Graphics Processing Unit-Accelerated Quantitative Trait Loci Detection

Guillaume Chapuis, Olivier Filangi, Jean Michel J. M. Elsen, Dominique Lavenier, Pascale Le Roy

JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 20, Number 9, 2013
© Mary Ann Liebert, Inc.
Pp. 672–686
DOI: 10.1089/cmb.2012.0136

Graphics Processing Unit–Accelerated Quantitative Trait Loci Detection

GUILLAUME CHAPUIS,[1,*] OLIVIER FILANGI,[2,*] JEAN-MICHEL ELSEN,[3]
DOMINIQUE LAVENIER,[4] and PASCALE LE ROY[2]

ABSTRACT

Mapping quantitative trait loci (QTL) using genetic marker information is a time-consuming analysis that has interested the mapping community in recent decades. The increasing amount of genetic marker data allows one to consider ever more precise QTL analyses while increasing the demand for computation. Part of the difficulty of detecting QTLs resides in finding appropriate critical values or threshold values, above which a QTL effect is considered significant. Different approaches exist to determine these thresholds, using either empirical methods or algebraic approximations. In this article, we present a new implementation of existing software, QTLMap, which takes advantage of the data parallel nature of the problem by offsetting heavy computations to a graphics processing unit (GPU). Developments on the GPU were implemented using Cuda technology. This new implementation performs up to 75 times faster than the previous multicore implementation, while maintaining the same results and level of precision (Double Precision) and computing both QTL values and thresholds. This speedup allows one to perform more complex analyses, such as linkage disequilibrium linkage analyses (LDLA) and multiQTL analyses, in a reasonable time frame.

Key words: Cuda, double-precision, GPGPU, GPU, linkage analysis, linkage disequilibrium, multiQTL analysis, QTLMap, QTL mapping.

1. INTRODUCTION

Most of the traits characterizing individuals (their ‘‘phenotypes’’: performance level, susceptibility to disease, etc.) are influenced by heredity. Geneticists are interested in detecting, localizing, and identifying genes, the polymorphism of which explains a part of observed trait variability. Such genes are often called QTLs (for quantitative trait locus), the term locus pointing to a physical position on the genome.

QTL detection procedures consist of a series of statistical hypotheses tests at successive putative locations on the genome. Many experimental designs, sampling protocols, and test statistics were proposed and used. We focus here on regression approaches performed on sets of large families. These approaches were

[1]GenScale Team, INRIA Rennes, Rennes, France.
[2]Pegase Team, INRA, Rennes, France.
[3]SAGA Team, INRA, Toulouse, France.
[4]CNRS Team, IRISA, Rennes, France.
*These authors contributed equally to this work.

developed for exploiting the linkage disequilibrium—the discrepancy between a random distribution of haplotypes and the observed distribution of haplotypes in the studied population—observed on a per family basis and/or at the population level. Amongst the available software dealing with QTL regression techniques, QTLMap was developed by some of the authors of the current article (Elsen et al., 1999).

The general principle of linkage analysis (LA) for detecting QTLs within a family is to correlate for each tested genome position the performance trait measured in the progenies and the grand parental origins of the piece of chromosome they received from a common parent. These origins are inferred from ''genomic marker information,'' which describes the parental chromosomes (in diploid species, chromosomes are in pairs and each individual carries two copies, or ''alleles'' of QTLs, say Q1 and Q2, and markers, say M1/M2, N1/N2) and the way they are transmitted to their progenies (Fig. 1). Locations of QTLs are pointed on chromosomal segments, which display high correlations.

Linkage disequilibrium analysis (LDA) does not exploit family structure but considers the whole population as a large sample of independent individuals. Because of various demographic events along the population histories (selection, breeds mixtures, bottlenecks, etc.), allelic forms found at two close chromosomal positions are generally not independent. A few measurements of this dependence were proposed in the past (Lewontin, 1964; Hill and Robertson, 1968). This phenomenon is fully proven for markers, which are easy to visualize (e.g., Farnir et al., 2000, for the bovine species) and certainly true between

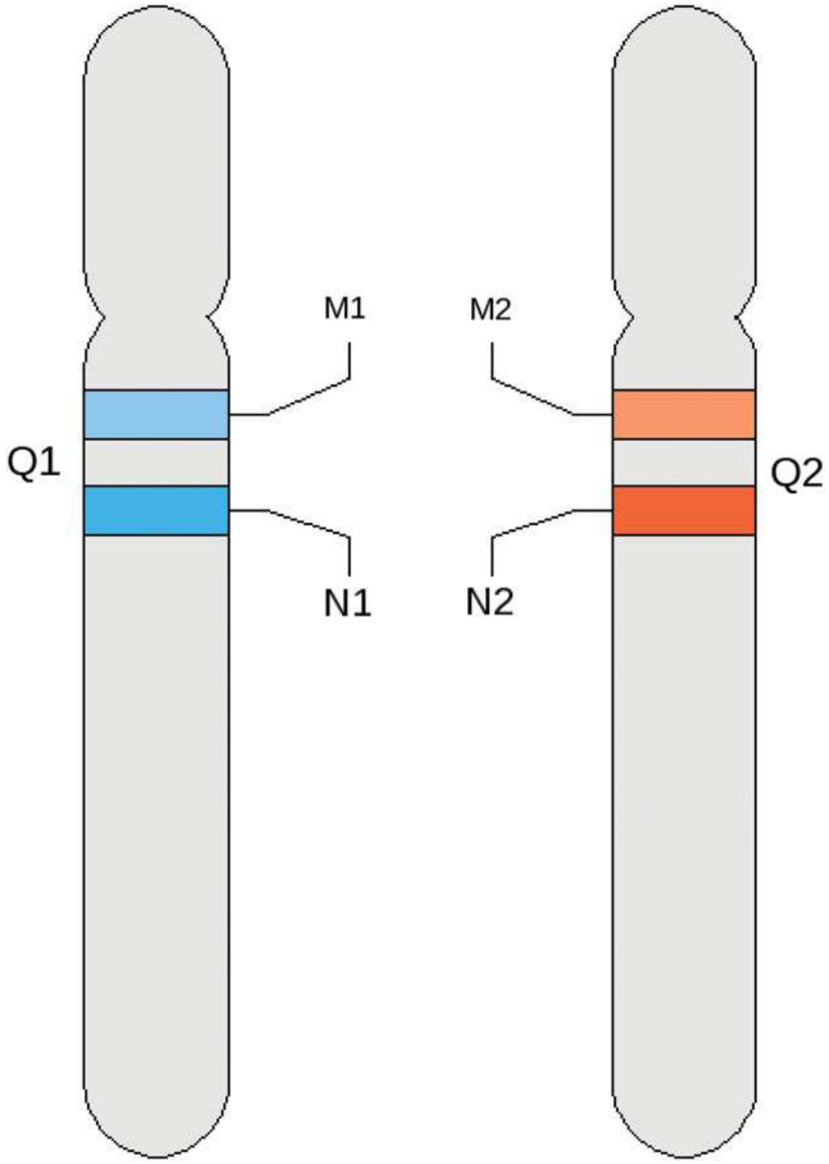

FIG. 1. Repartition of markers M1/M2 and N1/N2 on alleles Q1 and Q2.

markers and QTLs. In LDA, the direct effect of genetic information (to make presentation simple, say the marker genotype effect) on the quantitative trait variability is tested at successive positions. The basic idea is that groups of individuals defined by their genetic class (e.g., their genotype for the marker located at the tested position) will display significant differences in their quantitative performance if a QTL is located close to this position. The signal will be stronger if:

1. the linkage disequilibrium between markers and QTL is higher and
2. the QTL explains a larger part of the variability.

