EPIClear: Exploiting Domain-Specific Features for Epistasis Detection Acceleration on Tensor Cores

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

Miguel Graça
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal
miguel.graca@inesc-id.pt

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

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

Abstract

High-order epistasis detection is challenging, making it important to efficiently leverage today's supercomputers. The fastest approaches are those relying on binary precision tensorized operations on modern GPUs. This paper presents a novel approach that significantly surpasses the state-of-the-art in high-order epistasis detection by leveraging previously unexplored domain-specific features on the genotype distribution patterns in the dataset. It accelerates time-to-solution with a computational step that reduces the volume of data that needs to be processed to count genotypes. The proposed approach achieves 4× higher performance on a A100 GPU than the previously fastest approach when processing balanced genotype distributions. Evaluation on datasets with unbalanced genotype distributions, which is something that is bound to happen in real datasets, results in significantly higher performance. The proposed accelerating scheme exhibits high scalability. Epistasis detection searches on the MeluXina supercomputer with 32 A100 GPUs resulted in a speedup of up to 30× in comparison to a single GPU, and in achieving a performance scaled to sample size of up to 13 Peta SNP combinations per second for the genotype distribution most unfavorable to the proposed accelerating scheme.

ICS '25, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3725769

CCS Concepts

• **Applied computing** → **Bioinformatics**; • **Computing methodologies** → **Parallel algorithms**.

Keywords

epistasis detection, high-performance computing, multi-GPUs.

ACM Reference Format:
Ricardo Nobre, Miguel Graça, Leonel Sousa, and Aleksandar Ilic. 2025. EPIClear: Exploiting Domain-Specific Features for Epistasis Detection Acceleration on Tensor Cores. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3721145.3725769

1 Introduction

Motivation. Epistasis detection is a bioinformatics application focused on finding combinations of single nucleotide polymorphisms (SNPs) that are associated with a given condition or trait [22]. SNPs can interact non-additively. As a result, considering epistasis (i.e. interactions between SNPs) allows to identify genotype-to-phenotype correlations that are not found considering only individual SNPs. Knowledge about novel associations can have several practical applications related to personalized treatment, such as identification of risk factors [33] and predicting drug [14] or infection response [13]. Epistasis searches have enabled finding gene interactions correlated to complex diseases, such as rheumatoid arthritis [11], bipolar disorder [28] and type 1 diabetes [36].

Parallel computing with tensor cores. The large SNP combination space to tackle has been addressed to a significant extent through the innovative use of the parallel computers' capabilities. In particular, the highest performing approaches for high-order epistasis detection map core epistasis operations related to genotype counting – the hotspot

of epistasis searches – to matrix operations on the tensor core units (TCUs) of NVIDIA GPUs [16, 24, 25]. However, even the fastest exhaustive methods can incur in a unreasonably large exploration time. For example, considering the fastest performance at third-order searches reported in literature on a GPU (Titan RTX) [25], processing a dataset with 100000 SNPs (≈ 166.7 trillion combinations) and 100000 samples would take around 305801 seconds (around 3 days and a half) at the reported maximum performance of 54.5 tera sets processed per second scaled to sample size. Doubling the amount of SNPs would result in a ≈ 8× increase in execution time, in proportion to the growth in combinations.

Accuracy and time-to-solution trade-off. Non-exhaustive methods have been proposed as a means to deal with the huge solution space of high-order epistasis detection. These include methods built around heuristics relying on optimization strategies such as genetic algorithms [6] and ant colony optimization [15], as well as filtering techniques [19] that reduce the amount of data to process. There is also literature on methods using machine learning algorithms. Several of those use deep neural network (DNN)-based methods [3, 5, 12, 20] supported on machine learning software platforms such as TensorFlow [1] or PyTorch [29]. Notice however, that while non-exhaustive methods can perform searches at an arbitrary high-order, they do so at the cost of a loss in accuracy that results from discarding most of the SNP combinations.

Proposed approach does not impact accuracy. Up to now, unlike non-exhaustive approaches, the exhaustive epistasis detection approaches proposed in the literature (e.g. [2, 16, 23, 25, 30, 31]) are not impacted in terms of performance by the particular genotypic data in the samples represented in the dataset being processed. For such approaches, only the dataset dimensions have an impact on execution time. For the first time, we are proposing an exhaustive search method for epistasis detection that leverages the genotypic features on the dataset for accelerating time-to-solution. The method efficiently prunes the matrix representation of the input dataset in intermediate stages of computation. By design, it exploits the necessarily significant amount of zeros that result from operating with matrices representing genotypic data in a one-hot-encoding scheme to reduce the computations to be performed on the TCUs.

Targeting challenging epistasis searches. We evaluate the proposed method in third-order searches, which are challenging for today's datasets, even when using large-scale systems (e.g. supercomputers). As part of an effort to design a high throughput solution, the proposed approach has been carefully crafted to combine the aforementioned optimization, which has been first explored in this paper, with the adaptation of several other optimizations explored in other approaches. To tackle the computational complexity of the problem under study, the proposed approach has also been developed to efficiently use multiple GPUs in multiple nodes.

As a result of the methods investigated and the development undertaken, a new tool called EPIClear [1] has been developed. This paper provides the following contributions:

(1) Novel method that uses domain-specific features to accelerate epistasis detection computations on TCUs.
(2) Integration of the proposed method as part of a novel epistasis detection tool targeting large-scale systems.
(3) Thorough evaluation considering datasets with different characteristics on a state-of-the-art supercomputer.

Contribution (1) is focused on the proposal of a method that reduces the amount of operations performed with no impact on accuracy through leveraging information on the dataset. Contribution (2) is concerned with developing a complete approach built around the proposed method, and other optimization techniques, and scaling it to the point of being capable of making use of a state-of-the-art large-scale computer system. Contribution (3) pertains with the evaluation of the proposed approach with a set of different datasets on the GPU partition of the MeluXina supercomputer [21].

Evaluation considered execution on a single GPU, the four GPUs of a node, and multiple nodes, taking into account datasets with different numbers of SNPs and samples, and different genotype distributions. All in all, the proposed approach achieves significantly higher performance than other exhaustive methods from literature, even when those are rerun on the same hardware (MeluXina's GPU partition).

The paper is structured as follows. Section 2 formulates the problem. Section 3 presents background on TCU-based epistasis detection and the main idea behind the proposed approach. The latter is detailed in the context of a complete approach in Section 4. Experimental results are presented in Section 5. Section 6 positions EPIClear in the context of existing approaches. Finally, Section 7 concludes the paper.

2 Problem Definition

Epistasis detection aims to identify sets of SNPs (genetic markers that represent variation in a single nucleotide of a DNA sequence) that are associated with an observable trait. This analysis is carried out in a case-control dataset, A, of size $(M + 1) \times N$, where M is the number of SNPs and N the number of samples. The entry $A[M + 1, j]$ denotes the phenotypic value for the j-th sample, which is 0 if it does not exhibit the studied trait (control) or 1 otherwise (case).

The genotypic value for the i-th SNP in the j-th sample is given by the entry $A[i, j], i \in \{1, 2, ..., M\}, j \in \{1, 2, ..., N\}$. Each SNP takes one of three genotypic values that result from the possible combinations between the major allele (most frequent in a population) and the minor allele (less frequent

[1] Source code and datasets available at: https://github.com/hiperbio/EPIClear

allele), i.e., homozygous major (genotype 0), heterozygous (genotype 1) or homozygous minor (genotype 2).

State-of-the-art exhaustive methods often employ an internal representation where SNP information for a given sample is represented using three bits. In such representation, which has been first used in [34], the particular SNP genotype of the sample is indicated by setting one (and only one) of those bits. Each SNP can be represented by six bitvectors, half (i.e. three) with as many bits as the number of controls and the other half with a number of bits equal to the number of cases.

To efficiently evaluate an SNP combination, one can use AND operations to combine the data of the involved SNPs, and count the occurrences of the genotypes by applying the POPC operation (population count), which counts the bits set to 1. Fig 1 represents the baseline approach for constructing a contingency table – pertaining to a combination including SNPs X, Y and Z – with one-hot-encoding of SNP data.

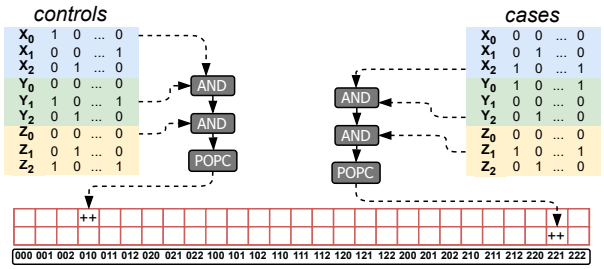

Figure 1: Third-order contingency table construction.

Given a case-control dataset with M SNPs, N samples, and an interaction order K, epistasis detection evaluates $\frac{M!}{K!(M-K)!}$ SNP sets, constructing a contingency table for each set with size 2×3^K to describe the count of each of the 3^K possible genotypes in cases and in controls. Time-to-solution increases exponentially with M and K, and linearly with N.

