2009 International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing

Accelerating Genome-Wide Association Studies Using CUDA Compatible Graphics Processing Units

Rui Jiang*, Feng Zeng, Wangshu Zhang, Xuebing Wu

MOE Key Laboratory of Bioinformatics
Bioinformatics Division, TNLIST/Dept. of Automation
Tsinghua University, Beijing, China

Zhihong Yu

Intel China Research Center

Beijing, China

Abstract — **Recent advances in highly parallel, multithreaded, manycore Graphics Processing Units (GPUs) have been enabling massive parallel implementations of many applications in bioinformatics. In this paper, we describe a parallel implementation of genome-wide association studies (GWAS) using Compute Unified Device Architecture (CUDA). Using a single NVIDIA GTX 280 graphics card, we achieve speedups of about 15 times over Intel Xeon E5420. We also implement a highly scalable, massive parallel, GWAS system using the Message Passing Interface (MPI) and show that a single GTX 280 can have similar performance as a 16-node cluster. We further apply the GPU program to two real genome-wide case-control data sets. The results show that the GPU program is 17.7 times as fast as the CPU version for an Age-related Macular Degeneration (AMD) data set and 25.7 times as fast as the CPU version for a Parkinson's disease data set.**

Keywords — Genome-wide association studies (GWAS); epistatic interactions; Compute Unified Device Architecture (CUDA); Graphics processing units; Message passing interface

I. INTRODUCTION

Recent developments in medical genetics suggest that genetic variation plays an important role in the pathogenesis of human inherited diseases [1]. It is therefore of great importance to detect causative genetic variants, for the purpose of understanding generic principles underlying these diseases. Human inherited diseases are typically classified into Mendelian diseases and complex diseases. Studies have shown that Mendelian diseases are relatively rare, and an individual genetic variant in a single gene is both sufficient and necessary to cause such a disease. This type of disease is therefore referred to as monogenic disease. In contrast, complex diseases are more common and are often referred to as polygenic diseases, because they are supposed to be caused by multiple genetic variants, their interactive effects, and/or their interaction with environment factors [2].

The interactive effect between two or more genetic variants is often referred to as the *epistatic interaction* or *epistasis*. Recent studies in medical genetics have suggested that epistatic interactions may contribute to the pathogenesis of complex diseases ubiquitously through sophisticated regulatory mechanisms in the molecular level of human genetics [3]. Examples of epistatis include synergistic effects of angiotensin-converting enzyme and angiotensin-II type 1 receptor gene on the risk of myocardial infarction [4], hypertension susceptibility through the interaction of angiotensin converting enzyme and G protein-coupled receptor kinase [5], association of a two-locus interaction between an uncoupling protein gene and a peroxisome proliferator-activated receptor gamma gene with type 2 diabetes mellitus [6], the influence of an interaction of KIR3DL1 and HLA-B loci on both AIDS progression and plasma HIV RNA abundance [7], and many others. With these examples, researchers now believe that epistatic interactions are the major causative patterns of complex diseases. The detection of epistatic interactions therefore plays a key role in the understanding of the pathogenesis of complex diseases.

Traditional statistical approaches such as family-based linkage analysis and population-based association studies have shown remarkable successes in the detection of individual causative genetic variants underlying Mendelian diseases, but they encounter difficulties in detecting epistatic interactions. For example, linkage analysis that uses a transmission model to explain the pattern of inheritance of phenotypes and genotypes exhibited in a pedigree is ineffective when a single locus fails to explain a significant fraction of a disease, though it works well for Mendelian diseases [2].

On the other hand, with the accumulation of well-phenotyped cases and carefully selected controls, as well as the emergence of high-throughput genotyping techniques, great opportunities and challenges are being presented for uncovering the genetic basis of complex diseases [8, 9]. Genetic variation occurring in the single basis of DNA sequences is referred to as *single nucleotide polymorphism (SNP)* and is the major form of genetic variation. The latest biotechnology has been able to detect several hundred thousand SNPs using a single chip. With such a huge number of genetic variants available for a large number of cases and controls, it becomes urgent to develop effective methods to detect epistatic interactions for *genome-wide association studies (GWAS)*.

To embrace the opportunities in genome-wide association studies, several multi-locus approaches that follow a "selection-testing-correction" scheme have been developed. In this scheme, one selects a small number of SNPs using statistical or machine learning methods, tests significance of interactions between the selected SNPs, and then adjusts the p-values to account for multiple-testing problem. For example, the step-wise logistic regression method selects a small fraction of ε% SNPs according to their marginal effects, uses

*: To whom correspondence should be addressed.
Email: ruijiang@tsinghua.edu.cn.