A third category of techniques combines LA and LDA. In ''linkage disequilibrium linkage analysis'' (LDLA), both the family structure and the population history are exploited. The population is described as a set of ''founders,'' supposed unrelated, but subject to linkage disequilibrium, and ''nonfounders,'' which inherited from the founders intact or recombinant chromosomes after one or more generations of transmission. In LDLA, the performance trait measured in the nonfounders are correlated, as in the LA, with the founder origins of the transmitted piece of chromosomes, and, as in the LDA, information about the population history is extracted from the degree of similarity between founder pieces of chromosomes. Thus, detecting QTLs is basically a three-step procedure:

1. Inferring from the marker information the ''phase'' of the parents (Elsen et al., 1999; Favier et al., 2010), that is, the way the two alleles of each marker are positioned on the chromosomes. The output is the haplotype pairs of each parent, that is, the list of successive marker alleles carried by each chromosome (e.g., M1N1 and M2N2, or M1N2 and M2N1);
2. estimating parents to progenies transmission probabilities of chromosomal segments; and
3. evaluating the likelihood of performance traits observation under alternative hypotheses (a QTL is present or not at the chromosomal segment location) and modeling (LA, LDA, LDLA).

A statistical test is operated at each genome location tested, and the best location for the QTL is given by the most significant test. In terms of both theoretical developments and/or computation burden, a major difficulty is to obtain correct statistical test rejection thresholds.

Different strategies exist to estimate these thresholds. Efforts were made to find the distribution of the statistical process under the null hypothesis (no QTL on the chromosome), but they did not fully consider the real-life situations in which unbalanced designs are the rules (e.g., Rabier et al., 2010). Alternatively, and this is the usual procedure, thresholds can be estimated empirically after many permutations of the data breaking the marker-phenotype correlations, or after many simulations under H0 (Churchill and Doerge, 1994).

QTL mapping analyses, such as LDLA, are computationally intensive. For QTLMap, run times increase linearly with the size of the studied population, the exploration step of the studied genome region, and the number of simulations required to determine the thresholds. As opposed to the sizes of studied populations, which should remain rather stable, the density of marker data increases exponentially and allows a finer exploration of the genome regions, with a higher number of tested positions. Current QTL mapping analyses may take weeks to run in the most difficult cases (e.g., when looking for QTL interactions) on modern computers, and run times will increase linearly with the density of available genetic markers. Therefore, dividing run times by an order of magnitude would allow geneticists to run multiple analyses or consider even more time-consuming analyses, such as multiQTL ones.

Despite the computational burden that QTL mapping represents, very few parallel tools exist. The first attempt was made by Seaton et al. (2006) with gridQTL. This tool is derived from QTLexpress (Seaton et al., 2002), a popular web-based tool for QTL analyses, and harnesses the power of computational grids to try and reduce run times. Another approach was developed by Filangi et al. (2010) with a tool called QTLMap. This tool takes advantage of modern CPUs by using all their cores simultaneously. Finally, epiGPU (Hemani et al., 2011) uses any commercially available GPUs for QTL analyses but focuses on the detection of epistasis—a complex interaction between genes in which the effect on a given phenotype of a single gene is altered by one or more other genes. Other QTL software, such as eQTL, specialized in detecting expression QTLs, still run on a single CPU core.

The empirical approach used in QTLMap makes it an ideal candidate for GPU computations. In order to determine efficient relevance thresholds, the analysis not only needs to be performed at each genome position but must also be repeated for each simulated dataset—typically 10^3 to 10^4 times. The increasing density of genetic marker data allows for ever more precise analyses, meaning that more and more genome locations need to be considered. Computations at neighboring genome locations are correlated; it is however more efficient in practice to consider them independant. Computations for each simulation are independant. These computations can therefore be run in parallel.

In this article, we propose a new version of QTLMap, which performs about 70 times faster than the previous multicore implementation, while maintaining the same level of precision. We first describe the empirical methods for QTL Mapping ported to GPU in QTLMap and give details about the algorithms of the methods. We then describe the implementation of these algorithms on the GPU. We finally show the details of the experiments we ran to test our new implementation, and the results we obtained.

2. METHODS AND ALGORITHMS

QTLMap relies on three methods to determine possible QTL locations on linkage groups. The first method, called linkage analysis (LA), aims at determining the transmission probability of each chromosomal segment based on available marker information in the studied population. The second method, referred to as linkage disequilibrium analysis (LDA), relies on studying the discrepancy between an expected random distribution of haplotypes in the studied population and the observed distribution. The third method, referred to as linkage disequilibrium linkage analysis (LDLA) combines the first two approaches. Contrary to a linkage analysis, a linkage disequilibrium linkage analysis does not solely take into account the length of chromosomal segments to determine transmission probabilities, it also takes into account the more complex linkage disequilibrium effect.

Once transmission probabilities have been determined, QTLMap statistically computes the likelihood of the observations under the hypothesis that a QTL is present at successive genome locations in the studied linkage groups. QTLMap then uses an empirical approach to determine thresholds, above which a QTL effect can be considered significant. The following two sections give a brief overview of the QTL mapping methods implemented in QTLMap. A more detailed description of these methods can be found in Elsen et al. (1999).

2.1. Linkage analysis

In QTLMap, the hypothesis is tested that one QTL affecting a single trait is located at a position x in a linkage group (e.g., a chromosome). Successive positions on this linkage group are scanned. The test is performed with the interval mapping technique applied to an approximation of the likelihood of having a QTL at a given location (Knott et al., 1996; Elsen et al., 1999; Le Roy et al., 1998).

Let ns and nd be the number of sires and dams, respectively, in the studied population. All parents are supposed heterozygous at the QTL, with specific alleles, giving a total of $2(ns + nd)$ QTL genotypes. Performance expectation of progeny k of parent i and j is described as the sum of parental mean values $\mu_i + \mu_{ij}$ and of the deviations $\pm \alpha l$ to this mean because of the QTL. In this model, it is assumed that the parents are unrelated, the markers in linkage equilibrium—i.e., a random distribution of haplotypes is assumed—and the trait normally distributed.

As proposed by Goffinet and Didier (1999) and by Le Roy et al. (1998), the residual variance of the quantitative trait is estimated within sire. Considering that subpopulations—in our case, descendants of a given sire—can have different variances is called a *heteroskedastic* hypothesis. This heteroskedastic parametrization better fits different patterns (between sires) of segregation of other QTLs unlinked to the tested position. The homoskedastic hypothesis, which considers that the variance is equal for any two subpopulations, is also implemented.

Parameters maximizing the likelihood can be obtained in an iterative two-step procedure:

1. Solving a linear system (see Elsen et al., 1999, for details); and
2. estimating within sire family variances.

The steps are repeated until convergence, detected when the distance between the likelihood ratio test (LRT) at iteration t and the likelihood ratio test at iteration $t+1$ is arbitrarily small enough.

2.2. Linkage disequilibrium and LDL analyses

QTLMap implements the ‘‘LDA decay’’ regression approach described by Legarra and Fernando (2009). This linkage disequilibrium analysis is particularly adapted to experimental populations characterized by a family structure, the target of this software. In this approach, parental haplotypes are pooled in classes, the classification being open to the user decision. In QTLMap, only the most probable sire and dam phases are considered, and the classes (following the example given by Legarra and Fernando, 2009) are simply defined by the haplotype (to a class corresponds a single haplotype). To a given class corresponds a specific effect on the quantitative trait. The quantitative performance of a progeny depends on the haplotypes as found in the parental chromosomes from which the putative QTL alleles are originating and not to the (possibly recombinated) haplotypes the progeny itself is carrying.