The objective function used to analyze sets of SNP is the Bayesian K2 score [7]. This function can be described as:

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

which calculates a score based on the occurrences (n_i) of each genotype ($I = 3^K$) in the samples (n_{ij}) with each phenotype ($J = 2$). The lower it is, the stronger the association between the evaluated SNP combination and phenotype. Thus, epistasis detection aims at finding the combination that minimizes the K2 score. If $K = 3$, then $I = 3^3 = 27$ and the contingency table for a given SNP triplet will be of size $2 \times 27 = 54$.

3 Enhancing TCU-based Epistasis Detection

This section focuses on the core algorithmic contribution of this paper. In particular, on the domain-specific method proposed to reduce the operations performed on TCUs, which is responsible for most of the performance improvements achieved in comparison to the state-of-the-art approaches.

3.1 Core operations accelerated on TCUs

As any other exhaustive epistasis detection approach, EPIClear performs searches by reading and combining data from the dataset to perform a statistical test and assess genotype-phenotype correlation. That involves the construction of one contingency table representing genotype counts for each combination of SNPs. A statistical test is performed on each contingency table to produce a score (one per SNP combination). The scores of different SNP combinations are compared to determine which is most statistically associated with the phenotype represented in the processed dataset.

Most epistasis detection approaches are focused on accelerating the construction of contingency tables (i.e. genotype counting), as it accounts for most of the hardware resources used and time-to-solution. This is still the case even for approaches using TCUs. For example, performance has been shown to be independent of using either K2 Bayesian score or mutual information as the scoring function [25].

For epistasis detection, instructions performing bitwise operations can be used to assess how many genotypes exist of any given type for a combination of SNPs, regarding each sample type. The two sample types – cases and controls – are represented in individual matrices and processed separately. This enables processing multiple samples at a time relying on the direct use of bitwise AND and POPC operations.

Combining data (AND) and counting the occurrences of genotypes (POPC) can be performed on CUDA/stream cores (e.g. [26, 31]), and/or on the TCUs in GPUs (e.g. [23–25]). TCUs target applications heavy on matrix multiplication. In addition to fused multiply-add, some modern microarchitectures also have support for tensorized fused bitwise operations such as XOR+POPC (introduced in Turing GPUs [8]) and AND+POPC (introduced in Ampere GPUs[9]).

Current state-of-the-art approaches for high-order searches rely on precombining genotypic data through additional AND operations on the GPU's general purpose cores (i.e., CUDA cores on NVIDIA microarchitectures). For example, in [23] and [25], third-order searches are accomplished by performing a bitwise AND operation between the genotype vectors of an SNP and those pertaining to a block of SNPs. Then, the output of that operation (a binary matrix) is combined with data from another block of SNPs (another binary matrix) from the original internal dataset representation.

In this type of approach, which is that used in state-of-the-art TCU-accelerated methods for exhaustive epistasis detection, the amount of data processed is invariant in relation to the genotypic patterns on the dataset to process. Notice, however, that this does not need to be the case, as

the processing of high-order combinations provides the opportunity of leveraging information from the input dataset.

3.2 Mitigation of redundant operations

In line with the state-of-the-art approaches from literature for third-order searches (e.g. [23, 25]), EPIClear infers the genotype counts for most genotypes (19 out of 27) relying on partial third-order contingency tables and on contingency tables for second-order interactions. For example, the counts for the third-order genotype $\{X_0, Y_0, Z_2\}$ – SNPs X, Y and Z with base genotypes 0, 0 and 2 – can be inferred by subtracting $(\{X_0, Y_0, Z_0\} + \{X_0, Y_0, Z_1\})$ to the occurrences of the second-order genotype $\{X_0, Y_0\}$. Thus, as with other methods, only two out of three genotypes are represented in the data processed by matrix operations. This allows to reduce memory usage and to significantly speed up computations.

Fig. 2 depicts two TCU-based approaches side-by-side, a state-of-the-art method (Tensor-Episdet [25]) and the proposed method (EPIClear). Notice that Tensor-Episdet uses XOR+POPC operations (instead of AND+POPC), the only 1-bit operation on TCUs available at the time on GPU microarchitectures [25]. However, it efficiently derives the output one would get with AND+POPC from XOR+POPC operations, relying on $\text{POPC}(A \cdot B) = \frac{\text{POPC}(A)+\text{POPC}(B)-\text{POPC}(A \oplus B)}{2}$.

EPIClear does not rely solely on bitwise AND operations as the core operation for preparing data for the TCUs. Instead, both input matrices fed to the kernel using the TCUs are compressed. The samples that will certainly not contribute to the count of occurrences of the combined third-order genotypes including a given genotype of SNP X are removed, i.e. the samples with a 0 in the SNP X bitvector representing cases or controls, which are processed separately.

Both EPIClear and Tensor-Episdet use the same type of interleaved binary representation – bitvectors for different genotypes are stored one after the other – with different SNPs represented in different rows and different samples in different columns. As in [25], it is used to improve utilization of the TCUs, with the input matrices to these units representing data from two genotypes of two blocks of SNPs (Y and Z). In Tensor-Episdet, the block of SNPs Y is combined with two genotype bitvectors of an SNP X, resulting in an expanded matrix that is operated on TCUs with another matrix representing SNPs Z. However, in EPIClear both SNPs Y and Z are preprocessed and only one genotype from a given SNP (X) is reflected in the input to TCUs at a given time.

For each SNP X, for a given genotype, only the bit columns representing samples in blocks of data pertaining to SNPs Y and Z that intersect with a 1 in the bit row representing that genotype (of SNP X) are kept. Thus, resulting in two sets of two horizontally compressed bit matrices, which are later to be combined on the TCUs at a significantly smaller

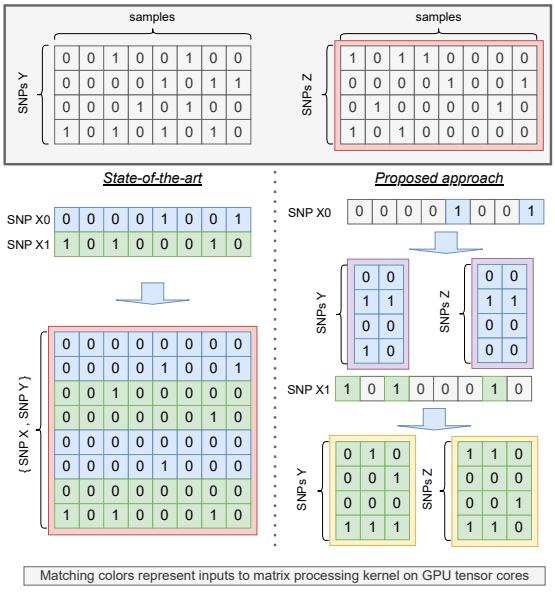

Figure 2: Tensor-Episdet [25] (left) and EPIClear (right) approaches preparing data to be processed on TCUs.

cost. In the example provided in Fig. 2, instead of processing 8×8 and 4×8 matrices ($8 \times 8 \times 4$, i.e. 256 AND+POPC), one processes two 4×2 matrices ($4 \times 4 \times 2$, i.e. 32 AND+POPC) and two 4×3 matrices ($4 \times 4 \times 3$, i.e. 48 AND+POPC).

To minimize the amount of computations, the pruning of samples is performed relying on the two genotypes from SNP X with the lowest counts. If two or more genotypes of a given SNP have an equal number of occurrences, the use of the lowest index genotypes is prioritized (i.e. genotype 0 has priority over genotype 1, and the latter has priority over genotype 2). Calling X_α and X_β the base genotypes from an SNP X used for compressing bit rows representing other SNPs, the example showcased by Fig. 2 represents the situation where $X_\alpha = X_0$ and $X_\beta = X_1$. Notice that the amount of data to be sent to the TCUs for processing will always be reduced and the most challenging genotype distribution is that with $\approx 33.3\%$ occurrence for each of the three genotypes.

4 The EPIClear Approach

To achieve breakthrough levels of performance that are able to significantly surpass the state-of-the-art, EPIClear had to be developed to exploit parallelism at different scales. With that goal in mind, this section explains not only how operations are orchestrated to maximize the benefits of the proposed data pruning method, but also other key aspects to achieving high performance. This includes describing how TCUs are leveraged, as well as the parallelization scheme used to target GPUs in multiple nodes of a supercomputer.

