Retargeting Tensor Accelerators for Epistasis Detection

Ricardo Nobre, *Member, IEEE*, Aleksandar Ilic, *Member, IEEE*, Sergio Santander-Jiménez, *Member, IEEE*, and Leonel Sousa, *Senior Member, IEEE*

Abstract—The substitution of nucleotides at specific positions in the genome of a population, known as single-nucleotide polymorphisms (SNPs), has been correlated with a number of important diseases. Complex conditions such as Alzheimer's disease or Crohn's disease are significantly linked to genetics when the impact of multiple SNPs is considered. SNPs often interact in an epistatic manner, where the joint effect of multiple SNPs may not be simply mapped to a linear additive combination of individual effects. Genome-wide association studies considering epistasis are computationally challenging, especially when performing triplet searches is required. Some contemporary computer architectures support fused XOR and population count as the highest throughput operations as part of tensor operations. This article presents a new approach for efficiently repurposing this capability to accelerate 2-way (pairs) and 3-way (triplets) epistasis detection searches. Experimental evaluation targeting the Turing GPU architecture resulted in previously unattainable levels of performance, with the proposal being able to evaluate up to 108.1 and 54.5 tera unique sets of SNPs per second, scaled to the sample size, in 2-way and 3-way searches, respectively.

Index Terms—GWAS, two- and three-way epistasis, performance evaluation, parallel architectures

1 Introduction

Variations of a nucleotide at specific positions in the genome of a given population, known as single-nucleotide polymorphisms (SNPs), have been correlated with a number of traits [1]. Genome-wide association studies (GWAS) often rely on SNPs as biological markers, identifying the SNPs that are statistically most correlated to a given trait. In a case-control study, this is achieved by processing a dataset representing the genotype of a given population for cases and controls. This has practical applications in the context of personalized medicine, discovery of disease cause, pharmacogenetics and forensics [2], [3].

Considering the combined effect of different SNPs, a phenomenon known as epistasis, allows correlating genotype and phenotype in situations for which considering SNPs individually is not sufficient [4]. However, this is computationally demanding, especially when a study must take into account more than two SNPs at a given time in the context of an exhaustive search. This raises the importance of synergically combining software development and hardware architectures to address this problem.

Epistasis detection suits parallel architectures. This includes multi-core central processing units (CPUs) [5], [6]. However, given the data-parallel nature of the problem, massively parallel architectures tend to be favoured. This includes specialized architectures in field programmable gate arrays (FPGAs) [7], [8] and, especially, graphics processing units (GPUs) [5], [8], [9], [10], [11], [12], [13]. Most methods adopted for epistasis detection only perform pairwise searches [5], [6], [7], [8], [9], [11], [12], [14], and a few triplet searches [10], [13], [15].

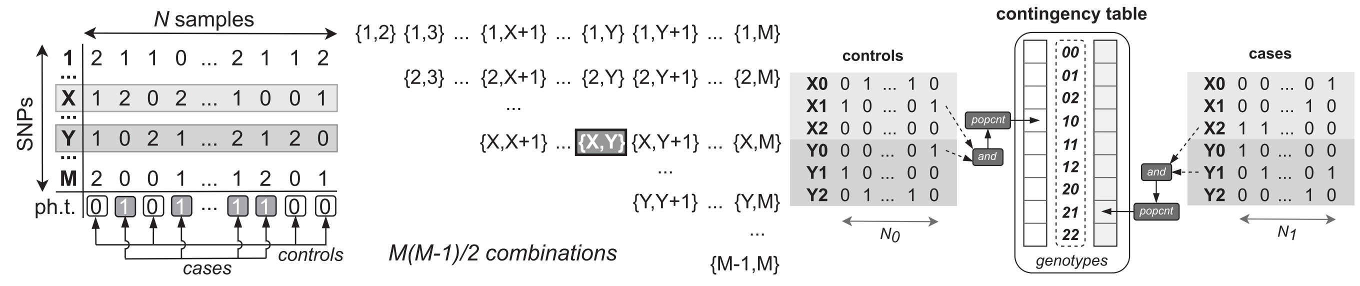
Current hardware trends are strongly focused on specialization [16]. Some contemporary accelerators have capabilities targeted at processing deep neural networks (DNNs). NVIDIA introduced in 2017, with the Volta architecture, the first generation of GPU *tensor cores*, units specialized to matrix multiplication that are mainly targeted at processing DNNs. Volta tensor cores, which operate at 16-bit precision, have been used in [10] to accelerate epistasis searches.

Advances in DNN research have allowed to further reduce compute precision. To process DNNs more efficiently, faster and/or in a more ubiquitous manner, new hardware has been introduced. Google's tensor processing unit (TPU) [17], NVIDIA Turing GPUs [18], and Intel Habanna architectures [19] are capable of performing integer matrix multiplication with eight or less bits. Quantization reduces the memory requirements and improves performance, especially if natively supported by the target architecture. The more aggressive the quantization is, the more performance, area and/or energy efficiency can improve, especially if relying on integer computation [20].

The most extreme quantization is displayed by binarized (or binary) neural networks (BNNs). Certain BNNs (e.g., XNOR-Networks [21]) approximate computation using binary operations. Through representing $+1$ by 1 and -1 by 0, a dot product of a matrix multiplication from a convolution layer is implemented with binary XNOR (or XOR),

• *Ricardo Nobre, Aleksandar Ilic, and Leonel Sousa are with the INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, 1649-004 Lisboa, Portugal. E-mail: {ricardo.nobre, aleksandar.ilic, leonel.sousa}@inesc-id.pt.*
• *Sergio Santander-Jiménez is with the Department of Computer and Communications Technologies, University of Extremadura, 06006 Badajoz, Spain. E-mail: sergio.jimenez@inesc-id.pt.*

Manuscript received 14 Oct. 2020; revised 4 Feb. 2021; accepted 14 Feb. 2021. Date of publication 18 Feb. 2021; date of current version 12 Mar. 2021. (Corresponding author: Ricardo Nobre.) Recommended for acceptance by F. Wolf. Digital Object Identifier no. 10.1109/TPDS.2021.3060322



(a) Case-control input dataset (b) All pair-wise combinations (c) Calculation of genotype frequencies for a pair of SNPs

Fig. 1. Essential concepts of epistasis detection in the context of pair-wise searches.

population count (POPC, the number of bits set to 1) and accumulation. The dot product in the -1 and $+1$ domain is derived with $2 \times \texttt{POPC}(\overline{\mathrm{A} \oplus \mathrm{B}}) - N$ (from XNOR) or $N - 2 \times \texttt{POPC}(\mathrm{A} \oplus \mathrm{B})$ (from XOR), where A and B are bit vectors and N is the number of bits per vector [22].

Some NVIDIA GPUs, starting from the Turing architecture, have native support for executing the matrix operations required to process BNNs. These GPUs deliver up to $16\times$ the throughput per cycle on tensor cores in comparison to 16-bit floating point arithmetic. However, on the Turing tensor cores, at the level of a single bit only fused XOR and population count operations are supported.

To address productivity and performance portability in deep learning and related application domains (e.g., dense linear algebra, stencil computations), recent developments in the software front include domain specific languages (DSLs) and toolchains with optimizing compilers for efficiently targeting GPUs or other parallel architectures [23], [24], [25], [26]. This paper proposes algorithms for exhaustive 2-way (pairs) and 3-way (triplets) epistasis detection searches[1] that efficiently exploit the current and future architectures with tensor processing hardware and native support for fused XOR and population count operations. We make the following three main contributions:

1) a method that enables the efficient use of fused XOR and population count as part of matrix operations in the context of 2-way and 3-way epistasis searches;
2) specialized internal representations of the input dataset are explored to boost core computations;
3) the amount of SNP combinations per round of evaluations is balanced to improve efficiency when datasets with different shapes are processed.

Contribution 1) pertains to the exploitation of architectures that support fused XOR and population count with significantly higher throughput in relation to other operations. This capability is used in the scope of an optimized strategy for counting the frequency of each genotype that results from combining two (2-way search) or three (3-way search) SNPs, considering all-to-all combinations. Contribution 2) consists in exploring the use of a specialized data representation in memory that better suits the particular computations performed and the targeted hardware. Contribution 3) concerns with the evaluation of the impact of using different amounts of SNPs in the individual matrix operations to tune the performance of epistasis searches.

Our work focuses on proposing an approach for repurposing the novel capabilities in current GPU architectures for epistasis detection purposes. The proposed approach largely derives its performance from translating computations pertaining to counting genotype frequencies to tensorized binary operations. These frequencies are the starting point for calculating a multitude of objective scoring functions, making the proposal suitable to accelerate different epistasis detection scenarios.

The organization of this paper is the following. Section 2 presents a formulation of the problem under study. Section 3 introduces the proposal, describing the internal dataset representation herein used and focusing on efficient construction of contingency tables in 2-way searches. This includes explaining, still in the context of pairwise searches, how the proposal exploits hardware capable of natively performing fused XOR and population count as part of matrix operations. Section 4 explains how the proposal tackles 3-way searches. Section 5 details how the proposed methods are efficiently targeted at modern GPUs capable of very high throughput tensorized XOR and population count. Section 6 introduces the systems that have been used, and presents and discusses the achieved experimental results. Section 7 presents related work, including a performance-wise comparison in relation to state-of-the-art approaches. Final remarks and conclusions are drawn in Section 8.

2 Problem Formulation

An epistasis detection search is concerned with the identification of correlation between sets of interacting SNPs (genotype) and a given condition or disease (phenotype). This is accomplished taking as input a dataset D, of size $N \times (M+1)$, where N represents the number of case-control samples and M is the number of SNPs to take into consideration. A given entry $D[i,j], i \in \{1,\ldots,N\}, j \in \{1,\ldots,M\}$ represents the genotypic configuration observed at the ith sample for the jth SNP. There are three possible genotypic configurations (alleles) that a sample can take for a given SNP. These are the homozygous major (AA), the heterozygous (Aa) and the homozygous minor (aa) alleles; encoded as 0, 1 or 2, respectively. Finally, the entries $D[i, M+1]$ hold a 0 if the ith sample is a control and 1 otherwise (i.e., if it is a case). Fig. 1a depicts a representation of a dataset in the format described above, where lines represent SNPs and columns represent samples.

Epistasis detection involves the identification of the combination of SNPs $[x_1, x_2, \ldots, x_k]$ that is most correlated with

1. Source code repository with implementations available at: https://github.com/hiperbio/tensor-episdet.

the trait under study, where k represents the interaction order. Exhaustive approaches, i.e., considering all-to-all combinations of k SNPs at a time (k-way search), can be implemented in the following three steps:

Step 1) assess frequencies of each of the 3^k combined genotypes in the input dataset, for each combination of SNPs, for cases and controls;

Step 2) score each set of SNPs, based on the number of occurrences of each of the combined genotypes for cases and controls;

Step 1) reduce scores to identify the set of SNPs that is most correlated to phenotype in regard to a given objective scoring function.

The amount of solutions to evaluate, depicted in Fig. 1b for 2-way searches (i.e., $k = 2$), is $\frac{M!}{k!(M-k)!}$ for the general case. The computational complexity is especially impacted by the amount of SNPs M in the input data and the interaction order k. The latter results in the exponential growth with the number of sets considered [27]. Moreover, the interaction order also impacts the number of combined genotypes (3^k) that need to be accounted per candidate set, which further contributes to increase the required computational resources and/or time-to-solution. The impact of the number of samples on the computational cost is linear.