The LDLA approach described by Legarra and Fernando (2009) was also implemented. This approach combines the previous LD ecay and LA models, the QTL effect being defined within the parental haplotype effect.

2.3. Thresholds detection

The QTL mapping procedures described in paragraphs 2.1 and 2.2 determine, for each position on the studied chromosomal region(s), the likelihood of having a QTL related to a given trait. This score can only be considered relevant if it is above a certain threshold that has yet to be determined.

We define $H(n)$ as the hypothesis of having an n-QTL at n given positions. QTLMap uses an empirical approach to determine relevance thresholds. In order to estimate the probability of having a QTL at a given location ($[H(1)]$), it is compared to the null hypothesis, $H(0)$. Distribution under $H(0)$ is calculated by running the previous algorithm on random sets of data. Randomly generated datasets share the same architecture as the actual dataset. They contain the same population, but for each invidual, performance vectors are randomly generated.

In order to compute the distribution under $H(0)$, a user set number of simulations are randomly generated and run. Efficient empirical thresholds can be obtained by computing a large number of simulations. A single analysis with QTLMap explores *npos* genome positions; rejection thresholds are obtained by running *nsim* analysis on simulated data, leading to a total of *nsim.npos* likelihood computations. Computations at each genome position are correlated, but it is better in practice to consider them independent. Computations for each simulation are independent. These computations can therefore be run in parallel.

2.4. Algorithms for QTL detection

QTLMap provides three types of analyses, presented in sections 2.1 and 2.2, and allows for two types of parametrizations: hetero- and homoskedastic parametrizations. In a heteroskedastic analysis, the variance of subpopulations can differ, whereas in a homoskedastic analysis, the variance is considered stable within the studied population. This section gives information about the structure of the algorithm depending on the analysis and on the parametrization. This section also describes the nature of the data used for computations.

Algorithm 1 describes the algorithm implemented in QTLMap for a heteroskedastic analysis. The listing does not describe in details how to solve the linear system—line 7 of Algorithm 1—nor how to estimate the variance—line 8 of Algorithm 1—(see Elsen et al., 1999, for details). Instead, attention is brought to the structure of the computations and more precisely to the three loops—lines 2, 3, and 5 of Algorithm 1—that are offset to the GPU. The type of analysis—LA, LDA, LDLA—does not change the structure of the algorithm and will only affect the way the linear system is solved.

Algorithm 1: Algorithm for heteroskedastic analysis

```
1  Begin
2      for each genome_position
3          for each simulation
4              $LRT = 0$
5              do
6                  $LRT_{old} = LRT$
7                  $solve\_linear\_system()$
8                  $LRT = estimate\_variance()$
9              while $|LRT - LRT_{old}| > \varepsilon$
10 End
```

Iterations of the first two loops—lines 2 and 3 of Algorithm 1—are completely independent and can be run in parallel. However, iteration n of the third loop—line 5 of Algorithm 1—depends on the result of iteration $n - 1$, therefore, the third loop cannot be parallelized.

In the more specific case of a homoskedastic analysis, results can be obtained in one pass without waiting for convergence. Details of the algorithm are given in Algorithm 2. As in Algorithm 1, iterations of the two loops—lines 2 and 3 of Algorithm 2—are independent and can be run in parallel.

Algorithm 2: Algorithm for homoskedastic analysis

```
1 Begin
2     for each genome_position
3         for each simulation
4             solve_linear_system()
5 End
```

At each iteration of the two independent loops, a contingency matrix, described in Figure 2, is used for computations. For each individual in the studied population—referred to as descendants—these matrices contain values in various effects, some of which are independent of the current genome position (*i.e.*, fixed) and others are dependent of the position (*i.e.*, variable) and a performance value, which describes their performance with respect to the studied trait. Performance values are also independent of the current genome position but change for each simulation. All matrices have strictly the same dimensions for every iteration. Typical sizes for these matrices are about 10^2 for the number of descendants and 10^2 for the total number of effects (including performance).

The properties exhibited by Algorithms 1 and 2 make them ideal candidates for computations on a GPU. First, both algorithms mainly consist of two independent loops, meaning that all iterations can be processed in any order or, in our case, simultaneously. Second, dimensions of input data (Fig. 2) are identical for every iteration. This consistency of the dimensions of input data allows for regular data access patterns as well as a stable number of instructions to process each iteration. Finally, input data for every iteration is partly redundant, thus leaving room for optimization.

3. GPU IMPLEMENTATION

Computations offset to the GPU consist mainly in operations on matrices. Operations such as Cholesky decompositions and matrix multiplications are ideally suited for execution on a GPU and can be achieved at near peak performance on such devices (Volkov and Demmel, 2008). Several highly optimized libraries exist, providing linear algebra routines benefitting from modern hybrid architecture (Humphrey et al., 2010;

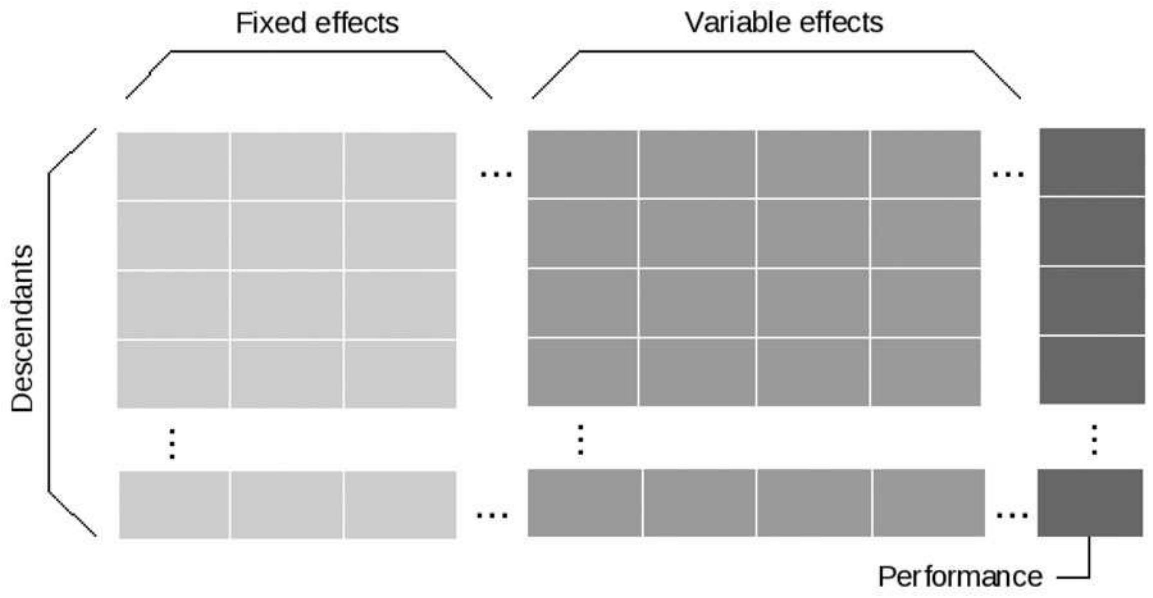

FIG. 2. Description of a contingency matrix used for computations at each genome position and for each simulation.

