TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRONICS AND COMPUTER ENGINEERING
MICROPROCESSORS AND HARDWARE LAB

Design and Implementation of a multi-FPGA Acceleration System for Large-scale Population Genomics Analyses based on Linkage Disequilibrium

Author:
Dimitrios BOZIKAS

Supervisor:
prof. Apostolos DOLLAS

March 7, 2017

Abstract

Modern sequencing technologies have contributed to the creation and rapid expansion of DNA databases, already numbering thousands of whole genomes. The astonishing rate at which genomic data are being collected, combined with the fact that it has outpaced Moore's law, establishes the necessity of the development of novel tools, capable of conducting large-scale genomics analyses efficiently. This work addresses the computational challenges inherent to the analysis of linkage disequilibrium (LD) levels in large-scale datasets. LD is a statistic that quantifies the non-random association between alleles at different genomic locations. While it contributes to a wide variety of genomics and genetics analyses, the compute- and memory-intensive operation of counting set bits (population count) in large vectors, required for the estimation of LD, hinders the efficient use of modern CPUs for such analyses, mainly due to the lack of a vectorized population counter. To overcome this obstacle, we present a novel hardware architecture for the calculation of pairwise LD scores based on reconfigurable logic. The proposed accelerator exploits the ability of reconfigurable machines to be programmed at the hardware level. The effective use of multiple levels of parallelism, combined with the efficient manipulation of the data structures on memory through the transformation of the memory layout, result in high throughput capabilities for the estimation of LD in arbitrarily large datasets. The architecture is, subsequently, mapped onto a high-performance heterogeneous computing platform that enables the parallel cooperation of 4 reconfigurable devices, while, simultaneously, providing a high-speed memory interface. The implemented accelerator is evaluated for analyses of simulated genomic data of varying sizes, through its comparison with corresponding state-of-the-art parallel software implementations, achieving speedups between 6.35X and 134.93X, depending on the dataset size. Concerning real-world analyses, such as scanning the 22nd chromosome of the human genome, the accelerator is capable of potentially achieving quintupled throughput when compared to highly optimized reference software running on multiple cores.

Acknowledgements

I would like to thank my supervisor, professor Apostolos Dollas for his trust in my abilities and his guidance during the course of this study, as well as for the opportunity he gave me to experiment with a novel and interesting platform and expand my horizons beyond my current field of expertise and into the fascinating domain of genetic biology.

I would also like to express my deepest gratitude to Dr. Nikolaos Alachiotis for his guidance, cooperation, creative conversations and the amount of time and energy he dedicated to helping me see this work realized. Furthermore, I would like to thank Dr. Euripides Sotiriades for his guidance and useful advice during a large part of the pursuit of this thesis. Additionally, I would like to thank Dr. Pavlos Pavlidis of the FORTH for his valuable input on matters of population genomics, as well as entrusting me, along with Dr. Nikolaos Alachiotis, with the task of building this work on their algorithm.

I would like to thank Emmanouil Kousanakis and Christos Roussopoulos for their valuable technical advice. Without them this work would not have been fully realized. I would also like to thank Dr. Gregory Chrysos for always happening to be nearby when I had a technical difficulty that otherwise seemed impossible to address. I would like to give my kind regards to all my fellow students at the office for providing a much-needed clarity of mind when my work seemed to have reached an impasse.

In addition, I would like to thank all my dear friends for standing beside me when I needed their support the most.

Finally, I would like to thank my brother, Panagiotis, for his moral support and his tolerance over the times I was a lousy roommate, and my parents and grand-parents for supporting me materially and immaterially and always allowing me to discover my own path in my life.

Dimitrios Bozikas
Chania, 2017

Contents

List of Figures

List of Tables

List of Abbreviations

LD	**L**inkage **D**isequilibrium
SNP	**S**ingle **N**ucleotide **P**olymorphism
GWAS	**G**enome-**W**ide **A**ssociation **S**tudies
MSA	**M**ultiple **S**equence **A**lignment
ISM	**I**nfinite **S**ites **M**odel
FSM	**F**inite **S**ites **M**odel
HWE	**H**ardy-**W**einberg **E**quilibrium
MC	**M**emory **C**ontroller
MCP	**M**emory **C**ontroller **P**air
BC	**B**it-**C**ount
AHF	**A**llele/**H**aplotype **F**requency
MSC	**M**utation/**S**tate **C**ounter
MCFSM	**M**emory **C**ontroller **F**inite **S**tates **M**achine
AEH	**A**pplication **E**ngine **H**ub
AE	**A**pplication **E**ngine

Dedicated to my family and friends.

Chapter 1

Introduction

The rapid technological advancements of recent years have paved the path for the refinement of DNA sequencing tools. Cheaper and faster processing units, paired with novel algorithms, have yielded cost-efficient solutions providing high accuracy and throughput for the sequencing of vast amounts of genomic data, in a scale never before precedented. Contemporary genomic databases already contain thousands of whole genomes, a number which is rapidly increasing. The highly successful 1,000 genomes project [56], for example, achieved the full sequencing of the genomes of 2,504 individual humans across 26 populations by 2015, long surpassing its initial goals. It's successor, the 100,000 genomes project [19], has set the goal of having sequenced 100,000 human genomes by the end of 2017, while numerous other localized projects have sequenced portions of individual populations [21].

It is predicted that, by 2025, between 100 million and 2 billion genomes will have been sequenced, of humans alone. This constitutes a four to five orders of magnitude growth in ten years, classifying Genomics as one of the Big Data domains, vastly overshadowing the growth of data sizes of the rest domains, namely Astronomy, Youtube and Twitter [55]. Currently, sequencing capacity doubles every seven months [55], meaning that conventional computing machines cannot continue to efficiently process data at this scale, as the rate by which genomic data are being sequenced has long outpaced Moore's Law. As a result, the need for novel custom solutions capable of efficiently processing future large-scale datasets has arisen.

1.1 Motivation

This work is focused around a widely used statistic in population genomics called linkage disequilibrium (LD) [34]. LD describes the non-random association between alternative forms of a gene, known as alleles, at different loci in a population, giving the capability to compute correlations between mutations that are inherited together. The practical applications of LD, either as a standalone statistic or as the basis of subsequent analyses, are numerous. The inclusion of past evolutionary events within the shape of LD in humans [49], for instance, has significant implications for disease gene mapping. Additionally, LD achieves higher accuracy and sensitivity than alternative statistics in analyses revolving around tests of positive selection [15]. Such analyses, when applied to populations of pathogens, may possibly lead to the development of more effective drugs [4], through the discovery of drug-resistant mutations in the population. In the scope of genome-wide association studies (GWAS) [59], focused around identifying mutations associated with diseases, LD enables the screening of the dataset for meaningful interactions between target highly linked loci, thus reducing the computational load of subsequent analyses.

LD calculations are conducted on so-called multiple sequence alignments (MSAs). An MSA can be represented as a structure of $m \times l$ dimensions consisting of m DNA sequences of l nucleotide sites each. If a column contains more than a single nucleotide base, therefore denoting the presence of one or more mutations in the specific site, it is categorized as a single-nucleotide polymorphism (SNP). Concerning LD-based analyses, non-polymorphic sites do not contain any meaningful information and, as a result, the dataset must be vetted prior to such analyses, so that it only contains SNPs. For a dataset containing m SNPs and l sequences to be correlated by SNP pairs in its entirety, a quadratic time complexity of a factor of $O(m \times n^2)$ is expected [59].

As a result, LD calculations represent a challenging Bioinformatics subject, in terms of both computational and memory standpoints. This is especially true in relation to the ever-growing data sizes that sequencers produce. On the one hand, the calculation of LD scores requires the application of the population count operation, that is the enumeration of set bits in registers, on the SNP vectors. With increasing sample sizes, this operation quickly becomes the bottleneck of the LD calculation, due to the increased length of the SNP vectors. Furthermore, the sequencing of more genomes inevitably leads to wider genetic variance within the dataset, thus inflating both the number of sequences and the number of SNPs of the produced datasets. This directly translates to increased computational and memory requirements for processing larger datasets.

To address the issues that large datasets present, several encoding models of genomic data have been proposed, aiming for the minimization of the computational load and the memory footprint. The infinite sites model [29] consists a widely used and simplified encoding method, which uses a binary system to designate the presence of mutations in a population based on a single reference sequence. In this work, however, the focus revolves around the compute-intensive, yet more informative, finite sites model (FSM) [24], which encodes each nucleotide base in its own vector, thus allowing the account of multiple mutations per locus. This design decision is motivated by the increase in interest around the particular model in recent biological studies [40]. Despite the several successful attempts at optimizing the LD algorithm and in contrast to the continuation of Moore's law, LD still presents a challenge for the computational sciences, mainly due to the lack of a vectorized population counter in current microprocessor architectures [1]. Thus we attempt to present a different approach of calculating LD by means of reconfigurable hardware.

1.2 Scientific Contributions

The scientific contribution of this work is focused on two aspects. We present a generic FPGA-based architecture to accelerate LD computations focused on processing arbitrarily large numbers of whole genomes. The design itself is agnostic towards the overlying application, rendering it able to be used in a large variety of different bioinformatics algorithms, provided the necessary minor changes. Driven by the volume of future datasets, we present the design decisions that were considered along the development and optimization process of the architecture for the accommodation of very large data sizes. The proposed accelerator architecture estimates LD under the FSM assumption, however, minor changes for ISM support are presented. In the case of the FSM model, which is the main focus of this work, DNA ambiguous characters (to correct for DNA sequencing errors and nucleotide base mis-calls [27]) and missing data are accounted for.

We subsequently implement the proposed architecture as a single-FPGA prototype system and employ a Convey HC-2ex platform, exploiting its high-end memory interface and synergistic quad-FPGA setup to enhance the performance capabilities of the fully implemented accelerator system. The implemented design is coupled with an efficient parallel algorithm for computing LD at the whole-genome scale, so as to deploy hardware acceleration efficiently for genome-wide analyses that comprise a large number of full genomes. To evaluate the performance of the implemented system, a series of comparisons based on simulated genomic data of up to 1 million sequences was performed between the accelerator and reference high-performance software implementations. Considering real-world applications, we estimate potential performance gains our system can achieve in the analysis for the detection of traces of positive selection in the 22nd chromosome of the human genome, based on real world data from the 1,000 genomes project [56].

As a synopsis, due to the ever-growing volume of genomic data being collected, along with the computational complexity of the problem at hand and the scarce hardware architectures that currently tackle it, we attempt to provide a first step towards a viable high-performance custom hardware architecture with real-world application capabilities.

1.3 Thesis Organization

The remainder of this work is organized as follows. In Chapter 2 we describe the mathematical operations and concepts for computing LD under both the ISM and the FSM evolutionary assumptions and analyze significant LD theoretical applications. In Chapter 3 we review related work on software and hardware solutions for population genomics and linkage analyses, while describing the LD computational core of the OmegaPlus algorithm [2] as the basis of our LD accelerator. Chapter 4 describes the proposed accelerator architecture and the necessary memory layout transformations that enable the efficient processing of large datasets, while Chapter 5 provides a short description of the target Convey HC-2ex platform, as well as the implementation details of the accelerator. In Chapter 6 we present the methodology followed for the verification of the architecture in detail and evaluate the performance of our accelerator by conducting comparisons with state-of-the-art parallel software implementations. Finally, Chapter 7 entails the conclusions drawn from this work, as well as proposals for improvements and possible studies that could stem from the current one.

Chapter 2

Theoretical Background

2.1 The Structure of Linkage Disequilibrium

Linkage disequilibrium (LD) quantifies the non-random association between alleles at different genomic locations. In the simplified case where no evolutionary forces are acting on the genomes during reproduction, such as recombination, mutation, genetic drift etc, all alleles that reside on the same chromosome are inherited to the offspring. In reality, however, several evolutionary forces contribute to the genetic constitution of the offspring's generation. LD is employed to identify which allelic combinations co-occur in a genome more frequently than expected by chance.

2.1.1 Biological Definition

According to Mendel's Law of Independent Assortment, alleles for separate traits are passed independently of one another from the parents to the offspring. In other words, the biological selection of an allele for one trait has nothing to do with the selection of an allele for any other trait. While this is true in principle, there have since been discovered several violations of the law, with genetic linkage consisting the most prominent one, stating that DNA sequences that are found in relatively short distances of one another within a chromosome exhibit a tendency to be inherited together during reproduction.

