Journal Title
XX(X):??–??

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Fast Search of Third-Order Epistatic Interactions on CPU and GPU Clusters

Christian Ponte[1], Jorge González-Domínguez[1] and María J. Martín[1]

Abstract

Genome Wide Association Studies (GWAS), analyses that try to find a link between a given phenotype (such as a disease) and genetic markers, have been growing in popularity in the recent years. Relations between phenotypes and genotypes are not easy to identify, as most of the phenotypes are a product of the interaction between multiple genes, a phenomenon known as epistasis. Many authors have resorted to different approaches and hardware architectures in order to mitigate the exponential complexity of the problem. However, these studies make some compromises in order to keep a reasonable execution time, such as limiting the number of genetic markers involved in the interaction, or discarding some of these markers in an initial filtering stage. This work presents MPI3SNP, a tool that implements a 3-way exhaustive search for cluster architectures with the aim of mitigating the exponential growth of the runtime. Modern cluster solutions usually incorporate GPUs. Thus, MPI3SNP includes implementations for both multi-CPU and multi-GPU clusters. To contextualize the performance achieved, MPI3SNP is able to analyze an input of 6,300 genetic markers and 3,200 samples in less than 6 minutes using 768 CPU cores or 4 minutes using 8 NVIDIA K80 GPUs. The source code is available at: https://github.com/chponte/mpi3snp.

Keywords

Bioinformatics, Epistasis, Genetic Interaction, GPU, GWAS, High Performance Computing, MPI, Mutual Information

Introduction

A Genome-Wide Association Study (GWAS) is an observational study that attempts to identify genes involved in the manifestation of a specific trait (for instance, suffering from a certain disease). Typically, a GWAS tries to find correlation between a phenotype of interest and genotype frequencies from population samples.

The genetic markers in GWAS usually take the form of a collection of Single-Nucleotide Polymorphisms (SNPs), which serve as a representation of the variation present in the genetic code. A SNP identifies a specific position (loci) in the genome where at least 1% of the population presents a genomic variation. Depending on where the SNP is located, it may affect how a specific phenotype is expressed. Individually assessing SNPs in order to determine phenotype susceptibility is widely acknowledged to be ineffective (**????**), as the genetic risk associated to a single SNP is usually low. High penetrance genes are the exception and not the norm, as evolution tends to discard them, therefore interactions among several genes need to be considered (**??**).

Genetic interaction between multiple loci when expressing a trait, known as epistasis, was first described in the early 20th century as a deviation from the Mendelian inheritance pattern. As summarized by **?**, eye color determination in *Drosophila* provides a classic example. Epistasis interactions need to be considered in order to find a relation between genotypes and phenotypes. Previous studies have found epistasis in multiple diseases such as breast cancer (**?**) or Alzheimer (**?**). These studies can be divided into two approaches, exhaustive search strategies where all combinations of SNPs are considered, and non-exhaustive search methods where only a reduced number of combinations are actually explored. Exhaustively testing for interaction between genes is a computationally expensive task as it requires checking all the combinations among the SNPs included in the study. Therefore, exhaustive approaches are usually impractical for large datasets as the number of possible combinations grows exponentially. Discarding combinations by following a non-exhaustive strategy, however, may lead to missing key combinations that could provide biological insight to some diseases

In a previous work, we proposed GPU3SNP (?), a tool that is able to exploit several GPUs within the same node to exhaustively search third order epistatic interactions. Although the results show that GPU3SNP achieves high performance and significantly reduces the execution times, the analysis of large GWAS datasets would still require a significant amount of time. For instance, the analysis of a dataset consisting of 50.000 SNPs and 1000 individuals needs nearly 22 hours on a computing node with 4 NVIDIA GTX Titan GPUs. Thus, for large datasets, HPC cluster architectures should be used instead.

[1]Universidade da Corua, Grupo de Arquitectura de Computadores, Departamento de Ingeniería de Computadores, Spain.

Corresponding author:
Christian Ponte, Universidade da Corua, Grupo de Arquitectura de Computadores, Facultad de Informtica, Campus de Elvia, 15071 A Coruña, Spain.
Email: christian.ponte@udc.es

The aim of this work is to provide researchers with a new tool, MPI3SNP, that is able to exploit the computational capabilities of both multi-CPU and multi-GPU clusters. The main contributions of this work are the following:

- **A hybrid implementation that combines Message Passing Interface (MPI) processes and threads to efficiently exploit the cores available on each node of a multicore CPU cluster.**
- **A hybrid MPI/CUDA approach to exploit the resources of a multi-GPU cluster. In this version, MPI is used for internode communications, and the CUDA kernels developed in (?) are used to take benefit from the massive parallelism inside the GPUs of each node.**
- **A workload distribution that avoids communications and synchronizations and balances the computation among the available nodes of the cluster.**

Both, the CPU and GPU cluster implementations are scalable and achieve next to ideal parallel efficiency

The rest of this work is organized as follows. Section **??** presents existing studies related to parallel epistasis detection and offers a brief comment on them. Section **??** explains the bioinformatic method followed by MPI3SNP to find epistatic interactions. Section **?? describes the parallel implementation proposed and how the work is scheduled among the architecture to obtain a good load balance**. Section **??** shows an experimental evaluation of MPI3SNP's **parallel efficiency**. Finally, Section **??** draws some conclusions and offers some lines of future work.

