High Performance Linkage Disequilibrium: FPGAs Hold the Key

Nikolaos Alachiotis
Carnegie Mellon University
nalachio@cmu.edu

Gabriel Weisz
Carnegie Mellon University
gweisz@cmu.edu

ABSTRACT

DNA sequencing technologies allow the rapid sequencing of full genomes in a cost-effective way, leading to ever-growing genomic datasets that comprise thousands of genomes and millions of genetic variants. In population genomics and genome-wide association studies, widely used statistics such as linkage disequilibrium become computationally demanding when thousands of whole genomes are investigated. Long analysis times and excessive memory requirements usually prevent researchers from conducting exhaustive analyses, sacrificing the ability to detect distant genetic associations. In this work, we describe a generic algorithmic approach for organizing arbitrarily distant computations on full genomes, and to offload operations from the host processor to accelerators. We explore FPGAs as accelerators for linkage disequilibrium because the bulk of required operations are discrete, making them a good fit for reconfigurable fabric. We describe a versatile and trivially expandable architecture, and develop an automatic RTL generation software to search the design space. We find that, when thousands of genomes from complex species such as humans, are analyzed, current FPGAs can achieve up to 50X faster processing than state-of-the-art software running on multi-core workstations.

Keywords

population genomics; linkage disequilibrium; accelerator

1. INTRODUCTION

Linkage disequilibrium (LD) is a fundamental technique that is widely used in population genomics and genome-wide association studies (GWAS) in order to understand how mutations interact with each other within a population. The term, coined in 1960 by Lewontin and Kojima [13], refers to the non-random association between alleles (variants of a gene) at different loci. Traditionally, LD-based analyses are conducted on human genomes to gain insight into the genetic composition of populations, as well as changes in this genetic composition that are driven by evolutionary forces such as natural selection and genetic drift [8]. Other applications of LD that are of significant importance include disease-gene mapping to understand genetic factors for inherited diseases [20], detecting drug-resistant mutations in pathogens such as HIV [9, 16], or revealing reasons for drug treatment failures [5]. With improvements to DNA sampling techniques and the continuous sequencing of genetic material from more species of flora and fauna, LD-based analyses are now even more pervasively used in the study of population genetics beyond that of humans [21]. The importance of LD-based analyses can be observed from titles of population genetics papers declaring "Population genomics: Linkage disequilibrium holds the key" [10].

Genomic datasets that are suitable for population genetic analyses comprise thousands of genomes from different individuals (also referred to as samples), and millions of genetic variants (e.g., the 1000 Genomes project, *http://www.1000genomes.org*). Genetic variation is observed in the form of so-called single-nucleotide polymorphisms (SNPs), a term that is used to describe single base-pair changes in a DNA sequence. The amount of genetic variation (discovered polymorphisms) increases with the number of sequenced genomes, leading to escalating dataset sizes. Roughly speaking, doubling the sample size leads to doubling the number of associated variants discovered [24]. Computing LD on this abundance of data can become prohibitively expensive, both in terms of long execution times as well as excessive memory requirements. Typically, researchers opt to narrow the extent of LD computations, using subsets of the available SNPs, in favor of significantly reduced analysis time and memory footprint, sacrificing the ability to detect highly correlated but potentially distant SNPs. Therefore, it is essential to devise high performance implementations that enable rapid and wider LD evaluations on complex populations such as humans ($\approx$ 10 million SNPs in the human genome).

Modern microprocessors are not adequately equipped to deliver high performance for LD computations. While the increase in transistors keeps pace with the rate at which molecular data become available (Fig. 1), the current trend to increase core performance by increasing the SIMD width does not yield a significant performance boost in LD calculations. The performance bottleneck for LD is the calculation of allele and haplotype (pairs of alleles on the same chromosome that tend to be inherited together) frequencies in a population. Since alleles are typically encoded as one- or two-bit entities (depending on certain biological assumptions such as the infinite sites model [12]), the allele and haplotype frequency calculation relies mainly on the enumeration of set

FPGA'16, February 21 - 23, 2016, Monterey, CA, USA

ISBN 978-1-4503-3856-1/16/02...$15.00

DOI: http://dx.doi.org/10.1145/2847263.2847271

bits in registers, an operation widely known as population count. While current microprocessor architectures provide hardware support for population count, in the form of an intrinsic instruction (since Intel Nehalem and AMD K10), the specific intrinsic command operates strictly on regular registers and does not support wider SIMD registers. Therefore, potential performance benefits from the use of SIMD instructions are diminished due to the required employment of a scalar instruction for the population counter at the microkernel level. To avoid reader confusion, we henceforth refer to the bit-enumeration operation as "popcount".

This study explores the potential of FPGAs to accelerate LD computations as required by large-scale population genomic and GWAS analyses. In addition to the apparent need for reducing execution time, a challenge lies in enabling arbitrarily wide LD evaluation on full genomes, a computation exhibiting memory requirements that typically exceed the amount of available memory on present-day accelerator platforms. The contribution of this work is two-fold:

- We implement a versatile accelerator architecture for faster analyses under the widely adopted infinite sites model [12]. In the interest of conducting a rapid design space exploration, as well as reducing development time and debugging effort, the architecture is described in the C language, through a series of custom RTL-specific library calls. Each of the custom library calls augments various instances of a unified data structure, where each instance represents a building component. A Verilog backend is deployed to generate the Verilog RTL description for the accelerator architecture. We find that performance gains become more prevalent (up to 2X higher throughput) when a moderate number of sufficiently wide, pipelined popcount operators are employed in contrast to an excessive number of narrow popcount operators.
- We present a generic algorithm to execute on the host processor in order to schedule and offload computations to the accelerator platform. The host implements an iterative procedure that splits large genomes into variable-size subgenomic regions depending on the distribution of SNPs along the genome, while ensuring that each subgenomic region is of the same size in terms of number of SNPs. The regions are paired on the host in order to cover all required LD computations and the pairs are offloaded to the accelerator. On the FPGA, dedicated circuitry organizes the SNPs into groups and carries out the LD computations for every pair of regions by pairwise combining all the SNP-groups in both regions. This approach facilitates the expansion of the hardware design to comprise more processing cores while not requiring changes on the software side. Furthermore, it permits LD evaluation between arbitrarily wide and potentially distant subgenomic regions without conducting redundant operations or requiring excessive memory space.

The remainder of this paper is organized as follows: In Section 2, we describe the mathematical operations and concepts required to compute LD. Thereafter, we discuss related work on software implementations in Section 3. Section 4 presents the proposed accelerator system, while Section 5 provides implementation details and a design space analysis. In the following Section 6 we present a performance evaluation, and conclude in Section 7.

