Tensor-Accelerated Fourth-Order Epistasis Detection on GPUs

Ricardo Nobre
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
ricardo.nobre@inesc-id.pt

Aleksandar Ilic
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
aleksandar.ilic@inesc-id.pt

Sergio Santander-Jiménez
University of Extremadura (UNEX)
Cáceres, Spain
sesaji@unex.es

Leonel Sousa
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
leonel.sousa@inesc-id.pt

ABSTRACT

The improved accessibility of gene sequencing technologies has led to creation of huge datasets, i.e. patient records related to certain human diseases (phenotypes). Hence, deriving fast and accurate algorithms for efficiently processing these datasets is a paramount concern to enable some key healthcare scenarios, such as personalizing treatments, explaining the occurrence of and/or susceptibility to complex conditions and reducing the spread of infectious diseases. This is especially true for high-order epistasis detection, one of the most computationally challenging problems in bioinformatics, where associations between a given phenotype and single nucleotide polymorphisms (SNPs) of a population can often only be uncovered through evaluation of a large number of SNP combinations. To tackle this challenge, we propose a novel fourth-order epistasis detection algorithm that leverages tensor processing capabilities of two distinct accelerator architectures by efficiently mapping core computations related to processing quads of SNPs to binary tensor-accelerated matrix operations. Experimental results show that the proposed approach delivers very high performance even in single-GPU environments, e.g., 27.8 and 90.9 tera quads of SNPs per second, scaled to the sample size, were processed on Titan RTX (Turing) and A100 (Ampere) PCIe GPUs, respectively. Being the first approach that exploits tensor cores for accelerating searches with interaction order above three, the proposed method achieved a performance of up to 835.4 tera quads of SNPs per second on the 8-GPU HGX A100 server, which represents performance two or more orders of magnitude higher than that of related art.

1 INTRODUCTION

Searching for correlations between genetic markers and phenotype (i.e. the expression of genes) has high utility for several applications related to medicine and biology, including explaining susceptibility to diseases, personalized treatment of complex conditions and to reduce the spread of infectious diseases [8, 9, 22]. Single nucleotide polymorphisms (SNPs) are commonly used genetic markers representing variations of a nucleotide in the genome of a population. Since SNPs can interact in regard to phenotype, a phenomenon known as epistasis, it is crucial to consider multiple at a time [13].

Depending on the interaction order one strives to tackle, exhaustive epistasis detection can be very computationally challenging, especially if large amounts of samples are considered in a given search. Exhaustive epistasis detection, given its data-parallel nature, has been a target of Graphical Processing Unit (GPU) acceleration [3, 5–7, 10, 14, 16, 20, 25, 26]. Modern GPU architectures include hardware specialized for accelerating tensor computations. This hardware, often marketed for neural network processing, is specialized in performing floating-point and/or integer matrix multiplication and other related operations at various levels of arithmetic precision. Approaches that make use of these hardware units beyond neural networks are rare in the literature. The first work using tensor cores for epistasis detection [10] targeted the Volta microarchitecture, making extensive use of accelerated half precision matrix multiplication operations. Superior performance has been achieved through mapping core epistasis detection calculations to fused XOR and population count operations (herein referred to as XOR+POPC) [14, 16] introduced with the Turing microarchitecture [19]. For second and third order exploration, these works achieve performance that is significantly superior than possible if relying exclusively on the traditional datapath in GPUs.

Considering higher interaction orders is important, since it might allow uncovering additional associations with phenotype. In fact, complex diseases like Alzheimer's are associated to fourth-order interactions [22]. However, approaches focused on high-performance fourth-order exhaustive epistasis detection searches are rare and most do not make use of GPU accelerators (e.g. [2, 21]). Recently, a fourth-order exhaustive search method was proposed for CPUs and GPUs [15], but since it does not make use of tensor cores, a large portion of the modern GPU performance potential is unused. Achieving efficient tensor-based acceleration as part of fourth-order searches requires novel specialized algorithms, in order to allow for efficient mapping of the required computations on tensor hardware.

In this paper we propose an approach[1] specialized for high throughput fourth-order searches that targets challenging datasets. The proposed approach, called *Epi4Tensor*, relies on unconventional parallel computing hardware, tensor cores, to perform the type of computations that are characteristic of exhaustive epistasis detection searches. The contributions of the paper are the following:

(1) an innovative algorithm for efficient epistasis detection at fourth interaction order that relies on tensor-based binarized processing;

[1]Source code repository with implementation and sample datasets available at: https://github.com/hiperbio/Epi4Tensor

(2) novel strategies to exploit tensor hardware with different capabilities in the context of epistasis detection on single- and multi-GPU systems;
(3) experimental evaluation and analysis of the epistasis detection performance and tensor throughput achieved performing fourth-order searches on Turing and Ampere GPUs;
(4) quantitative evaluation of performance with regard to state-of-the-art methods.

Contribution 1) is related to the proposal of an approach that tackles fourth interaction-order searches using tensor-based bit-level computations. Contribution 2) pertains with presenting how core computations related to epistasis detection have been efficiently mapped to hardware with different levels of binary tensor-processing capabilities, in the context of single or multi-GPU systems, striving to achieve high computation efficiency and hardware interoperability. Contribution 3) is related to the experimental campaign undertaken to evaluate the proposed approach, which considered different GPU microarchitectures (Turing and Ampere). Contribution 4) compares the proposed approach with other methods focused on epistasis detection performance using modern parallel computing devices through specialized representations and algorithms. To the best of our knowledge, the algorithm proposed in this paper, implemented relying on CUDA and on the CUTLASS and OpenMP libraries, is the first to address fourth-order epistasis detection searches using tensor cores. It is also worthwhile to note that none of the existing epistasis detection tensor-based approaches (second and third order) explore the use of fused AND and population count operations (herein referred to as AND+POPC), introduced in the recently launched Ampere microarchitecture [17].

This paper is organized as follows: Section 2 formulates the problem, introducing important concepts on exhaustive epistasis detection. Section 3 proposes the novel algorithm for fourth-order detection. Section 4 is focused on the experimental campaign undertaken to evaluate the proposed approach, including a detailed analysis of the obtained experimental results. Section 5 introduces related work in epistasis detection, especially in the context of exhaustive fourth-order searches, comparing the proposed approach to related art. Finally, Section 6 concludes the paper.

2 PROBLEM FORMULATION

Given a dataset D with M SNPs and N samples, an epistasis detection search performed at fourth interaction-order consists on finding which combination of four SNPs (from the M SNPs) is most statistically correlated with a given phenotype, taking into account the genotype combination presented in the SNPs of each of the N samples. In case-control datasets, the particular type of datasets that has been considered in this work, the phenotype can be only of one of two possible types (control or case).

Most human SNPs are biallelic, i.e. there are two allele variants [27]. Individuals inherit one allele from the mother and one from the father, resulting in three possible genetic configurations for any given SNP: the homozygous major (AA), the heterozygous (Aa) and the homozygous minor (aa). Thus, an increase in the interaction order results in a 3× expansion in regard to the possible genotypes (3^k) for a given interaction between k SNPs. Performing an exhaustive fourth-order epistasis detection search consists on,

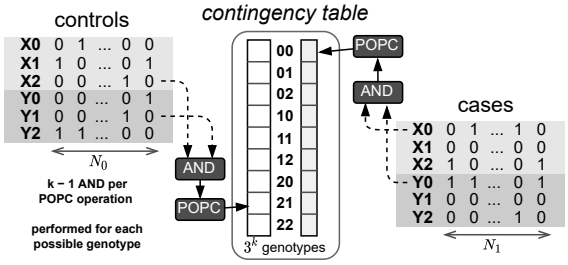

Figure 1: Example of contingency table construction pertaining to a second-order interaction (i.e. $k = 2$) relying on a bitwise representation and methods.

for any given combination of four SNPs, determining the number of occurrences of each of the 81 (= 3^4) genotypes and calculating a score representing predictive strength. The set of the occurrences of all the possible genotypes, for cases and controls (i.e. 162 values), is typically referred to as a *contingency* or *frequency* table.