Algorithm 1: Pseudocode for the proposed approach (host side) applied to third-order epistasis detection.

```
   Data: d_0, d_1, M, N_{0|1}, S, B
   Result: s
1  for SY_i = 0; SY_i < M; SY_i += S do
2    for SZ_i = SY_i; SZ_i < M; SZ_i += S do
3      popSYandSZ_{0|1} = pairPop(d_{0|1}, SY_i, SZ_i);
4      for X_i = 0; X_i < SY_i + S; X_i += 1 do
5        pSum_{0|1} = countBitsPrefixSum(d_{0|1}, X_i);
6        compactedSY_{0|1} = compactBits(d_{0|1}, pSum_{0|1}, X_i, SY_i);
7        compactedSZ_{0|1} = compactBits(d_{0|1}, pSum_{0|1}, X_i, SZ_i);
8        popXandSY_{0|1} = pairPop(compactedSY_{0|1}, SY_i);
9        popXandSZ_{0|1} = pairPop(compactedSZ_{0|1}, SZ_i);
10       for Y_i = MAX(SY_i, ⌊X_i/B⌋ · B); Y_i < SY_i + S; Y_i += B do
11         for Z_i = MAX(Y_i, SZ_i); Z_i < SZ_i + S; Z_i += B do
12           andPop_{0|1} = tensorPopAnd(compactedSY_{0|1},
               compactedSZ_{0|1});
13           s = scoringFunction(andPop_{0|1}, popSYandSZ_{0|1},
               popXandSY_{0|1}, popXandSZ_{0|1});
14         end
15       end
16     end
17   end
18 end
```

4.1 Overview of the proposed method

The proposed pruning scheme has been integrated into a full epistasis detection approach targeting modern GPUs with TCUs. Overall, the host orchestrates computations, while the GPU (or GPUs) performs the actual dataset processing related to the search. One of the key aspects of the proposed approach is that the dataset is subdivided at two different levels, to which we call *super-block* (S SNPs) and *block* (B SNPs). Blocks determine the granularity at which the matrix operations combining genotypic data are processed on TCUs, i.e. the size of the matrices used as input to the GPU kernel that uses these processing units. Super-blocks represent the granularity at which most other auxiliary computations are performed. This multilayered data organization is adopted to achieve high utilization of the GPU resources while working within the constraints imposed by the GPU memory capacity.

Algorithm 1 represents the EPIClear approach at a high level from the host (i.e. CPU) side, assuming the dataset is already in GPU memory. It represents execution on a single GPU, being its mapping to multiple GPUs and multiple computer nodes explained in Section 4.4. Some input parameters are abstracted from the function calls, such as the B (block) and S (super-block) parameters. That is also the case for those related to the dataset dimensions – M (number of SNPs) and N_0 / N_1 (number of controls / cases) – which are also used by most device functions of the proposed approach.

The algorithm receives as input the dataset that is to be processed, represented by d_0 (controls) and d_1 (cases), the number of SNPs (M) and controls/cases ($N_{0|1}$), and the super-block (S) and block (B) sizes. Processing is divided into different phases. At the most fine-grained level – to which we call an evaluation round – an SNP X is combined with two SNP blocks (Y_i and Z_i) from two super-blocks (SY_i and SZ_i).

To reduce redundant operations, the combination of blocks from two super-blocks takes into account two different scenarios – the super-blocks can point to different data or to the same data. For example, considering a super-block size (S) of 8192 and a block size (B) of 512, when combining a super-block with itself, there are up to 136 combinations of blocks to evaluate – calculated with the combinations with repetitions formula, choosing 2 at a time from 16 possibilities (= 8192/512). This is different from combining two different super-blocks, in which case there would be 256 combinations to evaluate, as it entails combining each block from one super-block with each block of the other super-block.

Contingency tables for second-order interactions are calculated at the level of combinations between super-blocks (line 3, Algorithm 1) to enable genotype inference from partial third-order tables, allowing the input dataset to have an arbitrarily large number of SNPs. The algorithm includes the construction of pairwise contingency tables resulting from combining a super-block with itself, as one also needs to consider SNP pairings within a given super-block.

For each SNP X – from 0 to SY_i+S-1 – the binary matrices representing controls (d_0) and cases (d_1) on super-blocks (SY_i and SZ_i) are compressed (lines 6-7). The compression needs to be performed efficiently (see Section 4.2) to not represent a significant overhead in relation to processing the resulting SNP blocks in the TCUs. This is accomplished combining efficient kernel code with high reuse of the pruned blocks of data. The latter is improved adopting a super-block size larger than the block size. At the output of this stage, the horizontally compressed super-blocks of SNPs only represent the samples that have the two processed base genotypes for SNP X (cases and controls processed separately).

The compression step involves reading bits representing samples from super-blocks SY_i and SZ_i and copying them to a compacted representation. In the devised parallelization scheme, each GPU thread processes the intersection between 32 consecutive bit columns – representing 32 samples in the uncompressed representation – and bit rows pertaining to multiple SNPs. In order for each GPU thread to know in which position to write in an output array used for storing the compressed representation in global memory, prior to executing the devised compression kernel, we call two GPU kernels in sequence for calculating the prefix sum, at a granularity of 32 bits, across the bit rows representing the two base genotypes processed for a given SNP X, for cases and controls. These two GPU kernels are collectively represented as `countBitsPrefixSum` in the pseudocode (line 5).

The pairwise contingency tables for combinations between an SNP X and super-blocks of SNPs (Y and Z) are constructed counting the bits set in the bit rows represented in the $compactedSY_{0|1}$ and $compactedSZ_{0|1}$ matrices (lines 8-9). Recall that as a result of compression, these only represent the samples where SNP X has a given genotype.

After the construction of the second-order tables, for each combination of SNP blocks indexed by Y_i and Z_i, the corresponding portions of pruned super-blocks are operated on the TCUs (line 12). The same type of dataset representation (SNP/genotype in rows and samples in columns) is used for both inputs to the TCUs, interpreting one as transposed in order to achieve the desired combination between samples of different {SNP, genotype} tuples. Finally, the objective scoring function is calculated (see Section 2) and scores are reduced. Those steps, as well as the genotype inference, are abstracted by the call to `scoringFunction` (line 13 of Algorithm 1).

Notice that in addition to processing for cases and controls (represented by 0|1), the algorithm needs to account for the first and second most frequent base genotypes of SNP X. The pseudocode is abstracting the latter. Calls to routines in lines 5-9 are mapped to twice the amount of kernel calls and TCU operations (line 12) consider eight third-order genotypes.

In order to accelerate the calculation of the objective scoring function, a lookup table with values for `lgamma(x)` – equivalent to the logarithm of $(x-1)!$ – is precomputed by the host. Its size reflects the number of samples in the input dataset. In order to minimize transfers during the actual search, this table is transferred to GPU memory jointly with the dataset at the start of the execution of the application.

At the start of the application, the host also performs a quick scan over the dataset to determine, for each SNP, how many samples have each of the base genotypes. The use of the two base genotypes with lowest counts is implemented reordering the dataset so that genotypes are ordered, individually for each SNP, in increasing frequency. Notice the calculation of the scoring function (see Section 2) is not influenced by the order in which base genotypes are processed.

4.2 High-performance data pruning

The removal of complete bit columns – each representing a different sample – from matrices of genotypic data pertaining to SNPs, effectively making them smaller prior to processing on the TCUs, is at the core of the proposed approach. Thus, it had to be developed targeting a highly efficient operation.

The kernel has been developed to scale with the number of threads, leveraging fine-grained parallelism and efficient memory access patterns. High throughput is achieved using fast bitwise operations, shared memory for temporary storage, and atomic operations for updates. The developed kernel code, which targets modern GPUs, is depicted in Listing 1.

Listing 1: GPU kernel for compressing samples.

```
1  __global__ void compactBits(const unsigned int*
       __restrict__ inMat, unsigned int* __restrict__
       outMat, const int* __restrict__ pSum, int nElemsOrig
       , int snpX, int nElemsComp, int firstSnp) {
2    __shared__ unsigned int sWriteBuf[TB_SIZE * 2];
3    __shared__ int sGlobalWriteStart, sExtraAlign;
4    int startSnpIdx = (firstSnp * 2) + (blockDim.x *
       blockIdx.x + threadIdx.x) * SNPS_PER_THREAD;
5    int elemIdx = blockDim.y * blockIdx.y + threadIdx.y;
6    if (elemIdx < nElemsOrig) {
7      int firstBit = (elemIdx > 0) ? pSum[elemIdx - 1] : 0;
8      if(threadIdx.y == 0) {
9        int firstBitpack = firstBit / 32;
10       sGlobalWriteStart = (((int)(firstBitpack / TB_SIZE)
       )) * TB_SIZE;
11       sExtraAlign = ((firstBitpack & ((TB_SIZE * 2) - 1))
       >= TB_SIZE) ? 1 : 0;
12     }
13     unsigned int xCached = inMat[snpX * nElemsOrig +
       elemIdx];
14     __syncthreads();
15     int globalWriteStart = sGlobalWriteStart;
16     int firstBitShared = (firstBit - (TB_SIZE * 32 *
       sExtraAlign)) % ((TB_SIZE * 2) * 32);
17     for (int snpIdx = startSnpIdx; snpIdx < (startSnpIdx
       + SNPS_PER_THREAD); snpIdx += 1) {
18       sWriteBuf[threadIdx.y] = 0;
19       sWriteBuf[threadIdx.y + TB_SIZE] = 0;
20       __syncthreads();
21       unsigned int x = xCached;
22       unsigned int y = inMat[snpIdx * nElemsOrig +
       elemIdx];
23       int currentBit = firstBitShared;
24       while (x != 0) {
25         int bitPos = currentBit & 31;
26         int sharedIndex = currentBit >> 5;
27         unsigned int leastSignificantBit = __ffs(x) - 1;
28         unsigned int y_and_one = (y >>
       leastSignificantBit) & 1u;
29         atomicOr(&sWriteBuf[sharedIndex], y_and_one <<
       bitPos);
30         currentBit++;
31         x &= ~(1u << leastSignificantBit);
32       }
33       __syncthreads();
34       atomicOr(&outMat[snpIdx * nElemsComp +
       globalWriteStart + threadIdx.y], sWriteBuf[threadIdx
       .y]);
35       atomicOr(&outMat[snpIdx * nElemsComp +
       globalWriteStart + TB_SIZE + threadIdx.y], sWriteBuf
       [TB_SIZE + threadIdx.y]);
36     }
37   }
38 }
```

The kernel processes a matrix (`inMat`) of bit-packs – each represented as an `unsigned int` – and compacts it into another matrix (`outMat`) using a prefix sum array (`pSum`) for alignment and indexing. The latter array has as many positions as the number of 32-bit bit-packs (`nElemsOrig`) representing cases or controls (processed separately) in the uncompressed dataset. The positions from where to extract bits from `inMat` are those corresponding to 1's in an {SNP, genotype} bit row indexed by another kernel input (`snpX`).

