Parallelizing Epistasis Detection in GWAS on FPGA and GPU-accelerated Computing Systems

Jorge González-Domínguez, Lars Wienbrandt, Jan Christian Kässens, David Ellinghaus, Manfred Schimmler, and Bertil Schmidt

Abstract—High-throughput genotyping technologies (such as SNP-arrays) allow the rapid collection of up to a few million genetic markers of an individual. Detecting epistasis (based on 2-SNP interactions) in Genome-Wide Association Studies is an important but time consuming operation since statistical computations have to be performed for each pair of measured markers. Computational methods to detect epistasis therefore suffer from prohibitively long runtimes; e.g. processing a moderately-sized dataset consisting of about 500,000 SNPs and 5,000 samples requires several days using state-of-the-art tools on a standard 3GHz CPU. In this paper we demonstrate how this task can be accelerated using a combination of fine-grained and coarse-grained parallelism on two different computing systems. The first architecture is based on reconfigurable hardware (FPGAs) while the second architecture uses multiple GPUs connected to the same host. We show that both systems can achieve speedups of around four orders-of-magnitude compared to the sequential implementation. This significantly reduces the runtimes for detecting epistasis to only a few minutes for moderately-sized datasets and to a few hours for large-scale datasets.

Index Terms—GWAS, epistasis, pairwise gene-gene interaction, contingency tables, parallel computing, FPGA, GPU

1 Introduction

Recent advances in high-throughput genotyping technologies allow the collection of hundreds of thousands to up to a few million genetic markers, such as Single Nucleotide Polymorphisms (SNPs), from individual DNA samples in only a few minutes. In Genome-Wide Association Studies (GWAS) these genotypes are typically measured for thousands of individuals and linked to a given phenotype. The simplest and most common phenotype classification is a binary trait, i.e. the presence (case) or absence (control) of an associated disease. By simply determining the genotype frequencies between cases and controls, as in classical GWAS, associations between a genetic disease and specific markers can be made. However, the separate identification of individual markers with differences in genotype frequencies between cases and controls is generally not powerful enough to detect joint genetic effects (epistasis). Hence, higher-order statistical methods considering the combination of at least two markers are necessary to model complex disease traits based on epistasis [1], [2]. Consequently, a number of algorithms have been developed to detect 2-SNP epistatic interactions in high-throughput GWAS datasets (e.g. [3], [4], [5]). The main goal of these approaches is to find pairs of SNPs whose joint genotype frequencies show a statistically significant difference between cases and controls which potentially explains the effect of the genetic variation leading to disease.

Computing epistasis is highly time-consuming due to the large number of pairwise tests to be calculated. For example, even for a moderately-sized dataset consisting of 500,000 SNPs there are about 125 billion pairwise interaction tests to be performed. This extensive analysis of all pairwise SNP combinations might require several hours or days for moderately-sized datasets as in [6]. Large-scale datasets even take weeks or months on standard computing platforms. Since both the availability and size of GWAS datasets are increasing rapidly, finding faster solutions is of high importance to research. In this paper we address this problem by using a combination of both coarse-grained and fine-grained parallelism on a multi-FPGA and a multi-GPU accelerated computing system. We base our approach on the popular BOOST [7] method which efficiently performs an exhaustive pairwise analysis of all SNP combinations of a dataset using statistical regression. We introduce a fast implementation of the BOOST filter by using FPGA technology on the RIVYERA architecture. Furthermore, an efficient novel implementation of this filter on GPU

- *J. González-Domínguez and B. Schmidt are with the Institute of Computer Science, Johannes Gutenberg University Mainz, Germany.*
  E-mail: {j.gonzalez,bertil.schmidt}@uni-mainz.de
- *L. Wienbrandt, J. C. Kässens, and M. Schimmler are with the Department of Computer Science, Christian-Albrechts-University of Kiel, Germany.*
  E-mail: {lwi,jka,masch}@informatik.uni-kiel.de
- *D. Ellinghaus is with the Institute of Clinical Molecular Biology, Christian-Albrechts-University of Kiel, Germany.*
  E-mail: d.ellinghaus@ikmb.uni-kiel.de

architectures is presented. Although the GBOOST tool already presents a significant improvement of BOOST using GPUs, our method is still faster by a factor of at least 1.8 with both methods tested on two different NVIDIA GPU systems (Tesla K20m and GTX Titan). Our design is also able to utilize several GPUs on the same system, showing high scalability at least up to 4 GPUs.

Taking advantage of both fine-grained parallelism (by using reconfigurable FPGA hardware and a specific hardware description) and coarse-grained parallelism (by using several FPGAs in parallel) our reconfigurable implementation utilizing 128 low-cost Spartan6-LX150 FPGAs on the RIVYERA architecture further reduces the runtime compared to the multi-GPU system. This results in a speedup compared to the original BOOST implementation on a 3GHz CPU of about four orders-of-magnitude.

The rest of the paper is organized as follows. Section 2 reviews some related work. Section 3 summarizes the methodology to find epistasis that has been adapted in our work. Section 4 describes the utilized FPGA and GPU technologies. The parallel GWAS implementations for FPGAs and GPUs are explained in Sections 5 and 6, respectively. The performance of both approaches is evaluated in Section 7. Finally, Section 8 concludes the paper.

2 Related Work

Targeting the problem of detecting epistasis in GWAS with novel computing architectures such as clusters, cloud computing or GPUs can help to speedup the process to become acceptable in a typical biologists workflow. Tools performing an exhaustive analysis on standard PC architectures include BOOST [7], MDR [8], MB-MDR [9], and iLOCi [10]. The underlying algorithms can be implemented on GPU systems for a significant runtime reduction, resulting in tools such as GBOOST [11], BOOST2 [12], SHEsisEpi [13], GWIS [14], EpiGPU [15] and others. Nevertheless, none of them is able to exploit multiple GPUs for the same analysis. Other tools such as SIXPAC [16], SNPRuler [17], SNPHarvester [18], TEAM [19] and Screen and Clean [20] apply pre-filtering techniques to reduce the number of analyzed SNP-pairs to perform a selective test only on a subset of all pairwise combinations. They can reduce runtimes at the expense of potentially losing some significant SNP combinations.

As mentioned in the previous section, our approaches are based on the statistical regression methods available in BOOST [7]. BOOST further applies a pre-filtering technique to all SNP-pairs called the Kirkwood Superposition Approximation (KSA) filter before performing a 2-test with four degrees of freedom. This approach has been proven to be accurate through cross-validation of exhaustive bivariate analysis [21]. Moreover, this tool is widely used in biomedical research to perform GWAS analyses in practice (see for instance [22], [23], [24]). Furthermore, statistical filters and methods of different tools could also be applied in our approach by merely changing the internal statistical tests, and without modifying the data distribution, optimization techniques or other features related to the parallel implementation.

Both our multi-FPGA and multi-GPU solutions consist of two main parts. Firstly, the parallel creation of contingency tables, and secondly, the application of the KSA and log-linear filters as statistical tests. Since the creation of contingency tables is a common task in many GWAS algorithms, our method can also be applied to other tools by simply replacing the filters. For example, the novel EDCF tool [25] does pre-filtering with a simple 2-test with two degrees of freedom on all pairwise SNP combinations to collect candidates for higher-order interactions. Our implementation could easily be adapted to this pre-filtering technique leading to a possible speedup of EDCF in the same order as for BOOST.

Although, up to our knowledge, we present the first comparison between a multi-FPGA and multi-GPU implementation to detect epistasis, the efficiency of these architectures to solve some other types of bioinformatics algorithms have been previously studied. These can be categorized into two areas: sequence alignment [26], [27], [28] and structural bioinformatics [29]. Chen et al. [26] compare a hybrid FPGA and CPU multicore implementation of a short read aligner with a GPU-based approach. Programmability and performance comparisons between these two architectures using, among others, pairwise sequence alignment and molecular docking are presented in [27], [28], [29]. The reported results show that both FPGA and GPU architectures are able to obtain high speedups compared to CPU multicore approaches. FPGAs usually obtain better performance than GPUs for these applications at the cost of more difficult programmability.

3 Background

3.1 Contingency Tables

A typical GWAS dataset consists of two groups of samples (cases and controls) which are genotyped at a set of marker positions (SNPs). Contingency tables are created for each SNP-pair separately for case and control group. Since SNP-pairs can be viewed as symmetric, $n(n-1)/2$ SNP-pairs have to be analyzed in an exhaustive test where n denotes the total number of SNPs. In this work we focus on GWAS datasets with biallelic markers as it is the common use case, i.e. genotypes may appear as homozygous wild (w), heterozygous (h) or homozygous variant (v) type. Hence, in pairwise interaction tests each contingency table for cases and controls has the dimension 3×3, one entry for each possible combination of genotypes.