This can be explained when genetic recombination is taken into account. Recombination takes place during reproduction and refers to the possible exchange of genetic material between the homologous chromosomes of the parents, producing by extension a slightly different chromosome that is inherited by the offspring. In this case, the chance that two alleles remain together during recombination is proportional to the physical distance between them in the chromosome.

Furthermore, genetic material may also be subject to mutations, either spontaneous or induced ones. Spontaneous mutations occur mostly by chance, while induced mutations may appear due to environmental factors. The scale of a mutation is varying, while the manifestation can be in the form of deletion, duplication or inversion of a set of genes within a chromosome or even insertion or translocation of genes between different chromosomes. The effect of a mutation may be anything from virtually insignificant, beneficial or even extremely harmful and the cause of genetic disease.

When considering whole populations, several more evolutionary forces affect the gene pool, either by chance or choice. Such forces may be genetic drift, which refers to the change in the allele frequencies of a population due to random events, non-random mating, where part of the population exhibits a preference in reproduction with another part of the population that retains certain specific traits, as well as natural selection, which occurs when specific traits in a portion of the population

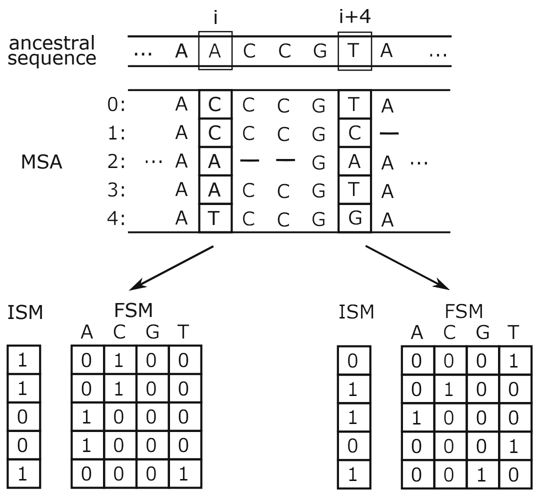


FIGURE 2.1: Example of an MSA that consists of an arbitrary number of sites, including two SNPs at locations i and $i+4$, along with their respective representations under the ISM and the FSM evolutionary models.

facilitate a higher chance of survival and, therefore, a higher chance to pass those traits down to the next generation.

Given all the aforementioned factors of the evolutionary process, linkage disequilibrium can be defined as the statistical measure that quantifies the degree to which alleles at different loci exhibit a non-random association. As a result, the rate with which the described evolutionary forces occur highly influences the pattern of LD, providing valuable information concerning allelic combinations that are both dependent on each other and deviate from a random association expected by chance.

2.1.2 Data Representation

LD computations are conducted on data structures that consist solely of SNPs, which we henceforth refer to as SNP maps. A SNP map is therefore a description of the existing mutations in a population along with their respective locations in the chromosome. The process of creating a SNP map begins with the sequencing of the genetic material of the individuals under investigation, which generates a DNA sequence per individual. In order to correctly represent the genetic variations between the collected data, an ancestral sequence must be selected as reference, thus producing a multiple sequence alignment (MSA). An example of an MSA that comprises 7 visible alignment columns and 5 sequences is shown in Figure 2.1, with hyphens inferring to missing genomic data. Note that, out of the 7 total sites only the 2 highlighted can be classified as SNPs, meaning that a mutation is present on the specific sites. The rest contain no useful information concerning LD, thus the MSA must be screened so that the final SNP map only contains polymorphic sites (i.e., SNPs).

While SNP maps are devoid of excess uninformative data, they remain an incomprehensible format in regards to computer processing. Therefore two options are presented for further encoding. The first, labeled as the infinite sites model (ISM), represents each SNP as a single binary vector. Such an encoding is achieved by comparing each allele of every individual to the corresponding allele of the ancestral sequence. If the comparison between the ancestral and a target sequence presents different nucleotide bases for an allele, thus denoting the presence of a mutation, the corresponding bit in the ISM alignment is set. In mathematical terms, given an allele a_i at a location i in the ancestral sequence S, the binary vector V describing SNP i is constructed as follows, where [] is the Iverson bracket notation:

$$V_i^{a_i} = \{[V_{i,0} = a_i], [V_{i,1} = a_i], ..., [V_{i,N-1} = a_i]\}. \tag{2.1}$$

Assume, for instance, the ancestral sequence of Figure 2.1, employed for a hypothetical analysis of the example MSA. The first SNP in the MSA, which is alignment site i, is therefore represented by the following binary vector:

$$V_1^A = \{1, 1, 0, 0, 1\}, \tag{2.2}$$

This example illustrates that, given a nucleotide base A in the i^{th} site of the ancestral sequence, three mutations are located in that site, at sequences zero and one with a nucleotide base C, and at sequence five with a nucleotide base T specifically, while sequences two and three contain the same base as the ancestral sequence.

The ISM offers a memory-efficient way of encoding large datasets in binary vectors. However, the trade-off for such efficiency is represented by the abstraction of information that occurs during the encoding process. Namely, we can deduce from the produced vector only the location of the mutation but not the information of the mutation itself. Furthermore, by using an ancestral sequence as a reference, mutations are only tracked based on that reference sequence and not between the individuals of the population under examination. One must also consider that if an analysis of a dataset must account for missing data and/or base miss-calls, an additional vector of equal length must be created for each SNP in order to validate the data, due to the fact that a binary encoding into a single vector does not have enough digits to include such information.

The second encoding method of SNP maps, which comprises the finite-sites model (FSM), utilizes a two-dimensional matrix to represent each SNP. The matrix's dimensions are equal to the number of sequences times four, with each of the four sub-vectors representing the presence or absence of each nucleotide base, namely adenine (A), guanine (G), thymine (T) and cytosine (C), at the corresponding location on the alignment. This method does not require a reference sequence to be presented, but a binary value of "1" is inserted at a position of a base sub-vector if the corresponding position in the alignment contains that base. Based on Equation 2.1, a SNP is described under the FSM as follows:

$$M_i = \{V_i^A V_i^C V_i^G V_i^T\}. \tag{2.3}$$

Again, Figure 2.1 illustrates the FSM encoding of our example MSA according to Equation 2.3.

At first glance, the finite sites model could be described as inefficient given that it requires 4 times the memory space compared to the ISM and it uses 4 bits to encode the 4 nucleotide bases instead of 2, which would be the obvious optimal choice. The 4-bit model is chosen due to the fact that DNA sequencing errors and base mis-calls [27] do not allow to accurately determine which of the 4 nucleotide bases exists at each location in the genome. Therefore, a total of 16 ambiguous DNA characters (see http://www.bioinformatics.org/sms/iupac.html) are employed in order to allow subsequent statistical analyses to account for such ambiguity. Furthermore, by not using a reference sequence, mutations may be tracked between each two individuals of the population, while the information of the nucleotide bases of the whole alignment is preserved, allowing for more intricate in-depth analyses.

2.2 Calculation of Linkage Disequilibrium Scores

In order to compute the LD score of a pair of SNPs, the frequencies with which mutations appear in each SNP and the pair combined must be deduced. These frequencies are henceforth denoted as allele and haplotype[1] frequencies respectively. When these statistics are known, an LD measure can be applied in order for the final score to be computed. For the sake of clarity, we initially describe LD when the ISM assumption holds, as it is also the basis for the operations required for the calculation of LD scores under the FSM model.

2.2.1 Allele and Haplotype Frequencies

In order to specify the allele and haplotype frequencies, first, the number of mutations at each SNP position as well as the number of common mutations appearing at both SNPs must be enumerated. Therefore, given a SNP at location i, the allele mutation count C is computed by the following equation:

$$C_i = \sum_{k=0}^{N-1} [V_{i,k} = 1]. \tag{2.4}$$

The aforementioned equation denotes the enumeration of set bits at locus i, or in essence, the enumeration of mutated alleles that appear in the sample SNP. This operation is henceforth referred to as population count operation.

Similar to Equation 2.4, albeit in a per-SNP-pair basis, the haplotype mutation count of a pair of SNPs located at loci i and j respectively, is given by the following equation:

$$C_{ij} = \sum_{k=0}^{N-1} [V_{i,k} = 1][V_{j,k} = 1]. \tag{2.5}$$

In this case, the equation represents the combined population count of the SNP pair. From a biological standpoint, this translates to the total number of genomes that exhibit the mutated allele at both chromosomal locations i and j.

When the allele and haplotype mutation counts are calculated, they can be normalized by the total number of sequences N in the alignment and produced the allele F and haplotype H frequencies as follows:

$$F_i = \frac{C_i}{N},\ H_{ij} = \frac{C_{ij}}{N}. \tag{2.6}$$

2.2.2 Measures of Linkage Disequilibrium

LD relies on the probability of independent events. Once the allele and haplotype frequencies for SNPs i and j are known, we can then compute LD as follows:

$$D_{ij} = H_{ij} - F_i F_j. \tag{2.7}$$

This equation calculates the difference between the probability of observing two mutations at two chromosomal locations in the same genome and the product of the probabilities of observing the two mutations independently. Biologically, this represents the degree to which the two mutations are likely to be dependent on each other, thus be inherited together during reproduction. To elaborate further, when the

[1] A haplotype is a set of mutations that are inherited together.

result of Equation 2.7 is equal to zero, the mutations at the chromosomal locations i and j occur independently of each other, thus can be said that SNPs i and j are in linkage equilibrium. On the other hand, in the case of a non-zero outcome, the mutations are not independent, but rather in linkage disequilibrium.

The formulation of Equation 2.7, however, is rarely used due to the fact that the sign and range of values that D assumes varies with the frequency that mutations occur, yielding the comparison of LD values highly problematic, even between genomic regions in the same chromosome. Even if the absolute value of D is used, leading to positively signed results, the range of values still presents a notable limitation for LD analyses. For this reason, several standardization methods have been proposed.

Lewontin [33] suggested normalizing D by dividing it by the theoretical maximum difference between the observed and expected allele frequencies as follows:

$$D'_{ij} = \frac{D}{D_m}, \tag{2.8}$$

where D_m is defined according to the sign of D as:

$$D_m = \begin{cases} \max[-F_iF_j, -(1-F_i)(1-F_j)], & \text{when } D < 0, \\ \min[F_i(1-F_j), F_j(1-F_i)], & \text{when } D > 0. \end{cases} \tag{2.9}$$

This measure significantly reduces the range of values produced and therefore constitutes a more accurate metric for linkage disequilibrium because the LD scores are disjoint from the fluctuation of allelic frequencies. However, when the sample size is small or one of the alleles under examination is rare, the scores reported have proven to be inflated, a fact which could potentially lead to false assumptions during an analysis, if it is not accounted for.

Another measure and, arguably, the most widely used one, relies on the Pearson's correlation coefficient, as shown by Hill and Robertson [22]. The squared Pearson's coefficient represents the focus of this work and can be described as:

$$r^2_{ij} = \frac{(H_{ij} - F_iF_j)^2}{F_i(1-F_i)F_j(1-F_j)}. \tag{2.10}$$

The fact that r^2 only assumes values in the range between one and zero presents multiple advantages over the rest of the proposed standardization methods. Mainly, the comparison between different genomic regions becomes much easier, while the interpretation of the scores becomes rather intuitive since values closer to one indicate high LD, whereas values closer to zero indicate reduced association. Equation 2.10 may also be multiplied by the number of sequences N in the alignment, and, by using Equations 2.6, it can be transformed in a simplified form as:

$$r^2_{ij} = \frac{(NC_{ij} - C_iC_j)^2}{C_i(N-C_i)C_j(N-C_j)}. \tag{2.11}$$

What has been described so far consists the measurement of LD scores according to the ISM model, which represents an approximation of the FSM assumption regarding the occurrence of mutations. Despite its simplified SNP representation, this approximation is designed to provide lower computational requirements and shorter analysis times. The computation of LD scores under the FSM model is not mathematically unrelated to the ISM. We could abstractly portray an LD score under the FSM as a series of LD computations using the ISM model. Each computation treats each pair of nucleotide bases, namely A, C, G and T, present at the pair of

SNPs as an independent ISM score. For that reason, a score computed under the FSM model may require up to 16 times more computations, in the case of all alleles being present in all SNPs.