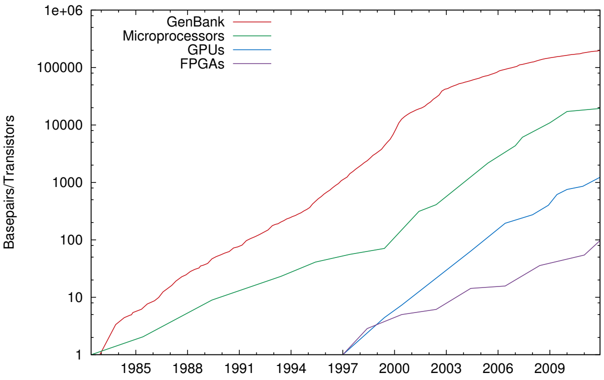

Figure 1: Growth of molecular data available in GenBank as well as of number of transistors in microprocessors, GPUs, and FPGAs (source: [1]).

2. LINKAGE DISEQUILIBRIUM (LD)

The primary source of genetic variation is the mutation. Population genetics investigate mutations by analyzing lists of individuals and their associated DNA sequences. Prior to the analysis, a so-called multiple sequence alignment (MSA) is computed, i.e., a $n \times m$ matrix that comprises n rows (one row per individual) of m columns each (also referred to as alignment sites). If a mutation has occurred at an alignment site, this site becomes a SNP, to differentiate from monomorphic alignment sites which are non-informative for evolutionary purposes. Therefore, a SNP is by definition polymorphic as it represents a single-nucleotide substitution of one DNA base (A, C, G, and T) for another. An example of an MSA of 4 individuals that comprises 30 alignment sites of which 5 are SNPs (highlighted in blue and italics) is shown below.

```
sample_0     ATGGCATACCCCT-CCAACTAGGATTCCAA
sample_1     ATGGCCTACCACTCCCAACTAGGCTTCC-A
sample_2     ATGGCATAC-C-TCCCAAGTAGGTTTTC-A
sample_3     ATGG----CCCCTCCCAACTTGGTTTCCAA
```

2.1 Representation of Genomic Data

Real-world analyses and *in silico* simulations (analyses on synthetic datasets) often adopt the infinite sites model [12] (henceforth denoted ISM). Under this model, it is assumed that there is an infinite number of sites, and consequently each new mutation appears on a site where previously no mutation has occurred. Due to the ISM assumption, SNPs can be represented by binary vectors, where each unset bit ('0') represents the initial–before mutation–state (usually referred to as the ancestral state), while each set bit ('1') indicates a new–after mutation–state (usually referred to as the derived state)[1]. For the rest of this paper, we assume that each SNP is represented by a group of N_{word} w-bit-long words with N_{word} defined as follows:

$$N_{word} = \left\lceil \frac{N_{seq}}{w} \right\rceil,$$

with zero padding if $N_{seq} \mod w \neq 0$, N_{seq} is the number of samples, and $w = 2^p$ with $p \in \mathbb{N}$. This also implies

[1] It should be noted that the assignment of '0' to the ancestral state is arbitrary. One could easily have used '1' to represent the ancestral and '0' for the derived state.

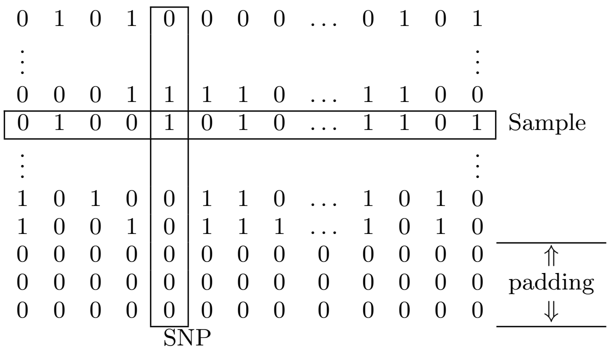

Figure 2: Pictorial representation of samples in the $(k \times w) \times n$ genomic matrix, G. Each row in G represents the genome of an individual (sample). Columns represent SNPs at different locations within the genome.

that the entire genomic dataset can be represented by a $(k \times w) \times n$ matrix, G, where $k = N_{word}$. Each row in G represents a particular sample while each column represents a SNP. We call this the genomic matrix, which is illustrated in Figure 2. We assume here that all monomorphic sites have been discarded through an initial prefiltering stage, and consequently the genomic matrix G is dense and consists solely of SNPs. For clarity reasons, we omit the SNP locations in Figure 2 but keep in mind that neighboring SNPs in the genomic matrix representation can be thousands of sites apart in the genome. In future discussions we represent a SNP as a column vector s.

2.2 Computing LD

The fundamental concept behind the computation of LD deals with the probability of independent events. If the probability of two mutations occurring at two sites in the same sequence is not the same as the product of the probabilities of the two mutations occurring independently, then the event that two mutations appear in the same sequence is said to be not independent, or the two SNPs are in linkage disequilibrium. Mathematically, we want to compute

$$D_{i,j} = P_{i,j} - P_i P_j, \tag{1}$$

for every pair of SNPs, s_i and s_j, where $P_{i,j}$ represents the probability that a sample has mutations in both SNPs, s_i and s_j, and P_i and P_j are the probabilities for the independent events that a mutation has occurred in s_i and s_j, respectively. When $D = 0$, s_i and s_j are in linkage equilibrium, i.e., mutations in s_i and s_j occur independently of each other. More interestingly, $D \neq 0$ if the two SNPs are in linkage disequilibrium.

Given SNPs of length N_{seq}, and ignoring (for the moment) that each SNP is stored as N_{word} words, the probability that a mutation occurs in a SNP s_x, denoted as P_x, can be obtained with the following equation:

$$P_x = \frac{POPCNT(s_x)}{N_{seq}}, \tag{2}$$

which counts the number of '1's in s_x and then divides it with the total number of bits in the s_x.

$P_{i,j}$ can be similarly computed by first counting the number of samples that have mutations in both SNPs, s_i and s_j, and then dividing that number by N_{seq}. Mathematically, $P_{i,j}$ can be computed as follows:

$$P_{i,j} = \frac{POPCNT(s_i \& s_j)}{N_{seq}} \tag{3}$$

Using Equations 2 and 3, we can then compute $D_{i,j}$ for all possible pairs of SNPs, s_i and s_j, using Equation 1.

The formulation of LD in Equation 1 can become problematic since the sign and range of $D_{i,j}$ vary with the choice of representation and the frequency at which different mutations occur. This makes it difficult to compare $D_{i,j}$ across different genomic regions. As such, several standardization methods have been proposed for D, with a commonly used one being the squared Pearson coefficient r_{ij}^2:

$$\begin{aligned} r_{ij}^2 &= \frac{(P_{i,j} - P_i P_j)^2}{P_i P_j (1 - P_i)(1 - P_j)} \\ &= \frac{D_{i,j}^2}{P_i P_j (1 - P_i)(1 - P_j)} \end{aligned} \tag{4}$$

This measure of LD has the advantage that all r_{ij}^2 values remain between 0 and 1, with higher numbers representing stronger association. More importantly, even with this representation of LD computation, notice that the cost of computing the $r_{i,j}^2$ values for all pairs of SNPs is dominated by the cost of D.

