Accelerating 3-way Epistasis Detection with CPU+GPU processing[⋆]

Ricardo Nobre, Sergio Santander-Jiménez, Leonel Sousa, and Aleksandar Ilic

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
{ricardo.nobre,sergio.jimenez,leonel.sousa,aleksandar.ilic}@inesc-id.pt

Abstract. A Single Nucleotide Polymorphism (SNP) is a DNA variation occurring when a single nucleotide differs between individuals of a species. Some conditions can be explained with a single SNP. However, the combined effect of multiple SNPs, known as epistasis, allows to better correlate genotype with a number of complex traits. We propose a highly optimized GPU+CPU based approach for epistasis detection. The GPU portion of the approach relies only on CUDA cores to score sets of SNPs, based on the copresence of genetic variants and a specific outcome (case or control), making it suitable for a large number of computing devices. Considering datasets with different shapes (more SNPs than patients, or vice versa) and sizes, combining an analytical analysis and an experimental evaluation with five CPU+GPU configurations covering different GPU architectures from the last five years, we show that the performance achieved by our proposal is close to what is theoretically possible on the targeted GPUs. Comparing, in 3-way epistasis detection, with a state-of-the-art GPU-based approach which also does not rely on specialized hardware cores, MPI3SNP, the proposal is on average 3.83×, 2.72×, 2.44× and 2.71× faster on systems with a Titan X (Maxwell 2.0), a Titan XP (Pascal), a Titan V (Volta) and a GeForce 2070 SUPER (Turing) GPU, respectively.

Keywords: Epistasis detection, parellel processing, GPU, heterogeneous system.

1 Introduction

Throughout the years, Genome-Wide Association Studies (GWAS) have been promoting significant advances in genetics research by shedding light into the relationship between genetic variants and phenotypic traits. The analysis of Single Nucleotide Polymorphisms (SNPs) plays a prevailing role in this context, since they represent the most frequent type of variation in human genome. In

[⋆] This work was supported by the FCT (Fundação para a Ciência e a Tecnologia, Portugal) and the ERDF (European Regional Development Fund, EU) through the projects UIDB/50021/2020 and LISBOA-01-0145-FEDER-031901 (PTDC/CCI-COM/31901/2017, HiPErBio). Sergio Santander-Jiménez is supported by the Post-Doctoral Fellowship from FCT under Grant SFRH/BPD/119220/2016.

the pursuit of a better understanding of genotype-phenotype relationships, increasing research efforts have been focused on identifying the impact of multiple, interactive SNPs, a phenomenon known as epistasis [10]. In fact, the accurate identification of epistatic interactions represents a hot research topic, since a number of complex traits (such as Alzheimer's disease [19], breast cancer [13] and Crohn's disease [8]) are consequence of the joint effect of several SNPs at different interaction orders.

The epistasis detection not only introduces additional layers of biological complexity but it also represents a computationally intensive optimization problem. This challenging nature becomes even more noticeable when an exhaustive (as precise as possible) evaluation of interactive SNP combinations (i.e., triplets in 3-way detection) is needed to explain very complex traits. In order to address this issue, given the data-parallel nature of epistasis detection searches (same operations performed to evaluate each set of SNPs), there are a number of exhaustive strategies implementing GPU-based approaches [3, 5, 11, 16, 18]. However, for a number of reasons (e.g., differences in the architectures), some of those approaches might not be able to efficiently use recent hardware.

This work proposes a novel GPU+CPU approach for epistasis detection based on exhaustive examination of SNP combinations, which aims at fully exploiting the computational capabilities offered by the joint, collaborative action of CPU and GPU devices. With this purpose in mind, the proposed method relies on the architectural differences between CPU and GPU to undertake the efficient evaluation of epistatic interactions, accurately scheduling and distributing these tasks according to their properties and suitability to the underlying hardware. The main contributions of this paper are the following:

- High-Performance GPU+CPU based high-order epistasis detection approach based on optimized parallel algorithms;
- Analytical and experimental analysis of the proposal when performing 3-way detection on five CPU+GPU systems (comprising Maxwell 2.0, Pascal, Volta and Turing GPU architectures and Xeon, i7, and i9 CPU architectures);
- In-depth discussion of theoretical and achieved parallel performance on different dataset shapes and problem sizes, with validation of results through comparisons with other state-of-the-art strategies.

Experimental results show that the proposed approach is capable of outperforming MPI3SNP [11], a state-of-the-art approach, for about 2.9× (on average) across four different NVIDIA architectures. In addition, the proposal achieves near-theoretical maximum performance according to the characteristics of the algorithm.

This paper is structured as follows. Section 2 formulates the epistasis detection problem. Section 3 provides insight into the proposed method and the design strategies adopted to enhance epistasis searches on CPU+GPU systems. Section 4 presents the experimental campaign herein undertaken and contrasts the results achieved with the state-of-the-art. Section 5 situates the proposal in the related work on optimized algorithms for epistasis detection. Section 6 concludes the paper.

2 Problem Formulation

Certain phenotypic traits, e.g. diseases, are strongly correlated to the joint interaction of different SNPs in the genome of a particular individual (patient). The accurate identification of such interactions represents the main goal in epistasis detection. Epistasis analyses therefore deal with the processing of information on SNPs from different individuals, which are classified according to a binary outcome (e.g., having –case– or not having –control– the phenotypic trait under study). Such information is contained in a dataset D of size $N \times (M+1)$, where N is the number of individual samples and M the number of SNPs subject to analysis. Each entry $D[i,j], i \in \{1,...,N\}, j \in \{1,...,M\}$ provides the genotypic value observed at the j-th SNP for the i-th individual, encoded as 0 (homozygous major allele, AA), 1 (heterozygous allele, aA or Aa), or 2 (homozygous minor allele, aa). The entry $D[i, M+1]$ in a sample represents the phenotypic state observed for the i-th individual (0 in control samples and 1 in case samples).

Epistasis detection is usually formulated as an optimization problem that is aimed at identifying the combination of k-interactive SNPs $x = [x_1, x_2, ..., x_k]$ ($x_i \in \{1,...,M\}$) that most likely governs the occurrence of the examined trait, where k is the epistasis interaction order assumed for that trait. The identification of the optimal solution to the problem requires exploring an epistasis search space of ${}^{M}C_k = \frac{M!}{k!(M-k)!}$ sorted and non-repeated combinations of SNPs [12]. The suitability of each possible combination x is measured by means of objective functions, which measure the degree of impact of the evaluated interaction in the studied trait. In this work, the proposed approach adopts the widely-used Bayesian K2 score [2, 14] as the scoring objective function:

$$K2 = \sum_{i=1}^{I} \left(\sum_{b=1}^{r_i+1} \log(b) - \sum_{j=1}^{J} \sum_{d=1}^{r_{ij}} \log(d) \right), \tag{1}$$

where I is the number of possible genotypic combinations among k SNPs ($I = 3^k$, since each SNP can take one out of three possible genotypic values), J the number of phenotypic states ($J = 2$ in case-control scenarios), r_i the frequency of a certain genotypic combination i at the evaluated SNPs $x = [x_1, x_2, ..., x_k]$, and r_{ij} the number of samples that satisfy the occurrence of the phenotypic state j with the genotypic combination i at x. That is, given a k-order SNP interaction, the occurrences of each possible genotype combination for the examined k SNPs are counted through all the samples recorded in the dataset, taking into account each record type (case or control). This results in a total of 2×3^k frequency values. These values, forming what is known as a ***contingency*** or ***frequency*** table, are used to compute a score that aims to discriminate predictive strength, between the different sets of SNPs, in relation to the binary outcome under study. Lower K2 scores denote better solution quality, in such a way that the interaction that minimizes Equation 1 represents the best potential solution.

