GPU-Accelerated Exhaustive Search for Third-Order Epistatic Interactions in Case-Control Studies

Jorge González-Domínguez*, Bertil Schmidt

Parallel and Distributed Architectures Group, Institute of Computer Science, Johannes Gutenberg University
Staudingerweg 9, 55128 Mainz, Germany

Abstract

Interest in discovering combinations of genetic markers from case-control studies, such as Genome Wide Association Studies (GWAS), that are strongly associated to diseases has increased in recent years. Detecting epistasis, i.e. interactions among k markers ($k \geq 2$), is an important but time consuming operation since statistical computations have to be performed for each k-tuple of measured markers. Efficient exhaustive methods have been proposed for $k = 2$, but exhaustive third-order analyses are thought to be impractical due to the cubic number of triples to be computed. Thus, most previous approaches apply heuristics to accelerate the analysis by discarding certain triples in advance. Unfortunately, these tools can fail to detect interesting interactions. We present GPU3SNP, a fast GPU-accelerated tool to exhaustively search for interactions among all marker-triples of a given case-control dataset. Our tool is able to analyze an input dataset with tens of thousands of markers in reasonable time thanks to two efficient CUDA kernels and efficient workload distribution techniques. For instance, a dataset consisting of 50,000 markers measured from 1,000 individuals can be analyzed in less than 22 hours on a single compute node with 4 NVIDIA GTX Titan boards. Source code is available at: `http://sourceforge.net/projects/gpu3snp/`

Keywords: GPU, CUDA, epistasis, GWAS, mutual information

*Principal corresponding author: Jorge González-Domínguez; Tel.: +49-6131-39-23615
Email address: `j.gonzalez@uni-mainz.de` (Jorge González-Domínguez)

1. Introduction

Genotype-phenotype association studies can contribute towards the identification of genetic variants that are associated with certain diseases. In classical analysis, Single Nucleotide Polymorphisms (SNPs) are studied separately in order to identify markers showing differences in genotype frequencies between cases and controls. Unfortunately, this approach is not powerful enough to model complex traits for which the detection of joint genetic effects (epistasis) needs to be considered [1–3].

However, detecting epistasis for k-tuples with $k > 2$ is computationally expensive due to the combinatorial explosion of combinations that arise. In fact, some related works have assumed that exhaustive search for interaction on all the third-order combinations is only feasible for small datasets with hundreds of SNPs [4]. Thus, most approaches discard a large number of non-interesting triples during the search procedure. For instance, BEAM [5] and its extension epiMODE [6] use Markov Chain Monte Carlo (MCMC) to calculate the probability of a SNP being part of a combination associated to the disease. Another approach consists of stepwise algorithms that only analyze those combinations that contain a subset of SNPs that are selected at the beginning [7]. Non-exhaustive approaches based on the clustering of relatively frequent items [4, 8] and on machine learning techniques [9, 10] are also becoming popular. Unfortunately, these non-exhaustive tools may discard interesting SNP-triples. Despite being highly time-consuming, there also exist exhaustive-search strategies that analyze all the possible triples. Some well-known examples are the Combinatorial Partitioning Method (CPM) [11], the Restriction Partition Method (RPM) [12] and the Multifactor Dimensionality Reduction (MDR) method [13] (and its extensions MB-MDR [14] or RMDR [15]). However, their use is limited to extremely small datasets because of their slow speed.

As an attempt to overcome this limitation, several tools use High Performance Computing (HPC) to accelerate the exhaustive analysis even for Genome Wide Association Studies (GWAS), either with GPUs [16–21], FPGAs [22], Xeon Phis [23] or clusters [24–27]. However, they are only able to look for pairwise epistatic interactions. In this work we present GPU3SNP, a new multi-GPU tool that addresses the exhaustive search for third-order epistatic interactions. It receives a dataset with biallelic information as input and returns a list of SNP-triples that, according to the interaction metric selected by the user, present the highest probabilities of discriminating between the presence and absence of the disease based on mutual information or information gain measure. Our approach is able to provide high accuracy in comparison to non-exhaustive methods. Furthermore, it is able to analyze datasets with tens of thousands of SNPs in reasonable time, which is not possible by existing exhaustive tools.

The rest of the paper is organized as follows. Section 2 provides some necessary background information about the utilized methodology to search for third-order epistatic interactions as well as information about GPUs and the CUDA language. Our parallelization approach and the employed optimization techniques are described in Section 3. Experimental evaluations are presented in Section 4. Section 5 concludes the paper.

2. Background

2.1. Contingency Tables

Case-control datasets contain information about a large number of biallelic genetic markers (typically SNPs) from many individuals. For each SNP there are three genotypes: homozygous wild (w), heterozygous (h) and homozygous variant (v). They are numerically represented as {0,1,2}, respectively. The number of SNPs and individuals is denoted as M and N, respectively. The individuals are categorized as cases (value 0) and controls (value 1). The first step in order to detect interaction among SNPs is the creation of contingency tables. The contingency tables are