To better illustrate the FSM computation method, assume a pair of SNPs, i and j and their respective sets of DNA characters, S_i and S_j, that contain the alleles found in this SNP pair. We may also describe the number of characters present at each allele as the size of the character set and denote them as V_i and V_j for the sets S_i and S_j respectively. Then, using Equation 2.10, we could describe LD under the FSM assumption as:

$$LD_{ij} = N\frac{(V_i - 1)(V_j - 1)}{V_i V_j} \sum_{m \in S_i} \sum_{l \in S_j} r_{ml}^2, \tag{2.12}$$

As an example, consider again Figure 2.1. In order to calculate an FSM LD score between the two SNPs that are illustrated, we would need the character sets of the aforementioned SNP pair, which would be $S_i = \{A, C, T\}$ and $S_{i+4} = \{A, C, G, T\}$, while their sizes are $V_i = 3$ and $V_{i+4} = 4$ respectively. Note that, in this example, the character G is absent from S_i, since guanine is not present in SNP i. In this case, the entire column referring to guanine in SNP i may be omitted, reducing the number of ISM computations required to 12.

2.3 Scientific Significance and Applications

Linkage disequilibrium has proven, since the 1980's, to be of high significance to biologists and geneticists due to its properties and usefulness in a wide range of different analyses [53]. Since technological progress has paved the path for gene-mapping, large-scale surveys upon genomic data and even genome-wide association studies, the plethora of factors that affect and are affected by LD have deemed it invaluable to evolutionary biology and genetics. The information it provides when examined throughout the whole genome can lead to deduction of a population's history [49] and various breeding and geographic patterns [37], as it encapsulates past evolutionary events, whereas an LD based analysis of a genomic region may shed light on the effect of evolutionary forces [34], such as natural selection, mutation etc.

The practical applications of LD so far have yielded numerous enlightening results. The inclusion of past evolutionary events in LD measurements, for instance, can provide valuable information for disease gene mapping by estimating the shape of LD in the human genome [49] and by mapping the LD patterns of complex diseases [57, 11]. Examples of such research around complex diseases include studies conducted in sub-genomic areas concerning genes that possibly associate with cancer [7], variations of genes linked with increased cancer risk among different human populations [30], analyses of random genetic events and their association with elevated risk of developing cancer [38] and the linkage between loci that may present susceptibility to highly complex diseases, such as type 1 diabetes [41], among others. Furthermore, when applied as a measure in genome-wide association studies (GWAS) or analyses of consequent data, it can further our understanding concerning mutations affecting or leading to genetic disease [31, 10, 14].

For the same reason, LD provides valuable insight into evolutionary biology concerning micro- and possibly macro-evolution of a population when its LD patterns over several generations are examined [15]. As a result, several studies have concluded on the significance of LD statistics as a means of tracking selective sweeps based on the genetic history of a population [42, 44]. More focused studies have even revolved around the estimation of the contemporary effective size of a population [60], the deduction of the admixture history in human populations [37] or even whether natural selection still occurs in contemporary populations [6]. Furthermore, LD may provide an indicator of the appearance of recent beneficial mutations [58] while it is also used to detect co-evolution between interacting genes [50], that is genes that mutate complementary in order to maintain the interaction between them.

Finally, valuable conclusions might be extracted from the analysis of LD patterns present in populations of harmful microorganisms [43]. In this case, further studies might be able to achieve the engineering of more potent drugs, thus better combating serious diseases, such as malaria.

Hopefully, through the examples presented above, we have succeeded in establishing the significance of linkage disequilibrium as a means of understanding the genetic origin of populations as well as aiding in the struggle for their future welfare. Therefore, we attempt to transform our theoretical understanding into a novel high-performance hardware architecture, with the aim to effectively address the need for faster LD calculations of large-scale datasets.

Chapter 3

Related Work

Due to its significance, as well as its computational complexity, linkage disequilibrium has become the subject of focus in various Bioinformatics studies, leading to numerous algorithmic models. Through the advancement of technology, several efficient implementations have been released, mainly confined in the scope of software. In this chapter, we attempt to present the more prominent ones, as well as relevant hardware implementations in order to establish a clear picture of the current state of the technological landscape around linkage disequilibrium implementations.

3.1 Software LD Implementations

The disproportion between the rate with which genomic data are being sequenced and Moore's law has established the need for fine-tuned high-performance software tools capable of efficiently processing LD scores of large-scale datasets. Since the early 2000's several successful attempts have been made.

LDA, a java-based linkage disequilibrium analyzer, was released by Ding, Zhou et al. [16]. The first stage of computations consists of either a Monte Carlo permutation or an x^2-test, depending on the user's choice, to check whether the alleles at each locus are in Hardy-Weinberg equilibrium (HWE)[1]. For those alleles that prove to be on HWE, an expectation-maximization algorithm is deployed to calculate the four haplotype frequencies of pairwise loci. Finally, several pairwise measures are implemented to quantify the degree of LD, such as the aforementioned D' and r^2 described in Section 2.2.2, while significant association between two loci is tested through a Monte Carlo simulation-based likelihood ratio test or an x^2 test. LDA is implemented as a graphical user interface, presents a significant variety of choices to the user, allowing for analyses tailored for different needs, and is capable of producing both text-based and graphical output. However, it lacks multi-core support and, therefore, is not suitable for analyses of large-scale datasets.

SNPStats is a web tool developed by Sole et al. [54] for association studies analyses. It is designed from a genetic epidemiology point of view and is focused on enabling research around association studies based on both SNPs and biallelic markers. The software package is able to compute a variety of metrics referring to both single-SNP (HWE) and multi-SNP (LD) association. The application itself is written using PHP, while the computational core is implemented as a series of R packages. However, the lack of parallelization of the computations hinders its performance.

VariScan, released by Hutter et al. [23], is a software suite for the analysis of DNA polymorphisms. It implements several genetic parameters, such as summary

[1]HWE states that allele and genotype frequencies will remain constant across generations in the absence of other evolutionary forces. It is not to be confused with LD, which quantifies the association between two or more alleles.

statistics of nucleotide and haplotype polymorphism levels, LD-based statistics and several coalescent-based tests of neutrality. Furthermore, VariScan allows for different genomic regions, functional categories or chromosome locations to be analyzed carefully. The main focus of the software revolves around searching for the genetic footprint of natural selection through the use of the aforementioned statistics. The software is written in ANSI C, supplemented by PERL scripts and a JAVA-based GUI, however, it lacks modern high-performance optimizations, such as multi-core support.

Haploview, released by Barett et al. [5], is a tool for haplotype analyses written in JAVA. Through a comprehensive GUI, the user is presented with several genomic metrics describing the input data, such as HWE, Mendelian inheritance errors etc. As a subsequent analysis, the program presents the option of pairwise LD calculations based on a wide array of LD measures (D', r^2, etc.) using either preset or user-defined groups of genetic markers. Other measures are also implemented, such as the Transmission Disequilibrium Test (TST), evaluating the presence of family-based genetic linkage between a genetic marker and a trait. Similar to LDA, due to the time the software was released, it suffers from a lack of a parallelized version. However, as it is licensed as open-software, it has been updated and built upon as recently as 2015, but its further support and development are reported as being currently frozen.

Another algorithm for LD and genetic association analyses implemented as a web interface is SNPAnalyzer 2.0, developed by Yoo et al. [66]. Information concerning the input dataset, such as allele/haplotype frequencies and analysis of HWE, is provided through a comprehensive user interface. The processing starts with the estimation of haplotypes in the dataset and, in turn, one of several LD measures is applied indicating the degree of association between a pair of SNPs. As a downstream analysis, haplotype estimation or LD data are used to search for genetic association between disease and haplotypes. The software itself is user-friendly since it implements a web-based GUI and supports both text and visual output formats. The computational core of the software is implemented in C, while the user interface in JAVA. However, even though it is reported that it performed better than various other similar software of its time, its sequential code is not suited for modern large-scale analyses.

A complete re-implementation of GenePop [48] was released by Rousset [51], implementing a variety of DNA statistic measurements. Such statistics include several HWE tests, tests of independence between contingency tables, either for spatial structure or two-locus composite linkage disequilibrium, allele frequencies and gene diversities at different hierarchical levels among others. However, despite the large volume of statistics that are implemented and the continuous support of the software until as recently as 2016, there is not an efficient parallel version of the software to address the issue of processing very large datasets.

The Arlequin suite, released by Excoffier et al. [17], is a software package integrating several statistics for population genetics data analysis. Such statistics include allele/haplotype frequencies based on maximum-likelihood estimation, parameters of a demographic or spatial expansion based on the mismatched distribution computed on DNA sequences, departure from Hardy-Weinberg equilibrium and various LD measures, among other intra- and inter-population statistics. The latest version [18] is implemented in C++, while the output is produced in HTML format with the option to be converted to XML, while an R package graphically represents the XML data. Furthermore, while a GUI is present, the use of the command-line versions of the software is advised by the authors for large-scale analyses. However, the absence of multi-core support is, once again, an obstacle towards the efficient processing of contemporary large-scale datasets.

GCTA is a tool for genome-wide trait analysis developed by Yang et al. [65]. The software is designed to address the issue of SNPs produced by GWAS not being able to fully explain the heritability of most complex human traits and diseases. As a result, the algorithm implemented is focused on estimating the variance for a complex trait based on SNPs on a chromosome- or genome-wide level, rather than testing the association of each SNP to the trait. The functionality of the software includes estimating the genetic relation of genome-wide SNPs, estimating the variance explained by genome-wide SNPs or the variance on the X chromosome as well as estimating LD and total heritability among others. From a computational standpoint, even though the cited publication presented a sequential version of the application, the software has since been updated to support multiple threads, thus operating on multiple processor cores (see http://cnsgenomics.com/software/gcta/index.html).

PopGenome[45], released by Pfeifer et al., consists a comprehensive R package that offers a variety of population genetics statistics, including LD. The focus of this work is to create a unified tool consisting of a wide range of statistics, all under the same software. Therefore, it supports a large number of widely-used input file formats, exhibits powerful numerical and graphical capabilities, is platform independent, and its open-source nature and comprehensive framework enables the easy assimilation of new features to the package. The set of implemented functionalities includes several neutrality and LD statistics, nucleotide and haplotype diversity tests, selective sweep analysis, recombination statistics and others. Furthermore, higher performance is achieved by implementing some computationally heavy calculations in C++, instead of R scripts, and by parallelizing parts of the software, thus enabling its multi-threaded execution. However, other features present in high-end processors, such as vector intrinsics or advanced caching techniques, are not exploited to their full potential.

PLINK [47], a widely used software for GWAS, received a significant update from Chang et al. [9]. PLINK 1.9, as the second-generation release is designated, builds upon all of its predecessor's functionalities, such as allele and genotype frequency computation and testing for HWE, while it employs LD patterns as a measure to test whether missingness at a site can be predicted by the local haplotypic background. From a performance standpoint, the software utilizes a lot of modern high-end processor features, such as bit-level parallelism, multi-core support and efficient use of memory by storing only a portion of the data on memory at any time. The LD measure itself is computed based on an efficient use of SSE2 vector intrinsics that calculate the squared Pearson's coefficient.

Finally, OmegaPlus, released by Alachiotis et al. [2], represents a scalable software implementation that scans whole genomes for traces of positive selection based on LD patterns. The ω statistic [28] is implemented as the measure of choice for the search of evidence of a population being affected by a selective sweep. Due to the way a beneficial mutation is observed to affect the genomic region surrounding, by reducing genetic diversity in a population affected by the selective sweep [28], the basis of the ω statistic is that a locus evaluated for traces of positive selection should present a higher cumulative LD score in the areas flanking the mutation than the area across the mutation, requiring a large number of pairwise LD calculations for the assessment of a single locus. Several LD measures are implemented, however, the squared Pearson's correlation coefficient (r^2) is selected as the default option by the software. Furthermore, multiple versions of the algorithm have been implemented in order to address certain performance issues. Apart from the sequential implementation of the algorithm, multiple parallel versions of the software have been released, including fine-, coarse- and multi-grain implementations utilizing Pthreads, as well as a generic parallel version built upon the OpenMP API. Furthermore, the software efficiently uses available memory, while the computational load is balanced in different ways depending on each of the aforementioned parallel versions. The LD core of OmegaPlus consists the basis of the accelerator system proposed in this document, therefore it will be elaborated upon in greater detail in Section 3.3.

3.2 Accelerators in Population Genomics

Even though the need for efficient tools that are capable of processing the ever-growing sizes of DNA datasets has given rise to multiple high-performance software implementations over the past two decades, the same does not hold true for the domain of reconfigurable hardware design or heterogeneous computing systems. While Bioinformatics, in general, has proven to be fertile ground for the development of high-performance heterogeneous applications and hardware architectures [46, 35, 68], the examples of such studies that are concerned with aspects of population genetics and genomics are scarcer.

