FaST-LMM for Two-Way Epistasis Tests on High Performance Clusters

Héctor Martínez[1,*]   Sergio Barrachina[1]   Maribel Castillo[1]
Enrique S. Quintana-Ortí[1]   Jordi Rambla De Argila[2]   Xavier Farré[2]
Arcadi Navarro[2]

May 29, 2018

Abstract

We introduce a version of the epistasis test in FaST-LMM for clusters of multi-threaded processors. This new software maintains the sensitivity of the original FaST-LMM while delivering an acceleration that is close to linear on 12–16 nodes of two recent platforms, with respect to the improved implementation of FaST-LMM presented in an earlier work. This efficiency is attained via several enhancements on the original single-node version of FaST-LMM, together with the development of an MPI-based version that ensures a balanced distribution of the workload as well as a multi-GPU module that can exploit the presence of multiple graphics processing units (GPUs) per node.

Key words: Epistasis, FaST-LMM, Genome-Wide Association Studies (GWAS), multicore processors, GPUs, clusters of computers.

1   Introduction

After years of accumulating technological improvements, we have finally embraced the post-omics era where methodological advances are quickly providing geneticists with tools to analyze massive genome-wide data sets. Recent genomic studies indicate that each healthy individual carries hundreds of loss-of-function variants as well as tens of thousands of other genomic variants in the coding and regulatory regions of their genomes. In this line, the aggregated analysis of Genome-Wide Association Studies (GWAS) (Lappalainen *et al.*, 2015) are accumulating evidence at increasing pace, providing medically-relevant predictions of phenotypic outcomes from genomic profiles based on Single Nucleotide Polymorphisms (SNPs) (Abraham

---

[1]Department of Computer Science and Engineering, Jaume I University, 12006–Castellón, Spain. {martineh,barrachi,castillo,quintana}@uji.es

[2]Department of Experimental and Health Sciences, Pompeu Fabra University, 08002–Barcelona, Spain. jordi.rambla@crg.eu, {xavier.farre,arcadi.navarro}@upf.edu

et al., 2014; Coleman *et al.*, 2016). Nonetheless, a problem of GWAS stems from the fact that most associated SNPs have been detected within a framework that assumes additive interaction and tests each SNP separately from others. In contrast, recent evidence suggests that the assumption of additivity is sometimes not fulfilled in complex organisms (Zuk *et al.*, 2012; Wan *et al.*, 2010; Hemani *et al.*, 2014). Thus, in order to evaluate the interplay that genetic variants have on phenotypes, we must compare linear additivity with more complex interactions –or *epistasis*– among markers, assessing which are the sets of relationships between SNPs that produce the most adequate genomic risk scores in order to predict disease status or treatment outcome.

There exist several software packages for genomic-wide epistasis analysis; see Table 1 for a representative selection. Among these, the epistasis test module in FaST-LMM (Lippert *et al.*, 2013, 2011) offers high sensitivity at the cost of a considerable execution time. This method inspects all possible interactions between two SNPs to detect those that best predict the phenotype but, from the computational perspective, it is far more expensive than other methods.To tackle this higher cost, the original epistasis test in FaST-LMM can exploit the presence of multiple cores on a server using Python processes.

In Martínez *et al.* (2017), we introduced an enhanced version of the epistasis test module in FaST-LMM that maintained its sensitivity while delivering an acceleration factor close to 7.5× on a (single) computer server equipped with a high-end graphics processing unit (GPU). In this work, we propose an extension of that implementation that considerably accelerates the FaST-LMM epistasis test, while preserving its sensitivity, using all the resources of a cluster of multicore computers, equipped with multicore processors and one or more GPUs.

2   Analysis of the FaST-LMM epistasis test

In this section we briefly describe the epistasis test module in FaST-LMM 0.2.31, and the acceleration strategies introduced in Martínez *et al.* (2017). These two components are the reference baseline for the extensions described later, in Section 3.

FaST-LMM epistasis test.   In order to identify whether there exists an epistatic interaction between two SNPs, the epistasis test module in FaST-LMM computes, for each possible combination of two SNPs, the p-value of the chi-square test associated with the difference between the log-likelihoods of two alternative hypotheses.