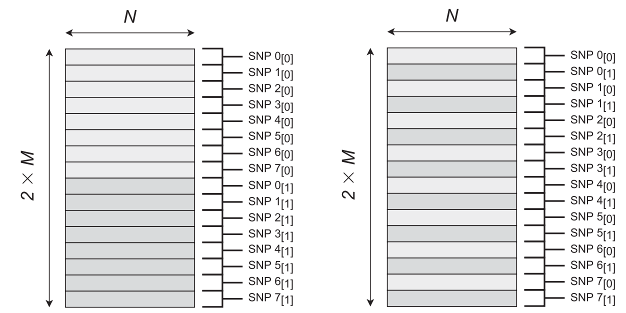
In order to achieve the performance levels that are required to process challenging datasets, current hardware trends need to be taken into account when designing and developing high-throughput high-order exhaustive epistasis detection solutions. Recurrent calculations for the determination of the frequencies of genotypes, stored in a *contingency* or *frequency* table, must be implemented in such a way that makes full use of the selected target architectures. Notice that the construction of these tables entails core calculations that are independent of the objective scoring function used to guide the search (e.g., G-test [28], Gini score [29], K2 Bayesian score [30], mutual information [31]).

3 Accelerating Epistasis Detection With Native Binary Processing Capabilities

Given the large amount of computation and memory accesses required in the context of exhaustive epistasis searches, it is imperative to efficiently use the available hardware. This section presents how the proposal in the paper is tailored for high performance, including how to exploit high throughput fused XOR and population count, a novel capability found in tensor-based architectures.

3.1 Data Representation

The internal representation proposed herein is based on a binarized representation of the input data first introduced in the context of epistasis detection for targeting CPUs [32]. Fig. 1c illustrates the use of this type of representation in the context of the calculation of genotype frequencies (*Step 1* in Section 2) for 2-way searches. The information pertaining to a given SNP is stored in six bit vectors, each representing the occurrence of one of the three possible individual genotypes in the samples in the input data, for controls or cases (three bits per sample). Each bit vector has a length of N_0 or N_1 bits, the number of controls or cases, respectively, with each position of a bit vector representing a single sample. A bit at a specific position of one of these bit vectors is set to 1 if the sample it represents (case or control) has the particular SNP allele that the bit vector represents, and 0 otherwise. Genotype frequencies are stored in a contingency table, represented as two columns at the center of Fig. 1c (left column for controls, right column for cases), each holding as many values as the amount of different possible genotype combinations in a given SNP interaction (9 in 2-way searches).



(a) Non-interleaved (b) Interleaved

Fig. 2. Non-interleaved and interleaved internal data representation.

On top of reducing the memory footprint of a given input dataset, which improves the effective memory bandwidth, such representation allows one to rely on bitwise operations and population counts. These operations are natively supported in hardware in contemporary parallel architectures, allowing to achieve higher performance compared with solutions directly operating, for example, on genotypic data represented by 0, 1 and 2. Notice that while splitting the dataset into controls and cases can be an optimization useful on its own, as it renders it unnecessary to load information about the phenotype during computation, it is a requirement in the context of the proposal for enabling direct mapping of the output of the binary operations to updates to the contingency tables.

The proposal uses a technique that allows achieving exhaustive detection relying on information about two genotypes out of the three possible genotypes (AA, Aa or aa). Thus, only four bit vectors per SNP (two for controls and two for cases), instead of six, are represented in memory. Frequency counts for the homozigous minor allele (aa) pertaining to each SNP, represented by the number 2 in the format in Fig. 1a, are calculated from the counts of the two other genotypes. Counts for combined genotypes having this genotype in any of the SNPs that are part of a given combination are also inferred as explained in detail in Sections 3.4 and 4.2, in 2-way and 3-way detection, respectively. The proposal relies on an interleaved memory representation, depicted besides an alternative non-interleaved representation in Fig. 2. In the interleaved representation, the two bit vectors pertaining to the two represented genotypes of a given SNP are contiguous in memory. This impacts on how data is processed, enabling improved efficiency in the context of tensorized computations.

3.2 Relating Epistasis Detection to Matrix Operations

DNN processing is typically accelerated using hardware specialized to perform matrix multiplication in floating-point or integer numerical types with reduced precision. The use of highly efficient general matrix multiply (GEMM) kernels to perform core computations as part of epistasis searches has been recently proposed for targeting tensor hardware [10]. In order to make efficient use of the tensor hardware capabilities targeting BNNs, one has to be able to relate epistasis detection searches to matrix operations where XOR and population count replace multiply-add.

The calculation of the occurrences of each of the nine genotypes in pairwise detection, for cases and controls, can be performed using matrix operations that resemble matrix multiplication, with bitwise AND and population count replacing multiply-and-add. Notice that applying the AND operation between pairings of bit vectors from different SNPs, representing the presence of specific alleles, results in bit vectors with set bits (i.e., set to 1) in positions representing samples (cases or controls) that have the particular resulting genotype. The mapping to fused XOR and population count operations is explained in the following section.

Relying on this type of matrix processing can result in a large number of repeated or non-useful sets of SNPs being evaluated. Notice that in 2-way searches, the ratio of combinations where repetitions are not allowed, herein referred to as *unique sets*, to all permutations (including repetitions) is given by the fraction $\frac{\binom{M}{2}}{M^2}$. For a large M, if all permutations were evaluated, only close to $1/2$ would represent unique sets. Notice that combining SNPs X and Y is not dependent on their order. Thus, instead of processing all genotypic data at once (for controls or cases), the data is processed in multiple rounds, where each round processes distinct combinations of blocks of contiguous SNPs from the controls matrix or the cases matrix. This has the additional effect of reducing the memory requirements. Following from $\frac{\binom{M}{2}}{(\sum_{i=0}^{M/B} \frac{M}{B} - i) \times B^2}$, considering blocks of B SNPs and a total of M SNPs, the ratio of unique sets evaluated is $\frac{M-1}{B+M}$. In short, this ratio improves with the use of smaller matrices, in regard to the amount of SNPs, in the kernel calls implementing these operations. Still, one should pay attention to the fact that these matrix operations, when individually executed, are expected to achieve higher performance with large inputs, as is typically the case with GEMM kernels.

3.3 Fused XOR and POPC as Core Operations

The calculation of genotype frequencies using binary operations typically entails using bitwise AND followed by population count. However, to leverage the type of hardware the proposal in this paper targets, it is required to express epistasis detection calculations resorting to fused XOR and population count operations. Contrary to what happens in the context of BNNs, matrix multiplication in the -1 and $+1$ domain is not a core calculation in epistasis searches. Thus, instead of using the equations used when processing BNNs, we devised a specialized formula based on Lemma 3.1.

Lemma 3.1. *Given the three allowed elementwise pairings of bits resulting from combining bit vectors A and B, where at least one of the bit vectors is set to 1 at the given position, then Equation (1) relates the number of bits set to 1 following the AND operation to the number of bits set to 1 following the XOR operation.*

$$\mathrm{POPC}(A \cdot B) = \frac{\mathrm{POPC}(A) + \mathrm{POPC}(B) - \mathrm{POPC}(A \oplus B)}{2}. \quad (1)$$

Proof. Using C_1, C_2 and C_3 to denote the number of positions set to 1 only in bit vector A, only in bit vector B and in both bit vectors, respectively, then one arrives to the equation above from the following equivalences.

$$\mathrm{POPC}(A) = C_1 + C_3 \quad (2)$$

$$\mathrm{POPC}(B) = C_2 + C_3 \quad (3)$$

$$\mathrm{POPC}(A \oplus B) = C_1 + C_2 \quad (4)$$

$$\mathrm{POPC}(A \cdot B) = C_3. \quad (5)$$

Subtracting the number of occurrences of 1's pairing with a 0 — Equation (4) — from the total number of positions set to 1 in bit vectors A and B — sum of Equations (2) and (3) — only the number of occurrences of 1's pairing with 1 remain — $2 \times C_3$. Through division by 2, one gets the number of pairs of intersecting 1's — Equation (5). This corresponds to counting the number of bits set to 1 after performing the AND operation between the bit vectors. □

Individual SNP population counts — represented by terms $\mathrm{POPC}(A)$ and $\mathrm{POPC}(B)$ for two different SNPs — are calculated once and used in all pair-wise combinations. This mitigates the computational cost of the computation.

3.4 Optimizing Calculations in 2-Way Searches

The counts pertaining to the three possible genotypes, considering a given SNP individually, have to add exactly to the number of records in the dataset. For a given SNP, samples that do not have one of two of the genotypes, out of the possible three, will have the remaining kind. Taking $\{X_\alpha, Y_\beta\}$ to denote the value corresponding to the hypothetical execution of a bitwise AND operation between bit vectors representing allele types (α and β) of SNPs X and Y followed by the population count operation on the resulting bit vector (mapped to XOR+POPC as detailed in the previous section); and $\{:, Y_\beta\}$ to represent the number of bits set to 1 (i.e., population count) in bit vector Y_β, one can rely on the equivalence:

$$\{X_2, Y_\beta\} = \{:, Y_\beta\} - (\{X_0, Y_\beta\} + \{X_1, Y_\beta\}). \quad (6)$$

This equivalence allows to analytically derive the number of occurrences of genotypes that have an allele of type 2 (homozygous minor: aa) in at least one of the SNPs (X in the equation). Having calculated $\{X_0, Y_0\}$ and $\{X_1, Y_0\}$, $\{X_2, Y_0\}$ is analytically derived. Similarly, after calculating $\{X_0, Y_1\}$ and $\{X_1, Y_1\}$, one can derive $\{X_2, Y_1\}$. Notice that the values for $\{:, Y_\beta\}$ corresponding to each of the possible allele types for SNP Y are reused multiple times for combination with any SNP X. At this point, the terms yet to determine are the ones involving the combination of X_0, X_1 and X_2 with Y_2, which are derived relying on the same principle, with X and Y swapped in the equation above.

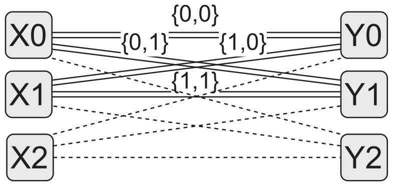


Fig. 3. Calculation of genotype frequencies for a 2-way SNP interaction.

Fig. 3 illustrates this process. Dashed lines represent genotypes whose frequencies are derived combining knowledge about frequencies for other genotypes (represented by double lines) with frequency counts of individual SNP alleles. The decomposition herein used only relies on processing genotypic data relative to two of the three possible alleles, which allows a proportional reduction in memory use.

The complete contingency table (2×3^2 values) for any given pair of SNPs in a 2-way search is constructed from the values calculated for 2^2 combined genotypes (separately for cases and controls). The remaining $3^2 - 2^2$ (i.e., five) can be calculated with simple arithmetic operations (sums and subtractions), allowing to reduce the number of fused XOR and population count operations by a factor of $2.25\times$.

3.5 Pairwise Detection Searches Overview

Algorithm 1 represents the pseudocode of a 2-way search, including the construction of contingency tables for all pairs of SNPs, scoring those pairs and finding the pair most correlated to phenotype (given a scoring function).