Some of these works are focused on accelerating the procedure of creating MSAs [36, 39], which is a vital part of any subsequent GWAS analyses. A large portion of population genetic accelerators is focused on the phenotypic effect of the dependency between two genes, also known as epistasis, by utilizing GPUs [64, 25], FPGAs [61, 26, 8] and other heterogeneous architectures [20] in an effort to provide high-performance tools for such analyses. Other GPU-based designs are meant to study the systematic difference in allele frequencies between populations [52], or conduct isolation-by-migration model analyses between populations and even species [67]. Another interesting topic in population genomics consists of the simulation of the genetic composition of a population over time. These forward simulators, as they are called, are highly compute-intensive, but also exhibit a potential of highly parallel approaches, since such simulations are comprised of a large number of independent calculations, thus allowing the utilization of modern GPUs [32] to yield significant performance gains over corresponding CPU implementations. Another approach to performance acceleration is the use of SIMD architectures built in modern CPUs. When attempting to verify the correct alignment of short DNA sequences to a long genome (read mapping), for instance, it has been demonstrated [63] that an efficient SIMD implementation, paired with bit-level parallelism, can perform better than state-of-the-art implementations based on traditional CPU utilization, while still yielding highly accurate results.

Even fewer studies, however, revolve around the efficient computation of LD. Liu et al. [62] recently presented a GPU-based accelerator for pairwise LD measurements. This implementation is also based on the r^2 coefficient as a measure of pairwise SNP correlation. The design itself exploits the large amount of CUDA cores operating in parallel on modern GPUs, while data re-organization and atomic instructions are deployed to efficiently use available memory bandwidth. The authors report a thousand-fold speedup when the design is instantiated on a multi-GPU cluster, compared to sequential CPU-based versions of LD calculators.

To the best of the author's knowledge, the only other work addressing the computational issues of LD by utilizing reconfigurable hardware architectures, is the previous work by Alachiotis and Weisz [3]. The architecture developed by Alachiotis and Weisz is implemented, as the authors describe it, following a database-like search model. Given two regions of the MSA, a fraction of the SNPs of one of the two regions are pre-loaded onto the FPGA on-chip memory, acting like queries, while the SNPs comprising the second region will be streamed through the pipeline, much like database objects, in order to be pairwise correlated with the query SNPs. The on-chip memory is organized in a grid of dual-port memory banks, with each port providing the query SNP data to an allele frequency calculator and one of the cores that will calculate the pairwise LD score. The database SNPs are retrieved from the main memory, passed through the allele frequency calculators and streamed through two LD core grids in order to be pairwise correlated with the query SNPs. The LD core consists of a haplotype frequency calculator, while the final allele and haplotype frequencies contribute to the calculation of the LD score through a floating-point implementation of the r^2 LD measure.

The authors have described an extensive design space exploration process, addressing issues concerning the maximum efficiency of the accelerator. Towards that end, double-buffering is implemented for the retrieval of the database SNPs from memory, while the on-chip memory grid size must be carefully selected so that each of the main memory buffers is able to fill before the data loaded on the other have been processed. Furthermore, a sufficient number of LD cores must be instantiated so that all pairwise scores between the query and database SNPs can be processed without delay, while, depending on the available on-chip memory and the dataset size, multiple iterations may be required for all SNPs of the query region to be correlated with the database SNPs. As a result, the authors report increased performance compared to state-of-the-art LD software implementations when the design is mapped to a Virtex7 FPGA.

However, the design goals of that work are different from what this study is attempting to achieve. For one, the former is only concerned with data following the ISM model and is mainly focused on processing simulated data, as it does not account for undefined or missing data along the MSA. On the contrary, our work is focused primarily on the FSM model, supports undefined and missing data in an effort to be better suited for real-world analyses, while potentially maintaining functionality for ISM-based analyses. Furthermore, the relatively small size of on-chip memory, even on the larger FPGAs available, limits the size of the dataset that the pipeline can support, which, in this case, enforces a limit to the number of sequences that are represented in the MSA. It was a conscious design decision of the authors to optimize their architecture for optimal performance rather than the support of larger datasets, while, on the other hand, we are focused on supporting arbitrarily large datasets with a possible handicap on the comparative performance to the previous work. Our point of view is equally valid when considering the current trend of increasing the dataset size, with projections of future datasets consisting of as many as 1,000,000 sequences over the next few years.

3.3 The OmegaPlus LD Kernel as Reference Software

The OmegaPlus [2] algorithm utilizes the LD statistic to determine the association between genomic regions as a precursor to the estimation of the degree that target loci have been subjected to positive selection. The specific algorithm was chosen as the basis for the development of the hardware accelerator for three specific reasons. First, the support of multiple widely-used genomic data input file formats,

```
ms 5 2 -t 3
53303 53650 13864

//
segsites: 6
positions: 0.4478 0.5128 0.5537 0.6123 0.7253 0.7368
000100
101010
010001
101000
101010

//
segsites: 4
positions: 0.0747 0.1319 0.4368 0.5681
0000
1100
0000
0010
0011
```

FIGURE 3.1: Example of an input file under the MS-like format.

```
>D_sec
GTTGTTTAAATACCAATCGATTTGCATTCAAGTTTGAGAATTCTAGGATTTTTCAATTTT
>Dmel_A82_1230
GTTGTTTAAA-------------GCATTTAAT-GTTTCAGCCATACGACTCTTCA-----
>Dmel_A84_1230
GTTGATTAGA-------------GCATTTAAT-CTTTCAGCCATACGACTCTTCA-----
>Dmel_A95_1230
GTTGTTTAAA-------------GCATTTAAT-CTTTCAGCCATACGACTCTTCA-----
```

FIGURE 3.2: Example of an input file under the FASTA format

as well as their transformation into both ISM- and FSM-based formats for more efficient processing, enable the design and implementation of a system that presents a greater variance of choices to the user by addressing specific needs of different analyses. Additionally, the structure of the algorithm presented for the computation of LD allows the easy transition from a software to a hardware implementation, as it is subdivided as two operations: a bit-wise calculation of allele and haplotype frequencies and a floating-point-based calculation of the final LD scores. Finally, through the adoption of a user-defined memory limit for the computations, the generic version of the software enables the calculation of LD scores to be conducted in large chunks of data, if not the entirety of the dataset, instead of a per-SNP-pair basis. This last feature is extremely important when considering a hardware implementation, as it reduces the number of function calls to the hardware co-processor, therefore minimizing the cumulative setup time the co-processor requires before it is able to begin processing the data.

3.3.1 Memory Layout

OmegaPlus is able to parse a wide array of genomic data files, encoded in both binary (ms-like, MaCS-like) and DNA (FASTA, VCF) formats. For the algorithm to be able to process the datasets more efficiently, a compression algorithm is used to transform the raw data into more suitable formats. In the case of binary data, which are already in ISM format, w consecutive sequence values of a SNP are organized in w-bit words and are then stored in contiguous memory space in a per-SNP manner, as illustrated in Figure 3.3. For DNA data there is the option of either performing

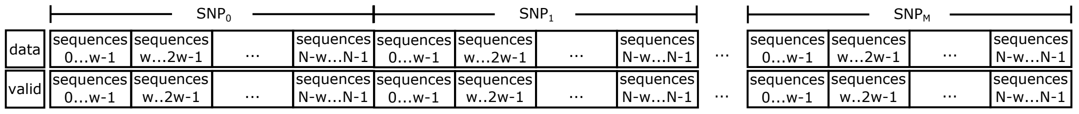


FIGURE 3.3: Structure describing the memory organization of ISM data in OmegaPlus for w-bit words, N number of sequences and M number of SNPs.

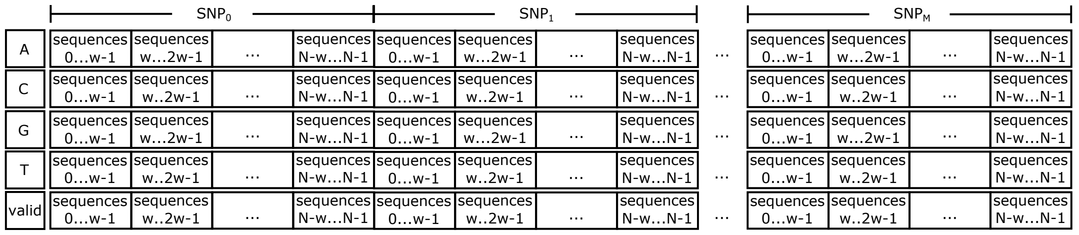


FIGURE 3.4: Structure describing the memory organization of FSM data in OmegaPlus for w-bit words, N number of sequences and M number of SNPs.

a binary deduction into an ISM model, thus adopting the previous memory layout, or translating the data according to the FSM model. In the latter case, the data representing each SNP are organized as 4 w-bit words, with each one representing the presence or absence of one of the four nucleotide bases in the corresponding position in the alignment. This is denoted, in respect to a base, sequence, and SNP, by whether the specific bit of the word is or is not set. The words are organized in contiguous memory space, first by nucleotide base, then by SNP, as illustrated in Figure 3.4. Note in both cases the presence of the extra valid vector, denoting whether portions of the data contain ambiguous characters or gaps.

The computed LD scores are stored in a 2-dimensional correlation matrix with dimensions $M \times M$, where M denotes the number of SNPs. Since the correlation between a pair of SNPs is bidirectional, it only needs to be calculated once. Furthermore, the correlation of a SNP with itself yields no valuable information, therefore it is excluded from the computations. Thus, the correlation matrix exhibits a triangular form, as illustrated in Figure 3.5A.

3.3.2 The LD Computational Core

The LD calculation algorithm can be described as a two-step process, also illustrated in Figure 3.6. The first step entails the enumeration of the mutations present in each and both SNPs that are going to be pairwise correlated. Since the SNP data are organized in w-bit words, multiple iterations of this step are necessary until the population count of all words comprising the SNP pair is accumulated. When the mutation count is calculated, a simple division by the sample size produces the allele and haplotype frequencies required for the calculation of the pairwise correlation score.

The estimation of LD between the SNP pair consists the second step in the algorithmic process. With the allele and haplotype frequencies calculated, OmegaPlus presents a variety of LD measures, including r^2, capable of quantifying the degree of correlation between the SNP pair. At this point, under the ISM model, the computation of LD is complete and the result is stored at the corresponding position in the correlation matrix. Under the FSM model, however, this result merely represents

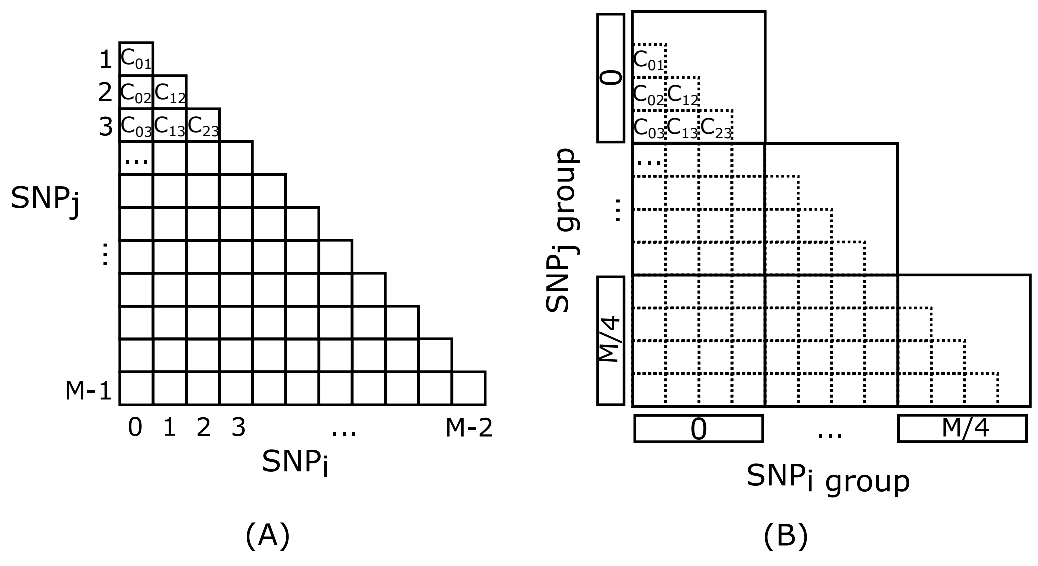


FIGURE 3.5: Structure describing (A) the triangular correlation matrix in OmegaPlus for M number of SNPs and (B) its organization in SNP and compute groups.

the correlation between two nucleotide bases of the SNP pair and not the final LD value. Therefore, multiple iterations are needed, both during the first and second stage of the algorithm, in order to calculate the partial correlations between all 16 possible combinations of nucleotide bases present in the SNP pair. As described by Equation 2.12, the final FSM LD score is calculated when all partial ISM scores have been accumulated and normalized by the number of non-zero states in the SNP pair.