3. RELATED WORK

Advances in modeling and statistical analysis for population genetics in the past decade [14, 17, 15] were not accompanied by high performance computing approaches due to the limited amount of available molecular data. More recently, driven by sequencing cost reductions that have generated a plethora of molecular data suitable for population genetics analyses, several software tools capable of analyzing genome-scale datasets have been released.

Pfeifer et al. [18] released PopGenome, an R package for population genetics analyses that can compute a wide range of statistics, including LD, on whole-genome SNP data. Although PopGenome can exploit multiple cores for faster execution, the computational kernel for pairwise LD assessment is not optimized to exploit intrinsics and the cache hierarchy.

Chang et al. [6] released a comprehensive update to the widely used PLINK software [19] for whole-genome association and population-based linkage analyses (over 9,000 citations according to Google scholar). The updated implementation (PLINK 1.9) exhibits significant performance and scalability improvements in comparison with the initial software. It heavily relies on bitwise operations, multithreading, and high-level algorithmic improvements for the most compute-demanding functions, such as distance-based clustering and LD-based pruning. PLINK 1.9 implements the squared Pearson coefficient as a measure of LD and deploys the SSE2-based Lauradoux/Walish popcount algorithm to achieve high performance [6].

Alachiotis et al. [4, 3] released the population genomics software OmegaPlus. Similarly to PLINK 1.9, OmegaPlus computes the squared Pearson coefficient as a measure of LD. The implementation differs at the microkernel level since the performance-critical popcount operation is computed via the intrinsic popcount instruction supported in hardware.

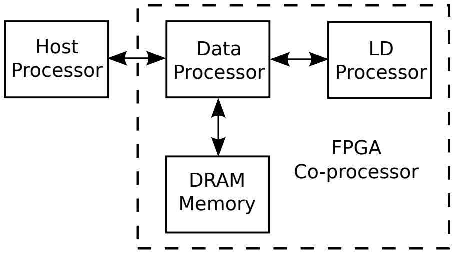

Figure 3: Accelerator system design overview.

FPGA researchers have successfully accelerated compute-intensive bioinformatics kernels in the past, such as sequence alignment [22] and phylogeny reconstruction [25]. However, to the best of the author's knowledge, the work presented here is the first to demonstrate the efficiency of FPGA technology for the acceleration of large-scale LD computations.

4. SYSTEM DESIGN

Our proposed accelerator system consists of four main components (Fig. 3). The host processor parses an input dataset in any of the widely used formats for genomic data (FASTA or VCF for real data, ms [11] or MaCS [7] for simulated data), and constructs the genomic matrix by filtering out the monomorphic–non-informative–sites. Thereafter, it executes the generic algorithm for scheduling and offloading computations to the FPGA co-processor. On the FPGA, the data processor handles the communication/synchronization with the host, as well as the data accesses to/from the on-board DRAM, whereas the LD processor is the computational kernel that carries out the population genomics calculations. Decoupling the data handling (data processor) and the execution (LD processor) parts on the co-processor permits the deployment of LD processors of any size (varying number of LD cores and/or popcount width) without requiring any changes to the data processor, which facilitates the exploration of the design space.

4.1 Host Processor

As already mentioned, a key challenge in conducting wide LD analyses efficiently is to avoid redundant computations. Our solution requires a pair of genomic regions of arbitrary size (number of SNPs), A and B, as input for any LD computation. While current software implementations compute all LD scores between SNPs in the same genomic region, allowing to provide a pair of regions as input attacks the problem of computing long-range LD efficiently. Association studies that investigate long-range LD in genomes might not require the computation of LD scores between SNPs located inbetween the distant regions of interest. For instance, if a study investigates association between two distant genes, computing the association between all intermediate genes as well is only going to result in many unnecessary calculations and excessive memory requirements. Thus, unlike existing software implementations, the generic algorithm on the host can consider only the SNPs in the distant genes to reduce execution time and memory footprint, while the single-region case can be trivially facilitated by setting $A = B$.

To enable the computation of arbitrarily large genomic datasets, the genomic matrix is organized by the host processor into large chunks, depending on the size of available main memory on the co-processor. Thereafter, each chunk is organized into a number of non-overlapping SNP-groups of fixed size S SNPs (e.g., $S = 128$). A subset of the SNP-groups in the same chunk (if $A = B$) or in two different chunks (if $A \neq B$) are used to construct two SNP-group lists, A_{list} and B_{list}, achieving full coverage of the regions of interest in A and B, respectively. Each pair of SNP-groups is a compute-group that entails all pairwise LD computations between the SNPs in the two SNP-groups. Consequently, each pair of SNP-group lists defines a compute-list that implicitly describes all required LD computations between the two regions of interest in A and B. Once the compute-list is constructed, it is offloaded to the co-processor for execution. Upon completion and return of the results, the steps of compute-list construction and offloading are repeated for the next chunk or pair of chunks until the entire genomic matrix is completed. Figure 4 illustrates the compute-list, where each tile represents the output of a compute-group, essentially a $S \times S$ matrix of LD values.

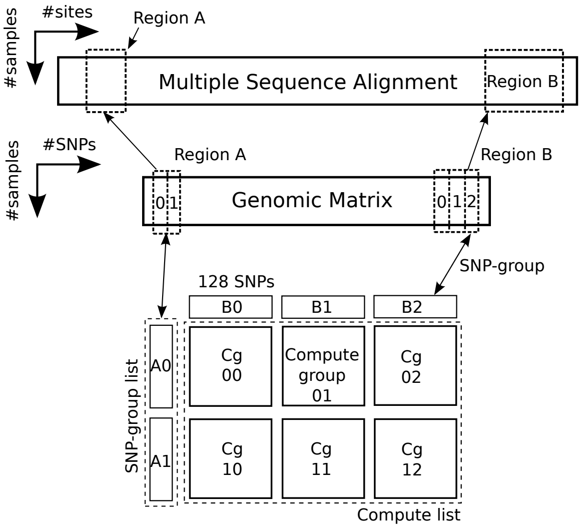

Figure 4: Illustration of the compute-list for arbitrarily distant LD computations.

4.2 Data Processor

The data processor architecture is depicted in Figure 5. To facilitate the trivial instantiation of variously sized design points (varying LD processor size), the data processor is implemented as a series of parameterized FSMs that are configured and controlled by a non-parameterized top-level FSM following a master-worker scheme. The top-level FSM, denoted DTOP in Figure 5, parses a set of configuration instructions that are used by the host processor to offload computations to the LD processor. These instructions are classified as follows: a) load instructions, b) dataset instructions, and c) store instructions. The DTOP FSM decodes the incoming instructions and configures the associated FSM of each class. Once all three worker FSMs are configured, DTOP initiates the LD computations by enabling the three FSMs. Thereafter, the LD processor is controlled by the worker FSMs which cooperate to load the genomic data from main memory, schedule operations on the LD processor, and store the results back in memory. To avoid increased row buffer misses and achieve high memory bandwidth, the SNP-

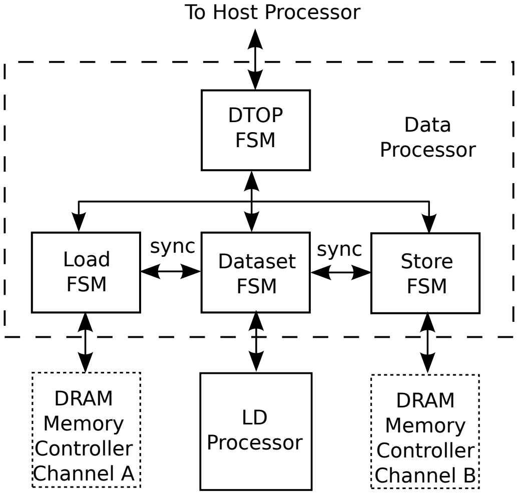

Figure 5: The data processor architecture.

group size S is appropriately set by the host processor to ensure that the SNP-groups, which are retrieved by the data processor during computations, do not exceed the DRAM row buffer size.