Besides the impact of the considered interaction order (k) on the number of genotypes that have to be taken into account, there is also significant growth in the number of combinations of SNPs to evaluate. Since, in an exhaustive search, contingency tables and scores are to be calculated for all valid combinations of SNPs, the amount of combinations that need to be evaluated is equal to the binomial coefficient (with k equal to 4), i.e. given by $\frac{M!}{k!(M-k)!}$ for an input dataset with M SNPs. As an example, for a dataset with 1000 SNPs, there are 249.25× more combinations to process in fourth-order (41417.1 million) than in third-order (166.2 million).

For illustration purposes, Fig. 1 exemplifies the construction of a contingency table. For a second-order interaction between SNPs X and Y, with a bitwise representation and processing method, using AND and population count (POPC) operations, as first described for pair-wise searches in [24]. To enable this type of processing, the dataset is separated into bitvectors (three per SNP, per phenotype) of length N_0 (controls) or N_1 (cases). Bitvectors X_1, X_2 and X_3 (Y_1, Y_2 and Y_3) represent SNP X (SNP Y). The set bits represent samples that have the genetic variant the bitvector represents.

Different statistical tests can weight the particular characteristics of the input dataset differently in regard to assessing correlation with phenotype. We rely on the Bayesian K2 score function [4, 23] in the experiments reported in this paper. In the context of a case-control dataset (i.e. two phenotype states), given a contingency table pertaining to a fourth-order interaction (81 possible genotype combinations), the K2 score is calculated as follows.

$$K2 = \sum_{i=1}^{81} \left(\sum_{b=1}^{r_i+1} \log(b) - \sum_{j=1}^{2} \sum_{d=1}^{r_{ij}} \log(d) \right), \quad (1)$$

where r_i represents the number of occurrences of the ith genotype out of the 81 genotypes, and r_{ij} is the amount (out of the former) that is associated to phenotype j (case or control). Note that, given its efficient implementation, the particular statistical test used is not expected to have a significant impact on the overall performance when processing challenging datasets, since its cost is invariant in relation to the number of samples to process.

3 TENSOR ACCELERATED FOURTH-ORDER EPISTASIS DETECTION SEARCHES

This section presents the proposed approach and algorithm, explaining the design choices undertaken. This includes how it exploits state-of-the-art accelerator hardware relying on specialized algorithms as a means to achieve breakthrough performance-levels in fourth-order exhaustive epistasis detection searches.

3.1 Data Representation in Memory

Genotypic data is split into cases and controls, i.e. allelic data of all samples of a given type (cases or controls) is stored in a contiguous memory region, relying on a binary representation format that is a specialization of the dataset representation first used in BOOST [24]. In the adopted format, each SNP is represented by two bitvectors (instead of three) per phenotype (control or case), for the homozygous major (AA) and the heterozygous (Aa) genetic configurations. The i-th bit is set in the first bitvector pertaining to a given SNP and to a given phenotype state (control or case) if the i-th sample has the AA genotype in regard to that SNP. If it has the Aa genotype, then the i-th bit is set in the second bitvector. The aa genetic configuration is not represented in memory, being contingency tables constructed relying only on information about the other two genetic configurations, as explained in Section 3.3.

Summarizing, pairs of rows represent genotypic information of SNPs and columns represent samples (controls or cases), being the genetic configuration g (0 or 1) of a given SNP m represented in a bit-vector in a row with index $2 \times m + g$. Thus, being N_0 and N_1 the number of controls and cases, the input dataset is represented as two separate matrices, one with $2 \times M \times N_0$ bits representing controls and other with $2 \times M \times N_1$ bits for cases.

3.2 Efficient algorithm for fourth-order searches

Tensor cores in GPUs, and other matrix processing accelerators, are designed to perform matrix multiplication and related operations. As such, in order to use these specialized processing units, computationally intensive portions of the problem under study have to be represented as matrix operations that are supported in hardware.

Extrapolating from the example shown in Fig. 1, the number of occurrences of a given genotype $\{W_a, X_b, Y_c, Z_d\}$ (one out of the 81 genotypes) for a fourth-order interaction, for cases or controls, can be counted using efficient bitwise operations. It can be achieved performing the bitwise AND between the bitvectors representing genotype a, b, c and d for SNPs W, X, Y and Z, followed by counting the number of resulting 1's. This entails 3 bitwise AND and a POPC operation per sample, per genotype to process. Notice that we mentioned *operations*, since we are referring to individual samples. In practise, these are processed in packs of bits, depending the total number of instructions executed on the width of the particular instruction used on the targeted hardware.

The method proposed in this paper processes sets of SNPs through combining blocks of SNPs, following a particular combination scheme for obtaining the contingency table values that suits fourth-order epistasis detection searches. The evaluation of sets of SNPs, i.e. the construction of contingency tables (accelerated using tensor cores) and the calculation and reduction of scores are performed relying on multiple evaluation rounds. Each evaluation round combines four blocks of data from SNPs that are contiguous in the input dataset. A block of SNPs W is precombined with a block of SNPs X, relying on general-purpose GPU cores. The outcome matrix is processed with matrices resulting from combining a block of SNPs Y with a block of SNPs Z. Thus, there is reuse of the bitwise operations that combine a block of SNPs W with a block of SNPs X. Algorithm 1 presents the pseudocode for the proposed approach, as seen at the host side interacting with an accelerator device.

Data: h_0, h_1, M, N_0, N_1, B
Result: s
$hostToDevice(h_{0|1}, d_{0|1})$;
$pop_{0|1} = indivPop(d_{0|1})$;
$popPairs_{0|1} = pairwPop(d_{0|1})$;
for $W_i = 0;\ W_i < M;\ W_i += B$ **do**
  for $X_i = W_i;\ X_i < M;\ X_i += B$ **do**
    $wx_{0|1} = combine(d_{0|1}, W_i, X_i)$;
    $ctab\_wxy_{0|1} = tensorOp\_3way(wx_{0|1}, d_{0|1}, X_i)$;
    for $Y_i = X_i;\ Y_i < M;\ Y_i += B$ **do**
      $wy_{0|1} = combine(d_{0|1}, W_i, Y_i)$;
      $xy_{0|1} = combine(d_{0|1}, X_i, Y_i)$;
      $ctab\_wyz_{0|1} = tensorOp\_3way(wy_{0|1}, d_{0|1}, Y_i)$;
      $ctab\_xyz_{0|1} = tensorOp\_3way(xy_{0|1}, d_{0|1}, Y_i)$;
      for $Z_i = Y_i;\ Z_i < M;\ Z_i += B$ **do**
        $yz_{0|1} = combine(d_{0|1}, Y_i, Z_i)$;
        $ctab\_wxyz_{0|1} = tensorOp\_4way(wx_{0|1},$
         $yz_{0|1})$;
        $scores =$
         $applyScore(ctab\_wxy_0, ctab\_wxy_1,$
         $ctab\_wyz_0, ctab\_wyz_1, ctab\_xyz_0,$
         $ctab\_xyz_1, ctab\_wxyz_0, ctab\_wxyz_1,$
         $pop_{0|1}, popPairs_{0|1})$;
        $s = findGloballyBestSol(scores, s)$;
      end
    end
  end
end

Algorithm 1: Pseudocode of the proposed algorithm. Variables M, $N_{0|1}$ and B, which are used in most routines, are omitted in this pseudocode due to space constraints.