Table 1: Example of contingency table

Cases		SNP3=0	SNP3=1	SNP3=2
SNP1=0	SNP2=0	n_{0000}	n_{0010}	n_{0020}
	SNP2=1	n_{0100}	n_{0110}	n_{0120}
	SNP2=2	n_{0200}	n_{0210}	n_{0220}
SNP1=1	SNP2=0	n_{1000}	n_{1010}	n_{1020}
	SNP2=1	n_{1100}	n_{1110}	n_{1120}
	SNP2=2	n_{1200}	n_{1210}	n_{1220}
SNP1=2	SNP2=0	n_{2000}	n_{2010}	n_{2020}
	SNP2=1	n_{2100}	n_{2110}	n_{2120}
	SNP2=2	n_{2200}	n_{2210}	n_{2220}
Controls		**SNP3=0**	**SNP3=1**	**SNP3=2**
SNP1=0	SNP2=0	n_{0001}	n_{0011}	n_{0021}
	SNP2=1	n_{0101}	n_{0111}	n_{0121}
	SNP2=2	n_{0201}	n_{0211}	n_{0221}
SNP1=1	SNP2=0	n_{1001}	n_{1011}	n_{1021}
	SNP2=1	n_{1101}	n_{1111}	n_{1121}
	SNP2=2	n_{1201}	n_{1211}	n_{1221}
SNP1=2	SNP2=0	n_{2001}	n_{2011}	n_{2021}
	SNP2=1	n_{2101}	n_{2111}	n_{2121}
	SNP2=2	n_{2201}	n_{2211}	n_{2221}

used in order to store the number of individuals for each combination of genotypes in each cell. Table 1 shows the contingency table of size 3x3x3x2 for a given SNP-triple, where the cell $ijkc$ stores the number of individuals categorized as c (case or control) with the value of the first SNP as i, the second SNP as j and the third SNP as k. We can also fill the contingency table with probabilities: $\pi_{ijkc} = n_{ijkc}/N$.

2.2. Filtering Stage

Once the contingency tables are created, their values are used to identify statistically significant SNP-triples presenting epistasis. Several measures have been used to define significance. Examples include Chi-square tests [7, 8], regression models [18, 28], ROC-curves [20], dependency difference [19], Mutual Information (MI) [4] and Information Gain (IG) [29]. GPU3SNP allows the user to choose between MI and IG as they have been shown to be accurate in the presence of contingency table cells with low counts, which is quite common when looking for third-order epistasis. MI is very efficient detecting epistasis with interaction effects from lower-order tuples. On the other hand, IG is more suitable for users who are only interested in pure three-way epistasis, where the lower-order effects (pairs) do not present interaction. Anyway, the tool is flexible enough to include more measures in next versions.

2.3. GPU Architecture and CUDA

GPU3SNP is able to exploit several GPUs within the same node using CUDA [30], a parallel programming language that extends the general programming languages with a set of abstractions to express parallelism. A CUDA program is comprised of code for the host and kernels for the devices. A kernel is a program launched over a set of lightweight parallel threads on GPUs, where the threads are organized into a grid of thread blocks. All threads in a thread block are split into small groups of 32 parallel threads, called warps, for execution. These warps are scheduled in a single instruction, multiple thread fashion. Full efficiency and performance can be obtained when all threads in a warp execute the same code path. CUDA threads cannot directly access to host memory, so input/output data must be copied between CPU and GPU (device) memory before and after computation. Moreover, CUDA threads within the same blocks can exploit a special shared memory, which is faster but smaller (around 48 KB) than device memory (around several GB). It is the programmers' responsibility to exploit the memory hierarchy in order to improve performance.

Figure 1 shows the structure of the GPU architecture. CUDA-enabled GPUs have evolved into highly parallel many-core processors with tremendous compute power and very high memory bandwidth. They are especially well-suited to address computational problems with high data parallelism and arithmetic density. A CUDA-enabled GPU can be conceptualized as a fully configurable array of Scalar Processors (SPs). These SPs are further organized into a set of Streaming Multiprocessors (SMs).

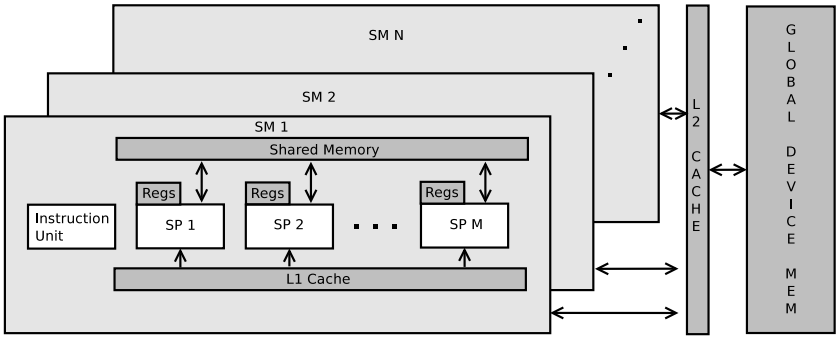

Figure 1: GPU architecture and memory hierarchy.

3. Parallel Implementation

3.1. Contingency Table Creation

The creation of the contingency tables is the most time-consuming step in our algorithm, with time complexity $O(M^3 * N)$. Thus, we optimize this step by using an efficient boolean representation of the input data, which has already been proved adequate for GPU computation in previous approaches [18, 21]. Instead of using the naive representation of the SNP information with an $M \times N$ table where each row is encoded using 2 bits per entry to distinguish among the three genotypes ({0,1,2}), we allocate three binary strings per row, one for each possible genotype. These strings use one bit per entry that indicates whether the sample has the corresponding genotype. We refer to [28] for further explanation.