The amount of bit-packs required to store bit rows of samples after filtering (`nElemsComp`) is determined at the start of the execution for the genotypes to be processed in regard to each of the SNPs in the dataset. This incurs in small overhead as it is first-order processing done once, while the

search itself is performed in third-order. Furthermore, as this kernel operates at the level of a super-block, it also receives as input the index of the first SNP to process (`firstSnp`).

As hinted by its name, each position of `pSum` stores the amount of set bits up to the corresponding bit-pack on the unfiltered representation (`inMat`). Four of these arrays are efficiently constructed before the calls to the filtering kernel for a given SNP X, taking into account cases and controls and two genotypes. Thus, resulting in 4 calls to the filtering kernel. The construction of each prefix-sum array is performed executing a CUDA kernel that counts the bits set on each bit-pack followed by calling `cub::DeviceScan::InclusiveSum` from the NVIDIA CUB library of GPU-accelerated primitives.

Each thread of the compression kernel starts by using its global index in dimension x of the CUDA grid to determine which SNP it starts processing from `inMatrix` into `outMatrix` (line 4). The second dimension (i.e. `y`) of the CUDA grid (and thread blocks) points to the different bit-packs of the unfiltered dataset (`inMat`). In each thread, the `elemIdx` variable (line 5) indexes a column of bit-packs of the unfiltered dataset from where data pertaining to the multiple SNPs (`SNPS_PER_THREAD`) being processed is to be read.

Coalesced accesses to and from memory are achieved making the second dimension of thread blocks the one that differs between threads. The desired outcome is achieved with a `{1, TB_SIZE, 1}` thread block since in the used representation bit-packs for a given SNP are contiguous in memory.

Each thread gets the index of the bit (`firstBit`) where to store – in the bit rows of the compressed representation – the information on the first sample it processes directly from the prefix sum array (line 7). Then, where to the threads write into global memory is determined by thread 0 – from the index of the first compressed bit-pack (`firstBitpack`) (line 9) to be constructed – using `sGlobalWriteStart` (line 10) to store the aligned starting index for global updates. The latter is stored in shared memory and is later copied to a thread-local variable following thread block level synchronization.

The compressed representation is efficiently constructed relying on shared memory to reduce the amount of global memory accesses. Furthermore, accesses to shared memory are aligned in a way that enables coalesced global memory updates – to make an efficient use of memory bandwidth – when the threads in a thread block complete processing assigned bit-packs from the uncompressed representation.

The efficient implementation of the used shared memory buffering scheme relies on an array with as many elements of type `unsigned int` (each corresponding to a bit-pack) as twice the number of threads in a thread block (`TB_SIZE`). This is to account for when a thread block needs to construct a range of compressed bit-packs that crosses from one chunk of `TB_SIZE` bit-packs to the next. Realignment of shared accesses is conditionally performed to address that (line 11).

Since each thread compresses data pertaining to multiple SNPs, the bit-pack from the uncompressed representation pertaining to the SNP X (and genotype) that is being processed on a given thread is cached from global memory into a variable (`xCached`) local to the thread (line 13). The latter is used to create a copy (`x`) at the start of a `for` loop that iterates through different SNPs (lines 17 to 36). Then, each thread reads the bit-pack to be processed from the uncompressed matrix (line 22), and instantiates an index (line 23) that serves to keep track of the current shared memory array bit being processed (`currentBit`) in a subsequent `while` loop (lines 24 to 32) that is responsible for the actual dataset filtering.

The `while` loop keeps executing until all bits in the bit-pack from SNP X being processed have been taken into account. Variables `bitPos` and `sharedIndex` (lines 25 and 26, both updated based on `currentBit`) represent the position of the current bit inside (i.e. aligned to) a compressed bit-pack and its position in the `sWriteBuf` shared memory array.

The efficient identification of set bits in the bit-packs of a given SNP X is a key factor in achieving a high throughput implementation. This is accomplished using the `__ffs(x)` CUDA intrinsic (line 27), which returns a value between 0 – no bits are set – and 32 (inclusive) representing the position of the first bit set. This circumvents having to iteratively go through all the samples (cases or controls) when copying data from the uncompressed dataset representation.

This kernel relies heavily on bit processing, which is also used to extract the bit indexed by `leastSignificantBit` from bit-pack `y` into `y_and_one` (line 28), as well as to write it to the correct position (`bitPos`) on the bit-pack being constructed at a given time in shared memory (line 29). Finally, each iteration of the `while` loop increases the value of `currentBit` (line 30) and clears the bit processed from `x`, indicating that it has already been handled (line 31).

Closing an iteration of the `for` loop are 2 aligned updates from shared to global memory (lines 34 and 35), with a synchronization (line 33) before to ensure all threads finished updating shared memory. As with shared memory, updates to global memory also rely on atomics (`atomicOr`) to prevent race conditions. This ensures correctness when multiple threads update the same position of the `outMat` array.

For the purpose of compressing a super-block, in regard to cases or controls, the CUDA grid used for instantiating the compression kernel is set as follows. Its first dimension is double the super-block size – to account for two genotypes – divided by the number of SNPs processed per thread, which is a divisor of the super-block size. The second is the rounded up division of the number of bit-packs to process by the number of threads per thread block. The third dimension of the grid is not used and is therefore assigned a value of 1.

4.3 Efficient use of TCUs on modern GPUs

The only requirement to enable mapping to the TCUs of the bitwise operations that construct contingency tables for SNP interactions is to express the combination of SNP data as matrix multiply operations (see Section 3.1). However, to achieve high TCU usage requires addressing other aspects.

Since we are using the information on a given {SNP, genotype} tuple to remove redundant samples from two blocks of SNP data, the two genotypes considered (out of the existing three) have to be processed separately. In comparison to [25], not only both inputs to the TCU-accelerated kernel are smaller in regard to the amount of samples represented (i.e. columns), but the first input also has 2× fewer rows.

One way to address the possible loss of efficiency in the use of TCUs is to adopt a larger SNP block size, i.e. number of SNPs in the inputs to the TCU-accelerated kernel. However, this must be done carefully, as it incurs in an increase of repeated computations. Notice that combining SNPs that are spatially close entails combining a block of SNP data with itself and/or with data from an individual SNP included in it.

To improve hardware utilization without having to increase as much the block size, we rely on batch processing. It provides a means to perform concurrent matrix multiplications, which individually are not able to saturate the GPU compute resources. Batching these operations entails passing to the function using the TCUs the pointers to several pairs of input matrices and to the locations to store the output matrices, i.e. as many pointers as three times the batch size.

We rely on the CUTLASS collection of CUDA C++ template abstractions to implement the routine that combines blocks of SNP data in TCUs using batched execution. CUTLASS supports many of the features in cuBLAS, while providing primitives that enable constructing customized matrix multiplication routines with comparable performance. Notably, in contrast to cuBLAS, it offers support for tensorized bitwise operations (fused AND+POPC), which is particularly valuable for achieving high-throughput epistasis detection.

4.4 Targeting a multi-GPU platform

To process large challenging datasets, it is imperative to make use of the resources available on supercomputing platforms. EPIClear uses the Message Passing Interface (MPI) de facto approach to programming in a multi-node environment.

Achieving high scalability poses additional challenges on how to implement epistasis detection algorithms. On one side, it is important to have a sufficiently fine-grained distribution of work. For instance, if one simply distributed different pairs of super-blocks to different MPI workers, then one would have poor scalability if there is not a large number of super-blocks. However, to have a large number of super-blocks might require a small super-block size, which does not promote high reuse of the output of compression.

An alternative would be to attribute to each GPU not only two super-blocks, from which blocks of SNPs Y and Z are gathered, but also an SNP X. However, SNPs X with large indexes are paired with fewer combinations of blocks Y and Z, which could result in load balancing issues, increasing idle time due to workloads having widely different durations.