The inputs to the algorithm are: matrices representing genotypic data for controls (h_0) and cases (h_1), the number of SNPs in the dataset (M), the number of controls (N_0) and cases (N_1), and the number of SNPs to consider when combining SNPs (B), herein referred to as *block size*. The output of the algorithm is solution (s), i.e. the most statistically associated quad of SNPs and corresponding score. Repeated routine calls for controls (0) and cases (1), processed separately, are represented in the pseudocode with variables subscripted with 0|1. Conversely, in the input to `applyScore`, $ctab\_wxyz_0$ and $ctab\_wxyz_1$ are separately represented in order to convey there is a single call with both variables. If the number of SNPs is not a multiple of B, then the dataset is padded.

Host to device memory transfers send the dataset from the host to the accelerator device(s). After termination of the search, the best found solution is sent from the device(s) to the host. Notice that given the fact the proposed approach is targeted at processing complex datasets, due to the nature of the problem, memory transfers are expected to be a negligible part of the execution time.

General purpose cores in the GPU are used for tasks that are not supported or cannot be efficiently implemented on tensor cores. This includes tasks such as pairwise combination of blocks of data for contiguous SNPs (`combine` routine), and the application of the scoring function (`applyScore` routine). The general purpose hardware in the GPU is also used for non-computationally intensive portions of the search such as computing contingency tables for individual SNPs (`indivPop`) and for second-order interactions (`pairwPop`). These represent a negligible cost in relation to the whole search, and are executed before the algorithm loops that perform the evaluation rounds implementing the exhaustive fourth-order search.

The calculation of contingency tables for second-order interactions (`pairwPop`) can be performed very fast due to the relatively small amount of combinations to process. These are used for partially inferring contingency table values pertaining to third-order interactions, which are in turn used to accelerate the construction of contingency tables pertaining to fourth-order interactions, as presented in Section 3.3. Contingency tables for pairwise interactions are also used, together with population counts of individual SNP data (calculated in `indivPop`), for translating the output of XOR+POPC tensor operations, when targeting architectures without support for AND+POPC (Section 3.4). Translating the output of XOR+POPC operations and the inference of genotype counts both for third and fourth-order interactions are integrated in the `applyScore` routine of Algorithm 1.

The `combine` routine, when passing the inputs W_i and X_i in this order (inside the loop with the X_i iterator), combines every row of a block of B SNPs (starting at W_i) with every row of another block of B SNPs (starting at X_i) relying on bitwise AND operations. Since two genotypes of a single SNP are represented in the dataset format used (see Section 3.1), the `combine` routine results in a matrix with $4 \times B^2 \times N_0$ bits for controls (or $4 \times B^2 \times N_1$ for cases), assuming no padding has been performed. This routine is used for combining blocks of SNPs in other points of the algorithm. The ranges of SNPs delimiting these blocks are determined by the corresponding loop iterator variables, which are passed as input.

Tensor cores are used for the most computationally intensive portions of the algorithm. These portions of the code are routines `tensorOp_3way`, and especially `tensorOp_4way`. The first uses the outcome of combining blocks of SNPs ($wx_{0|1}$, $wy_{0|1}$, $xy_{0|1}$) in the `combine` routine, and, relying on tensor acceleration, performs core computations for the construction of contingency tables of third-order interactions. For example, when passing both $wx_{0|1}$ and X_i to this routine (inside the loop with the X_i iterator), tensor core operations are used to construct the contingency tables for third order interactions (or its proto-values if using XOR+POPC tensor operations). These tables pertain to all third-order sets combining SNPs from W_i to $W_i + B$ (not inclusive), SNPs from X_i to $X_i + B$ and SNPs from X_i to M. This step results in a matrix with $8 \times B^2 \times (M - X_i)$ integer values, which represents the frequency counts of 8 genotypes (out of 27) for $B^2 \times (M - X_i)$ sets of three SNPs.

Notice that a block of SNPs (each with B SNPs) with a given index is only combined with other blocks of SNPs with the same or a larger index. In order to minimize non-useful work, the routines processing the blocks of SNPs receive as parameters the relevant loop iterators. For example, `tensorOp_3way`, when called inside the loop iterating Y_i, receives as input the value of this iterator because at this stage of the algorithm it is only required to combine $wy_{0|1}$ (the ouput of combining a block of SNPs W with a block of SNPs Y) with SNPs with index equal or larger than this value.

Out of the tensor-accelerated routines, `tensorOp_4way` is the most computationally intensive. While the combination of a block of SNPs Y with a block of SNPs Z is also performed inside the innermost loop, processing fourth-order interactions entails performing more calculations. The output of the latter step, which is repeated for cases and controls, is a matrix with 16 values per each quad of SNPs processed in the combination round. The frequency counts for genotypes with one or more alleles of the homozygous minor configuration (*aa*) are derived with simple arithmetic operations (see Section 3.3), before application of the scoring function.

3.3 Reducing the cost of evaluating each combination of SNPs

Consider the *yes / no* question asking if an SNP pertaining to a given sample has a given allele type. The answer to this question is only affirmative in regard to one of the three possible allelic configurations. Thus, knowing the answer in relation to two of the three possible allele types, always allows to infer the answer in relation to the remaining allele type. Applying this knowledge to fourth-order searches allows to systematically derive the counts for most genotypes resulting from combining four SNPs. Given the frequency counts for the 16 genotypes (32 values, considering cases and controls) that only have the first two allelic configurations (out of the possible three), the remaining 2×65 counts can be analytically derived. For example, the counts of genotype $\{W_1, X_0, Y_2, Z_1\}$ can be derived subtracting the counts for genotypes $\{W_1, X_0, Y_0, Z_1\}$ and $\{W_1, X_0, Y_1, Z_1\}$ from the counts for the third-order genotype $\{W_1, X_0, Z_1\}$. However, for performing fourth-order searches relying on such calculation method, it is required to precalculate the values of contingency tables for third-order interactions.

The problem is that doing so in a single phase, e.g. generating the contingency tables for all third-order combinations of SNPs at the application start (as in [15]), significantly restricts the type of datasets that can be processed in a given fourth-order search. This limitation is due to rapidly growing memory requirements with increase in the number of SNPs. In the face of the increase in performance that can be gained with efficient use of tensor acceleration, it becomes especially important to mitigate such limitation. That is the reason behind the separation of the contingency table construction for third-order interactions into three distinct computational phases (see `tensorOp_3way` routines in Algorithm 1).

Notice that contingency tables for third-order interactions are also accelerated deriving the frequency counts for genotypes where any of the SNPs in the triplet has allele type of the homozygous minor (*aa*) configuration. Relying on counts for 8 out of the 27 genotypes in a given third order interaction and on contingency tables precalculated for second-order interactions, the counts for

the remaining 19 third-order genotypes (for cases and controls) are similarly calculated relying on sums and subtractions.

3.4 Compatibility layer for using XOR+POPC acceleration

The proposed approach supports architectures with different 1-bit tensor processing capabilities. On one end, it supports those with native support for fused AND+POPC operations on the tensor cores. However, it also supports devices with a more restricted feature set, e.g. accelerators with functions tailored at processing BNNs, that perform XOR+POPC as part of tensor operations.

For pairwise searches, output of AND+POPC can be derived from XOR+POPC such that $\text{POPC}(A \cdot B) = \frac{\text{POPC}(A)+\text{POPC}(B)-\text{POPC}(A \oplus B)}{2}$. The term $\text{POPC}(A \cdot B)$ represents the output (an integer value) that one gets when performing AND between the bitvectors A and B followed by population count (i.e. counting bits set to 1). The term $\text{POPC}(A \oplus B)$ represents the output (an integer value) that one gets when performing XOR between the bitvectors A and B followed by population count (or fused XOR+POPC). The equivalence (proof in [16]) encapsulates the knowledge about the fact that removing the bits that do not match between bitvectors A and B — $\text{POPC}(A \oplus B)$ — from all the bits set in these bitvectors, results in the number of matching 1's. Dividing the latter by 2 (i.e. a right shift) results in the exact same output as if executing the AND operation followed by population count — the term $\text{POPC}(A \cdot B)$. Performing this calculation is efficient in the context of the problem under study because there is extensive reuse of the terms $\text{POPC}(A)$ and $\text{POPC}(B)$, for combination of A and B with other bitvectors.