978-0-7695-3739-9/09 $25.00 © 2009 IEEE
DOI 10.1109/IJCBS.2009.32

logistic regression with likelihood ratio test to obtain *p*-values of the pairwise interactions of the selected SNPs, and then applies Bonferroni correction to adjust the *p*-values [10]. BEAM selects a small number of SNPs using a Bayesian model with an MCMC sampling strategy, uses a B-statistic to obtain *p*-values of up to three-way interactions of the selected SNPs, and applies Bonferroni correction to adjust the *p*-values [11]. *epi*Forest selects a small number of SNPs using a machine learning method called random forest, uses the B-statistic to obtain *p*-values of up to three-way interactions of the selected SNPs, and again applies Bonferroni correction to adjust the *p*-values [12].

The principle of the selection-testing-correction scheme is to reduce the number of statistical tests for interactions by discarding SNPs that have weak marginal effects and thus reduce the computational burden to a manageable level. For example, in the step-wise logistic regression method, a SNP needs to be ranked at the top ε% according to its marginal effect in order to pass the initial screening. Obviously, with the restriction of the computational power, one has to set this ε% to a small number (e.g., 10%). As a result, a large proportion of data is purely wasted.

A solution to avoid the waste of the data is to resort to the massive parallel computing techniques to increase the computational power. For example, with the use of a cluster that has hundreds of nodes, exhaustively searching for all pairwise interactions can be done in reasonable time. With this consideration, we implement a program called *epi*MPI for genome-wide association studies using the *Message Passing Interface (MPI)* [13]. This program can be scaled up to clusters with hundreds of nodes and is capable of exhaustively searching for pairwise interactions between hundreds of thousands of SNPs in a large data set that contains thousands of cases and controls.

We further explore the use of *Graphics Processing Units (GPU)* for genome-wide association studies. As a proof of concept, we implement the classical χ^2 test for detecting pairwise interactions with the use of the *Compute Unified Device Architecture (CUDA)* [14] on the latest NVIDIA GTX 280 graphics card and call this program *epi*CUDA. Results show that a single GTX 280 can be about 15 times as fast as an Intel Xeon E5420 CPU. Comparisons of the GPU program versus the MPI version suggest that a single GTX 280 have similar performance as a 16-node cluster. We further apply the GPU version of the program to two real genome-wide association studies data. For an *Age-related Macular Degeneration (AMD)* data set [8], the GPU program finishes the exhaustive search in about 6 minutes and is 17.7 times as fast as the CPU version. For a Parkinson's disease data set [9], the GPU program finishes the exhaustive search in about 2.8 hours and is 25.7 times as fast as the CPU version.

II. METHODS

A. Genome-Wide Association Studies

A genome-wide case-control data set includes L SNP markers genotyped for a number of cases and controls. Each genotype may take three possible values, i.e., homozygosity of major alleles, homozygosity of minor alleles, and heterozygosity. In addition, a missing value may be placed if a genotype is unavailable due to experimental failure.

Given such a data set, the classical approach to the detection of single-locus association fits a full logistic regression model with a parameter for each observed genotype for each SNP and then tests the significance of the fitted model via a χ^2 approximation of the likelihood ratio test statistic [10]. Alternatively, a χ^2 test with up to 2 degrees of freedom can be directly applied to the contingency table that records the numbers of cases and controls for each genotype to test whether the distributions of a SNP are significantly different in the case and the control populations. Because a family of L tests should be performed for a total of L SNPs, a Bonferroni correction that multiplies the *p*-values with the number of SNPs is typically applied to ensure the family-wise error rate not exceeding a preset significance level α.

Similarly, in order to detect the epistatic interaction of a pair of SNPs, a full logistic regression model with at most 9 parameters can be fitted and tested, and the *p*-values should be multiplied by $L(L-1)/2$ according to the Bonferroni correction [10]. As an alternative, a χ^2 test with up to 8 degrees of freedom can be applied. Note that the marginal effects of individual SNPs should be subtract from the joint influence for the pair of interacting SNPs in the χ^2 test. Because the number of SNPs is typically huge (e.g., several hundred thousand), and the numbers of cases and controls are large (e.g., several thousand) in genome-wide case-control data, an exhaustive search for all possible combinations of two SNPs is usually computationally impractical unless some massive parallel computing techniques are used. To overcome this limitation, the stepwise approach first selects a small fraction of SNPs according to the significance of their single-locus associations and then tests the interactions between the selected SNPs [10].