The original workflow of the epistasis test in FaST-LMM is divided into four stages (S1–S3 and RR in Figure 1). The first stage, S1, is performed by a single process that assembles the initial matrices, splits the set of all SNP pairs into packages of fixed size, spawns a user-specified number of processes, and distributes the packages among those. The assembly in this stage requires the computation of matrix-matrix multiplications, singular value decompositions, and the solution of eigenvalue problems. Next, each spawned process operates on stages S2 and S3, retrieving the data corresponding to its SNP-pair packages, computing the p-values associated to each pair, and storing these intermediate results to disk. Here, the second stage again involves a number of matrix-matrix multiplications. The third stage requires the computation of the log-likelihood of the alternative hypotheses, together with p-value of the chi-square test for the difference of the previous log-likelihoods. Finally, the last stage, RR, is executed by a single process that combines the partial results into a single data structure. (See Martínez *et al.* (2017) for details.)

FaST-LMM epistasis test enhancements. In Martínez *et al.* (2017) we evaluated the parallel performance of the original epistasis test module in FaST-LMM, and proposed some enhancements that were specifically designed to speed-up the most time-consuming operations: i) a large-scale matrix-matrix multiplication in S2; ii) the log-likelihood tests in S3; and iii) the insertion of the local results in a Python database in S3. Our modifications to FaST-LMM targeted these performance bottlenecks by respectively: i) off-loading the costly matrix-matrix multiplication to a GPU; ii) removing some clutter code in the log-likelihood test; and iii) collapsing the storage of the epistasis test results into a single point. These three improvements will be hereafter referred to as GPU, LC+LOG, and D4D, respectively. The experimental analysis in Martínez *et al.* (2017), on a server equipped 2 Intel Xeon E5-2620v4 8-core processors (16 cores), 32 GiB of memory, and an NVIDIA P100 "Pascal" GPU, reported a speed-up factor that was around 7.5× with respect to the original implementation of FaST-LMM running on 16 cores.

3 Extending FaST-LMM epistasis test to multi-GPU and clusters

In this section we introduce two extensions of the enhanced FaST-LMM epistasis test module for clusters of nodes equipped with multi-core processors and multi-GPUs, available under an Apache License 2.0 on `https://github.com/epiproject/FaST-LMM-HPC/`.

3.1 Multi-GPU extension

One of the enhancements to the standard implementation of the epistasis test in FaST-LMM proposed in Martínez *et al.* (2017) consisted in off-loading some computations of stage S2 to a (single) GPU. As multiple processes are simultaneously executing stages S2 and S3, this can lead to conflicts in the access to the GPU. To alleviate this bottleneck, we have developed a multi-GPU extension of the enhanced FaST-LMM epistasis test that can leverage more than one GPU, if available in the server.

The multi-GPU extension distributes the workload among all available GPUs on a server by cyclically attaching them to each spawned process. In addition, the GPU memory manager implemented in Martínez *et al.* (2017) was extended to suspend a process request until enough memory is available in the GPU that is assigned to it. A more detailed explanation appears in Section 1 of the supplementary material.

3.2 Cluster extension

This extension distributes the SNP-pairs packages among the nodes of a cluster, instructs the processes running on each node to operate on a disjoint set of SNP-pairs packages, and finally gathers the results. It is implemented using MPI[1] to create $np(+1)$ MPI ranks (processes) (one per node), which will in turn spawn p Python processes each.

We have implemented the following static and dynamic workload scheduling strategies to distribute the input data (workload) among the cluster nodes.

---

[1] `http://www.mpi-forum.org`

Static schedule. The SNP-pairs packages are evenly distributed among the $np \times p$ processes (workers) on the system. Thus, for a dataset with k SNPs and w workers, each worker analyzes about $k^2/(2wp)$ packets, with p standing for the packet size in terms of number of SNP pairs. In this solution, each rank can independently compute which set of SNP-pairs packages it should process from the dataset size, the package size, the number of processes per node, the number of nodes, and its rank identifier.

Dynamic schedule. The SNP-pairs packages are assigned to the MPI rank/nodes on demand. We have implemented this schedule by creating $np + 1$ MPI ranks, where node 0 contains two MPI ranks, one of them —the manager— is in charge of the work distribution, while the remaining np worker MPI ranks, one of them in node 0, perform the actual computations (see Figure 2). The manager first computes the total number of SNP-pairs packages and the row and column index of the first pair of each SNP-pairs package. Next, it carries out an initial distribution of SNP-pairs packages among the workers, and then waits until a worker asks for a new task. In response to this event, the manager assigns a bundle of pending SNP-pair packages to the demanding worker. This is repeated until the pending queue is emptied. In this solution, only the row and column indexes of the first SNP-pair for each SNP-pairs package is transmitted, yielding a negligible communication overhead. Due to its low overhead, dynamic scheduling should always be the preferred option. This strategy is selected for the next experiments unless otherwise stated.