Algorithm 1. Pseudocode of a 2-Way Search

```
Data: $h_0, h_1, M, N_0, N_1, B$
Result: $s$
$hostToDevice(h_{0|1}, d_{0|1}, M, N_{0|1}, B, 0);$
$pop_{0|1} = indivPop(d_{0|1}, M, N_{0|1}, B, 0);$
for $X_i = 0;\ X_i < M;\ X_i += B$ do
    for $Y_i = X_i;\ Y_i < M;\ Y_i += B$ do
        if $(X_i == 0)$ && $((Y_i + B) < M)$ then
            $hostToDevice(h_{0|1}, d_{0|1}, M, N_{0|1}, B, Y_i + B);$
            $pop_{0|1} = indivPop(d_{0|1}, M, N_{0|1}, B, Y_i + B);$
        end
        $xorPop_{0|1} = tensorXorPop(d_{0|1}, M, N_{0|1}, B, X_i, Y_i);$
        $cTab_{0|1} = xorPop2andPop(xorPop_{0|1}, pop_{0|1}, M, B, X_i, Y_i);$
        $cTab_{0|1} = genFreqInfer(cTab_{0|1}, pop_{0|1}, M, X_i, Y_i);$
        $scores = applyScore(cTab_0, cTab_1);$
        $s = findGloballyBestSol(scores, s);$
    end
end
```

The matrices composed of bitvectors representing genotypic data (see Section 3.1), h_0 (controls matrix) and h_1 (cases matrix), the amount of controls and cases, N_0 and N_1, the number of SNPs M, and the number of SNPs per block used for combination, B, are inputs to Algorithm 1. The algorithm assumes that the dataset is in (or has been preprocessed to) the binary representation. Inputs with `0|1` in subscript (e.g., $h_{0|1}$, $d_{0|1}$, $N_{0|1}$, $xorPop_{0|1}$) represent separated execution for controls (0) or cases (1).

Counting genotype frequencies taking into account SNPs individually (`indivPop`) is very fast. This is performed in blocks because the dataset is transferred to the accelerator (`hostToDevice`) in blocks. Transferring to the device memory can take non-negligible time for some datasets, which is dealt with by overlapping with computation on the device. Notice that the cost of transferring the dataset grows linearly both with the number of samples and the number of SNPs. In contrast, while the computational cost grows linearly with the number of samples, it grows by a factor of close to $4\times$ with the number of SNPs, as $\binom{M}{2}$ represents the number of combinations that need to be processed. The X_i and Y_i loop iterators are incremented by B every iteration. In order to avoid processing (as much as possible) non-unique sets of SNPs, the iterator of the innermost loop in Algorithm 1, Y_i, assumes values from X_i (iterator of the outermost loop) to $M - B$. The matrices representing the dataset are padded in situations where M is not a multiple of B.

The execution of the XOR+POPC matrix operations using tensor hardware (`tensorXorPop`) followed by the construction of complete contingency tables from the output of the former (`xorPop2andPop` and `genFreqInfer`), the application of a scoring function on top of contingency tables (`applyScore`) and the reduction of scores to find the best candidate solution (`findGloballyBestSol`) correspond to *Step 1*, *Step 2* and *Step 3* of the epistasis detection algorithm introduced in Section 2. Calls to `tensorXorPop` and `xorPop2andPop` implement the approach detailed in Sections 3.2 and 3.3; `genFreqInfer` is explained in Section 3.4, while the particular type of computations and implementation details pertaining to `findGloballyBestSol` and `applyScore` are introduced in Sections 5.3 and 5.4, respectively. Execution of these steps is performed in rounds for each pair of blocks of SNPs. The use of the interleaved layout (see Section 3.1) makes each round map to a single matrix operation per phenotype state. A total of 8 matrix operations are required for the non-interleaved approach.

The proposal in this paper allows by design the concurrent execution of evaluations pertaining to different combinations of blocks of SNPs. This allows maximizing the use of the targeted device, especially in situations where the block size is small in relation to the available resources. Moreover, concurrent execution results in the overlap of different compute and memory related operations, enabling the more efficient usage of different types of execution units and the memory hierarchy of the targeted device as a whole.

Fig. 4 depicts the combination of two blocks of SNPs X and Y, considering, for illustration purposes, a dataset with 8 SNPs and a block size B of 2 SNPs. The 16 cells, represent 4 (proto) genotype frequencies for each of the 4 pairs of SNPs resulting from combining (all-to-all) 2 SNPs X with 2 SNPs Y. These values are used to derive the 4 frequencies for the genotype combinations (0,0), (0,1), (1,0) and (1,1). Then, from those, the remaining 5 frequencies, for the genotypes (0,2), (1,2), (2,0), (2,1) and (2,2), are calculated.

In the depicted example, X_i equals 0 and Y_i equals 2, thus it represents the second round executed by the pair of loops from Algorithm 1 (i.e., the second combination of a block of SNPs X with a block of SNPs Y). In this evaluation round, all pairs of SNPs are unique. The only situations that result

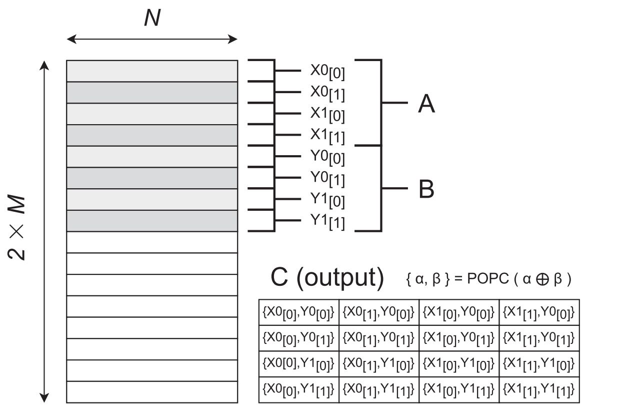

N
2 × M
X0[0]
X0[1]
X1[0]
X1[1]
Y0[0]
Y0[1]
Y1[0]
Y1[1]
A
B
C (output)
{ α, β } = POPC (α ⊕ β)
{X0[0],Y0[0]} {X0[1],Y0[0]} {X1[0],Y0[0]} {X1[1],Y0[0]}
{X0[0],Y0[1]} {X0[1],Y0[1]} {X1[0],Y0[1]} {X1[1],Y0[1]}
{X0[0],Y1[0]} {X0[1],Y1[0]} {X1[0],Y1[0]} {X1[1],Y1[0]}
{X0[0],Y1[1]} {X0[1],Y1[1]} {X1[0],Y1[1]} {X1[1],Y1[1]}

Fig. 4. Combining blocks of SNPs X and Y. Each cell represents the operation POPC $(\alpha \oplus \beta)$.

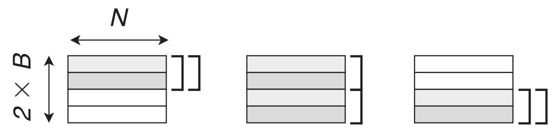


Fig. 5. Processing all pairwise combinations inside a block of B SNPs using subblocks of $\frac{B}{2}$ SNPs.

in some of the sets of SNPs processed in the matrix operations being non-unique are when X_i equals Y_i, such as is the case of the first iteration of the nested loops (both X_i and Y_i equal to 0). These combinations of blocks are still performed, as $\frac{B-1}{2B}$ of the resulting pairs of SNPs are unique.

Our proposal also supports using a *variable* block size strategy. This strategy relies on the use of smaller blocks (i.e., subblocks) to process combinations of SNPs that, if using a fixed block size strategy, would result in processing all intra-block permutations of SNPs in each of the $\lceil\frac{M}{B}\rceil$ blocks of B SNPs. In some cases, the use of subblocks can lead to an improvement of the ratio of unique sets that is sufficient to offset the possible loss in efficiency in the use of the GPU resources, resulting in overall higher performance at epistasis detection searches. Fig. 5 represents, for illustrative purposes, considering a block size B of 2 SNPs, how all pairings of SNPs that are inside of any given block of size B (i.e., $X_i == Y_i$) can be processed using subblocks with $\frac{B}{2}$ SNPs in the context of a 2-way search. In total, considering the whole dataset, the final pass of such 2-step approach to combining SNPs entails performing $\lceil\frac{M}{B}\rceil \times 3$ matrix operations per sample type. The terms on the left and right side of the multiplication sign represent the number of blocks of size B in the dataset and the number of combinations of subblocks of size $\frac{B}{2}$ per block of B SNPs, respectively.

In cases where B is sufficiently large, it is possible for such strategy to benefit from additional intermediate passes. Considering one additional pass (i.e., three in total), the second pass (i.e., after processing all combinations of distinct blocks of SNPs of size B) entails a single matrix operation using subblocks of $\frac{B}{2}$ SNPs, per block of size B, per sample type (cases and controls). Combination of subblocks of the same index is only performed in the final pass, considering subblocks of $\frac{B}{4}$ SNPs, resulting in $\lceil\frac{M}{(\frac{B}{2})}\rceil \times 3$ additional matrix operations. Notice that only the final pass results in processing repeated combinations of SNPs.

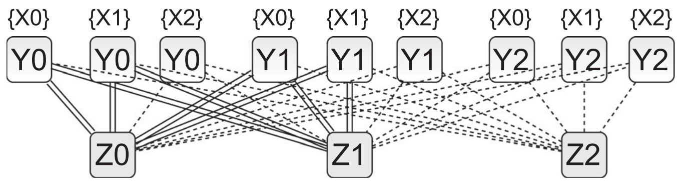


Fig. 6. Calculation of genotype frequencies for a 3-way SNP interaction.

4 Generalizing for Third-Order Searches

In 3-way epistasis detection searches, if evaluating all permutations, only close to $1/6$ (from $\frac{\binom{M}{3}}{M^3}$) of the evaluated triplets would be of real use. Following from $\frac{\binom{M}{3}}{(\sum_{i=0}^{M/B}(i+1)\times B\times(\frac{M}{B}-i))\times B^2}$, the ratio of useful triplets of SNPs is $\frac{(M-2)(M-1)}{(B+M)(2B+M)}$. As is the case for 2-way searches, as the number of SNPs per combination block grows, the ratio of unique sets (i.e., combinations of SNPs divided by possible permutations) decreases. However, here the effect of B is significantly more pronounced. Moreover, notice that the method for using XOR and population count operations for processing a binarized dataset, as described in the previous section, has been tailored for pairwise searches. Exploiting these operations in the context of 3-way searches poses new challenges. In this section we introduce their nature and explain how the proposal in this paper overcomes them.

4.1 Fused XOR and POPC in 3-Way Searches

If relying only on general-purpose cores, combining a triplet of SNPs (X, Y, and Z) can be performed through the application of the AND operation. First, between bit vectors representing alleles of SNPs X and Y, followed by the combination with SNP Z, via an additional AND operation, per allele type, with the bit vectors resulting from the previous step. Considering cases and controls, this results in 4×3^3 AND operations and 2×3^3 population counts (across the whole vectors). The exact number of instructions depends on their data-width and the number of samples.

The proposed approach iteratively combines an SNP X with a block of SNPs Y, relying on general-purpose execution units, and offloads the combination of the output of the previous step with a block of SNPs Z to the specialized units that perform fused XOR and population count operations. The cost of combining an SNP X with each SNP in a block of SNPs Y in general-purpose execution units is hidden behind the considerably larger computational complexity related to the final combination with a block of SNPs Z.

There is extensive reuse of operations. The output of the AND operations performed for the construction of the matrix combining an SNP X with a block of SNPs Y is reused for combining with multiple blocks of SNPs Z. Notice that the bit vectors resulting from each combination of a given SNP X with an SNP from a block of SNPs Y are used B times per combination with a block of SNPs Z.