4.3 LD Processor

The LD processor architecture is depicted in Figure 6. The processing scheme is similar to searching a database for given queries. The SNPs in one of the two input regions are the queries while the SNPs in the other region are the database objects. A fraction of the query SNPs are initially loaded into a dual-port multi-bank memory subsystem (Data Mem A in Fig. 6). Thereafter, the database SNPs are passed through the two compute grids of LD cores (LD Core Grids A and B in Fig. 6) which calculate the LD scores between the database SNPs and all the query SNPs in Data Mem A. Query SNPs are provided to the two grids of LD cores through the two available ports on every DM Bank. Only a small fraction of the query SNPs are expected to fit in the Data Mem A on-chip memory, thus several iterations are needed to complete the required all-to-all LD calculations. To improve performance, double buffering is employed for loading the database SNPs from memory and streaming them through the LD Core Grids. For this reason, it is of vital importance for performance that the DM Bank size is chosen carefully so that it can store a sufficiently large number of query SNPs to allow enough time to refill the second database input buffer. This is because, each SNP in the database input buffer (not shown in the figure, it is part of the data processor) is parsed as many times as half the number of SNPs in a DM Bank before all query SNPs in Data Mem A are pairwise combined with the database SNPs in any of the two input buffers.

An LD Core Grid consists of $x \times y$ cores, organized into y rows of x cores each. The LD processor instance in Figure 6 contains 8 cores per grid ($x = 4$, $y = 2$). Every LD core matches a query SNP to a database SNP. Every SNP is simultaneously streamed through an LD core and an AFC block (Allele Frequency Calculator, not part of the LD core), which computes the frequency of the allele represented by the set bits. Note that, it is possible to avoid instantiating AFC blocks when genomic sequences without missing data are analyzed, since the per-SNP allele frequencies can be computed during the prefiltering stage and added to the genomic matrix as metadata.

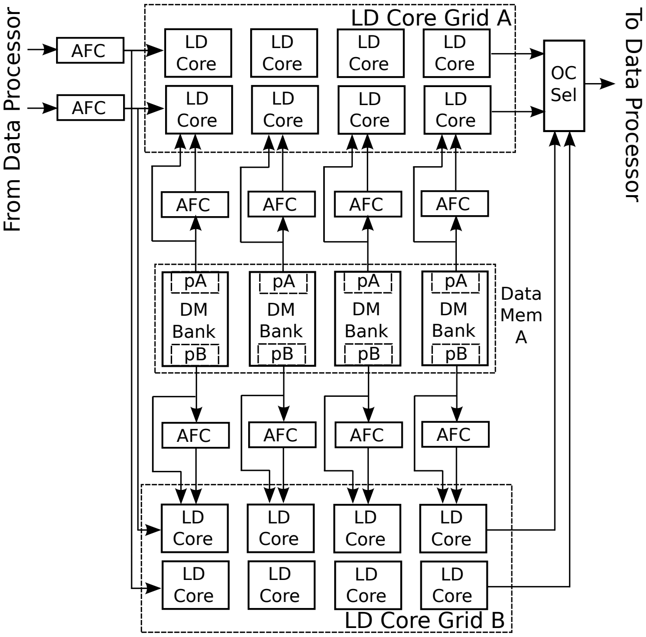

Figure 6: The LD processor architecture for a design point that exhibits 16 LD cores.

Figure 7 shows the pipelines of the AFC block and the LD core to demonstrate how the AFC block operations overlap with the HFC block operations (Haplotype Frequency Calculator), which is part of the LD core. Both AFC and HFC blocks contain a pipelined popcount implementation (POPCNT), which exhibits an array of ROM blocks at the lowest level and an adder reduction tree afterwards, to count the existing alleles and haplotypes, respectively. The SNPs are loaded in segments of size equal to the POPCNT input width. Therefore, the final count of alleles and haplotypes is only available at the output of the accumulator (ACCUM) after all the samples per SNP have been loaded. The frequencies are computed through a floating-point division (DIV) with the total sample size. Note the additional component in the HFC block, denoted HAPCONST, which constructs a bit vector where every set bit represents a valid haplotype. Currently, this is implemented as a logical *and* operation, which covers the cases of high-quality real-world datasets as well as simulated datasets. Additional logic is required in HAPCONST to account for alignment gaps or missing data in the dataset.

When the allele and haplotype frequencies are computed (output of AFC and HFC blocks in Figure 7), the LD Pipe implements Equations 1 and 4 to compute D and r^2, respectively. Changing the LD metric would only require changing the floating-point LD Pipe, and would not affect the rest of the LD processor. The results are temporarily stored in a FIFO buffer. The OC Sel module of the LD processor (see Fig. 6) retrieves LD scores from the FIFOs in a Round-Robin fashion, and transfers them to the data processor. Scores from several FIFOs may be retrieved simultaneously to consume all the available memory bandwidth.