In terms of computational effort, the exhaustive identification of optimal solutions depends on the interaction order and the dimensions of the input data.

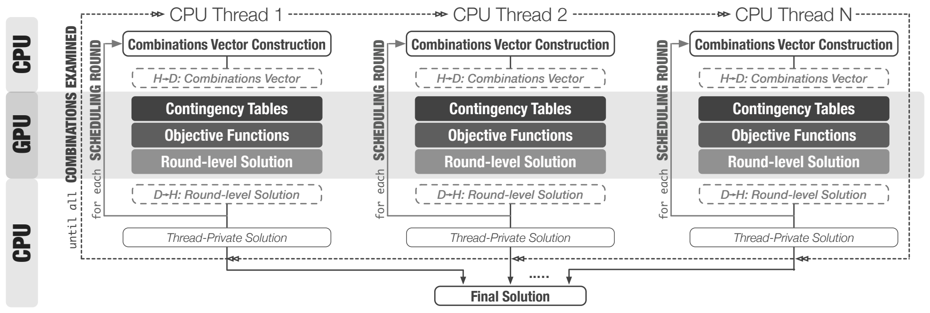


Fig. 1. Overview of scheduling between CPU and GPU.

Specifically, the number of possible candidate solutions to the problem exponentially grows in harder epistasis scenarios (increasing k and M), while the amount of calculations in the objective function depends on N. These issues demand efficient parallel strategies and orchestration of computing devices to tackle the problem, accurately exploiting the combined capabilities of CPU+GPU systems.

3 GPU+CPU Hybrid Epistasis Detection

The approach proposed in this paper is aimed at exploiting the collaborative execution among CPU and GPU devices to conduct computationally demanding epistasis detection tasks, particularly the exhaustive search for optimal 3-way SNP interactions according to the K2 score. Fig. 1 provides the general overview of the proposed CPU+GPU approach, which is designed in such a way that both device types cooperate in the process of determining the optimal solution by executing different sets of tasks. This task distribution is conducted due to the architectural differences between CPU and GPU, thus each device type is assigned with the tasks whose characteristics allow better exploitation of the computational and memory resources on a per device architecture basis.

In a nutshell, the host CPU cores are responsible for generating the sets of combinations of SNP indexes (*Combinations Vector*) to be processed in the GPU. The evaluation of SNP combinations is a task rich in data-parallelism, thus highly suitable for GPU execution, while the generation of combinations vectors requires a large amount of complex control and synchronization primitives, thus better fitting the CPU architecture. The combinations vectors (chunks) are simultaneously constructed within several CPU threads and offloaded to the GPU in a streaming fashion. This process is repeated in several scheduling rounds until all possible SNP combinations have been examined.

Upon reception of a combinations vector, the GPU performs the following set of tasks: 1) *Contingency table construction*, i.e., counting the frequencies of combined genotypes for each of the unique sets of SNPs (resulting from combining SNPs all-to-all) generated by a particular CPU thread in a given round, 2) *Objective function (K2 score) calculation*, i.e., scoring each set of SNPs based on the constructed contingency table; and 3) determination of the local optimal solution (the one with the minimum K2 score) from the set of tuples of SNP indexes in the combinations vector, i.e., *Round-level Solution*.

For each scheduling round, the host CPU threads maintain the information regarding the best solution obtained across all executed rounds on the GPU (and initiated by that thread). Upon all possible SNP combinations are processed, the final/global optimum solution is determined by the CPU master thread by gathering the partial solutions from each host CPU thread.

3.1 CPU as orchestration engine and generator of unique sets

Exhaustive epistasis searches in 3-way fashion require evaluating all-to-all combinations three SNPs at a time. There are algorithms specialized in the generation of a combination of elements from a given index (e.g., [1]). The use of these algorithms allows arriving at a given combination faster than exhaustively and iteratively enumerating all combinations up to that combination. This is especially the case when those combination indexes are large.

Direct calculation of the combination (i.e., the triplet of indexes of SNPs) to process inside each GPU thread, from the GPU global thread identifiers, introduces a considerable overhead as it does not suit the parallel architecture of GPUs. Generation of the triplets of indexes of SNPs inside the GPU threads can result in an overhead that can only be offset when processing datasets with a large number of patient records. Notice that, given the nature of epistasis detection, we only want to process unique sets of SNPs. If what was required was to process all permutations, it would not be challenging to achieve direct mapping of thread identifiers into the triplets to process inside the GPU threads.

In the proposal presented in this paper, it is the host (i.e., the CPU) that generates combinations of indexes of SNPs, to be sent to the GPU for evaluation. Generation of combination vectors and communication of these vectors to the GPU for evaluation are performed in multiple rounds. The number of rounds is given by $\left\lceil \frac{^{M}C_3}{s} \right\rceil$, where $^{M}C_3$ is the total number of combinations to evaluate in a particular epistasis detection 3-way search (SNPs combined all-to-all in sets of three) and s is the number of combinations processed per round. Herein we refer to the later as *chunk size*.

Multiple rounds are processed concurrently by multiple CPU threads (a parameter). This results in more efficient use of the CPU resources to generate sets while also improving bandwidth utilization between CPU and GPU devices. While some CPU threads are generating combination vectors or waiting for GPU kernel executions to complete, other threads are sending new combination vectors to the GPU. Each CPU thread, when starting a new round, generates the first combination of three SNPs (i.e., three indexes) from a given starting combination index (assigned to that round). The calculation of the first combination for a given round is accomplished through an adaptation of the algorithm presented in [1].

Algorithm 1 depicts the operations the proposal performs for determining the configuration of the l_{th} combination (starting from 1), represented in output vector c, considering combinations of three SNPs chosen from m SNPs numbered from 0 to $m-1$.

```
Data: $m$, $l$
Result: $c$
$r = l$;
$c[0] = -1$;
for $i = 0;\ i < 2;\ i = i + 1$ do
    while $r > 0$ do
        $c[i] = c[i] + 1$;
        $d = comb(m - (c[i] + 1), 2 - i)$;
        $r = r - d$;
    end
    $r = r + d$;
    $c[i + 1] = c[i]$;
end
$c[2] = c[2] + r$;
```

Algorithm 1: Generation of the l_{th} combination of three elements chosen from m elements.