B. Graphics Processing Units

Graphics Processing Units (GPUs) are traditionally used as specialized accelerators for triangle rasterization. Because a major application of GPUs is to service the large gaming market, GPUs are ubiquitous and relatively inexpensive. However, recent advances in the design of GPUs have also enabling the transition of their traditional role to general purpose computing such as performing intensive high throughput floating-point computation. The latest GPU is designed as a highly parallel, multithreaded, manycore processor, which can perform single-precision floating-point calculation at least one order of magnitude faster than the latest CPU. For example, the NVIDIA GTX 280 GPU (GT200 architecture) is designed to have 30 multiprocessors, each of which has 8 processors. As a result, this GPU has 240 cores and can offer about 1T FLOP/s (floating-point operations per second) single-precision floating-point computational power. In addition, the memory subsystems for GPUs are typically endowed with more than 10 times higher memory bandwidth than that of CPUs [14].

With the manycore architecture, the performance of a GPU is largely dependent on finding high degrees of parallelism, that is, an application running on a GPU must express

thousands of threads in order to effectively use the hardware capabilities. It is therefore of great importance to find large scale parallelism in order to fully utilize the computational power of GPUs. Another major concern is that GPUs may not be able to provide sufficient floating-point accuracy to be generally useful, because in the previous design (e.g., the NVIDIA G80 architecture) GPUs can only offer single-precession floating-point operations. Nevertheless, the support for double-precision floating-point operations has been incorporated into the latest GT200 architecture. As a result, GPU computing is now more mature to handing general purpose tasks.

C. *Compute Unified Device Architecture*

In order to help developers to harness the computational power of GPUs, NVIDIA offers the *Compute Unified Device Architecture (CUDA)*, a programming environment for its GPUs [14]. Using CUDA, software engineers can write GPU applications using the C programming language, thus greatly shortening the development cycle. Writing codes in C instead of low level programming languages also allows researchers to focus on the scientific problem itself instead of technique details.

As illustrated in Fig. 1, CUDA assumes that an application may execute on a physically separate device (GPU) that operates as a coprocessor to the host (CPU). A programmer usually organizes the computation into many *grids*, each of which is further organized as a set of *thread blocks*. Grids run on the device in a sequential way. Therefore all computation in a grid must finish before another grid is invoked. A thread block is composed of many threads that execute concurrently on one multiprocessor. A multiprocessor contains several processors, each of which can use a set of registers and an amount of local memory. All processors in a multiprocessor share a small amount of shared memory and can access the global memory of the device. There are in addition two types of specially designed caches: constant cache and texture cache. Besides, a CUDA application can also use the host memory. In the latest NVIDIA GTX 280, a device has 1GB global memory and contains 30 multiprocessors, each of which has 8 processors, 16K registers, and 16K shared memory. A thread block can include at most 1024 active threads, which can be identified using a one-, two-, or three-dimensional index.

It should be noted that the shared memory can be as faster as the registers, while the access to the local memory and global memory is relatively slow. The access to the host memory is even slower. Therefore, threads in a block should always try to use the shared memory. A typical way is to load data from hard drive to the host memory, copy them to the global memory, and then transfer them to the shared memory. It should also be noted that, although grids run sequentially on a device, enormous numbers of thread blocks can be launched in parallel in one grid. As a result, the total number of threads that can be launched in parallel is very high. In practice, one needs a large number of thread blocks to ensure that the computational power of the GPU can be efficiently utilized.

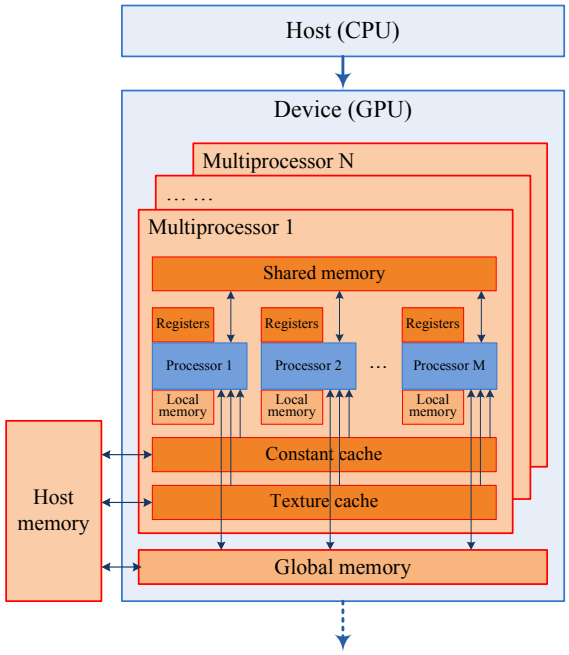

Figure 1. Illustration of the CUDA hardware model.

D. *GWAS using CUDA*

CUDA requires large scale parallelism to fully utilize the computational power of GPUs. For example, the calculation of pairwise epistatic interactions between a large number of L SNPs (e.g., L=500K) is analogous to fill an upper triangle matrix that contains as many as $L(L\text{-}1)/2$ test statistics. To implement large scale parallelism, we use one thread to calculate the test statistic of a single interaction between two SNPs. This idea suggests the following design of *epi*CUDA, as illustrated in Fig. 2.