3.3.3 The Generic Algorithm

Besides the simplified sequential version presented in the previous section, several parallel implementations of the OmegaPlus software have been released. The generic version of the software, specifically, exhibits the greater potential for the transformation into a hardware implementation due to the inclusion of one significant feature, namely the organization of SNPs into SNP groups of fixed size. As a result, the correlation matrix can be organized in compute groups (see Figure 3.5B). Furthermore, the user may define an upper limit to the memory footprint of the algorithm. Based on that memory limit, a compute list is constructed containing a number of compute groups, enabling the processing of the dataset to be conducted in large groups of data. For the purpose of detecting positive selection, where biologically distant associated SNPs exhibiting a meaningful impact is improbable, compute groups comprised of such SNPs are excluded from the compute list. For different analyses this may well not be the case, thus appropriate actions must be taken. Additionally, if multiple calculations on different subgenomic regions contain overlapping compute groups, dedicated code organizes the compute list in a manner that eliminates redundant computations.

The specific organization scheme enables the implementation of the algorithm on heterogeneous architectures, such as FPGAs and GPUs, since it removes a significant portion of the communication overhead between the main and co-processor. This is achieved through the ability to group the input data and subsequent computations into large blocks, possibly containing even the whole dataset, before proceeding with their processing. Thereafter, a minimal number of function calls to

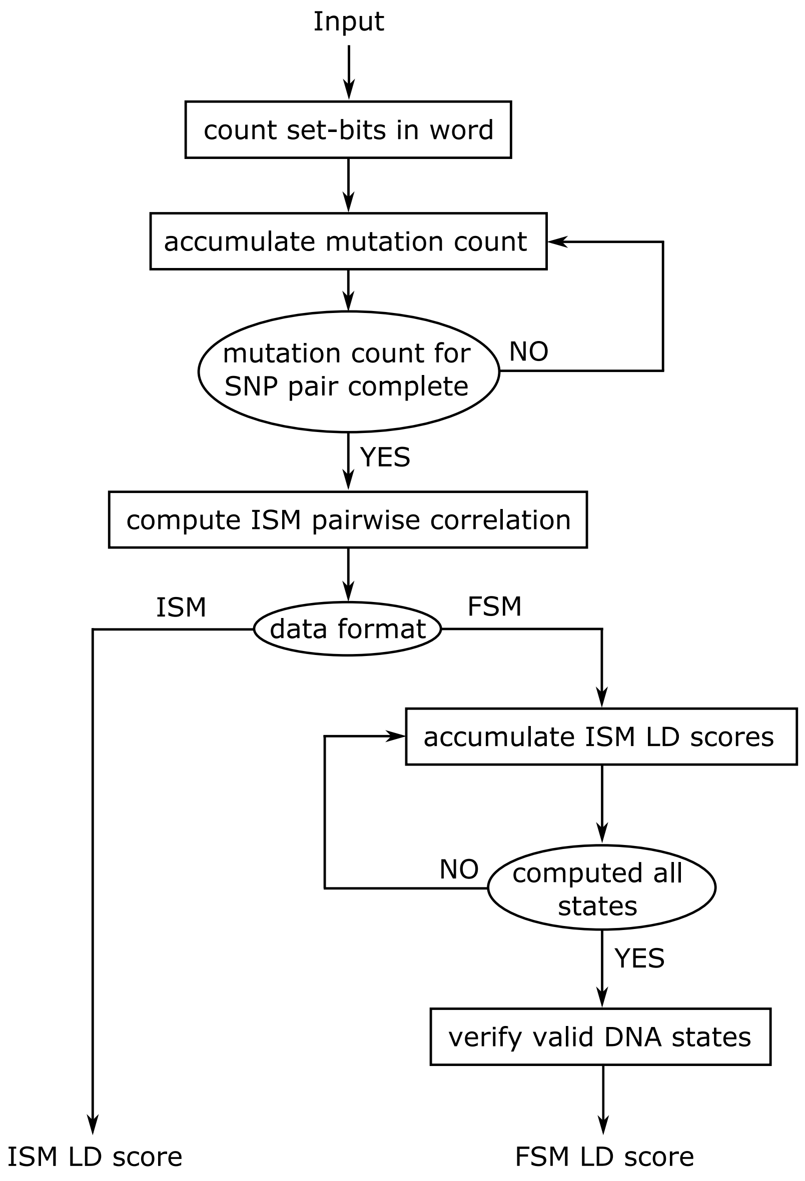


FIGURE 3.6: Dataflow of the LD sequential algorithm, as implemented in OmegaPlus.

the co-processor needs to be facilitated, thus reducing the cumulative downtime required for the communication between the different processing units.

Chapter 4

The LD Accelerator Hardware Architecture

Computing LD is a memory-bound operation, since several memory accesses must be facilitated in order to retrieve all the data required for the calculation of the allele and haplotype frequencies of a target SNP pair. Therefore, maximizing the number of operations per memory access is of vital importance to the overall performance of the accelerator architecture. This chapter focuses on the accelerator's design, starting with the description of a necessary memory layout transformation, in order to efficiently exploit the available memory bandwidth. Thereafter, we proceed to describe the hardware architecture purposed to efficiently process the large bulk of input data and produce the LD scores. Concluding this chapter, we discuss the design space around our accelerator architecture.

4.1 Memory Layout

For the calculation of each LD score multiple memory accesses need to be facilitated in order to retrieve the total amount of the SNP pair data. Thus, the efficient use of available memory is extremely important to the optimal operation of the accelerator. As a reference, we initially consider the OmegaPlus memory layout, described extensively in Section 3.3.1. As illustrated in Figure 4.1A, OmegaPlus stores whole SNPs contiguously in memory. Given a sample size of N sequences and a word width of w bits, each SNP is represented by a total of five memory blocks of size B each, with B defined as:

$$B = \left\lceil \frac{N}{w} \right\rceil, \tag{4.1}$$

with zero padding. The first 4 memory blocks of the SNP space in Figure 4.1A correspond to the SNP data for the 4 nucleotide bases (adenine, thymine, guanine, cytosine), while the fifth block contains the validation bits of the corresponding base locations in the alignment. The validation vector is used by the software to account for missing data, ambiguous characters or various nucleotide base miss-calls.

However, this memory layout is highly inefficient for hardware acceleration purposes. For one, computations cannot proceed until the data for all four nucleotide bases has been retrieved. With the current scheme, the SNP data would have to be loaded by iteratively accessing these four memory blocks with strides far larger than the memory page size, which would lead to prohibitively long overall retrieval times. Furthermore, the validation of said data can be easily deduced in hardware implementations from the data itself, in the form of a short series of logical operations.

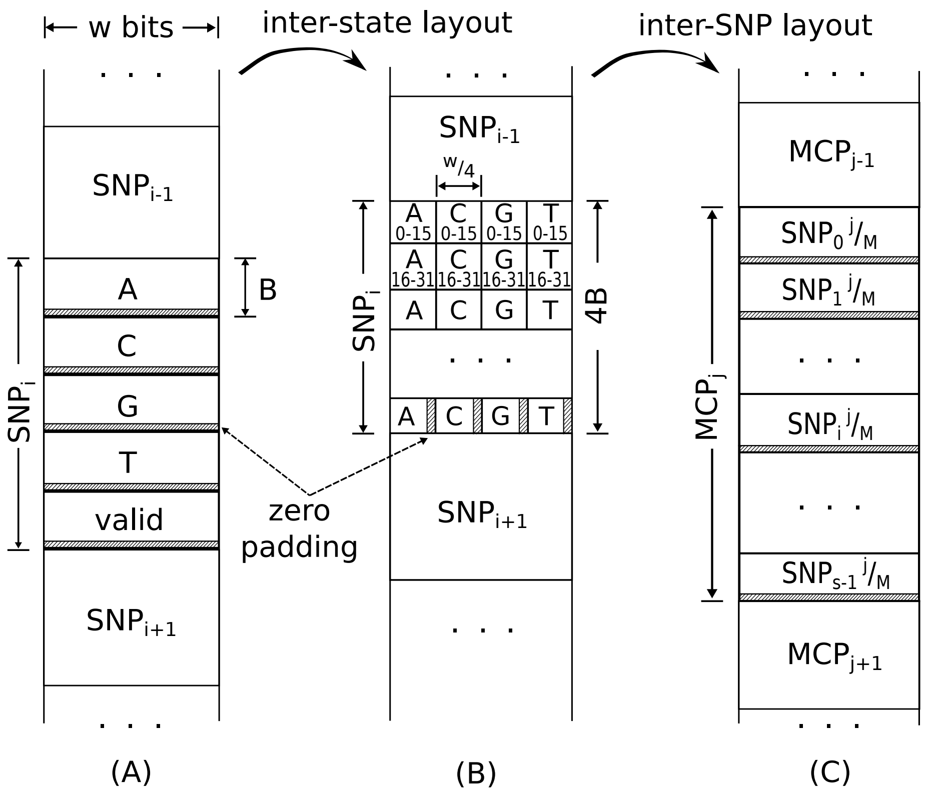


FIGURE 4.1: The two-step transformation approach of the standard OmegaPlus memory layout to (A) an inter-state one, which facilitates processing of multiple states, and (B) an inter-SNP one, which allows shorter data retrieval times.

4.1.1 Inter-state Memory Layout Transformation

The first step of the layout transformation leads to the inter-state SNP representation illustrated in Figure 4.1B. When considering the OmegaPlus layout, all w bits in a word are indicating the presence or absence of a nucleotide base for w sequences. The transformed layout of 4.1B now comprises words that contain the information of all four DNA bases from $w/4$ sequences. If, for instance, we assume a word width $w = 64$ bits and a sample size of $N = 64$ sequences, which leads to a memory block size of $B = 1$. Then, according to the OmegaPlus layout, we would need 5 total memory blocks for all the nucleotide bases and the corresponding validation vectors, consisting of one word each, while each word would contain the SNP or validation information of exactly all 64 sequences. According to our inter-state transformation, this would lead to a memory layout of 1 memory block consisting of 20 words, with each word containing the information of all four nucleotide bases, albeit for 16 sequences.

This transformation enables a memory layout superior to the previous one in the context of hardware architecture because it enables the deployment of custom bit-length vector arithmetic, leading to reduced resource utilization while maximizing the data-per-cycle that reach the input of our design. To elaborate a while further, consider the calculation of allele and haplotype frequencies as described in Sections 2.2.1. In order to compute those frequencies, we need a portion of the SNP information for all four nucleotide bases at each moment in time. Supposing that two w sized words can be retrieved from memory at each clock cycle, then, in the case of the OmegaPlus memory layout, 4 clock cycles would be required in order to acquire the information of all four bases and proceed with the calculation of the frequencies. And therein lie two problems. First, it is obvious that more resources are required in order to process w bit quantities than to process $w/4$ quantities in the same manner and, secondly, those resources will be underutilized, since they would receive data for processing only once every 4 clock cycles. However, by adopting an inter-state layout, we introduce the capability of fully utilizing a smaller amount of resources while processing the same volume of data per clock cycle.

4.1.2 Inter-SNP Memory Layout Transformation

The second and final transformation, depicted in Figure 4.1C, interleaves data from all s SNPs within a genome region of interest based on the number of input ports that the accelerator can deploy for SNP data retrieval. In this paradigm, we assume that the transfer of data between the memory and the accelerator is facilitated with the use of M pairs of memory controllers (MCPs). Each MCP is tasked with retrieving $1/M^{th}$ of the total data that correspond to the SNPs that are to be pairwise correlated. Since the number of SNP pairs that need to be processed is user-defined and varies across different executions, multiple iterations are needed in order to retrieve both the data of each SNP pair under examination at each time interval, as well as the data required for the pairwise correlation of all assigned SNPs.