Binary processing using tensor cores is used both in the context of calculation of contingency tables for fourth-order interactions and contingency tables for third-order interactions, which are used to partially derive the former contingency tables. Notice that mapping to AND+POPC (e.g. if using a device that only supports XOR+POPC) is performed on top of the output of the accelerator routines that access the tensor cores. When processing fourth-order interactions, the terms $\text{POPC}(A)$ and $\text{POPC}(B)$ represent the number of samples that have a given genotype for two pairwise interactions of SNPs. When processing third-order interactions, $\text{POPC}(B)$ represents the number of samples that have a given genetic configuration for the third SNP of a given triplet.

3.5 Optimizing implementation for GPU devices

The K2 score, the scoring function used in the context of the proposed approach, has been optimized relying on the `lgamma()` intrinsic, representing sums of logarithms by logarithms of factorials and, making use of the equivalence ($\Gamma(x) = (x-1)!$) [1], mapping the factorial to the gamma function. Moreover, we rely on a lookup table, efficiently constructed at the application start, that stores all the `lgamma()` values that can be requested during the search phase.

The indexes of the locally best set of SNPs are stored in a 64 bit integer. These bits are shared by the four indexes that represent the set of SNPs. The largest SNP index that can be represented is 65535, which in the context of fourth-order searches allows for up to 768.54 peta combinations to be evaluated.

Reduction of the scores, required to identify the best candidate solution (i.e. quad of SNPs), are performed taking into account the particularities of the GPU accelerator memory subsystem. This means, leveraging the particular latency and bandwidth of the different types of memory. Targeting the GPU memory subsystem, reduction of scores is accomplished relying on private memory within individual threads, shared memory between threads in a thread block, and on global memory inter thread blocks.

The proposed approach is tailored for systems with NVIDIA GPUs due to the native support for 1-bit precision operations (XOR+POPC or AND+POPC) in tensor cores. This is the numerical precision used since it allows achieving the highest performance for the targeted problem. However, the proposed processing scheme in blocks of SNPs is expected to be applicable to tensor cores from other vendors using other numerical precisions (e.g. INT4). Moreover, in the absence of tensor cores, the matrix-matrix operations could also be offloaded to the available general-purpose GPU cores.

3.6 Targeting Multi-GPU systems

The proposed approach supports multi-GPU systems. Relying on the OpenMP parallel programming library, workload is partitioned between different CPU threads at the level of the outerloop, i.e. the loop with the `W` iterator (see Algorithm 1, line 4). The number of OpenMP threads is equal to the number of GPUs, being each assigned to a particular thread. Thus, since OpenMP threads execute complete loop iterations, all calls to GPU routines pertaining to a given loop iteration, i.e. `combine`, `tensorOp_3way`, `tensorOp_4way`, `applyScore` and `findGlobalyBestSol`, are executed on the same GPU, i.e. that which is assigned to the OpenMP thread executing the iteration. The workload per loop iteration differs, decreasing for iterations with higher index. Relying on the dynamic OpenMP schedule policy, each CPU thread (to which a specific GPU has been assigned) requests the execution of another iteration (after terminating execution of that currently assigned), until all iterations have been processed. This allows balancing the total workload between the GPUs, avoiding instances of idle GPUs.

The proposed approach is constructed in such a way that no communication between GPUs is required during a given search. Locally best scores are reduced inside each GPU, being reduction at the host performed only when there are no more loop iterations to process. Moreover, the full dataset is transferred at the application start to each of the GPUs, which does not pose a challenge in relation to memory resources. As an example, a dataset with 16384 SNPs and 1 million samples, which is significantly larger and more computationally challenging than the largest dataset considered in the experiments, represents only $\sim$ 3.8 GB of information in the dataset format used (each A100 SXM4 GPU has 80GB). The runtime cost of the memory transfers from host to the GPUs is also not a major issue, since the search time is always expected to be multiple orders of magnitude higher. For fast access, each GPU also stores a full copy of the `lgamma()` lookup table, and of the contingency tables for individual SNPs (`indivPop()`) and for pairwise combinations (`pairwPop()`).

On systems with the NVLINK direct GPU-to-GPU interconnect, one could exploit the increased bandwidth between GPUs in relation to that with the connection to the host (NVLINK Gen3: 600GB/s, PCIe Gen4: 64GB/s). For example, the dataset could be partitioned in as many parts as the number of GPUs in the system, with each

system receiving a portion of the dataset from the host. Then, each GPU would receive the remaining parts of the dataset through direct comunication with the other GPUs. Notice however, that while interesting, this optimization will not affect the overall runtime, due to the relative magnitude of the search time.

4 EXPERIMENTAL RESULTS

This section presents the performance evaluation of the proposed approach for exhaustive fourth-order epistasis detection.

4.1 Targeted Systems

We targeted three systems, which are depicted in Table 1. The NVIDIA CUDA compiler (NVCC) V11.4.120 (part of CUDA 11.4) and the CUTLASS library v2.5, a set of template abstractions for implementing computations related to matrix multiplication, were used for compiling and implementing the kernel that uses the tensor cores. Systems S1 and S2 are single-GPU systems, with a Titan RTX and an A100 PCIe GPU, respectively. System S3 is the 8-GPU variant of the NVIDIA HGX A100 supercomputing platform [18].

The proposed approach has been devised to efficiently make the most out of the available tensor acceleration hardware. Thus, the computational resources of the tensor cores in the targeted GPU accelerators and their capability in terms of operations supported are two of the most relevant hardware features. Although the A100 GPU has less tensor cores than the Titan RTX GPU, each individual tensor core in the former is capable of achieving 4× the throughput. Each tensor core in Turing is capable of performing 1024 fused bit-level XOR+POPC operations per clock cycle. Since the Titan RTX (Turing) has 576 tensor cores, it has a theoretical throughput of 2088 TOPS (each XOR+POPC counted as two operations) at the advertised boost clock frequency of 1770 MHz. In Ampere, each tensor core performs 4096 fused bit-level XOR+POPC or AND+POPC operations per cycle. Thus, since the A100 (Ampere) has 432 tensor cores, it has a theoretical throughput of 4992 TOPS at the advertised boost frequency of 1410 MHz (both PCIe and SXM4 form factors).

Notice that due to the nature of the problem at hand, the proposed algorithm and its implementation, the bus interface speeds and even the performance of the CPU (or its number of cores) are not significant to the performance that can be achieved with a given GPU accelerator. Transferring the dataset to the GPU represents a negligible cost (however, still accounted for in the reported metrics) in comparison to the search portion of the proposed approach, which is completely device-bound. The Host, i.e. the CPU device(s), assumes the role of orchestrating the use of the targeted accelerator.

4.2 Target Metrics

To enable direct comparison between executions with different datasets, the main metric reported in this paper is the number of fourth-order combinations of SNPs evaluated per second, scaled to sample size. This metric is calculated by taking into account the complete execution of the search, including loading of the input genotypic data from disk and all host-device communication.

We also report the average tensor Tera Operations Per Second (TOPS) achieved for some epistasis detection runs, particularly those achieving highest performance. This metric takes into account all bit-level operations performed on tensor cores. The higher it is, the closer are tensor cores to being exercised to their full potential. Fused operations, i.e. XOR+POPC or AND+POPC, are counted as two operations, as usual for the fused (multiply-add) operation.