First, we divide the upper triangle matrix into many grids, each of which can be thought of as a small square matrix with each dimension being *GridSize* (or *GS* for short). For simplicity, we assume that #{SNPs}/*GridSize* is an integer, and we call this number *GridDim* (or *GD* for short). In the case that #{SNPs}/*GridSize* is not an integer, we can simply add some pseudo-SNPs to make *GridDim* an integer. After this step, we obtain $GD(GD\text{+}1)/2$ grids, which will be send to the GPU in a sequential way.

Then, we divide each grid into many square blocks. The size of a block (*BlockSize* of *BS* for short) should be chosen so that the block can be fitted into a single multiprocessor of the GPU (remember the limitation of 1024 threads per thread block). For simplicity, we assume that *GridSize*/*BlockSize* is an integer, and we call this number *BlockDim* (or *BD* for short). After this step, we obtain $BD\times BD$ blocks, each of which corresponds to a thread block and will be dealt with by a single multiprocessor in the GPU.

Finally, each thread block contains $BS\times BS$ threads, each of which is identified by a two-dimensional index and used to calculate the test statistic or the p-value for a pair of two

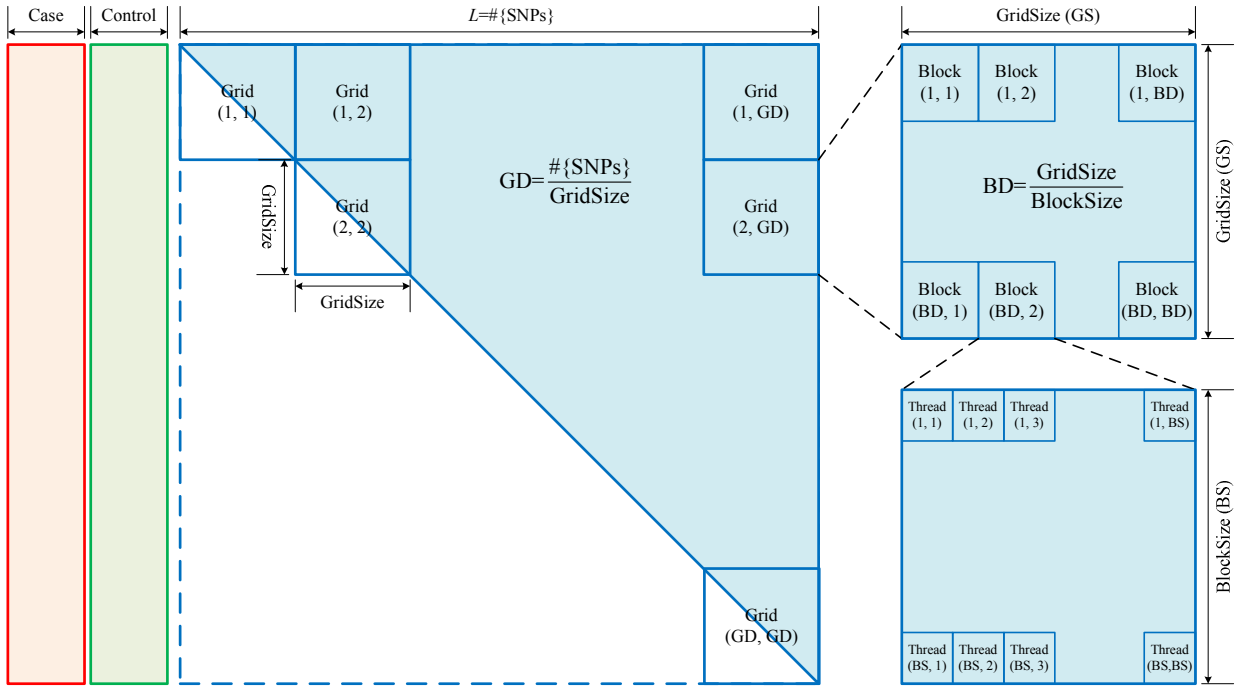

Figure 2. Principles of *epi*CUDA (detecting pairwise *epi*static interactions using CUDA).

SNPs. We use the first dimension (row) to index the first SNP in a pair and the second dimension (column) to index the second SNP in the pair. Consequently, each thread block calculates *BS*×*BS* interactions between *BS* row SNPs and *BS* column SNPs.

The selection of the parameters *BS* depends on the CUDA specification. In our studies, we select *BS*=16 so that each thread block contains 256 threads. After *BS* being determined, we select *BD*=64 so that each grid contains 4096 thread blocks and corresponds to pairwise interactions between *GS*=1024 SNPs. This parameter setting enables us to efficiently utilize the computational power of the latest GPU and is scalable to several generations of future GPUs.