The purpose of this transformation is to achieve the minimal possible retrieval times when fetching data from different SNPs using multiple MCPs. Consider, for instance, the calculation of two pairwise correlation scores, one between SNPs i and $i+1$ and the second between SNPs $i+1$ and $i+2$. We could speed up the process by parallelizing the population count operation and the allele and haplotype frequency calculations by dividing the number of words required for those computations by the total number M of MCPs and assign only a portion of the SNP data to each MCP. If this scenario was to be implemented using the default OmegaPlus memory layout or the layout produced by our first transformation, the transition between the two pairs of SNPs that need to be correlated would require at least a $4B(M-1)/M$ sized leap in memory space. When considering large datasets, this quantity far exceeds the memory page size, resulting in long retrieval times, since any sort of caching or buffering is underutilized. This is a result of the decision to optimize the pipeline for the minimum possible time required to correlate a pair of SNPs, by assigning the retrieval of different regions of the dataset to different MCs. Therefore, a large part of the dataset is of no concern to the MCP and it even presents an obstacle to the efficient access of the memory. However, by grouping together the partial SNP data that each MCP will be assigned to retrieve, we reduce both the size of the memory stride for each MCP, as well as the total number of strides that are needed for the computation of all the pairwise correlations requested, as a certain number of backwards leaps to previous memory address is, in most cases, unavoidable in the calculation of the LD scores for large combinations of SNPs in the dataset.

Finally, the zero padding introduced in all three memory layouts serves a triple purpose. In the first two layouts, designated in Figure 4.1 as A and B, zero padding represents the excess bits in the final word of each SNP, which contain no valuable data. This becomes the case when the word length is not fully contained in number of sequences and the last word would contain the remainder of the division, namely less than w bits per base in the case of A and less than $w/4$ in the case of B. Since zeros do not contribute to the computation of the allele or haplotype frequencies, the excess bits are handled by simply subtracting their count from the valid sample size.

In the case of the final memory layout transformation of Figure 4.1C, zero padding is introduced for two reasons. First, when the total number of words does not fully contain the number of MCPs, one part of the parallel logic of the design needs to perform one more iteration than the rest in order for all the data of the SNP pair to be processed. In this case, instead of implementing resource-demanding control logic for the synchronized operation of different units, the zero padding is utilized in an effort to force all parallel units to perform the same number of iterations. The second reason is tied to the minimum number of sequences that the accelerator can support, as well as the depth of parallelism of the correlation unit. Depending on the number of correlation arithmetic units, there may arise the need for resource reutilization in order for all 16 combinations of nucleotide bases of the two SNPs to be processed. In this case, due to the fact that the design is fully pipelined, zero padding eliminates the hazard of having more than one scheduled operation for the correlation arithmetic unit, a case that would arise if the dataset is smaller than a certain size. The drawback of this technique is that it establishes a minimum execution time which corresponds to the aforementioned minimum dataset size. In essence, every dataset smaller than the minimum dataset can be processed correctly, however, this cannot be accomplished faster than the minimum execution time defined by the latter. This will become clearer to the reader after becoming familiar with the next subsection of this chapter, describing the hardware accelerator's design.

4.2 LD Accelerator Design

The challenge in designing the LD hardware accelerator lies in the correct detection of the algorithm's bottlenecks and the efficient use of the available memory bandwidth which, along with the optimization of the design to best tackle the performance issues that might arise from said bottlenecks, will maximize the data throughput of the accelerator. In this section, we attempt to describe the generic design of our accelerator system, following a top-down approach which starts by describing the top-level design and, subsequently, elaborating on each individual component in a larger extent.

4.2.1 Top-Level Design

The proposed LD accelerator design, illustrated in Figure 4.2 with the datapath starting from the bottom and moving upwards, operates on a pairwise basis, calculating the LD score for a single pair of SNPs at each time. The pipelined design is separated into four distinct stages, each attempting to efficiently parallelize the individual computations needed for the calculation of the final pairwise LD score. Processing starts with the counting of the DNA states as per the 4 nucleotide bases, achieved by the first two pipeline stages, denoted as BC (Bit Count) Stages 1 and 2. Thereafter, the required allele and haplotype frequencies are computed in the AHF (Allele/Haplotype Frequency) stage, and the required correlation values per pair of DNA states of different SNPs are calculated in the final COR stage. Before presenting the outline of each stage, note that all individual components of this architecture are finely tuned in order to receive new data for processing once every clock cycle, thus eliminating the need for stalls for the purpose of avoiding possible hazards.

The first stage, or $BC\ stage\ 1$, as denoted in the top level block diagram, implements the population count operation on the 64-bit quantities that reach the architecture's input ports at each clock cycle. Multiple population counters operating in

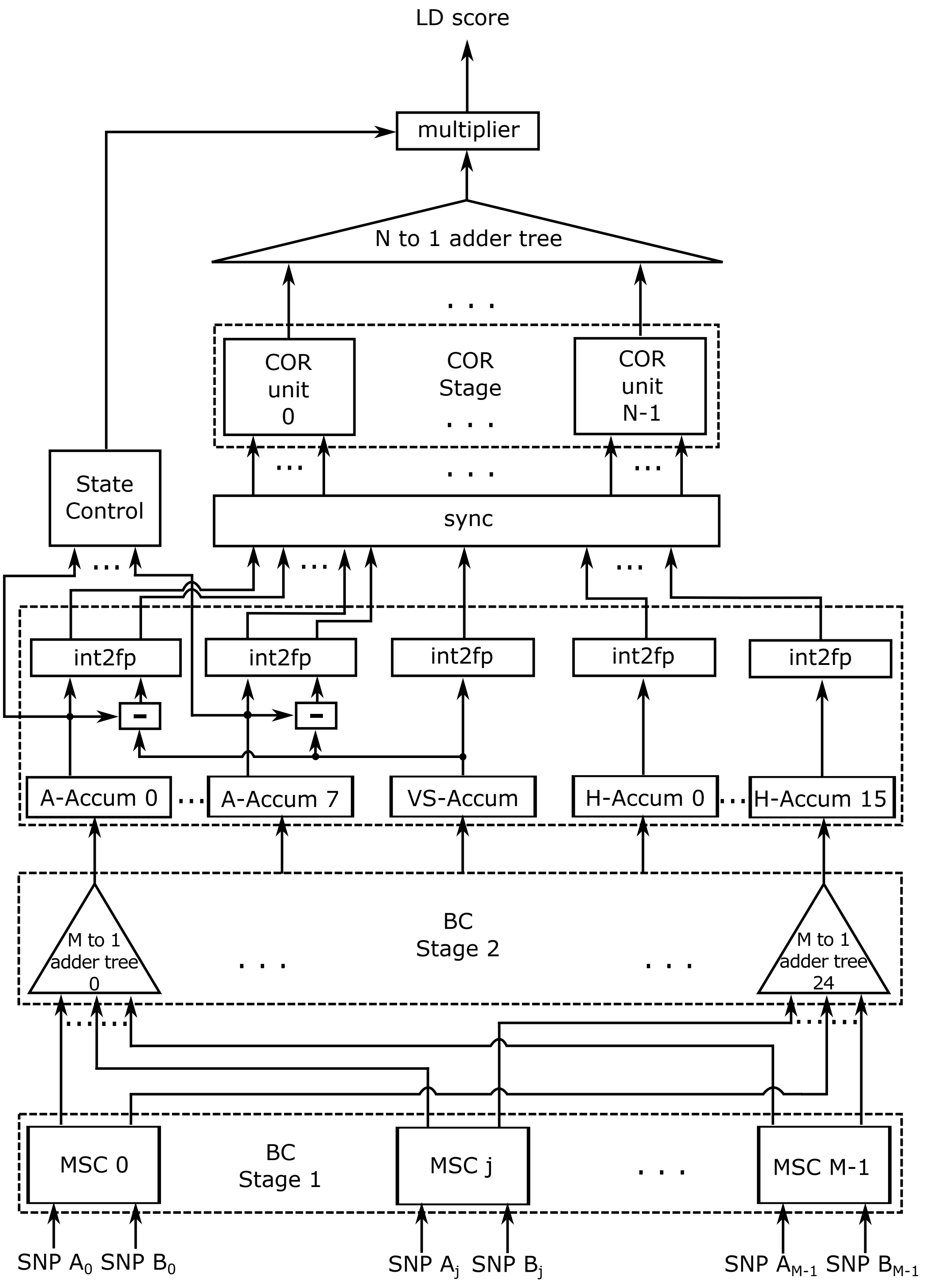


FIGURE 4.2: Top-level design of the accelerator architecture. Processing proceeds from the bottom to the top, through four discrete pipeline stages, i.e., BC Stages 1 and 2 for the partial accumulation of population count results, AHF for a final accumulation and calculation of allele and haplotype frequencies, and a final COR stage for the floating-point correlation calculations.

parallel calculate the number of set bits in the incoming binary vectors. SNP data are provided via a number of $2M$ input ports, organized in M pairs, to facilitate pairwise calculations. As already mentioned, each pair of ports (referred to as MCPs in Section 4.1) retrieves a fraction of the entire SNP. For the purpose of processing the incoming data, a total number of M Mutation/State Counters are deployed, each producing a partial count of the total number of mutations/states present in the SNP pair under examination. When considering the finite sites model, a total of 25 binary vectors are required for the calculation of the LD scores, namely 4 vectors referencing the DNA states of SNP A, 4 referencing the DNA states of SNP B, 16 for all the possible combinations of the per-state vectors for the pair of SNPs A and B, and 1 internally computed validation vector. Each MSC unit calculates a partial value of all 25 vectors.

The second bit-counting stage, labeled $BC\ stage\ 2$, simply accumulates the M individually produced values from all M Mutation/State Counters into a single vector for each of the 25 distinct allele, haplotype, and valid counts. This is accomplished by implementing a total of 25 M-to-1 adder trees, with each tree accumulating all M partial scores of one of the aforementioned vectors produced by all MSCs. The 25 partial sums that are passed to the next pipeline stage reflect all mutations or states that have been previously counted by the M MSCs in BC Stage 1. This allows proceeding with the remaining computational steps in the following pipeline stages independently of the available number of input ports, thus facilitating a potential future migration to a different device/platform. In addition to alleviating the migration overhead, organizing the enumeration step of alleles and haplotypes into a platform-specific (BC Stage 1) and a dataset-specific stage (BC Stage 2) allows to easily support other dataset types as well, such as RNA secondary structure (6, 7 states) or amino acids (20 states). This requires additional logic and population count instances in BC Stage 1, as well as additional M-to-1 adder trees in BC Stage 2, while the rest of the design remains largely intact.

The partial allele and haplotype counts produced by BC stage 2 are in turn accumulated in the allele/haplotype frequency stage ($AHF\ stage$). As illustrated in the block diagram of the architecture's top level, the AHF stage consists of 3 discrete operations. First, the partial per-state sums along with the validation bits are accumulated by utilizing a total of 25 W-bit-wide accumulators. The denoted A-$Accum$ units accumulate the allele counts of each individual SNP of the pair, while H-$Accum$ units accumulate the per-state haplotype counts between the two SNPs. In order to yield comparable LD scores along different genomic regions within the same chromosome, the valid SNP size per SNP pair is calculated on a pairwise basis as well, by the VS-$Accum$ (Valid Size) accumulator, since the number of valid pairs of bits in a SNP pair may vary, depending on the amount of missing data and alignment gaps in both SNPs. This additional accumulation stage allows to correctly calculate the number of mutations or DNA states in large-scale datasets that comprise thousands to millions of genomes, given that sufficiently wide accumulators are deployed. However, for the calculation of the LD scores, the count of non-mutated alleles is needed along with the count of the mutated ones. Following Equations 2.10 and 2.11, the non-mutated allele count is computed by subtracting the total count of the mutated ones, accumulated by the A-Accum units, from the total number of valid sequences for the SNP pair, accumulated by the VS-Accum unit. Finally, all 33 total counts of mutated and non-mutated alleles, mutated haplotypes, and valid samples are converted from an integer to a floating-point format. This conversion is essential at this point of the architecture as it enables the reduction of utilized resources, while, simultaneously, ensuring a significantly higher degree of precision in the final results.

