Journal of Computational Science 30 (2019) 183–193

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

1000× faster than PLINK: Combined FPGA and GPU accelerators for logistic regression-based detection of epistasis

Lars Wienbrandt*, Jan Christian Kässens, Matthias Hübenthal, David Ellinghaus

Institute of Clinical Molecular Biology, Kiel University, University Medical Center Schleswig-Holstein, Campus Kiel, Rosalind-Franklin-Str. 12, 24105 Kiel, Germany

A R T I C L E I N F O

Article history:
Received 14 September 2018
Received in revised form
19 December 2018
Accepted 22 December 2018
Available online 24 December 2018

Keywords:
Genome-wide association study (GWAS)
Gene-gene (G × G) interaction
FPGA GPU hardware accelerator
Hybrid computing
Heterogeneous architecture

A B S T R A C T

Logistic regression as implemented in PLINK is a powerful and commonly used framework for assessing gene-gene interactions. However, fitting regression models for each pair of markers in a genome-wide dataset is a computationally intensive task, for which reason pre-filtering techniques and fast epistasis screenings are applied to reduce the computational burden.

We demonstrate that employing a combination of a Xilinx UltraScale FPGA with an Nvidia Tesla GPU leads to runtimes of only minutes for logistic regression tests on a genome-wide scale, resulting in a speedup of more than 1000 up to 1600 when compared to multi-threaded PLINK on a server-grade computing platform.

This article is an extended version of our conference paper [1].

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

Gene–gene (G × G) interactions (epistasis) are believed to be a significant source of unexplained genetic variation causing complex chronic diseases. Several studies provided evidence for statistical G × G interaction between the top disease-associated single nucleotide polymorphisms (SNPs) of complex chronic diseases, including ankylosing spondylitis [2], Behçet's disease [3], type 2 diabetes [4], and psoriasis [5]. Particularly, in psoriasis a significant interaction ($p = 6.95 \times 10^{-6}$) as measured by logistic regression has been detected between the genes *ERAP1* (*rs27524*) and *HLA-C* (*rs10484554*). The biological consequence of this interaction is that the *ERAP1* SNP only has an effect in individuals carrying at least one copy of the risk allele at the *HLA-C* SNP.

In general, detection of G × G interactions poses a great challenge for genome-wide association studies (GWAS) due to the computational burden of testing billions of pairs of SNPs (as a result of the number of tests being quadratic in the number of SNPs). Traditional logistic regression analysis is still the gold-standard to detect statistical G × G interactions in case/control studies, but too slow in practice to screen for G × G interactions on a genome-wide scale. Thus, numerous approximate methods for epistasis screening have been proposed applying a variety of heuristic and filtering techniques to conduct genome-wide interaction studies (GWIS) in a reasonable amount of time. Well-established methods include the Kirkwood Superposition Approximation (KSA) of the Kullback–Leibler divergence implemented in BOOST [6] as well as the joint effects test introduced by Ueki et al. [7]. Another exhaustive interaction method, called GWIS [8], employs a permutation-based approach to calibrate test statistics. Similarly, MBMDR [9] uses permutations to adjust the p-value of the significance test. However, it is able to reduce the dimensionality of any problem into one dimension categorizing into high-risk, low-risk and no evidence groups before calculating a chi-squared test statistic. Other tools defining different test statistics include BiForce [10], iLOCi [11] and EDCF [12]. The latter uses a clustering approach in order to reduce the computational burden. Recently, entropy-based measures for G × G interaction detection gained increasing attention. A well-written overview can be found in [13].

However, no convincing G × G loci have been identified exclusively from GWIS using these approaches. Many of the methods derive an upper bound on the test statistic in order to prune the search space and conduct follow-up model-fitting analysis using logistic regression on a pre-filtered subset of pairs [14]. Furthermore, the computational load for preliminary epistasis screenings is not negligible. Accordingly, several tools emerged to speedup this

* Corresponding author.
E-mail address: l.wienbrandt@ikmb.uni-kiel.de (L. Wienbrandt).
URL: http://www.ikmb.uni-kiel.de (L. Wienbrandt).

https://doi.org/10.1016/j.jocs.2018.12.013
1877-7503/© 2018 Elsevier B.V. All rights reserved.

process employing hardware accelerators, such as graphics processing units (GPUs) in GBOOST [15] or SHEsisEpi [16]. Another way to reduce the computational burden is to reduce the number of SNPs in advance by pre-filtering for linkage disequilibrium (LD), although it can be shown that SNPs supposed to be in LD may also reveal an interaction effect [17,18].

An attempt to reduce the computational load for logistic regression tests is made in [19] by using GLIDE [20]. To our knowledge, GLIDE is the fastest currently available GPU implementation of the logistic regression $G \times G$ interaction test. More recently, CARAT-GxG [21] emerged. It also offers linear regression including covariate analysis on GPUs, but provides a poor performance when compared to GLIDE (12 days for a dataset containing 500,000 SNPs and not more than 1000 samples using CARAT-GxG with 32 Nvidia Tesla M2070 GPUs vs. 6 hours using GLIDE on 12 Nvidia GTX 580 GPUs).

In this article, we show that we are able to perform an exhaustive genome-wide logistic regression analysis for SNP-SNP interactions on datasets consisting of hundreds of thousands of SNPs and tens of thousands of samples in minutes, thus eliminating the needs for epistasis screening or LD-filtering as a preprocessing step. If required, LD-filtering can directly be applied as a postprocessing step, thanks to on-the-fly calculation of r^2. Furthermore, we perform our calculations in double-precision floating point format in order to overcome precision problems that may occur during floating point accumulations.

As the PLINK software is well established and considered the gold-standard in the GWAS community, we run our benchmark against PLINK v1.9 using 32 threads on a computing system with two Intel Xeon E5-2667v4 eight-core CPUs. We achieve performance improvements in two steps. Firstly, we gain a 10–15-fold speedup by software modification alone. In particular, we sacrifice the support for sample covariates (re-enabling it in our method is still under development) and adapt the logistic regression test to be based on contingency tables. This reduces the computational complexity from $\mathcal{O}(NT)$ to $\mathcal{O}(N+T)$ (with N indicating the number of samples and T the number of iterations required for a single test).

Secondly, by employing a combination of only two hardware accelerators, namely a Xilinx Kintex UltraScale KU115 Field Programmable Gate Array (FPGA) and an Nvidia Tesla P100 Graphics Processing Unit (GPU), we gain another 100-fold speedup resulting in a total speedup factor of >1,600 compared to multi-threaded PLINK on a server-grade platform. Exemplary, for analyzing a real-life dataset consisting of 130 k SNPs and 48 k samples our method requires only 7.25 min while PLINK running with 32 threads requires 5.5 days. An imputed dataset with 1.2 M SNPs and 16 k samples requires only 3.75 h in contrast to 246 days runtime required by PLINK.

For a more detailed comparison, we developed and evaluated an alternative GPU-only approach that uses a single Nvidia Tesla P100 GPU accelerator. The GPU is able to accelerate the host-only version by a factor of 5 resulting in a 50–80-fold speedup when compared to PLINK. In particular, above mentioned datasets require 2.5 h and 3.3 days, respectively. Furthermore, it is revealed that combining the strengths of different accelerator techniques, namely FPGAs and GPUs, leads to a disproportionally high speedup when compared to using a single technique alone. In this case, combining a Kintex KU115 FPGA and a Tesla P100 GPU is around 20 times faster than using the GPU alone.

In order to analyse the effect on the runtime of varying input characteristics, we prepared several datasets based on real data with a varying number of samples and SNPs and ran a benchmark on all of them with PLINK and our host-only, GPU-only and hybrid implementations.

2. Pairwise epistasis testing

2.1. Logistic regression test

In this article we address the efficient implementation of a genotype-based statistical test for binary traits. Let Y be a random variable correlated with the trait. Correspondingly, for the trait being a disease, we define the two possible outcomes of Y as $Y=1$ if the sample is a *case* affected by the disease, and $Y=0$ if the samples is a *control* unaffected by the disease. Furthermore, for a pairwise test, we define X_A and X_B as random variables correlated with the observation of genotypes at SNPs A and B, respectively. The possible outcomes of $X_{A/B}$ are $g_{A/B} \in \{0, 1, 2\}$ representing the observed genotype (0 = homozygous reference, 1 = heterozygous, 2 = homozygous variant). PLINK [22,23] uses the following multiplicative logistic regression affection model with β_3 indicating the interaction effect of SNPs A and B:

$$\ln \frac{P(Y=1|X_A=g_A, X_B=g_B)}{P(Y=0|X_A=g_A, X_B=g_B)} = \beta_0 + \beta_1 g_A + \beta_2 g_B + \beta_3 g_A g_B. \quad (1)$$

PLINK employs Newton's method to iteratively obtain maximum likelihood (ML) estimates of the model parameters. It firstly generates a covariate matrix $\mathbf{C}$ with entries C_{ij} whereby i indicates a sample of the input dataset and $j \in 0, 1, 2, 3$ indicates a column for each β_j. The matrix is defined as follows:

$$C_{i0} = 1, \quad C_{i1} = g_{iA}, \quad C_{i2} = g_{iB} \quad \text{and} \quad C_{i3} = g_{iA} g_{iB}. \quad (2)$$

In detail, for a variable number of iterations $t = 0, \ldots, T-1$, fitting the vector $\boldsymbol{\beta}$ is performed in a stepwise manner. $\boldsymbol{\beta}^{(0)}$ is initialized with $\beta_j^{(0)} = 0 \ \ \forall j$ for the first iteration $t=0$.