Related work

Multiple authors have addressed the combinatorial problem inherent to the interaction between genes, and the most common approach is to discard a large number of SNPs and only consider a subgroup with biological interest. There is a large variety of methods to identify the interesting SNPs, from following a statistical approach (**???**), to applying clustering strategies (**??**), or using different machine learning techniques (**??**). Nevertheless, initially discarding some SNPs can lead to worse accuracy, as proven by **?** for third-order epistatic interactions.

Similarly to our project, other works focus on performing an exhaustive search and reducing the runtime by exploiting High Performance Computing (HPC) architectures. However, these works are limited to pairwise interactions. The target architectures vary from one to another, including GPUs (**??????**), FPGAs (**?**) and multi-node clusters (**???**).

The number of studies that carry on high-level (higher than two) exhaustive epistasis detection is more reduced. Non-parallel approaches (**?????**) are limited to small datasets, as the number of operations grow exponentially with the number of SNPs involved in the epistasis process. Parallel approaches avoid this limitation by exploiting highly parallelizable architectures in order to reduce the computation time, such as GPUs (**?**) and FPGAs (**?**). However, to the best of our knowledge, there is no previous work that considers cluster architectures for third or higher epistasis level.

Background

The **bioinformatic method of MPI3SNP to detect epistasis** follows the same approach as **?**, which, in turn, is based on information theory measures proposed by **?** and generalized to third-order epistatic interactions. The time complexity of the algorithm is $O(n^3 * m)$, with m being the number of individuals and n the number of SNPs included for each individual.

MPI3SNP takes a collection of SNPs from multiple individuals in PLINK/TPED format, and stores it in memory using a bit-level representation. The SNPs are then combined in groups of three and the frequencies of all their possible combination of values are kept in contingency tables. Once the tables are built, interactions between the SNPs are tested using an entropy-based ranking algorithm. The actual implementation will be discussed in the following sections, giving a more insightful view of the method. It is separated into the data representation and the ranking algorithm used.

Data representation

The program input consists of a large collection of SNPs from different individuals, separated in two case-control classes. The first step of the method consists in constructing the contingency tables of the SNPs triples, which represents the genotype combination frequencies associated with a certain phenotype. These tables used together with a statistical test are proven to be a common, efficient solution for finding associations between a combination of genotypes and a certain phenotype (**?????**).

Creating the contingency tables from the textual representation of the dataset is inefficient, so an intermediate bitwise representation is employed (**?**). Individuals are segregated attending to their phenotype into two tables, one for cases and the other for controls. Each table is made of multiple bit-vectors encoding the genotype information for all SNPs of each individual. A `1` in the bit-vector indicates the presence of that genotype on the SNP of the individual corresponding to that position, and a `0` the absence. Since most SNPs are biallelic, only three different genotypes are considered: homozygous wild (w), heterozygous (h) and homozygous variant (v), labeled as 0, 1 and 2 respectively. This makes a total of three bit-vectors for each table.

Table **??** illustrates with an example the bitwise representation of three SNPs from six different individuals, three belonging to the case group and the other three to the control group. Each position $s_{i,j}$ corresponds to the SNP i from individual j of each group. For instance, the genotype associated to the first SNP ($i = 0$) of the first individual ($j = 0$) from the control group is homozygous wild (`0`).

Contingency tables are computed from this bitwise representation using efficient bit-level operations. Counting the number of individuals that present a combination of genotypes only requires selecting the individuals' genotypes from the bit-vectors, applying a logical `AND` operation between them, and counting the number of `1`'s in the resulting vector.

Table **??** represents the contingency table associated with the genotype information from Table **??**. Each cell of the contingency table indicates how many individuals meet the corresponding genotype combination. The notation $s_i = k$

Table 1. Example of a bitwise representation for three SNPs from six individuals.