The main advantage of this approach is that the cells of the contingency table can be calculated using bit operations, which is faster than working with integer arithmetic. Specifically, the logical AND and the counting of the number of ones in a bit string (*popcount*) are used. We have adapted the idea present in [18, 21] to third-order analyses so that the cell $ijk0$ is calculated just counting the 1-bits in the part for the cases of the string resulting from SNP_i^1 & SNP_j^2 & SNP_k^3 (SNP_i^1, SNP_j^2 and SNP_k^3 are the bit strings that represent whether the individuals have the genotype i, j and k in $SNP1$, $SNP2$ and $SNP3$, respectively). The same approach could also be applied for controls. The *popcount* is implemented using the highly efficient *__popc* routine available in the Integer Intrinsics library of CUDA. An example using 3 SNPs and 6 cases is illustrated in Table 2. In general, we can represent the value of the cases cells of the contingency table as:

$$n_{ijk0} \quad = \quad popcount(SNP_i^1 \& SNP_j^2 \& SNP_k^3) \qquad (1)$$

The calculation of n_{ijk1} is similar but using the bits related to the controls. According to Equation 1, we would need two logical AND operations per cell of each SNP-triple (in total, 108 AND operations per SNP-triple). However, we can reuse the value of $SNP_i^1 \& SNP_j^2$ for three cells (n_{ij00}, n_{ij10} and n_{ij20}), reducing the number of AND operations to 72. As each AND operation requires many memory accesses, this 33% reduction of AND operations can lead to a significant performance improvement. Thus, GPU3SNP initially calculates auxiliary results of the AND operations for the SNP-pair formed by the two first SNPs of the triple in order to reuse them for the creation of the contingency table.

3.2. CUDA Kernels

The goal of our implementation is to provide a list with the l third-order combinations containing the highest MI or IG. We store the auxiliary results of the 2-SNP-AND operations in the device memory. The computation of SNP-pairs is divided into batches and two consecutive

Table 2: From left to right: naive input data; input data of the tool using binary strings; calculation of the n_{0000} cell of the contingency table.

$SNP1 = 000000$	$SNP_0^1 = 111111$	$SNP_1^1 = 000000$	$SNP_2^1 = 000000$	$n_{0000} = popcount\Big((111111)\&(111000)\&(101010)\Big)$
$SNP2 = 000112$	$SNP_0^2 = 111000$	$SNP_1^2 = 000110$	$SNP_2^2 = 000001$	$n_{0000} = popcount\Big(101000\Big)$
$SNP3 = 010201$	$SNP_0^3 = 101010$	$SNP_1^3 = 010001$	$SNP_2^3 = 000100$	$n_{0000} = 2$

kernels are applied to each batch. Both kernels use the same number of threads per block.

1. Kernel to create the 2-SNP-AND. Each CUDA thread works with one SNP-pair. It reads the information of the two SNPs (using the boolean representation explained in the previous section), calculates the nine possible &s and stores the results in device memory in order to be used by the next kernel.
2. Kernel to perform the third-order analysis. In this case we create one thread block per SNP-pair that analyses all the SNP-triples in which the first two SNPs are included. Thus, each thread will compute more than one SNP-triple. For each SNP-triple, the thread creates the contingency table as shown in Equation 1 and calculates the MI or IG value.

The output of the second kernel is a list per thread block that contains the l SNP-triples with the highest measure value. The host compares this new list to the list obtained from previous batches and gathers them in order to save only the overall l highest values.

Exploiting the strengths of the different levels of the memory hierarchy on the GPU is key to achieve high performance. In order to improve the memory accesses to device memory we have reorganized the biallelic information of each SNP. As we pack the bit values of the boolean representation in 32-bit arrays, the information is stored in three 32-bit arrays (one per genotype) of length $N/32$. Figure 2 shows how each array would look like if the entries of each SNP were consecutively ordered. In the second kernel consecutive threads access the information of consecutive SNPs. However, as there are $N/32$ entries per SNP (and N might be very large, in the order of several thousands of samples), we would generate uncoalesced memory accesses on the GPU because consecutive threads would access to positions of the arrays with distance $N/32$. This is the reason why the data of the two arrays are reordered when loaded into device memory, following the structure shown in Figure 3. In this case consecutive threads access consecutive memory positions, increasing the coalescence of the accesses and, thus, significantly improving performance.

Furthermore, the second kernel exploits CUDA shared memory in order to accelerate memory accesses. As explained before, each kernel call creates one block per SNP-pair (i,j) and all the threads within the block analyze all the possible SNP-triples that contain this pair: all (i,j,k) for $i < j < k$. The starting point for this computation is the 2-SNP-AND generated by the first kernel. This configuration blocks/threads has been chosen so that all threads within the same block can access the same 2-SNP-AND results. Therefore, threads can collaborate at the beginning of the kernel to copy the auxiliary 2-SNP-AND results to shared memory. After synchronization, all threads create the contingency tables with fast accesses to these values through shared memory.

Finally, at the end of the kernel each thread has a list with the l SNP-triples analyzed by it with highest measure value. These lists are reduced into only one l-sized list per block using a tree-based approach that also exploits shared memory.

3.3. Workload Distribution Among GPUs

In GPU3SNP batches of SNP-pairs are assigned to different GPUs so that each GPU can analyze all the possible third-order combinations generated with the pairs of the batch. The gathering of SNP-pairs into batches and

Figure 2: Example of one array with the information of one SNP without reordering the entries.

Figure 3: Example of one array with the information of one SNP when reordering the entries.

their distribution to GPUs is performed by the CPU using PThreads (one thread per GPU). Two types of distributions are used:

- Static approach where the workload is distributed in advance as each SNP-pair is initially associated to exactly one GPU. Each batch is created by using only pairs associated to the corresponding GPU. Note that the number of third-order combinations depends on the pair: for pair (i,j) there exist $M-j$ SNP-triples. Thus, in order to balance the workload among the GPUs, we follow a cyclic approach for the SNP-pair distribution.
- Dynamic distribution where only one batch is initially assigned to each GPU. Once a GPU completes the two kernels it requests another batch and the CPU provides the next available one. The assignment of batches to GPUs is controlled in a flexible way, since the GPUs can finish their computation in any order.