4.2 Optimizing Calculations in 3-Way Searches

The strategy used to determine the combined genotype frequencies in the context of 3-way interactions (illustrated in Fig. 6) requires processing only eight genotypes with tensorized operations. The genotypes represented by dashed lines are calculated using previously calculated population counts for pairwise SNP interactions. Notice that this represents only an increase of $2\times$ the number of bit-level XOR operations in relation to pairwise searches, while the number of genotypes increases by a factor of $3\times$.

Combined genotypes with one or more interacting SNPs with allele of kind 2 (aa) are analytically derived. Equation (7) represents the equivalence that allows one to derive the frequencies of combined genotypes where SNP X assumes the aa allele. Frequencies where the SNP Y and/or the SNP Z is of kind 2 are found relying on the same approach.

$$\{X_2, Y_\beta, Z_\gamma\} = \{:, Y_\beta, Z_\gamma\} - (\{X_0, Y_\beta, Z_\gamma\} + \{X_1, Y_\beta, Z_\gamma\}). \tag{7}$$

The overall impact of the analytical derivation is larger than in 2-way. In 3-way searches, the amount of tensorized operations is reduced by $3.375\times$ (versus $2.25\times$), as only 8 of the 27 genotypes are mapped into matrix operations.

4.3 Triplet Detection Searches Overview

In the particular case of 3-way searches, the algorithm must be devised in a way that takes into account memory use, given the growth in terms of number of combinations to evaluate. Algorithm 2 presents the pseudocode for a 3-way search. In relation to pairwise searches, triplet searches rely on the following additional steps: a) calculation of population counts for allelic combinations between B SNPs in a block of SNPs Y, starting at Y_i, and SNPs with indexes larger or equal to Y_i (`pairPopBlockY`); b) combination of a SNP X with a block of SNPs Y using the AND operation (`combineXandY`); c) calculation of population counts for allelic combinations between an individual SNP X and SNPs with index larger or equal to Y_i (`pairPopX`).

Algorithm 2. Pseudocode of a 3-way Search

```
Data: $h_0, h_1, M, N_0, N_1, B$
Result: $s$
$hostToDevice(h_{0|1}, d_{0|1}, M, N_{0|1});$
$pop_{0|1} = indivPop(d_{0|1}, M, N_{0|1});$
for $Y_i = 0;\ Y_i < M;\ Y_i += B$ do
  $popBlockY_{0|1} = pairPopBlockY(d_{0|1}, M, N_{0|1}, B, Y_i);$
  for $X_i = 0;\ X_i < (Y_i + B);\ X_i += 1$ do
    $XandY_{0|1} = combineXandY(d_{0|1}, M, N_{0|1}, B, X_i, Y_i);$
    $popX_{0|1} = pairPopX(d_{0|1}, B, N_{0|1}, X_i, Y_i);$
    for $Z_i = Y_i;\ Z_i < M;\ Z_i += B$ do
      $xorPop_{0|1} = tensorPopXor(XandY_{0|1}, d_{0|1}, M, N_{0|1}, B, Z_i);$
      $cTab_{0|1} = xorPop2andPop(xorPop_{0|1}, pop_{0|1}, popX_{0|1},$
      $M, B, Y_i, Z_i);$
      $cTab_{0|1} = genFreqInfer(cTab_{0|1}, popX_{0|1},$
      $popBlockY_{0|1}, M, X_i, Y_i, Z_i);$
      $scores = applyScore(cTab_0, cTab_1);$
      $s = findGloballyBestSol(scores, s);$
    end
  end
end
```

The remaining parts of the algorithm are mostly equivalent to the ones performed in 2-way searches, with a core difference related to having to deal with larger contingency

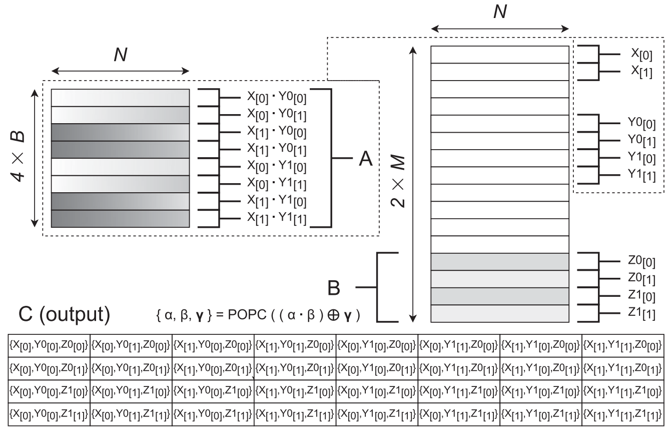

N
4 × B
X[0] · Y0[0]
X[0] · Y0[1]
X[1] · Y0[0]
X[1] · Y0[1]
X[0] · Y1[0]
X[0] · Y1[1]
X[1] · Y1[0]
X[1] · Y1[1]
A
N
2 × M
X[0]
X[1]
Y0[0]
Y0[1]
Y1[0]
Y1[1]
Z0[0]
Z0[1]
Z1[0]
Z1[1]
B
C (output)
{ α, β, γ } = POPC ((α · β) ⊕ γ)
{X[0],Y0[0],Z0[0]} {X[0],Y0[1],Z0[0]} {X[1],Y0[0],Z0[0]} {X[1],Y0[1],Z0[0]} {X[0],Y1[0],Z0[0]} {X[0],Y1[1],Z0[0]} {X[1],Y1[0],Z0[0]} {X[1],Y1[1],Z0[0]}
{X[0],Y0[0],Z0[1]} {X[0],Y0[1],Z0[1]} {X[1],Y0[0],Z0[1]} {X[1],Y0[1],Z0[1]} {X[0],Y1[0],Z0[1]} {X[0],Y1[1],Z0[1]} {X[1],Y1[0],Z0[1]} {X[1],Y1[1],Z0[1]}
{X[0],Y0[0],Z1[0]} {X[0],Y0[1],Z1[0]} {X[1],Y0[0],Z1[0]} {X[1],Y0[1],Z1[0]} {X[0],Y1[0],Z1[0]} {X[0],Y1[1],Z1[0]} {X[1],Y1[0],Z1[0]} {X[1],Y1[1],Z1[0]}
{X[0],Y0[0],Z1[1]} {X[0],Y0[1],Z1[1]} {X[1],Y0[0],Z1[1]} {X[1],Y0[1],Z1[1]} {X[0],Y1[0],Z1[1]} {X[0],Y1[1],Z1[1]} {X[1],Y1[0],Z1[1]} {X[1],Y1[1],Z1[1]}

Fig. 7. Combining an SNP X with blocks of SNPs Y and Z. Each cell represents the operation POPC $((\alpha \cdot \beta) \oplus \gamma)$.

tables. As is the case of 2-way searches, the fused XOR and population count operations (`tensorPopXor`) followed by the construction of contingency tables (`xorPop2andPop` and `genFreqInfer`), application of the scoring function (`applyScore`) and the reduction of the scores (`findGloballyBestSol`) map to *Step 1*, *Step 2* and *Step 3* introduced in Section 2; and can be executed concurrently by processing different combinations of blocks of SNPs at the same time. Execution of `pairPopX` and `pairPopBlockY` can be done concurrently with the next algorithm steps until right after execution of `tensorPopXor`. The use of the interleaved representation allows the step performing the XOR+POPC to be implemented in a single matrix operation per phenotypic state. A total of 16 matrix operations are performed with the non-interleved approach.

The calculation of population counts for pairwise interactions, used in `xorPop2andPop` and `genFreqInfer` to construct contingency tables from the output of execution on the tensor cores, is split into two parts for improving algorithmic efficiency, particularly in relation to memory usage. Compared with calculating population counts for all-to-all pairwise combinations of SNPs all at once, this decomposition requires significantly less memory, while supporting efficient overlapping with other algorithm steps.

Fig. 7 depicts the input matrices to the XOR+POPC matrix operations and the output matrix, for an example with 8 SNPs and relying on a block size B of 2 SNPs. The input *matrix A* is generated from the binarized dataset, as represented at the right side of that figure. The 32 cells of the output *matrix C* represent 8 (proto) genotype frequencies for each of the 4 triplets of SNPs resulting from combining (all-to-all) a single SNP X with 2 SNPs Y and 2 SNPs Z. These values are used to calculate the 8 frequencies for the genotypes (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0) and (1,1,1). From those frequencies, the remaining 19 are derived with simple arithmetic operations.

5 Targeting GPUs With Tensor Cores Providing High XOR and POPC Throughput

In certain modern GPUs, such as is the case with NVIDIA Turing GPUs with tensor cores, fused XOR and population count, executed in tensorized fashion with 32-bit integer accumulation, are the highest throughput operations. Thus, such GPUs are highly suitable as a target for the proposal. Due to their nature, they are also suited to the computations required for the classification (i.e., scoring) based on the contingency tables and for the reduction operations. This allows one to rely on the GPU to perform all types of computations in epistasis detection searches, eliminating the need for potentially costly synchronization primitives and/or memory transfers between different kinds of accelerators via the host such as in [8].

5.1 Turing GPU Architecture

The Turing streaming multiprocessor (SM) is the most basic building block in the Turing architecture [18]. Each SM is composed by four main processing blocks. Each one of the latter has one instruction scheduling and dispatch unit, 64 KBytes of register file space, $16\times$ IEE754 32-bit floating point scalar arithmetic logic units (ALUs), $16\times$ 32-bit integer scalar ALUs, two tensor cores, four load/store units and four special function units. In addition, each SM has four texture units, 96 KBytes of L1 cache/shared memory, one ray tracing core and two scalar 64-bit floating-point ALUs.

5.2 Throughput of Core Operations

In order to assess the importance of mapping core computations of epistasis detection to XOR and population count operations when targeting Turing GPUs, it is important to take into consideration the peak performance advantage compared with relying on the more conceptually straightforward use of AND and population count operations.

On the general-purpose cores of the GPU (i.e., CUDA cores), the bitwise AND, XOR and POPC instructions operate on 32-bit words. AND and XOR instructions execute, on the CUDA cores in the Turing architecture, with a throughput of 64 instructions per cycle per SM, resulting in 2048 bits processed per clock cycle per SM. The throughput of POPC instructions per SM on the general-purpose cores is 16 per cycle ($4\times$ lower than for AND or XOR), which results in 512 bits per clock cycle per SM. However, the eight tensor cores in a SM are capable of operating 8192 bits (fragment size is $8 \times 8 \times 128$) per cycle with fused XOR + POPC, the only operation supported at the level of a single bit.

Notice that there would be no advantage in relying on XOR instead of AND (in addition to POPC) if not using the tensor cores, sinse there is no difference in throughput between these operations on CUDA cores. The throughput of POPC represents the absolute maximum performance ceiling for the epistasis searches. Fused XOR and population count on the tensor cores allows achieving a considerably higher throughput in comparison to executing AND followed by POPC on the general-purpose cores. Moreover, POPC has a lower throughput than AND (or XOR), which results in a potential for a performance improvement of an order of magnitude, if the tensor cores are used as proposed.