We opted to rely on a fixed amount of rounds per workload unit. To that end, we rely on $G + 1$ MPI processes: each of the G processes is attached to a different GPU, and the remaining process distributes work to the former. Dynamic scheduling and asynchronous computing with a parametrizable workload size ensure high utilization and load balance.

The used workload distribution scheme is represented in Fig. 3. Since each work unit includes the processing of a fixed amount of evaluation rounds, each MPI process attached to a GPU – upon completion of the previous workload unit – only needs to request the starting evaluation round configuration (i.e. indexes of an SNP X and of the start of SNPs Y_i and Z_i).

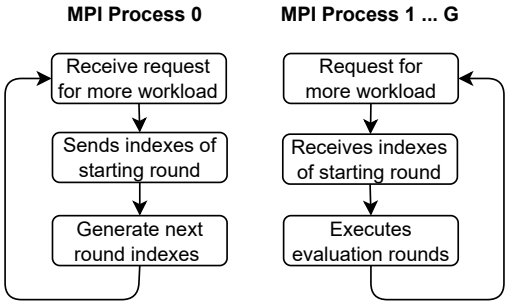

Figure 3: Distribution of workload across processors.

Several optimizations integrated into EPIClear come from the devised parallelization scheme. For example, we reuse the second-order contingency tables that result from combining the SNPs in two super-blocks when subsequent workloads on an MPI worker involve the same pair of super-blocks.

5 Experimental Results

This section presents experimental results for several different parametrizations of the proposed method, considering synthetic and real genotype patterns. We also report a scalability analysis conducted on datasets of different dimensions.

5.1 Targeted system and metric

Experiments have been performed on the MeluXina EuroHPC supercomputer. More specifically, on nodes of its GPU-accelerated partition; each having two AMD EPYC Rome 7452 CPUs and four A100 SXM4 40GB GPUs. The CUDA Toolkit 12.2 and GPU driver 535.161.08 were used.

The kernel that uses the TCUs has been constructed using CUTLASS 3.4.1. In all experiments, we use a matrix multiply-add (MMA) shape of 16×8×256, a warp shape of 64×64×512 and a thread block shape of 128 × 256 × 512. These settings have been experimentally found to achieve high throughput.

For all experiments herein presented, each thread of the pruning kernel processes data for 8 SNPs, as part of a 256 threads thread block, and set the evaluation rounds per workload unit to 2048. Both values have been experimentally determined to enable a good use of the GPUs and explore well the potential of multi-GPU configurations. The block size (B) is set to 512 in all experiments, with the exception of those in Section 6, which also include results with B set to 256.

The main metric of interest is the amount of useful SNP combinations processed per second scaled to the sample size. Scaling to the sample size enables the direct comparison between runs performed for datasets that have different sample sizes and with other methods from the state-of-the-art.

5.2 Datasets used

Experiments were performed with a wide selection of synthetic datasets to show the behaviour of EPIClear under different scenarios. We first show the performance on datasets with close to an even distribution between the three base genotypes. We also consider other ranges of genotype distributions, showing the performance that can be achieved under conditions that are more favorable to EPIClear. Relying on variations with a resolution of ≈ 10%, we experiment with datasets that closely follow the following distributions: (10,10,80), (10,20,70), (10,30,60), (10,40,50), (20,20,60), (20,30,50), (20,40,40) and (30,30,40) – least frequent to most frequent genotype. Finally, we perform experiments with a real Type I Diabetes dataset from the WTCCC1 study [35].

5.3 Effect of the batch size on performance

Batching multiple evaluation rounds into a single call to the routine that uses TCUs provides a means to accelerate performance while keeping GPU memory usage under control. Fig. 4 shows the performance achieved for batch sizes of 1, 2, 4, 8, 16, 32, 64 and 128. The super-block size (S) is set to the number of SNPs (M), since the considered amounts of SNPs are not high enough to require the dataset to be split.

Overall, more samples and/or SNPs results in higher performance. Higher sample sizes essentially allow to make a better use of the TCUs, as it results in a larger input in the inner dimension of the matrix-matrix operations. Handling more SNPs improves the ratio of useful computations, as fewer repeated combinations of SNPs are processed. Given that the block size used is fixed ($B = 512$), an increase in the amount of SNPs to process increases the ratio of useful computations. Notice that, in terms of percentage, the amount of instances of combining a block with itself (i.e. block Y pointing to the same SNPs as block Z) is higher when processing fewer SNPs. The other factor is that reuse of second-order contingency tables and of the operations performed in the pruning stage increases when using larger super-blocks.

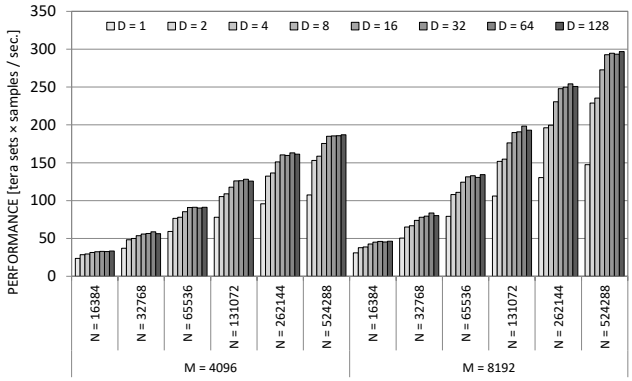

Figure 4: Performance for different numbers of SNPs (M) and samples (N) under different batch sizes (D).

5.4 Impact of different genotype patterns

EPIClear is expected to excel at processing datasets with uneven base genotype distributions. Fig. 5 represents the performance achieved on different genotype distributions, accounting for all instances considering a resolution of 10%.

Processing the datasets with 16384 SNPs and 524288 samples, EPIClear attains an epistasis detection performance of 423.59 (30,30,40), 418.85 (20,40,40), 467.03 (20,30,50), 526.98 (20,20,60), 464.76 (10,40,50), 522.86 (10,30,60), 589.07 (10,20,70) and 679.09 (10,10,80) tera sets / second scaled to sample size.

For the same number of SNPs and samples, the more compressible genotypic distribution, i.e. (10,10,80), consistently results in the highest performance. Notice also that processing datasets with the (30,30,40) and (20,40,40) genotypic distributions results in a similar performance. The same is the case as well for (20,30,50) and (10,40,50), and for (20,20,60) and (10,30,60). This can be attributed to the fact that these pairs entail the same amount of operations on the TCUs.

The effect of the reduction in calculations due to pruning is larger for datasets with more samples. For 16384 SNPs, when processing only 16384 samples, the most compressible genotype distribution, i.e. (10,10,80), results in a performance that is 1.078× (58.468/54.221) higher than that processing the

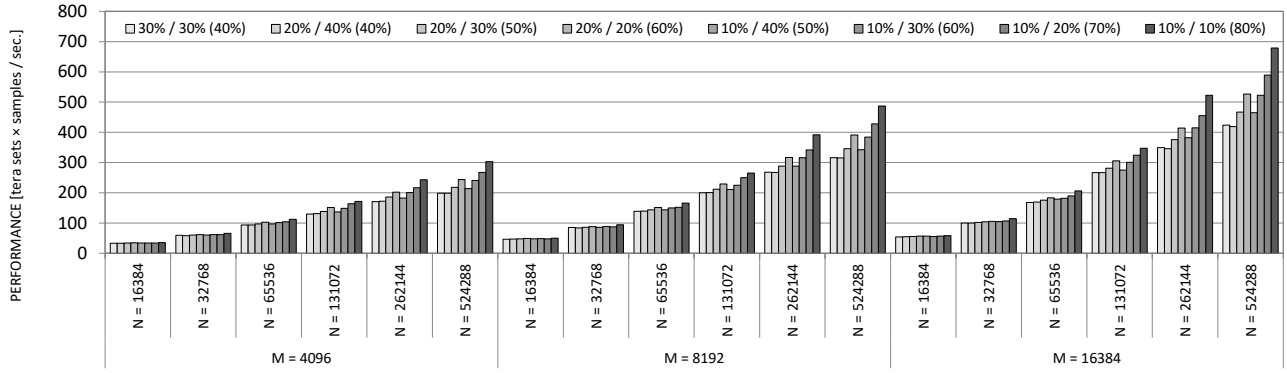

Figure 5: Performance for different numbers of SNPs (M) and samples (N) under different genotype patterns.

(30,30,40) genotype distribution. However, when processing 524288 samples, the most compressible genotype distribution results in 1.603× (679.09/423.59) higher performance. Part of this difference is due to the scoring function calculations. The latter involves a fixed cost in each evaluation round, which becomes further amortized with more samples.

5.5 Scalability on multiple GPUs and nodes

Fig. 6 depicts the performance of EPIClear when different numbers of GPUs are used. For these experiments we considered datasets with 16384, 32768 and 65536 SNPs. When processing the datasets with $M = 32768$ or $M = 65536$, we set S (the super-block size) to half or to one quarter of M (i.e. $S = 16384$ in both cases). A single super-block covering the complete dataset represents the ideal situation in regard to maximizing the reuse of computations, but a super-block with 32768 or 65536 SNPs would require too much memory.