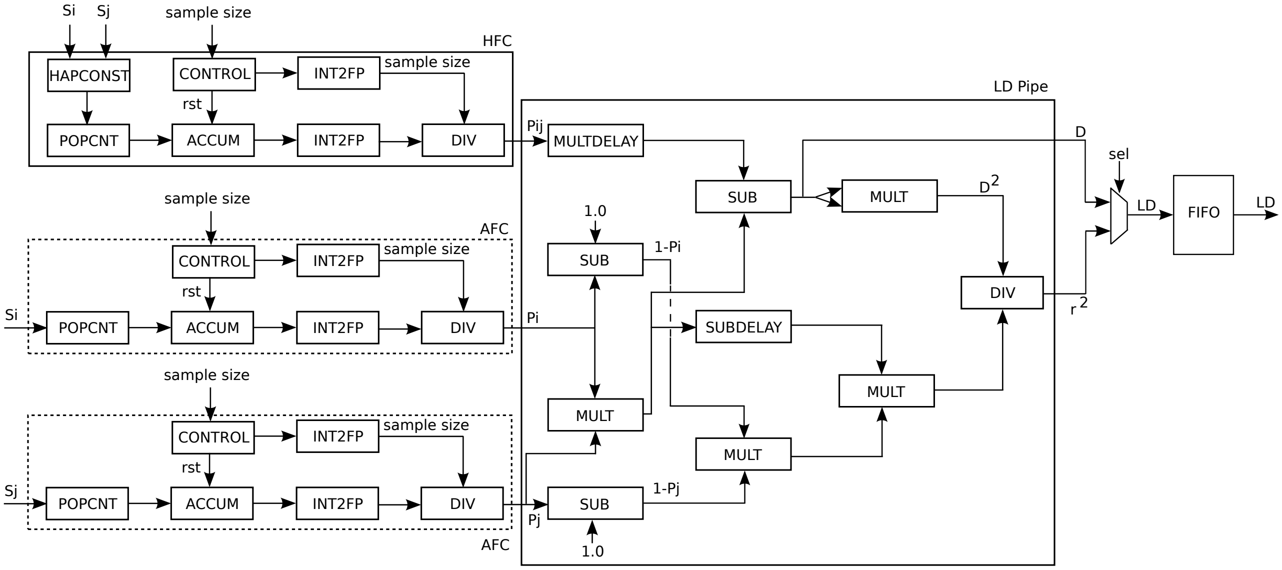

Figure 7: Pipelined datapath of the LD core (HFC and LD Pipe) and the two AFC blocks (one for the query SNP and one for the database SNP) that provide the allele frequencies.

5. IMPLEMENTATION

The generic algorithm executed on the host processor is implemented in C and the parallel processing mechanism of the compute list was modeled using the OpenMP API for parallel programming. The RTL description of the data processor is directly implemented in Verilog HDL, while the LD processor is automatically generated by an appropriately adapted RTL generation software implemented in C, which was initially designed for chemical similarity assessment [2]. Note that, the LD Pipe is also directly implemented in Verilog HDL, and employs single-precision floating-point cores. In the following, we present the input parameters for the RTL generation software, describe the verification process, and analyze the design space of the LD processor.

5.1 Automatic RTL Generation

A single software invocation generates a multi-module Verilog file that contains an LD processor configuration. Two types of tunable parameters are exposed to the user: Type S and Type R. Type S parameters are used to specify the size of the LD processor, such as the LD Core Grid size (number of LD cores vertically and horizontally), the size of the popcount units (input width and latency), the width of the accumulator units (to support arbitrarily large sample sizes), the size of the Data Mem A memory subsystem (number of SNP segments per DM Bank), the size of the FIFO buffer in every LD core (number of floating-point results), and the FIFO-group size (number of FIFOs grouped together to saturate the available memory bandwidth). The optional Type R parameters can be employed to achieve a more balanced distribution of the FPGA resources to the building blocks. The use of Type R parameters deploys synthesis attributes individually for each one of the Core Grids, the Data Mem A memory, and the LD core FIFOs. The popcount units and the accumulators in the LD cores can be implemented on DSP blocks or FPGA LUTs, while the DM Banks and the FIFOs can be mapped on block or distributed RAM. A summary of the parameters is presented in Table 1, while the RTL generation software is available for download at *https://github.com/alachins/fpga-ld*.

Table 1: RTL generation parameters

	Parameter	Description
Type S	m	popcount width (bits)
	t	popcount latency (cycles)
	n	accumulator width (bits)
	x	LD cores horizontally
	y	LD cores vertically
	z	words in DM Bank
	r	words in FIFO
	o	number of FIFOs grouped together
Type R	*gba*	AFC synthesis code[2]
	gbb	HFC Core Grid A synthesis code[2]
	gbbs	HFC Core Grid B synthesis code[2]
	qmem	DM Bank memory type[3]
	rmem	FIFO memory type[3]

5.2 Verification

To verify correctness of the accelerator architecture, we generated several LD processor instances and initially conducted extensive post-place and route simulations using Modelsim 6.3f. Additionally, we verified correctness in hardware, deploying a Xilinx ZC706 board with a Zynq Z-7045 FPGA. We configured a Verilog testbench to run a limited-performance design point (8 LD cores, 32-bit popcount operators) at 200 MHz, and used ChipScope to monitor the output of the LD processor.

5.3 Design Space Analysis

Given a sample size l, the popcount width and latency are set according to the following formulas in order to avoid

[2] Synthesis codes: 0 for BRAMs and DSPs, 1 for BRAMs and LUTs, 2 for Distributed memory and DSPs, 3 for Distributed memory and LUTs

[3] Memory type: 0 for BRAM, 1 for Distributed memory

excessive allocation of resources (Eq. 5) and to ensure that the bit-enumeration logic does not become the critical path in the design (Eq. 6).

$$m \le l \tag{5}$$

$$t \ge log2(m) \tag{6}$$

The above formulas ensure that we do not instantiate unnecessarily wide popcount units, i.e., popcount units with input width larger than our target dataset's sample size, as well as restrict the computational path between registers in the adder reduction tree structure to one addition at most, in order to maximize the popcount operating clock frequency.

The LD processor size (total number of LD cores) is given by Eq. 7. Each of the two LD Core Grids contains $x \times y$ cores.

$$Cores_{total} = 2 \times (x \times y) \tag{7}$$

Assume a DRAM bandwidth B, a floating-point precision $prec$ (bit width), and a function $f(B, prec)$ which calculates the number of results that can be offloaded to DRAM per clock cycle. The minimum number of LD cores that need to be instantiated to ensure that performance is not bounded by computation is provided by Eq. 8, while Eq. 9 sets the size of the FIFO groups, i.e., a parameter that configures the OC Sel module to fully utilize the available bandwidth B.

$$Cores_{min} = \left\lceil \frac{l}{m} \right\rceil \times f(B, prec) \tag{8}$$

$$o = f(B, prec) \tag{9}$$

As already mentioned, the data processor employs double buffering to stream one of the two genomic regions, the one considered the database. Each database buffer can store a fixed number of d SNPs, which is set to 64 SNPs in our system. A prerequisite for performance is to overlap the database buffer loading time with computations, which can be achieved by tuning the z parameter according to d. Consider a function $c(d \times y, l, B)$ which calculates the number of clock cycles required to retrieve $d \times y$ SNPs of length l from DRAM at an available bandwidth B. The DM Bank depth z must satisfy the following:

$$d \times \left\lceil \frac{z}{2 \times \lceil \frac{l}{m} \rceil} \right\rceil \times \left\lceil \frac{l}{m} \right\rceil \ge c(d \times y, l, B), \tag{10}$$

where the left-hand side is the number of clock cycles required by an LD core to compute the scores (e.g., D or r^2) between d database SNPs in one of the two database buffers and half of the query SNPs in a DM Bank. Assuming that l is the smallest multiple of m (with zero padding) and z is restricted to be a multiple of $2 \times \lceil \frac{l}{m} \rceil$, then the above formula can be simplified as follows:

$$z \ge \frac{2 \times c(d \times y, l, B)}{d}. \tag{11}$$

6. PERFORMANCE EVALUATION

To assess performance of the FPGA-accelerated system, we used a workstation with an Intel Xeon E5-2630 6-core Sandy Bridge processor running at 2.60 GHz and 32GB of main memory, as a test platform for the software benchmark. For the hardware evaluation, a series of design points were generated and mapped on a Virtex 7 VX980T-2 FPGA. We assess throughput performance in terms of million LD scores per second (mLD/sec). Note that this metric is tightly dependent on the sample size. In our comparisons, we use a subset of 10,000 SNPs from the first chromosome of the human genome (available from the 1000 Genomes project, *http://www.1000genomes.org*), with a sample size of 2,504 genomes (phase 3).