4 Experimental results

In this section we describe the setup for the experiments and perform the experimental evaluation of the extensions.

4.1 Experimental setup

Our experiments employ a sample from the Welcome Trust Case Control Consortium (WTCCC) bipolar disorder (The Wellcome Trust Case Control Consortium, 2007), which provides a total of 455,086 SNPs for n=4,804 individuals. To reduce the execution time, we have only used a fraction of the original SNP dataset, consisting of $k = 2,000$ to 32,000 SNPs (or 1,999,000 to 511,984,000 SNP pairs). This reduced sample of SNPs is enough to analyze the performance of the proposed extensions, as FaST-LMM splits the work in packages comprising 1,000 SNP pairs each (the default package size in the FaST-LMM epistasis test) and, once $k \geq n$, the matrices sizes are constrained to the number of individuals. Hence, an execution involving a dataset with a larger number of SNPs should roughly increase the execution time linearly on the number of SNP pairs ($\approx k^2/2$). This linear relationship was already observed in Martínez *et al.* (2017). Furthermore, the execution time does not depend on the specific subset of SNP pairs being selected. Therefore, we can expect that the conclusions extracted from the experiments in this paper, with these (fragments of) datasets, carry over to other cases with a larger number of SNPs or a different set.

The experiments were performed using the next two cluster platforms:

- *Piz Daint* at the Swiss National Supercomputing Centre (CSCS). This cluster contains 5,272 nodes, each with an Intel Xeon E5-2690 v3 at 2.60 GHz with 12 cores, 64 GiB

RAM and an NVIDIA Tesla P100 with 16 GiB. The system interconnect is based on Aries routing and ASIC communications, and features a Dragonfly network topology.

- *Minotauro* at the Barcelona Supercomputing Center (BSC). It is an heterogeneous cluster with 39 nodes, each equipped with two NVIDIA K80 GPU cards, 2 Intel Xeon E5–2630 v3 (Haswell) 8-core processors (each core at 2.4 GHz with 20 MiB L3 cache), and 128 GiB of main memory. The cluster interconnect employs the Mellanox FDR Infiniband technology.

We spawned one Python process per hardware core. The operating system was RedHat Linux 2.6.32. The following software modules were used: FaST-LMM 0.2.31, Python 2.7.13, PyCUDA 2016.1.2, Scikit-CUDA 0.5.1, Intel MKL Update 11 (ICC 17.0.1), and NVIDIA CUBLAS 8.0. The MPI versions were: MPI Slurm 17.02.7 on Piz Daint and Open MPI 1.6.5 on Minotauro.

We have repeated all experiments 5 times and the coefficient of variation (CV) (Everitt and Skrondal, 2010) was always below 10% and rapidly decreased with the number of SNPs per node (see Section 2 of the supplementary material). For example, with 6,000 or more SNPs per node, the CV was always below 3%, and in most cases, around 1%.

4.2 FaST-LMM enhancements and multi-GPU version

We have first evaluated the speed-ups due to the enhancements introduced in Martínez *et al.* (2017) on a single node of each one of the aforementioned clusters. As Minotauro includes 4 GPUs per node, we have also tested the multi-GPU extension on this platform.

Figure 3 shows the execution time obtained for different numbers of SNP pairs and the accumulative acceleration attained with each additional enhancement over the original FaST-LMM epistasis test. As anticipated in the previous subsection and reported in Figures 3a) and 3b), when the number of SNPs that are processed exceeds the number of individuals, the execution time grows linearly with the number of SNP pairs. When processing pairs of 8,000 SNPs, the modifications to the original FaST-LMM test reduce the execution time from 9 hours 13 minutes to 1 hour 12 minutes on Piz Daint, and from 4 hours 50 minutes to 41 minutes on Minotauro.

Figures 3c) and 3d) show that applying the enhancements described on Martínez *et al.* (2017) yield speed-ups around 7.6 on Piz Daint and 5.5 on Minotauro. Here, the speed-up of the accelerated FaST-LMM epistasis test on Piz Daint is higher than that observed on Minotauro. The reason for this behavior is that one of the enhancements off-loads a key computation to the graphics accelerator, and Piz Daint integrates a faster GPU than Minotauro.