TABLE 1
A contingency table for cases and controls for the SNP-pairs (A,B). n_{ijk} reflects the count for the corresponding genotype combination in the current SNP pair.

controls ($k=0$)		**SNP A** w	h	v
SNP B	w	n_{000}	n_{010}	n_{020}
	h	n_{100}	n_{110}	n_{120}
	v	n_{200}	n_{210}	n_{220}
cases ($k=1$)		**SNP A** w	h	v
SNP B	w	n_{001}	n_{011}	n_{021}
	h	n_{101}	n_{111}	n_{121}
	v	n_{201}	n_{211}	n_{221}

The entries reflect the number of occurrences each combination of genotypes appears in the dataset for the corresponding SNP-pair in either case or control group. See Table 1 for an example. The cells of the contingency table (for $i, j = 0, 1, 2$; $k = 0, 1$) can also be filled with probabilities: $\pi_{ijk} = n_{ijk}/n$.

3.2 The BOOST Filters

For our parallel implementations we use the same statistical tests as in BOOST [7]. This tool measures the interaction between two markers via log-linear models. We use the dot convention to indicate summation over a subscript. For instance, $\pi_{i\cdot\cdot} = \sum_{jk} \pi_{ijk}$ is the marginal probability of the first SNP to be equal to i. The notation extends to two dimensions as well; e.g. $\pi_{ij\cdot} = \sum_k \pi_{ijk}$ is the marginal probability of having values i and j for the first and second SNP, respectively.

Wan et al. [7] define the interaction as the difference $\hat{L}_S - \hat{L}_H$ between the log-likelihoods of the homogeneous association model and the saturated model. They prove that this difference is proportional to the Kullback-Leibler divergence of $\hat{\pi}_{ijk}$ and $\hat{p}_{ijk}$, whereby $\hat{\pi}_{ijk}$ is the joint distribution obtained under the saturated model and $\hat{p}_{ijk}$ is the distribution obtained under the homogeneous association model:

$$\hat{L}_S - \hat{L}_H = n \cdot \sum_{ijk} \left[\hat{\pi}_{ijk} \cdot \log\left(\frac{\hat{\pi}_{ijk}}{\hat{p}_{ijk}} \right) \right] \tag{1}$$
$$= n \cdot D_{KL}\left(\hat{\pi}_{ijk} || \hat{p}_{ijk} \right) \tag{2}$$

They establish that all pairs with their log-linear measure higher than a certain threshold τ show epistasis. Although this log-linear model is affordable, there exists no closed-form solution for the homogeneous association model. Thus, the authors in [7] propose the Kirkwood Superposition Approximation (KSA) for $\hat{p}_{ijk}$

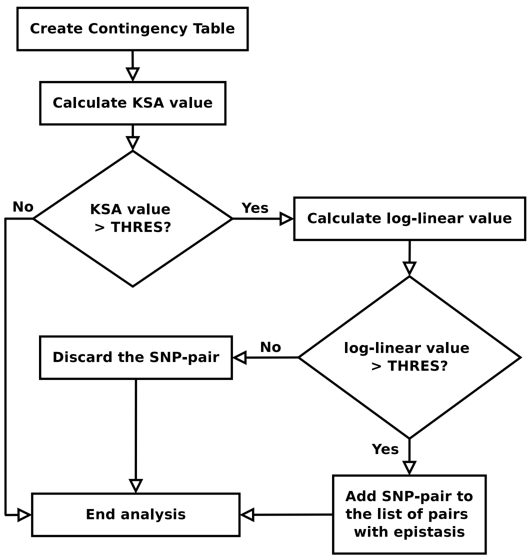

Fig. 1. Workflow of the tests applied to each SNP-pair

$$\hat{p}^K_{ijk} = \frac{1}{\eta} \frac{\pi_{ij\cdot}\pi_{i\cdot k}\pi_{\cdot jk}}{\pi_{i\cdot\cdot}\pi_{\cdot j\cdot}\pi_{\cdot\cdot k}} \tag{3}$$

whereby

$$\eta = \sum_{ijk} \frac{\pi_{ij\cdot}\pi_{i\cdot k}\pi_{\cdot jk}}{\pi_{i\cdot\cdot}\pi_{\cdot j\cdot}\pi_{\cdot\cdot k}} \tag{4}$$
$$= \sum_{ijk} \frac{n_{ij\cdot}n_{i\cdot k}n_{\cdot jk}}{n_{i\cdot\cdot}n_{\cdot j\cdot}n_{\cdot\cdot k}}. \tag{5}$$

It can be shown that with the replacement of $\hat{p}_{ijk}$ by $\hat{p}^K_{ijk}$ in (2) the resulting difference $\hat{L}_S - \hat{L}_{KSA}$ is an upper bound of the interaction defined by $\hat{L}_S - \hat{L}_H$; .i.e. it holds:

$$\hat{L}_S - \hat{L}_H \leq \hat{L}_S - \hat{L}_{KSA} \tag{6}$$

The equations above show that the KSA value can be directly calculated from the cells of the contingency table without the need for iterative methods. Therefore, BOOST accelerates its analysis using the KSA filter ($\hat{L}_S - \hat{L}_{KSA}$). From now, we call the value of $\hat{L}_S - \hat{L}_{KSA}$ for a specific SNP-pair its "KSA value". As the KSA value is an upper bound of the log-linear measure, we calculate it for all SNP-pairs and discard those where the following equation does not hold:

$$2\left(\hat{L}_S - \hat{L}_{KSA} \right) \geq \tau \tag{7}$$

where τ is a user-defined threshold.

Finally, BOOST only applies the log-linear filter to the remaining pairs. Figure 1 summarizes the workflow of the application. We refer to [7] to find the proofs and further explanation of the KSA and log-linear filters.

4 Technology

4.1 FPGA Technology

FPGA technology offers a flexible way of introducing very fine-grained on-chip parallelism. Compared to CPUs the functionality of an FPGA is configured by an application specific description of the underlying hardware instead of running a program consisting of a sequence of commands from a pre-defined instruction set. The hardware description is typically written in a special programming language such as Verilog or VHDL. It usually consists of a set of freely designable processing elements (PEs) which may have different functionality but can be utilized concurrently. The number of PEs and their functionality is only limited by the available resources of the FPGA. These resources typically consist of lookup tables (LUTs) where almost any logical or arithmetic function can be mapped into, registers, local RAM (Block RAM) and embedded hardware units such as digital signal processors (DSPs) which may directly be used for certain purposes.

4.2 RIVYERA S6-LX150 Architecture

The computing platform RIVYERA was originally developed for problems related to cryptanalysis. It was first introduced to bioinformatics in 2008 [30]. For this work, we have used the specific model RIVYERA S6-LX150. Its basic structure consists of two parts, a multi-FPGA system and a server grade mainboard with standard PC components acting as host system. The FPGA system consists of up to 16 FPGA modules with 8 Xilinx Spartan6-LX150 FPGAs each (with an optional upgrade possibility to 16 FPGAs per module). Each FPGA is connected to 512 MB DDR3-SDRAM divided into two modules. The host system runs a Linux operating system on two Intel Xeon E5-2620 CPUs (6 cores @ 2 GHz each) with 128 GB of RAM.

The bus system is organized as a systolic chain, i.e. each FPGA on an FPGA module is connected by fast point-to-point connections to both neighbors forming a ring. One additional member of this ring acts as communication controller. It provides the interconnection of each module to its neighboring modules and, on the first module, the uplink to the host via PCI Express. An application developer uses an API to transparently communicate between host and FPGAs. Besides regular point-to-point transmissions, the API provides broadcast facilities and methods for configuring the FPGAs. The design structure of the RIVYERA S6-LX150 system is shown in Figure 2.

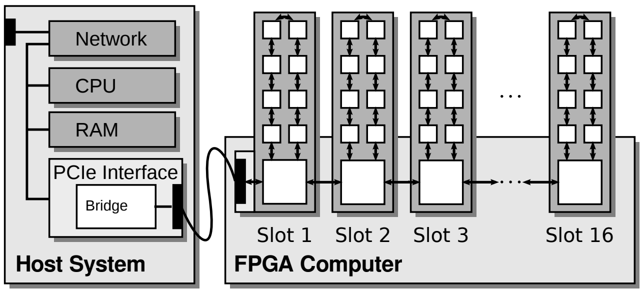

Fig. 2. The RIVYERA S6-LX150 system.

4.3 CUDA and GPU Technology

CUDA is a parallel programming language that extends the general programming languages, such as C/C++ and Fortran, with a minimalist set of abstractions to express parallelisms. A CUDA program is comprised of code for the host and kernels for the devices. A kernel is a program launched over a set of lightweight parallel threads on GPUs, where the threads are organized into a grid of thread blocks. All threads in a thread block are split into small groups of 32 parallel threads, called warps, for execution. These warps are scheduled in a single instruction, multiple thread (SIMT) fashion. Full efficiency and performance can be obtained when all threads in a warp execute the same code path. It is the programmers' responsibility to limit the amount of thread divergence.