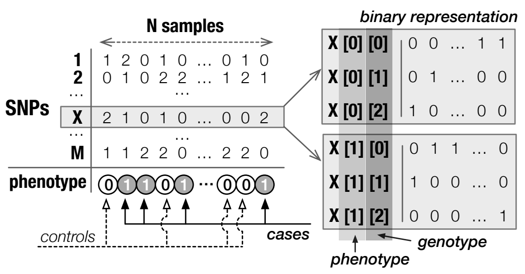
The *for* loop executes for two iterations ($i = 0$ and $i = 1$), determining the value for the first and the second positions — $c[0]$ and $c[1]$ — in the combination configuration vector c. The value for the third (and last) position — $c[2]$ — is equal to the value found for the previous position (the second) — $c[1]$ — plus a remainder (rest) stored in the r variable. In each of the two *for* loop iterations, the nested *while* loop executes until $r \leq 0$, which signals that the SNP index value for $c[i]$ ($i = 0$ or $i = 1$) has been found. Variable r, decremented by $comb(m - (c[i] + 1), 2 - i)$ — number of combinations that exist between $\{c[0], c[0] + 1, c[0] + 2\}$ and $\{c[0] + 1, c[0] + 2, c[0] + 3\}$ ($i = 0$) or between $\{c[0], c[1], c[1] + 1\}$ and $\{c[0], c[1] + 1, c[1] + 2\}$ ($i = 1$) — for any increment to $c[i]$, holds the distance to the l_{th} combination. When the *while* loop condition is evaluated to *false*, the previous value of r (the remainder) is restored — $r = r+d$ — and the starting value of $c[i + 1]$, the next combination configuration index, is set $c[i]$, the value found in the current *for* loop iteration. In order to avoid having to calculate (and recalculate) ${}^{j}C_2$ during execution of a given epistasis detection search, the proposal relies on a lookup table with values of ${}^{j}C_2$ for up to a large j (maximum number of SNPs). Thus, in the first *for* loop iteration ($i = 0$), the term $comb(m - (c[i] + 1), 2 - i)$ maps to precomputed values that are loaded to main memory at the application start. In the second (and final) *for* loop iteration ($i = 1$), there is no need to resort to precomputed values, as the number of combinations of one element chosen from $m - (c[i] + 1)$ elements is equal to the latter.

After the configuration of the first combination to be processed in a given round is known, all other combinations to be processed in that round are sequentially enumerated (assuming lexicographical order) by the CPU thread operating the round. For example, the next combination after $\{32, 41, 1854\}$ is $\{32, 41, 1855\}$ if the number of SNPs to select from is larger than 1855 (the indexes of the SNPs start at 0), and $\{32, 42, 43\}$ otherwise. This offsets the initial cost of determining the configuration of the first combination to be part of the combinations vector to be generated on a given round, as generating the next combination from a given other combination is considerably less computationally expensive.

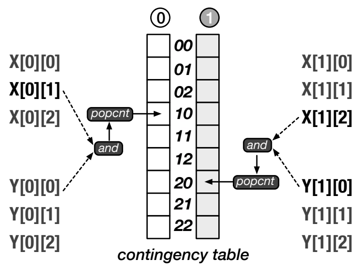
While combination vectors are being processed on GPU kernel executions launched by some CPU threads, additional combination vectors are concurrently being generated on the CPU by other threads, and combination vectors pertaining to other rounds are being transferred to the GPU (i.e., host to device memory transfer). This behavior is achieved using OpenMP, with a parallel *for* loop (`#pragma omp parallel for schedule(dynamic)`) iterating over the total number of sets to process (i.e., ${}^{M}C_{3}$) with a step equal to the number of combinations to be evaluated on a given round (i.e., the chunk size). These chunks must be large enough (a parameter of the proposal) to saturate the GPU compute units, while at the same time not large to the point of severely reducing the number of threads that can execute concurrently on the host. Each iteration of the parallel OpenMP loop, executed by a given CPU thread (each with its own private CUDA stream) is responsible for generating a chunk of triplets of indexes representing combinations of SNPs (i.e., the combinations vector), sending them to the GPU and launching the GPU kernel that evaluates the combinations. After that particular kernel execution completes, the CPU thread transfers back (i.e., device to host memory transfer) the index identifying the locally optimal combination found on the GPU kernel execution and corresponding score. Each CPU thread keeps track of the best solution found up to any given point, with the final reduction done on the master thread after the parallel *for* loop completes execution, i.e., once all combinations of SNPs are exhausted.

3.2 GPU as a combinations processor

Due to the challenging nature of the epistasis detection problem tackled herein, the way the input dataset is represented and processed represents a crucial aspect in ensuring its high performance execution, especially on GPU devices. In order to efficiently encode the input data set, we rely on binarization techniques, first used in the context of epistasis detection in [15]. In the proposed approach, the GPU receives as input two arrays, representing the controls and cases in the dataset (phenotype), respectively. As it is depicted in Fig. 2(a), all genotypic information pertaining a given SNP X in relation to the samples in the dataset is represented by $3 \times (N_0 + N_1)$ binary values, where N_0 and N_1 represent the number of controls and the number of cases, respectively. Each of these binary values is stored in the array representing controls (phenotype 0) or in the array representing cases (phenotype 1), respectively. For each pair composed of a sample and a SNP, one of three particular bits is set to 1 (in the cases or the controls array), identifying which genotype (0, 1 or 2) the sample has regarding



(a) Binary representation of input dataset



(b) Contingency table construction from binary encoding

Fig. 2. Representation of basic notions in epistasis detection.

that SNP. In total, these two arrays represent $M \times 3 \times N$ binary values, where M is the number of SNPs and N is the total number of samples ($N_0 + N_1$).

As previously referred, the GPU receives an array holding triplets of indexes of the SNPs to process in a specific kernel execution (i.e., the combinations vector), one triplet to be processed per GPU thread. When processing a single SNP combination, the binary representation of input data allows calculating the frequency of any given combined genotype (out of the 27 possible genotypic combinations) using only bitwise AND and POPC (population count, which counts the number of bits set to 1) instructions, and accumulating until all bit-packs concerning cases or controls (for a given genotype) have been processed.

Cases and controls are represented in distinct bit arrays and thus processed from separate bit-packs of data. Both AND and POPC instructions process 32-bit of data in all GPUs that are part of the systems targeted in this paper. Thus, for interaction order k (i.e., k-way epistasis detection searches), the total number of POPC instructions is equal to ${}^{M}C_k \times 3^k \times (\lceil \frac{N_0}{32} \rceil + \lceil \frac{N_1}{32} \rceil)$, while the total number of AND instructions is $(k-1)$ times higher (M being the number of SNPs, N_0 the number of controls and N_1 the number of cases). In particular, for 2-way epistasis detection, depicted in Fig. 2(b), an equal amount of AND and POPC instructions is required. However, for 3-way detection (considered in the work proposed herein), the number of AND instructions to be performed is 2 times higher than the amount of POPC instructions. This is due to the fact that, in order to calculate a single entry in the contingency table, two AND operations are needed to be performed over three binary encoded SNP inputs, before a single POPC operation is applied.