As will be shown in the next section, the dynamic approach is especially useful for systems with different types of GPUs, as the workload is variable and can be adapted to the speed of each GPU. More powerful GPUs complete their computations faster and request more batches from the CPU. However, as the SNP-pair distribution is not performed in advance, the CPU computation of the dynamic approach is more complex. The static distribution reduces this overhead as the CPU does not need to calculate the distribution. Consequently, it is appropriate for platforms with the same type of GPUs, as it equally distributes the workload among them.

4. Experimental Evaluation

4.1. Accuracy

We have compared GPU3SNP to the EDCF tool [8] in order to demonstrate that our exhaustive approach is more accurate than a representative non-exhaustive one for searching third-order epistatic interactions. EDCF has been selected as state-of-the-art because it has been shown in [8] to be more accurate than epiMode [6] and SNPRuler [10] (for its part, more accurate than MegaSNPHunter [9]). The EDCF statistic to measure interactions is based on the clustering of relatively frequent items. Additionally, the authors assume that tuples with epistasis must contain at least one lower-order interaction. Therefore, they reduce the search space by only analyzing the triples with at least one SNP-pair that presents interaction.

We have considered three disease models by expanding them from widely used pairwise models [31]. They are represented in Tables 3, 4 and 5 using the same notation as in several previous works [4, 5, 8, 31]. The major and minor alleles of the first, second and third SNP are represented as Aa, Bb and Cc, respectively. For each model, we have tried four different configurations by varying the values of α and θ. We have selected the proper values for α and θ so that the configurations combine two different minor allele frequencies (MAF) at 0.2 and 0.5, and two different marginal effect sizes (λ) at 0.3 and 0.5. Let p_0 and p_1 denote the probability that the genotype of a SNP

Table 3: Disease model with multiplicative effects between and within loci (Model 1)

	CC	**Cc**	**cc**
AABB	α	$\alpha * (1+\theta)$	$\alpha * (1+\theta)^2$
AABb	$\alpha * (1+\theta)$	$\alpha * (1+\theta)^2$	$\alpha * (1+\theta)^3$
AAbb	$\alpha * (1+\theta)^2$	$\alpha * (1+\theta)^3$	$\alpha * (1+\theta)^4$
AaBB	$\alpha * (1+\theta)$	$\alpha * (1+\theta)^2$	$\alpha * (1+\theta)^3$
AaBb	$\alpha * (1+\theta)^2$	$\alpha * (1+\theta)^3$	$\alpha * (1+\theta)^4$
Aabb	$\alpha * (1+\theta)^3$	$\alpha * (1+\theta)^4$	$\alpha * (1+\theta)^5$
aaBB	$\alpha * (1+\theta)^2$	$\alpha * (1+\theta)^3$	$\alpha * (1+\theta)^4$
aaBb	$\alpha * (1+\theta)^3$	$\alpha * (1+\theta)^4$	$\alpha * (1+\theta)^5$
aabb	$\alpha * (1+\theta)^4$	$\alpha * (1+\theta)^5$	$\alpha * (1+\theta)^6$

Table 4: Disease model with multiplicative effects between loci (Model 2)

	CC	**Cc**	**cc**
AABB	α	α	α
AABb	α	α	α
AAbb	α	α	α
AaBB	α	α	α
AaBb	α	$\alpha * (1+\theta)$	$\alpha * (1+\theta)^2$
Aabb	α	$\alpha * (1+\theta)^2$	$\alpha * (1+\theta)^4$
aaBB	α	α	α
aaBb	α	$\alpha * (1+\theta)^2$	$\alpha * (1+\theta)^4$
aabb	α	$\alpha * (1+\theta)^4$	$\alpha * (1+\theta)^7$

involved in a disease is equal to 0 and 1, respectively, λ is defined as:

$$\lambda = \frac{p_1/p_0}{(1-p_1)/(1-p_0)} \quad (2)$$

For each configuration we have simulated 100 datasets with 2,000 SNPs, 2,000 cases and 2,000 controls using genomeSIMLA [32]. There is one SNP-triple with epistasis per dataset (ground-truth). As EDCF also provides a list with the l SNP-triples with the highest interaction, we measure the power as the number of datasets on which the ground-truth is in the output list, with l equal to 10.

In order to provide a fair comparison between exhaustive and non-exhaustive methods, MI has been used for GPU3SNP as this measure has similar characteristics to EDCF. IG has been proved more accurate on some scenarios [29], but a deep analysis of the advantages and drawbacks of different measures is out of the scope of this work.

Table 5: Disease model with threshold effect (Model 3)

	CC	**Cc**	**cc**
AABB	α	α	α
AABb	α	α	α
AAbb	α	α	α
AaBB	α	α	α
AaBb	α	$\alpha * (1+\theta)$	$\alpha * (1+\theta)$
Aabb	α	$\alpha * (1+\theta)$	$\alpha * (1+\theta)$
aaBB	α	α	α
aaBb	α	$\alpha * (1+\theta)$	$\alpha * (1+\theta)$
aabb	α	$\alpha * (1+\theta)$	$\alpha * (1+\theta)$

Users can select the most suitable approach for their analyses when employing GPU3SNP.

The results for all the configurations are presented in Figure 4. They show that both algorithms are perfectly accurate for the three models when both MAF and λ are high (0.5). However, GPU3SNP outperforms EDCF for all the other scenarios. Models 2 and 3 with low MAF (0.2) are especially remarkable as EDCF is never able to detect the ground-truth while GPU3SNP achieves power 63% and 94% for Model 2 and 100% in both cases of Model 3. On average, GPU3SNP is 34% more accurate than EDCF. This improvement increases to 46% if we do not take into account the scenarios with MAF and λ equal to 0.5, where both tools present 100% power.