CUDA-enabled GPUs have evolved into highly parallel many-core processors with tremendous compute power and very high memory bandwidth. They are especially well-suited to address computational problems with high data parallelism and arithmetic density. A CUDA-enabled GPU can be conceptualized as a fully configurable array of Scalar Processors (SPs). These SPs are further organized into a set of Streaming Multiprocessors (SMs) under three architecture generations: Tesla [31], Fermi [32] and Kepler [33].

Figure 3 shows the structure of the GPU architecture. The fastest type of memory are the registers, whose data is visible only to the SM that wrote it. In the Kepler generation each SM comprises 192 CUDA SP cores sharing a configurable 64 KB on-chip memory. The on-chip memory is slower than registers but faster than global memory, and it is divided into shared memory and L1 cache. It can be configured at runtime as 48 KB shared-memory with 16 KB L1 cache, 32 KB shared-memory with 32 KB L1 cache, or 16 KB shared-memory with 48 KB L1 cache, for each CUDA kernel. Finally, the largest but slowest type of memory is global memory, whose loads can only be cached in L2 cache and the 48 KB read-only data cache.

5 Mapping onto the FPGA-Accelerated Computing System

5.1 Creation of Contingency Tables

The first step of the application is the parallel creation of contingency tables, which is required as input for

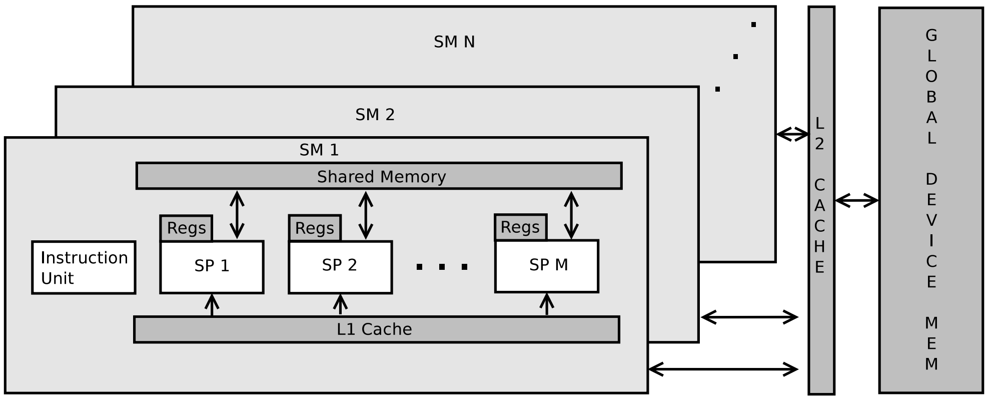

Fig. 3. GPU architecture and memory hierarchy.

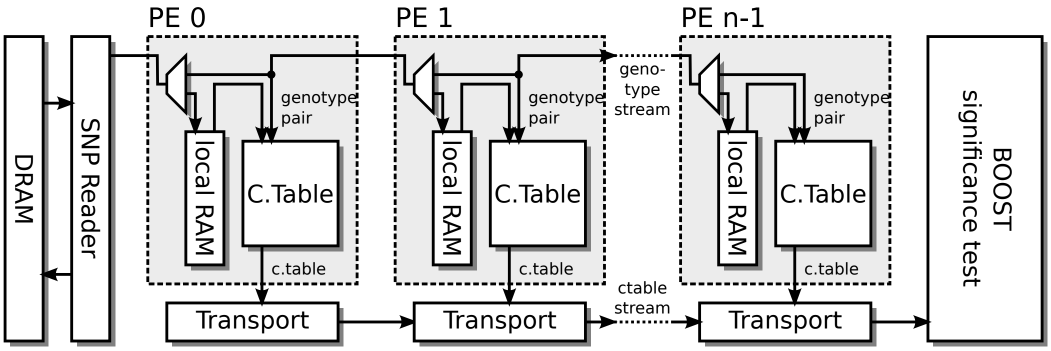

Fig. 4. Overview of the systolic chain of PEs for contingency table creation.

the subsequent KSA filter. We have designed a chain of PEs (further referred to as PE-chain) with a systolic-like dataflow (see Figure 4). The raw data has to be organized in genotypes grouped by cases and controls for each SNP. Each FPGA processes two intervals of SNPs which are stored in the local memory of the FPGA (see Section 5.3 for more details about the overall distribution of SNPs).

Each PE contains three main components, a local RAM to store all genotypes of one SNP, the required counters of one contingency table, and a bus transfer unit to send a completed contingency table to the unit performing the subsequent statistical test.

The SNP data is streamed in several iterations from the FPGA RAM to the first PE in the chain. The first SNP arriving at each PE is simply stored in the local RAM. Any further SNPs are streamed to the next element in the chain. This way all k PEs contain the first k SNPs in their local RAM where k denotes the total number of PEs. The contingency table in each PE is created while streaming SNP data to the next PE. Each genotype of the data in the stream is compared to the corresponding genotype of the SNP stored in the local RAM and the appropriate counter of the contingency table is incremented. When all genotypes of either cases or controls of one SNP have been streamed, the contents of the contingency table are provided via the bus transfer unit to a FIFO buffer collecting all tables from all PEs, and the local counters are reset. The calculation of the KSA value starts the moment the tables for cases and controls of a SNP-pair are complete (see Section 5.2). After all SNP data from both intervals have been streamed, the next iteration begins. Each iteration starts streaming all SNPs from both intervals omitting the first k SNPs of the previous iteration and all SNPs which were omitted before as well. The process is finished when all SNPs of the first interval have to be omitted. This way the partitioning of SNP data into two intervals for each FPGA creates the base for an efficient distribution of SNPs among all FPGAs. Each FPGA computes the statistics for all possible SNP pairings in the first interval and all pairings between the first and the second interval.

Figure 5 illustrates how this process works for a PE-chain with three PEs on a small dataset with six SNPs. The distribution of SNPs among FPGAs is discussed in Section 5.3.

5.2 KSA Filter Pipeline

The FPGA implementation of the KSA filter is designed in a completely unrolled pipeline allowing to get the result of one test every 18 clock cycles (i.e. the number of entries in a contingency table) after a certain latency. In order to achieve an efficient FPGA implementation using the available resources, we have transformed the calculation of $\hat{L}_S - \hat{L}_{KSA}$ as

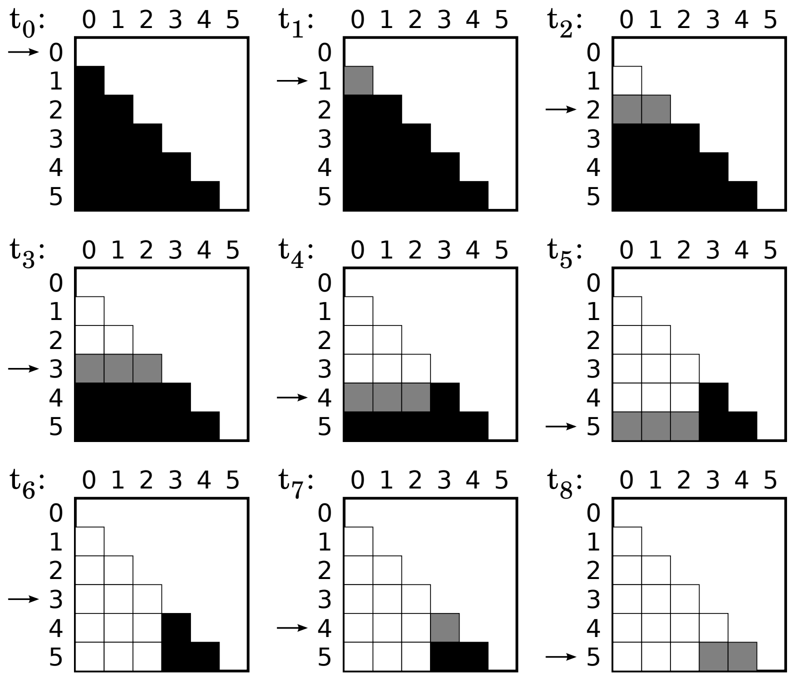

Fig. 5. Sequence of processing an example dataset of six SNPs with a chain of three PEs in nine time steps. Black squares indicate SNP pair combinations to be processed while white squares indicate already processed pairs. Grey squares are currently being processed while an arrow on the vertical axis indicates the currently streamed SNP.

