significantly accelerate certain types of parallel applications, such as the detection of epistatic interactions in GWAS.

TABLE 6
Performance comparison of multiEpistSearch and the hybrid implementations using the WTCCC dataset

Version	Hardware	Runtime	Speed (10^3 pairs/s)
Hybrid intra-pair	2 K20m & 1 Xeon Phi	9 m 56 s	210,229
Hybrid inter-pair	2 K20m & 1 Xeon Phi	13 m 15 s	157,538
multiEpistSearch	2 K20m	13 m 37 s	153,323
Hybrid intra-pair	1 K20m & 1 Xeon Phi	19 m 52 s	105,099
Hybrid inter-pair	1 K20m & 1 Xeon Phi	25 m 33 s	81,734
multiEpistSearch	1 K20m	27 m 15 s	76,621

7 Conclusion

The application of HPC techniques to pairwise epistasis detection is key to enable the analysis of large-scale datasets while maintaining reasonable runtimes in a biologist's workflow. This work has explored the combination of different accelerators in order to enhance the performance of an epistasis detection workload. All the implementations presented in this work use regression models to detect which SNP-pairs present epistasis. The employed filters were selected because they have been proved to be efficient in previous works. Nevertheless, the conclusions related to the efficiency of the parallel techniques presented in this work could be applicable to other parallel implementations even if they use different statistical filters.

Firstly, we have provided an efficient implementation for the Xeon Phi. Although porting codes to Xeon Phi involves using well-known languages and traditional paradigms, extra effort has to be spent in optimizing codes for the specific architecture. Our implementation exploits Xeon Phi features like the 512-bit vector instructions and it significantly outperforms the only state-of-the-art tool for this coprocessor. We have also compared its performance to GPU counterparts. Although current GPUs are faster, the combination with Xeon Phi coprocessors can serve to accelerate parts of the code that are not able to fully exploit the GPU hardware, benefiting from the use of a more flexible x86 architecture. Secondly, two hybrid approaches have been analyzed to conclude that different manycore architectures like GPUs and Xeon Phis are complementary and can be successfully used in collaboration to accelerate different sections of a workload. In fact, the intra-pair approach vs the GPU-only implementation obtains a maximum speedup of around 1.6x for the synthetic datasets, and a runtime reduction of 27% for the WTCCC dataset.

There is an increasing number of heterogeneous clusters with both GPUs and Xeon Phi coprocessors. Up to now, users execute their parallel applications using only one type of accelerator and thus not exploiting the whole computational capacity of the system. In this work we have shown that efficient hybrid GPU/Xeon Phi implementations can significantly improve the performance of parallel tools. The experimental results and conclusions provided by this work can lead in the future to implement hybrid approaches for other parallel applications.

An interesting approach to implement hybrid parallel algorithms is the usage of platform-independent languages such as OpenCL. This would have the advantage of providing potability across different devices; e.g., NVIDIA as well as AMD GPUs could be supported. However, the performance of OpenCL kernels is not always portable across platforms; i.e., to achieve optimal performance a different kernel often has to be written for each device. Studying the performance of an OpenCL implementation of the algorithm to detect pairwise epistasic interactions is an interesting research direction.

Acknowledgments

This study makes use of data generated by the Wellcome Trust Case-Control Consortium. A full list of the investigators who contributed to the generation of the data is available from http://www.wtccc.org.uk. Funding for the project was provided by the Wellcome Trust under award 076113 and 085475. This work was also supported by the Ministry of Economy and Competitiveness of Spain and FEDER funds of the EU (Project TIN2013-42148-P).

References

[1] B. Maher, "Personal Genomes: the Case of the Missing Heritability," *Nature*, vol. 456, no. 7218, pp. 18–21, 2008.

[2] P. C. Phillips, "Epistasis, the Essential Role of Gene Interactions in the Structure and Evolution of Genetic Systems," *Nature Review Genetics*, vol. 9, no. 11, pp. 855–867, 2008.

[3] J. H. Moore, F. W. Asselbergs, and S. M. Williams, "Bioinformatics Challenges for Genome-Wide Association Studies," *Bioinformatics*, vol. 26, no. 4, pp. 445–455, 2010.