To facilitate the calculation of the objective function, the main GPU kernel also receives as input an array with precalculated values, specific to the particular function used (logarithms of factorials for the K2 Bayesian score [2, 14]), for up to the maximum that can possibly be needed to calculate scores given the number of cases and controls in the dataset, the number of SNPs, and the total number of sets resulting from the number of SNPs and the interaction order. Some of these values could be calculated inside each particular GPU thread, but it makes

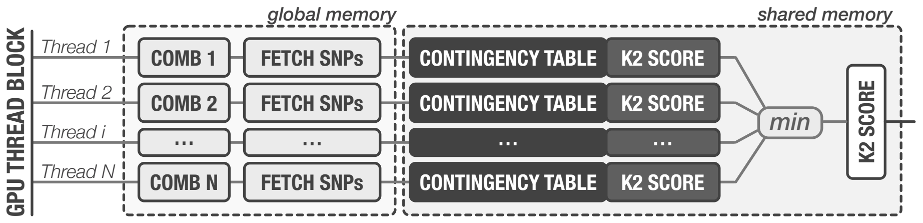


Fig. 3. Round of combinations being processed on a given GPU thread block.

for a more efficient solution to pass values that are to be the same in all GPU threads as input to the kernel.

The main GPU kernel is launched with a grid composed of as many thread blocks as the number of sets of SNPs to be processed by it (the chunk size) divided by the number of threads per thread block. The global identifier of each GPU thread pertaining to a given kernel execution (`blockDim.x * blockIdx.x + threadIdx.x`) identifies the combination of SNPs to be processed from the sets of SNPs represented in the combinations vector sent by the CPU in a given round. Thus, it is mapped to the positions in global memory holding the indexes representing the three SNPs to be evaluated by a given GPU thread.

Each GPU thread in a thread block processes a set of SNPs, whose indexes are identified in the combinations vector previously transferred from the CPU for that particular round, at positions $s \times i + global\_id$, where s is the number of combinations of SNPs to be processed per round and i is the particular SNP index out of the three involved. Reading the triplet of indexes identifying the SNPs to evaluate from the combinations vector from global memory in each GPU thread does not significantly increase the execution time of the GPU kernel. These indexes are read in a coalesced access pattern, at a very small cost in relation to the overall execution of the GPU kernel.

Fig. 3 illustrates how combinations are processed within a single GPU thread block, including which parts in the memory subsystems are used at each step. Evaluating a combination of SNPs entails taking into account genotypic data from all cases and controls regarding that set of SNPs. Each thread in a thread block iterates over all cases and then over all controls, reading packed binarized genotype patient data for a particular triplet of SNPs. The read operations to fetch patient data from global memory are in general efficiently performed. Within each array, SNP genotype values are binary encoded as follows. Each bitpack n of w binary encoded patient data records (i.e., representing w patients) concerning one of the three possible genotype configurations (0, 1 or 2), represented as g, of an SNP of index m is stored in position $g \times M \times \lceil \frac{N}{w} \rceil + n \times M + m$ in global memory. Given that the number of SNPs in the dataset is expected to be significantly larger than the size of a warp, and that GPU threads with contiguous *thread id* access lexicographically contiguous combinations, GPU threads in a given thread warp are likely to access data concerning the third SNP in the combination being processed from contiguous positions in memory, i.e., in

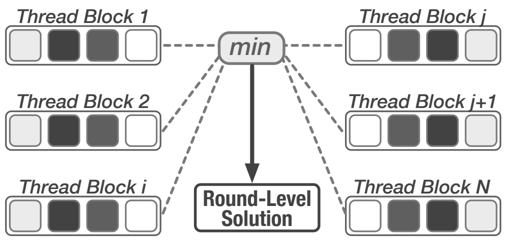


Fig. 4. Reduction of scores between different GPU thread blocks.

a coalesced access pattern. For instance, the combination after $\{23, 354, 661\}$ is likely $\{23, 354, 662\}$. In addition, the global memory reads to bit-packs of binarized data concerning the first and the second SNPs from the combination being evaluated in each GPU thread will typically access the same memory positions as other threads from the same warp.

Iterating over individuals, cases or controls, the three vectors of data per SNP and per genotype (a total of nine vectors of binarized patient data) in a given GPU thread are combined to determine the number of occurrences of each genotype combination (27 possible genotype combinations) in the samples represented by the bit-packs of binarized data. This is accomplished with bitwise AND and POPC instructions. The proposal relies on `__popcll()`, which is the 64-bit equivalent of the `__popc()` 32-bit CUDA intrinsic. Both map to the POPC 32-bit instruction, but processing bit-packs of 64-bit of data at a time allowed to achieve a slightly higher overall performance. Iteratively processing 64-bit of binarized patient data instead of 32-bit allows a better use of the GPU memory subsystem, by reducing the amount of write accesses needed for incrementing the frequency counters during construction of the contingency table to half. Genotype frequencies are accumulated for cases and controls (processed separately) until all bit-packs of patient data concerning a given triplet of SNPs have been processed, at which point the construction of the contingency tables for all GPU threads in a warp is complete.

The GPU threads in a given thread block store the counts of observed combined genotype frequencies (for cases and controls) in an array of size $2 \times 27 \times T$ in shared memory, where T represents the number of GPU threads in a thread block. All write memory accesses during the generation of the contingency tables are coalesced. Each frequency counter for a given combined genotype g (from 0 to 26) and phenotype p (0 for controls and 1 for cases), pertaining to a GPU thread with a given *local id*, is mapped to position $2 \times g \times T + p \times T + local\_id$ of the array in shared memory. In any given thread block, the specific shared memory 32-bit slot used in a GPU thread is indexed by its local identifier (`threadIdx.x`) in the thread block. Thus, there is no possibility of collision with other threads in the same warp (i.e., multiple threads accessing the same memory slot). These optimizations maximize the utilization of shared memory, thus benefiting from its low latency and high bandwidth nature (when compared to the global memory).

After all data (for all cases and controls) regarding the set of SNPs assigned to a given GPU thread has been processed, a score is calculated based on the counts for the 27 combined genotype frequencies (for cases and controls) stored, per GPU thread, in shared memory. Afterwards, the score is reduced in shared memory across the GPU threads in the same thread block, and finally, in global memory across all thread blocks, as illustrated in Fig. 4, in order to determine the best overall objective score (minimum in the case of the K2 score) for a given round. The *global id* of the thread achieving local minimum and the corresponding score are transferred to the CPU thread responsible for that particular GPU kernel execution and stored, after mapping the *global id* to a combination index (adding the index of the first combination processed in the round), if the score is better than the best score found in the previous rounds in that CPU thread.

Particularities of the exhaustive epistasis detection algorithm used and knowledge about the type of data being processed are leveraged to attain high performance. For instance, the proposal improves performance of the objective function by using a lookup table for storing precomputed values that are to be used by the scoring function. Note that this makes the performance of the proposal, in regards to the evaluation of the sets of SNPs from the genotype frequency counts (i.e., the contingency table values for cases and controls), to a great extent independent of the particular scoring function used. The only requirement is that the range of precomputed values includes all the values that might be needed when computing the scores, which depends both on the scoring function and the number of cases and controls, and that access to those values is fast. The later aspect is assured by the use of the read-only GPU data cache load function (i.e., `__ldg()`). In the case of the K2 Bayesian score, the logarithms of factorials calculated by this scoring function are precomputed on the CPU and sent to the GPU (once for the whole execution). Thus, assuring that no actual expensive computation of logarithms of factorials needs to be performed on the GPU. Even compared with the use of the `lgamma()` CUDA math intrinsic, which allows to calculate the factorials efficiently through the use of the gamma function ($\Gamma(x) = (x-1)!$), using the lookup tables allows achieving highest performance.