follows:

$$\hat{L}_S - \hat{L}_{KSA} = n \cdot D_{KL}\left(\hat{\pi}_{ijk} || \hat{p}^K_{ijk}\right) \tag{8}$$

$$= n \sum_{ijk} \left[\hat{\pi}_{ijk} \log \frac{\hat{\pi}_{ijk}}{\hat{p}^K_{ijk}} \right] \tag{9}$$

$$= n \sum_{ijk} \left[\frac{n_{ijk}}{n} \log \frac{\frac{n_{ijk}}{n}}{\frac{1}{\eta}\frac{\pi_{ij\cdot}\pi_{i\cdot k}\pi_{\cdot jk}}{\pi_{i\cdot\cdot}\pi_{\cdot j\cdot}\pi_{\cdot\cdot k}}} \right] \tag{10}$$

$$= \sum_{ijk} \left[n_{ijk} \log \frac{n_{ijk}}{\frac{n}{\eta}\frac{n_{ij\cdot}n_{i\cdot k}n_{\cdot jk}}{n_{i\cdot\cdot}n_{\cdot j\cdot}n_{\cdot\cdot k}}} \right] \tag{11}$$

$$= \sum_{ijk} \left[n_{ijk} \left(\log \frac{n_{ijk} n_{i\cdot\cdot} n_{\cdot j\cdot} n_{\cdot\cdot k}}{n_{ij\cdot} n_{i\cdot k} n_{\cdot jk}} + \log \frac{\eta}{n} \right) \right] \tag{12}$$

$$= \sum_{ijk} \left[n_{ijk} \log \frac{n_{ijk} n_{i\cdot\cdot} n_{\cdot j\cdot} n_{\cdot\cdot k}}{n_{ij\cdot} n_{i\cdot k} n_{\cdot jk}} \right] + n\left(\log \eta - \log n\right) \tag{13}$$

Equation (13) is directly implemented into the hardware description of the FPGA with the following optimizations. Firstly, the accumulations $n_{\cdot jk}$ and $n_{i\cdot k}$ are calculated on-the-fly and stored into separate FIFOs while receiving n_{ijk} from the PE-chain. The sums $n_{ij\cdot}$, $n_{\cdot j\cdot}$ and $n_{i\cdot\cdot}$ can then be calculated using a simple adder resource each at the corresponding FIFO outputs. Secondly, a divider resource is required to calculate η. It is easy to see, that each fraction in (5) is between zero and one (if none of the factors in the denominator is zero). Hence, an efficient divider unit is implemented which generates a fixed point format with 32 fraction bits. The special case, if one of the factors in the denominator is zero, directly triggers a division-by-zero error and regards the calculated value as insignificant.

To save further resources, only two logarithm units are implemented. Both units iteratively generate a fixed point format with 8 integer and 56 fraction bits. The first one directly calculates the logarithm of a fraction (i.e. $\log \frac{a}{b}$) to save another divider unit. The second calculates $\log \eta$. $\log n$ is a constant and can be precalculated.

This way after several latency cycles, each clock cycle produces one summand of the left part in (13) and one summand of η in (5). After 18 cycles the accumulations are complete and the result of the whole equation is obtained and compared to a user definable threshold. Of each significance value passing the threshold the identifier to the corresponding SNP pair is stored in a FIFO. Due to the pipeline nature of this design, the calculation of each significance value can be started every 18 clock cycles. There is no need to ever block the data flow, resulting in very fine-grained concurrent processing and therefore very efficient utilization of FPGA resources.

Since the log-linear filter still has to be applied to the SNP-pairs passing the KSA filter, the CPU concurrently fetches the corresponding IDs from the

FPGA FIFOs and stores them in a local buffer. The items in the buffer are submitted to a number of CPU worker threads calculating the necessary log-linear tests. In most cases, (i.e. if the threshold is not set too low) the FPGA processing time dominates the CPU runtime. Therefore, the log-linear filter computation does usually not influence the overall execution time.

The log-linear filter is not implemented on the FPGA for two reasons. Firstly, the additional resources would only be used very infrequently, but would reduce the number of available PEs for the cost of concurrency. Since the number of iterations in the log-linear test is unpredictable, they would potentially create pipeline stalls increasing the runtime as well. Secondly, the CPU is idle during the FPGA processing. Hence, its resources can positively be utilized to concurrently calculate log-linear tests.

5.3 Distribution of Data

Due to the limited amount of memory available on each FPGA, it is generally not possible to store the complete input data in their memory modules. In addition, the speed of the bus system connecting the FPGAs and the host is limited and there is no implicit shared memory. Thus, a scheme is required to reduce the traffic between host and FPGAs. This implies that a dynamic workload distribution approach, as described for the GPUs in Section 6.3, cannot be applied for the FPGAs efficiently, since it would require continous traffic on the RIVYERA bus. Our solution therefore uses broadcast to transfer the complete input data only once for initialization while each FPGA picks only those parts of data that it requires for the whole run. Since in our FPGA design each SNP pair requires the same amount of runtime, a well-balanced workload can be accomplished by equally distributing the number of SNP-pairs to be processed on each FPGA. We achieve this by assigning two intervals of input data to each FPGA as described in the following.

The number of SNP-pairs each FPGA processes directly results from the sizes of the two intervals of SNPs. We set the size of the first interval of FPGA i to a_i. The size of the second interval is then calculated by $\sum_{j=0}^{i-1} a_j$ starting with $a_0 = 0$. To calculate a_i the total number of SNP-pairs $n(n-1)/2$ is equally distributed over all FPGAs resulting in $n(n-1)/(2t)$ SNP pairs to be processed by each FPGA (with t denoting the number of available FPGAs). Then, the following equations hold for $i > 0$:

$$\frac{n(n-1)}{2t} = \frac{a_i^2}{2} + a_i \sum_{j=0}^{i-1} a_j \tag{14}$$

$$\Leftrightarrow \quad 0 = a_i^2 + 2a_i \sum_{j=0}^{i-1} a_j - \frac{n(n-1)}{t} \tag{15}$$

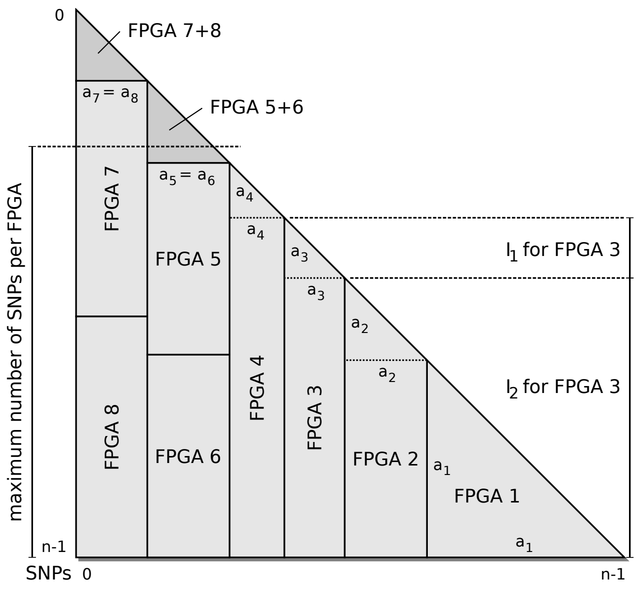

Fig. 6. Example SNP distribution among 8 FPGAs. The two intervals I_1 and I_2 for SNP processing of FPGA 3 are exemplarily marked.

$$\Leftrightarrow \quad a_i = -\sum_{j=0}^{i-1} a_j + \sqrt{\left(\sum_{j=0}^{i-1} a_j\right)^2 + \frac{n(n-1)}{t}} \tag{16}$$

It is likely for larger datasets, that for some FPGAs the calculated SNP intervals do not fit into the local FPGA memory. In this case, the second interval is split over multiple FPGAs and the interval sizes are recalculated considering the adjusted size of the second interval. Unfortunately, it has to be taken into account that the SNP pairs from the first interval are then processed multiple times. This amount of redundant calculations has to be considered for the total SNP distribution. Furthermore, redundant results need to be removed.

The particular SNPs are mapped directly to the calculated interval sizes and are assigned to the corresponding FPGAs for an equal workload. Figure 6 illustrates an example distribution.

If the total number of SNPs exceeds the maximum possible number of SNPs that can be distributed among the FPGAs, the whole dataset is partitioned. Each partition is then processed iteratively.

6 Mapping onto the GPU-Accelerated Computing System

6.1 CUDA Kernel

Instead of separating the creation of the contingency tables and the filters, a single kernel that performs the whole analysis of a set of SNP-pairs is developed. Therefore, the values of the contingency tables do not need to be saved in the device memory, as they are discarded as soon as the tests are performed. The SNP information is stored in the device pinned memory previously to any kernel call. The GPU computation is divided into batches, i.e. the CPU calls the GPUs several times to analyze all the pairs by batches. In every kernel call:

- The kernel receives the first pair of the batch as well as the batch size as input.
- Depending on this size, each thread calculates the KSA filter for several pairs within each kernel call. Consecutive threads analyze consecutive SNP-pairs.
- If the pair passes the KSA filter, the thread also computes the subsequent log-linear filter.
- The kernel output is a binary value for each pair that indicates whether it presents epistasis or not.

Unlike the FPGA implementation, our GPU approach does not implement the rearranged equation of the KSA filter, but the same way as it is done in GBOOST. Moreover, when assigning the GPU resources to the different parts of the code, we give the highest priority to the KSA filter. Therefore, this filter is implemented mainly using registers and avoiding direct accesses to the device memory.