1. For each sample i, compute intermediate variables

$$p_i^{(t)} = \bar{p}_i^{(t)} - y_i \quad \text{and} \quad v_i^{(t)} = \bar{p}_i^{(t)} \left(1 - \bar{p}_i^{(t)}\right) \quad (3)$$

where

$$\bar{p}_i^{(t)} = \left(1 + e^{-\sum_j \beta_j^{(t)} C_{ij}}\right)^{-1} \quad (4)$$

2. Compute gradient

$$\boldsymbol{\nabla}^{(t)} = \left(\nabla_j^{(t)}\right)_{j=0}^{3} \quad \text{with} \quad \nabla_j^{(t)} = \sum_i C_{ij} p_i^{(t)} \quad (5)$$

3. Compute symmetric Hessian matrix

$$\boldsymbol{H}^{(t)} = \left(h_{jk}^{(t)}\right)_{j,k=0}^{3} \quad \text{with} \quad h_{jk}^{(t)} = \sum_i C_{ij} C_{ik} v_i^{(t)} \quad (6)$$

4. Compute $\boldsymbol{\Delta\beta}^{(t)} = \left(\Delta\beta_j^{(t)}\right)_{j=0}^{3}$ by efficiently solving the linear system

$$\boldsymbol{L}^{(t)} \boldsymbol{L}^{(t)T} \boldsymbol{\Delta\beta}^{(t)} = \boldsymbol{\nabla}^{(t)} \quad (7)$$

using the Cholesky decomposition $\boldsymbol{L}^{(t)} = \left(l_{jk}^{(t)}\right)_{j,k=0}^{3}$ of $\boldsymbol{H}^{(t)}$ with

$$l_{jk}^{(t)} = \begin{cases} 0 & \text{if } k > j \\ \sqrt{h_{jj}^{(t)} - \sum_{s=1}^{j-1} l_{js}^2} & \text{if } k = j \\ \frac{1}{l_{kk}} \left(h_{jk}^{(t)} - \sum_{s=1}^{k-1} l_{js} l_{ks}\right) & \text{if } k < j \end{cases} \quad (8)$$

5. Update model parameters

$$\boldsymbol{\beta}^{(t+1)} \longleftarrow \boldsymbol{\beta}^{(t)} - \boldsymbol{\Delta}\beta^{(t)} \tag{9}$$

If $\sum_j \Delta\beta_j^{(t)}$ approaches zero, i.e. there is no more significant change, the process stops with $\boldsymbol{\beta}^{(t+1)}$ as the current result. Otherwise, the next iteration is started with step 1. However, if the change does not converge to zero, the process stops after a fixed number of iterations. PLINK uses at maximum 16 iterations and a close-to-zero threshold of 0.0001. Additional tests for convergence failure are implemented but omitted here for the sake of brevity.

The result of the logistic regression test in PLINK is composed of three components, namely the test statistic, its approximate p-value and the odds-ratio. The test statistic χ^2 is calculated as

$$\chi^2 = \frac{\beta_3}{\varepsilon^2}. \tag{10}$$

ε is the standard error for the $g_A g_B$-term in (1). It can directly be determined by solving the linear system $\boldsymbol{H}^{(t)}\boldsymbol{e} = (0, 0, 0, 1)$ and defining $\varepsilon^2 = e_3$.

Accordingly, it follows

$$\varepsilon^2 = \frac{1}{\left(l_{33}^{(t)}\right)^2}. \tag{11}$$

The test statistic is assumed to follow a chi-squared distribution χ_1^2 with one degree of freedom. Accordingly, the p-value can directly be approximated from the respective cumulative distribution function (CDF).

Finally, the odds-ratio is defined as

$$\mathrm{OR} = e^{\beta_3}. \tag{12}$$

Obviously, steps 1–3 in each iteration have linear complexity in N, i.e. $\mathcal{O}(N)$ whereby N is the number of samples. Let T be the number of iterations, then $\mathcal{O}(NT)$ is the total complexity for a single test. In Sections 2.2 and 2.3, we show how to generate a contingency table by performing a linear precomputing step and how to apply the contingency table in the logistic regression test, which results in a constant computation complexity for each iteration.

2.2. Contingency tables

For any SNP pair (A, B) a contingency table represents the number of samples in a dataset that carry a specific genotype information. In particular, an entry n_{ij} represents the number of samples that carry the information $g_A = i$ at SNP A and $g_B = j$ at SNP B. Thus, a contingency table for pairwise genotypic tests contains 3×3 entries. Since we are focusing on binary traits, we require a contingency table for each state, w.l.o.g. one for the *case* and one for the *control* group, respectively, and denote their entries by n_{ij}^{case} and n_{ij}^{ctrl} (see Fig. 1).

For a given SNP pair generating the contingency tables is clearly linear in the number of samples. In the next section (Section 2.3) we show how to incorporate contingency tables into logistic regression.

2.3. Logistic regression with contingency tables

The information stored in the contingency tables can be used to simplify steps 1–3 in Section 2.1. Steps 4 and 5 as well as the calculation of the test statistic, the odds-ratio and the p-value remain the same.

1. From a given contingency table we compute the following intermediate variables:

$$p_{ij}^{(t)} = \left(1 + e^{-\left(\beta_0^{(t)} + i\beta_1^{(t)} + j\beta_2^{(t)} + ij\beta_3^{(t)}\right)}\right)^{-1} \tag{13}$$

$$p_{ij}^{(t),\mathrm{ctrl}} = p_{ij}^{(t)}, \quad p_{ij}^{(t),\mathrm{case}} = p_{ij}^{(t)} - 1, \quad v_{ij}^{(t)} = p_{ij}^{(t)}\left(1 - p_{ij}^{(t)}\right)\left(n_{ij}^{\mathrm{case}} + n_{ij}^{\mathrm{ctrl}}\right) \tag{14}$$

2. The gradient $\nabla^{(t)}$ from (5) can now be computed as

$$\nabla^{(t)} = \left(\sum_{ij} N_{ij}^{(t)}, \sum_{ij} iN_{ij}^{(t)}, \sum_{ij} jN_{ij}^{(t)}, \sum_{ij} ijN_{ij}^{(t)}\right) \tag{15}$$

where

$$N_{ij}^{(t)} = \left(n_{ij}^{\mathrm{case}} p_{ij}^{(t),\mathrm{case}} + n_{ij}^{\mathrm{ctrl}} p_{ij}^{(t),\mathrm{ctrl}}\right) \tag{16}$$

3. The symmetric Hessian matrix $\boldsymbol{H}^{(t)}$ from (6) evaluates to

$$\boldsymbol{H}^{(t)} = \left(h_{pq}^{(t)}\right)_{p,q=0}^{3} = \begin{pmatrix} \sum v_{ij}^{(t)} & & \cdots & \\ \sum iv_{ij}^{(t)} & \sum i^2 v_{ij}^{(t)} & & \vdots \\ \sum jv_{ij}^{(t)} & \sum ijv_{ij}^{(t)} & \sum j^2 v_{ij}^{(t)} & \\ \sum ijv_{ij}^{(t)} & \sum i^2 jv_{ij}^{(t)} & \sum ij^2 v_{ij}^{(t)} & \sum i^2 j^2 v_{ij}^{(t)} \end{pmatrix} \tag{17}$$

where each sum is evaluated over all indexes i and j

4. Solve $\boldsymbol{H}^{(t)}\boldsymbol{\Delta\beta}^{(t)} = \nabla^{(t)}$ as in (7) and (8)
5. Update model parameters $\boldsymbol{\beta}^{(t+1)} \longleftarrow \boldsymbol{\beta}^{(t)} - \boldsymbol{\Delta}\beta^{(t)}$ as in (9)

Obviously, the complexity of each iteration step is now constant, i.e. $\mathcal{O}(1)$. As in Section 2.1, let N be the total number of samples and T the number of iterations. We recall the complexity of the method used by PLINK with $\mathcal{O}(NT)$. Our proposed method improves this complexity to $\mathcal{O}(N+T)$ which can directly be observed in a significant increase in computation speed (see Section 4).