3.3 CPU+GPU execution orchestration

In order to get the maximum performance out of a given CPU+GPU system, the proposal relies on several CPU threads executing concurrently. Thus, overlapping the generation of combination vectors on the CPU with computation on the GPU and with data transfers between CPU/GPU at any given time. This results in making efficient use of the available bandwidth between CPU/GPU, all available CPU cores and GPU compute resources. This kind of orchestration is suitable even in the presence of multi-GPU systems. In such systems, GPU devices on the same machine are assigned to CUDA streams private to different CPU threads. The way the proposal divides combinations of SNPs to be evaluated into smaller work units improves efficiency of work distribution between GPUs on the same system. Depending on the capability of a given GPU, compared with other GPUs in the system, more/less rounds are assigned during execution to the CPU threads responsible for orchestrating work for that GPU.

The overall wall-clock time required to execute the proposal is dominated by execution on the GPU. Generation of combinations on the CPU might only be a performance deterrent in unbalanced systems (i.e., high-end GPU or multi-GPU paired with weak CPU). The number of threads is by default set to be equal to the number of CPU cores identified at runtime to exist in the system. This assures that there are enough concurrently running CPU threads to send the GPU new combination vectors to process at a pace that allows CUDA cores to be utilized close to their potential.

A suitable chunk size, i.e., number of sets to process per kernel execution, is important to achieve high performance. Chunk size determines the time taken by CPU threads to generate combinations and the time required to evaluate them on the GPU. The range of values that best suit the execution of the proposal is expected to depend on the dataset (number of SNPs and patient records) and on the given target CPU+GPU configuration. A value that is too low implies that the GPU compute resources will be severely underutilized and/or result in more time spent on preparation and launching the kernel in relation to the kernel execution. A value that is too large can also result in performance degradation, impacting negatively the load balancing between the CPU threads. This is especially the case in scenarios where the input dataset has a small number of SNPs, and thus the resulting number of total sets to process is also small. The latter scenario can result, in the worst case, in the complete serialization of the work, being all SNP combinations processed in a single round.

Data transfer between CPU and GPU devices is mostly in the direction from the CPU to the GPU. Data transfer from the GPU to the CPU is only performed at the end of the execution of the main GPU kernel in a given CPU thread, i.e., an index identifying the SNP set evaluated as having the best score and the score itself. Compared with the wall-clock time required by the GPU to evaluate combinations of SNPs, the time required for sending the combination vectors is not expected to limit overall performance on well balanced systems. This is especially the case on systems with CPUs with a large number of cores.

4 Experimental results

Our experimental campaign is aimed at thoroughly evaluating the impact of different numbers of SNPs and individual samples on the throughput of the proposal for systems with different CPU+GPU configurations. We performed experiments on datasets with different shapes (more SNPs than patient records, or vice versa) and sizes in order to cover different use cases. In addition, we evaluate the combined performance impact of the number of concurrently executing CPU threads and the number of combinations processed per scheduling round. Finally, the performance of the proposal is compared with MPI3SNP [11], a GPU-based state-of-the-art approach for 3-way epistasis detection.

4.1 Experimental setup and datasets

We rely on five different systems for our experiments. The systems are comprised of a Xeon E3-1245 V3 and a GeForce 2070 SUPER (designated as S1), a Core i9-7900X and a Titan V (S2), a i7-4770K and a Titan Xp (S3), a i7-5960X and a Titan X (S4) or a i7-6700K and two GeForce 980 GPUs (S5). Table 1 shows the specifications of the systems, including GPU, CPU, DRAM, operating system, CUDA and GPU driver versions. The systems are ordered based on GPU architecture, from the most recent (Turing) to the one released earliest (Maxwell 2.0).

Table 1. Overview of systems used in the experiments.

Systems	**GPU (NVIDIA)** arch. | cuda | driver	**CPU (Intel)** #cores | freq.	**DRAM** #channels | freq.	**Operating System**
S1	**GeForce 2070S** Turing | 10.1 | 430.40	**Xeon E3-1245 V3** 4 | 3.6GHz	16GB DDR3 dual | 2400MHz	Ubuntu 18.04
S2	**Titan V** Volta | 9.2 | 396.54	**Core i9-7900X** 10 | 4.0GHz	64GB DDR4 quad | 2400MHz	CentOS 7.5
S3	**Titan XP** Pascal | 10.1 | 418.56	**Core i7-4770K** 4 | 3.5GHz	32GB DDR3 dual | 1333MHz	CentOS 7.6
S4	**Titan X** Maxwell 2.0 | 8.0 | 375.26	**Core i7-5960X** 8 | 3.3GHz	32GB DDR4 quad | 2133MHz	Fedora 21
S5	2×**GeForce 980** Maxwell 2.0 | 8.0 | 410.48	**Core i7-6700K** 4 | 4.2GHz	32GB DDR4 dual | 2133MHz	CentOS 7.3

In Volta (compute capability 7.0) and in Turing (compute capability 7.5), each Streaming Multiprocessor (SM), the most basic building block of NVIDIA GPUs, has four processing blocks. Each processing block has one instruction scheduling and dispatch unit, 64 KBytes of register file space, 16 IEE754 32-bit floating point scalar Arithmetic Logic Units (ALUs) (i.e., Floating-Point Units (FPUs)), 16 32-bit integer scalar ALUs, four load/store units and four Special Function Units (SFUs). In addition, each SM has 96 KBytes (Turing) or 128 KBytes (Volta) of L1 configurable cache/shared memory, shared between the four processing blocks. In comparison, in Maxwell 2.0 (compute capability 5.2) and Pascal (compute capability 6.1), each processing block has double the amount of ALUs, FPUs, SFUs and load/store units. All four GPU architectures use a quadrant-based design, therefore the amount of units per SM in Maxwell 2.0 and Pascal is double the amount in Volta and Turing. There are 96 KB of shared memory per SM on the two former architectures.

The GPU configurations considered have each a total of 2560 (GeForce 2070S), 5120 (Titan V), 3840 (Titan Xp), 3072 (Titan X) and 4096 (2 × GeForce 980) CUDA cores. Boost frequencies are 1770, 1455, 1582, 1089 and 1216 MHz, respectively. An important factor in the execution of the proposal is given by the throughputs of AND and POPC instructions. The considered GPU configurations are capable of executing 4.5 (GeForce 2070S) and 7.4 (Titan V), 6.1 (Titan Xp), 3.3 (Titan X) and 5 (2× GeForce 980) tera 32-bit bitwise AND instructions per second at their respective boost clocks. The rate of 32-bit POPC instructions per second is a quarter that of AND instructions for all GPUs.