Tomov et al., 2009). However, these libraries specifically target operations on large matrices. In our case, computations are done on a large number of rather small matrices, typically $10^2 * 10^2$, therefore, no performance would be gained from using these libraries for such small matrix sizes. Single instruction multiple data (SIMD) parallelism can nevertheless be drawn from the large number of matrix operations performed on different small matrices.

This section describes how the algorithms presented in Section 2.4 are mapped onto the GPU's architecture and what otpimizations were applied to accelerate computations.

3.1. Mapping computations on the GPU

Parallelism is present at two levels on a GPU:

- SIMD cores running simultaneously on the GPU; and
- hardware threads running simultaneously within a single core.

Mapping computations on a GPU, a process also called gridification, consists therefore in separating the given problem on these two levels of parallelism. The problem must first be broken down into blocks; each block is executed on a single SIMD core. Each block must then be divided into threads, which will be mapped to the hardware threads of the SIMD core.

Several gridifications are implemented in QTLMap depending on the nature of the computations and data access paterns. Figure 3 shows an example of such a gridification. In this example, each block handles computations on $32*16$ matrices (described in 2) corresponding to different genome positions and simulations. A single thread handles the computations for a single genome position and a single simulation.

3.2. Optimizing GPU memory usage

As mentioned in Section 2.3, the algorithm for QTL detection needs to be run on the actual input dataset and a large number of randomly generated datasets in order to test the results against the null hypothesis. In our case, these computations are independent and an obvious data parallelism pattern can be exploited. The amount of available memory on a GPU is nevertheless limited when compared to a CPU's memory. Therefore, great care must be exercised when offsetting data to a GPU.

The amount of memory required for a single analysis can be divided into three categories:

- memory for input data;
- memory for intermediate results; and
- memory for end results.

Input data consist of contingency matrices at each position and simulation (Fig. 2). The amount of memory M required for contingency matrices for a linkage analysis is given by the following formula:

$$M = nsim * npos * (1 + (1 + nqtl) * ns) * sizeof(DOUBLE)$$

Where *nsim* is a user set number of simulations to run, *npos* is the number of positions to test on the linkage group, *nqtl* is the number of QTL to look for, *ns* is the number of sires, and *sizeof*(*DOUBLE*) is the size in bytes of a double precision float on the given architecture. The previous formula is valid if one decides to store integrally every incidence matrix. Memory optimizations can however be performed. Each matrix contains a first set of population averages, which are independent of both the position in the genome and the family, and a second set of polygenic effects, which are independent of the genome position. All these effects can be factored out and stored only once. The resulting amount of memory required M_{input} is now:

$$M_{input} = nsim * npos * nqtl * ns * sizeof(DOUBLE)$$

The amount of memory required for intermediate results depends on the type of analysis and represents a significant part of the total memory requirements. For large analyses, the input data can be sent to the GPU but computations are limited by the low amount of memory available for intermediate results. In these cases, computations are grouped into workloads, which are then handled sequentially on the GPU. Each workload consists of a number of genome positions that can be computed together within the GPU's global memory. The optimal size *Max_pos* for a workload is calculated using the following formula:

$$Max\_pos = \lfloor (Free\_mem - M_{input}) / IRsize \rfloor$$

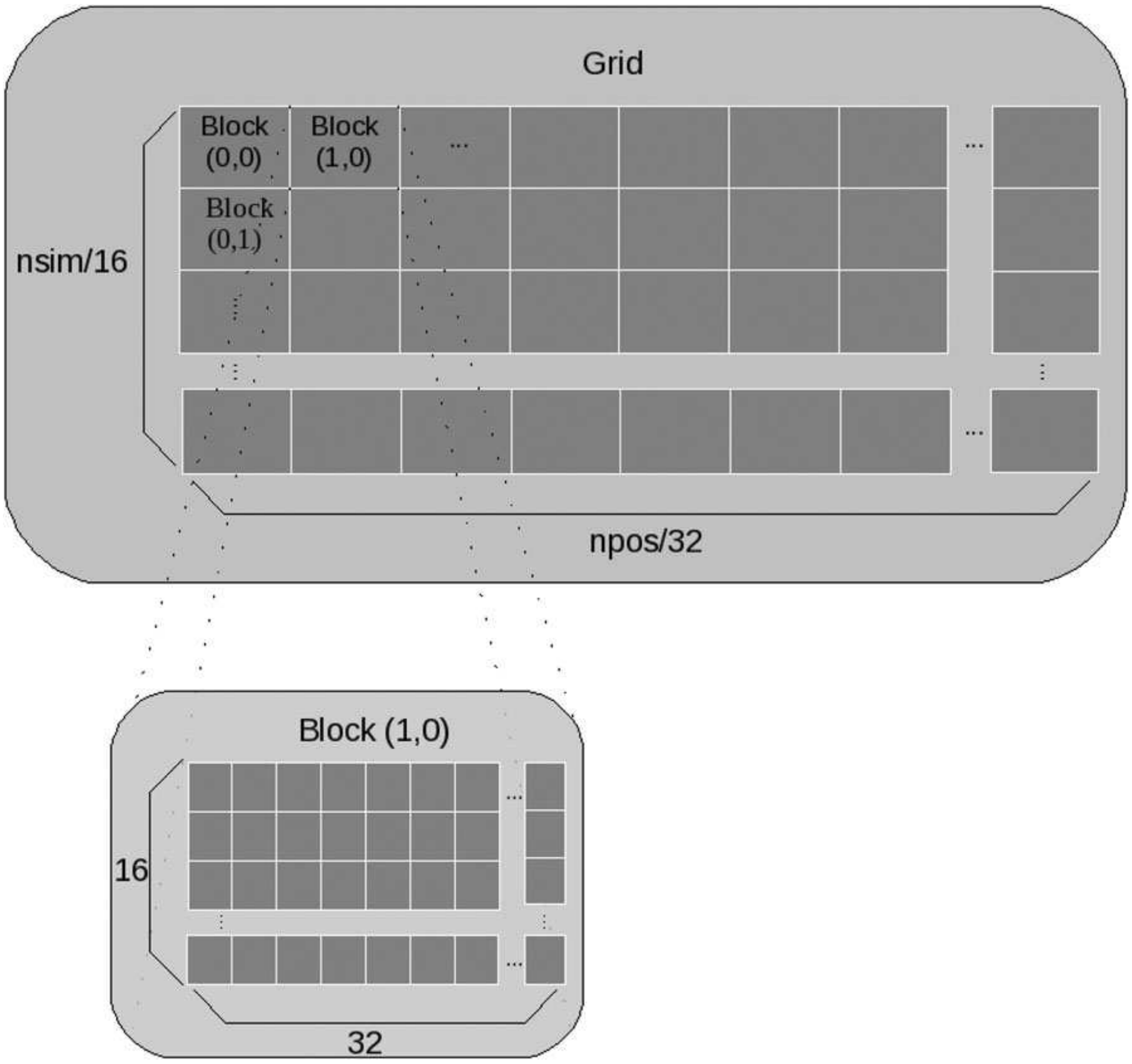

FIG. 3. Example of gridification on the GPU.

where *Free_mem* is the amount of memory available on the GPU, M_{input} is the total amount of memory required for input data, and *IRsize* is the amount of memory required for intermediate results for a single genome position.

Recent cuda versions allow data transfers between the CPU and the GPU to overlap with computations on the GPU. A possible optimization would be to reduce the size of a workload to half the available memory on the GPU and transfer a workload while the previous one is being computed. Another optimization would be to partition the input data into workloads as is done with intermediate results.