[4] H. J. Cordell, "Detecting Gene-Gene Interactions that Underlie Human Diseases," *Nature Review Genetics*, vol. 10, no. 6, pp. 392–404, 2009.

[5] K. van Steen, "Travelling the World of Gene-Gene Interactions," *Briefings in Bioinformatics*, vol. 13, no. 1, pp. 1–19, 2011.

[6] D. Sluga, T. Curk, B. Zupan, and U. Lotric, "Heterogeneous Computing Architecture for Fast Detection of SNP-SNP Interactions," *BMC Bioinformatics*, vol. 15, no. 6, pp. 216:1–216:8, 2014.

[7] L. Wienbrandt, J. C. Kässens, J. González-Domínguez *et al.*, "FPGA-Based Acceleration of Detecting Statistical Epistasis in GWAS," in *Proc. 14th Intl. Conf. on Computational Science (ICCS 2014)*, Cairns, Australia, 2014.

[8] B. Goudey, D. Rawlinson, Q. Wang *et al.*, "GWIS - Model-Free, Fast and Exhaustive Search for Epistatic Interactions in Case-Control GWAS," *BMC Genomics*, vol. 14, no. 3, pp. S10:1–S10:18, 2012.

[9] J. Piriyapongsa, C. Ngamphiw, A. Intarapanich *et al.*, "iLOCi: a SNP Interaction Priorization Technique for Detecting Epistasis in Genome-Wide Association Studies," *BMC Genomics*, vol. 13, no. 7, pp. S2:1–S2:15, 2012.

[10] L. S. Yung, C. Yang, X. Wan *et al.*, "GBOOST: A GPU-Based Tool for Detecting Gene-Gene Interactions in Genome-Wide Case Control Studies," *Bioinformatics*, vol. 27, no. 9, pp. 1309–1310, 2011.

[11] T. Kam-Thong, C. Azencott, L. Cayton *et al.*, "GLIDE: GPU-Based Linear Regression for Detection of Epistasis," *Human Heredity*, vol. 73, pp. 220–236, 2012.

[12] G. K. Chen and Y. Guo, "Discovering Epistasis in Large Scale Genetic Association Studies by Exploiting Graphics Cards," *Frontiers in Genetics*, vol. 4, no. 266, 2013.

[13] J. González-Domínguez, B. Schmidt, J. C. Kässens, and L. Wienbrandt, "Hybrid CPU/GPU Acceleration of Detection of 2-SNP Epistatic Interactions in GWAS," in *Proc. 15th Intl. European Conf. on Parallel and Distributed Computing (Euro-Par'14)*, Porto, Portugal, 2014, pp. 680–691.

[14] B. Pütz, T. Kam-Thong, N. Karbalai *et al.*, "Cost-effective GPU-Grid for Genome-wide Epistasis Calculations," *Methods of Information in Medicine*, vol. 52, pp. 91–95, 2013.

[15] J. González-Domínguez, J. C. Kässens, L. Wienbrandt, and B. Schmidt, "Large-Scale Genome-Wide Association Studies on a GPU Cluster Using a CUDA-Accelerated PGAS Programming Model," *Intl. Journal of High Performance Computing Applications*, 2015 (in press).

[16] X. Wan, C. Yang, Q. Yang *et al.*, "BOOST: a Fast Approach to Detecting Gene-Gene Interactions in Genome-Wide Case-Control Studies," *The American Journal of Human Genetics*, vol. 87, no. 3, pp. 325–340, 2010.

[17] J. González-Domínguez, L. Wienbrandt, J. C. Kässens *et al.*, "Parallelizing Epistasis Detection in GWAS on FPGA and GPU-accelerated Computing Systems," *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, 2015 (in press).

[18] "Texas Advance Computing Center (TACC) User Portal. Stampede User Guide." https://portal.tacc.utexas.edu/user-guides/stampede, [Last visited June 2015].

[19] "The Navy DoD Supercomputing Resource Center (Navy DSRC). High Performance Computing Systems: Shepard." http://navydsrc.hpc.mil/hardware/index.html, [Last visited June 2015].