4.2. Execution Times and Speedups

Most of the experiments for the performance evaluation have been conducted on a system with a hex-core Intel Core i7 Sandy Bridge 3.20 GHz CPU with 12 MB cache, and two different NVIDIA Kepler GPUs, whose specifications are shown in Table 6. Firstly, we have evaluated the parallel efficiency of GPU3SNP. Instead of comparing it to very slow exhaustive tools such as CPM [11], RPM [12] or MDR [13] (see Section 1), we have developed an efficient PThreads CPU implementation of GPU3SNP. Note that, in order to provide a fair comparison, this CPU version includes the optimization techniques presented in Section 3.1 (i.e. a boolean representation of data and the reuse of the 2-SNP-AND information for the third-order

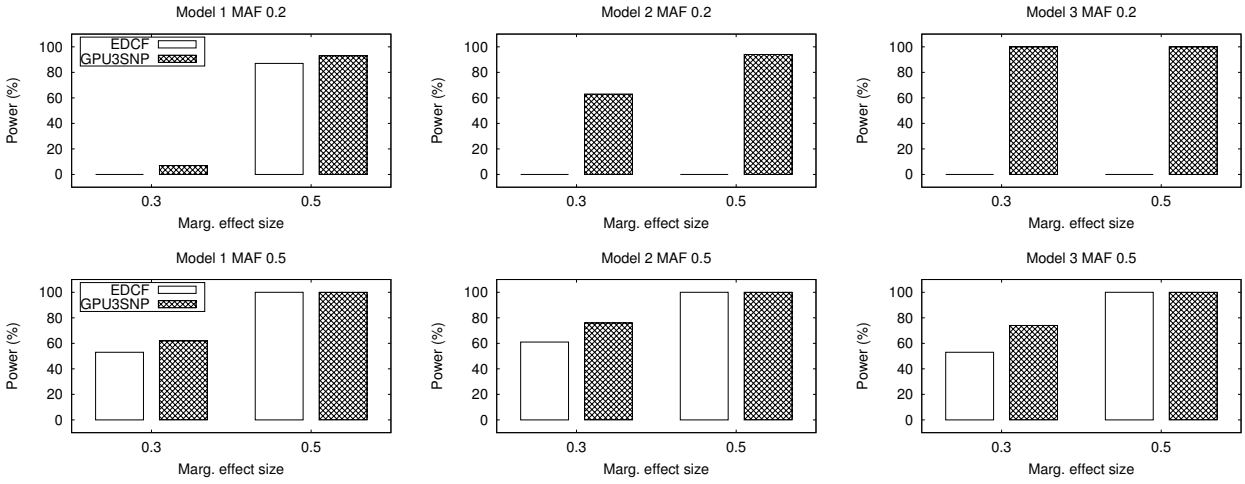

Figure 4: Accuracy comparison between EDCF and GPU3SNP (with MI as measure).

Table 6: Specifications of the GPUs used for the experimental evaluation

	GTX 650Ti	**GTX Titan**
Num. of cores	768	2,688
Core frequency	928 MHz	837 MHz
Memory size	2 GB	6 GB
Memory bandwidth	86.4 GBs	288.4 GBs

Table 7: Execution times (in minutes) of GPU3SNP and the multithreaded CPU implementation. In parenthesis, speedups of GPU3SNP over the CPU version.

SNPs	**Ind.**	**CPU (6th)**	**650Ti**	**Titan**
5,000	1,000	356.60	16.00 (22.29)	3.97 (89.82)
5,000	2,000	559.13	25.61 (21.83)	6.54 (85.49)
10,000	1,000	2,477.27	125.63 (19.72)	30.73 (80.61)

analyses). Moreover, it uses the same *popcount* and the same PThreads approach that has been proved efficient for the pairwise interaction tool presented in [21] so that it exploits the 6 cores of the system. The execution times (in minutes) of GPU3SNP for the two GPUs and the CPU implementation are shown in Table 7 for three simulated datasets with different number of SNPs and individuals. The speedups of GPU3SNP compared to the CPU version are include in parenthesis. Despite comparing to a multithreaded CPU version, the speedups are high for the three datasets: around 20 for the 650Ti and between 80 and 90 for the more powerful Titan. As there are no dependencies among the analysis of different SNP-triples, our kernels are able to fully exploit the parallel capacities of the GPUs.

Besides the high computational power of the GPUs, memory access optimizations explained in Section 3.2 are key in order to justify these high speedups. Figure 5 shows the percentage of the runtime used by each part of the computation (calculation of the 2-SNP-ANDs, creation of the contingency tables and filtering). While the CPU implementation spends more than 98% of the total time for any input datasets by creating the contingency tables (where most of memory accesses are performed), the GPU implementation reduces this percentage to around 75-82% thanks to the efficient coalesced accesses and the use of the fast shared memory. The filtering is independent of the number of individuals as it only accesses the 54 values of the contingency tables. Therefore, its influence on the runtime is higher for the smallest number of samples (1,000 individuals and around 22%) while it is only around 16% for datasets with 2,000 samples.