The achieved results show that having a large sample size is not a requirement for high scalability. In fact, when the number of SNPs is small (which is not the case on real datasets), higher scalability can be achieved with a small amount of samples. For example, for 16384 (32768) SNPs, processing 16384 samples results in a speedup of 29.6× (30.1×).

Having more SNPs to process increases scalability, as processing more SNPs means more evaluation rounds to distribute to the targeted GPUs. On average, the performance achieved using 32 GPUs is 1.65×, 1.85× or 1.93× higher than that achieved with 16 GPUs, when processing 16384, 32768 or 65536 SNPs, respectively. Notice that datasets with real genotypic data often have many more SNPs than these datasets.

5.6 Profiling the GPU kernels

Table 1 shows the percentage of cycles the integer MMA pipes are active across the TCUs on the first execution of the kernel that constructs the partial third-order contingency tables, when processing a dataset with 4096 SNPs ($B = 512$) and a number of samples between 16384 and 2097152 (half cases/controls). The considered genotype distributions in these profiled runs are (33,33,33), (30,30,40), (20,20,60) and (10,10,80). Since the particular kernel execution being analyzed is the first one, the volume of data processed on the integer MMA pipes is determined by the most frequent genotype of the distribution (i.e. 33%, 30%, 20% and 10%).

Samples / Occurrence	33%	30%	20%	10%
16384	28.68	28.80	21.87	12.22
32768	44.83	40.37	35.00	21.77
65536	59.22	57.07	48.75	35.31
131072	73.37	71.60	62.84	48.08
262144	83.29	82.21	76.22	62.92
524288	89.61	88.72	85.35	76.26
1048576	93.03	92.55	90.67	85.55
2097152	94.51	94.44	93.49	90.66

Table 1: Percentage of cycles the TCUs are operating to process 4096 **SNPs and different sample sizes.**

Compressing the matrices representing genotypic data affects TCU utilization. For example, when dealing with a genotype with occurrence rate of 10%, 90% of peak utilization was only surpassed with 2097152 samples – recall that half are cases/controls and each kind is processed separately by the TCU-accelerated kernel. The efficiency in the use of the TCUs is smaller when processing fewer samples as a result of the matrix inputs to the CUTLASS kernel being also smaller. Still, the lower performance at the TCUs is largely compensated by processing significantly fewer samples.

Processing a given SNP X in relation to two super-blocks of SNPs entails counting the bits set in each of its bit-packs (`countSetBits`), performing the prefix sum on the resulting array (`DeviceScan`, from the CUB library), compressing samples based on the prefix sum array (`compactBits`),

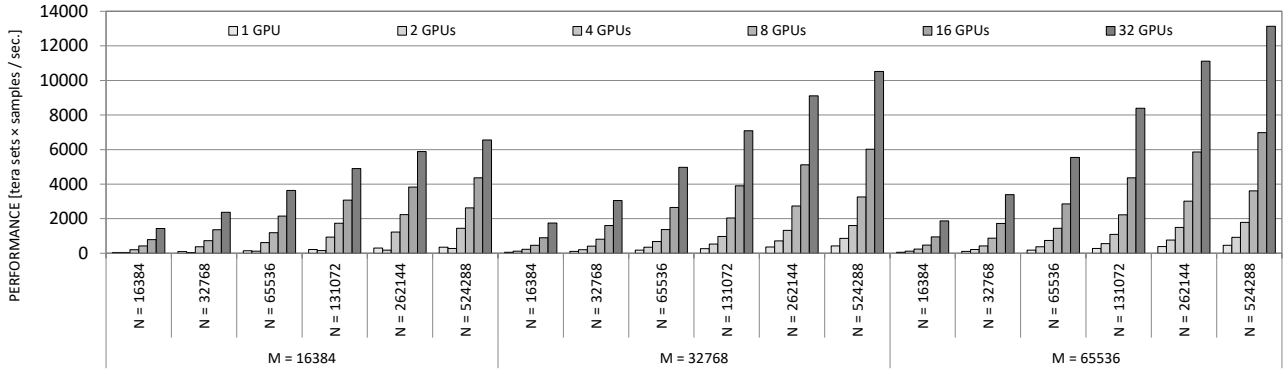

Figure 6: Performance for different numbers of SNPs (M) and samples (N) with 1, 2, 4, 8, 16 or 32 GPUs.

constructing second-order contingency tables from the compressed genotype matrices (`pairPop`), performing batched matrix multiplication (`tensorPopAnd`), and calculating the association scores (`scoringFunction`). The `countBits` and `DeviceScan` kernels are represented in the pseudocode of Algorithm 1 by a call to `countBitsPrefixSum` (see Section 4.1).

Profiling the complete processing of the first SNP X from a dataset with 4096 SNPs and 524288 samples with the (33,33,33) genotype distribution, using $S = M$ and $D = 32$, resulted in a total of 6.14 and 9.19 million clock cycles being used to execute `compactBits` and `tensorPopAnd`. These results indicate it payed off to apply the proposed technique, which resulted in reducing the operations on TCUs by one third. The next kernel with most weight is `scoringFunction` (2.26 million cycles). The remaining kernels take 0.65 (`pairPop`), 0.04 (`DeviceScan`) and 0.02 (`countSetBits`) million cycles.

Nsight Compute also reports a compute throughput of 87% for `tensorPopAnd`, at processing a full batch of matrix operations, and of 84% for the `compactBits` kernel. The former kernel is more affected by the sample size. On a run with 4× the amount of samples (2097152), the throughput for `compactBits` and `tensorPopAnd` is 85% and 91%. However, if processing 131072 (16384) samples, the compute throughput achieved for these kernels drops to 83% (65%) and 74% (33%).

Profiling a dataset with 8192 (16384) SNPs and 524288 samples results in `compactBits` and `tensorPopAnd` taking 12.29 (24.38) and 32.79 (126.03) million cycles to execute, respectively. Notice that the super-block size is set to the number of SNPs in the datset (i.e. $S = M$), and that the block size stayed the same ($B = 512$). Thus, the doubling of the amount of SNPs in the dataset results in having many more combinations of blocks of SNPs to process. For example, the latter is 136 ($=\frac{(16+1)\times 16}{2}$) for 8192 SNPs – the value 16 comes from the fact that the super-block (S) is 16× larger than the block size (B) – and 36 for 4096 SNPs. Thus, resulting in 3.57× the number of cycles for `tensorPopAnd`; while the cost of `compactBits` is only 2× higher (12.29 vs. 6.14 million cycles).

Overall, the efficiency of compression in relation to the other kernels increased when processing larger datasets. The cost of compressing samples – including also the cost of `countSetBits` and `DeviceScan`, which produce the prefix sum array – in relation to that of all other kernels on the profiled evaluation rounds represents 34%, 22% and 13%, on the runs with 4096, 8192 and 16384 SNPs, respectively. Thus, clearly showcasing the trend of the cost of the compression kernel becoming smaller when using larger super-blocks.

5.7 Experiments with real data

As part of the experimental campaign, we aim to assess the expected time-to-solution and the required hardware resources for processing a dataset with real genotypic data. For this purpose, we rely on data from the WTCCC1 study [35], which is highly represented in the epistasis detection literature. The WTCCC1 study represents 500568 SNPs, making it challenging to perform an exhaustive high-order search as there are 20.1 Peta (i.e. $\times 10^{15}$) third-order combinations.

The WTCCC1 study represents seven diseases – coronary artery disease (CAD), hypertension (HT), inflammatory bowel disease (IBD), rheumatoid arthritis (RA), type-1 diabetes (T1D), type-2 diabetes (T2D) and bipolar disorder (BD). We focus all our experiments on a dataset composed with the WTCCC1 type-1 diabetes data (2000 cases) and the 3004 WTCCC1 shared controls from the 1958 British Birth Cohort and UK National Blood Service datasets. As represented in Table 2, which shows the ratio of occurrence of each of the three base genotypes, the cases for the seven diseases in this dataset collection display a similar genotype distribution.

Processing the first 16384 (32768) SNPs of the Type-1 diabetes dataset on a single A100 takes 205.203 (1522.954) seconds, which represents a performance scaled to sample size of 17.87 (19.27) tera sets per second. Since doubling the

Dataset / Genotype	0	1	2
CAD	0.389	0.269	0.342
HT	0.388	0.270	0.341
IBD	0.372	0.271	0.357
RA	0.375	0.272	0.353
T1D	0.381	0.270	0.349
T2D	0.380	0.270	0.350
BP	0.393	0.265	0.342

Table 2: Distribution of base genotypes in WTCCC1.

amount of SNPs to process is expected to increase the exploration time by ≈ 8×, processing all the 500568 SNPs, results in having to process 3566× more combinations than for 32768 SNPs (although the amount of SNPs is only 15.277× more SNPs than 32768). This would result in a time-to-solution of ≈ 2 months. The performance at processing the first 16384 (32768) SNPs decreases to 15.83 (16.97) if not selecting the genotypes to operate for each SNP based on their frequency.