Figure 3d) also shows that applying the new multi-GPU extension on Minotauro, the speed-up is increased, on average, to 6.9. Although this could seem a small improvement (only a 1.25× factor over the acceleration of 5.5 stated earlier), we must take into account that only the matrix-matrix multiplications in stage S2 benefits from this extension.

4.3 Cluster version and workload schedules

We next evaluate the two workload schedules, static and dynamic, using the extension of the enhanced FaST-LMM epistasis test for clusters, with a 16,000–SNP dataset on the Piz Daint

cluster only. (We do not expect to gain different insights from performing a similar evaluation on the Minotauro system.) For this purpose, we run the extension using different numbers of nodes, and measure the execution time of the test on each node when using an static or a dynamic workload distribution. Table 2 shows the minimum and maximum execution times among all nodes. These results show that the static schedule exhibits a small deviation in the workload distribution balance. This variability is due to differences in the efficiency when the GPU(s) is shared between all processes of a node. In comparison, the dynamic schedule produces an even better workload balance, and its overhead is negligible compared with the execution time of the application.

At this point, it should be stressed that, although both workload schedules obtain close results in this experiment, the dynamic configuration will lead to much better results in case the cluster integrates a collection of heterogeneous nodes.

4.4   Cluster extension: performance and scalability

In order to assess the *strong scalability* of the cluster extension, we have employed from 1 to 16 nodes of the Piz Daint and Minotauro clusters to perform two epistasis tests, with 8,000 and 16,000 SNPs. The execution time of these experiments and their speed-ups are presented in Figure 4. The speed-up factors reported in our manuscript correspond to the acceleration with respect to the improved version of the FaST-LMM code presented in our earlier work (in Martínez *et al.* (2017)). Note that such version is parallel and exploits *all the resources from the node*, for example, all 12 cores and the GPU on the Piz Daint system. The cost of the enhanced version of FaST-LMM with 16,000 SNPs on Piz Daint is reduced from 4 hours 39 minutes when executed on 1 node to 20 minutes when executed on 16 nodes; see Figure 4a). Hence, we attain a speed-up around $14.1\times$ on 16 nodes of Piz Daint, see Figure 4c); and a similar acceleration factor on the same number of nodes of Minotauro ($14.6\times$), see Figures 4b) and 4d). These plots also show that the speed-up is very close to the ideal one when 8 or fewer nodes are employed; for a larger number of nodes, the speed-up approximates the ideal one as the number of SNPs being processed is increased (which should be the case in most epistasis test scenarios).

Finally, we have also performed a *weak scalability* analysis. In this type of study, we evaluate the variation of execution time as we increase the number of nodes while maintaining *the problem size per node constant*. For this purpose, as the execution time should be linear with the number of SNP pairs, we fix the problem size per node to 31,996,000 SNP pairs (3,000 SNPs), and then we vary the number of nodes. The results on the Piz Daint and Minotauro clusters with up to 16 nodes are shown in Table 3. The average execution time per node on Piz Daint was 4,281 seconds with a standard deviation of 21 seconds. The same experiment on Minotauro offered 2,590.73 seconds with a standard deviation of 152 seconds. These results show the notable weak scalability of the parallel epistasis solution, for up to 12–16 nodes and 192–256 cores. The extrapolation of the experiments in Martínez *et al.* (2017) and the cluster extension of FaST-LMM in this work shows that performing the Epistasis analysis for a very large dataset, with $k = 500,000$ SNPs, would require an execution time of about 50 months when using all the resources of a single node of Piz Daint. Assuming the weak scalability efficiency is maintained, the same dataset could be analyzed in around 19 hours using our cluster version on 320 nodes of this platform. Moreover, even if the weak scalability efficiency was degraded to only a 50%, this dataset could still be analyzed in around 30 hours.

5 Conclusions

We have evolved our multi-threaded software for epistasis tests based on FaST-LMM to produce a parallel version that can leverage the computational resources of a cluster of nodes equipped with one or more GPUs per node. The new parallel epistasis framework delivers remarkable accelerations when executed on a moderate number of nodes from two current supercomputers. The parallel extension relies on the standard MPI for communication and well-known high performance libraries for dense linear algebra so it should be portable, with good efficiency, to other recent high performance clusters. As the number of nodes is increased beyond a few dozens, we can expect that the distribution of the input data through the filesystem becomes a performance bottleneck, asking for the application of input/output optimization techniques.