5.3 Exploiting the Heterogeneous GPU Architecture

To fully utilize a GPU with tensor cores, one has to map a given application in such a way that fully exploits the heterogeneity of the architecture. The proposal allows offloading a key portion of the calculation of genotype frequencies in 2-way and 3-way epistasis searches to the tensor cores. The kernel that relies on fused XOR and population count on the tensor cores has been implemented using CUTLASS [33], a collection of CUDA template abstractions. The first and second inputs to the binary matrix operations in CUTLASS are row-major and column-major, respectively. Thus, SNP allelic combinations using the matrix-matrix operations are produced from the same genotypic data representation in memory. The remaining steps of the proposal rely exclusively on the general-purpose GPU cores.

The implementation targeting the Turing GPUs exploits the fact that the proposal allows concurrent execution of operations pertaining to different combinations of SNPs. Multiple CUDA streams are used, each processing a particular combination of blocks of SNPs at a given time. This results in overlapping kernel calls that make use of the tensor cores and calls that exclusively use the regular datapath. Overlapping different combination rounds allows masking the cost of certain computations and memory accesses, as more resources of the GPU (of the same and of different kind) have a chance of being exercised at any given time.

Derivation of genotype frequencies from the output of the kernel that uses the tensor cores, derivation of the remaining $3^k - 2^k$ frequencies, application of the scoring function and reduction, including identification of the set most correlated with phenotype, are tasks that are combined into the same kernel. In 3-way searches, the additional steps related to the calculation of genotype frequencies of pairwise combinations (two distinct phases) and the precombination of a block of SNPs with a given individual SNP are conducted by three additional GPU kernels.

The proposal has been implemented in a way that strives to optimize the use of the GPU memory subsystem, taking into account both bandwidth and latency. The reduction of scores relies on registers, shared memory and global memory, at the thread-level (each thread of the kernel processes multiple contingency tables), at the level of individual thread blocks and between thread blocks, respectively.

5.4 Objective Scoring Function

We rely on the Bayesian K2 score [30], [34] as the main objective function. The K2 score is widely used in the context of epistasis detection and can be defined as:

$$K2 = \sum_{i=1}^{I} \left(\sum_{b=1}^{r_i+1} \log(b) - \sum_{j=1}^{J} \sum_{d=1}^{r_{ij}} \log(d) \right). \tag{8}$$

The K2 score, as is the case for other objective functions, calculates a score based on the number of occurrences r_i of each of the I possible genotypes ($I = 3^k$) in samples, taking into account the frequency r_{ij} in samples pertaining to the specific considered J phenotypic states ($J = 2$).

Multiple strategies for optimizing the scoring of sets of SNPs have been devised and implemented in the scope of this paper. First, the proposal relies on the gamma function. This function widens the scope of the factorial function to complex numbers as per $\Gamma(n) = (n-1)!$. The `lgamma` intrinsic, which computes the natural logarithm of the gamma function, is used to calculate the sums of natural logarithms after translation to logarithms of factorial. In order to achieve maximum efficiency, the GPU kernel computing the scores accesses a table with precomputed `lgamma` values through read-only data cache (`__ldg()`). The proposal computes at the application start the `lgamma` values that might be required during a given epistasis run, based on the amount of cases and controls in the dataset. This step is overlapped with the transfer of a portion of the dataset to GPU memory.

Mutual information scoring [31], popular in the context of GWAS (e.g., [13], [15]), has also been implemented. This allows to experimentally show (see Section 6.3) that the proposal can operate efficiently with other scoring functions. As is the case with K2, mutual information also operates over the 3^k contingency table values pertaining to each phenotypic state. The implementation optimizations previously described for the former, including the use of a lookup table, are also applicable to the latter.

6 Experimental Results

This section presents the experimental methodology and the results obtained with the proposal when processing different datasets with different numbers of SNPs and samples, highlighting relevant parameters.

6.1 Experimental Setup

6.1.1 Target systems

The main target of the experimental evaluation is a workstation with an Intel Xeon E3 1245 V3 (Haswell @3.6 GHz) quad-core CPU, 16 Gigabytes of DDR3 and a NVIDIA GeForce 2070S graphics card (TU104-410-A1 @1,605/1,785 MHz). The CUDA 10.1 toolkit (V10.1.243), CUTLASS 1.3 and GCC 8.4 have been used on top of Ubuntu 20.04. Additional experiments have been conducted on a system with an AMD Ryzen 3600 (Zen 2 @4.2 GHz) six-core CPU, 32 Gigabytes of DDR4 and an NVIDIA Titan RTX graphics card (TU102-400-A1 @1,350/1,770 MHz). This system has been used for performing a final evaluation of the proposal, relying on the highest performing implementation.

The GeForce 2070S and the Titan RTX have 40 and 72 SMs, resulting in 2560 and 4608 CUDA cores (i.e., number of 32-bit integer and 32-bit floating-point scalar ALUs), and 320 and 576 tensor cores, respectively. Both are connected to the host through a PCI Express 3.0 bus with 16 lanes.

6.1.2 Performance Metric and Evaluated Parameters

The performance metric the proposal strives to maximize is the number of unique sets of SNPs evaluated per second scaled to the sample size. This metric allows comparing the throughput of different approaches, even when using datasets with different numbers of SNPs and/or samples. We rely on this metric for comparing the proposal to the related state-of-the-art. Tensor tera operations per second (TOPS) are also reported for selected runs, as this metric is representative of efficiency in usage of the GPU resources.

In order to fully analyse the performance attained by the proposal, we evaluate it under different conditions. We evaluate the benefit of the usage of the interleaved data representation, comparing the achieved performance with that of the usage of a non-interleaved representation (see Section 3.1). The impact of different block sizes of a fixed amount of SNPs (128, 256, 512, 1024 or 2048), as-well of using a variable block size strategy to achieve higher performance through improving the ratio of unique sets processed, are also evaluated. In addition, we evaluate the use of different numbers of CUDA streams (1, 2, 4 or 8) for concurrent execution of evaluation rounds. In comparison with 3-way searches, 2-way searches are expected to reach completion in orders of magnitude less time. Experiments pertaining to 2-way searches entailed 20 independent runs, as a measure to reduce the possible effect of other processes executing on the targeted system due to the small execution time of some searches. The herein presented results correspond to the median value for those runs.

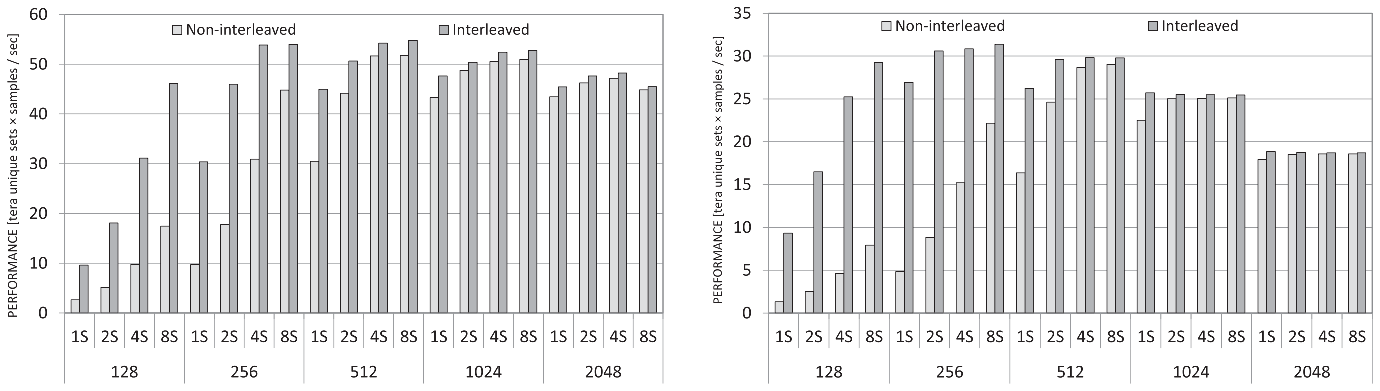
The experiments reported in this paper process synthetic datasets with 2048, 4096, 8192, 16384 or 32768 SNPs. In pairwise searches, these result in the evaluation of close to 2, 8, 34, 134 and 537 mega ($\times 10^6$) combinations, respectively. In triplet searches, these entail 1430, 11445, 91592, 732874 and 5863525 mega combinations, respectively. Different numbers of samples are also considered, through evaluation of datasets with 8192, 16384, 32768, 65536, 131072 or 262144 patient records (half cases/controls). Notice that although the data used in the experiments is synthetic, the specific values on the dataset do not affect the volume of operations to be performed by the proposal, thus making the reported performance representative of what one would achieve using data obtained through DNA sequencing.

6.2 Effect of Number of SNPs per Block and of Streams

Figs. 8a and 8b show the performance achieved in 2-way and 3-way searches, with a dataset with 8192 SNPs and 262144 samples, when blocks with different amounts of SNPs (128, 256, 512, 1024 or 2048) and of CUDA streams (1, 2, 4 or 8) processing evaluation rounds are adopted.

The use of multiple streams for processing the evaluation rounds enables concurrent execution of instances of the kernel that operates allelic data using tensor core operations and of the kernel code that derives the contingency table values, calculates the scoring function and reduces the scores. The use of multiple streams is particularly helpful when processing small blocks of SNPs, as it allows to significantly increase the utilization of the available GPU resources.

The interleaved representation achieved higher performance for any given block size and number of streams. More streams tends to diminish the difference between the performance achieved with interleaved and non-interleaved representations. Moreover, when using the interleaved representation, the advantage of using multiple streams is exhausted with fewer streams. It is often the case that using 4 or even only 2 streams already allows achieving performance that is very close to the one obtained with 8 streams. Doubling the SNPs per block results in the number of thread blocks per each individual kernel execution performing the matrix operations on a particular stream increasing by $4\times$, which is in line with the increase in computation per round. However, the number of thread blocks per grid is $4\times$ ($8\times$) larger in 2-way (3-way) searches with the interleaved approach, in relation to that with the non-interleaved approach. This allowed the interleaved approach to make,



(a) 2-way searches

(b) 3-way searches

Fig. 8. Number of unique pairs / triplets ($\times 10^{12}$) processed per second, scaled to the sample size, achieved for the use of different numbers of streams (1, 2, 4 or 8) and different numbers of SNPs per block (128, 256, 512, 1024, or 2048) in 2-way (pairs) and 3-way (triplets) searches on a system with a Xeon E3-1245 V3 CPU and a GeForce 2070S GPU, for a dataset with 8192 SNPs and 262144 samples.

TABLE 1
Ratio of Unique Sets for 2-Way and 3-Way Searches for Datasets With M SNPs and Blocks of B SNPs

k	2-way					3-way				
M / B	128	256	512	1024	2048	128	256	512	1024	2048
2048	0.941	0.888	0.800	0.666	0.500	0.835	0.710	0.533	0.333	0.166
4096	0.969	0.941	0.889	0.800	0.667	0.912	0.836	0.711	0.533	0.333
8192	0.984	0.970	0.941	0.889	0.800	0.954	0.912	0.836	0.711	0.533
16384	0.992	0.985	0.970	0.941	0.889	0.977	0.955	0.912	0.836	0.711
32768	0.996	0.992	0.985	0.970	0.941	0.988	0.977	0.955	0.913	0.837

for the same number of streams and number of SNPs per block, a more efficient use of the GPU resources. For example, using blocks of 256 SNPs, in 2-way (3-way) searches, with the interleaved approach there are 16 (32) thread blocks per kernel execution. Using the non-interleaved approach only 4 thread blocks are instantiated per grid in a given stream, both in 2-way or 3-way searches, which is not enough to fully saturate all available SMs (40) even if using 8 streams.