2.4. Linkage disequilibrium

Our ultimate aim is the exhaustive testing of all SNP pairs on a genome-wide scale without pre-filtering with regard to linkage disequilibrium (LD). However, to be able to apply posthoc LD-filtering we compute the r^2-score on-the-fly. r^2 is a measure of similarity between two SNPs. It is defined as

$$r^2 = \frac{D^2}{p_A(1-p_A)p_B(1-p_B)} \quad \text{with} \quad D = p_{AB} - p_A p_B. \tag{18}$$

D is the distance between the observed allele frequency p_{AB} at loci A and B and the expected allele frequency $p_A p_B$ assuming statistical independence. Thus, r^2 is a normalized measure for D which can be used for comparison of different SNP pairs. The allele frequencies p_A and p_B can directly be determined as

$$p_A = \frac{2n_{00} + 2n_{10} + 2n_{20} + n_{01} + n_{11} + n_{21}}{2N} \tag{19}$$

and

$$p_B = \frac{2n_{00} + 2n_{01} + 2n_{02} + n_{10} + n_{11} + n_{12}}{2N}, \tag{20}$$

cases ($Y = 1$)		SNP A 0	1	2
SNP B	0	n_{00}^{case}	n_{01}^{case}	n_{02}^{case}
	1	n_{10}^{case}	n_{11}^{case}	n_{12}^{case}
	2	n_{20}^{case}	n_{21}^{case}	n_{22}^{case}

controls ($Y = 0$)		SNP A 0	1	2
SNP B	0	n_{00}^{ctrl}	n_{01}^{ctrl}	n_{02}^{ctrl}
	1	n_{10}^{ctrl}	n_{11}^{ctrl}	n_{12}^{ctrl}
	2	n_{20}^{ctrl}	n_{21}^{ctrl}	n_{22}^{ctrl}

Fig. 1. Contingency tables for cases and controls. n_{ij} reflect the number of occurrences for the corresponding genotype combination in a given pair of SNPs.

respectively, where $n_{ij} = n_{ij}^{\text{case}} + n_{ij}^{\text{ctrl}}$ for all i, j. Unfortunately, the determination of the allele frequency p_{AB} from genotypic data is not trivial. This is due to the unknown phase when two heterozygous genotypes face each other in a SNP pair. Basically, it can be defined as

$$p_{AB} = \frac{2n_{00} + n_{01} + n_{10} + x}{2N} \tag{21}$$

with x meeting $x \le n_{11}$. x has to satisfy the following equation whose solution is omitted here for simplicity:

$$(f_{00} + x)(f_{11} + x)(n_{11} - x) = (f_{01} + n_{11} - x)(f_{10} + n_{11} - x)x \tag{22}$$

where f_{ij} is the number of allele combinations ij we know for sure, e.g. $f_{00} = 2n_{00} + n_{01} + n_{10}$ and $f_{11} = 2n_{22} + n_{21} + n_{12}$.

PLINK does not compute the r^2-score jointly with the logistic regression test. However, one can create a table of r^2 scores explicitly for all pairs of a given range (`-r2` switch), or compute the r^2-score for a single pair (`-ld` switch). The process of determining the respective allele frequencies is of linear complexity in the number of samples for each pair of SNPs. In contrast, we are using the information of the precomputed contingency table which allows us to calculate the r^2-score in constant time. Thus, we do not need to scan through the samples again.

2.5. Result filtering with min–max heap

We use a fixed-sized *min–max fine heap* [24] for result collection. The *min–max heap* can be efficiently implemented as an array-backed implicit data structure with direct access in constant time (i.e. $\mathcal{O}(1)$) to its minimum and maximum values [25]. Insertion and deletion of elements has logarithmic complexity, i.e. $\mathcal{O}(\log t)$ with t being the current number of elements in the heap.

The min-max heap allows us a quick solution for storing only the best k results in sorted order. The decision whether an element has to be inserted into the heap is made in constant time, and in that case, the insertion is inexpensive, i.e. logarithmic in k. Given a large number n of test results with $n \gg k$, the heap insertion overhead becomes negligible as the chance of finding a randomly distributed score larger than the heap's current maximum approaches zero. Thus, the amortized overall runtime of result processing becomes $\Theta(n)$.

In contrast, PLINK does not sort the results at all. It is only able to filter on-the-fly by a given significance threshold, but the results are generated and stored in the order of their calculation. However, to be compliant to PLINK, we allow a joint application of a significance threshold and the min-max heap.

3. Implementation

We implemented three versions of our G × G interaction detection application, a host-only version in order to measure the effect of using contingency tables, and a GPU-only and a hybrid FPGA-GPU version to measure the effects of the different accelerator techniques.

The input dataset for all versions is assumed to be in binary PLINK format (i.e. provided as `.bed`, `.bim`, `.fam` files). We parse and store the data internally in accordance to the `.bed` file format, i.e. in a packed 2-bit representation of a single genotype, organized in SNPs as rows and samples as columns. We apply padding encoded as unknown genotypes to ensure the SNPs to be word aligned in the local RAM. For hybrid runs padding ensures that a minimum number of samples is present as it is required by the FPGA pipeline.

The output is in plain text format containing two header lines with call options and column names, followed by one line for each result with the information on the respective SNP pair (name and ID), χ^2 test statistic, odds-ratio, approximate p-value and r^2-score.

3.1. Heterogeneous FPGA-GPU computing architecture

Our implementation targets a heterogeneous computing architecture with Field Programmable Gate Array (FPGA) and Graphics Processing Unit (GPU) accelerators. We improved our architecture proposed in [26] by employing high-end off-the-shelf components, namely a server-grade mainboard hosting two Intel Xeon E5-2667v4 8-core CPUs @ 3.2 GHz and 256 GB of RAM, an NVIDIA Tesla P100 GPU, and an Alpha Data ADM-PCIE-8K5 FPGA accelerator card.

The GPU accelerator is equipped with 16 GB of graphics memory and is connected via PCI Express Gen3 x16. The FPGA accelerator hosts a recent Xilinx Kintex UltraScale KU115 FPGA with two attached 8 GB SODIMM memory modules. It is connected via PCI Express Gen3 x8 allowing high-speed communication with the host and the GPU. The system is being run by a Linux operating system (currently Ubuntu 18.04 with Kernel version 4.15). Due to driver restrictions, it is currently not possible to perform direct peer transfers, i.e. moving data directly from an FPGA accelerator to a GPU or vice-versa. Therefore, data transfer is redirected via the host memory and both devices are placed in slots that are served by the same CPU to reduce transmission overhead as described in [25].

According to the PCIe specifications, the net transmission rate between FPGA and GPU is about 7.3 GB/s. This absolutely fits regular application demands, such that the transmission interface does not become a bottleneck (see Section 3.2 for more details on transmission speed).

3.2. Task distribution for the hybrid and GPU-only solutions

Similar to our method for testing third-order SNP interactions based on information gain [26], we split the hybrid version into three main subtasks. Firstly, the creation of pairwise contingency tables (see Section 2.2) is done by the FPGA module. Secondly, all computations required for the logistic regression test based on the contingency tables are performed by the GPU. And thirdly, the host collects and filters the results created by the GPU.

For our GPU-only solution, the FPGA accelerator is skipped and the contingency tables are created directly on the GPU before the logistic regression test is performed. The host-only version uses no accelerator modules at all but performs each of the three subtasks on the CPU.

As before, data transmission between the modules is performed by Direct Memory Access (DMA) transfers via PCI Express (except for the host-only version where no data transmissions are required between hardware components), and since there is no direct connection between the FPGA and the GPU module, the transmission of the contingency tables is redirected via the host memory in our hybrid version.

3.2.1. Contingency table creation on the FPGA

The FPGA pipeline for contingency table generation is based on our previous work for pairwise [27,28] and third-order interactions [26].

Shortly summarized, the pipeline consists of a chain of 480 process elements (PEs) divided into two subchains with 240 PEs each. After a short initialization phase, the chain produces 480 contingency tables in parallel while the genotype data of one SNP is streamed through the pipeline at a speed of 266 MHz and 8 genotypes/cycle. This sums up to a peak performance of about 20.4 million contingency table pairs per second for a dataset containing about 50,000 samples or 63.8 million contingency table pairs per second for a dataset containing about 16,000 samples, as both used in our performance evaluation in Section 4.

In detail, each PE in a chain is organized into a memory buffer to store the genotypes of all samples at a single SNP position, implemented in local block RAM (BRAM), and a counter file for the contingency table entries (see Fig. 2). The PE is able to receive a stream of genotype data from a previous PE and to send it to the next PE. The buffers are filled with the first incoming SNP data, i.e. the data for the first SNP is only stored in the local memory and not forwarded to the next PE. Subsequent SNPs are then forwarded and simultaneously used to form genotype pairs with the data stored in the local memory. Dependent on the current pair, the corresponding counter of the contingency table has to be incremented. By streaming eight genotypes at once for each SNP of the pair, i.e. incrementing the corresponding counters of eight genotype combinations at the same time, we could achieve a high throughput of SNP data in each PE.