Depending on the results of the KSA filter, GPU threads that test pairs discarded by this preliminary filter would be idle while other threads are performing the log-linear filter. For instance, in a scenario where the probability of a SNP-pair passing the KSA filter is 0.01, 99% of threads would finish their computation in the kernel after the KSA filter, but for the remaining 1% all threads in the respective warps have to wait. GBOOST addresses this thread divergence problem by performing the log-linear filter on the CPU. Although this approach eliminates CUDA thread divergence, it significantly decreases performance if a large percentage of SNP-pairs passes the first filter. An alternative solution would be the distribution of the computation into two different kernels: the first one for the generation of the contingency tables and the KSA filter (performed for all SNP-pairs), and the second kernel for the log-linear filter. However, the overhead of copying the corresponding contingency tables to global memory between kernels would cause a significant performance overhead. Therefore, our multi-GPU approach maintains only one kernel with the whole computation. The experiments shown in Section 7.2 prove that our solution is superior to the GBOOST approach.

6.2 Reordering Information for Coalesced Accesses

For a dataset with n SNPs from m individuals, the naive representation uses an $n \times m$ table where each row represents one SNP and each column represents one sample. Each element needs 2 bits to distinguish among the three genotypes ({w,h,v}). GBOOST applies a boolean representation of genotype data in order to calculate the values of the 18 cells of the

TABLE 2
Values of the contingency tables calculated with logical AND operations on the GPUs

controls		**SNP A**		
($k = 0$)		w	h	v
SNP B	w	n_{000}	-	n_{020}
	h	-	-	-
	v	n_{200}	-	n_{220}

cases		**SNP A**		
($k = 1$)		w	h	v
SNP B	w	n_{001}	-	n_{021}
	h	-	-	-
	v	n_{201}	-	n_{221}

contingency tables in a fast manner. Three rows per SNP are used, each one consisting of two-bit strings, one for the case samples and the other for the controls. Each bit in the string represents one sample, and its value indicates whether the individual has the corresponding genotype. The main advantage of this approach is that the creation of the contingency table can be performed using only logical AND operations. See [7] for more details.

We modify this approach for our GPU implementation by reducing the information loaded into the GPU memory and, thus, increasing the size of the datasets that fit in the device memory. For each SNP, only strings for the homozygous wild (w) and the homozygous variant (v) are created. Moreover, for each SNP the information is packed into two arrays with entries of 32 bits (one for the "w" and one for the "v" genotypes). The length of each array is $m/32$ entries. Using only logical AND operations we explicitly calculate 8 cells of the contingency table (shown without "-" in Table 2). Additionally, when loading the datasets, the sums of all the "w" and "v" biallelic values are calculated per SNP. These "sums" can then be used to calculate the remaining cells of the table if needed by the statistical filters. There are two advantages of this modification. Firstly, we reduce the device memory consumption compared to the GBOOST approach, as we only store 8 of the 18 cells during the kernel. Moreover, performance is also reduced because only 8 complete AND operations must be always performed. Although some filters may need to calculate some of the other cells, they only need to apply two arithmetic operations to integer values (very efficient on GPUs) instead of the whole AND operation to all the individual values.

Figure 7 shows how each one of the two arrays would look like if the entries of each SNP were consecutively ordered. In most cases consecutive threads also analyze consecutive SNPs. However, as there are $m/32$ entries per SNP (and m might be very large, in the order of several thousands of samples), we would generate uncoalesced memory accesses on the GPU because consecutive threads would access positions of the arrays with distance $m/32$. Thus, we reorder the data of the two arrays when loading it into pinned device memory, following the structure shown in Figure 8. In this case consecutive threads can

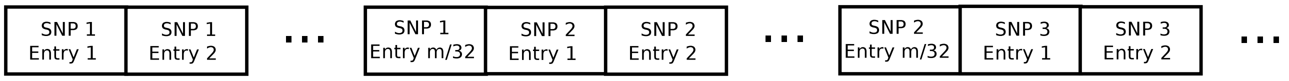

Fig. 7. Example of one array with the information of one SNP without reordering the entries.

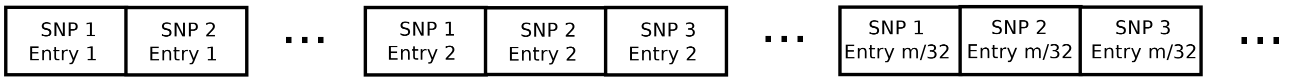

Fig. 8. Example of one array with the information of one SNP when reordering the entries.

also access consecutive memory positions, increasing the coalescing of the accesses and, thus, significantly improving performance.

6.3 Dynamic Work Distribution among GPUs

As explained in Section 6.1, the analysis of the SNP-pairs is divided into batches. Every time the CUDA kernel is launched it performs the filters for a certain batch, specified by the initial pair and the batch size. Unlike previous GPU-accelerated GWAS tools, our implementation is able to utilize several GPUs within the same system. The generation of the batches and their assignment to the GPUs is dynamically performed by the CPU. Figure 9 illustrates the CPU/GPU interactions when working on a system with 2 GPUs. The CPU starts assigning a different initial batch of SNP-pairs to each GPU and launches the kernel. Once a GPU completes the filter of all the pairs within the batch, it returns the pairs with interaction to the CPU and requests another batch. The CPU controls the assignment of pairs to GPUs in a flexible way, since the GPUs can finish their computation in any order. At the end of the computation the CPU stores a list with all SNP-pairs showing epistasis.

This dynamic distribution is of importance especially for systems with different types of GPUs. As, the most powerful GPUs finish their computation earlier than slower ones, they can request more batches from the CPU. Therefore, the most powerful GPUs analyze more SNP-pairs, maximizing their resource utilization. On homogeneous systems all GPUs perform the same amount of analyses and the performance would be similar than using a static distribution that assigns the batches to the GPUs in advance.

As for the FPGA version, our GPU implementation is able to analyze very large datasets, even if they do not fit in the device memory. In these cases the CPU splits the information and, while assigning batches to the GPUs, it also keeps trace of which SNPs are stored in the memory of each GPU. If the information necessary to compute a batch is not available in GPU memory, the CPU transfers the remaining part.

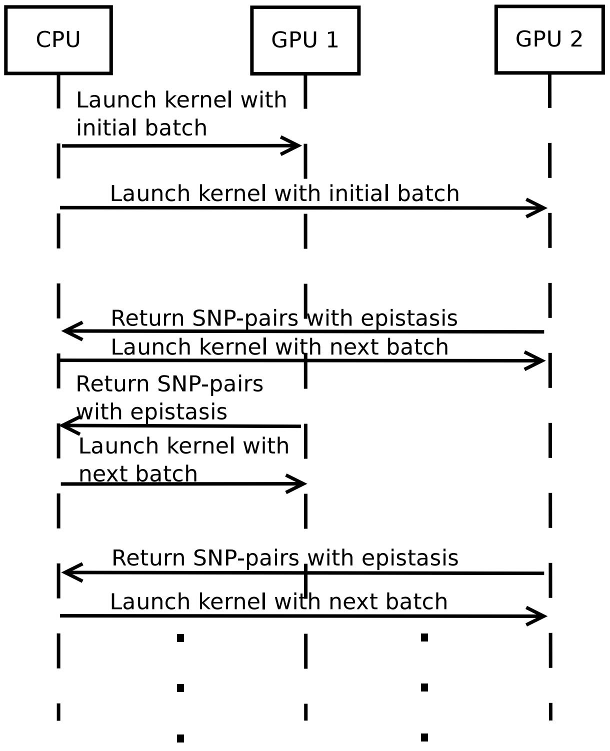
CPU
GPU 1
GPU 2
Launch kernel with initial batch
Launch kernel with initial batch
Return SNP-pairs with epistasis
Launch kernel with next batch
Return SNP-pairs with epistasis
Launch kernel with next batch
Return SNP-pairs with epistasis
Launch kernel with next batch

Fig. 9. Interactions between the CPU and two GPUs to perform GWAS.

7 Performance Evaluation

7.1 Experimental Setup

The complete FPGA design has been implemented on a Spartan6-LX150 using Xilinx ISE 14.7 and the VHDL programming language. Half of the resources are assigned to the creation of the contingency tables and the other half to compute the KSA filter. Limited by the FPGA resources, we have managed to implement a PE-chain with 56 PEs. The clock frequency for the PEs as well as for the KSA filter is 133 MHz. The host software including the log-linear filter has been implemented using the C++ programming language and the gcc compiler v4.4.7. The design has been tested on a RIVYERA S6-LX150 system as described in Section 4.2. The runtimes shown in this section also include the log-linear filter on the CPU.