	Genotype	$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{2,0}$	$s_{2,1}$	$s_{2,2}$
Cases	0 (w)	1	1	1	0	0	0	0	0	0
	1 (h)	0	0	0	0	0	0	0	0	0
	2 (v)	0	0	0	1	1	1	1	1	1
		$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{2,0}$	$s_{2,1}$	$s_{2,2}$
Controls	0 (w)	1	0	0	0	0	0	1	0	1
	1 (h)	0	0	1	0	0	1	0	0	0
	2 (v)	0	1	0	1	1	0	0	1	0

Table 2. Contingency table example, using the genotype data from previous Table **??**.

			$s_2 = 0$	$s_2 = 1$	$s_2 = 2$
Cases	$s_0 = 0$	$s_1 = 0$	0	0	0
		$s_1 = 1$	0	0	0
		$s_1 = 2$	0	0	3
	$s_0 = 1$	$s_1 = 0$	0	0	0
		$s_1 = 1$	0	0	0
		$s_1 = 2$	0	0	0
	$s_0 = 2$	$s_1 = 0$	0	0	0
		$s_1 = 1$	0	0	0
		$s_1 = 2$	0	0	0
Controls	$s_0 = 0$	$s_1 = 0$	0	0	0
		$s_1 = 1$	0	0	0
		$s_1 = 2$	1	0	0
	$s_0 = 1$	$s_1 = 0$	0	0	0
		$s_1 = 1$	1	0	0
		$s_1 = 2$	0	0	0
	$s_0 = 2$	$s_1 = 0$	0	0	0
		$s_1 = 1$	0	0	0
		$s_1 = 2$	0	0	1

expresses genotype k for the SNP i. Considering that we are using combinations of three SNPs, that each SNP has three possible genotypes, and that tables are segregated by case-control groups, there will be a total of $2 \times 3 \times 3 \times 3 = 54$ resulting frequencies for each combination.

As an example, obtaining the frequency of the combination $(s_0 = 0, s_1 = 2, s_2 = 0)$ for the control group would involve first selecting genotypes 0, 2 and 0 for SNPs 0, 1 and 2 respectively from the control bit-vectors, which would be `100`, `110` and `101`. Then the three vectors would be combined by applying two successive AND operations, `100 AND 110 AND 101 = 100`. Finally, the frequency is obtained by counting the number of `1`'s in the resulting vector: `count(100) = 1`. It is worth noting that many of the intermediate results can be reused in following operations. Using the same example, the combination $(s_0 = 0, s_1 = 2, s_2 = 0)$ shares the operation `100 AND 110` with combinations $(s_0 = 0, s_1 = 2, s_2 = 1)$ and $(s_0 = 0, s_1 = 2, s_2 = 2)$. Reusing intermediate results means reducing the number of AND operations required to calculate a contingency table from 108 to 72.

Ranking algorithms

A look into the values of the contingency table can help to identify whether the SNP combination is able to explain the presence or absence of the disease. Tables with significantly different probabilities comparing the half related to each group indicate that the SNP triple is able to distinguish between cases and controls. Automated programs use metrics in order to quantify the correlation of a combination of SNPs with a certain class. Not every metric is appropriate to find this correlation. Chi-square tests, frequently used for pairwise interactions, are inaccurate when low-frequencies are present (very common in the case of our third-order studies) as not every combination represents a significant population number. Mutual Information (MI), as stated by **?**, is adequate for this purpose.

MI quantifies the amount of knowledge provided by one variable when another is given. The two stochastic variables defined in our problem are the genotype value (homozygous wild, heterozygous and homozygous variant) of a SNP, denoted as X, and the phenotype associated with a sample, denoted as Y. MI is defined by the formula:

$$I(X;Y) = H(X) + H(Y) - H(X,Y)$$

where $H(X)$ and $H(Y)$ are the marginal entropies and $H(X,Y)$ is the joint entropy of X and Y. The marginal and joint entropies are given by the following formulas:

$$H(X) = -\sum_{x \in X} p(x) log(p(x))$$

$$H(X,Y) = -\sum_{x,y} p(x,y) log(p(x,y))$$

with $p(x)$ representing the probability of the random variable X taking the value x, p(y) the probability of the random variable Y taking the value y, and $p(x,y)$ the joint probability of both events.

Although MPI3SNP offers MI as the only ranking algorithm available for both the CPU and GPU architectures, it is implemented following a modular design so that it will be easy in the future to include more metrics if required by the users.

Parallel implementation

Finding third-order epistatic interactions among SNPs is a combinatorial problem, where every combination without repetition of SNPs needs to be tested by creating

Table 3. Example of the balancing technique with 6 SNPs and 3 threads. For each SNP pair, it is shown the assigned thread, the possible non repeating combinations beginning with that pair and its count.