Streaming is organized by sending the genotypes of the case samples at the current SNP position first, followed by the genotypes of the control samples. Thus, the contingency tables for cases and controls are alternately generated using the same logic resources in the PE. Currently, our design supports a maximum of 65,536 samples, implying 16 bit counters for the entries of the contingency table and block RAM (BRAM) resources for the memory buffer of size 128 kbit.

The contingency tables are retrieved from the PEs again via the PE chain. After all genotypes from either the cases or the controls were streamed, the table is provided to the next PE in one clock cycle. Each PE hands over incoming tables to the next PE first before it sends its own table. All tables are collected at the end of the chain in a separate buffer before transmission to the GPU (via the host). Thus, the collection of tables requires as many clock cycles as there are PEs in the chain implying that the minimum number of either cases or controls for the most efficient utilization of the chain is exactly eight times the number of PEs in the chain. In order to prevent delays resulting from collecting the contingency tables, we divided the complete chain into subchains with continuous genotype streaming but separate table collection units.

The limiting factor for the number of processing elements was the amount of available block RAM on the Xilinx Kintex KU115 FPGA. By utilizing the complete FPGAs resources, we managed to implement 480 PEs in total divided into two subchains with 240 PEs each, as already mentioned above. Therefore, the minimum number of cases or controls for maximum efficiency is $8 \times 240 = 1920$. If a dataset contains a smaller number of samples for either group, the software automatically applies a padding with unknown genotypes.

In previous publications, we used a sparse contingency table representation lacking support for unknown genotypes. The disadvantage of such a design is, that datasets containing unknown genotypes could not be supported because the assumption that the sum of all entries stays the same over all tables is disproved in the presence of unknowns. In order to remove this limitation, we now transfer complete tables from the FPGA to the GPU. Unfortunately, this increases the transmission rate significantly. Therefore, we encode each table entry into a 16-bit integer, i.e. $9 \times 2\text{B} = 18\text{B}$ per table. For each pair of corresponding case and control tables two 32-bit integers = 8 B for the pair ID are added, which accumulates to 44 B per table pair. Hence, the peak transmission rate for the examples above are about 900 MB/s for 50 k samples and about 2.8 GB/s for 16 k samples, compared to 490 MB/s and 1.5 GB/s, respectively, that would be required for the sparse representation.

These transmission rates are well below the capability of the architecture with about 7.3 GB/s theoretically, which allows us to process datasets down to 6150 samples without the transmission link becoming the bottleneck. In fact, we already observed a performance drop in the analysis of datasets with around 10,000 samples and less, indicating the actual transmission speed must be around 5 GB/s for our application (see Section 4).

3.2.2. Contingency table creation on the GPU

The GPU-only implementation requires the contingency tables to be created directly on the accelerator. The host loads the genotype database in the GPU local memory beforehand. Afterwards, the host provides a continuous flow of buffers to the GPU that contain index pairs indicating which SNP pairings are to be analysed next. For each buffer a number of CUDA threads is launched, each thread processing one SNP pair.

Each thread firstly creates the contingency table by reading the required SNP data from local GPU memory and then, continues the calculation of the test statistic in the same way as with an attached FPGA accelerator (see Section 3.2.3). There is no need to store the contingency table anywhere other than in the local thread memory. Thus, we only create warp divergence while reading the individual SNP data for each thread, but due to the packed SNP-wise data format with a 2-bit encoding per genotype, each thread only requires a few kilobytes which can be processed almost immediately. Furthermore, in most cases, the SNP pairs in the current warp share the first SNP and the second SNPs in the pairs are in consecutive order. Together with the word-aligned SNP data, this is an optimal scenario for fast access to the local GPU memory.

3.2.3. Processing contingency tables on the GPU

In the hybrid solution, the buffers from the FPGA containing contingency tables are transferred to local GPU memory. We use a transmission buffer size of 256 MB which may hold up to 6.7 million table pairs. The computation process follows a simple parallelization scheme over GPU threads. By setting the block size to the maximum supported block size and the grid size to evenly distribute the contingency tables over the blocks, each thread processes exactly one contingency table pair, and only one kernel call per buffer is required. Besides the distribution of the contingency table data and the writeback of the results no calls to local GPU memory are required from a GPU thread. For the GPU-only implementation, the contingency tables are created and kept in local thread memory as described above in Section 3.2.2.

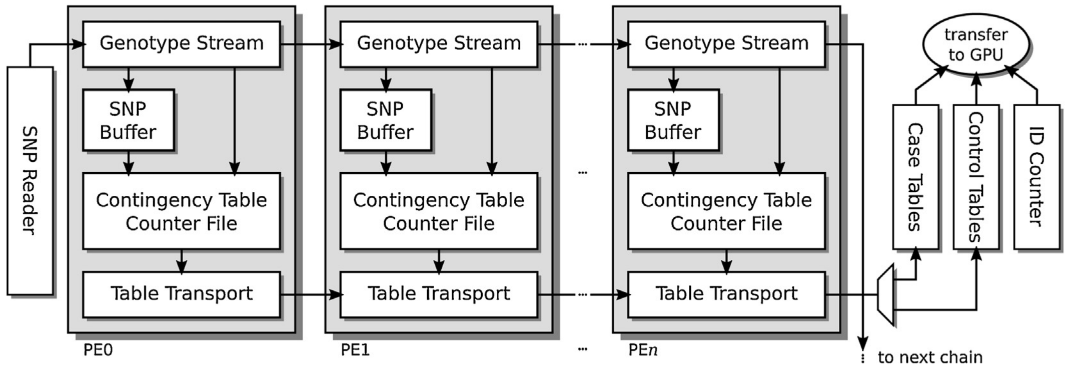

Fig. 2. Chain of processing elements (PEs) on the FPGA. Our design for the Kintex KU115 FPGA contains 480 PEs divided into two subchains with 240 PEs each.

Logistic regression and LD computation have been implemented as described in Sections 2.3 and 2.4. However, in contrast to PLINK, we use the double precision floating point format in all our computations. The output is written into a result buffer. We provide one result buffer for each table transmission buffer, which is transferred to the host as soon as processing a table buffer has finished.

By evenly distributing the contingency tables over the blocks, we most likely introduce an unequal load resulting from a varying number of Newton iterations per thread. However, the average number of iterations per block remains virtually constant.

3.2.4. *Transmission buffer management and result collection on the host*

We use a similar transmission buffer management as presented in [26], but introduced some improvements. In order to reduce transmission overhead, we apply different adapted buffer sizes for contingency table transmission between FPGA and GPU, and result transmission from GPU to host. Exemplary, the default transmission buffer size of 256 MB for contingency tables leads to 230.4 MB for results (reserving space for one result per contingency table pair). As before, the buffers are page-locked to ensure a fast transmission without delay, and the number of buffers allocated for each connection is equal (eight per default).

Multiple threads on the host system perform the collection of results by filtering by a given significance threshold and finally providing them sorted with regard to the test statistic. For this purpose, the min–max fine heap data structure [24,25] is employed. Each thread keeps its own instance of a min–max heap to avoid lock conditions and inserts a result only if the test statistic exceeds the threshold. Then, the output file is composed by iteratively extracting the single best result over all heaps until the heaps are drained or the number of requested results is reached, whichever occurs first.

The complete workflow on our heterogeneous FPGA-GPU-based architecture is illustrated in Fig. 3, the GPU-only approach in Fig. 4.

4. Performance evaluation

4.1. *Datasets*

For performance evaluation on real data we prepared eight datasets based on in-house cohorts. Dataset “A” is a very small dataset regarding the number of SNPs, but with a large number of samples. It contains 19,085 cases of an autoimmune disease and 34,213 healthy controls typed at only 282 SNPs in the HLA region on chromosome 6. Dataset “B” is typed genome-wide at 185,239 SNPs, but contains only 373 cases and 590 controls. Datasets “C1”, “C2” and “C3” share the same 1913 cases of another autoimmune disease and 14,295 healthy controls. “C2” corresponds to the original genome-wide dataset typed at 144,238 SNPs. “C1” is based on the same set with the SNPs being filtered according to a maximum allele frequency (MAF) threshold of 5%, and “C3” is imputed to 1,245,184 markers. Datasets “D1”, “D2” and “D3” share 14,513 cases of an autoimmune disease and the same 34,213 healthy controls as in dataset “A”. Dataset “D3” is the original genome-wide set with 130,052 markers while “D2” is filtered for linkage disequilibrium based on an r^2-score threshold of 0.2. “D1”, in turn, is the same as “D2” but reduced to markers at chromosomes 5 and 6. An overview of these datasets can be found in Table 1.