3.3. Reducing CPU/GPU transfers

Data transfers between the host (CPU) and the accelerator (GPU) are rather time consuming and need to be optimized. Part of solving the linear system, as mentioned in Algorithm 1, consists of determining confounding effects, that is, effects correlated with other effects. These effects are identified by a Cholesky decomposition and need to be removed from the dataset for further computations. Subsequent computations are performed on a subset of each matrix—the structure of these matrices is described in Section 2—excluding confounding effects. To avoid recopying the matrices, confounding effects are excluded using conditional statements. Nevertheless, branching statements (such as ''if'' and while) can significantly reduce performances on a GPU. This is because consecutive Cuda threads are grouped together in warps of

16 threads. Whenever a branching statement occurs, if threads within a single warp take different paths, both paths are executed sequentially, thus breaking parallelism at this level. However, due to the biological nature of the problem, diverging branches never occur since confounding effects are identical for each matrix. Therefore, the overhead induced by adding these branching statements is negligible compared to the overhead induced by transferring the matrices back and forth between the CPU and the GPU.

3.4. Optimizing homoskedastic analyses

Each step of the analysis, either using real data or a simulated set, shares a small amount of computations with other steps. This is because only performance vectors are randomly generated for simulations. In order to avoid redundancy, matrix multiplications involved in solving the linear system, described at line 4 in Algorithm 2 are split into three phases:

- multiplications solely involving fixed effects, shared by all matrices (*i.e.*, without performance effects);
- multiplications involving performance effects that differ from one dataset to another, as well as fixed effects; and
- multiplications solely involving performance effects.

The first phase is computed only once on the CPU, while the second and third phases are computed for each dataset in parallel on the GPU. Computations that are common to each matrix multiplication are thus factored out. This represents a very slight improvement over the previous CPU implementation and was only relevant in the GPU implementation, where these computations are done simultaneously. Dividing these computations also allows us to only keep one copy of the part common to all matrices while the rest is stored on the GPU in a compact form; only the relevant halves of the triangular matrices are kept contiguously in memory.

Phases two and three are computed in two distinct Cuda kernels on the GPU in order to optimize memory accesses. Requirements imposed by the Cuda model on memory access patterns to the GPU's global memory are very strict and have a tremendous impact on performances. Memory accesses in these two kernels are optimized for coalescing either by reorganizing data on the GPU or by preloading subsets of the data into shared memory. When breaking coalescing is unavoidable, keeping data locality allows us to benefit from the small cache available on recent graphics cards.

4. EXPERIMENTS AND RESULTS

Tests were run on machines with two quadcore Intel® Xeon® E5420 (12M Cache, 2.50 GHz, 1333 MHz FSB) processors. Multicore CPU tests were run on the Genotoul platform. GPU tests were run on a machine equipped with an Nvidia® C2050 card. Each test consists of an LDLA analysis over simulated datasets from the 2011 QTL-MAS workshop. Two versions of QTLMap are compared here:

- the previous multicore CPU version running with eight threads (Filangi et al., 2010); and
- the new GPU version in double precision.

For the CPU executions, each of the eight threads had a dedicated CPU core. Input parameters ranged from 500 to 10000 for the number of simulations, from 9 to 998 for the number of genome positions, and from 5 to 20 for the number of sires.

4.1. Execution times

Figures 4, 5, and 6 show the evolution of execution times for both the CPU and the GPU versions over the number of simulations, the number of half-sib families, and the number of genome positions respectively. Times for the CPU version are given on the left Y-axis, while times for the GPU version are given on the right Y-axis.

The amount of computations required for the analysis grows linearly with the number of simulations (Fig. 4) and the number of considered genome positions (Fig. 6). These linear growths were expected, given the structure of the algorithm—lines 2 and 3 of Algorithm 1. On the other hand, run times grow polynomially with the number of sires—the polynome depends on the type of analysis performed (Fig. 5).

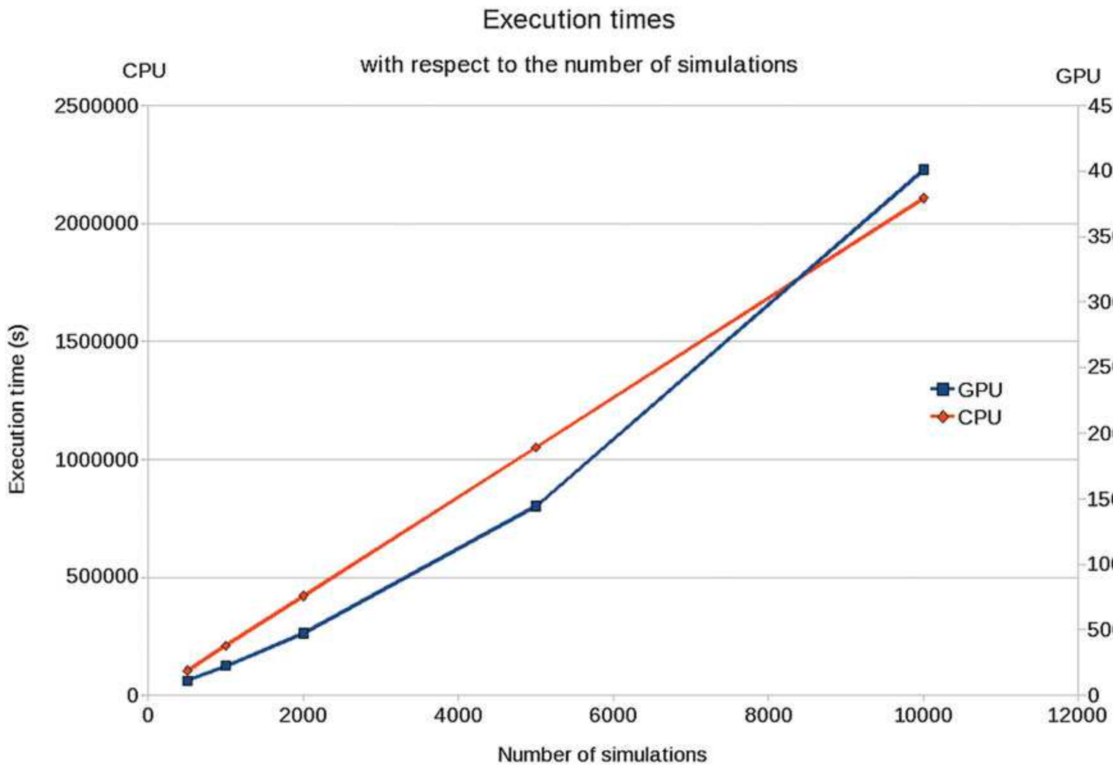

FIG. 4. Evolution of the execution time with respect to the number of simulations.