The final stage of the design entails the computation of the pairwise correlation between the two SNPs, thus denoted as $COR\ stage$. This is achieved via the instantiation of N floating-point pipelines ($COR\ units$) operating in parallel, each implementing the same measure for the calculation of LD scores, which in our case is r^2. The number N of COR units must be carefully selected, as it also indicates the number of iterations that each unit must perform in order to process all 16 pairwise combinations. In the case of the FSM, this refers to the total combinations between the nucleotide bases between the SNP pair, according to Equation 2.12. For this reason and for the purpose of the calculation of LD in DNA datasets following the FSM model, which is the scope of this work, N must be between 1 and 16, while it may vary if this work is to be repurposed to accommodate different datasets, i.e., RNA, etc. If the number of COR units is smaller than the total number of pairwise calculations required, dedicated synchronization/scheduling logic is inserted between the AHF and COR stages, ensuring access to the available COR units in a round-robin fashion. Instantiating a smaller number than the expected 16 COR units does not affect performance due to the expected large sample size, which shifts the calculation bottleneck to the enumeration of alleles and haplotypes, i.e., first three stages of the accelerator pipeline. However, this implementation method would facilitate the risk of possible hazards if new data from the AHF stage were to arrive before all previous LD calculations had been scheduled. This is the reason that the zero padding and the minimum sequence limit has been established depending on the number of COR and MSC units, as described in Section 4.1.2. This number can be calculated as:

$$S_{min} = W_S(NM - 1) + 1, \tag{4.2}$$

where W_S describes the number of sequences represented in each word, N is the number of COR units and M the number of MSC units. As an example, if a total number of $N = 4$ COR and $M = 7$ MSC units was to be implemented in a design based on words containing $W_S = 16$ sequences, this would introduce a minimum requirement of 433 sequences.

At this point, the calculation of the partial scores for each pair of nucleotide bases is completed. When considering the FSM model, however, additional operations are required, as described by Equation 2.12. The 16 pairwise correlation scores produced by the COR stage must be added together, which is accomplished by an N-to-1 adder tree. Note here that, depending on the number of COR units, the adder tree may require an extra accumulator added to its structure in the case of less COR units having been instantiated than the number of necessary pairwise calculations. Furthermore, the State Control unit operating in parallel with the COR stage calculates the numbers of valid states per SNP in the pair under examination, as required by Equation 2.12. A final multiplier completes the calculation of the LD score for the SNP pair by multiplying the pairwise correlation of all SNP states with the number of valid states per SNP and the final result can be scheduled to be stored in memory.

The architecture described so far is tailored and optimized for the efficient processing of FSM datasets. However, a handful of minor changes could establish the support of ISM data, as an added functionality. Since the estimation of LD according to the FSM model can be seen as a series of ISM calculations, each of the partial FSM scores produced by the COR units is essentially a pairwise LD score under the ISM. To facilitate such a functionality, first, the ISM dataset should be organized in a form supported by the accelerator. This could be achieved by reorganizing the dataset, as per the memory layout transform of Figure 4.1, with each word containing the partial data of 4 SNPs under the ISM, instead of the 4 nucleotide bases required by the FSM. Additionally, since the accelerator provides only one output port, serializer logic would have to be implemented in order of all ISM LD scores produced simultaneously by the COR units to stored in memory through the single output port, while a multiplexer would select the correct output, depending on the data

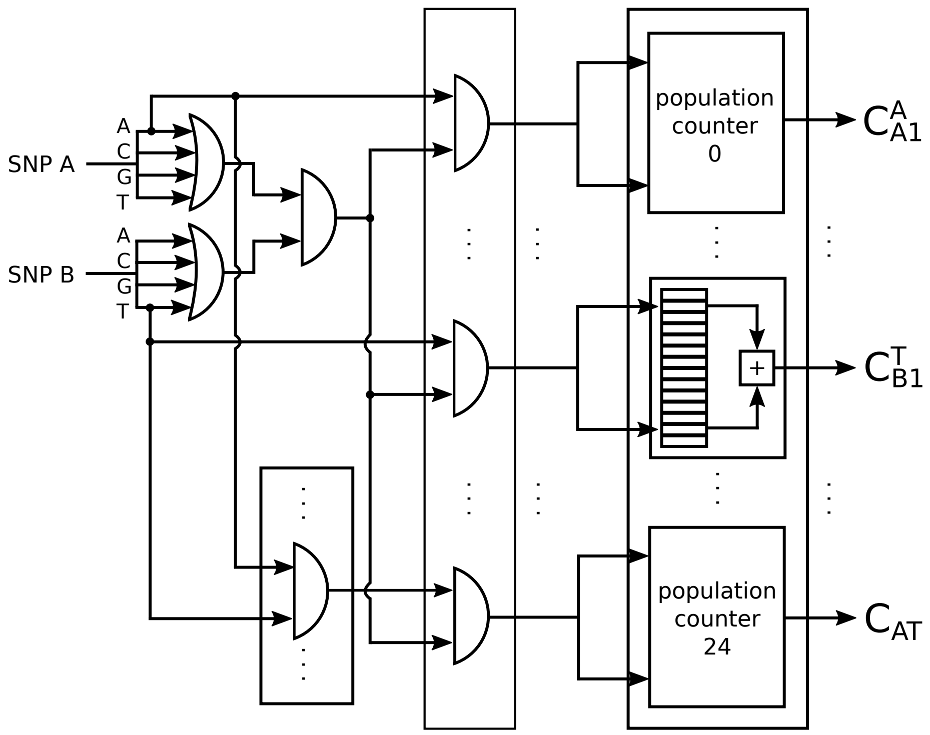


FIGURE 4.3: Block diagram of the MSC unit. As the SNP length (the number of genomes) increases, overall acceleration performance relies on the number of MSC units operating in parallel to boost population count capacity of the FPGA. Prior to the array of 16-bit population count blocks, dedicated logic calculates on the fly all the required pairwise combinations of the input SNPs.

model. Finally, the validation of the data, which transpires in BC stage 1, cannot be facilitated without for the ISM without an additional validation vector that would be retrieved from memory, due to the binary nature of the ISM data. Therefore the validation of the ISM data cannot be efficiently supported by the current pipeline without possibly hamstringing the performance of the accelerator overall. These changes would enable the support of both FSM and ISM datasets, however, an alternate pipeline dedicated to ISM data processing would definitely present greater performance capabilities and potential application for real-world analyses.

4.2.2 Mutation/State Counter

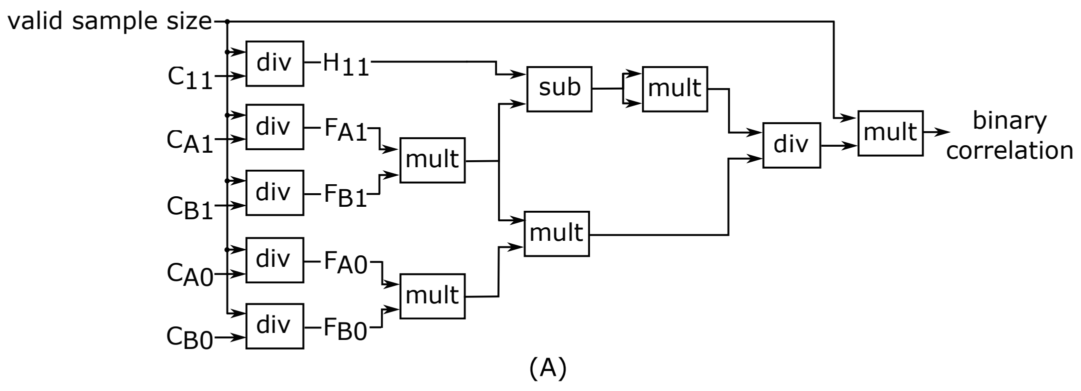
The Mutation/State Counter computes the partial counts of the mutated DNA states, along with the validation bits corresponding to them. As illustrated in Figure 4.3, the MSC unit receives two inputs in the form of the W-bit data values per SNP, each consisting of $W/4$ sequence data per nucleotide base under the FSM model. The valid sample vector is calculated with two 4-input OR gates with their outputs inserted in an AND gate, while a series of parallel AND gates compute all the 16 pairs of haplotype values between the SNPs. As a third stage, all allele and haplotype values are validated with the valid sample vector through the use of AND gates. Then, all allele, haplotype, and valid sample vectors are able to proceed through a grid of 25 population counters for the partial mutation/state and valid counts to be computed. Each population counter consists of a dual-port $k \times n$ block ROM, where k refers to the number of bits required to enumerate the set bits in a nucleotide base residing in a quarter of a word with length $W/2$, while n refers to all $2^{W/8}$ different possible combinations of aligned bits in a $W/8$-bit-wide vector. The $W/4$ output of each gate of the previous stage are split in half in order to take advantage of the full functionality of a dual port ROM, thus halving its depth and conserving resources. The population count ROM functions as a look-up table, with each memory cell containing the count of set bits corresponding to its address. The two outputs of each population count ROM are added together to produce the final partial bitcount of the BC stage 1 described in the previous subsection.

The MSC unit is the most important unit of the architecture. The computational bottleneck of the design lies exactly in the population count operation. With increasing dataset size, the iterations needed in order to count the set bits of a SNP pair require multiple iterations of the population count operation and, consequently, multiple iterations through the MSC unit. Therefore, the overall performance of the architecture is dependent and proportional to the available bandwidth and the number of parallel MSC units that complement it. Thus, it is of vital importance that the number of implemented MSC units working in parallel be carefully selected so that the available bandwidth is maximally utilized.

4.2.3 Correlation Unit

The correlation (COR) unit implements Equations 2.10 or 2.11, as illustrated in Figure 4.4A and B, respectively, as a measure of binary LD scores. A single COR unit calculates the binary r^2 statistic for a pair of nucleotide bases of two SNPs. The unit's structure consists of a pipeline of floating-point arithmetic operators, according to the aforementioned equations. The floating-point units are organized in a fashion that aims to maximally exploit the parallelization of operations wherever it is possible. Furthermore, all floating-point operators are fully parallelized, allowing the COR unit to receive new input data per clock cycle, thus achieving the minimal delay possible. As clearly visible in Figure 4.4, there are presented two distinct implementations of the COR unit, denoted A and B, each according to one of the two equations describing r^2. Correlation unit A is based on Equation 2.10 and calculates the allele and haplotype frequencies by dividing the allele and haplotype counts by the valid sample size prior to any other operation. This method, while relatively computationally heavy, achieves higher overall accuracy. The second correlation unit implementation bypasses the frequency conversion and computes the LD score based on the allele and haplotype counts directly, as described by Equation 2.11. Correlation unit B requires significantly fewer resources than A, while it displays lower latency at the cost of reduced accuracy.

Both pipelines normalize the correlation coefficient by multiplying it with the valid sample size. Therefore the value of r^2 does not reside within the (0,1) set, but rather may take any value, depending on the size of the dataset. As a reference for comparison, implementation A is accurate on the fourth decimal digit, while implementation B is only accurate on the second. Resource utilization and latency comparisons for each floating-point operator, as well as between the two implementations as a whole, are presented in Tables 4.1 and 4.2, respectively. From a coding standpoint the two implementations are interchangeable, provided that the correct values are assigned to the delay variables in the top-level package, in order to maintain a correctly synchronized design.



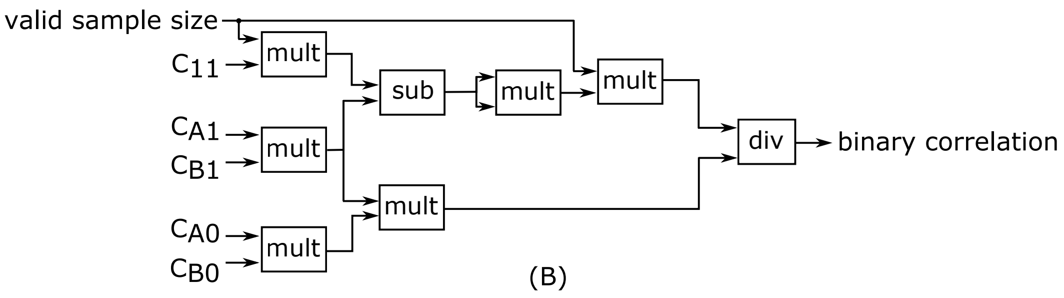


FIGURE 4.4: Block diagrams of the two versions of the COR unit. Diagram A presents higher result accuracy while B requires fewer resources and achieves lower execution times.

TABLE 4.1: Resource utilization and latency of 32-bit floating-point operators

		Slice Logic Utilization			
Operator	Slices	Registers	LUTs	DSPs	Latency
subtractor	91	335	227	2	11
multiplier	36	115	109	3	6
divider	309	1359	1043	0	28

4.2.4 State Control

The State Control unit implements the first part of Equation 2.12. The equation denotes that, under the FSM model, the number of valid nucleotide states per SNP must be taken into account along with the count of mutated alleles and haplotypes. As illustrated in Figure 4.5, the unit receives an input consisting of the allele base counts for each nucleotide base of the two SNPs, as well as the valid sample size, all calculated in the AHF stage. Since the maximum number of non-zero states is '4' for each SNP, the Valid State Check unit simply examines whether any of the nucleotide base counts amounts to zero and, therefore, reducing the appropriate valid state count V of the corresponding SNP accordingly.