The multi-GPU version has been tested on two types of compute nodes. Both nodes have a dual

TABLE 3
Performance comparison of different approaches when looking for epistasis in the WTCCC dataset. Results obtained from the corresponding publications are marked with (*). The results for CPU versions are estimated from a smaller dataset (**)

Design	Architecture	Runtime	Speed (10^6 pairs/s)	Energy consumption (kWh)
FPGA	RIVYERA S6-LX15	6 m	348.01	0.07
multi-GPU	4 GTX Titan	9 m	208.81	0.15
multi-GPU	4 Telsa K20m	12 m	139.20	0.18
multi-GPU	1 GTX Titan	35 m	56.43	0.15
multi-GPU	1 Tesla K20m	47 m	38.67	0.18
GBOOST	1 GTX Titan	1 h 15 m	34.23	0.31
GBOOST	1 Tesla K20m	1 h 26 m	24.28	0.32
EpiGPU*	1 GTX 580	2 h 55 m	11.90	0.71
SHEsisEPI*	1 GTX 285	27 h	1.29	5.51
PThreads**	Intel Core i7-3930K (6 cores)	19 h	1.82	2.47
BOOST**	Intel Core i7-3930K (1 core)	7 d	0.21	21.84

octa-core Intel(R) Xeon(R) E5-2670 CPU (16 cores) with a clock rate of 2.60 GHz. Each node further contains four Kepler-based GPUs with the following characteristics:

- NVIDIA Tesla K20m with 5 GB of memory, 208 GB/s of device memory bandwidth, and 2,496 CUDA cores (grouped into 13 SMs) running at 706 MHz.
- NVIDIA GTX Titan, which has 14 SMs (2688 CUDA cores) with a clock rate of 875.5 MHz. Device memory is of size 6 GB and can be accessed with a bandwidth of up to 288 GB/s.

The GPU kernel has been compiled with the CUDA v5.5 based on the gcc compiler v4.6.2.

7.2 Performance Comparison

Firstly, we tested our designs with the real-world WTCCC dataset [34] with 3,004 controls from the 1958 British Birth Cohort and 2,005 cases with bipolar disorder (BD) genotyped at 500,568 SNPs. We have initially compared the calculated significance values from our implementations against the original BOOST. As expected, our multi-GPU approach obtains the same results, as it uses exactly the same equation. Nevertheless, due to our imprecise calculation using fixed-point arithmetics on the FPGAs, we expected a slight deviation of our results from the original. However, the total error of the KSA filter score has never been greater than 0.01. After application of the exact log-linear test on the CPU, the output list of SNP-pairs with espistasis on the RIVYERA is identical to the one returned by GBOOST.

Table 3 shows the performance of our implementations on the described platforms, specifying the runtime and the speed in terms of the number of analyzed pairs per second. It also includes the total power consumption. Regarding the related work, GBOOST has been executed utilizing only one GPU per node, as it does not support computation for multiple GPUs. Furthermore, we have included additional results for two other GPU-based tools (EpiGPU [15] and SHEsisEPI [13]) in the table, obtained from the corresponding publications. Finally, we have measured the execution time for CPU-based approaches. On the one hand, we have run BOOST on an Intel Core i7 analyzing a smaller simulated dataset (40,000 SNPs and 5,009 individuals) with the same ratio of pairs with potential epistasis as the WTCCC dataset. Assuming quadratic increase of runtime with the number of SNPs, we have estimated that BOOST would need more than 7 days to process the WTCCC dataset. On the other hand, we have implemented an efficient PThreads version of the whole algorithm (creation of contingency tables, KSA and log-linear filters). We have followed the same approach as for BOOST to estimate that this implementation would need more than 19 hours to process the WTCCC dataset using the 6 cores of the Intel Core i7 machine.

Additionally, we have simulated a dataset with 2M SNPs and 10,000 individuals in order to test the behavior of the parallel designs for large-scale GWAS. The results shown in Table 4 are obtained with a threshold of $\tau = 28$ for the filters, which generates a similar percentage of SNP-pairs with potential epistasis as the default threshold ($\tau = 30$) for the WTCCC dataset. GBOOST runtimes are not included because this tool is not able to analyze such a large dataset due to out-of-bounds problems in the internal arrays.

The experimental results further show that our CUDA implementation is faster than GBOOST. The improvement of the CUDA kernel and the device memory accesses (increasing coalescence) make our approach 1.83 and 2.14 times faster than GBOOST when analyzing the WTCCC dataset for the Tesla K20 and the GTX Titan, respectively. Although results for EpiGPU and SHEsisEPI must be treated carefully since the comparison is performed on older architectures, we can infer that they are significantly slower

TABLE 4
Performance comparison of different approaches when looking for epistasis in a simulated dataset with 2M SNPs and 10,000 individuals.

Design	Architecture	Runtime	Speed (10^6 pairs/s)	Energy consumption (kWh)
FPGA	RIVYERA S6-LX15	3 h 07 m	178.25	2.19
multi-GPU	4 GTX Titan	4 h 08 m	134.41	4.13
multi-GPU	4 Telsa K20m	4 h 43 m	117.79	4.25
multi-GPU	1 GTX Titan	16 h 02 m	34.65	4.01
multi-GPU	1 Tesla K20m	18 h 39 m	29.79	4.20

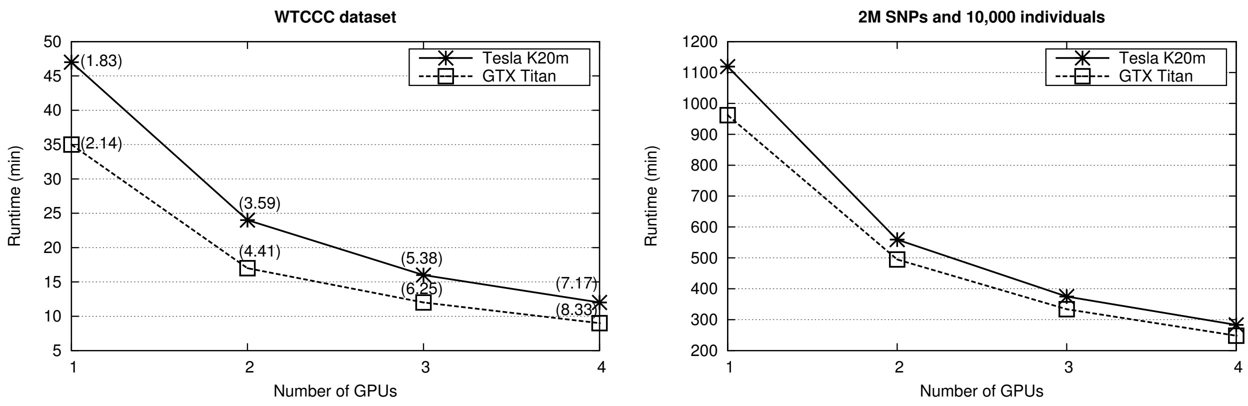

Fig. 10. Runtimes of the multi-GPU version for a varying number of GPUs on the nodes with Tesla K20m and GTX Titan GPUs. The label of each point shows the speedup compared to GBOOST (running on the respective single GPU).

than any of our designs. An additional advantage of our multi-GPU design is that it is able to work with the four GPUs available on each node to significantly decrease the runtime. The speedups over GBOOST of our multi-GPU implementation are 7.17 and 8.33 using four Tesla K20 and four GTX Titan, respectively. Figure 10 shows the runtimes and scalability of our multi-GPU implementation depending on the number of employed GPUs. Speedups over GBOOST are included in brackets for the WTCCC dataset. The achieved parallel efficiency on four GPUs is higher than 90%.

The runtimes of our multi-GPU implementation with four GTX Titan (9 minutes for the WTCCC dataset and about 4 hours for the large simulated dataset) are still higher than the execution time of our FPGA counterpart on the RIVYERA S6-LX150 system (6 minutes and about 3 hours, respectively). The corresponding speedups are 1.5 and 1.3. According with the estimation of the CPU-based BOOST runtime shown in Table 3, the FPGA version obtains a speedup of more than 1,680 against a 3 GHz CPU. Speedup over the efficient Pthread implementation that uses the 6 cores is still higher than 190. Furthermore, the energy efficiency of the FPGA version is around 2 times higher than the best multi-GPU implementation.

Although our FPGA implementation targets the RIVYERA system, it can generally be adapted to any FPGA platform assuming it provides enough resources to implement the KSA filter and DRAM to store a significant part of the input data. In addition, our distribution scheme is not limited to a specific number of FPGAs. The length of the chain of PEs is only limited by available resources. Due to the multitude of available FPGA types and platforms it is impossible to provide a performance estimation for all of them. Hence, we have kept at the Spartan6 FPGAs of the RIVYERA system to produce a representative result. However, we have inspected the ability to map our design to a more up-to-date FPGA of the Xilinx' Kintex7 series. A Kintex7-480T offers more than three times more LUT resources than a Spartan6-LX150, and due to a high amount of integrated DSPs and the new 28 nm manufacturing technology we expect a doubling of the available clock frequency. Therefore, we estimate that a system equipped with 128 FPGAs of this type would reduce the runtime of pairwise SNP analysis by a factor of around 6. Although portability of VHDL code may be difficult in general, our design uses generic variables for parameters such as the number of PEs etc. Therefore, the adaption to a Kintex7-FPGA would require only top-level interface changes.