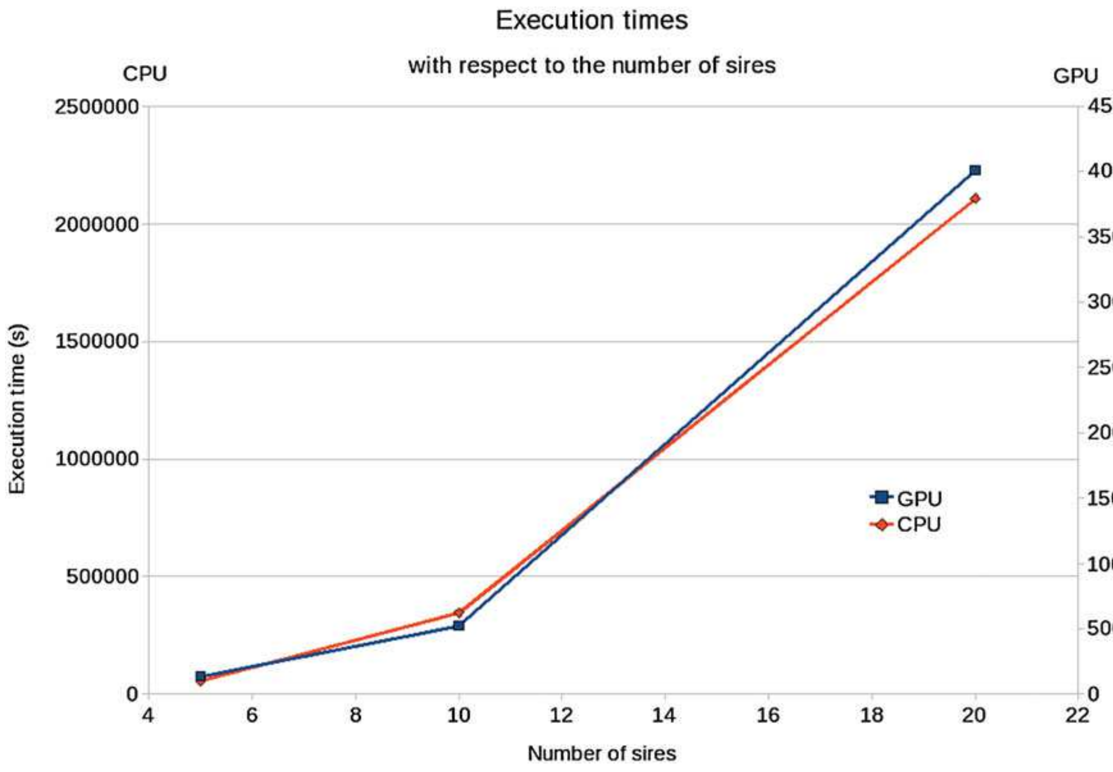

FIG. 5. Evolution of the execution time with respect to the number of half-sib families.

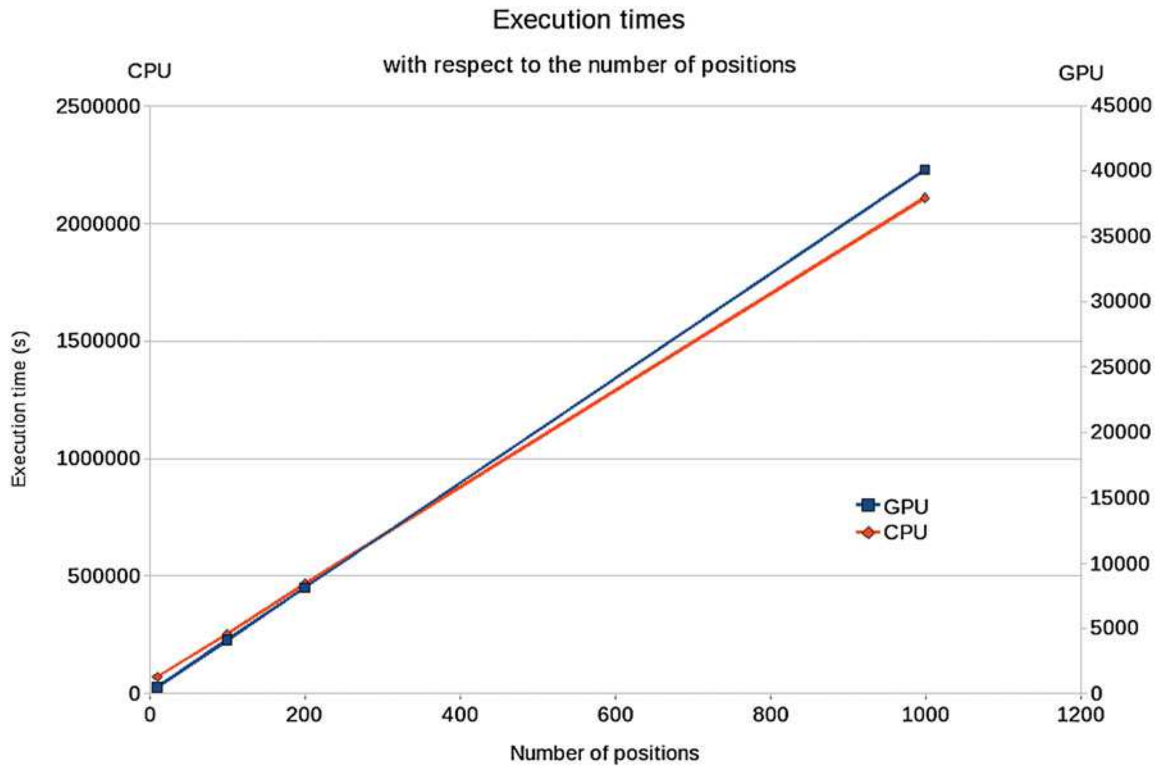

FIG. 6. Evolution of the execution time with respect to the number of genome positions.

Table 1 shows the values and ranges of values for fixed and variable parameters used in Figs. 4, 5, and 6. The most time consuming analysis, using 10,000 simulations, 20 sires, and covering 998 genome positions, took more than 3 weeks to compute on the CPU, and slightly more than 11 hours on the GPU.

4.2. Speedups

Figure 7 shows the evolution of the speedups between the two versions of QTLMap according to the number of simulations (regarding the number of sires and the number of genome positions as shown in Figs. 8 and 9, respectively). Figures 7, 8, and 9 show that speedups remain stable with increasing values in all three dimensions—number of genome positions, number of simulations, and size of the population. Overall, the GPU version performs about 70 times faster than the multicore CPU version. This speedup, however, cannot entirely be attributed to the use of a graphics card. Indeed, the CPU version does not benefit from certain optimizations applied specifically to the GPU version, one of which is described in Section 3.4, nor does it take advantage of SSE instructions. Optimizing the CPU version would probably reduce its run times by a factor of three or four.

Table 2 shows values and ranges of values for fixed and variable parameters used in Figures 7, 8, and 9.

The multicore CPU version of QTLMap is not designed to run optimally for low numbers of genome positions. In the multicore CPU version of QTLMap, data structures are allocated and initialized for each simulation and then amortized over computations for each genome position. On the contrary, in the GPU version of QTLMap, a single set of data structures is allocated and initialized for a large set of simulations

Table 1. Values and Ranges of Values for Fixed and Variable Parameters Used in Figures 4, 5, and 6

	No. of simulations	*No. of sires*	*No. of genome positions*
Figure 4	500–10000	20	998
Figure 5	10000	5–20	998
Figure 6	10000	20	9–998