In order to fully exercise the capabilities of the proposal, taking into account that real datasets can vary substantialy in regard to size and shape, we have examined synthetic datasets with different numbers of SNPs and patient samples. Notice that that the specific values pertaining SNP data of cases or controls in the dataset do not affect the efficiency of the algorithm, thus the achieved performance is representative of what would be achieved with real datasets. A total of 16 datasets were generated, each having a combination of 1000, 2000, 4000 or 8000 SNPs with 1000, 2000, 4000 or 8000 patient records (half cases/controls). Considering 3-way epistasis detection scenarios, the numbers of unique sets of SNPs to be evaluated per problem instance are the following: 166167000 (1000 SNPs), 1331334000 (2000 SNPs) and 10658668000 (4000 SNPs) and 85301336000 (8000 SNPs). Each doubling of the number of SNPs results in a growth of about 8× regarding the total number of combinations of SNPs to evaluate. We rely on the same datasets in the experiments performed to examine the effect in performance of the number of sets to process per scheduling round, setting it to 10000, 20000, 40000, 80000, 160000 or 320000, combined with using 1, 2, 4 or 8 concurrently executing CPU threads. Finally, for the experiments comparing the proposal with MPI3SNP we rely on the two datasets available on the MPI3SNP source code repository[1]. These datasets represent 10000 SNPs from 1600 patients and 40000 SNPs from 6400 patients, half cases and half controls. The number of SNPs in these datasets results in 166616670000 and 10665866680000 combinations of SNPs to evaluate, respectively.

4.2 Performance analysis across different datasets and platforms

Fig. 5 depicts the performance achieved by the proposal on the different systems employed in the experimentation. Overall, performance tends to increase with the number of patient records. This is especially the case when going from a dataset with 1000 patients to a dataset with 2000 patients. Certain costs associated with execution of the GPU kernel implemented in the proposal are independent from the number of samples in the input dataset. In addition to the cost of initializing the GPU kernel execution on the host in a given round, there are other operations with cost invariant to the number of patient records, such as the calculation of the score on top of the contingency table and the score reduction operations, first between GPU threads in shared memory and then between thread blocks in global memory. After a certain number of patients, which depends on the system targeted, the potential for exploitation of resources gets saturated. Thus, considering more patients makes the wall-clock time increase close to linearly, resulting in similar performance levels. The number of SNPs also impacts performance, which increases considerably from 1000 to 2000 SNPs.

[1] https://github.com/chponte/mpi3snp/wiki/Sample-files

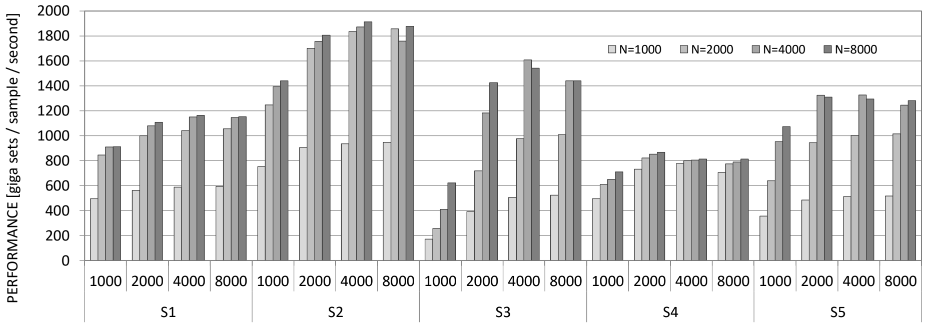


Fig. 5. Number of giga ($\times 10^9$) sets processed per second normalized to the number of patient samples (N) for the proposal on five different systems (S1, S2, S3, S4 and S5) and considering different numbers of SNPs (represented at the bottom) and patients (the different chart bars).

Overall, the attained results suggest that the main performance limiting factor, provided that the input dataset saturates the implementation of the proposal (i.e., sufficient ammount of SNPs and patient records), is the throughput of the CUDA cores, and not that of memory bandwidth. Performing the population counts, which uses the POPC instruction, is the operation that imposes the main restrictions on the maximum performance achievable in this problem for the considered GPU architectures. POPC is more challenging than bitwise AND as, although two instructions of the latter type have to be executed per a single instruction of the former type, POPC instructions execute at a quarter the rate of AND instructions. Even when considering a simplified performance model where only the execution of the POPC instruction is taken into account, it can be observed from the achieved results that the performance of the proposal on all GPUs is close to their compute capabilities. Using such simplified model, one can estimate the performance for evaluation of a given dataset from the total number of CUDA cores and boost frequency of each GPU configuration. The different GPU configurations are capable of executing POPC instructions at different rates (see subsection above). Given that each POPC instruction processes 32 patient records for a given genotype out of the 27 allowed genotypes (for a given triplet of SNPs), each GPU configuration has the potential to achieve 1.185$\times$ (from $\frac{32}{27}$) the sets of SNPs per patient record processed per second in relation to the rate of POPC instructions executed per second. Therefore, the maximum performance at boost clocks considering only the POPC instruction would be 1343 (GeForce 2070S), 2207 (Titan V), 1800 (Titan Xp), 991 (Titan X) and 1476 (2$\times$ GeForce 980) giga ($\times 10^9$) triplets of SNPs per sample processed per second. For the largest dataset (8000 SNPs and 8000 patients), the proposal achieved 86, 85, 80, 82 and 87 percent of those estimated maximum values, respectively. One can infer from this analysis, which in fact is not even taking into account the cost of executing the AND instructions, that the implementation of the proposal is able to efficiently use the targeted GPUs.

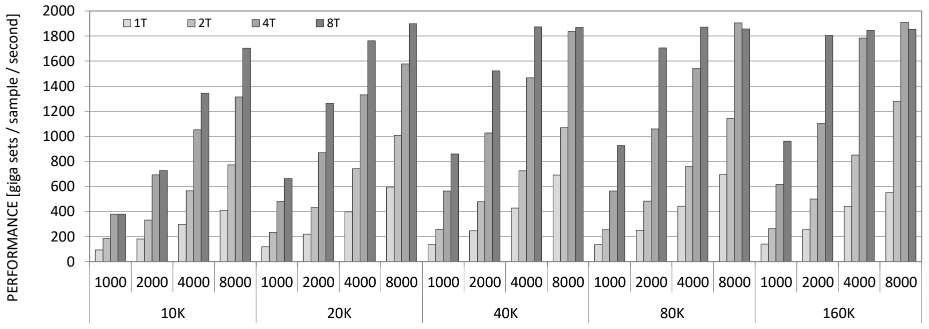


Fig. 6. Effect of the number of combinations to process per round (between 10000 and 160000) and the number of CPU threads (between 1 and 8) on the performance achieved by the proposal on system S2 (Titan V and i9-7900X) considering 4000 SNPs and different numbers of patients (between 1000 and 8000).

Overall, the system with the Titan V (S2) is the fastest at executing the proposal. For the largest dataset, the system S2 is 1.63×, 1.30×, 2.31× and 1.12 faster than systems S1 (GeForce 2070S), S3 (Titan Xp), S4 (Titan X) and S5 (2× GeForce 980), respectively. The experimentally achieved performance ratios, calculated from comparing performance on all other systems with system S2 (Titan V), are very close to the values that can be predicted based on GPU compute resources and clock frequency. This points to the fact that the execution of the proposal is not being limited by CPU performance.