Acknowledgments

References

Abraham, G., Tye-Din, J. A., Bhalala, O. G., Kowalczyk, A., Zobel, J., and Inouye, M. 2014. Accurate and robust genomic prediction of celiac disease using statistical learning. *PLoS Genet* 10, e1004137.

Coleman, C., Quinn, E. M., Ryan, A. W., Conroy, J., Trimble, V., Mahmud, N., Kennedy, N., Corvin, A. P., Morris, D. W., Donohoe, G., *et al.* 2016. Common polygenic variation in coeliac disease and confirmation of znf335 and nifa as disease susceptibility loci. *European Journal of Human Genetics* 24, 291–297.

Everitt, B. and Skrondal, A. 2010. *The Cambridge Dictionary of Statistics, Fourth Edition.* BusinessPro collection. Cambridge University Press.

Hemani, G., Shakhbazov, K., Westra, H.-J., Esko, T., Henders, A. K., McRae, A. F., Yang, J., Gibson, G., Martin, N. G., Metspalu, A., *et al.* 2014. Detection and replication of epistasis influencing transcription in humans. *Nature* 508, 249.

Lappalainen, I., Almeida-King, J., Kumanduri, V., Senf, A., Spalding, J. D., Saunders, G., Kandasamy, J., Caccamo, M., Leinonen, R., Vaughan, B., *et al.* 2015. The european genome-phenome archive of human data consented for biomedical research. *Nature genetics* 47, 692–695.

Lippert, C., Listgarten, J., Davidson, R. I., Baxter, J., Poon, H., Kadie, C. M., and Heckerman, D. 2013. An exhaustive epistatic SNP association analysis on expanded wellcome trust data. *Scientific reports* 3, 1099.

Lippert, C., Listgarten, J., Liu, Y., Kadie, C. M., Davidson, R. I., and Heckerman, D. 2011. Fast linear mixed models for genome-wide association studies. *Nature methods* 8, 833–835.

Martínez, H., Barrachina, S., Castillo, M., Quintana-Ortí, E. S., De Argila, J. R., Farré, X., and Navarro, A. 2017. Accelerating FaST-LMM for epistasis tests. In *Algorithms and Architectures for Parallel Processing: 17th International Conference, ICA3PP 2017, Helsinki, Finland, August 21-23, 2017, Proceedings*, pages 548–557. Springer International Publishing.

The Wellcome Trust Case Control Consortium 2007. Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls. *Nature* 447, 661–678.

Wan, X., Yang, C., Yang, Q., Xue, H., Fan, X., Tang, N. L., and Yu, W. 2010. Boost: A fast approach to detecting gene-gene interactions in genome-wide case-control studies. *The American Journal of Human Genetics* 87, 325–340.