In order to efficiently utilize the shared memory, we need to first copy the case-control data from the global memory into the shared memory. Since a thread block contains *BS*×*BS* threads, an efficient way is to use one thread to copy one case/control for a single SNP and synchronize the threads after all data being copied. Considering the limited size of the shared memory (16K), we compress the case-control data to enable a 32-bit word store 16 SNPs (2 bits for a SNP). With this design, we can deal with large scale case-control data in which the total number of cases and controls does not exceed 6000 in the current GPU. We can also reduce the parameter *BS* in the case that we need to deal with larger case-control data.

In order to efficiently utilize the computational power of the GPU, we use one thread to calculate the test statistic of a pair of SNPs. For example, in order to calculate the pairwise χ^2 test statistic, we need to first count the numbers of combinatory genotypes to fill out a 2×9 contingency table, and then calculate a 2×9 table for the expected numbers. Finally, the χ^2 test statistic is calculated as

$$\chi^2 = \sum_{i}^{2}\sum_{j}^{9}\frac{(o_{ij}-e_{ij})^2}{e_{ij}},$$

where o_{ij} is the number of observed, and e_{ij} is the number of expected. It should be noted that calculating *p*-values according to the χ^2 test statistics is straightforward in CPU, but not computationally economy in GPU. Therefore, the GPU will only report the calculated χ^2 statistics back to the CPU, and the calculation of the *p*-values using these statistics will be done in CPU.

When the number of SNPs is huge, it is not feasible to even store the calculated test statistics for all pairs of interactions in memory. For example, for a case-control data set that contains 500K SNPs, we need about 512G memory to store all test statistics as single-precision floating-point numbers. We therefore adopt a priority queue to store only a small fraction of calculated test statistics. For example, we only store #{SNPs} test statistics that can yield the smallest *p*-values. This procedure is equivalent to perform a partial sort routine on all pairwise interactions and report the most significant #{SNPs} ones. Since grids are sent to the GPU in a sequential way, we also sequentially insert the test statistics calculated for a grid into the priority queue.

It should be noted that the partial sort routine is not computational economy, because it runs on the CPU instead of the GPU. To overcome this limitation, we maintain an array of dynamic thresholds which corresponds to the minimum test statistics for different degrees of freedom (corresponds to

TABLE 1. PERFORMANCE OF *epi*CUDA in terms of the computation time (standard derivation), in seconds, and acceleration rate.

#{SNPs}	NVIDIA GTX 280	Intel Xeon E5420	Acceleration rate
1K	0.76 (0.01)	8.05 (0.02)	10.59
2K	1.82 (0.01)	23.63 (0.02)	12.98
4K	5.50 (0.01)	78.12 (0.03)	14.20
8K	19.14 (0.02)	280.57 (0.18)	14.66
16K	70.82 (0.64)	1058.88 (0.97)	14.95
32K	275.82 (1.56)	4115.95 (13.21)	14.92
64K	1091.41 (2.97)	16185.38 (28.36)	14.99
128K	4334.30 (11.33)	64189.48 (116.25)	14.81

the maximum *p*-value) in the priority queue, and we use the GPU to do the initial screening according to the thresholds. With this technique, only interaction pairs that can pass the thresholds will be inserted into the priority queue, and thus the computation time can be reduced. After a grid being computed and inserted into the priority queue, we update the array of thresholds. Consequently, as the computation goes on, thresholds for the test statistics increase, and the number of interaction pairs that need to be inserted into the priority queue decreases. We store this array of dynamic thresholds in the constant cache of the GPU.

With the priority queue that stores the test statistics for the most significant pairs, we can calculate their *p*-values, apply the Bonferroni correction to account for the multiple-testing problem, and report only significant pairs according to a predefined significance level.

E. GWAS using MPI

The above design for detecting pairwise epistatic interactions can be easily modified to run on a cluster following a master-slave design with the use of *message passing interface (MPI)* [13]. We refer to this design as *epi*MPI.

For a cluster has a number of N nodes, we use a node as the master to assign tasks to slaves, and we use the rest nodes as slaves, each of which deals with one grid. However, we do not need to further partition the grid into blocks. After the pairwise interactions for SNPs in a grid being calculated, we store the results in a temporary file system and use the master to insert the results into the priority queue, which is maintained by the master.