[20] Y. Zheng, A. Kamil, M. Driscoll *et al.*, "UPC++: a PGAS Extension for C++," in *Proc. 28th IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS'14)*, Phoenix, AR, USA, 2014, pp. 1105–1114.

[21] C. Yang, Z. He, X. Wan *et al.*, "SNPHarvester: a Filtering-Based Approach for Detecting Epistatic Interaction in Genome-Wide Association Studies," *Bioinformatics*, vol. 25, no. 4, pp. 504–511, 2009.

[22] X. Zhang, S. Huang, F. Zou *et al.*, "TEAM: Efficient Two-Locus Epistasis Tests in Human Genome-Wide Association Study," *Bioinformatics*, vol. 26, no. 12, pp. i217–i227, 2010.

[23] J. Wu, B. Devlin, S. Ringquist *et al.*, "Screen and Clean: a Tool for Identifying Interactions in Genome-Wide Association Studies," *Genetic Epidemiology*, vol. 34, no. 3, pp. 275–285, 2010.

[24] S. Prabhu and I. Peer, "Ultrafast Genome-Wide Scan for SNP-SNP Interactions in Common Complex Disease," *Genome Research*, vol. 22, no. 11, pp. 2230–2240, 2012.

[25] J. González-Domínguez and B. Schmidt, "GPU-Accelerated Exhaustive Search for Third-Order Epistatic Interactions in Case-Control Studies," *Journal of Computational Science*, vol. 8, pp. 93 – 100.

[26] R. L. Milne, J. Herranz, K. Michailidou *et al.*, "A Large-Scale Assessment of Two-Way SNP Interactions in Breast Cancer Susceptibility Using 46,450 Cases and 42,461 Controls from the Breast Cancer Association Consortium," *Human Molecular Genetics*, vol. 23, no. 7, pp. 1934–1946, 2014.

[27] M. Chu, R. Zhang, Y. Zhao *et al.*, "A Genome-Wide Gene-Gene Interaction Analysis Identifies an Epistatic Gene Pair for Lung Cancer Susceptibility in Han Chinese," *Carcinogenesis*, vol. 32, no. 3, pp. 572–577, 2014.

[28] J. Bi, J. Gelernter, J. Sun *et al.*, "Comparing the Utility of Homogeneous Subtypes of Cocaine Use and Related Behaviors with DSM-IV Cocaine Dependence as Traits for Genetic Association Analysis," *American Journal of Medical Genetics*, vol. 165, no. 2, pp. 148–156, 2014.

[29] Y. Wang, G. Liu, M. Feng *et al.*, "An Empirical Comparison of Several Recent Epistatic Interaction Detection Methods," *Bioinformatics*, vol. 27, no. 21, pp. 2936–2943, 2011.

[30] R. Rahman, "Intel® Xeon Phi™ Core Micro-architecture," *Intel Press*, 2012.

[31] G. Chrysos, "Intel® Xeon Phi™ Coprocessor (Codename Knights Corner)," Keynote talk at the 24th Hot Chips: A Symposium on High Performance Chips, Cupertino, CA, USA, 2012.

[32] "Intel® Xeon Phi™ Coprocessor: Software Developers Guide," https://www-ssl.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessor-system-software-developers-guide.html, 2014, [Last visited June 2015].

[33] T. El-Ghazawi, W. Carlson, T. L. Sterling *et al.*, *UPC: Distributed Shared-Memory Programming*. John Wiley & Sons Inc, 2005.

[34] C. Bell, D. Bonachaea, R. Nishtala, and K. Yelick, "Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap," in *Proc. 20th IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS'06)*, Rhodes Island, Greece, 2006.

[35] R. Nishtala, Y. Zheng, P. Hargrove, and K. Yelick, "Tuning Collective Communication for Partitioned Global Address Space Programming Models," *Parallel Computing*, vol. 37, no. 9, pp. 576–591, 2011.

[36] J. Reinders, "An Overview of Programming for Intel® Xeon® and Intel® Xeon Phi™ Coprocessors," https://software.intel.com/sites/default/files/article/330164/an-overview-of-programming-for-intel-xeon-processors-and-intel-xeon-phi-coprocessors_1.pdf, 2012, [Last visited June 2015].