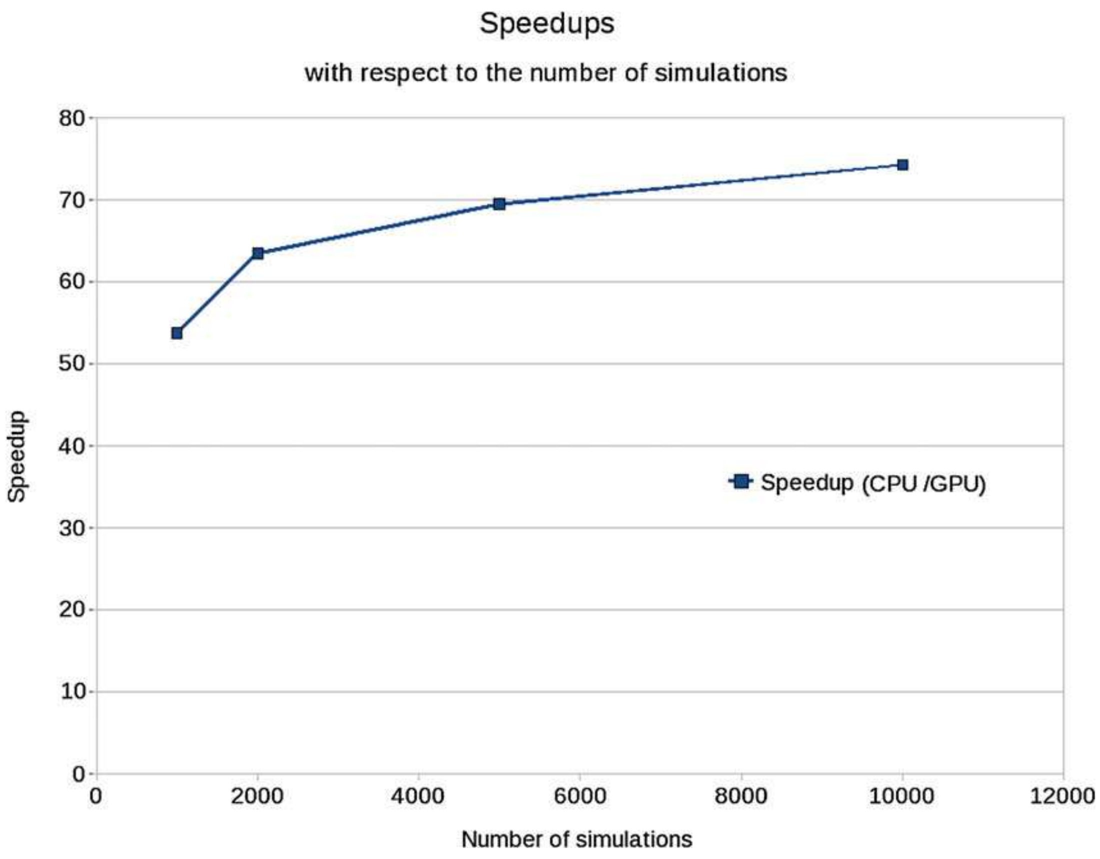

FIG. 7. Speedup with respect to the number of simulations.

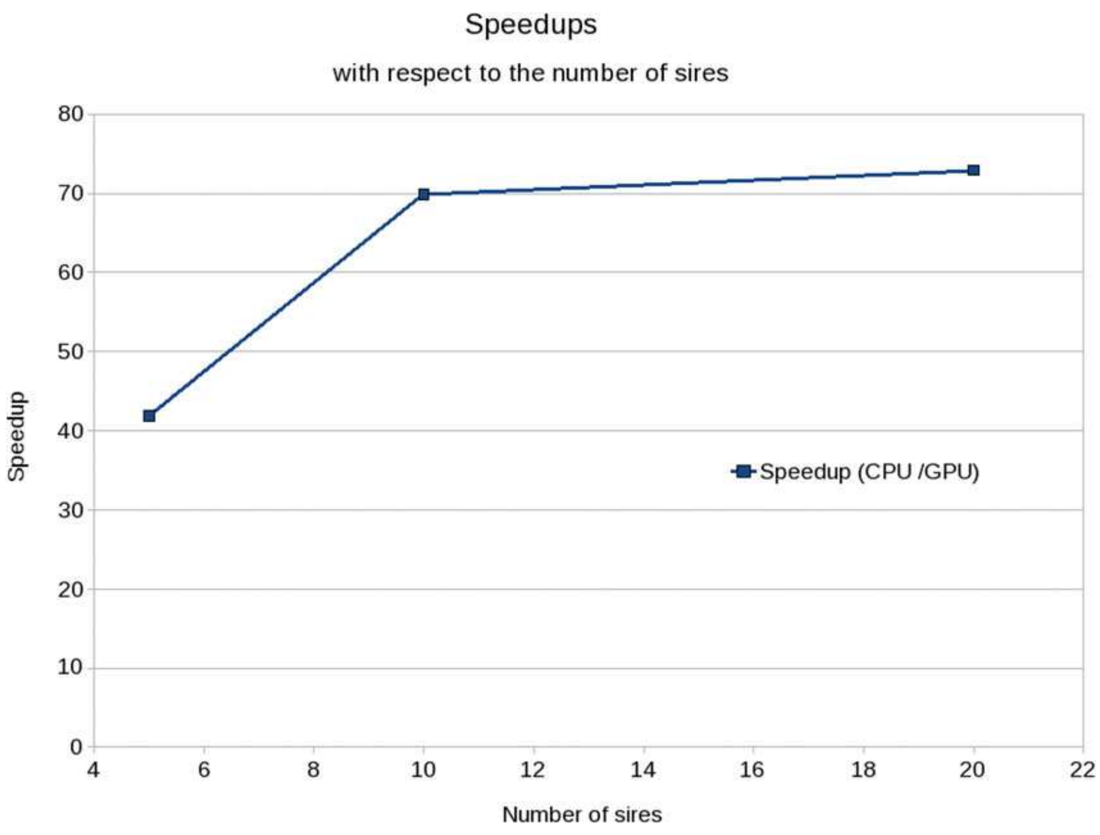

FIG. 8. Speedup with respect to the number of half-sib families.