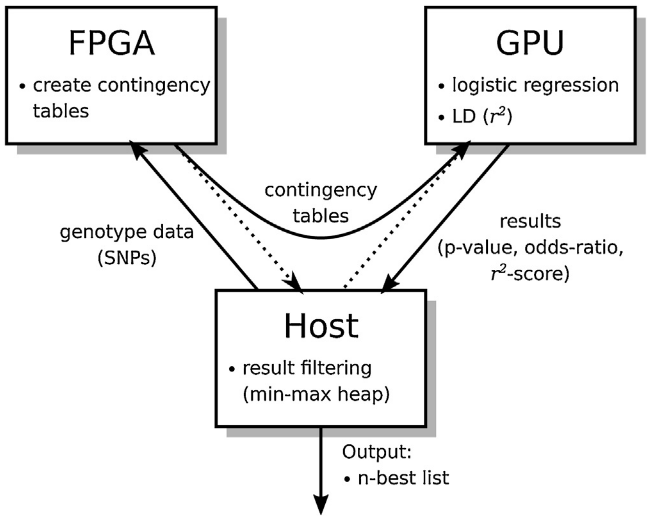
FPGA
• create contingency tables
GPU
• logistic regression
• LD (r^2)
contingency tables
genotype data (SNPs)
results (p-value, odds-ratio, r^2-score)
Host
• result filtering (min-max heap)
Output:
• n-best list

Fig. 3. Workflow on our heterogeneous system. (1) Genotypic data is sent to the FPGA. (2) For each pair of SNPs the FPGA creates a contingency table. (3) The contingency tables are sent to the GPU employing a memory buffer on the host. (4) The GPU calculates the logistic regression test and linkage disequilibrium. (5) Results (*p*-value, odds-ratio and LD-score) are transferred back to host. (6) Results are filtered using a min–max heap on the host.

Additionally, we analysed the effect of varying numbers of samples and varying numbers of SNPs on the runtime. For this purpose, we draw 3500, 7000, 10,500, 14,000, 17,500, 21,000, 28,000 and 35,000 random samples (approximately preserving the original case-control ratio of 2:5) as well as 5000, 15,000, 30,000, 60,000 and 100,000 random SNPs from “D3”. Then, for each combination (also including the original number of 48,736 samples and 130,052 SNPs), we generated a new dataset resulting in 54 datasets with a varying number of samples and SNPs. For the samples, we kept the approximate case-control ratio of the original set with 2:5.

Table 1
Overview of datasets.

Dataset	# SNPs	# cases	# controls	# samples	Description
A	282	19,085	34,213	53,298	Disease A, only HLA region
B	185,239	373	590	963	Disease B, complete
C1	20,096	1,913	14,295	16,208	Disease C, 5% MAF-filtered
C2	144,238	1,913	14,295	16,208	Disease C, complete
C3	1,245,184	1,913	14,295	16,208	Disease C, imputed
D1	5,725	14,513	34,213	48,726	Disease D, LD-filtered, chr. 5+6
D2	37,358	14,513	34,213	48,726	Disease D, LD-filtered
D3	130,052	14,513	34,213	48,726	Disease D, complete

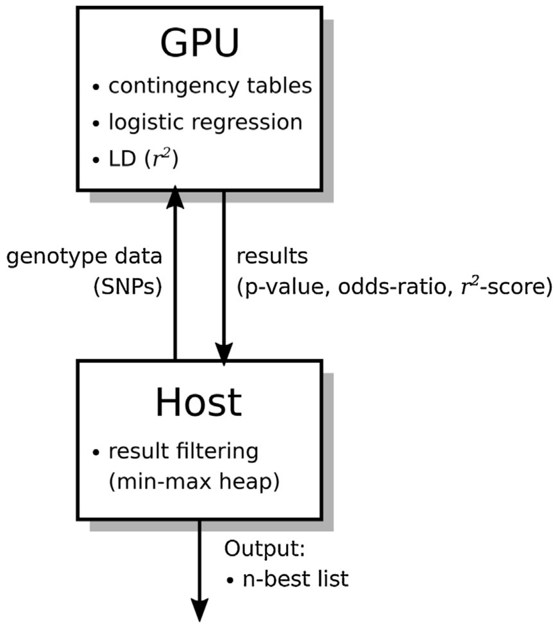

Fig. 4. Workflow for our alternative GPU-only solution. Contingency table creation moved from the FPGA to the GPU. *(1) Genotypic data is sent to the GPU. (2) For each pair of SNPs the GPU creates a contingency table. 3. Each contingency table is directly used to calculate the logistic regression test and linkage disequilibrium. 4. Results (p-value, odds-ratio and LD-score) are transferred back to host. 5. Results are filtered using a min-max heap on the host.

4.2. *Software-only evaluation*

Our target system for all software and hardware accelerated runs was the architecture described in Section 3.1, equipped with two Intel Xeon E5-2667v4 eight-core CPUs @ 3.2 GHz and 256 GB of RAM. We compiled the host-only version with GCC 7.3.0. For comparison to PLINK, we used the to date most recent 64-bit PLINK v1.9 built published on January 9, 2018 [29]. We compiled the PLINK code with the provided Makefile.

We ran PLINK on all eight real datasets. We computed the standard logistic regression tests with flags `–epistasis –epi1 5e-8` which filters the results by a genome-wide significance threshold of 5×10^{-8} according to the approximate p-value. The runs used all available 32 threads (`–threads 32`) on our described system.

For the benchmark datasets, we decided against running PLINK on all these datasets since we already encountered long runtimes especially for the datasets "D2" and "D3". Instead, we generated another set of 54 datasets exactly as described above in Section 4.1 but with only 10% of the SNPs, and ran PLINK on these datasets with the same parameters as before.

We ran our host-only implementation on the same system as well with 32 threads. We emphasize that in contrast to PLINK, our implementation performs all calculations in double-precision floating point format, while PLINK only uses single-precision. Furthermore, we calculated the r^2-score in order to test for linkage disequilibrium on all SNP pairs, which PLINK does not.

We verified the correctness of our implementation by comparing our results to the PLINK results. At first, we encountered differences in the score and also in the order. Thus, we modified the source code of PLINK to perform the calculations in double-precision as well. This modification increased the runtime of PLINK by a factor of about 5.7, but the results were virtually equal now, indicating that the inconsistencies were caused by the different precisions. We believe the remaining deviations resulted from numerical instabilities in PLINK when accumulating small floating-point values over all samples in steps 2 and 3 of computing the logistic regression test (see (5) and (6) in Section 2.1). Table 2 exemplarily shows the differences we encountered between single and double precision calculations in dataset "D1".

All wall-clock runtimes were measured with the GNU time command. Measurements were averaged over multiple runs with very low variance in runtimes (<1%). The results for the logistic regression tests on the eight real datasets are listed together with the results using the hardware accelerators in Table 3. The measures demonstrate that by applying our method that employs contingency tables, we gain a 10–15-fold speedup due to the reduced runtime complexity.

For the runtimes of all 54 benchmark datasets (together with the benchmarks from the hardware accelerated runs) Fig. 5 illustrates the expected quadratic relationship between runtime and number of SNPs as well as the linear relationship between runtime and number of samples.

However, Table 3 also reveals, that the speedup with our solution on smaller datasets with shorter runtimes is lower than for larger datasets. For a more detailed analysis, we created a graphical representation of the speedups against PLINK on our benchmark datasets in Fig. 6. Since PLINK has been applied to smaller datasets only, runtimes required to estimate the speedups were extrapolated based on a linear model.

What was to be expected and clearly can be seen is that the speedups are almost unaffected by the number of SNPs, but the positive effect from changing the runtime complexity of the logistic regression test (see Section 2.3) is visible not before a certain number of samples.

4.3. *Evaluation of hardware acceleration*

Our hardware-accelerated implementations were compiled with GCC 7.3.0 again and CUDA 9.0.176. The FPGA code was written in VHDL and compiled with Xilinx Vivado 2017.3. We tested our implementation with a hybrid build, i.e. using both accelerators (Xilinx Kintex UltraScale KU115 FPGA and Nvidia Tesla P100 GPU), and a GPU-only build, solely using the Nvidia Tesla P100 GPU accelerator. As for the host-only runs, we performed all calculations in double-precision floating point format and calculated the r^2-score in order to test for linkage disequilibrium on all SNP pairs. Furthermore, we applied a faster epistasis screening test on the eight real datasets with the BOOST [6] method (included in PLINK) with

Table 2
Differences in precision exemplary on the first 10 results of dataset “D1”. “sp.” is the value calculated by PLINK in single precision and “dp.” is the value calculated by our implementation in double precision. (Depicted variants are for demonstration purposes only and do not guarantee nor imply proven statistical interaction.)