The grid sizes mentioned here result from the use of thread block and warp tile sizes of (1024,128,128) and (1024,32,64), respectively. This particular setting within CUTLASS has been experimentally determined to result in higher performance in comparison with a number of other tiling configurations; even in relation to those enabling higher SM occupancy than the one achieved (256 active threads, out of a maximum of 1024 in Turing). Notice that a compromise must be made when setting these parameters, since there are conflicting goals. Enlarging the thread block tile results in higher SM occupancy and has the potential to improve usage of the device global memory bandwidth. However, it also results in smaller grid sizes for a given amount of SNPs per block. Higher occupancy can also be achieved by reducing the warp tile, but doing so decreases data reuse within warps, resulting in a performance loss for any of the other smaller tile configurations tested.

The selection of a suitable block size for a given epistasis detection run is of utmost importance. Table 1 depicts the ratios of unique sets that are processed for 2-way and 3-way searches, which depends on the number of SNPs in a given dataset (M), and the number of SNPs per block (B). These ratios have a significant impact on the overall performance. Higher performance is achieved through striking a balance between efficient use of the GPU resources and a favorable ratio of unique sets (i.e., as close to 1 as possible). Especially when processing datasets with a relatively small amount of SNPs, the use of large block sizes results in processing a large amount of non-unique sets. For 8192 SNPs, in a 2-way search, the use of blocks of 128 SNPs results in a ratio of 0.984 unique sets, while using blocks of 2048 SNPs results in a ratio of 0.8 unique sets. In a 3-way search these numbers drop to 0.954 and 0.533, respectively. Notice that while a block of 128 SNPs results in a ratio of unique sets closer to 1 than any other amount of SNPs per block evaluated, using more SNPs per block has resulted in higher performance because of improved utilization of the GPU resources. It is also important to notice that for a given block size, the ratio of unique sets of SNPs processed is the same in both the interleaved and non-interleaved approaches. Thus, any increase in throughput processing sets of SNPs translates directly into performance improvements in epistasis searches.

In the 2-way search experiments, depicted in Fig. 8a, higher performance has been achieved with a block of 512 SNPs, for both memory representation approaches. However, when using the interleaved representation, a block size of 256 SNPs achieves similar performance. In contrast, when using the non-interleaved approach, a block of 256 SNPs is not sufficient to achieve the highest attainable performance. Such block size is too small for achieving full use of the GPU resources, even with 8 streams. As expected, using fewer streams, requires the use of larger block sizes. In the interleaved approach, if using only 1 stream, highest performance is achieved with a block of 1024 SNPs. If using the non-interleaved approach, similar performance is achieved with blocks of 1024 or 2048 SNPs.

Selecting a block size is especially important in 3-way searches, as the impact on the ratio of unique sets is more pronounced. The achieved performance drops sharply when using a block with more SNPs than required to

TABLE 2
Average Tera Operations Per Second (TOPS) Achieved at the Tensor Cores of the GeForce 2070S GPU in 2-way or 3-way Searches (8192 SNPs and 262144 Samples) With the Non-Interleaved or Interleaved Approaches, Using a Single or Multiple Streams (8) and Considering Blocks of B SNPs

k	2-way					3-way				
Implementation and # streams / B	128	256	512	1024	2048	128	256	512	1024	2048
Non-interleaved — 1 stream	21	80	259	389	435	22	85	313	507	538
Non-interleaved — 8 streams	142	370	440	459	449	133	389	555	566	558
Interleaved — 1 stream	78	251	382	429	454	157	472	502	579	566
Interleaved — 8 streams	375	446	466	475	455	490	550	570	573	561

efficiently use the GPU. In 3-way, depicted in Fig. 8b, for the non-interleaved approach, a block of 512 SNPs resulted in higher performance than when using a block of 256 SNPs. However, when relying on the interleaved approach, using a block of 256 SNPs led to higher performance. In addition, when using small blocks, such as that block size, the difference in performance between the non-interleaved and interleaved approaches is larger in 3-way than in 2-way searches. Notice that in 3-way searches, for the interleaved approach the first input to the matrix operations is $2\times$ larger than the second input, which results from the combination of a given a block of SNPs Y with two bit vectors (genotypes 0 and 1) of a given SNP X into a single matrix. In 3-way, for a block size of 256 SNPs, using the non-interleaved approach, both outer dimensions in the binarized matrix operations equal 256. With the interleaved approach, these become 1024 and 512, respectively. As happens in 2-way, when using a small amount of streams in 3-way searches, a larger block is often required. Using only 1 stream to perform the evaluation rounds, the highest performance is achieved with a block of 256 or 1024 SNPs, for the interleaved and the non-interleaved approaches, respectively.

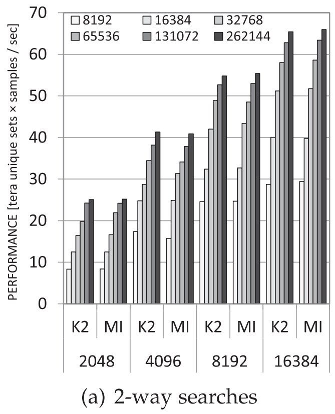
Table 2 shows the average TOPS achieved in the tensor cores, for 2-way and 3-way searches considering either 1 or 8 concurrent steams, for the non-interleaved or the interleaved representation and different block sizes. The TOPS metric reported herein counts fused XOR and population count as two operations. For the same parameters, the interleaved representation always achieved more Tensor TOPS. Increasing the block size up to 1024 SNPs always improved this metric. If using a single stream, blocks of 2048 SNPs further improved TOPS in 2-way searches. In 3-way, TOPS only increased further if using the non-interleaved representation. The interleaved approach achieves saturation of the GPU resources with a smaller block size due to larger inputs in the matrix operations. All other parameters being equal, Tensor TOPS tends to be higher in 3-way searches. This is largely due to a higher ratio of computation in relation to memory transfers. As expected, the use of multiple streams tends to result in higher TOPS. High throughput can be achieved for the largest block sizes using a single stream. However, achieving high throughput with small blocks of SNPs is required for highest detection performance, due to an improved ratio of unique sets.

Independent runs of 2-way (3-way) searches relying on the interleaved approach using 8 streams have been profiled. On instances considering, on top of data transfers, only the kernel performing the matrix operations, 381 (684), 457 (704), 484 (706), 497 (698) and 476 (693) TOPS were achieved for block sizes of 128, 256, 512, 1024 or 2048 SNPs. Comparing with the TOPS reported in Table 2, it is clear that the proposal is operating at close to the throughput of the kernel that performs the matrix operations using the tensor cores. Profiling the complete proposal in 2-way (3-way) searches reported a GPU utilization of 77.11 percent (76.42 percent) for the matrix-matrix operations, 12.27 percent (0.00 percent) for memory transfers, 5.47 percent (15.40 percent) for the kernel code that derives contingency tables, scores sets of SNPs and reduces scores, and 5.15 percent (0.00 percent) for counting genotype frequencies for individual SNPs. The remaining 8.18 percent in the 3-way search is attributed to the three additional kernels (see Section 4.3). In 3-way, the amount of combinations evaluated ($2730\times$ in relation to 2-way) makes the cost of transfers and of counting individual genotype frequencies negligible. Notice that for larger datasets, the matrix operations are expected to dominate even more over the other kernel codes.

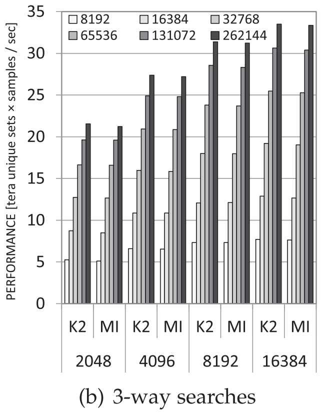
6.3 Scaling With the Number of Samples

The effect of the amount of samples in the performance is of a different nature than that of the number of SNPs. Fig. 9 depicts the results achieved for runs relying on K2 or mutual information scoring. These include the use of the interleaved representation, 8 streams and the number of SNPs per block that individually resulted in the highest performance, for datasets with different numbers of SNPs (2048, 4096, 8192 or 16384) and between 8192 and 262144 samples.

As expected, each successive doubling of the number of samples results in a less substantial relative performance improvement, as the use of the GPU resources approaches the maximum. In 2-way searches, when considering 2048 SNPs, increasing the number of samples from 8192 to 16384



(a) 2-way searches



(b) 3-way searches

Fig. 9. Performance scaling with number of samples (between 8192 and 262144) on system with a Xeon E3-1245 V3 CPU and a GeForce 2070S GPU, for datasets with different numbers of SNPs (2048, 4096, 8192, 16384), using 8 streams and block size that resulted in the highest performance. K2: K2 Bayesian scoring. MI: Mutual Information scoring.

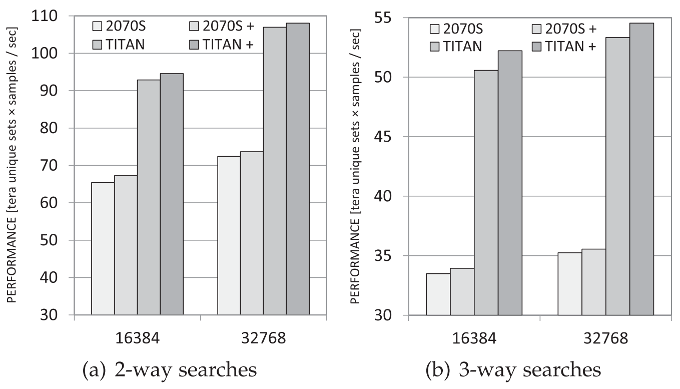


Fig. 10. Performance achieved on Xeon E3-1245 V3 + Geforce 2070S and Ryzen 3600 + Titan RTX with fixed or variable block size (+) for datasets with 16384 or 32768 SNPs and 262144 samples.

resulted in a $1.5\times$ improvement. For 16384 SNPs, performance improves by $1.39\times$. In 3-way searches, performance is more affected by the number of samples. Performance increases by $1.66\times$ and $1.68\times$ when going from 8192 to 16384 samples, for the datasets with 2048 and 16384 SNPs, respectively. As expected, highest performance is achieved with 262144 samples, both in 2-way and 3-way runs. The same pattern can be seen when dealing with datasets with different numbers of SNPs. Finally, identical performance has been achieved with both scoring functions, since both rely on the same calculation approach using a lookup table.

6.4 Tuning and Evaluation on Additional System

Fig. 10 shows the performance achieved on two systems with Turing GPUs, for 2-way and 3-way searches, datasets with 262144 samples, and 16384 or 32768 SNPs. These results are for a fixed block size of 512 SNPs, identified as the size that provides the highest performance, and for a variable block size strategy that relies on two block size halving steps (see Section 3.5). The latter strategy relied on a starting block size of 1024 (2-way) or 512 (3-way) SNPs.

The achieved performance improved in 3-way due to a significant increase of the ratio of unique sets, as a result of a considerable portion of the matrix operations being performed with small blocks (256 or 128 SNPs). In 2-way, the achieved improvement results from the combined benefit of part of the computations being performed with larger blocks of data (1024 SNPs) and a slight improvement of the ratio of unique sets processed. In 2-way and 3-way searches, for the largest dataset (32768 SNPs and 262144 samples), the system with a Titan RTX (GeForce 2070S) achieves 869 and 894 (588 and 591) Tensor TOPS using a fixed block size, while 871 and 898 (594 and 586) Tensor TOPS are achieved using the variable block size strategy, respectively.