Zuk, O., Hechter, E., Sunyaev, S. R., and Lander, E. S. 2012. The mystery of missing heritability: Genetic interactions create phantom heritability. *Proceedings of the National Academy of Sciences* 109, 1193–1198.
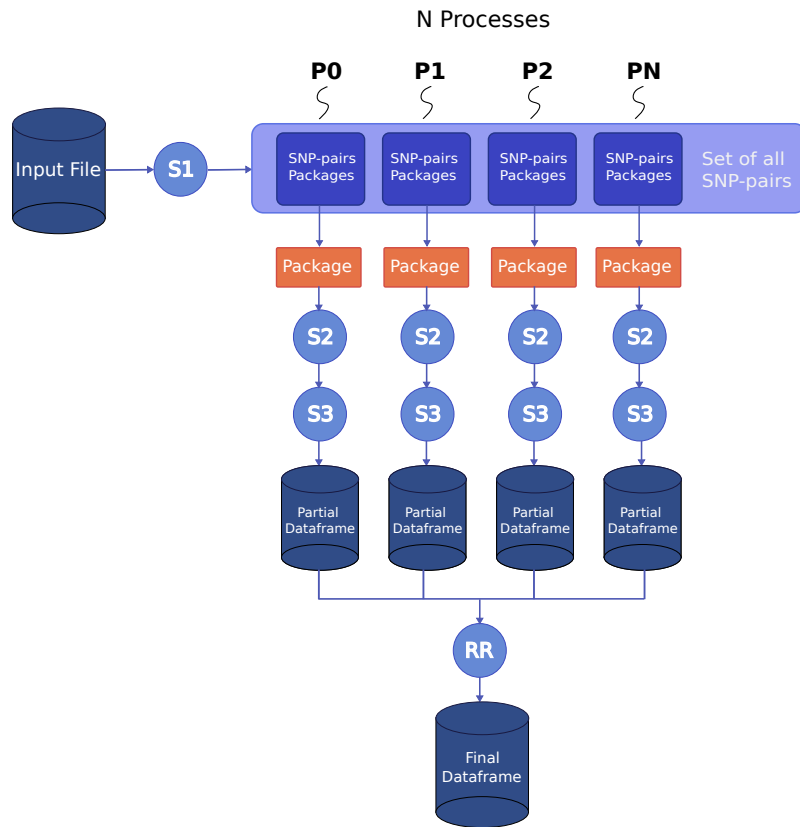
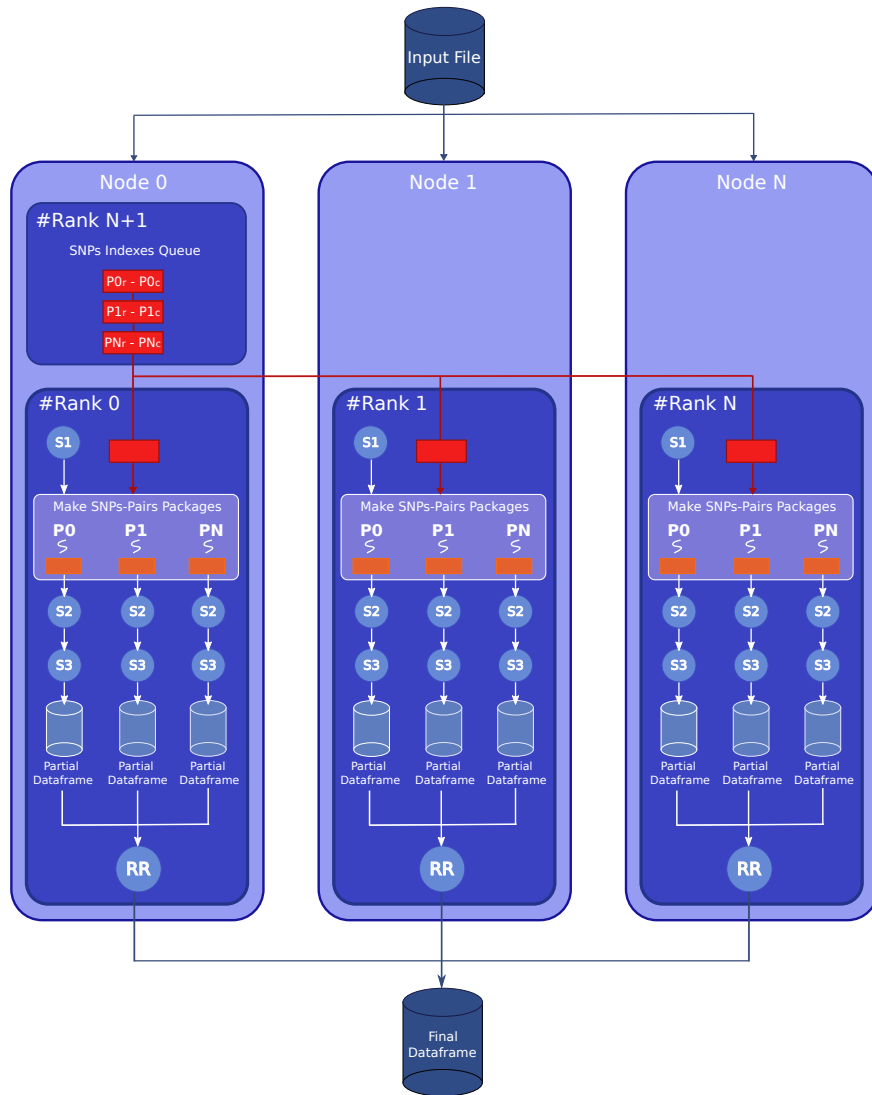Figure 1: Pipeline of the FaST-LMM epistasis test module.