4.3 Performance impact of the number of combinations per round and CPU threads

The number of concurrently executing CPU threads and the number of combinations of SNPs processed per round can have a significant impact on the performance achieved by the proposed approach. Fig. 6 depicts the impact, on the system with the Titan V (S2), of setting these parameters, when processing datasets with different numbers of patients. Since datasets with 4000 SNPs have been shown to be sufficient to achieve the maximum performance (see Section 4.2), this analysis is focused on examining results for that number of SNPs. System S2 has the CPU with the most cores out of the ones considered in the experiments and the overall most capable GPU. Thus, we will rely on results obtained on that system to demonstrate the importance of the selection of suitable values for these parameters.

The impact on performance of adding more patients observed in the experiments across multiple systems is also observed here. For a given chunk size and number of CPU threads, increasing the amount of patients to process tends to improve performance, as there is more data to process per GPU kernel execution. This allows to better use the available GPU resources. As expected, the increase in performance only happens up to a point. For example, when using 8

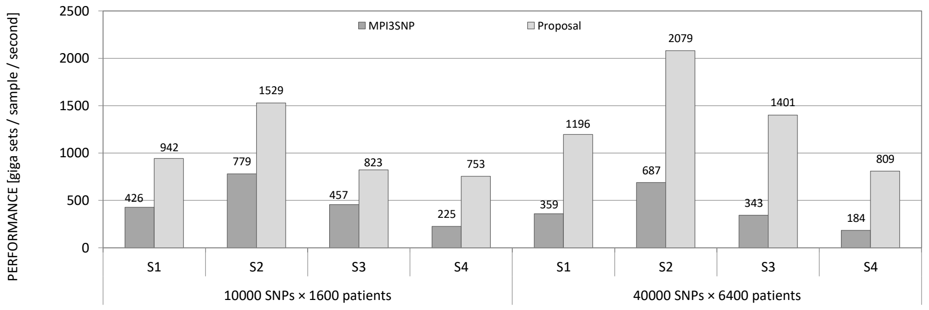


Fig. 7. Number of giga ($\times 10^9$) sets processed per second normalized to the number of patients in the dataset for MPI3SNP and the proposal on different systems.

CPU threads and a chunk size of 40000 combinations, going from 4000 patients to 8000 patients did not improve performance. This happens because the GPU kernel is already fully saturated processing the dataset with 4000 patients.

For datasets with the same amount of patients, and for the same chunk size, using more CPU threads tends to result in improved performance, as there are more rounds being concurrently executed. Thus, there is more opportunity to more efficiently utilize the CPU (for generating combination vectors) and the GPU resources (for evaluating combinations) by overlapping execution between the different phases of the proposal, including overlapping computation with memory transfers concerned with the combination vectors from the CPU to the GPU. The few existing outliers where using 8 CPU threads (e.g., 80000 combinations per chunk and 8000 patients) resulted in a small decrease in performance compared with using 4 threads might be explained by additional (or more aggressive) instances of reduction of GPU frequency under heavy load due to power, voltage, and thermal specifications.

For the same number of threads and number of patient records, relying on processing larger chunks (i.e., number of combinations processed per round) tends to result in increased performance for the configurations herein evaluated. More combinations to process per round can help to make each individual kernel execution better use the GPU compute resources. However, saturation is achieved at some point, depending on the number of patients and the number of threads. Note that this does not mean that chunk size can simply be set to an arbitrarily large value. Processing too many combinations per round, in relation to the total amount of combinations to process can result in the number of rounds being too small, at which point parallelism, achieved by concurrently executing rounds, can get severely restricted.

4.4 Comparing performance with MPI3SNP

Fig. 7 depicts results for the systems S1, S2, S3 and S4, comparing the performance archived by the proposal with MPI3SNP [11]. The performance improvement achieved by the proposal compared with MPI3SNP is 2.21×, 1.96×, 1.80× and 3.35×, for a dataset with 10000 SNPs and 1600 patients, and 3.33×, 3.03×, 4.09× and 4.39× for a dataset with 40000 SNPs and 6400 patients, on the systems S1 (Xeon E3-1245 V3 + GeForce 2070S), S2 (i9-7900X + Titan V), S3 (i7-4770K + Titan Xp) and S4 (i7-5960X + Titan X). This makes the proposal, on average 2.71×, 2.44×, 2.72× and 3.83× faster on those systems. Therefore, these experimental results reveal that the proposed approach shows improved behaviour from a performance-wise perspective with regard to MPI3SNP. The performance improvement of the proposal in relation to MPI3SNP is more substantial on the larger dataset. This is consistent with the results of the performance evaluation presented in the previous section, where it is shown that datasets with more patients are more satisfactorily handled by the proposal by better exploiting the available computing resources.

The wall-clock time required for evaluation of the dataset with 40000 SNPs and 6400 patients helps at showing the importance of efficiently using all available computing resources. For that dataset, MPI3SNP takes 4 days and 7 hours to completion on the system with the less resourceful CPU+GPU configuration (S4). In contrast, processing that dataset on system S4 using the proposal takes slightly less than a full day.

5 Related Work

Given the combinatorial characteristics and the data-parallel nature of epistasis detection, the methods proposed in the literature can be classified into different categories depending on the targeted computing systems and device architectures. There are approaches aimed at using multicore CPUs [5, 7], Intel Xeon Phi accelerators [3, 9], GPUs [3–6, 11, 16–18] and specialized architectures in FPGAs [17]. These approaches can also be categorized based on the interaction order they tackle in the context of epistasis detection. The majority of these state-of-the-art approaches focus on 2-way detection [3, 5, 7, 9, 16, 16–18], while only rare attempts are made on performing 3-way detection [4, 6, 11]. Given a set of SNPs to analyze, the evaluation of SNP triplets results in an enormous growth in the number of combinations to be considered when compared to the pairwise evaluation. As such, it is of utmost importance to efficiently leverage the capabilities of computation and memory resources of a given system in order to perform 3-way epistasis detection.

In this paper, the performance of the herein proposed approach is compared with MPI3SNP [11] (a recently published GPU-based approach for 3-way epistasis detection using mutual information as scoring function), achieving considerably higher parallel performance for different datasets. In fact, the performance of the proposal is close to the theoretical maximum achievable across all the GPU architectures considered. This is mainly due to the efficient workload distribution across multi-core CPU and GPU devices, as well as due to the highly optimized nature of the proposed algorithms for data-parallel processing of different SNP combinations on GPU cores. It is also worth to emphasize that significant efforts were made in order to ensure the high-performance calculation of the K2

Bayesian score (the objective function), which is computationally more demanding than the mutual information scoring used in MPI3SNP [11].

Another distinctive aspect lies in the way the workload is scheduled/distributed between the different system processors and/or accelerators. For instance, the herein proposed approach differs from related work that employs matrix operations to combine SNP data, such as [6], which may attain high performance for certain dataset types by relying on General Matrix Multiply (GEMM). However, these approaches require sufficiently large data-sets (especially in terms of the number of SNPs) to achieve efficiency, in order to ensure near-maximum performance of the GEMM kernels, as well as to minimize the impact of inefficient resource utilization when evaluating non-unique combinations.