SNP pair		χ^2-score		p-Value	
		sp.	dp.	sp.	dp.
chr6:2205110	chr6:32682207	260.627	260.642	1.324e−58	1.313e−58
chr6:32682207	chr6:32732937	238.125	238.176	1.061e−53	1.034e−53
chr6:32171683	chr6:32427748	205.669	205.735	1.265e−46	1.223e−46
chr6:32373312	chr6:32658079	193.223	193.222	6.562e−44	6.565e−44
chr6:32377284	chr6:32658079	183.571	183.565	8.376e−42	8.4e−42
chr6:32235384	chr6:32377284	175.345	175.345	5.227e−40	5.226e−40
chr6:31431874	chr6:31575276	174.612	174.616	7.557e−40	7.542e−40
chr6:32289594	chr6:32377284	166.193	166.192	5.203e−38	5.204e−38
chr6:32377284	chr6:32430729	165.754	165.750	6.489e−38	6.5e−38
chr6:31575276	chr6:32373312	162.932	162.932	2.681e−37	2.681e−37

Table 3
Wall-clock runtimes and speedups of the hybrid FPGA-GPU logistic regression test compared to PLINK [29] logistic regression (`-epistasis`), our GPU-only implementation and our host-only implementation. PLINK and our host-only version use all available 32 threads on two Intel Xeon E5-2667v4 processors. The host-only, GPU-only and hybrid implementations additionally calculate the r^2-score (LD) and do all computations in double-precision format (vs. single-precision without r^2 in PLINK).

Data	Runtime				Speedup vs. PLINK		
	PLINK	Host-only	GPU-only	Hybrid	Host-only	GPU-only	Hybrid
A	2.3 s	1.5 s	1 s	5 s	1.53	2.30	0.46
B	3 h 55 m	2 h 56 m	11 m 11 s	4 m 39 s	1.34	21.04	50.60
C1	1 h 39 m	6 m 26 s	1 m 29 s	14 s	15.40	66.78	424.50
C2	3 d 13 h	5 h 32 m	1 h 04 m	3 m 19 s	15.39	79.81	**1,538.48**
C3	264 d 02 h	17 d 10 h	3 d 08 h	3 h 47 m	15.16	79.06	**1,674.62**
D1	15 m 48 s	1 m 32 s	24 s	7 s	10.30	39.50	135.43
D2	11 h 10 m	1 h 04 m	13 m 58 s	46 s	10.40	47.95	873.43
D3	5 d 14 h	13 h 13 m	2 h 38 m	7 m 18 s	10.15	51.09	**1,102.19**

Bold values indicate the speedups exceeding 1,000×.

Table 4
Wall-clock runtimes and speedups of the hybrid FPGA-GPU logistic regression test compared to the heuristic screening test in PLINK BOOST [29,6] (`-fast-epistasis boost`) using all available 32 threads on two Intel Xeon E5-2667v4 processors. Our hybrid implementation additionally calculates the r^2-score (LD) and does all computations in double-precision format (vs. single-precision without r^2 in PLINK BOOST).

Data	Runtime		Speedup vs.
	PLINK BOOST	Hybrid	PLINK BOOST
A	0.2 s	5 s	0.04
B	8 m 16 s	4 m 39 s	1.78
C1	22 s	14 s	1.57
C2	17 m 30 s	3 m 19 s	5.28
C3	21 h 44 m	3 h 47 m	5.74
D1	7 s	7 s	1.00
D2	4 m 04 s	46 s	5.30
D3	49 m 17 s	7 m 18 s	6.75

the same threshold flag, but replacing `-epistasis` with `-fast-epistasis boost`. Table 3 shows all runtime measures for the logistic regression tests on the eight real datasets, and the PLINK BOOST results in comparison to our hybrid solution are listed in Table 4.

We observe that on top of the 10–15-fold speedup from the host-only solution, the GPU-only implementation adds a 4–5-fold speed increase, resulting in a total speedup of 40 to 80 against PLINK. With a single additional FPGA hardware accelerator we gain another 20-fold speedup over the GPU-only implementation, resulting in a total computation speed that is more than 1100–1600 times faster than that of PLINK when executed on our high-performance evaluation system. The performance is underlined by the additional burden on our implementation, which is a higher calculation precision and the additional on-the-fly r^2-score computation not performed by the PLINK software. We exemplarily measured the overhead which is caused by the LD-computation on some of our datasets. The host-only implementation generates a runtime overhead of less than 1.5% with LD-computation compared to the same runs without LD-computation. The overhead for the GPU-only implementation was less than 0.4%, and for the hybrid solution no overhead was measurable. Furthermore, our full logistic regression test is still almost 7 times faster than the quick but imprecise pre-scanning method BOOST [6].

As in the software-only evaluation, the graphical representation of the hardware accelerated runtimes of all 54 benchmark datasets in Fig. 5 shows the expected quadratic correlation of the runtime to the number of SNPs and the linear correlation of the runtime to the number of samples as well. And again, the speedup on smaller datasets is lower than for larger datasets. The representation of our speedups against PLINK in Fig. 6 reveals that this effect is stronger for the GPU-only runs and even more conspicious for the hybrid runs. We explain this with a larger constant overhead for buffer allocation, GPU kernel launches and additional communication. Thus, short runs, i.e. especially runs with a small number of SNPs, result in a lower speedup. The same effect is stronger in our hybrid solution because runtimes are still low even for larger datasets and a larger constant overhead is produced due to FPGA-GPU communication buffer allocation and device initialization.

Another clearly visible effect in the runtime plot for the hybrid implementation is that the runtimes are almost equal for samples below around 14,000 and a fixed number of SNPs. This has at least two reasons. Firstly, a dataset containing less than the minimum number of cases or controls, which is 1920 for either group here (see Section 3.2.1), is padded and then treated as a larger dataset. Secondly, a smaller number of samples indicates a faster creation of contingency tables on the FPGA and thus, a larger data transmission speed required for transferring the tables from the FPGA to the GPU. Since the transfer rate is limited, the communication link becomes the bottleneck and the whole process is stalled resulting in a measurable drop of performance.