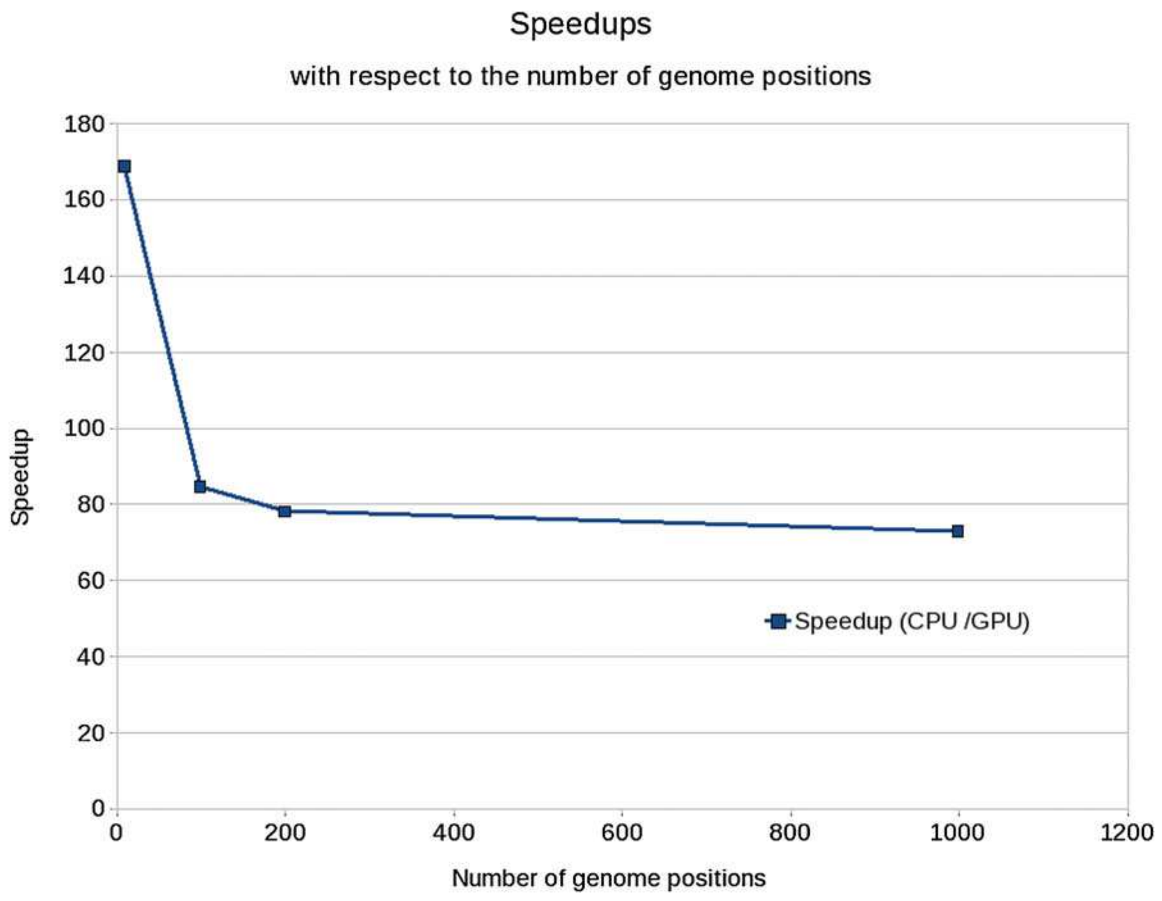

FIG. 9. Speedup with respect to the number of genome positions.

and then amortized over both genome positions and simulations. Consequently, large speedups are observed in Figure 9 between the GPU and the CPU versions for low numbers of genome positions. These speedups are not representative of the true acceleration obtained by porting QTLMap on the GPU; they simply illustrate the fact that the CPU version does not perform optimally for low numbers of genome positions.

5. CONCLUDING REMARKS

In this article, we propose a new version of existing software, QTLMap. QTLMap is a tool for QTL detection, a computationally heavy procedure. This new version takes advantage of GPUs to speed up computations. Computations using this new version are between 50 and 75 times faster than computations using the previous multicore implementation, while maintaining the same results and precision. Reduced runtimes allow geneticists to consider more precise and time consuming analyses by increasing the number

Table 2. Values and Ranges of Values for Fixed and Variable Parameters Used in Figures 7, 8, and 9

	No. of simulations	*No. of sires*	*No. of genome positions*
Figure 7	500–10000	20	998
Figure 8	5000	5–20	998
Figure 9	5000	20	9–998

of simulations or the number of studied genome positions. Reduced runtimes also allow geneticists to consider new analyses, such as multiQTL analyses. All versions of QTLMap are available online under CeCILL licences.

Future work includes the promotion and use of parallel computing in statistical genetics, focusing on two applications of the single nucleotide polymorphism (SNP) chip technology:

- Dissection of the genetic architecture of characters through genome-wise association studies (GWAS) and
- genomic selection (GS).

SNP chip technology now makes possible the genotyping of millions of SNPs for tens or hundreds of thousands of individuals, thus increasing the demand for much faster computations. Faster computations are needed both for implementing more precise genetic models in research of trait genetic determinants and for the industrial exploitation of genomic data, with production of statistical information at regular time intervals.

Three objects largely used in GWAS and GS are targeted:

- G-matrices, a molecular-based measure of between individual genetic relationships;
- c relationships;
- haplotype reconstruction, a problem modeled by hidden Markov chains; and
- identification of causal SNPs with the help of selection variables techniques.

The aim is to produce, when needed, new algorithms better suited for parallel architectures (GPUs and/or clusters of computers).

ACKNOWLEDGMENTS

This work was supported by the AAP INRA/INRIA and by the region of Brittany, France.

AUTHOR DISCLOSURE STATEMENT

No competing financial interests exist.

REFERENCES

Churchill, G.A., and Doerge, R.W. 1994. Empirical threshold values for quantitative trait mapping. *Genetics* 138, 963.

Elsen, J.M., Mangin, B., Goffinet, B., et al. 1999. Alternative models for QTL detection in livestock. I. general introduction. *Genet. Sel. Evol.* 31, 1–12.

Farnir, F., Coppieters, W., Arranz, J.J. 2000. Extensive genome-wide linkage disequilibrium in cattle. *Genome Res.* 10, 220–227.

Favier, A., Elsen, J.M., De Givry, S., and Legarra, A. 2010. Optimal haplotype reconstruction in half-sib families. In *ICLP-10 Workshop on Constraint-Based Methods for Bioinformatics, Edinburgh, United Kingdom.*

Filangi, O., Moreno, C., Gilbert, H., et al. 2010. QTLmap, a software for QTL detection in outbred populations. In *Proceedings of the 9th World Congress on Genetics Applied to Livestock Production* August, 1–6 Leipzig, Germany.

Goffinet, B., and Didier, R. 1999. Alternative models for QTL detection in livestock. III. heteroskedastic model and models corresponding to several distributions of the QTL effect. *Genet. Sel. Evol.* 31, 341–350.

Hemani, G., Theocharidis, A., Wei, W., and Haley, C. 2011. Epigpu: Exhaustive pairwise epistasis scans parallelised on consumer level graphics cards. *Bioinformatics* 27, 1462–1465.

Hill, W. G., and Robertson, A. 1968. Linkage disequilibrium in finite populations. *TAG Theoretical and Applied Genetics* 38, 226–231.

Humphrey, J.R., Price, D.K., Spagnoli, K.E. et al. 2010. Cula: hybrid GPU accelerated linear algebra routines. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* 7705, 1.

Knott, S., Elsen, J., and Haley, C. 1996. Methods for multiple-marker mapping of quantitative trait loci in half-sib populations. *TAG Theoretical and Applied Genetics* 93, 71–80.

Le Roy, P., Elsen, J.M., Boichard, D., et al. 1998. An algorithm for QTL detection in mixture of full and half sib families. In *Proceedings of the 6th World Congress on Genetics Applied to Livestock Production* 26, 257–260.

Legarra, A., and Fernando, R.L. 2009. Linear models for joint association and linkage QTL mapping. *Genet. Sel. Evol.* 41, 43.

Lewontin, R.C. 1964. The interaction of selection and linkage. II. Optimum models. *Genetics* 50, 757–782.

Rabier, C.E., Azais, J.M., Elsen, J.M., and Delmas, C. 2010. Threshold and power for quantitative trait locus detection. Available online at http://hal.archives-ouvertes.fr/hal-00482142/en/

Seaton, G., Hernandez, J., Grunchec, J.A., et al. 2006. Gridqtl: a grid portal for qtl mapping of compute intensive datasets. In *Proceedings of the 8th World Congress on Genetics Applied to Livestock Production*, pages 13–18.

Seaton, G., Haley, C.S., Knott, S.A., et al. 2002. QTL express: mapping quantitative trait loci in simple and complex pedigrees. *Bioinformatics applications note* 18, 339–340.

Tomov, S., Dongarra, J., Volkov, V., and Demmel, J. 2009. Magma library, version 0.1.

Volkov, V., and Demmel, J.W. 2008. Benchmarking GPUs to tune dense linear algebra. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11. IEEE Press, Piscataway, NJ.

Address correspondence to:
Dr. Guillaume Chapuis
GenScale Team
Campus Universitaire de Beaulieu
INRIA Rennes
35042 Rennes
France

E-mail: guillaume.chapuis@irisa.fr