Figure 2: Pipeline of the MPI extended FaST-LMM epistasis test module (dynamically distributed workload version).
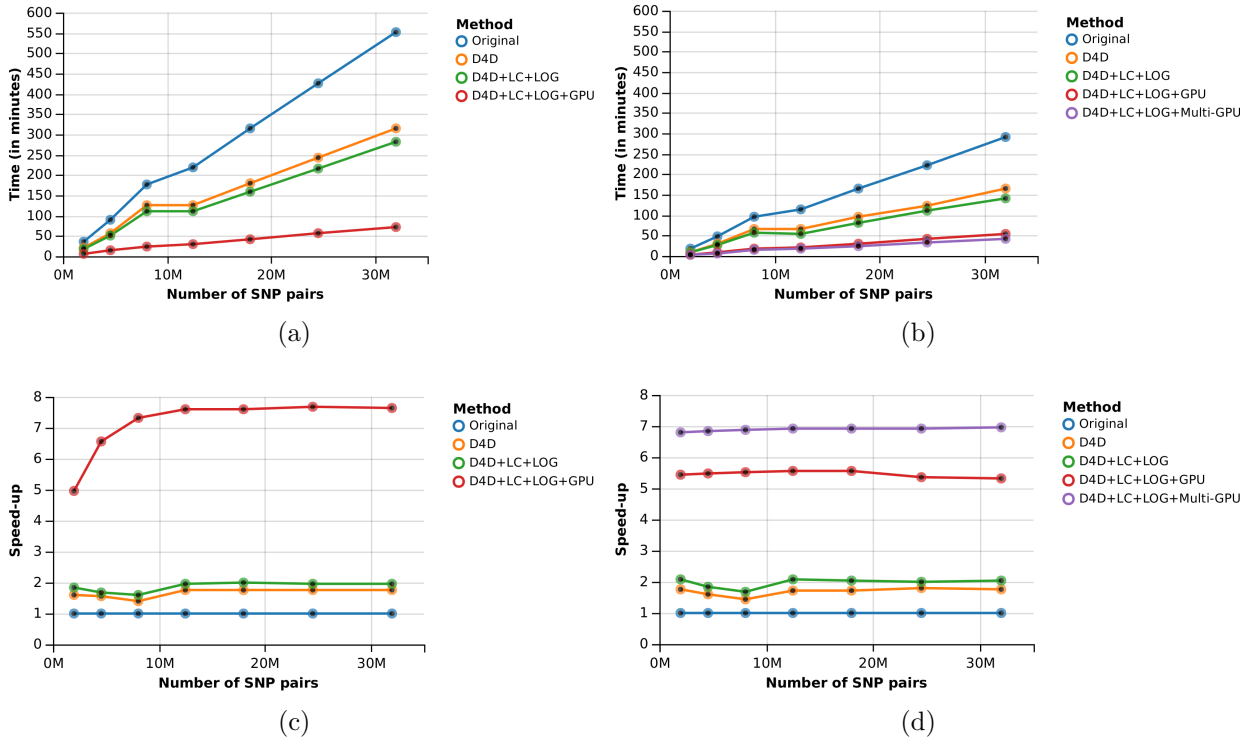
Figure 3: Execution time of the original FaST-LMM epistasis test, and of FaST-LMM with the previously proposed enhancements and the multi-GPU extension, and their speed-ups, for different numbers of SNP pairs on a single node of: Piz Daint, using 12 processes per node —figures a) and c)—; and Minotauro, using 16 processes per node —figures b) and d)—.
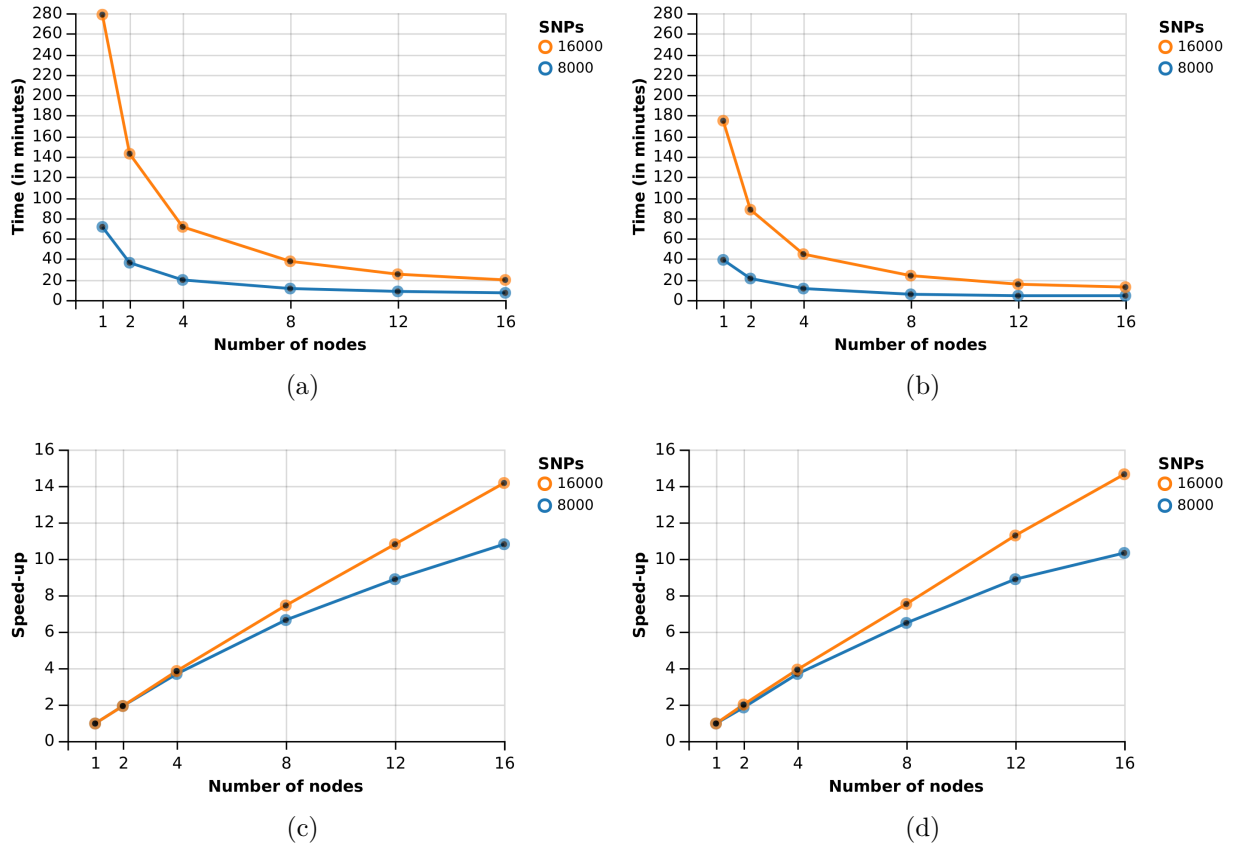
Figure 4: Execution time of the cluster extension of FaST-LMM and its speed-up for different number of nodes on: Piz Daint —figures a) and c)—; and Minotauro —figures b) and d)—.