[37] S. Ramos and T. Hoefler, "Modeling Communication in Cache-Coherent SMP Systems: a Case-study with Xeon Phi," in *Proc. of the 22nd Intl. Symp. on High-perf. Parall. and Distrib. Computing (HPDC'13)*, New York, NY, USA, 2013, pp. 97–108.

[38] S. U. Thiagarajan, C. Congdon, S. Naik, and L. Q. Nguyen, "Intel® Xeon Phi™ Coprocessor Developer's Quick Start Guide," https://software.intel.com/sites/default/files/managed/26/d6/Intel_Xeon_Phi_Quick_Start_Developers_Guide-MPSS-3.4.pdf, 2014, [Last visited June 2015].

[39] S.-H. Chan, J. Cheung, E. Wu, H. Wang, C.-M. Liu, X. Zhu, S. Peng, R. Luo, and T. W. Lam, "MICA: A Fast Short-read Aligner that Takes Full Advantage of Intel Many Integrated Core Architecture (MIC)," *arXiv preprint*, 2014.

[40] T. L. Edwards, W. S. Bush, S. D. Turner *et al.*, "Generating Linkage Disequilibrium Patterns in Data Simulations Using genomeSIMLA," in *Proc. 6th European Conf. on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBIO'08)*, Naples, Italy, 2008, pp. 736–743.

[41] A. Ramachandran, J. Vienne, R. V. D. Wijngaart *et al.*, "Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi," in *Proc. of the 42nd Intl. Conf. on Parallel Processing (ICPP'13)*, Lyon, France, 2013, pp. 763–743.

[42] G. Misra, N. Kurkure, A. Das *et al.*, "Evaluation of Rodinia Codes on Intel Xeon Phi," in *Proc. of the 4th Intl. Conf. on Intelligent Systems, Modelling and Simulation (ISMS'13)*, Bangkok, Thailand, 2013, pp. 415–419.

[43] The Wellcome Trust Case Control Consortium, "Genome-Wide Association Study of 14,000 Cases of Seven Common Diseases and 3,000 Shared Controls," *Nature*, vol. 447, no. 7145, pp. 661–78, 2007.

Jorge González-Domínguez received the B.S., M.S. and PhD degrees in Computer Science from the University of A Coruña, Spain, in 2008, 2010 and 2013, respectively. He is currently a postdoctoral researcher in the Parallel and Distributed Architectures Group at the Johannes Gutenberg University Mainz, Germany. His main research interests are in the areas of high performance computing for bioinformatics and PGAS programming languages.

Sabela Ramos received the B.S. (2009), M.S. (2010) and Ph.D. (2013) degrees in Computer Science from the University of A Coruña, Spain. Currently she is a postdoctoral researcher and a Teaching Assistant in the Department of Electronics and Systems at the University of A Coruña. Her research interests are in the area of High Performance Computing, focused on message-passing communications and performance modelling on multi and manycore architectures.

Juan Touriño (M'01-SM'06) received the B.S. (1993), M.S. (1993) and Ph.D. (1998) degrees in Computer Science from the University of A Coruña, Spain. In 1993 he joined the Department of Electronics and Systems at the University of A Coruña, where he is currently a Full Professor of Computer Engineering. He has extensively published in the area of High Performance Computing (HPC): parallel algorithms and applications, programming languages and compilers for HPC, high performance networks and architectures, etc. He is coauthor of over 140 papers on these topics in international conferences and journals.

Bertil Schmidt (M'04-SM'07) is tenured Full Professor and Chair for Parallel and Distributed Architectures at the University of Mainz, Germany. Prior to that he was a faculty member at Nanyang Technological University (Singapore) and at University of New South Wales (UNSW). His research group has designed a variety of algorithms and tools for Bioinformatics mainly focusing on the analysis of large-scale sequence and short read datasets. For his research work, he has received a CUDA Research Center award, a CUDA Academic Partnership award, a CUDA Professor Partnership award and the Best Paper Award at IEEE ASAP 2009.