Thread	Pair	Combinations	Count
0	(s_0, s_1)	$(s_0, s_1, s_2), (s_0, s_1, s_3), (s_0, s_1, s_4), (s_0, s_1, s_5)$	4
1	(s_0, s_2)	$(s_0, s_2, s_3), (s_0, s_2, s_4), (s_0, s_2, s_5)$	3
2	(s_0, s_3)	$(s_0, s_3, s_4), (s_0, s_3, s_5)$	2
0	(s_0, s_4)	(s_0, s_4, s_5)	1
1	(s_1, s_2)	$(s_1, s_2, s_3), (s_1, s_2, s_4), (s_1, s_2, s_5)$	3
2	(s_1, s_3)	$(s_1, s_3, s_4), (s_1, s_3, s_5)$	2
0	(s_1, s_4)	(s_1, s_4, s_5)	1
1	(s_2, s_3)	$(s_2, s_3, s_4), (s_2, s_3, s_5)$	2
2	(s_2, s_4)	(s_2, s_4, s_5)	1
0	(s_3, s_4)	(s_3, s_4, s_5)	1

its contingency table and using the ranking algorithm previously explained. The number of possible non-repeating combinations for three elements is given by the following expression:

$$C_{n,3} = \binom{n}{3} = \frac{n!}{3!(n-3)!} = \frac{n*(n-1)*(n-2)}{6}$$

being n the number of SNPs. It can be seen that the number of combinations for a given number of SNPs is proportional to its cube. With the hope of mitigating the inherent cubic time complexity of the problem, **and thus to be able to deal with large datasets,** MPI3SNP is designed to exploit cluster architectures. It is a hybrid two-level parallelization approach supporting nodes with both multicore CPUs and GPUs. On the inter-node level, MPI is used to communicate different nodes for both implementations. On the intra-node level, Pthreads and CUDA are used for the CPU and GPU implementations, respectively.

Multicore CPU clusters

The parallel implementation begins creating one or several MPI processes per node. Every MPI process reads all the input data (kept in memory using the previously explained bitwise representation). This data replication is beneficial in terms of performance, since distributing it would lead to communication bottlenecks when creating the contingency tables. The consequent memory overhead due to replication is affordable on current multicore clusters. For instance, storing the genotype data of the biggest dataset employed in the experimental evaluation (6,300 SNPs and 3,200 samples) only requires 58MB of memory.

Then, each MPI process launches as many threads as available cores and assigns to each one a fraction of the workload to compute. The best balancing choice, in terms of number of operations, would be using the combinations of three SNPs as our distribution unit for the threads, and thus assigning the same number of SNP triples to each thread. However, this would not be the most efficient approach as many intermediate results (logical AND operations) can be reused between SNP triples that share two common SNPs.

The distribution unit chosen instead is a SNP pair. Table **??** gives an example of this approach, distributing all SNP triplets resulting from 6 SNPs among 3 threads. Pairs are assigned following a round robin distribution. Then, from each pair it is possible to calculate all non-repeating combinations that begin with it, maximizing the reuse of logical operations. The example in Table **??** uses a very small number of SNPs and some differences in the number of combinations per thread can be noted. However, with a more realistic number of SNPs, the differences are unnoticeable in relation with the total number of combinations assigned to each thread, as will be seen in the posterior experimental evaluation.

Remark that this balancing approach is static and completely distributed, involving only a single collective communication at the very end of the execution in order to gather the results into the only MPI process that will write the output.

GPU clusters

The GPU parallelization strategy follows the same approach as the CPU one with two variations.

1. An MPI process is created for each available GPU device and the workload is partitioned among all MPI processes following the same pairwise distribution explained in the previous subsection. Thus, each MPI process is responsible of transferring its fraction of the workload to its associated GPU and calling the appropriate CUDA kernels to compute the corresponding triples. However, since memory is more limited in the GPUs than in a CPU node, SNPs combinations are transfered in blocks following an iterative procedure until their fraction of the computation is completed. Results are kept in the GPU memory until the end, and then gathered into a single MPI process.
2. As explained in the previous section, MPI3SNP employs a bitwise data representation to accelerate the contingency table construction step. In the CPU version, all the data is consecutively stored by SNP to exploit cache locality. Nevertheless, a data transposition is applied in the GPU version in order to increase the coalescence and thus the performance of the device memory accesses.

More details about the data transposition and its benefits, as well as the CUDA kernels and the iterative SNP block transfer can be found in the original work of **?**.

Experimental evaluation

Performance evaluation consists of a series of tests using different configurations and problem sizes in order to measure the **parallel efficiency** and scalability of both multi-CPU and multi-GPU implementations. The dataset used in all executions is the same, and different sizes are obtained by trimming either the number of individuals (from the original 6,400 samples to 3,200, 1,600 and 800) or the number of SNPs included (from 40,000 to 6,300, 5,000, 4,000 and 3,200 SNPs), making a total of 12 different tests. It is a synthetic dataset created by **?** following a uniform distribution of the genotypes, and it is available at the MPI3SNP's Github repository.