Top performance on 2-way searches, shown in Fig 10a, improves by $1.41\times$ from the Geforce 2070S to the Titan RTX system on the dataset with 16384 SNPs. On the 32768 SNPs dataset, the improvement is $1.47\times$. On 3-way searches, the relative performance improvement from the Geforce 2070S to the Titan RTX system, shown in Fig 10b, is more substantial than in 2-way searches. In addition, between these systems, performance improves as much on the 16384 SNPs dataset ($1.54\times$) as on the 32768 SNPs dataset ($1.53\times$). In 3-way searches computation completely dominates over the cost of transferring data to the GPU. For the same reason, when analysing the systems individually, in relation to 2-way searches, the performance achieved processing 32768 SNPs is not significantly higher than that for 16384 SNPs.

The improvements result from the efficient use of the additional resources in the Titan RTX, which has 80 percent more SMs than the GeForce 2070S. Although they have the same advertised boost frequency (1770 MHz), the former has a more modest nominal frequency (1350 MHz) compared to the latter (1605 MHz). Given that the proposal exercises the GPU to a high degree and that the TDP of the Titan RTX is only 30 percent higher (280W versus 215W), more aggressive throttling is to be expected. As a matter of fact, the GeForce 2070S consistently achieved higher clock frequencies.

7 Related Work

Given the additional complexity of exhaustive 3-way searches, most approaches only tackle pairwise searches (e.g., [5], [6], [7], [8], [9], [11], [12], [14], [35]). The former are rarely tackled and often rely on multi-GPU systems (e.g., [13]) or supercomputers (e.g., [10], [15]). Non-exhaustive methods (e.g., [36], [37], [38], [39]) can achieve faster detection. However, that comes at the cost of decreased detection accuracy.

State-of-the-art approaches relying on exhaustive methods have targeted multicore CPUs (e.g., [5], [6]), specialized architectures in FPGAs (e.g., [7], [8]), GPUs (e.g., [5], [8], [9], [10], [11], [12], [13], [35]) and other parallel architectures, such as Intel Xeon Phi accelerators (e.g., [11], [14]). GPUs are especially suitable targets, because of their compute resources in relation to cost and the data-parallel nature of epistasis detection.

Table 3 shows the performance reported for a number of epistasis detection approaches that rely on exhaustive methods targeted at systems with accelerators. The development of algorithms and implementations enabling efficient exploitation of novel hardware features and accelerator devices for epistasis detection purposes has been one of the main performance drivers. Notice that compared with other approaches, the proposal herein presented achieves higher performance per node, considering both targeted systems. This is the case even in comparison with approaches that rely on the use of multiple GPUs per node.

The comparison with related art is focused on MPI3SNP [15] and CoMet [10]. These approaches have been recently published, representing the state-of-the-art in GPU-based epistasis detection approaches using exhaustive search methods. MPI3SNP has been compiled from source and executed on the same systems targeted by the proposal. In the case of CoMet, which is especially interesting because it uses tensor cores, although it targets a different precision and uses a different exploitation method, we relied on comparing the performance achieved by the proposal with that reported in the corresponding paper, taking into account that systems with different capabilities have been used.

MPI3SNP [15], an evolution of the GPU3SNP [13] multi-GPU tool for triplet searches, when evaluated in the target systems with the datasets available in the source code repository using mutual information scoring, achieved a maximum throughput of 426.2 and 681.5 giga ($\times 10^9$) triplets per second, scaled to the sample size, on the systems with the GeForce 2070S and the Titan RTX, respectively. On the

TABLE 3
Performance in Sets ($\times 10^{12}$) Processed Per Second Scaled to Number of Samples of Epistasis Detection Approaches That Target Hardware Accelerators Relying on 2-way and 3-way Exhaustive Search Methods

kth order	Approach	(# nodes×) Accelerator configuration	Performance	Performance / node
2-way	epiSNP [14]	(126×) 2 Intel Phi SE10P	1.593	0.013
	GBOOST [9]	(1×) 1 GeForce GTX 285	0.064	0.064
	multiEpistSearch [35]	(24×) 1 Titan	12.626	0.526
	GWIS [5], [40]	(1×) 1 GeForce GTX 470	0.658	0.658
	GWIS$_{FI}$ [12]	(1×) 1 GeForce GTX 470	0.767	0.767
	Wienbrandt *et al.* [8]	(1×) 1 Tesla P100 + 1 Kintex UltraScale KU115	0.941	0.941
	multiEpistSearch (Hybrid) [11]	(1×) 2 Tesla K20m + 1 Intel Phi 5110P	1.053	1.053
	CoMet [10]	(4560×) 6 Tesla V100	295633	64.831
	Proposal	(1×) 1 GeForce 2070S / 1 Titan RTX	73.69 / 108.08	**73.69 / 108.08**
3-way	GPU3SNP [13]	(1×) 4 Titan	0.265	0.265
	MPI3SNP [15]	(4×) 2 Tesla K80	0.555	0.139
	CoMet [10]	(4373×) 6 Tesla V100	81611	18.658
	Proposal	(1×) 1 GeForce 2070S / 1 Titan RTX	35.56 / 54.54	**35.56 / 54.54**

same systems, the proposal in this paper has achieved an improvement in throughput of $83.5\times$ and $80\times$ in 3-way searches, on the GeForce 2070S and the Titan RTX, respectively. MPI3SNP relies on AND and population count instructions to construct contingency tables. There is reuse of intermediate results resulting from execution of the former instruction, but the searches are still performed on top of a 3-bit per SNP per sample representation. The performance improvement in comparison to MPI3SNP can be attributed to the efficient use of the highest throughput operation on the novel GPU tensor cores, the use of an optimized calculation strategy and the techniques devised to achieve higher overall efficiency when using the GPU. As a result, the GPUs were exercised to a significantly higher extent in comparison to when executing MPI3SNP.

CoMet [10] has been evaluated on the Summit supercomputer [41], targeting thousands of nodes with six Tesla V100 (Volta architecture [42]) each with up to 98 percent weak scaling efficiency. CoMet relies on a binarized representation where the allele types are represented by the number of set bits, but it requires mapping to a different type of representation for the searching process (as it incurs in a $16\times$ memory expansion) in order to use the Volta tensor cores. Each entry of two bits representing an allele becomes represented by counts of 0's and 1's, two 16-bit floating point values. For 2-way and 3-way searches, up to 10.805×10^{12} and 3.111×10^{12} evaluations of sets of SNPs per second per GPU scaled to the sample size have been reported. Thus, compared with CoMet, the proposal achieves improvement factors of $6.8\times$ and $11.4\times$, on the system with the GeForce 2070S, in 2-way and 3-way searches, respectively. On the system with the Titan RTX, the improvement factors are of $10\times$ and $17.5\times$, respectively. The proposal in this paper can accomplish 2-way and 3-way searches with comparable efficiency, achieving similar Tensor TOPS with challenging datasets. The performance improvement in relation to CoMet is considerably larger in 3-way searches because triplet searches incur in close to a $2\times$ reduction in the number of sets processed per second in relation to that achieved in 2-way searches. This is consistent with the use of the calculation reduction technique presented in Sections 3.4 (2-way) and 4.2 (3-way). In contrast, the performance reported for CoMet in 3-way searches is close to $3\times$ less than that in 2-way searches, which is in line with the fact that the approach requires $3\times$ the number of matrix-matrix operations for processing each plane of triplets of SNPs in a 3-way search, in comparison to evaluation of the pairs of SNPs in a 2-way search. Notice that although the CoMet approach is based on vector similarity searching using the Custom Correlation Coefficient [43] statistical measure, the volume and the kind of computation (i.e., evaluating all-to-all pairs/triplets of vectors of SNP allelic data) is directly comparable.

In [10], GPU tensor cores are used in 16-bit floating-point operation relying on standard half-precision cuBLAS GEMM calls (`cublasGemmEx`). In contrast, the proposed approach targets tensor cores relying on matrix operations structurally similar to GEMM, with fused XOR and population count replacing multiply-and-add. Moreover, given that the tensor cores in the targeted hardware (Turing GPUs) only support this operation when working at the level of single bits, we had to go through the additional step of finding an efficient way to leverage the highest throughput operation. This allows the proposal to achieve a significantly higher throughput than possible when operating 16-bit floating-point data or any other datatype or data-with. In relation to implementation, the kernel that uses the tensor cores has been implemented relying on CUTLASS [33], a set of CUDA templates for linear algebra subroutines, instead of relying on the cuBLAS [44] library. At present time, to the best of our knowledge, cuBLAS does not provide a means to perform GEMM-like computations in 1-bit precision.

It is important to emphasize that the performance achieved is not simply due to using a more powerful GPU. In standard 32-/16-bit floating point precision using the CUDA cores and in 16-bit floating point precision using the tensor cores, the Tesla V100 (15.67/31.33 CUDA and 125 Tensor TFLOPS) is overall comparable to the Titan RTX (16.31/32.62 CUDA and 130 Tensor TFLOPS) and significantly faster than the GeForce 2070S (9.06/18.12 CUDA and 73 Tensor TFLOPS). Memory bandwidth is $1.33\times$ higher in the Tesla V100 (897 GB/s) in relation to that of the Titan RTX (672 GB/s). Thus, it is expected that the proposal would still achieve higher performance compared with that achieved through execution of CoMet on a Turing GPU.

Fully exploiting GPUs with the Turing architecture requires a different solution design than if targeting Volta GPUs. The proposal has been fully designed around the use of fused XOR and population count operations, a novel capability in modern tensor cores. Notice that this capability, at present time not leveraged by most applications, will likely remain in future architectures because of its usefulness in the context of processing BNNs. Combining the exploitation of the highest throughput operations on a modern accelerator architecture, the use of algorithmic optimizations, a carefully devised orchestration strategy and meticulously optimized GPU code allows the proposal to significantly outperform other state-of-the-art approaches.

Compared to [45] and [46], two approaches for high-order epistasis detection from the authors of this paper, the proposal herein presented achieves significantly higher performance when executed on the same CPU+GPU system (Xeon E3-1245 V3 and GeForce 2070S). Compared with the former, which is a highly optimized 3-way detection approach that relies only on general-purpose GPU cores (useful for GPU architectures without tensor cores), the proposal achieves up to $29.73\times$ improved performance. This is due to the combined use of the tensor cores and an improved calculation strategy. The latter approach represents a preliminary iteration of the proposal presented in this paper. The algorithms for 2-way and 3-way searches and their implementations have been extensively modified in order to allow for an increased usage of the GPU compute resources, achieved through the use of the interleaved internal data representation and a more efficient overlapping of GPU kernel execution. As a result, processing the same data becomes significantly faster. For example, a dataset with 16384 SNPs and 131072 samples, which has been used in the evaluation of both iterations of the proposal, is processed $1.3\times$ and $1.58\times$ faster in 2-way and 3-way searches, respectively.

8 Conclusion

The proposal presented in this paper accelerates epistasis detection searches in case-control datasets relying on fused XOR and population count as part of matrix operations. These operations have the highest throughput on some current processing devices, such as Turing GPUs. The proposal achieved a performance of up to 108.1 and 54.5 tera ($\times 10^{12}$) combinations processed per second, scaled to the number of samples, on a system with a NVIDIA Titan RTX GPU, in 2-way and 3-way epistasis detection searches, respectively. This represents, per GPU, performance of an order of magnitude higher than the highest performing related art approach.