Additional experiments where the two GPUs are collaborating to analyze a dataset with 5,000 SNPs and 1,000 individuals have been performed in order to study the performance of the workload distribution approaches presented in Section 3.3. Execution times and speedups over the multithreaded CPU and the single-GPU executions are shown in Table 8. The dynamic approach, where the

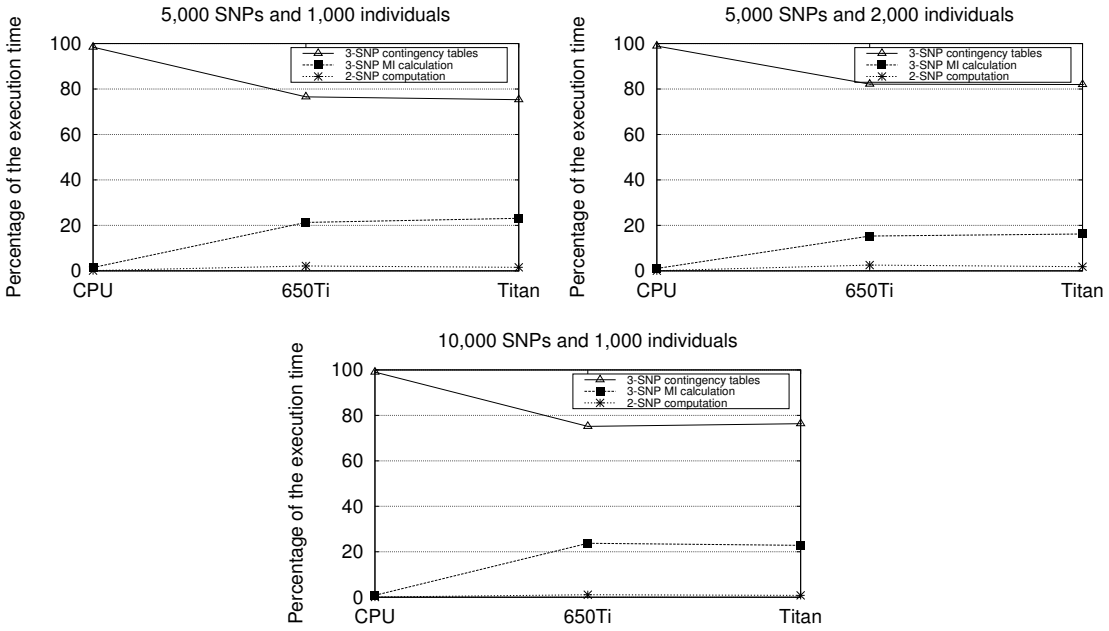

Figure 5: Runtime breakdown of GPU3SNP and the multithreaded CPU approach.

Table 8: Runtime and speedup comparison of the static and dynamic distributions when analyzing a dataset with 5,000 SNPs and 1,000 individuals on a heterogeneous platform with one NVIDIA GTX 650Ti and one NVIDIA GTX Titan.

Distribution	Static	Dynamic
Runtime (m)	7.96	3.21
Speedup over CPU	44.80	111.09
Speedup over 650Ti	2.01	4.98
Speedup over Titan	0.50	1.24

slowest GPU (650Ti) analyzes less SNP-triples than the most powerful one (Titan), leads to 24% of performance improvement compared to the fastest single-GPU execution. It proves that collaboration among GPUs is beneficial even on platforms with heterogeneous GPUs. However, for these scenarios the static approach does not provide a performance improvement. The reason is that the workload is equally distributed among both GPUs, and the computation of half of the analyses on the slowest GPU leads to a bottleneck.

A single node of the MOGON supercomputer containing 4 NVIDIA GTX Titans, installed at the Johannes Gutenberg University of Mainz, is employed to evaluate the scalability of GPU3SNP for homogeneous GPUs. Figure 6 shows the execution time and speedups (compared to the single-GPU version) for both the static and dynamic approaches when analyzing again the dataset with 5,000 SNPs and 1,000 individuals on a varying number of GPUs. Note that the single-GPU runtime is higher than for the Titan of the heterogeneous system, as the characteristics of the host are different. The results show that static distribution is better for this type of system, with parallel efficiency over 98%. For homogeneous GPUs the on-demand version tends to equally distribute the workload. Thus, the workload distribution is the same for both approaches but the CPU computation before submitting each batch to the GPU is more complex in the dynamic approach. Nevertheless, as we have optimized the batch generation as much as possible, the difference between the two approaches is not significant (between 1% and 4%) for just four GPUs. However, it is expected to increase for a larger number of GPUs.

Table 9 compares the runtimes for the dataset with 10,000 SNPs and 1,000 individuals of GPU3SNP on the two platforms and two publicly available non-exhaustive tools: SNPRuler [10] and EDCF [8]. EDCF has been

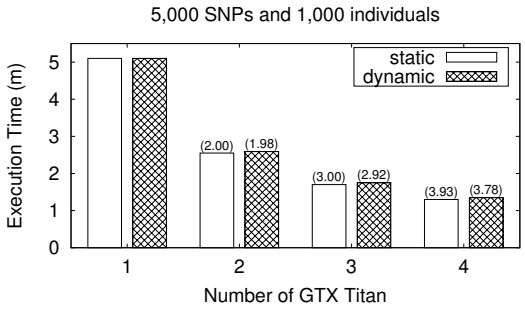

Figure 6: Runtime of the static and dynamic multi-GPU approaches for a varying number of NVIDIA GTX Titan. In parenthesis, speedups over the single-GPU version.

briefly described in Section 4.1. SNPRuler follows a machine learning approach to apply the χ^2 statistic only to the triples that pass a rule searching algorithm. We have excluded comparisons to other exhaustive tools presented in Section 1 since they are extremely slow and would require days or weeks to finish. As expected, thanks to their non-exhaustive search methods, EDCF and SNPRuler are faster than GPU3SNP. Nevertheless, while exhaustive analyses on CPU would need prohibitive runtimes, our multi-GPU parallelization is able to increase accuracy compared to these tools (see Section 4.1) with only a reasonable increase of time. In fact, we have been able to exhaustively analyze a dataset with 50,000 SNPs and 1,000 individuals in less than one day (21 hours and 52 minutes) on a single MOGON node.