7.3 Scalability Analysis

We have simulated additional datasets with different numbers of SNPs and individuals in order to

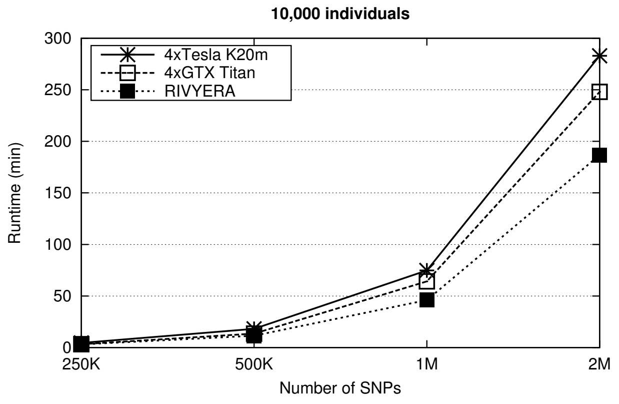

Fig. 11. Runtimes for datasets with 10,000 individuals and varying number of SNPs.

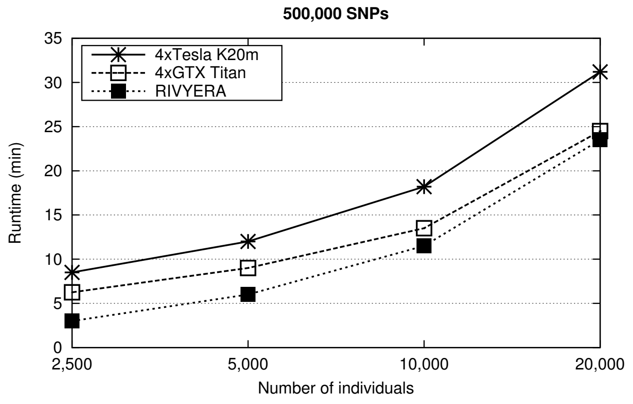

Fig. 12. Runtimes for datasets with 500,000 SNPs and varying number of individuals.

study the behavior of our parallel implementations depending on these parameters. Figure 11 shows the scalability in terms of number of SNPs. The number of individuals is fixed to 10,000 and the threshold to $\tau = 28$, in order to obtain a percentage of SNP-pairs with potential epistasis similar to the WTCCC dataset. As the number of pairs to compute is $n(n-1)/2$, a linear increase of the number of SNPs leads to a quadratic increase of the number of pairs. Hence, as expected, this reflects in a quadratic increase of runtime.

Figure 12 shows the runtimes for datasets with 500,000 SNPs, varying the number of individuals, and the same percentage of SNP-pairs with potential epistasis. In this case the increase of runtime is linear and, the more individuals are in the dataset, the more significant this increase is. The graph shows that the slope of the FPGA line is higher than for the multi-GPU. This means that the performance of the FPGA version is more affected by the number of individuals. The total runtime can be modeled by the addition of the time for calculating the contingency tables and for filtering: $T = T_{tab} + T_{fil}$. The filtering part (T_{fil}) does not depend on the number of individuals. Therefore, only the creation of the contingency tables (T_{tab}) is influenced by the number of individuals. The increase of runtime is more significant on the FPGAs than on the GPUs. This indicates that T_{tab} takes a lower percentage of time on the latter architectures.

7.4 Impact of Thread Divergence

Finally, we have analyzed the impact of the thread divergence on the multi-GPU runtimes by varying the threshold of the filters. The lower the threshold, the more pairs pass the KSA filter and need to be evaluated with the log-linear filter. Table 5 shows the percentage of interactions found by our implementations depending on the value of τ. Table 6 compares the speedup over GBOOST of our multi-GPU design for different thresholds. Using a threshold lower than 20 makes no sense for this dataset as the output would have so many pairs with potential epistasis that it would be unfeasible for biologists to further analyze all of them. The speedups are higher for lower thresholds (more computations of the log-linear filter). It proves that, although our approach presents thread divergence due to gathering both filters in a single CUDA kernel (see Section 6.1), it outperforms GBOOST, which needs to transfer the contingency table to global memory in order to compute the log-linear filter on the CPU.

TABLE 5
Percentage of pairs with potential epistasis for the WTCCC dataset depending on the threshold.

Threshold τ	Percentage of pairs with potential epistasis
20	8.28×10^{-2}
30	1.13×10^{-3}
40	9.22×10^{-5}
50	4.59×10^{-5}

8 Conclusion

Recent advances in high-throughput genotyping technologies establish the need for fast implementations of statistical epistasis in GWAS. In this article we describe parallel implementations for FPGA and GPU-accelerated systems that increase the computational speed and allow us to analyze large-scale input datasets in reasonable time. Both designs exhaustively measure the interaction of all SNP-pairs by creating their contingency tables and by subsequently applying statistical tests based on regression models. Specifically, they apply the KSA and log-linear filters present in (G)BOOST. This tool has been determined as one of the fastest available GWAS method and presents high accuracy. Nevertheless, since contingency table creation is common to most epistasis tools, our implementations can also be used to accelerate a large variety of tools by simply interchanging the statistical test.

TABLE 6
Speedups of the multi-GPU implementation over GBOOST for different thresholds using the WTCCC dataset.

Threshold τ	Tesla K20	4×Tesla K20	GTX Titan	4×GTX Titan
20	2.18	8.57	2.44	9.52
30	1.83	7.17	2.14	8.33
40	1.80	6.75	2.11	8.22
50	1.78	6.67	2.11	8.22

Although other tools already support GPU computation, our CUDA design is optimized by transforming the kernel and reordering the information to increase the coalescing of the memory accesses. Moreover, up to our knowledge, we have designed the first GWAS tool that is able to run on multiple GPUs. Our FPGA design uses a workload distribution that is able to exploit the resources of hundreds of FPGAs. Additionally, it adapts the formula of the KSA filter that allows designing a pipeline which computes the result of one test every 18 clock cycles.

The presented parallel designs have been evaluated using a real-world dataset from the WTCCC association with 500,568 SNPs and 5,009 individuals. While (G)BOOST needs more than 7 days on an Intel Core i7 CPU and 75 minutes on a NVIDIA GTX Titan GPU, our multi-GPU implementation is able to analyze this moderately-sized dataset in only 9 minutes running on four GTX Titan GPUs. Morever, our version is more than two times faster than GBOOST even using only one GPU. The FPGA version is even faster and only needs 6 minutes for this GWAS analysis on a RIVYERA S6-LX150 system with 128 Xilinx Spartan6-LX150 FPGAs. Furthermore, we simulated a large-scale dataset with 2M SNPs and 10,000 samples. The multi-GPU implementation completely analyzed it in 4 hours and 8 minutes using the four GTX Titan GPUs. The FPGA-based version reduced this time to 3 hours and 7 minutes on the RIVYERA system. Finally, our implementations are as accurate as (G)BOOST, as they always obtain the same list of SNP-pairs with potential epistasis.

Since both architectures (GPU and FPGAs) have their benefits as well as drawbacks, our future research includes investigating a hybrid approach where tasks are distributed among both architectures such that each one processes only those parts which it suits best. However, this would require a fast direct connection between FPGAs and GPUs which is unfortunately not the case in the setup described here.

ACKNOWLEDGMENTS

This study makes use of data generated by the Wellcome Trust Case-Control Consortium. A full list of the investigators who contributed to the generation of the data is available from http://www.wtccc.org.uk. Funding for the project was provided by the Wellcome Trust under award 076113 and 085475.

The project was supported through the DFG Clusters of Excellence "Inflammation at Interfaces".