The evaluation was performed on two different clusters, Pluton and Finisterrae II, described in Table **??**. A very similar software environment was used in both clusters, composed by the C++ compiler GCC 5.3, the CUDA compiler NVCC 8.0 and the MPI library OpenMPI versions 1.8 and 1.10 in the Pluton and Finisterrae II clusters, respectively.

It is worth mentioning that both the NVIDIA Tesla K80 and the NVIDIA GRID K2 are dual GPU cards, meaning that each one contains two GPUs, thus making a total of 8 GPUs in each cluster queue.

Evaluation on CPU multicore clusters

Since both clusters have dual-socket CPU nodes, the best configuration of number of processes and threads was first checked by using different combinations of MPI processes and threads on a single node. The smallest dataset (3,200 SNPs and 800 samples) was used to keep runtimes manageable. Each test was repeated three times and, as a non-significant variability among executions was observed, the mean time was considered. The results can be found in Table **??**, where no significant difference can be seen. Therefore, the configuration adopted for all CPU tests was the most intuitive: one MPI process per node with one thread per core.

Speedup, calculated as the fraction between the sequential and parallel runtime, is used to measure the performance of the parallel implementation. Table **??** shows the sequential runtime for the different datasets considered, i.e., executing MPI3SNP on one core (one process with only one thread). Some experiments, underlined in Table **??**, could not be completed on a single core on Pluton due to the time constraint of this cluster (3 days per execution). In those scenarios, the runtime is estimated as the product of the previous largest execution time (corresponding to 6300 SNPs and 800 samples) and the linear slopes calculated from previous increases between the 800 samples, and 1600 and 3200 ones (values 1.446 and 1.906), as the time scales linearly with the number of samples.

Figures **??** and **??** compare the speedups achieved as the number of nodes increases. The ideal speedups are annotated in the graphs as the horizontal grids, calculated as the product between the number of nodes used and the number of cores per node. The first figure (**??**) corresponds to the Pluton cluster, showing a linear scalability and some instances of superlinearity. The second figure corresponds to the Finisterrae cluster and, in this case, speedups are hindered by the Intel Turbo Boost technology, since the single-threaded execution is running at 3.30Ghz and the multi-threaded one at 2.90GHz. If the CPU frequencies were kept constant, the resulting speedups would be in line with those of Pluton.

These good results are mainly due to the replication of the SNP information avoiding communications among processes, and the proposed workload distribution that allows the reuse of logical operations and ensures a good load balancing without any synchronization overhead.

Table **??** shows the workload balance among threads. Each cell from the table represents, for each configuration, the maximum relative deviation from the number of triples that should be equally assigned to each thread, since the largest deviation will be determining the global runtime. The maximum relative deviation is calculated as the maximum difference to the mean number of operations divided by the mean. The scheduling strategy implemented is very effective as the largest deviation is 0.00418, which represents a negligible effect on the runtime.

To give some perspective, remark that the sequential runtime for the dataset with 6,300 SNPs and 800 individuals on the Finisterrae cluster is roughly 62 hours, and it is reduced to under 6 minutes by exploiting 32 nodes of the cluster. Moreover, datasets with 6,300 SNPs and 1,600 or 3,200 individuals could not be completed on a single CPU in Pluton due to the maximum execution time allowed by the cluster administrators (three days), while the parallel version with eight nodes only needs around 33 and 46 minutes, respectively.

Evaluation on GPU Clusters

Following an analogous approach, speedups are used to evaluate the performance of MPI3SNP on heterogeneous clusters, where each node contains one or several GPUs. **In this case, for each GPU-cluster we calculated the speedups using as reference the execution time in a single GPU of that cluster (which is equivalent to execute the version implemented by ? on that GPU).** Table **??** shows these baseline runtimes. GPU affinity does not affect the performance, thus no particular binding options was used.

Figures **??**, **??** and **??** compare the speedups obtained when increasing the number of GPUs. Again, the ideal speedups are annotated as horizontal grids, which would be the number of GPUs used. The speedups obtained with all three different GPUs can be considered almost linear.

In terms of load balance, the workload is partitioned among the MPI processes (one per GPU). Thus, the number of units on which the workload has to be divided is smaller, and the load balance is even better than in previous section. Using the same metric as before, the maximum difference to the mean number of operations divided by the mean, the deviation values range from 0 to 6.05×10^{-7}.