III. RESULTS

A. Performance of epiCUDA

In order to evaluate the performance of the CUDA implementation for detecting pairwise epistatic interactions, we simulate a number of case-control data sets on the basis of a genome-wide case-control data set of the Parkinson's disease, which contained 408,803 SNPs genotyped with 270 cases and 271 controls [9]. Briefly, we first add extra cases and controls to the data set so that the extended data set contains 1024 cases and 1024 controls. In this procedure, the genotypes for the added cases or controls are sampled according to the frequencies of the genotypes in the original data of the Parkinson's disease. Then, we extract from the extended data set 1K, 2K, 4K, 8K, 16K, 32K, 64K, and 128K SNPs. Consequently, we obtain 8 data sets that contain 1K to 128K SNPs, and each of these data sets contains 1024 cases and 1024 controls.

With these data sets, we compare the running time of the CUDA implementation for detecting pairwise epistatic interactions with that of the CPU counterpart. The computation time for the CUDA version is obtained using the latest NVIDIA GTX 280, and the time for the CPU version is obtained using Intel Quad-Core Xeon E5420, which runs at 2.5GHz. Note that the CPU implementation is written the using C programming language and is already dozens of times faster than the Matlab or R implementation.

For each of the above simulated data set, we run the program on GPU and CPU separately, and we record their running time. We further repeat the computation for every data set 10 times to estimate the standard derivation. The results are shown in Table 1. As we can see from the table, for the smallest data set (1K), the CUDA version can be about 11 times as faster as the CPU version, while for larger data sets, the acceleration rate can be about 15. The main reason that the acceleration rates for small data sets is not as high as those for large data sets is that the partial sort routine that is maintained in CPU is not accelerated by the GPU. Consequently, this part of computation time is the same for the CUDA version and the CPU counterpart. When the data set goes large, the cost for the partial sorting routine can almost be ignored, due to the dynamic threshold.

We also observe that the computation time for the CUDA program increases in a quadratic way with the increase of the number of the SNPs. Simple linear regression using the computation time as the response variable and the square of the number of SNPs as the predictor variable yields a very significant model ($r^2 \approx 1$, p-value $< 2.2\times10^{-16}$). With this model, we infer that the computation time for exhaustively searching for pairwise epistatic interactions for a typical 500K data set that contains 1000 cases and 1000 controls will be about 18 hours with the use of the CUDA program. In contrast, the CPU version will use more than 11 days to finish the same calculation.

We also evaluate the performance of the MPI implementation for detecting pairwise epistatic interactions. Taking the 64K data set as an example, with 16 slave nodes (each of which has an Intel Quad-Core Xeon E5420), the acceleration rate is 14.96, almost the same as the CUDA implementation. This observation suggests that the CUDA program is comparable to a cluster with 16 nodes.

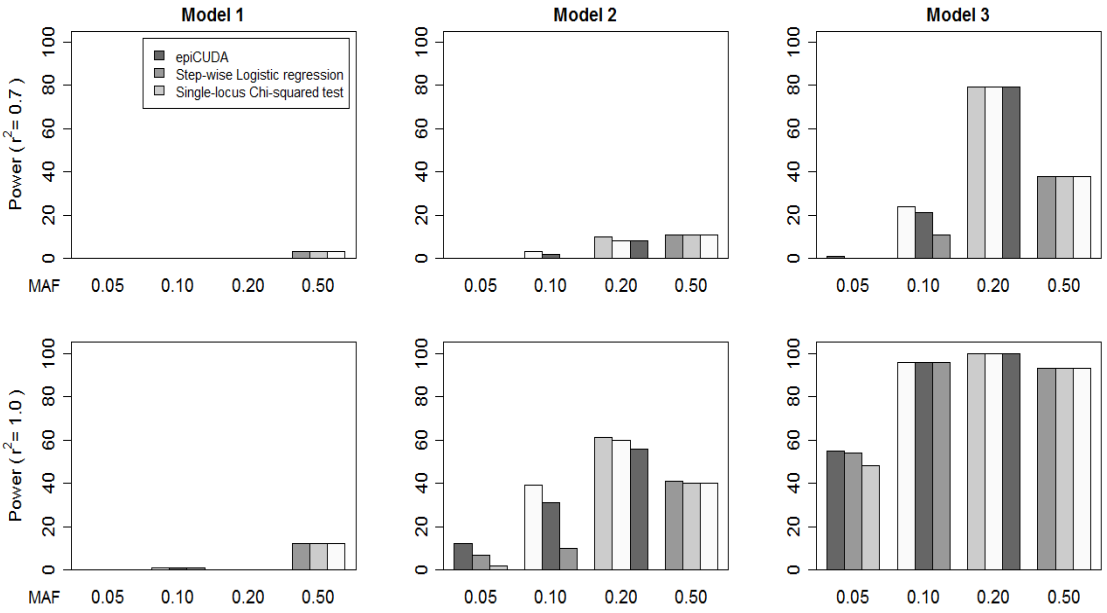

Figure 3. Accuracy of *epi*CUDA. The power of *epi*CUDA is compared with that of the step-wise logistic regression and the single-locus Chi-squared test.