4.3 Datasets

For evaluating the approach proposed in this paper we performed experiments with a collection of synthetic datasets. We consider datasets with 256 (174792640), 512 (2829877120), 1024 (45545029376), 2048 (730862190080) and 4096 (11710951848960) SNPs. The number of fourth-order combinations to evaluate is represented within brackets. Experiments on systems S1 and S2, both single-GPU systems, have been performed with datasets composed of up to 2048 SNPs, each with 32768, 65536, 131072, 262144 or 524288 samples. Due to the required execution time, results for experiments with datasets with 4096 SNPs and 262144 or 524288 samples, are reported exclusively for runs on the multi-GPU system (S3), in addition to evaluation of datasets with 1024 and 2048 SNPs for these numbers of samples. All these datasets have half samples of each kind (cases or controls). Datasets of diferent shapes and sizes have been considered in the experiments in order to cover a larger space (i.e. to be representative) of possible dimensions of real complex datasets. Notice that although the genotypic data in the datasets is synthetic (i.e. randomly generated), the type and the volume of operations performed does not depend on the particular genotypic data.

4.4 Parametrization of the experiments

The code using the CUTLASS library to perform the binarized matrix operations relying on tensor cores was subjected to a tuning phase, to select the most suitable values for parameters identified as having a significant effect on performance. For the Ampere (Turing) GPU microarchitecture, each thread block computes a $128 \times 256 \times 1024$ ($128 \times 128 \times 1024$) tile and a warp computes a $64 \times 64 \times 1024$ ($64 \times 32 \times 1024$) tile. On Turing, at the instruction level, a $8 \times 8 \times 128$ tile has been used. This is the only instruction tile supported on Turing for binary operations. On Ampere, a $16 \times 8 \times 256$ tile has been used, which allowed to achieve higher tensor throughput.

The Ampere GPU architecture has support for both AND+POPC and XOR+POPC operations on tensor cores. The experimental evaluation targeting the Titan RTX GPU (system S1) only considers the use of fused XOR+POPC, since this is the only bit-level operation supported on the Turing tensor cores.

Most of the experiments for evaluating the proposed approach have been performed by relying on a block size of 32 SNPs, which has been found to be sufficiently large to allow achieving high throughput in the tensor cores. This is the amount of contiguous SNPs that are the basis of the combination process, i.e. using a block size of 32 SNPs results in the combination of 32 SNPs W, 32 SNPs X, 32 SNPs Y and 32 SNPs Z on a given round. In order to show the effect of using a block size that is larger than strictly required to achieve sufficient tensor core throughput, some experiments have been performed relying on a block size of 64 SNPs.

The experiments also consider runs allowing the concurrent execution of multiple evaluation rounds, which has been implemented relying on the use of multiple CUDA streams per GPU device. Even considering that a suitable value has been selected for the block size parameter of the proposed approach, concurrent execution of

Table 1: The three targeted GPU-based systems with binary processing capable tensor cores.

Systems	CPU (arch.)	**GPU (arch.)** | **Driver** #stream — tensor cores | freq. | mem. (band.)	DRAM	Operating System
S1	Intel Core i9-10980XE (Cascade Lake)	**NVIDIA Titan RTX (Turing)** | **470.42.01** 4608 — 576 (1350/1770MHz) | 24GB (672 GB/s)	128GB DDR4	CentOS 7.8
S2	AMD EPYC 7452 (Zen 2)	**NVIDIA A100 PCIe (Ampere)** | **460.73.01** 6912 — 432 (765/1410MHz) | 40GB (1555 GB/s)	512GB DDR4	Ubuntu 20.04
S3	(2×) AMD EPYC 7763 (Zen 3)	**(8×) NVIDIA A100 SXM4 (Ampere)** | **495.29.05** 6912 — 432 (1275/1410MHz) | 80GB (2039 GB/s)	2048GB DDR4	Ubuntu 18.04

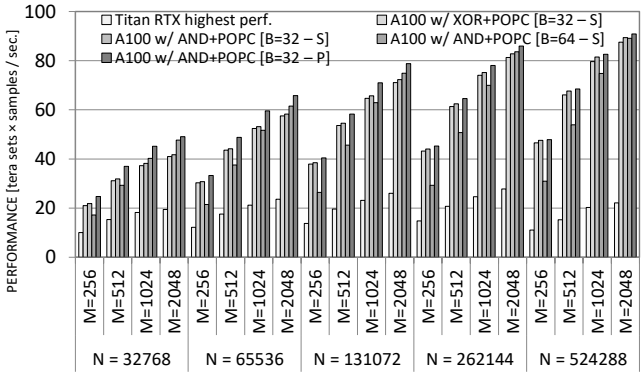

Figure 2: Performance in tera ($\times 10^{12}$) quads of SNPs processed per second, scaled to sample size, achieved on systems S1 (Titan RTX) and S2 (A100 PCIe).

multiple rounds can be useful when processing datasets that are not sufficiently large to saturate the computational resources of a given GPU by executing an evaluation round at a time.

4.5 Performance analysis on single-GPU

Fig. 2 shows the performance achieved when targeting systems S1 (Titan RTX) and S2 (A100 PCIe). The evaluations consider the use of XOR+POPC (supported both in Turing and Ampere) or AND+POPC operations (supported only in Ampere) for the construction of contingency tables, relying on 32 or 64 SNPs per block (B) and on serialized (S) or parallel (P) execution of evaluation rounds, for processing datasets with 256, 512, 1024 or 2048 SNPs (M) and 32768, 65536, 131072, 262144, or 524288 samples (N). While the analysis is focused on Ampere, the most recent microarchitecture, we also report for the Turing GPU the highest performance achieved when processing the different considered datasets.

On the A100 PCIe GPU, the proposed approach can achieve similar levels of performance relying on either XOR+POPC or AND+POPC operations. This experimentally shows that the layer implemented for enabling interoperability with devices that only support native XOR+POPC operations, through efficiently mapping the output of these operations to the desired output (see Section 3.4), does not incur any significant overhead.

Due to the increase in tensor core throughput in the Ampere architecture, performance on the A100 PCIe GPU is significantly higher than on the Titan RTX GPU. In particular, for the dataset for which the highest performance was achieved on the Titan RTX GPU (2048 SNPs and 262144 samples), the A100 provides a speedup of 2.84× (2.81× if using XOR+POPC), delivering a performance of 78.78 (AND+POPC) or 78.01 (XOR+POPC) quads of SNPs per second (normalized to sample size).

Processing more than 262144 samples resulted overall in a more efficient use of the computational resources of the A100 PCIe GPU. A performance of 90.9 (AND+POPC) or 90.0 (XOR+POPC) quads of SNPs per second has been achieved on the A100 PCIe GPU when processing a dataset with 2048 SNPs and 524288 samples. This was not the case on the Titan RTX. For the same CUTLASS parameters used in other runs on the Titan RTX, profiling of the kernel that performs the tensor-accelerated matrix operations showed that performance degrades when processing inputs with 524288 samples in relation to that achieved processing inputs with 262144 samples, which explains the drop in epistasis detection performance. Processing datasets with 524288 (or more) samples while keeping close to the highest performance achieved on the Titan RTX (27.8 quads per second) can be achieved dividing the processing of samples into multiple matrices covering 262144 samples each (2 if processing 524288 samples), adding element-wise the output matrices (partial contingency tables) of the matrix operations.

Notice that for epistasis detection purposes, one is concerned in evaluating combinations of SNPs, i.e. not permutations or sets with repeated SNPs. The percentage of unique combinations processed using a 32 SNPs block is 50.5%, 69.6%, 83.0% or 90.9%, for datasets with 256, 512, 1024 or 2048 SNPs, while adopting a 64-SNP block leads to 29.8%, 51.1%, 70.0% or 83.2% unique combinations being processed, respectively. Thus, relying on a block size of 64 SNPs only particularly improved performance for datasets that are both large in regard to number of SNPs (drop in ratio of useful computation not too steep) and small in the number of samples (larger input matrices in regard to amount of SNPs represented compensate for small amount of samples), being the most extreme case the processing of the dataset with 2048 SNPs and 32768 samples. In most cases, due to the increase in wasteful computation, increases in average TOPS did not translate into higher performance.