The thousands of cores available in a GPU make it adequate to accelerate applications that are highly parallelizable. This is the case for epistatis detection. For instance, using the largest dataset size (6,300 SNPs and 3,200 samples), eight GPUs reduce the compute time from approximately 65, 92 and 30 minutes for K20m, K2 and K80 respectively, to 9, 12 and 4.

Table 4. Hardware characteristics of the clusters used in the performance evaluation.

Cluster	Node configurations			
	CPUs/node	Cores/node	GPUs/node	Nodes
Pluton	2x Intel Sandy Bridge-EP 2660	16	NVIDIA Tesla K20m	8
Finisterrae II	2x Intel Haswell 2680v3	24	-	32
	2x Intel Haswell 2680v3	24	2x NVIDIA Tesla K80	2
	2x Intel Haswell 2650v3	20	NVIDIA GRID K2	4

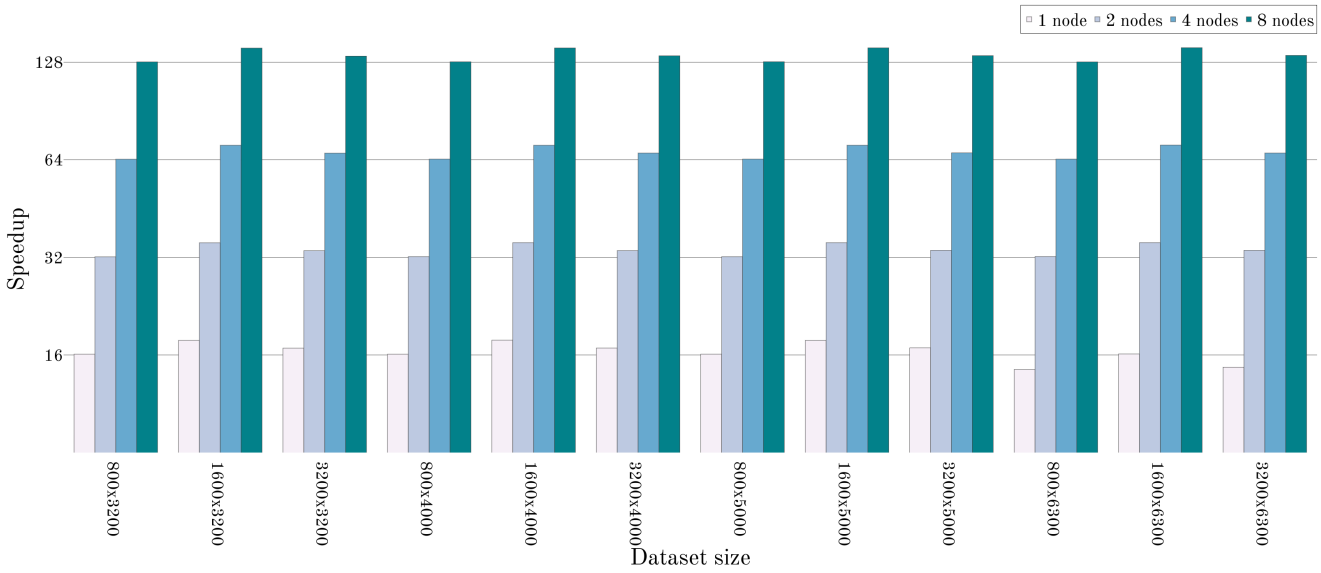


Figure 1. CPU speedups on the Pluton cluster for different dataset sizes and number of nodes.