Finally, Table 10 compares the speed of GPU3SNP and other GPU tools. As parallel tools for third-order analysis do not exist, we have used the results provided in [21] for two of the fastest pairwise interactions tools: EpistSearch and GBOOST. In order to provide a normalized comparison, speed is indicated as millions of contingency table cells per second. Furthermore, all experiments are performed with datasets consisting of 5,000 individuals on the same GPUs (Titan and 650Ti). Results show that GPU3SNP achieves high performance which indicates that it exploits more efficiently the GPU resources than GBOOST and EpistSearch.

Table 9: Runtime comparison of different tools when analyzing a dataset with 10,000 SNPs and 1,000 samples.

Tool	Exhaustive?	Architecture	Time
EDCF	No	Intel Core i7	44 s
SNPRuler	No	Intel Core i7	5 m
GPU3SNP	Yes	4 Titan	10 m
GPU3SNP	Yes	Titan & 650Ti	25 m

Table 10: Speed comparison of GPU3SNP and two GPU-based tools for pairwise studies. Speed is measured in millions of contingency table cells per second when analyzing 5,000 individuals.

Tool	Speed 650Ti	Speed Titan
GPU3SNP	365.85	1,432.63
EpistSearch	320.51	872.09
GBOOST	232.92	614.75

5. Conclusions

The search for third-order epistatic interactions plays a key role in order to find genetic explanations for several common human diseases. However, due to the high number of possible combinations, exhaustive analysis of all SNP-triples is not possible in a reasonable period of time on a CPU even for moderately-sized datasets with thousands of SNPs. Most current solutions follow a stepwise approach that reduces the search space. Nevertheless, these approaches can lead to a loss of accuracy. In this paper we have presented GPU3SNP, a GPU-accelerated exhaustive tool to search for third-order epistatic interactions. The main contributions of our work are:

- Development of the first open-source GPU-based tool that is able to analyze all the 3-SNP combinations. EDCF, one of the most powerful non-exhaustive publicly available tools, has been compared to GPU3SNP using three disease models with different minor allele frequencies and marginal effects. GPU3SNP is on average 34% more powerful than EDCF.

- Implementation of two highly efficient CUDA kernels that reuse auxiliary 2-SNP-AND values in all the SNP-triples that contain the pair. Memory accesses are optimized thanks to the exploitation of shared

memory. Furthermore, necessary accesses to device memory are fully coalesced.

- Development of two multiGPU approaches (static and dynamic) that exploit the capacities of systems with not only homogeneous but also heterogeneous GPUs.

The runtime of GPU3SNP has been tested on two different GPUs (GTX 650Ti and Titan), obtaining speedups over an efficient multithreaded implementation on a hex-core Intel Core i7 of up to 22.29 and 89.82, respectively. Moreover, thanks to the dynamic distribution, these two GPUs can collaborate to further increase performance by 24%. On a system with 4 NVIDIA GTX Titans the speedup over the single-GPU version is 3.93 (98% of parallel efficiency). We have also demonstrated that GPU3SNP is able to analyze tens of thousands of SNPs in a reasonable time (for instance, it needs less than 22 hours for 50,000 SNPs on 4 NVIDIA GTX Titan), while a similar approach on a modern hex-core machine would need several months.

Due to its exhaustive nature, the runtime of GPU3SNP scales at a cubic rate in terms of the number of SNPs. Thus, the analysis of GWAS datasets containing hundreds of thousands of genetic markers would still require significant amount of time on a single MOGON node. To address this issue, we are planning the extension of GPU3SNP to run on a large number of GPU-accelerated compute nodes efficiently. This approach should enable the exhaustive third-order analysis of such datasets in reasonable time. As future work, our plan is to extend GPU3SNP to provide more measures apart from MI and IG. Therefore, the users will be able to choose the measure more suitable for the type of analyses that they need. In fact, as GPU3SNP is open-source, it can be used to analyze different datasets with several measures in order to study their accuracy for different scenarios.

References

[1] B. Maher, Personal Genomes: the Case of the Missing Heritability, Nature 456 (7218) (2008) 18–21.

[2] H. Cordell, Detecting Gene-Gene Interactions that Underlie Human Diseases, Nature Review Genetics 10 (6) (2009) 392–404.

[3] J. H. Moore, F. M. Asselbergs, S. M. Williams, Bioinformatics Challenges for Genome-Wide Association Studies, Bioinformatics 26 (4) (2010) 445–455.

[4] S. Leem, H. hwan Jeong, J. Lee, et al, Fast Detection of High-Order Epistatic Interactions in Genome-Wide Association Studies Using Information Theoretic Measure, Computational Biology and Chemistry 50 (2014) 19–28.

[5] Y. Zhang, J. S. Liu, Bayesian Inference of Epistatic Interactions in Case-Control Studies, Nature Genetics 39 (9) (2007) 1167–1173.

[6] W. Tang, X. Wu, R. Jiang, Y. Li, Epistatic Module Detection for Case-Control Studies: a Bayesian Model with a Gibbs Sampling Strategy, PLoS Genetics 5 (5) (2009).

[7] G. Fang, M. Haznadar, W. Wang, et al, High-Order SNP Combinations Associated with Complex Diseases: Efficient Discovery, Statistical Power and Functional Interactions, PLoS ONE 7 (4) (2012).