Due to the small amount of samples, WTCCC1 datasets are particularly challenging for matrix-processing based approaches, including others from the state-of-the-art (e.g. Tensor-Episdet [25]). However, as previously pointed out, the small number of samples is not an impediment for EPIClear in achieving high scalability. Since EPIClear scales well with large amounts of GPUs, a more manageable exploration time may be possible, depending on the computing resources available. Representing unprecedented performance at high-order searches, it should be possible to process all datasets resulting from combining the two sets of controls (3004 samples) with the cases for each of the 7 diseases (13795 samples) represented in the WTCCC1 study in just under two weeks ((((3566×1522.954)×((3004×7)+13795)/5004)/32)/604800) on the 32-GPU configuration considered in the experiments.

In addition to considering the original data, we also perform runs using a dataset constructed by replicating the WTCCC1 type-1 diabetes cases and the shared controls by a factor of 100×. This allows to assess the performance at processing larger datasets with a similar genotype distribution.

EPIClear attains much higher performance scaled to sample size on the enlarged WTCCC1 dataset, having achieved 651.02 (317.80 if not fully leveraging genotype patterns) tera SNP sets per second at processing the first 16384 SNPs. The performance improvement from relying on the two genotypes with lowest frequency for each SNP is much more pronounced on the replicated dataset, as the impact of the reduction in TCU operations does not get as much masked by the other stages of the algorithm. On average, on the cases, the two genotypes with lowest frequency (individually for each SNP) have an occurrence of 6.742% and 23.723%. These represent 6.703% and 23.706% of the data for the first controls dataset, and 6.762% and 23.724% for the second controls dataset. The performance achieved is between that for synthetic datasets with (10,10,80) and (10,20,70) distributions.

6 Related Work

Parallel processors have been used for a long time in epistasis detection as a means to tackle the computational requirements of processing challenging datasets. Some approaches target CPUs (e.g. [30]) for their higher interoperability and ubiquitous presence in all systems. Specialized designs in Field-Programmable Gate Arrays (FPGAs) have also been proposed [17, 32]. However, GPUs are the preferred accelerator type for high-throughput epistasis detection, due to combining high availability with suitability for data-parallel processing. In particular, the use of GPUs with TCUs have been shown to achieve the highest performance [16, 23, 25, 31].

CoMet [16] was the first approach to use TCUs for epistasis detection. It targeted the Volta GPU microarchitecture, which was the first to include TCUs. CoMet also supports multiple GPUs. On a node of the Summit supercomputer (each with six V100 GPUs), it achieves a scaled performance of 18.66 tera SNP combinations processed per second on third-order searches. In contrast, EPIClear achieved a performance of 1603.89 tera SNP combinations per second scaled to sample size on a MeluXina GPU-accelerated node – each with four A100 GPUs – at processing a dataset that is representative of the most challenging type of genotype distribution to the proposed compression method. This represents a baseline performance that is, per node, 85.95× higher (128.93×, per GPU) than reported in the CoMet paper [16]. Each V100 is capable of 125 FP16 Tensor TFLOP/s (CoMet processes 0's and 1's as 16-bit floating-point values with fused multiply-add) while the A100 GPU is capable of 4992 INT1 Tensor TOP/s (EPIClear processes 0's and 1's as individual bits with fused AND+POPC). Thus, there is a 39.94× (4992/125) ratio in peak throughput between GPUs at combining genotype data (i.e. contingency table construction). Notice that both methods have been effectively optimized to the targeted GPUs (the CoMet paper reports achieving 82.4% of the peak throughput in core epistasis detection operations per GPU), being the observed performance of EPIClear much higher than that ratio as a result of the employed pruning method.

Tensor-Episdet, first presented in [23] and later improved in [25], was the first approach to use native 1-bit processing capabilities on TCUs. Relying on the XOR+POPC operations added to TCUs on the Turing architecture, it combines genotypes more efficiently than CoMet [16]. As a result, a single Titan RTX running Tensor-Episdet is 2.92× (= 54.54/18.66) faster at third-order searches than a full Summit node (six V100 GPUs) with the former approach. Tensor-Episdet processes third-order combinations through precombining data

on CUDA cores using the bitwise AND operation, using the TCUs to achieve the final combination order. One of the two matrices to be processed on TCUs, i.e. the one resulting from precombining data, will necessarily have many zeros. Thus, resulting in a significant waste of computation resources. In contrast, the proposed approach eliminates the need of performing AND operations on the CUDA cores to precombine data, replacing that step with an efficient pruning stage.

Fig. 7 depicts the performance of Tensor-Episdet on the same GPU used for EPIClear (A100), for datasets with up to 16384 SNPs and with genotypes approximating the (33,33,33) distribution (i.e. evenly distributed). These experiments have been performed on a single A100 GPU, as Tensor-Episdet does not support multiple GPUs. For each of the codes considered, we performed two runs for each dataset, one with an SNP block size set to 256 (the default of Tensor-Episdet) and another with it set to 512. To show that the performance of EPIClear does indeed come from pruning of data, Tensor-Episdet has been ported to better exploit the Ampere architecture (Tensor-Episdet-A). This includes using AND+POPC (instead of XOR+POPC) and relying on the same CUTLASS parametrization as used in the proposed approach.

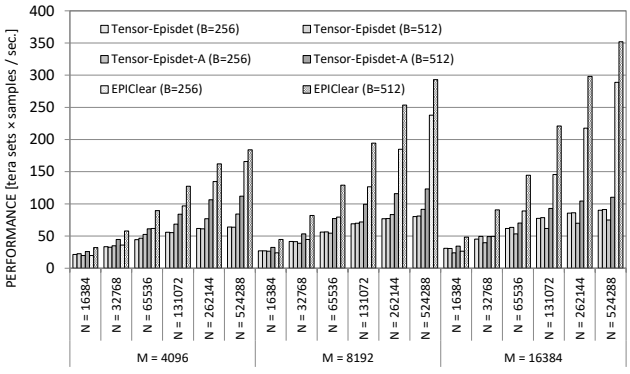

Figure 7: EPIClear and Tensor-Episdet [25] at processing different numbers of SNPs (M) and samples (N).

EPIClear achieves up to 352.06 tera sets per second scaled to sample size at processing the dataset with 16384 SNPs and 524288 samples. Tensor-Episdet achieves a performance of up to 91.39 on that dataset. Highest performance has been achieved with $B = 512$ due to improved hardware utilization.

Most of the 3.85× higher performance achieved with EPIClear can be attributed to the pruning method. Notice that in the case of the datasets used for these experiments, the amount of data processed on TCUs is reduced to one third as a result of the proposed scheme. Compared to the modified Tensor-Episdet version (Tensor-Episdet-A), the EPIClear approach is 3.20× faster for the largest dataset in these experiments (16384 SNPs and 524288 samples). The remaining improvement achieved with EPIClear is due to optimization, including a faster scoring function implementation.

The Tensor-Episdet tool does not natively support datasets as small in regard to sample size as those from WTCCC1 [35]. Using a block size of 512 SNPs – since as we previously showed achieves higher performance than the default one ($B = 256$) for 16384 SNPs – it achieved a performance of 85.01 tera sets processed per second computing the replicated type-1 diabetes dataset. In comparison, EPIClear achieves 7.66× higher performance (651.02), showing – with a genotype pattern from a real dataset – that uneven genotype distributions allow it to excel more in comparison to the state-of-the-art.

The proposed approach is built around a specialized module implemented in software on the targeted parallel processors that leverages domain-specific information to reduce the data processed on its TCUs, without any impact on the quality of the output. TCUs have successfully been used to accelerate combinatorial problems other than exhaustive epistasis detection. This includes applications from domains such as quantum circuit simulation [27], molecular dynamics simulations [10], financial modeling [4] and climate clustering [18]. Thus, there might exist potential for extra performance on some of these applications. In particular, those performing high-order searches operating with highly sparse data.

7 Conclusions

This paper proposed an innovative method, based on domain-specific data pruning, for accelerating genotype counting in the context of high-order epistasis detection searches on TCUs. The method was applied for designing the EPIClear epistasis detection tool. The performance and scalability of EPIClear was evaluated on an EuroHPC supercomputer with NVIDIA A100 GPUs (MeluXina). Compared to a state-of-the-art approach using 1-bit operations on TCUs, it achieved approximately 4× higher performance at processing challenging datasets with a balanced genotype distribution, having potential for even higher speedups been shown processing unbalanced datasets. As a result of the high scalability of the proposed approach, a scaled performance of 13.132 Peta SNP combinations per second has been achieved using 32 GPUs.

Acknowledgments

This work was supported by FCT (Fundação para a Ciência e a Tecnologia, Portugal) through the UI/BD/154603/2022 grant, FCT and EuroHPC Joint Undertaking through the UIDB/50021/2020 project, and grant agreement No 101143931 (POP3). We acknowledge EuroHPC Joint Undertaking for awarding us access to MeluXina at LuxProvide, Luxembourg.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 265–283.

[2] Arash Bayat, Brendan Hosking, Yatish Jain, Cameron Hosking, Milindi Kodikara, Daniel Reti, Natalie A. Twine, and Denis C. Bauer. 2021. Fast and accurate exhaustive higher-order epistasis search with BitEpi. *Scientific Reports* 11, 1 (05 Aug 2021), 15923. https://doi.org/10.1038/s41598-021-94959-y.