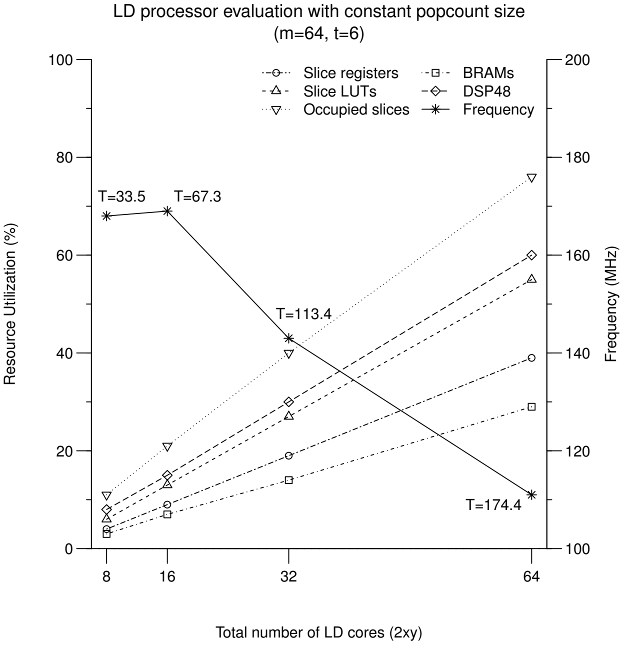

Figure 8: Post-PAR resource utilization, maximum operating clock frequency, and throughput performance T (mLD/sec, based on 2,504 samples) when the total number of LD cores increases.

6.1 Tuning Type S Parameters

Initially, we explored the design space of the LD processor by tuning only Type S parameters while maintaining the default Type R values, which map all memory components to BRAMs and attempt to maximize the utilization of DSP slices. Figure 8 illustrates post-place and route (PAR) resource utilization, maximum operating clock frequency, and estimated throughput performance for an increasing number of LD cores. When the number of cores increases, by tuning the x and y parameters, the popcount width and latency are constant and set to $m = 64$ bits and $t = 6$ cycles, respectively. The figure reveals that the highest throughput performance (based on a sample size of 2,504 genomes) which can be achieved with the current configuration is 174.4 mLD/sec. Furthermore, it is observed that DSPs and FPGA slices rapidly become the limiting factors and prevent the instantiation of additional LD cores on the target device. This is not surprising, particularly because the LD pipe heavily relies on floating-point operators.

Thereafter, we explore performance for increasing popcount width and latency. For this configuration, the total number of LD cores is fixed to 16, i.e., 8 LD cores per grid, by setting $x = 8$ and $y = 1$. Figure 9 demonstrates that the highest throughput performance now is 349.4 mLD/sec. This performance improvement over the largest previous

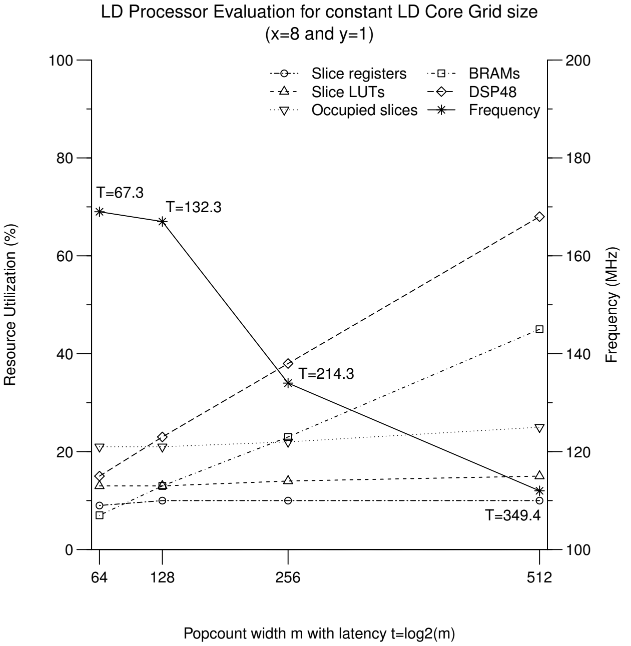

Figure 9: Post-PAR resource utilization, maximum operating clock frequency, and throughput performance T (mLD/sec, based on 2,504 samples) when the popcount width increases.

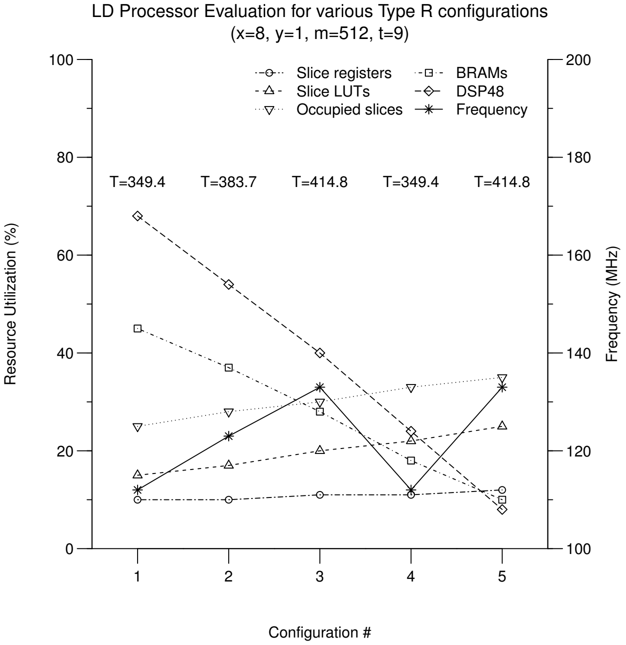

Figure 10: Post-PAR resource utilization, maximum operating clock frequency, and throughput performance T (mLD/sec, based on 2,504 samples) for various R Type configurations.

configuration (Figure 8) comes from the wider popcount operators which significantly reduce the latency for computing the allele and haplotype frequencies. Reducing the latency for computing or estimating allele and haplotype frequencies has great significance for the population genomics community. The computational demands for computing allele/haplotype frequencies grow with an increasing sample size, and the sample sizes of present-day datasets are only expected to increase (due to the continuous advances in DNA sequencing technologies), whereas the number of SNPs is bounded by the chromosomal length. Furthermore, in addition to association studies that rely on LD, rapid allele and haplotype counts are also required for the evaluation of mutation rates in a population [23], or to investigate the genetic differentiation between populations.