Given sufficiently large input matrices, which tends to happen as a consequence of how they are constructed, individual matrix operations can achieve high utilization of the available hardware resources. Concurrent execution of rounds only resulted in significantly improved performance for datasets with small amounts of samples, therefore it has not been considered for experiments with 64-SNPs blocks. Notice that relying on 32-SNP blocks results in

matrices of $4096 \times N_0$ bits (assuming no padding) when processing controls (cases processed separately), where N_0 is the number of controls, being fed as input to the most computationally intensive kernel of the approach, i.e. the kernel that relies on tensor cores to construct contingency tables for fourth-order interactions.

The A100 GPU has a theoretical throughput in regard to tensor-accelerated bit-level operations that is 2.39× higher than that of the Titan RTX GPU. However, considering the highest performing search on each GPU, the performance improvement achieved with the A100 GPU in relation to the Titan RTX is higher (3.24×). This can be explained by the fact that higher efficiency has been achieved when targeting the Ampere GPU. The NVIDIA Nsight Compute kernel profiler has revealed that a significantly higher speed-of-light metric (percentage of utilization in relation to theoretical maximum) is consistently achieved for the kernel that uses the tensor cores for calculating fourth-order contingency tables on the Ampere (~ 90%) GPU in relation to Turing (~ 65%).

Achieving relatively high throughput at the tensor cores during full epistasis detection runs is indicative that the application is not being significantly slowed down by the remaining supporting code executing on the traditional data-path. On the highest performing parametrizations, the A100 PCIe and the Titan RTX achieved 66% (3305 out of 4992 TOPS) and 48% (1010 out of 2088 TOPS) of the theoretical throughput calculated at boost frequencies. The software power cap was consistently reported by the NVIDIA System Management Interface (`nvidia-smi`) as being active, preventing the A100 and Titan RTX GPUs from achieving the advertised boost frequencies of 1410 and 1770 MHz, respectively. This can be attributed to the simultaneous use of a significant amount of resources. In fact, profiling a complete epistasis detection run on the Titan RTX with a dataset representing 512 SNPs and 262144 samples reveals that a total of 82.85% of the GPU time is spent performing the tensor-accelerated operations for constructing contingency tables for fourth and third interaction orders. The scoring function, including the XOR+POPC interoperability layer (for targeting Turing GPUs) and the inference of genotype frequencies, take 8.58% of the time. Finally, 8.41% is spent on the precombination of blocks of allelic data, i.e. preparing data to be feed to the tensor cores, 0.15% on calculating the contingency tables for pairwise searches and only 0.01% on host-device memory transfers.

4.6 Performance on 8-GPU HGX A100 platform

Fig. 3 shows the performance achieved on system S3 (8-GPU HGX A100), a supercomputer class system with eight A100 SXM4 GPUs, when using 1, 2, 4 or 8 GPUs, for datasets with 1024, 2048 or 4096 SNPs and 262144 or 524288 samples. The parametrization used for runs executed on this system corresponds to the one that resulted in highest performance for the system S2 (A100 PCIe GPU).

Overall, strong scaling improves when processing larger datasets. Speedups of 1.98× (2 GPUs), 3.79× (4 GPUs) and 7.11× (8 GPUs) in relation to using a single GPU on the same system have been achieved when processing the largest dataset considered in the experimental campaign in regard both to SNPs (4096) and samples (524288). This represents a reduction of the execution time from close to 14.5 hours to around 2 hours of search time.

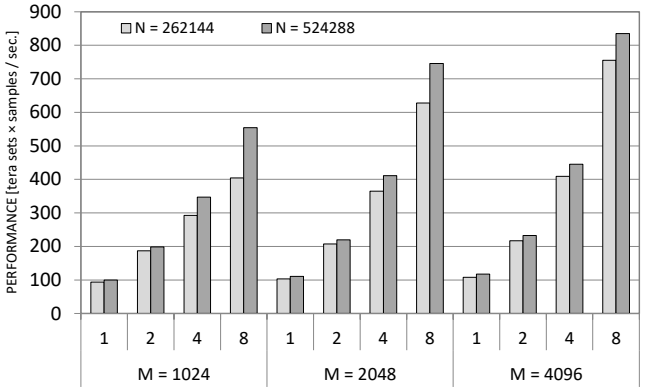

Figure 3: Performance in tera ($\times 10^{12}$) quads of SNPs processed per second, scaled to sample size, achieved on system S3 (8-GPU HGX A100), using 1, 2, 4 or 8 A100 SXM4 GPUs.

Up to 28947 Tensor TOPS have been achieved when processing this dataset with the full HGX A100 system, which represents 72% of the theoretical maximum (8×4992 TOPS) calculated at vendor-announced boost clocks. Such throughput resulted in 835.4 tera sets processed per second (scaled to sample size).

Notice that for a fixed problem size, the total individual workload processed by each GPU decreases with increases in the number of GPUs targeted during a given epistasis detection run. Multi-GPU paralelization has been performed through division of work at the level of the outermost loop in Algorithm 1 (see Section 3.2), assigning rounds to GPUs in a dynamic fashion. Work division could also have been implemented with finer-grained approaches. Alternative parallelization schemes include division of work at the level of other loops (e.g. the one iterating over X in Algorithm 1, line 5) and/or at the level of the samples, i.e. assigning different ranges of samples to different GPUs and combining the frequency counts for each genotype configuration between GPUs to produce the contingency tables. Out of the alternative parallelization schemes considered during development of the proposed approach, none resulted in overall higher performance for the datasets considered in the experiments. As can be extrapolated from the experimental results reported in Section 4.5, dividing the samples between GPUs is expected to negatively impact the performance, unless processing datasets with significantly more samples than those considered.

The number of SNPs to process has an effect on base performance, i.e. performance considering execution on a single GPU. Overall, higher performance is achieved when processing the datasets with 4096 SNPs. This happens because a higher ratio of useful computations (to the total amount of computations) is achieved in relation to when datasets with fewer SNPs are processed. In addition, as previously discussed for system S2 (with A100 PCIe GPU), performance is higher when processing datasets with significant amount of samples on system S3. However, the number of samples does not have an effect on scaling as pronounced as that of the number of SNPs being processed, since the number of samples has no effect on workload distribution between the GPUs in the system.

Compared with the tensor throughput achieved on system S2, higher tensor throughput has been achieved on system S3 even

when targeting a single GPU. For the datasets evaluated on both systems, epistasis detection is always faster on one of the A100 SXM4 GPUs than on the A100 PCIe GPU on system S2. For example, for a dataset with 2048 SNPs and 524288 samples, performance on the A100 SXM4 is 1.23× higher than that on the A100 PCIe (110.5 vs. 90.09 tera quads evaluated per second, scaled to sample size). The vendor announced GPU boost frequency is the same for both A100 GPUs. However, since the A100 SXM4 has a higher TDP (400W vs. 250W), it achieves higher frequencies, being memory bandwidth also higher (2039 vs. 1555 GB/s) due to higher memory frequency.

5 RELATED WORK

Parallel architectures and accelerator devices suit the type of data parallelism exposed by exhaustive epistasis detection searches. There are approaches derived to GPU accelerators [3, 5–7, 10, 15, 25, 26], multicore CPUs [2, 12], other parallel devices such as the x86-based Xeon-Phi accelerators [5], and specialized architectures in Field Programmable Gate Array (FPGA) devices [21, 25].

Given the computational complexity of epistasis detection, most of these approaches are focused on second-order searches [5, 7, 12, 25, 26]. Some approaches support third-order searches [2, 3, 10, 14, 16, 20], while exhaustive methods focusing on fast fourth-order searches are rare and recent in the literature [2, 15, 21]. In fact, this paper is the first one that exploits tensor hardware for performing epistasis detection using fourth-order search methods.