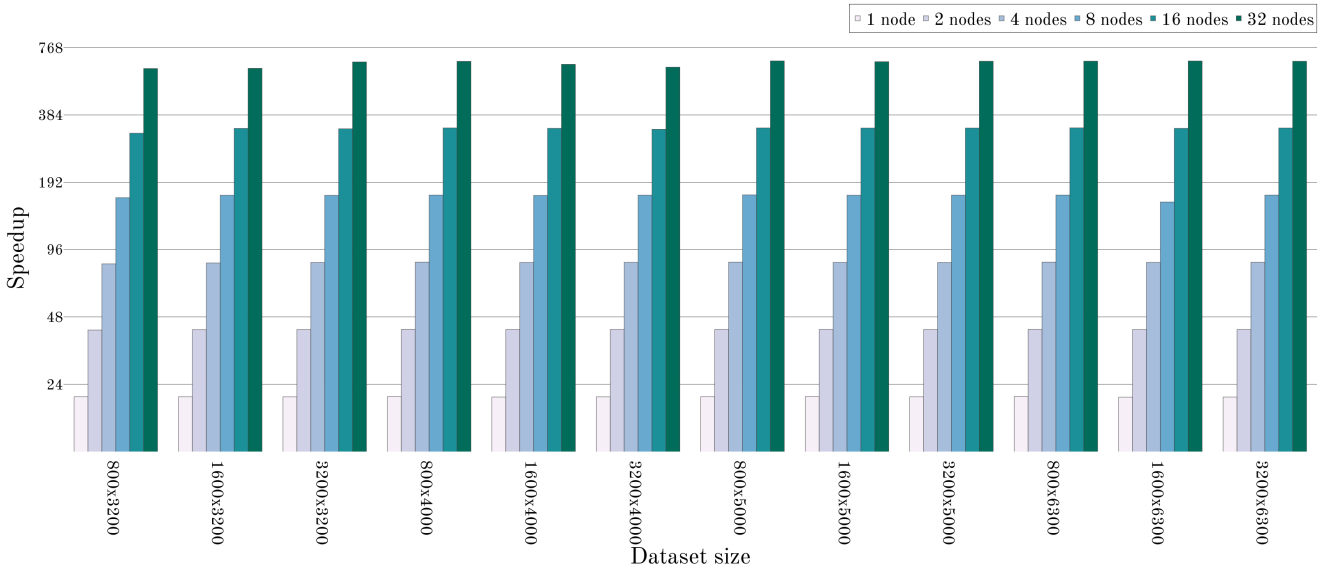
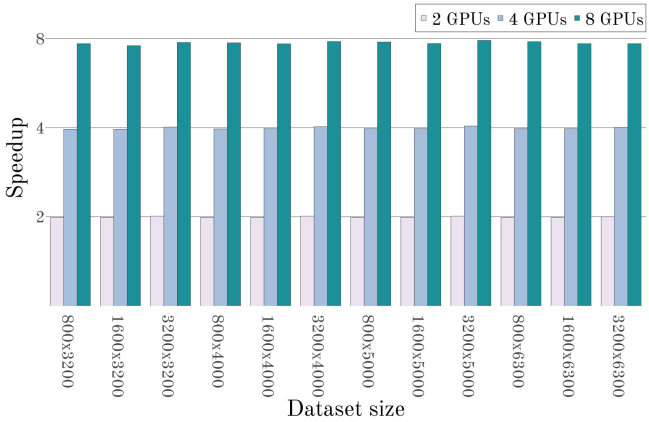


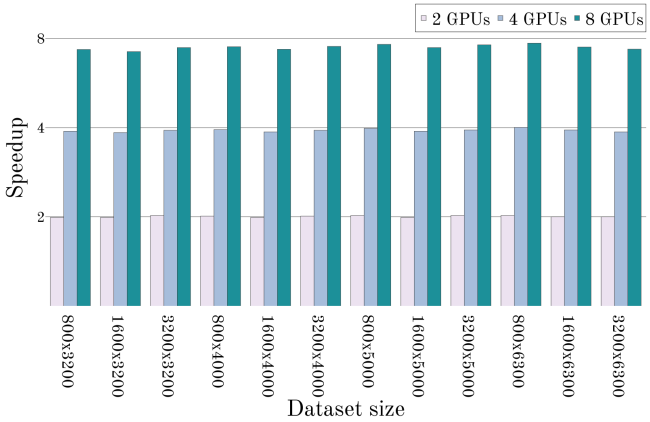
Figure 2. CPU speedups on the Finisterrae II cluster for different dataset sizes and number of nodes.

Conclusions and future work

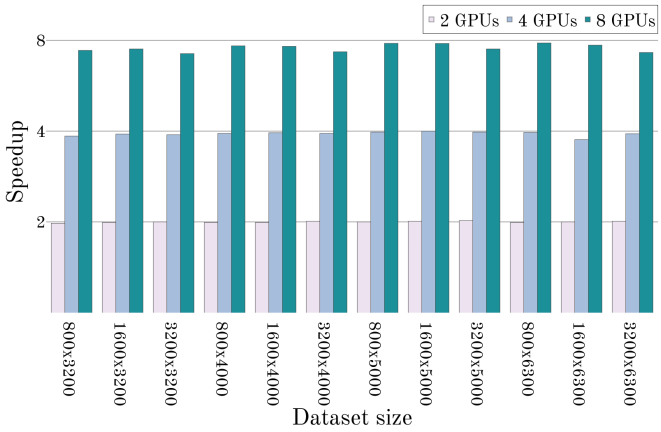
The principal limiting factor of epistatic searches is the exponential growth of the number of combinations with the number of SNPs involved on the interaction. Instead of reducing the scope of the search by means of a previous filtering stage, or limiting the usability to small datasets, our approach relies on cluster architectures to overcome this exponential growth and achieve a manageable runtime. The current implementation obtains next to linear speedups thanks to the use of a static distribution of the workload that avoids communications and synchronizations and provides an almost perfect load balance.