B. Accuracy of epiCUDA

To verify the accuracy of the CUDA program, we consider three disease models with different characteristics (see [10] for details). Model 1 contains two disease loci that contribute to the disease risk independently. Model 2 is similar to model 1, except that the disease risk is present only when both loci have at least one disease allele. Model 3 is a threshold model in which additional disease alleles at each locus do not further increase the disease risk. Assuming the disease prevalence to be 0.1, each model has three parameters associated: the marginal effect of each disease locus (λ), the minor allele frequencies (MAF) of both disease loci, and the strength of linkage disequilibrium (LD) between the unobserved disease locus and a genotyped locus (r^2). We select only some typical values for each parameter. For λ, we set it to 0.3, 0.5, and 1.0 for model 1, 2, and 3, respectively. For MAF, we considered four values, 0.05, 0.1, 0.2, and 0.5, for each model. For r^2, we simulated for each model two values, 0.7 and 1.0. There were therefore 8 parameter settings for each disease model and a total of 24 comparisons in our simulation studies.

For each parameter setting of each model, we simulate 100 datasets, each containing 1,000 markers genotyped for 1,000 cases and 1,000 controls. The minor allele frequency for each non-disease marker is randomly chosen from a uniform [0.0. 0.5] distribution. The power of a method on a specific parameter setting is defined as the fraction of simulated datasets in which all disease loci are identified at the significance level $\alpha = 0.05$ after the Bonferroni correction.

In the comparison, *epi*CUDA performs exhaustive search for pairwise interactions. The stepwise logistic regression first selects the most significant ε% of SNPs according to their marginal effects, and then tests all pairwise interactions of these SNPs using logistic regressions with likelihood ratio tests [10]. Here we use ε%=25% to ensure that the overall computation time for this method is comparable to that of *epi*CUDA, thus achieving a fair comparison. The classical single-locus χ^2 test is used as a benchmark.

As shown in Fig. 3, *epi*CUDA achieve superior power over the stepwise logistic regression, while both of them are superior to the single-locus χ^2 test. Specifically, all methods have similar powers in model 1 regardless of the LD strength. This observation is natural because model 1 is actually a non-epistasis model, in the sense that the two causative loci contribute to the disease independently. In models 2 and 3, *epi*CUDA show superior power, especially when the minor allele frequencies of the disease markers are small. This might be attributed to the benefit of using the exhaustive search strategy. We also notice that the standard χ^2 test, as a single-locus method, performs poorly when the minor allele frequencies of the disease markers are small, suggesting the necessity of developing multi-locus approaches.

It is not surprising that *epi*CUDA can achieve the highest power for all models over all parameter settings, because the exhaustive search strategy, in nature, exams all possible pairwise interactions. What we want to emphasize is that, with similar computation time, *epi*CUDA can be more powerful. As the scale of case-control studies goes larger, the benefit of using CUDA to accelerate the search for epistatic interactions will become more obvious.

C. Applications to real data sets

We further apply the CUDA program to two real genome-wide case-control data sets. The first one is an Age-

related Macular Degeneration (AMD) data set [8], which contains 116,204 SNPs genotyped with 96 cases and 50 controls. The second data set is about the Parkinson's disease and contains 408,803 SNPs genotyped with 270 cases and 271 controls [9].

For the AMD data, the CUDA program exams all pairwise interactions of these SNPs within 6 minutes and achieve an acceleration rate of 17.1 over the CPU version. For the Parkinson's disease data, the CUDA program exhaustively searches for all pairwise interactions in about 2.8 hours and is 25.7 times as fast as the CPU counterpart.

Nevertheless, we do not find any pairwise epistatic interactions in these two data sets. The main reason might be the small sample size is insufficient for detecting subtle epistatic interactions [11]. Another reason may be the problem of genotype missing which aggravates the insufficiency of sample size in mapping epistatic effects.

IV. Conclusions and Discussion

In this paper, we describe an approach that uses graphics processing units (GPU) to accelerate the detection of pairwise epistatic interactions for genome-wide association studies. We implement the GPU program based on the compute unified device architecture (CUDA), and we compare this program against its CPU counterpart. Results show that the GPU program running on NVIDIA GTX 280 can be 15 times as fast as the CPU version running on Intel Xeon E5420 and is comparable to a cluster with 16 nodes. Applications to two real data sets of AMD and Parkinson's disease suggest even higher acceleration rate of 17.1, and 25.7, respectively.