We directly compare the proposed approach with three recently published works [2, 15, 21] that use bitwise representations and methods for achieving epistasis detection with support for fourth-order searches. Table 2 presents the performance, and the hardware configurations used in related art, as well as the performance achieved with the proposed approach on the three targeted systems. The datasets sizes (i.e. SNPs and samples) that resulted in the performance (scaled to sample size) reported for the different approaches are also referenced. Notice the datasets referenced are those that were reported as resulting in highest performance in the corresponding publications. Thus, while significantly smaller (in regard to SNPs and samples) than the largest datasets used to evaluate the proposed approach, those datasets are challenging enough to saturate the related art approaches in regard to performance.

Table 2: Performance in tera ($\times 10^{12}$) sets processed per second scaled to number of samples of epistasis detection approaches relying on fourth-order exhaustive search methods.

Approach	Targeted Device(s) (CPU, GPU or FPGA)	Dataset SNPs × samples	Performance (combs × samples / sec)
BitEpi [2]	(2×) Xeon E5-2660 V3 (CPU)	500 × 2000	0.011
HEDAcc [21]	(1×) Zynq-7000 (FPGA)	2000 × 4000	0.28
	(1×) Zynq-US+ (FPGA)	2000 × 4000	0.35
	(1×) Virtex-7 690T (FPGA)	2000 × 4000	0.42
Nobre et al. [15]	(1×) Core i9-10920X (CPU)	350 × 40000	0.032
	(2×) Xeon Gold 6128 (CPU)	350 × 10000	0.034
	(1×) UHD P630 (iGPU)	250 × 80000	0.045
	(1×) Iris Xe MAX (GPU)	200 × 80000	0.371
	(1×) Titan RTX (GPU)	250 × 80000	2.245
Proposed approach	(1×) Titan RTX (GPU)	2048 × 262144	27.782
	(1×) A100 PCIe (GPU)	2048 × 524288	90.904
	(8×) A100 SXM4 (HGX A100)	4096 × 524288	835.366

The BitEpi method [2], which also relies on bitwise representations of data and on fast binary operations, is a method implemented in C++ that targets only multicore CPUs. As a result, the achieved performance is lower than that of other approaches. Executing on a machine with two deca-core Intel Xeon E5-2660 V3 CPUs, the authors of BitEpi report a performance of 0.011 tera quads of SNPs processed per second (scaled to sample size) for a run with a dataset with 500 SNPs and 2000 samples.

The HEDAcc FPGA-based approach [21] is able to perform second, third and fourth-order searches with a strong emphasis on energy-efficiency. Execution on Virtex-7 690T, Zynq-US+ and Zynq-7000 FPGA-based platforms achieves 0.42, 0.35 and 0.28 tera quads of SNPs processed per second. These performance values are scaled to the sample size of the dataset used (2000 SNPs × 4000 samples).

The previously highest performing approach in fourth-order searches is a method that targets systems with SYCL-compatible accelerators [15]. Although multi-core CPUs are supported, significantly higher performance has been achieved targeting GPUs, given the data-parallel nature of the problem (see Table 2). A performance of 2.25 tera quads per second (scaled to sample size) has been achieved on a Titan RTX GPU [15], when processing a dataset with 250 SNPs and 80000 samples. Processing a dataset of the same dimensions, a performance of 14.42 quads per second has been achieved using the proposed approach on the same GPU model, which represents a 6.4× speedup in relation to the state-of-the-art. Notice that the proposed approach is capable of achieving even higher performance with larger datasets (due to better saturation of the compute resources), which results in a performance improvement of up to 12.4× in relation to the state-of-the-art approach from [15]. Naturally, a performance improvement of 41.1× can be achieved on the significantly more resourceful A100 PCIe GPU, and 372.1× when targeting the 8-GPU HGX A100 super-computing platform. Notice that the majority of this improvement came from the usage of the tensor cores and that just an insignificant amount could possibly be attributed to the differences on the runtime systems (CUDA vs. SYCL). Furthermore, the algorithm proposed in [15] has a significant limitation in regard to the amount of SNPs that can be processed in a given search, due to all contingency tables for third-order interactions (used to derive some values for fourth-order interactions) being constructed in a single phase. Such approach to the construction of contingency tables requires an amount of memory, which depending on the number of SNPs in the input dataset, will be larger than the memory available at the device. This is a limitation that we had to overcome through dividing the construction of contingency tables for third-order interactions into multiple phases (see Algorithm 1 in Section 3.2).

Due to the type of matrix decomposition and operations used to enable the efficient exploitation of tensor cores, larger datasets are required to achieve performance saturation on a given system in comparison to other state-of-the-art approaches for fourth-order detection. As experimentally demonstrated, runs using the proposed approach with datasets that are small in regard to either the number of SNPs or samples are not able to extract highest performance (see Section 4.5). This is due to an increase in the percentage of repeated combinations of SNPs that are evaluated and/or lower throughput at the tensor cores. As a result, some datasets might be processed more efficiently using other state-of-the-art approaches. Notice, however,

that the impetus behind the development of the proposed approach is precisely to, in the context of fourth-order epistasis detection searches, enable the efficient processing of challenging datasets, which are those that benefit most from acceleration through the full use of all available hardware resources.

Related art using tensor cores tackles only second and third order searches. Relying on floating-point matrix multiplication operations on the Volta microarchitecture (2017), the CoMet approach [10] has been the first to adopt GPU tensor cores in the context of epistasis detection. The Turing [19] microarchitecture (2018) adds support for integer numerical types, including a bit-level tensor operation where fused XOR and population count operations (herein referred to as XOR+POPC) are used in the place of fused multiply-add. Relying on these operations, the highest performing approach in exhaustive epistasis detection is an approach that uses binary processing (XOR+POPC) on tensor cores [14, 16]. The Ampere [17] microarchitecture (2020) adds support for fused AND and population count operations (herein referred to as AND+POPC), which up to this point had not been explored for epistasis detection purposes.

In relation to second or third-order searches (e.g. [14, 16]), implementing fourth-order searches using tensor cores imposes additional challenges. As a result, the proposed approach includes memory/compute algorithmic optimizations and parameters specialized to fourth-order. On top of having to take GPU memory size limitations into consideration when designing the proposed algorithm, due to the increase in interaction order, a special attention had also to be given to the way sets of SNPs are combined into matrix operations to guarantee that a large portion of the operations performed in tensor cores are useful for epistasis detection purposes and that throughput of operations in tensor cores is high. In order to guarantee high fourth-order epistasis detection performance, the proposed algorithm had to be devised around the use of significantly smaller blocks of SNPs, combined in a particular way that makes the inputs to the tensor-accelerated matrix operations large enough to extract high throughput at the tensor cores. Also worthwhile to mention is the fact that achieving support for past tensor-core GPU architectures (Turing) through the use of the XOR+POPC compatibility layer, in a manner that is both computationally and memory-wise efficient, is more complex in the context of fourth-order searches than in second or third-order searches.

While the presented approach performs epistasis detection relying on an exhaustive method, its applicability can be extended to non-exhaustive approaches. For example, the GPU-based SingleMI approach implements an exhaustive search on top of a filter heuristic which extracts suitable SNP candidates [11]. In such approaches, reduced execution times can be achieved as a result of not all SNP combinations being evaluated. This is especially the case for datasets with large amounts of SNPs. Notice that our search method can also be used as part of such approaches. In fact, the use of a fourth-order exhaustive method that makes full use of modern GPU architectures with tensor cores (such as ours) can potentially result in achieving increased accuracy, since more SNPs can be considered during the search performed after filtering.