6 Conclusions

Epistasis detection is a computationally challenging problem that has been attracting increased research efforts in recent years. In order to deal with this problem, this paper introduced a high-performance GPU+CPU hybrid approach for exhaustively identifying 3-way epistatic interactions according to the Bayesian K2 criterion. The proposed approach relies on the exploitation of architectural differences between CPU and GPU to orchestrate the key tasks involved in these biological analyses, defining the scope of the execution according to the properties of each operation and the characteristics of the underlying hardware.

The results of an experimental campaign covering different possible dataset shapes (i.e., more SNPs than patients, and vice versa) and sizes are presented. The achieved results show that the proposal is very close at extracting maximum performance out of all five targeted GPU+CPU configurations. The proposed method achieves high performance on 3-way searches in relation to other approaches that do the same amount of core computations for calculating the genotype frequency tables regarding all-to-all combinations of SNPs and which do not rely on specialized cores (e.g., tensor cores or specialized hardware in FPGAs). Compared with MPI3SNP [11], a recently published GPU-based approach, the proposal has been evaluated to be on average 3.83×, 2.72×, 2.44× and 2.71× faster at evaluating all unique combinations of SNPs, on systems with Xeon, i7 and i9 CPU architectures paired with a Titan X (Maxwell 2.0), a Titan Xp (Pascal), a Titan V (Volta) and a GeForce 2070S (Turing), respectively.

Ongoing work includes an extension of the proposed approach to support multiple nodes in a cluster configuration. Using as a baseline the method herein presented, inter-node processing and load balancing strategies will be integrated to exploit the characteristics of these systems, in the pursuit of highly efficient, higher-order epistasis detection.

References

[1] Buckles, B.P., Lybanon, M.: Algorithm 515: Generation of a vector from the lexicographical index [g6]. ACM Trans. Math. Softw. **3**(2), 180–182 (Jun 1977). https://doi.org/10.1145/355732.355739

[2] Cooper, G.F., Herskovits, E.: A bayesian method for the induction of probabilistic networks from data. Machine Learning **9**, 309–347 (Oct 1992). https://doi.org/10.1007/BF00994110

[3] González-Domínguez, J., Ramos, S., Touriño, J., Schmidt, B.: Parallel pairwise epistasis detection on heterogeneous computing architectures. IEEE Transactions on Parallel and Distributed Systems **27**, 2329–2340 (Aug 2016). https://doi.org/10.1109/TPDS.2015.2460247

[4] González-Domínguez, J., Schmidt, B.: GPU-accelerated exhaustive search for third-order epistatic interactions in case–control studies. Journal of Computational Science **8**, 93 – 100 (2015). https://doi.org/10.1016/j.jocs.2015.04.001

[5] Goudey, B., et al.: High performance computing enabling exhaustive analysis of higher order single nucleotide polymorphism interaction in genome wide association studies. Health Information Science and Systems **3**, S3 (Feb 2015). https://doi.org/10.1186/2047-2501-3-S1-S3

[6] Joubert, W., et al.: Attacking the opioid epidemic: Determining the epistatic and pleiotropic genetic architectures for chronic pain and opioid addiction. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis. pp. 57:1–57:14. SC '18, IEEE Press, Piscataway, NJ, USA (2018). https://doi.org/10.1109/SC.2018.00060

[7] Kässens, J.C., González-Domínguez, J., Wienbrandt, L., Schmidt, B.: UPC++ for bioinformatics: A case study using genome-wide association studies. In: 2014 IEEE International Conference on Cluster Computing (CLUSTER). pp. 248–256 (Sep 2014). https://doi.org/10.1109/CLUSTER.2014.6968770

[8] Lin, Z., et al.: Genetic association and epistatic interaction of the interleukin-10 signaling pathway in pediatric inflammatory bowel disease. World journal of gastroenterology **23**(27), 4897–4909 (Jul 2017). https://doi.org/10.3748/wjg.v23.i27.4897

[9] Luecke, G.R., et al.: Fast epistasis detection in large-scale GWAS for Intel Xeon Phi clusters. In: 2015 IEEE Trustcom/BigDataSE/ISPA. pp. 228–235 (Aug 2015). https://doi.org/10.1109/Trustcom.2015.637

[10] Niel, C., et al.: A survey about methods dedicated to epistasis detection. Frontiers in genetics **(6):285**, 1–19 (Sep 2015). https://doi.org/10.3389/fgene.2015.00285

[11] Ponte-Fernández, C., González-Domínguez, J., Martín, M.J.: Fast search of third-order epistatic interactions on cpu and gpu clusters. The International Journal of High Performance Computing Applications **34**(1), 20–29 (2020). https://doi.org/10.1177/1094342019852128

[12] Ritchie, M.D.: Finding the epistasis needles in the genome-wide haystack. In: Epistasis, Methods in Molecular Biology vol. 1253, pp. 19–33. Springer (2014)

[13] Ritchie, M.D., Hahn, L.W., Roodi, N., Bailey, L.R., Dupont, W.D., Parl, F.F., Moore, J.H.: Multifactor-dimensionality reduction reveals high-order interactions among estrogen-metabolism genes in sporadic breast cancer. American journal of human genetics **69**(1), 138–147 (Jul 2001). https://doi.org/10.1086/321276

[14] Sun, Y., et al.: epiACO - a method for identifying epistasis based on ant colony optimization algorithm. BioData mining **10**, 23–23 (Jul 2017). https://doi.org/10.1186/s13040-017-0143-7

[15] Wan, X., et al.: BOOST: a fast approach to detecting gene-gene interactions in genome-wide case-control studies. American journal of human genetics **87**, 325–340 (Sep 2010). https://doi.org/10.1016/j.ajhg.2010.07.021

[16] Wang, Q., et al.: GWISFI: A universal GPU interface for exhaustive search of pairwise interactions in case-control GWAS in minutes. In: 2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). pp. 403–409 (Nov 2014). https://doi.org/10.1109/BIBM.2014.6999192

[17] Wienbrandt, L., Kässens, J.C., Hübenthal, M., Ellinghaus, D.: 1000x faster than PLINK: Combined FPGA and GPU accelerators for logistic regression-based detection of epistasis. Journal of Computational Science **30**, 183 – 193 (2019). https://doi.org/https://doi.org/10.1016/j.jocs.2018.12.013

[18] Yung, L.S., Yang, C., Wan, X., Yu, W.: GBOOST. Bioinformatics **27**, 1309–1310 (May 2011). https://doi.org/10.1093/bioinformatics/btr114

[19] Zubenko, G.S., Hughes, H.B.r., Zubenko, W.N.: D10s1423 identifies a susceptibility locus for alzheimer's disease (ad7) in a prospective, longitudinal, double-blind study of asymptomatic individuals: results at 14 years. American journal of medical genetics. Part B, Neuropsychiatric genetics : the official publication of the International Society of Psychiatric Genetics **153B**(2), 359–364 (Mar 2010). https://doi.org/10.1002/ajmg.b.31017