The proposal in this paper computes genotype frequency tables faster than any other approach. In addition to K2 Bayesian and mutual information scoring, two functions considered in this paper, these tables are used by a number of other objective scoring functions. This makes the proposal useful for other challenging bioinformatics problems.

Ongoing work includes adding support for exploiting the usage of multiple computer nodes through MPI and accelerating epistasis detection relying on additional classes of devices, such as FPGAs and accelerators specialized to neural network training/inferencing.

Acknowledgments

This work was supported in part by the FCT (Fundação para a Ciência e a Tecnologia, Portugal) and in part by 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).

References

[1] C. Niel *et al.*, "A survey about methods dedicated to epistasis detection," *Front. Genetics*, vol. 6, pp. 1–19, Sep. 2015.

[2] N. Cullell *et al.*, "Pharmacogenetic studies with oral anticoagulants. Genome-wide association studies in vitamin k antagonist and direct oral anticoagulants," *Oncotarget*, vol. 9, no. 49, pp. 29 238–29 258, Jun. 2018.

[3] E. Poṡpiech *et al.*, "Towards broadening forensic DNA phenotyping beyond pigmentation: Improving the prediction of head hair shape from DNA," *Forensic Sci. Int., Genetics*, vol. 37, pp. 241–251, 2018.

[4] T. F. Mackay and J. H. Moore, "Why epistasis is important for tackling complex human disease genetics," *Genome Medicine*, vol. 6, no. 6:42, pp. 1–3, 2014.

[5] B. Goudey *et al.*, "High performance computing enabling exhaustive analysis of higher order single nucleotide polymorphism interaction in genome wide association studies," *Health Inf. Sci. Syst.*, vol. 3, pp. 1–11, Feb. 2015.

[6] J. C. Kässens, J. González-Domínguez, L. Wienbrandt, and B. Schmidt, "UPC++ for bioinformatics: A case study using genome-wide association studies," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2014, pp. 248–256.

[7] L. Wienbrandt *et al.*, "FPGA-based acceleration of detecting statistical epistasis in GWAS," *Procedia Comput. Sci.*, vol. 29, pp. 220–230, 2014.

[8] L. Wienbrandt, J. C. Kässens, M. Hübenthal, and D. Ellinghaus, "1000× faster than PLINK: Combined FPGA and GPU accelerators for logistic regression-based detection of epistasis," *J. Comput. Sci.*, vol. 30, pp. 183–193, 2019.

[9] L. S. Yung, C. Yang, X. Wan, and W. Yu, "GBOOST," *Bioinformatics*, vol. 27, no. 9, pp. 1309–1310, May 2011.

[10] W. Joubert *et al.*, "Attacking the opioid epidemic: Determining the epistatic and pleiotropic genetic architectures for chronic pain and opioid addiction," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2018, pp. 57:1–57:14.

[11] J. González-Domínguez, S. Ramos, J. Touriño, and B. Schmidt, "Parallel pairwise epistasis detection on heterogeneous computing architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2329–2340, Aug. 2016.

[12] Q. Wang *et al.*, "GWISFI: A universal GPU interface for exhaustive search of pairwise interactions in case-control GWAS in minutes," in *Proc. IEEE Int. Conf. Bioinf. Biomedicine*, 2014, pp. 403–409.

[13] J. González-Domínguez and B. Schmidt, "GPU-accelerated exhaustive search for third-order epistatic interactions in case–control studies," *J. Comput. Sci.*, vol. 8, pp. 93–100, 2015.

[14] G. R. Luecke *et al.*, "Fast epistasis detection in large-scale GWAS for Intel Xeon Phi clusters," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, 2015, pp. 228–235.

[15] C. Ponte-Fernández, J. González-Domínguez, and M. J. Martín, "Fast search of third-order epistatic interactions on CPU and GPU clusters," *Int. J. High Perform. Comput. Appl.*, vol. 34, no. 1, pp. 20–29, 2020.

[16] J. Hennessy and D. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, pp. 48–60, 2019.

[17] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 1–12.

[18] NVIDIA Corporation, "NVIDIA Turing GPU architecture whitepaper," Aug. 2020. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[19] E. Medina and E. Dagan, "Habana Labs purpose-built AI inference and training processor architectures: Scaling AI training systems using standard ethernet with Gaudi processor," *IEEE Micro*, vol. 40, no. 2, pp. 17–24, Mar./Apr. 2020.

[20] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf. Digest Tech. Papers*, 2014, pp. 10–14.
[21] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.
[22] Y.-M. Qian and X. Xiang, "Binary neural networks for speech recognition," *Front. Inf. Technol. Electron. Eng.*, vol. 20, no. 5, pp. 701–715, May 2019.
[23] N. Vasilache *et al.*, "The next 700 accelerated layers: From mathematical expressions of network computation graphs to accelerated GPU kernels, automatically," *ACM Trans. Architecture Code Optim.*, vol. 16, no. 4, pp. 1–26, Oct. 2019.
[24] A. Rasch, R. Schulze, and S. Gorlatch, "Generating portable high-performance code via multi-dimensional homomorphisms," in *Proc. 28th Int. Conf. Parallel Architectures Compilation Techn.*, 2019, pp. 354–369.
[25] T. Chen *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. 13th USENIX Symp. Operating Syst. Des. Implementation*, 2018, pp. 578–594.
[26] V. Elango *et al.*, "Diesel: DSL for linear algebra and neural net computations on GPUs," in *Proc. 2nd ACM SIGPLAN Int. Workshop Mach. Learn. Program. Languages*, 2018, Art. no. 42–51.
[27] M. D. Ritchie, "Finding the epistasis needles in the genome-wide haystack," in *Epistasis, Methods in Molecular Biology*. Berlin, Germany: Springer, 2014, pp. 19–33.
[28] D. H. Stamatis, *Essential Statistical Concepts for the Quality Professional*. Boca Raton, FL, USA: CRC Press, 2012.
[29] C. Gini, *Variabilità e Mutabilità*. Rome: Libreria Eredi Virgilio Veschi, 1912.
[30] G. F. Cooper and E. Herskovits, "A Bayesian method for the induction of probabilistic networks from data," *Mach. Learn.*, vol. 9, no. 4, pp. 309–347, Oct. 1992.
[31] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York, NY, USA: Wiley, 2006.
[32] X. Wan *et al.*, "BOOST: A fast approach to detecting gene-gene interactions in genome-wide case-control studies," *Amer. J. Human Genetics*, vol. 87, no. 3, pp. 325–340, Sep. 2010.
[33] NVIDIA Corporation. CUTLASS: CUDA templates for linear algebra subroutines," Accessed: Oct. 7, 2020. [Online]. Available: https://github.com/NVIDIA/cutlass
[34] Y. Sun *et al.*, "epiACO - A method for identifying epistasis based on ant colony optimization algorithm," *BioData Mining*, vol. 10:23, pp. 1–17, Jul. 2017.
[35] J. González-Domínguez, J. C. Kässens, L. Wienbrandt, and B. Schmidt, "Large-scale genome-wide association studies on a GPU cluster using a CUDA-accelerated PGAS programming model," *Int. J. High Perform. Comput. Appl.*, vol. 29, no. 4, pp. 506–510, 2015.
[36] X. Sun *et al.*, "Analysis pipeline for the epistasis search – statistical versus biological filtering," *Front. Genetics*, vol. 5, pp. 1–17, 2014.
[37] Y. Guo *et al.*, "Epi-GTBN: An approach of epistasis mining based on genetic Tabu algorithm and Bayesian network," *BMC Bioinf.*, vol. 20, no. 1, pp. 444:1–444:18, 2019.
[38] J. Shang *et al.*, "A review of ant colony optimization based methods for detecting epistatic interactions," *IEEE Access*, vol. 7, pp. 13 497–13 509, 2019.
[39] R. Jiang, W. Tang, X. Wu, and W. Fu, "A random forest approach to the detection of epistatic interactions in case-control studies," *BMC Bioinf.*, vol. 10, pp. 1–12, 2009.
[40] B. Goudey *et al.*, "GWIS–model-free, fast and exhaustive search for epistatic interactions in case-control GWAS," *BMC Genomics*, vol. 14, pp. 1–18, 2013.
[41] J. A. Kahle, J. Moreno, and D. Dreps, "2.1 Summit and Sierra: Designing AI/HPC supercomputers," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2019, pp. 42–43.
[42] NVIDIA Corporation, "NVIDIA Tesla V100 GPU architecture," Aug. 2020. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf
[43] S. Climer, W. Yang, L. Fuentes, V. Davila-Roman, and C. C. Gu, "A custom correlation coefficient (CCC) approach for fast identification of multi-SNP association patterns in genome-wide SNPs data," *Genetic Epidemiol.*, vol. 38, no. 7, pp. 610–621, 2014.
[44] NVIDIA Corporation, "cuBLAS: Dense linear algebra on GPUs," Accessed: Oct. 7, 2020. [Online]. Available: https://developer.nvidia.com/cublas
[45] R. Nobre, S. Santander-Jim énez, L. Sousa, and A. Ilic, "Accelerating 3-way epistasis detection with CPU+GPU processing," in *Proc. 23rd Workshop Job Scheduling Strategies Parallel Process.*, 2020, pp. 106–126.
[46] R. Nobre, A. Ilic, S. Santander-Jim énez, and L. Sousa, "Exploring the binary precision capabilities of tensor cores for epistasis detection," in *Proc. Int. Parallel Distrib. Process. Symp.*, 2020, pp. 338–347.

Ricardo Nobre (Member, IEEE) received the PhD degree in informatics engineering from Faculdade de Engenharia da Universidade do Porto (FEUP), Porto, Portugal, in 2017. He is currently a researcher at Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID), Lisbon, Portugal. His interests include high-performance computing, compilers, parallel programming, and machine learning. He has contributed close to 20 papers in international journals and conferences.

Aleksandar Ilic (Member, IEEE) received the PhD degree in electrical and computer engineering from Instituto Superior Técnico (IST), Universidade de Lisboa, Portugal, in 2014. He is currently an assistant professor with the Department of Electrical and Computer Engineering, IST, and a senior researcher with the Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID), Lisbon, Portugal. His research interests include high-performance and energy-efficient computing, and modeling on parallel heterogeneous systems. He has contributed to more than 60 papers in international journals and conferences.

Sergio Santander-Jiménez (Member, IEEE) received the PhD degree in computer engineering from the University of Extremadura (UEx), Spain, in 2016. He is currently an assistant professor with the Department of Computer and Communications Technologies, UEx, and a senior researcher with the R&D Instituto de Engenharia de Sistemas e Computadores (INESC-ID). He has authored or co-authored more than 60 publications, edited three special issues and served as a reviewer for more than 30 JCR-indexed journals. His main research interests include evolutionary computation, multi-objective optimization, parallel and distributed computing, and computational biology.

Leonel Sousa (Member, IEEE) received the PhD degree in electrical and computer engineering from the Instituto Superior Técnico (IST), Universidade de Lisboa (UL), Lisbon, Portugal, in 1996. He is currently a full professor at UL and a senior researcher at the R&D Instituto de Engenharia de Sistemas e Computadores (INESC-ID). He has authored or coauthored more than 250 papers in journals and international conferences, and has edited four special issues of international journals. His research interests include VLSI architectures, computer architectures and arithmetic, parallel computing, and signal processing systems. He is fellow of IET and distinguished scientist of ACM.