As a proof of concept implementation, we only report the χ^2 test results in this paper. An extension that uses the logistic regression is under development. Besides, a step-wise approach that can detect three-way interactions using CUDA is also under development. We also plan to integrate other methods in statistical genetics and release a CUDA package to facilitate genome-wide association studies. Recently, NVIDIA releases its personal super-computer that contains four Tesla GPU cards, each with 240 processors and 4G global memory. With the arrival of such a powerful product that includes 960 processors and 16G memory, our CUDA package could be widely used by many laboratories in their genome-wide association studies.

We have demonstrated that appropriate usage of GPU can greatly reduce the computation time of bioinformatics applications. The improvement in performance mainly depends on the large scale parallelism of the applications. It is therefore very important to divide an application into a large number of independent units. These units should share identical codes and deal with different part of the data. This programming model is called fine-grained data parallelism and thread parallelism [14]. Nevertheless, the current CUDA library still lacks sufficient support for functions in statistics, e.g., the support for probability distribution functions that are heavily used in Markov Chain Monte Carlo simulations. This limitation has been restricting the application of the CUDA programming model to many bioinformatics applications. Recently, Intel announces its roadmap of the *Larrabee* GPU architecture, which integrates dozens of simplified x86 processors. Since each of the processors has much stronger control units over the multiprocessors in the current GPU, large-scale parallel implementations of many sophisticated bioinformatics applications might become possible with the use of this architecture.

Acknowledgments

This work was partially supported by Natural Science Foundation of China grant 60805010, Tsinghua National Laboratory for Information Science and Technology (TNLIST) Cross-discipline Foundation, Research Fund for the Doctoral Program of Higher Education of China, Scientific Research Foundation for Returned Overseas Chinese Scholars, Research Fund from Intel China Research Center, and a starting up supporting plan at Tsinghua University.

References

[1] E. S. Lander, and N. J. Schork, "Genetic dissection of complex traits," *Science,* vol. 265, no. 5181, pp. 2037-48, Sep 30, 1994.

[2] A. M. Glazier, J. H. Nadeau, and T. J. Aitman, "Finding genes that underlie complex traits," *Science,* vol. 298, no. 5602, pp. 2345-9, Dec 20, 2002.

[3] J. H. Moore, and S. M. Williams, "New strategies for identifying gene-gene interactions in hypertension," *Ann Med,* vol. 34, no. 2, pp. 88-95, 2002.

[4] L. Tiret, A. Bonnardeaux, O. Poirier *et al.*, "Synergistic effects of angiotensin-converting enzyme and angiotensin-II type 1 receptor gene polymorphisms on risk of myocardial infarction," *Lancet,* vol. 344, no. 8927, pp. 910-3, Oct 1, 1994.

[5] S. M. Williams, M. D. Ritchie, J. A. Phillips, 3rd *et al.*, "Multilocus analysis of hypertension: a hierarchical approach," *Hum Hered,* vol. 57, no. 1, pp. 28-38, 2004.

[6] Y. M. Cho, M. D. Ritchie, J. H. Moore *et al.*, "Multifactor-dimensionality reduction shows a two-locus interaction associated with Type 2 diabetes mellitus," *Diabetologia,* vol. 47, no. 3, pp. 549-54, Mar, 2004.

[7] M. P. Martin, Y. Qi, X. Gao *et al.*, "Innate partnership of HLA-B and KIR3DL1 subtypes against HIV-1," *Nat Genet,* vol. 39, no. 6, pp. 733-40, Jun, 2007.

[8] R. J. Klein, C. Zeiss, E. Y. Chew *et al.*, "Complement factor H polymorphism in age-related macular degeneration," *Science,* vol. 308, no. 5720, pp. 385-9, Apr 15, 2005.

[9] H. C. Fung, S. Scholz, M. Matarin *et al.*, "Genome-wide genotyping in Parkinson's disease and neurologically normal controls: first stage analysis and public release of data," *Lancet Neurol,* vol. 5, no. 11, pp. 911-6, Nov, 2006.

[10] J. Marchini, P. Donnelly, and L. R. Cardon, "Genome-wide strategies for detecting multiple loci that influence complex diseases," *Nat Genet,* vol. 37, no. 4, pp. 413-7, Apr, 2005.

[11] Y. Zhang, and J. S. Liu, "Bayesian inference of epistatic interactions in case-control studies," *Nat Genet,* vol. 39, no. 9, pp. 1167-73, Sep, 2007.

[12] R. Jiang, W. Tang, X. Wu *et al.*, "A random forest approach to the detection of epistatic interactions in case-control studies," *BMC Bioinformatics,* vol. 10, no. Suppl 1, pp. S65, 2009.

[13] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*: MIT Press, 1994.

[14] NVIDIA, *NVIDIA CUDA Programming guide (version 2.0)*: NVIDIA, 2008.