REFERENCES

[1] B. Maher, "Personal Genomes: the Case of the Missing Heritability," *Nature*, vol. 456, no. 7218, pp. 18–21, 2008.
[2] J. H. Moore, F. W. Asselbergs, and S. M. Williams, "Bioinformatics Challenges for Genome-Wide Association Studies," *Bioinformatics*, vol. 26, no. 4, pp. 445–455, 2010.
[3] H. J. Cordell, "Detecting Gene-Gene Interactions that Underlie Human Diseases," *Nature Review Genetics*, vol. 10, no. 6, pp. 392–404, 2009.
[4] K. van Steen, "Travelling the World of Gene-Gene Interactions," *Briefings in Bioinformatics*, vol. 13, no. 1, pp. 1–19, 2011.
[5] Y. Wang, G. Liu, M. Feng, and L. Wong, "An Empirical Comparison of Several Recent Epistatic Interaction Detection Methods," *Bioinformatics*, vol. 27, no. 21, pp. 2936–2943, 2011.
[6] J. Marchini, P. Donnelly, and L. R. Cardon, "Genome-wide Strategies for Detecting Multiple loci that Influence Complex Diseases," *Nature Genetics*, vol. 37, pp. 413–417, 2005.
[7] X. Wan, C. Yang, Q. Yang, H. Xue, X. Fan, N. L. Tang, and W. Yu, "BOOST: A Fast Approach to Detecting Gene-Gene Interactions in Genome-wide Case-Control Studies," *American Journal of Human Genetics*, vol. 87, no. 3, pp. 325–340, Sep. 2010.
[8] M. D. Ritchie, L. W. Hahn, N. Roodi, L. R. Bailey, W. D. Dupont, F. F. Parl, and J. H. Moore, "Multifactor-Dimensionality Reduction Reveals High-Order Interactions among Estrogen-Metabolism Genes in Sporadic Breast Cancer," *American Journal of Human Genetics*, vol. 69, no. 1, pp. 138–147, Jul. 2001.
[9] F. V. Lishout, J. M. M. John, E. S. Gusareva1, V. Urrea, I. Cleynen, E. Théâtre, B. Charloteaux, M. L. Calle, L. Wehenkel, and K. V. Steen, "An Efficient Algorithm to Perform Multiple Testing in Epistasis Screening," *BMC Bioinformatics*, vol. 14, no. 138, 2013.
[10] J. Piriyapongsa, C. Ngamphiw, A. Intarapanich, S. Kulawonganunchai, A. Assawamakin, C. Bootchai, P. J. Shaw, and S. Tongsima, "iLOCi: a SNP Interaction Prioritization Technique for Detecting Epistasis in Genome-Wide Association Studies," *BMC Genomics*, vol. 13, no. Suppl 7, pp. S2+, 2012.
[11] L. S. Yung, C. Yang, X. Wan, and W. Yu, "GBOOST: a GPU-based tool for detecting gene-gene interactions in genome-wide case control studies," *Bioinformatics*, vol. 27, no. 9, pp. 1309–1310, 2011.
[12] X. Wan, C. Yang, Q. Yang, H. Zhao, and W. Yu, "The Complete Compositional Epistasis Detection in Genome-Wide Association Studies," *BMC Genetics*, vol. 14, no. 7, 2013.
[13] X. Hu, Q. Liu, Z. Zhang, Z. Li, S. Wang, L. He, and Y. Shi, "SHEsisEpi, a GPU-Enhanced Genome-Wide SNP-SNP Interaction Scanning Algorithm, Efficiently Reveals the Risk Genetic Epistasis in Bipolar Disorder," *Cell Research*, vol. 20, pp. 854–857, 2010.
[14] B. Goudey, D. Rawlinson, Q. Wang, F. Shi, H. Ferra, R. M. Campbell, L. Stern, M. T. Inouye, C. S. Ong, and A. Kowalczyk, "GWIS - Model-Free, Fast and Exhaustive Search for Epistatic Interactions in Case-Control GWAS," *BMC Genomics*, vol. 14, no. Supl 3, 2012.
[15] G. Hemani, A. Theocharidis, W. Wei, and C. Haley, "EpiGPU: Exhaustive Pairwise Epistasis Scans Parallelized on Consumer Level Graphics Cards," *Bioinformatics*, vol. 27, no. 11, pp. 1462–1465, 2011.

[16] S. Prabhu and I. Pe'er, "Ultrafast Genome-Wide Scan for SNPSNP Interactions in Common Complex Disease," *Genome Research*, vol. 22, no. 11, pp. 2230–2240, 2012.

[17] X. Wan, C. Yang, Q. Yang, H. Xue, N. L. Tang, and W. Yu, "Predictive Rule Inference for Epistatic Interaction Detection in Genome-Wide Association Studies," *Bioinformatics*, vol. 26, no. 1, pp. 30–37, 2010.

[18] C. Yang, Z. He, X. Wan, Q. Yang, H. Xue, and W. Yu, "SNPHarvester: a Filtering-Based Approach for Detecting Epistatic Interactions in Genome-Wide Association Studies," *Bioinformatics*, vol. 25, no. 4, pp. 504–511, 2009.

[19] X. Zhang, S. Huang, F. Zou, and W. Wang, "TEAM: Efficient Two-Locus Epistasis Tests in Human Genome-Wide Association Study," *Bioinformatics*, vol. 26, no. 12, pp. 217–227, 2010.

[20] J. Wu, B. Devlin, S. Ringquist, M. Trucco, and K. Roeder, "Screen and Clean: a Tool for Identifying Interactions in Genome-Wide Association Studies," *Genetic Epidemiology*, vol. 34, no. 3, pp. 275–285, 2010.

[21] J. Bedö, D. Rawlinson, B. Goudey, and C. S. Ong, "Stability of bivariate GWAS biomarker detection," *PLOS One*, vol. In Press, 2014.

[22] R. L. Milne, J. Herranz, K. Michailidou, and et al, "A Large-Scale Assessment of Two-Way SNP Interactions in Breast Cancer Susceptibility Using 46,450 Cases and 42,461 Controls from the Breast Cancer Association Consortium," *Human Molecular Genetics*, vol. 23, no. 7, pp. 1934–1946, 2014.

[23] M. Chu, R. Zhang, Y. Zhao, and et al, "A Genome-Wide Gene-Gene Interaction Analysis Identifies an Epistatic Gene Pair for Lung Cancer Susceptibility in Han Chinese," *Cancinogenesis*, vol. 32, no. 3, pp. 572–577, 2014.

[24] J. Bi, J. Gelernter, J. Sun, and H. R. Kranzler, "Comparing the Utility of Homogeneous Subtypes of Cocaine Use and Related Behaviors with DSM-IV Cocaine Dependence as Traits for Genetic Association Analysis," *American Journal of Medical Genetics*, vol. 165, no. 2, pp. 148–156, 2014.

[25] M. Xie, J. Li, and T. Jiang, "Detecting Genome-Wide Epistases Based on the Clustering of Relatively Frequent Items," *Bioinformatics*, vol. 28, no. 1, pp. 5–12, 2012.

[26] Y. Chen, B. Schmidt, and D. L. Maskell, "A Hybrid Short Read Mapping Accelerator," *BMC Bioinformatics*, no. 14, 2013.

[27] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating Compute-Intensive Applications with GPUs and FPGAs," in *SASP20808*. Springer, 2008, pp. 101–107.

[28] K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu, and X. Tian, "High Performance Biological Pairwise Sequence Alignment: FPGA versus GPU versus Cell BE versus GPP," *International Journal of Reconfigurable Computing*, vol. 2012, 2012.

[29] I. Pechan and J. Feher, "Molecular Docking on FPGA and GPU Platforms," in *FPL2011*. Springer, 2011, pp. 474–477.

[30] G. Pfeiffer, S. Baumgart, J. Schröder, and M. Schimmler, "A Massively Parallel Architecture for Bioinformatics," in *ICCS2009, Lecture Notes in Computer Science*, vol. 5544. Springer, 2009, pp. 994–1003.

[31] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: a Unified Graphics and Computing Architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[32] "NVIDIAs Next Generation CUDA Compute Architecture: Fermi," NVIDIA Corporation Whitepaper, USA, Tech. Rep., 2009.

[33] "NVIDIAs Next Generation CUDA Compute Architecture: Kepler," NVIDIA Corporation Whitepaper, USA, Tech. Rep., 2012.

[34] The Wellcome Trust Case Control Consortium, "Genome-wide Association Study of 14,000 Cases of Seven Common Diseases and 3,000 Shared Controls," *Nature*, vol. 447, no. 7145, pp. 661–78, Jun. 2007.

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

Lars Wienbrandt received his M.Sc. (Dipl-Inf.) in Computer Science from the Christian-Albrechts-University of Kiel, Germany, in 2009. He is currently a research assistant at the Technical Computer Science group at the CAU and working on his PhD. His research area includes the parallelization and implementation of bioinformatics algorithms on FPGA architectures.

Jan Christian Kässens received his M.Sc. in Computer Science from the Christian-Albrechts-University of Kiel, Germany, in 2012. He is currently a research assistant at the Technical Computer Science group at the CAU and working on his PhD. His research focus lies on hardware/software co-development, hardware-assisted parallelization and FPGA technology.

David Ellinghaus received his M.Sc. (Dipl.-Bioinf.) in Bioinformatics from the University of Hamburg, Germany, in 2007. In 2012 he earned his PhD from the Christian-Albrechts-University of Kiel, Germany. He is currently a postdoctoral researcher at the Institute of Clinical Molecular Biology in Kiel. His main research interests are in the areas of genetic/epidemiological studies for immune-mediated diseases, bioinformatics and statistical genetics.

Manfred Schimmler received his diploma in Computer Science in 1980 and his PhD in 1991 from the CAU Kiel, Germany. Later he became professor in Stralsund, Braunschweig, and Kiel, where he currently works as head of the group Technical Computer Science. His main research interests are massively parallel architectures and low energy computers, as well as bioinformatics and financial prognostics as application fields.

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