Table 1: Some applications for genome-wide analysis of two-way epistasis.

Software	Method	Parallel model	URL
BOOST	Regression	—	http://bioinformatics.ust.hk/BOOST.html#BOOST
Encore	Gen. linear model	OpenMP	https://github.com/insilico/encore
EpiGPU	Regression	OpenCL	https://github.com/explodecomputer/epiGPU
EpistSearch	Regression (BOOST)	CUDA/FPGA	—
FastEpistasis	Brute force	SMP/MPI	https://www.cog-genomics.org/plink/1.9/epistasis
FaST-LMM	Linear-mixed model	Hadoop	https://github.com/MicrosoftGenomics/FaST-LMM
GBOOST	Regression (BOOST)	CUDA	http://bioinformatics.ust.hk/BOOST.html#GBOOST 2.0
SNPRuler	Machine learning	—	http://bioinformatics.ust.hk/SNPRuler.zip
SUPER	Linear-mixed model	—	—
TEAM	Data mining	—	http://www.csbio.unc.edu/epistasis/download.php

Table 2: Minimum and maximum time per node (in seconds), and their difference, when using an static or a dynamic workload distribution for the 16,000 SNPs dataset on the Piz Daint platform.

Nodes	Static distribution			Dynamic distribution		
	Minimum	Maximum	Diff	Minimum	Maximum	Diff
2	8,482.62	8,555.93	73.31	8,525.11	8,528.32	3.21
4	4,271.68	4,359.87	88.19	4,299.37	4,300.76	1.39
8	2,140.83	2,240.32	99.49	2,219.90	2,223.12	3.22
12	1,465.22	1,558.30	93.08	1,521.50	1,524.34	2.84
16	1,109.62	1,192.63	83.01	1,171.14	1,174.06	2.92

Table 3: Execution time (in seconds) for different number of nodes and a fixed problem size (3,000 SNPs) per node on Piz Daint and Minotauro.

Nodes	SNPs Pairs	Time	
		Piz Daint	Minotauro
1	31,996,000	4,244.78	2,351.77
2	63,997,641	4,260.44	2,402.85
4	127,992,000	4,300.12	2,694.75
8	255,979,251	4,286.53	2,684.12
12	383,991,328	4,300.18	2,695.05
16	511,984,000	4,294.01	2,715.84