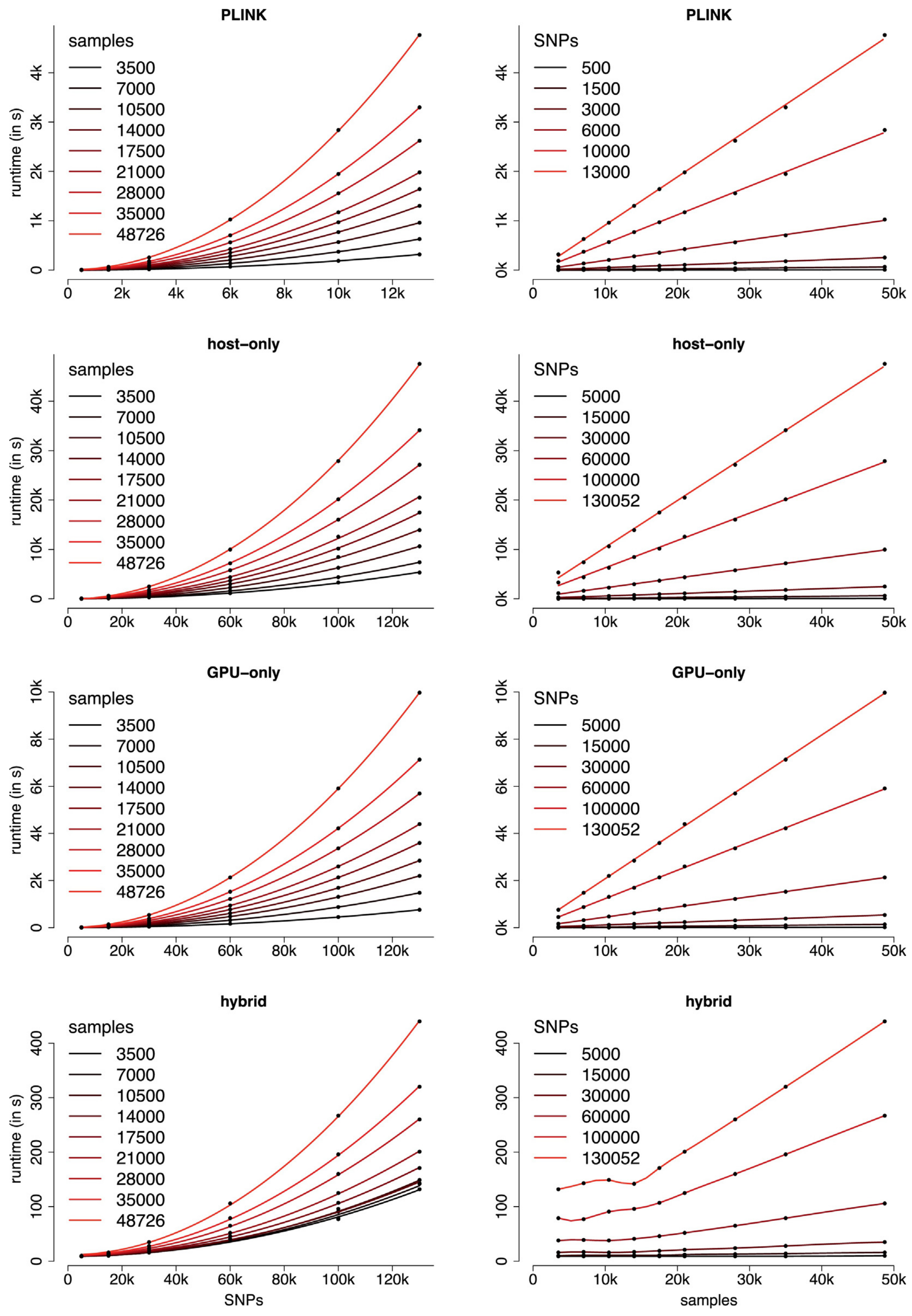

Fig. 5. Wall-clock runtimes (s) of PLINK and our host-only, GPU-only and hybrid implementations with varying numbers of SNPs and samples. Runtime as a function of SNPs and samples has been modeled by quadratic and linear functions respectively. The runtime of the hybrid system as a function of samples has been modeled by cubic splines. The resulting interpolated runtimes have been added to the plot as coloured lines. Note that PLINK was benchmarked with datasets containing only 10% of the SNPs from the benchmark datasets for our implementations.

Please note that this article does not have a medical or biological focus. We explicitly do not give details to the analysed diseases neither interpret our results against a medical or biological background.

5. Conclusions and future work

In this article, we presented two ways of improving performance of PLINK's logistic regression epistasis test [22,23]. Firstly, we reduced the computational complexity from $\mathcal{O}(NT)$ to $\mathcal{O}(N+T)$

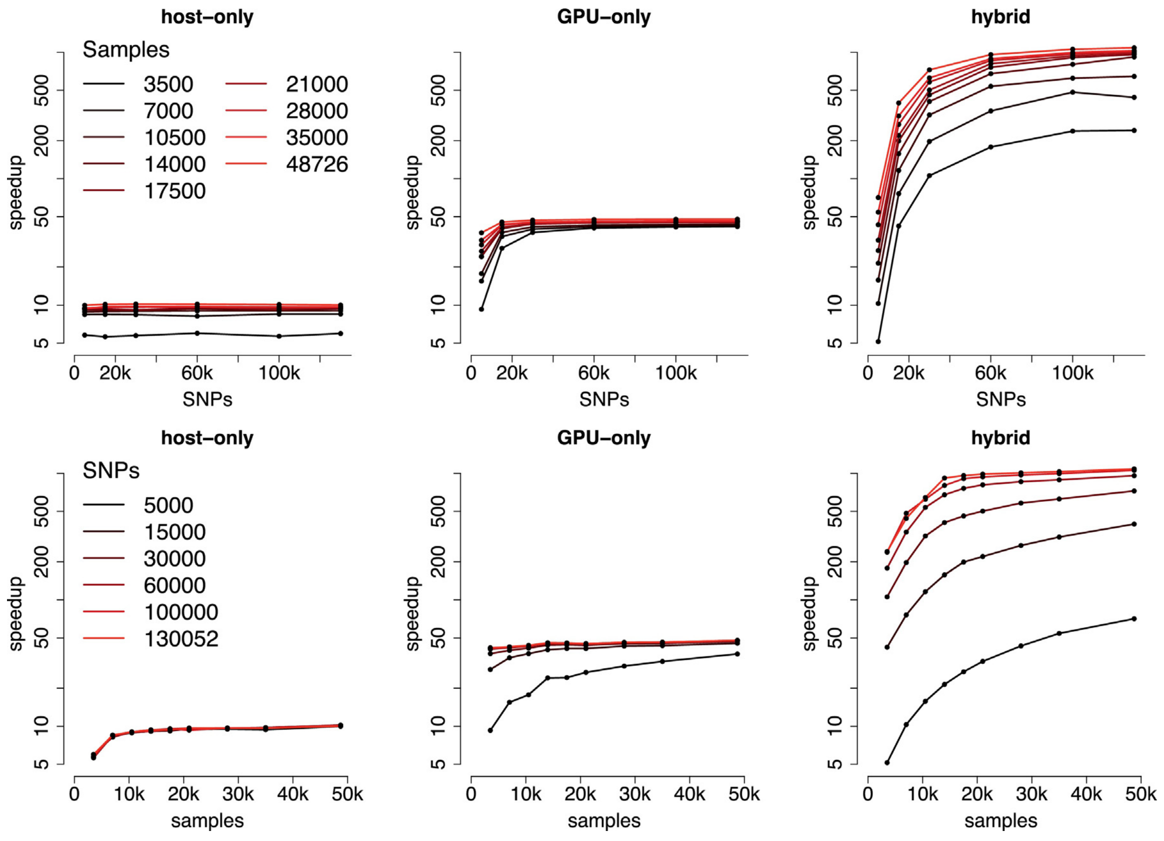

Fig. 6. Speedups of our host-only, GPU-only and hybrid implementations compared to PLINK with varying numbers of SNPs and samples. For a better comparability the speedup-axis is in logarithmic scale.

for a single test by introducing contingency tables (see Sect. 2). This already led to a speedup of more than 10 to 15 for our example datasets, although even calculating in double-precision.

The second improvement was made by applying a two-step hardware acceleration pipeline (see Section 3). By generating contingency tables on a Kintex UltraScale KU115 FPGA and computing the logistic regression based on the tables on an Nvidia Tesla P100 GPU, we gained a total speedup of more than 1100 to 1600 when compared to the original PLINK v1.9 software executed with 32 threads on a server-grade two processor (Intel Xeon E5-2667v4) system. Our GPU-only approach exposed that a single GPU is able to speed up the process by a factor of 40–80. This demonstrates that combining the advantages of two different kinds of accelerator architectures, namely FPGA and GPU, leads to a disproportionally high speedup than using the GPU alone. In numbers, adding the FPGA resulted in another speedup factor of 20 when compared to the GPU-only implementation. Furthermore, we demonstrated that by employing contingency tables, the r^2-score for tests for linkage disequilibrium (LD) can be computed on-the-fly. In combination, this provides a powerful tool for epistasis analysis on large datasets, making LD-filtering deprecated as a pre-processing step.

Consequently, we are able to calculate a full logistic regression test in double-precision format on all pairs of hundreds of thousands of SNPs with tens of thousands of samples in a few minutes and allow to filter the results by score and/or by LD in the post-processing stage.

Currently, our method does not support the use of a covariate matrix as additional user input. However, we are currently working on a solution based on weighted contingency tables in order to be able to incorporate covariate information.

In order to make the system available for the scientific community, we are currently working on a much more powerful successor by enhancing it with three additional Xilinx UltraScale FPGAs, upgrading the Nvidia Tesla P100 to four Tesla V100 boards and optimizing the remaining hardware to the specific workloads. Additionally, we are planning a sophisticated web interface for public use to allow scientists to perform genome-wide epistasis analyses on our new system. We aim to provide a user experience similar to other widely known services, such as the Sanger Imputation Server [30], and also offer a scripting interface for easier integration in existing workflows and infrastructure.

References

[1] L. Wienbrandt, J.C. K&ldquo;assens, M. H&rdquo;ubenthal, D. Ellinghaus, 1,000x faster than plink: genome-wide epistasis detection with logistic regression using combined FPGA and GPU accelerators, in: Y. Shi, H. Fu, Y. Tian, V.V. Krzhizhanovskaya, M.H. Lees, J. Dongarra, P.M.A. Sloot (Eds.), Computational Science – ICCS 2018, Springer International Publishing, Cham, 2018, pp. 368–381, http://dx.doi.org/10.1007/978-3-319-93701-4_28.

[2] The Australo-Anglo-American Spondyloarthritis Consortium (TASC), et al., Interaction between ERAP1 and HLA-B27 in ankylosing spondylitis implicates peptide handling in the mechanism for HLA-B27 in disease susceptibility, Nat. Genet. 43 (2011) 761–767, http://dx.doi.org/10.1038/ng.873.

[3] Y. Kirino, G. Bertsias, Y. Ishigatsubo, et al., Genome-wide association analysis identifies new susceptibility loci for Behcet's disease and epistasis between HLA-B*51 and ERAP1, Nat. Genet. 45 (2013) 202–207, http://dx.doi.org/10.1038/ng.2520.

[4] J.M. Keaton, J.N. Hellwege, M.C.Y. Ng, et al., Genome-wide interaction with selected type 2 diabetes loci reveals novel loci for type 2 diabetes in African Americans, Pac. Symp. Biocomput. 22 (2016) 242–253, http://dx.doi.org/10.1142/9789813207813_0024.