(a) Results for NVIDIA K20m GPUs.



(b) Results for NVIDIA K2 GPUs.



(c) Results for NVIDIA K80 GPUs.

Figure 3. Speedups using a variable number of GPUs and diffferent datasets.

Table 5. CPU affinity test combining different number of processes and threads, using the dataset with 3,200 SNPs and 800 samples.

	Processes	Threads	Runtime
Pluton	1	16	1,795s
	2	8	1,796s
	4	4	1,795s
	8	2	1,794s
	16	1	1,788s
Finisterrae II	1	24	760s
	2	12	760s
	4	6	760s
	12	2	760s
	24	1	758s

Table 6. Sequential runtime of MPI3SNP on both clusters. Underlined data means estimated runtime.

SNPs	Samples	Pluton	Finisterrae II
3,200	800	25,672s	15,902s
3,200	1,600	37,115s	20,791s
3,200	3,200	48,935s	29,323s
4,000	800	50,193s	31,132s
4,000	1,600	72,596s	40,644s
4,000	3,200	95,631s	57,274s
5,000	800	98,004s	60,784s
5,000	1,600	141,740s	79,432s
5,000	3,200	187,006s	111,887s
6,300	800	196,428s	121,842s
6,300	1,600	*284,035s*	159,097s
6,300	3,200	*374,391s*	224,132s

Take as an example the largest dataset tested, composed by 3,200 samples of 6,300 SNPs each. When using 8 nodes of 16 CPU cores each, the obtained speedup is 134 compared to a completely sequential execution. On the GPU implementation the same level of **parallel efficiency** is achieved, as the total runtime is reduced 7.69 times when using 8 NVIDIA K20m GPUs against one.

MPI3SNP, however, has its limitations. The current implementation only supports third-order epistatic searches and, given the promising performance results, it would be interesting to see how many SNPs can we combine before the runtime becomes unmanageable. In terms of the ranking algorithms, Mutual Information is the only one currently offered and it is of great interest to expand the number of evaluation metrics, keeping in consideration the possibility of variable number of SNPs in the interaction. These two lines will be addressed as future work.

MPI3SNP is open-source software and available to download at the following Github repository: https://github.com/chponte/mpi3snp.

Table 7. Maximum relative deviation from the mean number of triples assigned to each thread, using multiple number of nodes and different dataset sizes for both clusters.

	Pluton	16	32	64	128		
SNPs	3200	0	0	0	7.41e-06		
	4000	0	1.23e-07	4.76e-05	1.58e-04		
	5000	3.83e-06	9.59e-06	4.02e-05	1.03e-04		
	6300	2.42e-06	7.24e-06	2.65e-05	7.03e-05		
	Finisterrae II	**24**	**48**	**96**	**192**	**384**	**768**
SNPs	3200	6.41e-04	6.68e-04	7.83e-04	9.39e-04	1.79e-03	4.18e-03
	4000	5.08e-04	5.26e-04	5.80e-04	6.47e-04	1.31e-03	3.21e-03
	5000	4.06e-04	4.23e-04	4.69e-04	5.61e-04	7.75e-04	1.89e-03
	6300	3.21e-04	3.32e-04	3.52e-04	4.02e-04	6.37e-04	1.22e-03

Table 8. Runtime of MPI3SNP on a single GPU.

Number of SNPs	Number of samples	K20m	K2	K80
3,200	800	138 s	200 s	104 s
3,200	1,600	220 s	315 s	149 s
3,200	3,200	528 s	748 s	241 s
4,000	800	268 s	393 s	203 s
4,000	1,600	428 s	616 s	290 s
4,000	3,200	1,013 s	1,459 s	469 s
5,000	800	522 s	774 s	397 s
5,000	1,600	833 s	1,208 s	570 s
5,000	3,200	2,027 s	2,866 s	919 s
6,300	800	1,039 s	1,550 s	785 s
6,300	1,600	1,649 s	2,407 s	1112 s
6,300	3,200	3,943 s	5,519 s	1775 s

Acknowledgements

We gratefully thank Galicia Supercomputing Center for providing access to the FinisTerrae-II supercomputer.

Funding

This research was supported by the Ministry of Economy and Competitiveness of Spain and FEDER funds of the EU [Project TIN2016-75845-P (AEI/FEDER, UE)]; the Xunta de Galicia and FEDER funds of the EU (Centro Singular de Investigación de Galicia) [grant number ED431G/01]; Consolidation Program of Competitive Research [grant number ED431C 2017/04]; and the FPU Program of the Ministry of Education of Spain [grant number FPU16/01333].

Declaration of conflicting interests

The Authors declare that there is no conflict of interest.