[3] Andrew L. Beam, Alison Motsinger-Reif, and Jon Doyle. 2014. Bayesian neural networks for detecting epistasis in genetic association studies. *BMC Bioinformatics* 15, 1 (Nov. 2014), 368. https://doi.org/10.1186/s12859-014-0368-0.

[4] Francois Belletti, Davis King, Kun Yang, Roland Nelet, Yusef Shafi, Yi-Fan Shen, and John Anderson. [n. d.]. *Tensor Processing Units for Financial Monte Carlo.* 12–23. doi:10.1137/1.9781611976137.2 arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611976137.2

[5] C. A. C. Montañez, P. Fergus, C. Chalmers, N. H. A. H. Malim, B. Abdulaimma, D. Reilly, and F. Falciani. 2020. SAERMA: Stacked Autoencoder Rule Mining Algorithm for the Interpretation of Epistatic Interactions in GWAS for Extreme Obesity. *IEEE Access* 8 (2020), 112379–112392. https://doi.org/10.1109/ACCESS.2020.3002923.

[6] Yuanyuan Chen, Fengjiao Xu, Cong Pian, Mingmin Xu, Lingpeng Kong, Jingya Fang, Zutan Li, and Liangyun Zhang. 2021. Epimoga: An epistasis detection method based on a multi-objective genetic algorithm. *Genes* 12, 2 (2021), 191.

[7] 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.

[8] NVIDIA Corporation. 2018. 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-2024].

[9] NVIDIA Corporation. 2020. 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-2024].

[10] Joshua Finkelstein, Justin S. Smith, Susan M. Mniszewski, Kipton Barros, Christian F. A. Negre, Emanuel H. Rubensson, and Anders M. N. Niklasson. 2021. Quantum-Based Molecular Dynamics Simulations Using Tensor Cores. *Journal of Chemical Theory and Computation* 17, 10 (2021), 6180–6192. doi:10.1021/acs.jctc.1c00726 arXiv:https://doi.org/10.1021/acs.jctc.1c00726 PMID: 34595916.

[11] Emmanuelle Génin, Baptiste Coustet, Yannick Allanore, Ikue Ito, Maria Teruel, Arnaud Constantin, Thierry Schaeverbeke, Adeline Ruyssen-Witrand, Shigeto Tohma, Alain Cantagrel, et al. 2013. Epistatic interaction between BANK1 and BLK in rheumatoid arthritis: results from a large trans-ethnic meta-analysis. *PLoS One* 8, 4 (2013), e61044.

[12] Miguel Graça, Ricardo Nobre, Leonel Sousa, and Aleksandar Ilic. 2024. Distributed transformer for high order epistasis detection in large-scale datasets. *Scientific Reports* 14, 1 (2024), 14579.

[13] Fernando Pires Hartwig, Ludmila Gonçalves Entiauspe, Emily Montosa Nunes, Fernanda Martins Rodrigues, Tiago Collares, Fabiana Kömmling Seixas, and Mariângela Freitas da Silveira. 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. https://doi.org/10.1371/journal.pone.0089489.

[14] Cindy Im, Kirsten K. Ness, Sue C. Kaste, Wassim Chemaitilly, Wonjong Moon, Yadav Sapkota, Russell J. Brooke, Melissa M. Hudson, Leslie L. Robison, Yutaka Yasui, and Carmen L. Wilson. 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* 26, 2 (01 Feb 2018), 275–286. https://doi.org/10.1038/s41431-017-0050-x.

[15] Peng-Jie Jing and Hong-Bin Shen. 2014. MACOED: a multi-objective ant colony optimization algorithm for SNP epistasis detection in genome-wide association studies. *Bioinformatics* 31, 5 (2014), 634–641.

[16] Wayne Joubert, Deborah Weighill, David Kainer, Sharlee Climer, Amy Justice, Kjiersten Fagnan, and Daniel Jacobson. 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. https://doi.org/10.1109/SC.2018.00060.

[17] Jan Christian Kässens, Lars Wienbrandt, Jorge González-Domínguez, Bertil Schmidt, and Manfred Schimmler. 2015. High-speed exhaustive 3-locus interaction epistasis analysis on FPGAs. *Journal of Computational Science* 9 (2015), 131–136. doi:10.1016/j.jocs.2015.04.030 Computational Science at the Gates of Nature.

[18] John Lagergren, Mikaela Cashman McDevitt, Veronica G. Melesse Vergara, Paul R. Eller, Joao Gabriel Felipe Machado Gazolla, Hari B. Chhetri, Jared Streich, Sharlee Climer, Peter E. Thornton, Wayne Joubert, and Daniel Jacobson. 2022. Climatic clustering and longitudinal analysis with impacts on food, bioenergy, and pandemics. *Phytobiomes Journal* 6, 3 (9 2022). doi:10.1094/pbiomes-02-22-0007-r

[19] Jie Liu, Guoxian Yu, Yuan Jiang, and Jun Wang. 2017. HiSeeker: detecting high-order SNP interactions based on pairwise SNP combinations. *Genes* 8, 6 (2017), 153.

[20] Yang Liu, Duolin Wang, Fei He, Juexin Wang, Trupti Joshi, and Dong Xu. 2019. Phenotype prediction and genome-wide association study using deep convolutional neural network of soybean. *Frontiers in genetics* 10 (2019), 1091. https://doi.org/10.3389/fgene.2019.01091.

[21] LuxProvide. [n. d.]. *MeluXina.* https://www.luxprovide.lu/meluxina

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

[23] 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)*. https://doi.org/10.1109/IPDPS47924.2020.00043.

[24] Ricardo Nobre, Aleksandar Ilic, Sergio Santander-Jiménez, and Leonel Sousa. 2023. Tensor-Accelerated Fourth-Order Epistasis Detection on GPUs. In *Proceedings of the 51st International Conference on Parallel Processing* (Bordeaux, France) *(ICPP '22)*. Association for Computing Machinery, New York, NY, USA, Article 84, 11 pages. https://doi.org/10.1145/3545008.3545066.

[25] 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. https://doi.org/10.1109/TPDS.2021.3060322.

[26] Ricardo Nobre, Sergio Santander-Jiménez, Leonel Sousa, and Aleksandar Ilic. 2020. Accelerating 3-way Epistasis Detection with CPU+GPU processing. In *23rd Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2020)*. https://doi.org/10.1007/978-3-030-63171-0_6.

[27] Hiryuki Ootomo, Hidetaka Manabe, Kenji Harada, and Rio Yokota. 2023. Quantum Circuit Simulation by SGEMM Emulation onTensor Cores and Automatic Precision Selection. In *High Performance Computing: 38th International Conference, ISC High Performance 2023, Hamburg, Germany, May 21–25, 2023, Proceedings* (Hamburg, Germany). Springer-Verlag, Berlin, Heidelberg, 259–276. doi:10.1007/978-3-031-32041-5_14

[28] A Pandey, NA Davis, BC White, Nicholas M Pajewski, J Savitz, Wayne C Drevets, and Brett A McKinney. 2012. Epistasis network centrality analysis yields pathway replication across two GWAS cohorts for bipolar disorder. *Translational psychiatry* 2, 8 (2012), e154–e154.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates Inc., Red Hook, NY, USA. https://dl.acm.org/doi/10.5555/3454287.3455008.

[30] Christian Ponte-Fernández, Jorge González-Domínguez, and María J. Martín. 2022. Fiuncho: a program for any-order epistasis detection in CPU clusters. *The Journal of Supercomputing* (16 Apr 2022). https://doi.org/10.1007/s11227-022-04477-5 https://doi.org/10.1007/s11227-022-04477-5.

[31] 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.

[32] 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. https://doi.org/10.1109/FCCM51124.2021.00022.

[33] Jiya Sun, Fuhai Song, Jiajia Wang, Guangchun Han, Zhouxian Bai, Bin Xie, Xuemei Feng, Jianping Jia, Yong Duan, and Hongxing Lei. 2014. Hidden Risk Genes with High-Order Intragenic Epistasis in Alzheimer's Disease. *Journal of Alzheimer's Disease* 41, 4 (July 2014), 1039–1056. https://doi.org/10.3233/jad-140054.

[34] Xiang Wan, Can Yang, Qiang Yang, Hong Xue, Xiaodan Fan, Nelson L.S. Tang, and Weichuan Yu. 2010. BOOST: A Fast Approach to Detecting Gene-Gene Interactions in Genome-wide Case-Control Studies. *The American Journal of Human Genetics* 87, 3 (Sept. 2010), 325–340. https://doi.org/10.1016/j.ajhg.2010.07.021.

[35] Wellcome Sanger Institute. [n. d.]. *Wellcome Trust Case Control Consortium*. [Online; visited January-2024].

[36] Zhaoxia Yu, Carey F Li, Haik Mkhikian, Raymond W Zhou, Barbara L Newton, and Michael Demetriou. 2014. Family studies of type 1 diabetes reveal additive and epistatic effects between MGAT1 and three other polymorphisms. *Genes & Immunity* 15, 4 (2014), 218–223.