6.2 Tuning Type R Parameters

Driven by the observation that wider popcount operators yield better performance than larger LD Core Grids, we now employ Type R parameters to redistribute the resources of the LD processor configuration that exhibits the largest bit-enumeration-per-cycle capacity, i.e., $x = 8$, $y = 1$, $m = 512$, and $t = 9$. There are five Type R parameters that can be tuned (see Table 1), however only the first three have an effect on the utilization of DSP slices, which is currently the resource with the highest percentage of occupancy. Figure 10 shows resource utilization and performance as more LD processor components are mapped to FPGA LUTs and distributed memory instead of DSP slices and block RAMs. The default LD processor configuration (#1) fully utilizes DSPs and block RAMs. The next two configurations map the HFC blocks of one (#2) and both (#3) LD Core Grids to LUTs and distributed memory. In addition to all HFC blocks, the final two configurations also map one (#4) and both (#5) arrays of AFC blocks to LUTs and distributed memory.

As can be observed, configuration #4 exhibits the lowest resource utilization over all resources, which is achieved at the price of a longer clock cycle and consequently poorer throughput performance. From the figure, we conclude that it is beneficial to accept a slight increase in occupied slices (configuration #5 versus #4) in order to boost throughput performance by nearly 19%.

6.3 Refining the LD Processor Size

Configuration #5 exhibits the highest throughput performance without excessively occupying the available resources on the target device. This permits to increase the LD processor size to boost performance further. At this point, we opt to continue exploring the design space by increasing the x parameter only, which will add more LD cores to the LD Core Grids horizontally. Increasing the LD Core Grid size by adding more cores vertically (y parameter), or widening the popcount operators (m parameter), requires additional memory bandwidth to sustain performance. On the other hand, placing more LD cores horizontally allows to increase parallelism by matching more query SNPs to the same number of incoming database SNPs.

Figure 11 reveals that increasing the size of the LD processor from 32 to 40 LD cores leads to a significantly longer clock cycle (maximum operating clock frequency drops from 137 MHz to 104 MHz) due to increased routing effort, which diminishes potential throughput performance improvements from the placement of more LD cores in parallel in each LD Core Grid. As can be observed, a 40-core LD processor exhibits poorer performance than a 32-core instance.

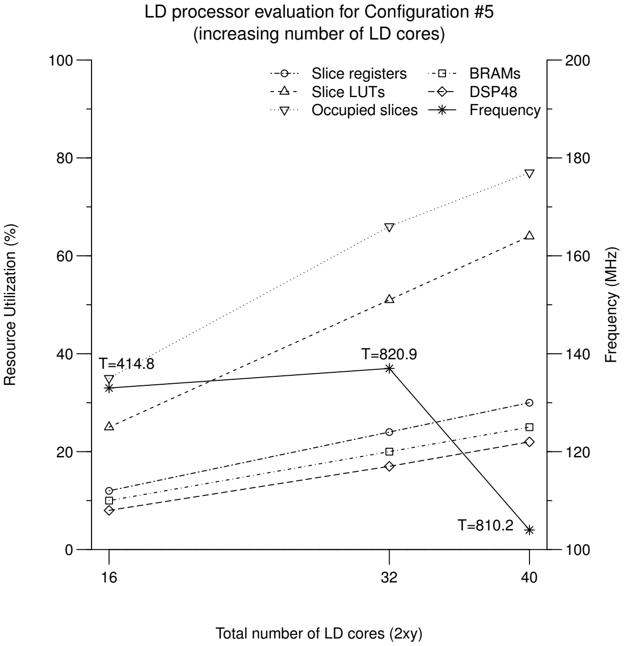

Figure 11: Post-PAR resource utilization, maximum operating clock frequency, and throughput performance T (mLD/sec, based on 2,504 samples) for Type R configuration #5 (all HFC and AFC blocks occupy distributed memory and FPGA LUTs) and increasing LD processor size (number of LD cores).

6.4 Comparison with Reference Software

The open-source tool PLINK 1.9 [6] was selected as the reference software due to superior parallel efficiency in comparison to OmegaPlus [4]. We conducted a preliminary performance comparison (results not shown) between PLINK 1.9 and the multi-grained parallel version of OmegaPlus [3], which was proposed by the authors to improve scalability of the tool. We measured throughput performance when both tools compute $10^4(10^4+1)/2$ LD scores. We observed that, for small numbers of threads (up to 4), both tools exhibit comparable throughput performance (OmegaPlus was up to 15% faster), whereas when the number of threads increases, PLINK 1.9 is up to 2X faster. Therefore, we opted to compare the FPGA-accelerated system against PLINK 1.9 to ensure an accurate performance evaluation based on the current state-of-the-art software.

Table 2 shows execution times and throughput performance for the analysis of the test dataset that comprises 10,000 SNPs from the genomes of 2,504 humans. The FPGA performance refers to the fastest LD processor configuration detected through the design space exploration, which comprises 32 LD cores and exhibits 512-bit-wide popcount units with 9 clock cycles latency, while all HFC and AFC blocks are mapped to FPGA LUTs and distributed memory. The comparisons include the memory access time, and exclude the amount of time required for prefiltering.

It could be argued that the execution times in Table 2 are too short to justify the need for acceleration. One should consider however that the 10,000 SNPs in the test dataset account for roughly 0.1% of the total amount of variants in the human genome. Typically, whole-genome analyses implement a sliding-window approach, where each window covers a narrow subgenomic region that comprises a limited number of SNPs (10,000 SNPs is a reasonable size for such a window). To cover the entire genome of complex species for instance, tens of thousands of sliding-window iterations may be needed, requiring several hours or days to complete an analysis on current workstations. Furthermore, PLINK 1.9 allows only a single region as input, which would require prohibitively large memory footprint on the test platform in order to cover larger genomic space in our tests, whereas our proposed generic algorithm accepts two subgenomic regions as input, which permits the sequential scheduling and calculation of arbitrarily distant LD associations.

Table 2: Performance comparison between PLINK 1.9 software (6-core Xeon CPU) and a 32-core LD processor instance (Virtex 7 FPGA).

	PLINK 1.9		FPGA LD Proc.
Threads	Exec. time (sec)	mLD/sec	Speedup (X)
1	12.3	4.1	200.2
2	9.6	5.2	157.8
4	5.9	8.4	97.7
8	3.8	13.0	63.1
12	3.0	16.4	50.1

Table 3 provides execution times and throughput performance when two simulated datasets with larger sample sizes are analyzed. Both datasets, D.1 and D.2, comprise 10,000 SNPs, while exhibiting sample sizes of 10,000 and 100,000 sequences, respectively. The 32-core LD processor achieves throughput performance of 205.7 mLD/sec for dataset D.1, and 20.4 mLD/sec for dataset D.2.

Table 3: Performance comparison based on simulated datasets D.1 and D.2 with sample sizes of 10,000 and 100,000 sequences, respectively.