[5] Genetic Analysis of Psoriasis Consortium, et al., A genome-wide association study identifies new psoriasis susceptibility loci and an interaction between HLA-C and ERAP1, Nat. Genet. 42 (2010) 985–990, http://dx.doi.org/10.1038/ng.694.

[6] X. Wan, C. Yang, Q. Yang, et al., BOOST: a fast approach to detecting gene–gene interactions in genome-wide case-control studies, Am. J. Hum. Genet. 87 (3) (2010) 325–340, http://dx.doi.org/10.1016/j.ajhg.2010.07.021.

[7] M. Ueki, H.J. Cordell, Improved statistics for genome-wide interaction analysis, PLoS Genet. 8 (4) (2012) e1002625, http://dx.doi.org/10.1371/journal. pgen.1002625.
[8] B. Goudey, D. Rawlinson, Q. Wang, et al., GWIS: model-free, fast and exhaustive search for epistatic interactions in case-control GWAS, Lorne Genome 2013 (2013), http://dx.doi.org/10.1186/1471-2164-14-S3-S10.
[9] T. Cattaert, M.L. Calle, S.M. Dudek, et al., Model-Based multifactor dimensionality reduction for detecting epistasis in case-control data in the presence of noise, Ann. Hum. Genet. 75 (1) (2011) 78–89, http://dx.doi.org/10.1111/j.1469-1809.2010.00604.x.
[10] A. Gyenesei, J. Moody, C.A. Semple, et al., High-throughput analysis of epistasis in genome-wide association studies with BiForce, Bioinformatics 28 (15) (2012) 1957–1964, http://dx.doi.org/10.1093/bioinformatics/bts304.
[11] J. Piriyapongsa, C. Ngamphiw, A. Intarapanich, et al., iLOCi: a SNP interaction prioritization technique for detecting epistasis in genome-wide association studies, BMC Genom. 13 (Suppl. 7) (2012), http://dx.doi.org/10.1186/1471-2164-13-s7-s2, S2+.
[12] M. Xie, J. Li, T. Jiang, Detecting genome-wide epistases based on the clustering of relatively frequent items, Bioinformatics 28 (1) (2012) 5–12, http://dx.doi.org/10.1093/bioinformatics/btr603.
[13] P.G. Ferrario, I.R. K&ldquo;onig, Transferring entropy to the realm of GxG interactions, Brief. Bioinform. (2016) 1–12, http://dx.doi.org/10.1093/bib/bbw086.
[14] Y. Wang, G. Liu, M. Feng, L. Wong, An empirical comparison of several recent epistatic interaction detection methods, Bioinformatics 27 (21) (2011) 2936–2943, http://dx.doi.org/10.1093/bioinformatics/btr512.
[15] L.S. Yung, C. Yang, X. Wan, et al., GBOOST: a GPU-based tool for detecting gene–gene interactions in genome-wide case control studies, Bioinformatics 27 (9) (2011) 1309–1310, http://dx.doi.org/10.1093/bioinformatics/btr114.
[16] X. Hu, Q. Liu, Z. Zhang, et al., SHEsisEpi, a GPU-enhanced genome-wide SNP–SNP interaction scanning algorithm, efficiently reveals the risk genetic epistasis in bipolar disorder, Cell Res. 20 (2010) 854–857, http://dx.doi.org/10.1038/cr.2010.68.
[17] B.K. Bulik-Sullivan, P.-R. Loh, H.K. Finucane, et al., LD Score regression distinguishes confounding from polygenicity in genome-wide association studies, Nat. Genet. 47 (2015) 291–295, http://dx.doi.org/10.1038/ng.3211.
[18] Z.M. Ibrahim, S. Newhouse, R. Dobson, Detecting epistasis in the presence of linkage disequilibrium: a focused comparison, CIBCB (2013) 96–103, http://dx.doi.org/10.1109/CIBCB.2013.6595394.
[19] E.M. van Leeuwen, F.A.S. Smouter, T. Kam-Thong, et al., The challenges of genome-wide interaction studies: lessons to learn from the analysis of HDL blood levels, PLoS One 9 (2014) e109290, http://dx.doi.org/10.1371/journal.pone.0109290.
[20] T. Kam-Thong, C.-A. Azencott, L. Cayton, et al., GLIDE: GPU-based linear regression for detection of epistasis, Hum. Hered. 73 (2012) 220–236, http://dx.doi.org/10.1159/000341885.
[21] S. Lee, M.-S. Kwon, T. Park, CARAT-GxG: CUDA-accelerated regression analysis toolkit for large-scale gene–gene interaction with GPU Computing system, Cancer Inform. 13s7 (2014), http://dx.doi.org/10.4137/CIN.S16349, CIN.S16349.
[22] C.C. Chang, C.C. Chow, L.C. Tellier, S. Vattikuti, S.M. Purcell, J.J. Lee, Second-generation PLINK: rising to the challenge of larger and richer datasets, Gigascience 4 (2015) 1–16, http://dx.doi.org/10.1186/s13742-015-0047-8.
[23] S. Purcell, B. Neale, K. Todd-Brown, et al., PLINK: a tool set for whole-genome association and population-based linkage analyses, Am. J. Hum. Genet. 81 (2007) 559–575, http://dx.doi.org/10.1086/519795.
[24] M.D. Atkinson, J.-R. Sack, N. Santori, et al., Min–max heaps and generalized priority queues, CACM 29 (10) (1986) 996–1000, http://dx.doi.org/10.1145/6617.6621.
[25] J.C. K&rdquo;assens, A Hybrid-parallel Architecture for Applications in Bioinformatics, no. 2017/4 in Kiel Computer Science Series, Department of Computer Science, Faculty of Engineering, Kiel University, CAU Kiel, 2017, http://dx.doi.org/10.21941/kcss/2017/4, Dissertation.
[26] L. Wienbrandt, J.C. K&ldquo;assens, et al., Fast genome-wide third-order SNP interaction tests with information gain on a low-cost heterogeneous parallel FPGA-GPU computing architecture, Proc. Comput. Sci. 108 (2017) 596–605, http://dx.doi.org/10.1016/j.procs.2017.05.210.
[27] J.C. K&rdquo;assens, L. Wienbrandt, et al., Combining GPU and FPGA technology for efficient exhaustive interaction analysis in GWAS, 2016 IEEE 27th Int. Conf. ASAP (2016) 170–175, http://dx.doi.org/10.1109/ASAP.2016.7760788.
[28] L. Wienbrandt, J.C. Kässens, J. Gonzalez-Dom nguez, et al., FPGA-based acceleration of detecting statistical epistasis in GWAS, Proc. Comput. Sci. 29 (2014) 220–230, http://dx.doi.org/10.1016/j.procs.2014.05.020.
[29] S. Purcell, C. Chang, PLINK v1.90p 64-bit, www.cog-genomics.org/plink/1.9/ (9.01.2018).
[30] Wellcome Sanger Institute, Sanger Imputation Service, https://imputation.sanger.ac.uk/.

Lars Wienbrandt received his M.Sc. (Dipl.-Inform.) in 2009 from Kiel University, Germany. For his doctoral thesis he was working on implementation and parallelization of bioinformatics algorithms on FPGAs at the Department of Computer Science at Kiel University. He received his degree (Dr.-Ing.) in 2016 and now continues his work on hardware accelerators in bioinformatics including FPGAs and GPUs at the Institute of Clinical Molecular Biology in Kiel.

Jan Christian Kässens received his M.Sc. and doctoral degree (Dr.-Ing.) in Computer Science in 2012 and 2017, respectively, from the Technical Faculty of Kiel University, Germany. His main research focus lies on the development of special-purpose hardware and software for high-performance compute architectures in bioinformatics at the Institute of Molecular Biology, Kiel University, Germany.

Matthias Hübenthal received his M.Sc. (Dipl.-Bioinform.) in Bioinformatics from the Martin Luther University of Halle-Wittenberg, Germany, in 2011. He is currently a research assistant and doctoral student at the Institute of Clinical Molecular Biology, Kiel, Germany. His research interests include statistical modeling approaches for biomarker detection and genetic analyses as well as their implementation based on heterogeneous computing architectures.

David Ellinghaus received his M.Sc. (Dipl.-Bioinf.) in Bioinformatics from the University of Hamburg, Germany, in 2007. In 2012, he earned his doctoral degree (Dr. rer. nat.) from Kiel University, Germany, and afterwards conducted post-doctoral research at the University of Southern Denmark. In 2015, he became professor for "Biomedical Informatics" at the Institute of Clinical Molecular Biology, Kiel University. His main research interests are in the areas of genetic/epidemiological studies for immune mediated diseases, bioinformatics and statistical genetics.