6 CONCLUSIONS

This paper presents an algorithm for exhaustive epistasis detection at fourth interaction-order that makes efficient use of modern tensor processing hardware. The proposed approach is the first one that exploits tensor cores to perform searches at such a high interaction-order. It is able to target devices with high-throughput fused XOR+POPC processing capabilities, typically targeted at neural network processing, and also devices that have a more complete feature set, with native support for tensorized fused AND+POPC operations. In order to achieve high efficiency at the targeted devices for epistasis detection purposes, an approach has been especially designed for tackling fourth order searches that allows to achieve high throughput at the tensor cores while keeping memory requirements reasonable and mitigating non-useful calculations to a large extent, including the processing of repeated quads of SNPs.

The obtained experimental results are evidence of overall efficient hardware utilization use across the targeted GPU architectures, Turing (XOR+POPC) or Ampere (adds support for AND+POPC), on the three considered systems. Targeting a system with a Titan RTX (Turing), a performance of 27.8 quads of SNPs per second has been achieved, while up to 90.9 quads of SNPs processed per second were achieved on a A100 PCIe (Ampere) GPU. On a 8-GPU HGX A100 supercomputing platform, a performance of 835.4 quads of SNPs processed per second was attained, which represents epistasis detection performance multiple orders of magnitude higher in relation to other state-of-the-art approaches.

The proposed approach can, due to the nature of the problem, scale well if targeting additional computer nodes. For this reason, ongoing work includes making multi-node implementations extending the current multi-GPU implementation and exploring other types of devices, including custom hardware on FPGAs. Finally, we are aiming at extending the work to higher-order SNP interactions.

ACKNOWLEDGMENTS

This work was supported by FCT (Fundação para a Ciência e a Tecnologia, Portugal) and EuroHPC Joint Undertaking through UIDB/50021/2020 project and grant agreement No 956213 (SparCity).

REFERENCES

[1] George E. Andrews, Richard Askey, and Ranjan Roy. 1999. *The Gamma and Beta Functions.* Cambridge University Press, 1–60.
[2] Arash Bayat et al. 2021. Fast and accurate exhaustive higher-order epistasis search with BitEpi. *Scientific Reports* 11, 1 (05 Aug 2021), 15923.
[3] Rafael Campos et al. 2020. Heterogeneous CPU+iGPU Processing for Efficient Epistasis Detection. In *Euro-Par 2020: Parallel Processing.* Springer International Publishing, Cham, 613–628.
[4] Gregory F. Cooper and Edward Herskovits. 1992. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning* 9, 4 (01 Oct 1992), 309–347.
[5] Jorge González-Domínguez, Sabela Ramos, Juan Touriño, and Bertil Schmidt. 2016. Parallel Pairwise Epistasis Detection on Heterogeneous Computing Architectures. *IEEE Transactions on Parallel and Distributed Systems* 27, 8 (Aug 2016), 2329–2340.
[6] Jorge González-Domínguez and Bertil Schmidt. 2015. GPU-accelerated exhaustive search for third-order epistatic interactions in case–control studies. *Journal of Computational Science* 8 (2015), 93 – 100.
[7] Jorge González-Domínguez, Jan Christian Kässens, Lars Wienbrandt, and Bertil Schmidt. 2015. Large-scale genome-wide association studies on a GPU cluster using a CUDA-accelerated PGAS programming model. *The International Journal of High Performance Computing Applications* 29, 4 (2015), 506–510.
[8] Fernando Pires Hartwig et al. 2014. Evidence for an Epistatic Effect between TP53 R72P and MDM2 T309G SNPs in HIV Infection: A Cross-Sectional Study in Women from South Brazil. *PLOS ONE* 9, 2 (02 2014), 1–11.

[9] Cindy Im et al. 2018. Genome-wide search for higher order epistasis as modifiers of treatment effects on bone mineral density in childhood cancer survivors. *European journal of human genetics : EJHG* 26, 2 (2018), 275–286.

[10] Wayne Joubert et al. 2018. Attacking the Opioid Epidemic: Determining the Epistatic and Pleiotropic Genetic Architectures for Chronic Pain and Opioid Addiction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis* (Dallas, Texas) *(SC '18)*. IEEE Press, Piscataway, NJ, USA, Article 57, 14 pages.

[11] Daniel Jünger, Christian Hundt, Jorge González Domínguez, and Bertil Schmidt. 2017. Speed and accuracy improvement of higher-order epistasis detection on CUDA-enabled GPUs. *Cluster Computing* 20, 3 (01 Sep 2017), 1899–1908.

[12] Jan C. Kässens, Jorge González-Domínguez, Lars Wienbrandt, and Bertil Schmidt. 2014. UPC++ for bioinformatics: A case study using genome-wide association studies. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. 248–256.

[13] Trudy FC Mackay and Jason H Moore. 2014. Why epistasis is important for tackling complex human disease genetics. *Genome medicine* (6):42 (2014), 1–3.

[14] Ricardo Nobre, Aleksandar Ilic, Sergio Santander-Jiménez, and Leonel Sousa. 2020. Exploring the Binary Precision Capabilities of Tensor Cores for Epistasis Detection. In *2020 International Parallel and Distributed Processing Symposium (IPDPS 2020)*.

[15] Ricardo Nobre, Aleksandar Ilic, Sergio Santander-Jiménez, and Leonel Sousa. 2021. Fourth-Order Exhaustive Epistasis Detection for the XPU Era. In *50th International Conference on Parallel Processing* (Lemont, IL, USA) *(ICPP 2021)*. Association for Computing Machinery, New York, NY, USA, Article 27, 10 pages.

[16] Ricardo Nobre, Aleksandar Ilic, Sergio Santander-Jiménez, and Leonel Sousa. 2021. Retargeting Tensor Accelerators for Epistasis Detection. *IEEE Transactions on Parallel and Distributed Systems* 32, 9 (2021), 2160–2174.

[17] NVIDIA Corporation. [n. d.]. NVIDIA A100 Tensor Core GPU Architecture. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf. [Online; visited January-2022].

[18] NVIDIA Corporation. [n. d.]. *NVIDIA HGX platform.* https://www.nvidia.com/en-us/data-center/hgx [Online; visited January-2022].

[19] NVIDIA Corporation. [n. d.]. NVIDIA Turing GPU Architecture Whitepaper. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf. [Online; visited January-2022].

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

[21] Gaspar Ribeiro, Nuno Neves, Sergio Santander-Jiménez, and Aleksandar Ilic. 2021. HEDAcc: FPGA-based Accelerator for High-order Epistasis Detection. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 124–132.

[22] Jiya Sun et al. 2014. Hidden Risk Genes with High-Order Intragenic Epistasis in Alzheimer's Disease. *Journal of Alzheimer's disease : JAD* 41 (04 2014).

[23] Yingxia Sun et al. 2017. epiACO - a method for identifying epistasis based on ant Colony optimization algorithm. *BioData mining* 10:23 (06 Jul 2017), 1–17.

[24] Xiang Wan et al. 2010. BOOST: A fast approach to detecting gene-gene interactions in genome-wide case-control studies. *American journal of human genetics* 87, 3 (10 Sep 2010), 325–340.

[25] Lars Wienbrandt, Jan Christian Kässens, Matthias Hübenthal, and David Ellinghaus. 2019. 1000× faster than PLINK: Combined FPGA and GPU accelerators for logistic regression-based detection of epistasis. *Journal of Computational Science* 30 (2019), 183 – 193.

[26] Ling Sing Yung, Can Yang, Xiang Wan, and Weichuan Yu. 2011. GBOOST. *Bioinformatics* 27, 9 (May 2011), 1309–1310.

[27] Shawna J. Zimmerman, Cameron L. Aldridge, and Sara J. Oyler-McCance. 2020. An empirical comparison of population genetic analyses using microsatellite and SNP data for a species of conservation concern. *BMC Genomics* 21, 1 (01 Jun 2020), 382.