	PLINK 1.9				FPGA LD Proc.	
Threads	Exec. time (sec)		mLD/sec		Speedup (X)	
	D.1	D.2	D.1	D.2	D.1	D.2
1	41.1	389.1	1.2	0.128	171.4	159.3
2	31.4	297.6	1.6	0.168	128.5	121.4
4	19.2	180.2	2.6	0.277	79.1	73.6
8	11.3	109.4	4.4	0.456	46.8	44.7
12	9.9	88.3	5.0	0.566	41.1	36.0

7. CONCLUSIONS

In this study, we explored the potential of FPGAs to be employed as accelerators that boost performance of linkage disequilibrium computations in population genomics and genome-wide association studies. We developed a hardware generation software that allowed rapid design space exploration that reached several high performance design points. Generating custom accelerator instances enables the deployment of the largest/fastest design point for the analysis of the genomic dataset at hand. This has great significance in real-world analyses that typically need to conduct thousands of simulations on synthetic datasets with similar characteristics (e.g., demographic model) and size as the real dataset under investigation to determine whether the results are statistically significant. We find that FPGAs can achieve up to 50X faster analysis than current software implementations executed on multi-core workstations. Finally, we described a generic algorithm that is executed on the host and enables the offloading of long-range association statistics without conducting redundant operations or requiring excessive memory resources.

As future work, we intend to port the generic algorithm on GPUs to explore the potential performance gains, as well as on systems that comprise different types of accelerator devices such as GPUs and FPGAs to exploit the aggregate computational capacity of such heterogeneous systems. Additionally, we intend to further develop the automatic generation software to accommodate other computationally intensive statistics that can benefit from rapid allele and haplotype frequency computations.

Acknowledgments

The authors thank Tze Meng Low (CMU) for constructive discussions, valuable insights on LD, and comments on the manuscript. Furthermore, the authors thank Xilinx for tools and hardware donated to CMU. This work is supported in part by NSF CCF-1320725 and by Intel.

8. REFERENCES

[1] N. Alachiotis. *Algorithms and Computer Architectures for Evolutionary Bioinformatics*. PhD thesis, München, Technische Universität München, 2012.

[2] N. Alachiotis. Generating FPGA accelerators for chemical similarity assessment. In *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, pages 1–4. IEEE, 2015.

[3] N. Alachiotis, P. Pavlidis, and A. Stamatakis. Exploiting multi-grain parallelism for efficient selective sweep detection. In *Algorithms and Architectures for Parallel Processing*, pages 56–68. Springer, 2012.

[4] N. Alachiotis, A. Stamatakis, and P. Pavlidis. OmegaPlus: a scalable tool for rapid detection of selective sweeps in whole-genome datasets. *Bioinformatics*, 28(17):2274–2275, 2012.

[5] M. T. Alam, D. K. de Souza, S. Vinayak, S. M. Griffing, A. C. Poe, N. O. Duah, A. Ghansah, K. Asamoa, L. Slutsker, M. D. Wilson, J. W. Barnwell, V. Udhayakumar, and K. A. Koram. Selective sweeps and genetic lineages of Plasmodium falciparum drug -resistant alleles in Ghana. *The Journal of infectious diseases*, 203(2):220–7, Jan. 2011.

[6] C. C. Chang, C. C. Chow, L. C. Tellier, S. Vattikuti, S. M. Purcell, and J. J. Lee. Second-generation PLINK: rising to the challenge of larger and richer datasets. *Gigascience*, (4), 2015.

[7] G. K. Chen, P. Marjoram, and J. D. Wall. Fast and flexible simulation of DNA sequence data. *Genome research*, 19(1):136–42, Jan. 2009.

[8] J. F. Crow, M. Kimura, et al. An introduction to population genetics theory. *An introduction to population genetics theory.*, 1970.

[9] N. G. de Groot and R. E. Bontrop. The HIV-1 pandemic: does the selective sweep in chimpanzees mirror humankind's future? *Retrovirology*, 10(1):53, Jan. 2013.

[10] D. B. Goldstein and M. E. Weale. Population genomics: linkage disequilibrium holds the key. *Current Biology*, 11(14):R576–R579, 2001.

[11] R. R. Hudson. Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics (Oxford, England)*, 18(2):337–8, 2002.

[12] M. Kimura. The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations. *Genetics*, 61(4):893, 1969.

[13] R. Lewontin and K. Kojima. The evolutionary dynamics of complex polymorphisms. *Evolution*, pages 458–472, 1960.

[14] H. Li. A new test for detecting recent positive selection that is free from the confounding impacts of demography. *Molecular biology and evolution*, 28(1):365–75, Jan. 2011.

[15] R. Nielsen, S. Williamson, Y. Kim, M. J. Hubisz, A. G. Clark, and C. Bustamante. Genomic scans for selective sweeps using SNP data. *Genome research*, 15(11):1566–75, Nov. 2005.

[16] P. S. Pennings, S. Kryazhimskiy, and J. Wakeley. Loss and recovery of genetic diversity in adapting populations of HIV. *PLoS genetics*, 10(1), 2014.

[17] P. Pfaffelhuber, A. Lehnert, and W. Stephan. Linkage disequilibrium under genetic hitchhiking in finite populations. *Genetics*, 179(1):527–537, May 2008.

[18] B. Pfeifer, U. Wittelsbürger, S. E. Ramos-Onsins, and M. J. Lercher. PopGenome: An Efficient Swiss Army Knife for Population Genomic Analyses in R. *Molecular biology and evolution*, 31(7):1929–36, 2014.

[19] S. Purcell, B. Neale, K. Todd-Brown, L. Thomas, M. A. Ferreira, D. Bender, J. Maller, P. Sklar, P. I. De Bakker, M. J. Daly, et al. PLINK: a tool set for whole-genome association and population-based linkage analyses. *The American Journal of Human Genetics*, 81(3):559–575, 2007.

[20] D. E. Reich, M. Cargill, S. Bolk, J. Ireland, P. C. Sabeti, D. J. Richter, T. Lavery, R. Kouyoumjian, S. F. Farhadian, R. Ward, et al. Linkage disequilibrium in the human genome. *Nature*, 411(6834):199–204, 2001.

[21] M. Slatkin. Linkage disequilibrium-understanding the evolutionary past and mapping the medical future. *Nature Reviews Genetics*, 9(6):477–485, 2008.

[22] E. Sotiriades and A. Dollas. A general reconfigurable architecture for the BLAST algorithm. *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, 48(3):189–208, 2007.

[23] F. Tajima. Statistical method for testing the neutral mutation hypothesis by dna polymorphism. *Genetics*, 123(3):585–595, 1989.

[24] P. M. Visscher, M. A. Brown, M. I. McCarthy, and J. Yang. Five years of GWAS discovery. *The American Journal of Human Genetics*, 90(1):7–24, 2012.

[25] S. Zierke and J. D. Bakos. FPGA acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods. *BMC bioinformatics*, 11(1):184, 2010.