[8] M. Xie, J. Lie, T. Jiang, Detecting Genome-Wide Epistases Based on the Clustering of Relatively Frequent Items, Bioinformatics 28 (1) (2012) 5–12.

[9] X. Wan, C. Yang, Q. Yang, et al, MegaSNPHunter: a Learning Approach to Detect Disease Predisposition SNPs and High Level Interactions in Genome Wide Association Study, BMC Bioinformatics 10 (2009).

[10] X. Wan, C. Yang, Q. Yang, et al, Predictive Rule Inference for Epistatic Interaction Detection in Genome-Wide Association Studies, Bioinformatics 26 (1) (2010) 30–37.

[11] M. R. Nelson, S. L. Kardia, R. E. Farrel, C. F. Sing, A Combinatorial Partitioning Method to Identify Multilocus Genotypic Partitions that Predict Quantitative Trait Variation, Genome Research 11 (3) (2001) 458–470.

[12] R. Culverhouse, the Use of the Restricted Partition Method with Case-Control Data, Human Heredity 63 (2) (2007) 93–100.

[13] M. D. Ritchie, I. W. Hahn, N. Roodi, et al, Multifactor-Dimensionality Reduction reveals High-Order Interactions among Estrogen-Metabolism Genes in Sporadic Breast Cancer., American journal of Human Genetics 69 (1) (2001) 138–147.

[14] T. Cattaert, M. L. Calle, S. M. Dudek, et al, Model-Based Multifactor Dimensionality Reduction for Detecting Epistasis in Case-Control Data in the Presence of Noise, Annals of Human Genetics 75 (1) (2011) 78–89.

[15] J. Gui, A. S. Andrew, P. Andrews, et al, A Robust Multifactor

Dimensionality Reduction Method for Detecting Gene–Gene Interactions with Application to the Genetic Analysis of Bladder Cancer Susceptibility, Annals of Human Genetics 75 (1) (2011) 20–28.

[16] X. Hu, Q. Liu, Z. Zhang, et al, SHEsisEpi, a GPU-Enhanced Genome-Wide SNP-SNP Interaction Scanning Algorithm, Efficiently Reveals the Risk Genetic Epistasis in Bipolar Disorder, Cell Research 20 (2010) 854–857.

[17] G. Hemani, A. Theocharidis, W. Wei, et al, EpiGPU: Exhaustive Pairwise Epistasis Scans Parallelized on Consumer Level Graphics Cards, Bioinformatics 27 (11) (2011) 1462–1465.

[18] L. S. Yung, C. Yang, X. Wan, W. Yu, GBOOST: a GPU-Based Tool for Detecting Gene-Gene Interactions in Genome-Wide Case Control Studies, Bioinformatics 27 (9) (2011) 1309–1310.

[19] J. Piriyapongsa, C. Ngamphiw, A. Intarapanich, et al, iLOCi: a SNP Interaction Priorization Technique for Detecting Epistasis in Genome-Wide Association Studies, BMC Genomics 13 (Supl 7) (2012).

[20] B. Goudey, D. Rawlinson, Q. Wang, et al, GWIS - Model-Free, Fast and Exhaustive Search for Epistatic Interactions in Case-Control GWAS, BMC Genomics 14 (Supl 3) (2012).

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

[22] L. Wienbrandt, J. C. Kässens, J. González-Domínguez, et al, FPGA-Based Acceleration of Detecting Statistical Epistasis in GWAS.

[23] D. Sluga, T. Curk, B. Zupan, U. Lotric, Heterogeneous Computing Architecture for Fast Detection of SNP-SNP Interactions, BMC Bioinformatics 15 (216) (2014).

[24] L. Ma, H. B. Runesha, D. Dvorkin, J. R. Garbe, Y. Da, Parallel and Serial Computing Tools for Testing Single-Locus and Epistatic SNP Effects of Quantitative Traits in Genome-Wide Association Studies, BMC Bioinformatics 9 (315) (2008).

[25] M. A. Steffens, T. A. Becker, T. Sander, et al, Feasible and Successful: Genome-Wide Interaction Analysis Involving All 1.9x1011 Pair-Wise Interaction Tests, Human Heredity 69 (4) (2010) 268–284.

[26] T. Schüpbach, I. Xenarios, S. Bergmann, K. Kapur, FastEpistasis: a High Performance Computing Solution for Quantitative Trait Epistasis, Bioinformatics 26 (11) (2010) 1468–1469.

[27] J. C. Kässens, J. González-Domínguez, L. Wienbrandt, S. B, UPC++ for Bioinformatics: A Case Study Using Genome-Wide Association Studies, in: Proc. 15th IEEE Intl. Conf. on Cluster Comp. (Cluster'14), 2014.

[28] X. Wan, C. Yang, Q. Yang, et al, BOOST: A Fast Approach to Detecting Gene-Gene Interactions in Genome-wide Case-Control Studies, American Journal of Human Genetics 87 (3) (2010) 325–340.

[29] T. Hu, Y. Chen, J. W. Kiralis, et al, An information-Gain Approach to Detecting Three-Way Epistatic Interactions in Genetic Association Studies, Journal of American Medical Information Association 20 (2013) 630–636.

[30] NVIDIA Developer CUDA Zone, `https://developer.nvidia.com/category/zone/cuda-zone` (Last visit: November 2014).

[31] Y. Wang, G. Liu, M. Feng, L. Wong, An Empirical Comparison of Several Recent Epistatic Interaction Detection Methods, Bioinformatics 27 (21) (2011) 2936–2943.

[32] genomeSIMLA Software, `http://chgr.mc.vanderbilt.edu/genomeSIMLA/` (Last visit: November 2014).

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

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