Then, the numerator and denominator of the equation must be calculated. Since V can only hold a maximum value of 4, the multiplication of V_A with V_B and $V_A - 1$ with $V_B - 1$ does not justify the use of full multipliers. Therefore, for the purpose of utilizing a minimum number of resources, the multiplier units are implemented as simple look-up tables consisting of 16 states in the case of the denominator and

TABLE 4.2: Resource utilization and latency of COR units A and B

COR unit	Slices	Slice Logic Utilization Registers	LUTs	DSPs	Latency
A	2125	9064	7030	17	85
B	616	2384	1924	20	57

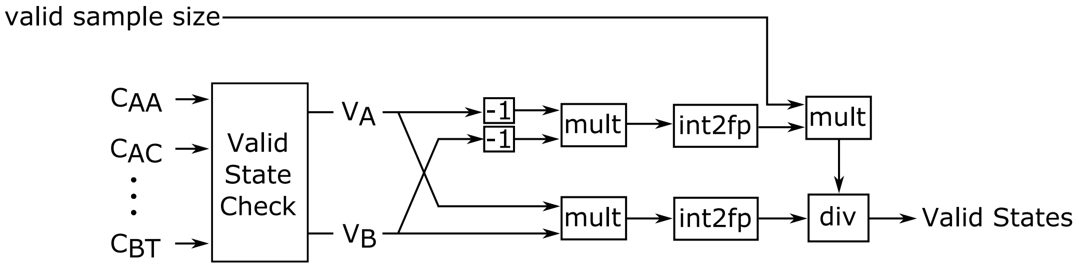


FIGURE 4.5: Block diagram of the State Control unit. The unit computes the number of non-zero nucleotide states of each allele under the Finite Sites Model.

9 states in the case of the numerator, where the valid state counts are reduced by a single numeric unit. At this point a conversion from integer to floating-point arithmetic is necessary as the valid sample size is already formatted for floating-point arithmetic and, mainly, due to the division that concludes the calculation of this specific part of the LD equation under the FSM. From a computational standpoint, since the calculation of valid states shares no dependency with the rest of the COR stage, it can be implemented in parallel to the latter, thus reducing the overall latency of the architecture.

4.3 Memory-Architecture Interconnect

The communication between the memory and the accelerator architecture comprises the most significant part of this work. Due to the nature of the problem under examination, maximizing the efficient use of available memory bandwidth is of utmost importance, since the enumeration of set bits in binary vectors in the scale of datasets that the current work is aiming for is, as mentioned before, a memory-bound operation. For this exact reason, the design of the memory interface must facilitate a high speed of data retrieval with the smallest possible downtime. Towards that end, and based on the memory layout described in Section 4.1, the implementation that handles the memory interface can be described as two stages.

The first stage consists of the logic that communicates with the Memory Controllers. Each MC is assigned an independent finite states machine (MCFSM) that dictates the memory operations that must be facilitated. These MCFSMs can be enumerated as M[1] MCFSMs for the retrieval of the data of SNP A, M MCFSMs for the retrieval of the data of SNP B and one MCFSM for storing the LD scores back to the memory, henceforth referred to as $MCFSM_A$, $MCFSM_B$ and $MCFSM_W$ respectively (see Figure 4.6). Considering the contiguous structure of the memory layout, the best way to access the data of all pairwise SNP combinations is by calculating the LD scores of the 2-dimensional correlation matrix in diagonals, i.e., starting with

[1]M refers to the number of MCPs as described in Section 4.1.

SNP pair [0,1], moving to pair [1,2] up until pair [N-1,N], then regressing back to pair [0,2] and so on. This way of accessing the memory ensures that each MC will retrieve the data of large contiguous memory spaces, thus minimizing the number of strides between distant memory blocks to the number of diagonals of the correlation matrix. Since the input cannot be buffered when accessing the memory backwards and the support of arbitrarily large datasets makes the calculation of all pairwise LD combinations without accessing previous memory addresses impossible, we believe that this is the most efficient memory access paradigm that can be accomplished for the scope of this work. Furthermore, by implementing the MCFSMs independently we ensure that possible stalls of one or more MCs will not affect the rest MCs, thus allowing each MC to follow its own pace and consequently allowing for an overall more optimized memory access. If a more centralized MCFSM controlling all MCs at the same time was to be implemented, that would mean that either stalls would have to be global, thus halting the operation of MCs that would otherwise continue to operate normally, or the MCFSM itself would grow into a prohibitively large amount of states in an effort to account for the optimal operation of each individual MC.

This methodology is implemented through the differentiation of the two read MCFSMs, $MCFSM_A$ and $MCFSM_B$, of each MCP, as illustrated in Figure 4.6A and 4.6B. Both share a similar structure consisting of 3 states, namely the IDLE, READ, and ROLLBACK states. The IDLE state initializes the starting address and waits for the execution to start. Consequently, the READ state dictates when and from which address should new data be retrieved, while simultaneously checking if data retrieval should be stalled, as is the case if the input buffers are about to be overflowed. When the input data of a whole diagonal of SNP pairs has been retrieved, the ROLLBACK state checks if all diagonals have been processed. If the above statement is true, the MCFSM returns to its IDLE state and remains there until the next dataset is routed for processing. If the statement is false, the two MCFSMs differentiate from each other. If we consider accessing a 2-dimensional triangular matrix by diagonals, then each diagonal starts at the cell denoted as $(0, j + 1)$, where j refers to the first cell of the previous diagonal, starting from 0, while the diagonal finishes at the cell denoted as $(i - 1, N - 1)$, where i refers to the last cell of the previous diagonal starting from $N - 1$. Therefore, in the case of the $MCFSM_A$, the first address remains the same for every diagonal, while the last address is always diminished by the memory size of a single SNP for a MCP. Similarly, $MCFSM_B$ always stops at the same last address, consisting the final word of the last SNP of the dataset, while the starting address of each consequent diagonal is increased by the memory size of a single SNP for a MCP. Finally, the $MCFSM_W$, illustrated in Figure 4.6C, consists of only a couple of states. The IDLE state functions in the exact same way as in the read MCFSMs, while the STORE state simply waits until new results are ready to be stored in memory and the MCFSM enables the controller to do so. Due to the fact that the input data in large datasets vastly overshadow the size of the output data, one MC dedicating to handling the results is enough.

As mentioned above, each MCFSM is implemented independently of the others. While this feature ensures an optimal access to memory, it presents a problem in the form of an unpredictability of the arrival time of data with a computational dependence. The second stage of the memory interface addresses precisely this issue by introducing input buffers. Each of the $2M$ memory ports, instead of feeding the input directly into the accelerator, stores the data in a 128-word-deep block RAM configured as a FIFO. The accelerator is then allowed to access the data only when all $2M$ buffers are not empty, thus ensuring that all dependent input data are reaching the accelerator's input ports simultaneously. By synchronizing the data this early

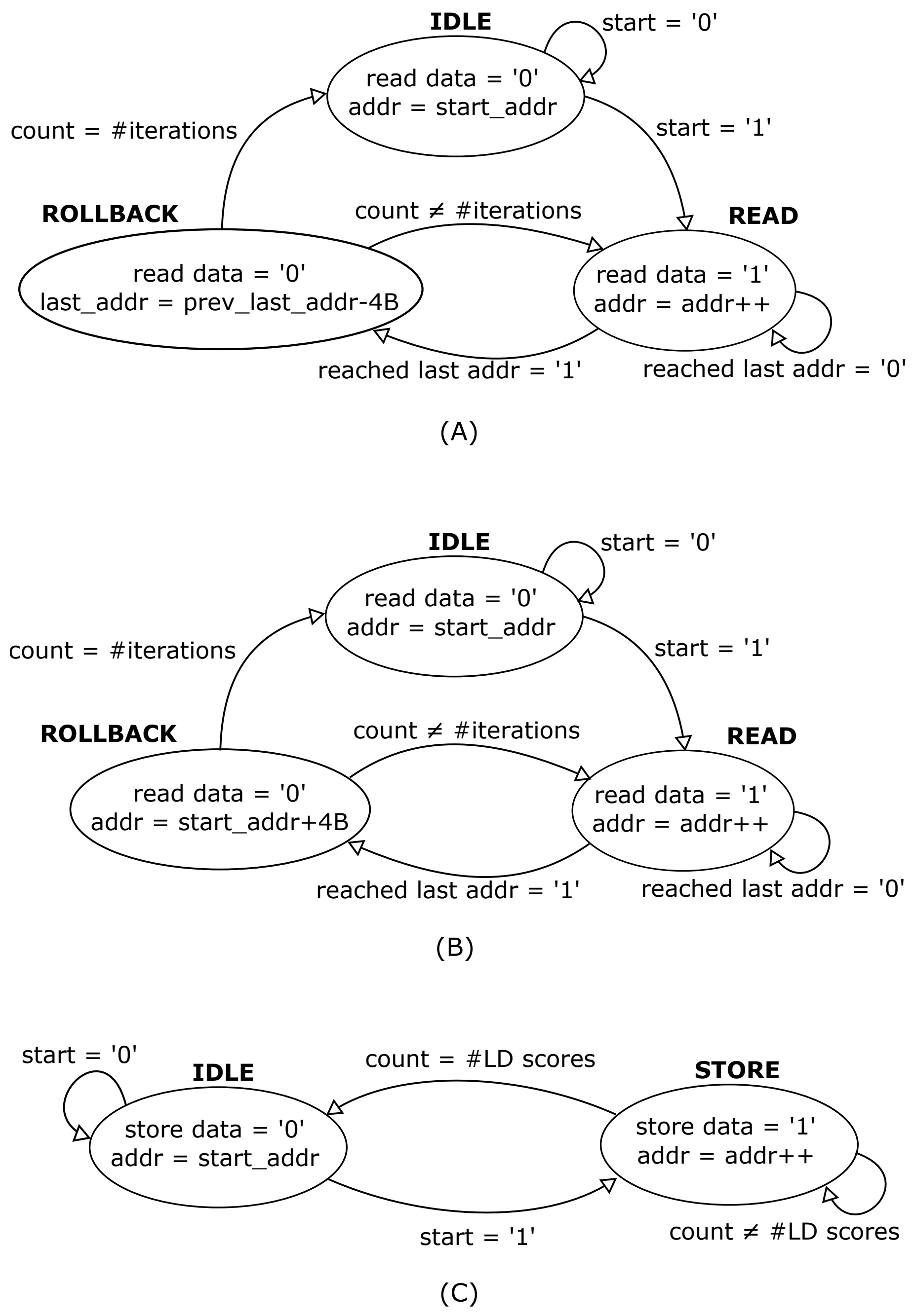


FIGURE 4.6: State diagrams of the MCFSMs for (A) reading the data of SNP A (B) reading the data of SNP B (C) storing the LD scores back to memory.

in the architecture the benefit is twofold. Firstly, the need of complicated synchronization and stall logic later on in the design is eliminated and, secondly, the core of the accelerator becomes detachable and more easily modified, especially when considering future possible applications. In a similar fashion to the input buffers, the accelerator stores the results in an output FIFO before the store MC is able to access them and route their storage to the memory.

4.4 Design Space Analysis

LD is a memory-bound computation, relying on the rapid retrieval of data from memory and their efficient processing without delay. The architecture has been intentionally described up to this point as a generic design, the purpose being the introduction of the degree of modularity that is allowed by the accelerator. At this point it should have become clear that the level of parallelism of each individual unit at each stage of the architecture must be carefully selected depending on the size of the datasets that a user intends to process, as well as the hardware limitations present in the form of the memory bandwidth and the size of the target device. Furthermore, the event-driven decentralized control that every stage of the pipeline operates under allows the replacement of each unit with modules providing different functionalities, i.e., different LD measures, etc., to become a relatively easy task.

Under these assumptions, depending on the available bandwidth and, therefore, the pairs of memory ports, an equal number of MSCs is required in BC Stage 1 in order to process data at the speed of arrival. As the number of sequences increases, the allele and haplotype frequency computations quickly become the limiting performance factor, even when considering moderately sized datasets. Therefore, suboptimal processing/delays during the allele/state enumeration stages can have a profound effect on the success of the acceleration system.

In a similar manner, the number of pairs of parallel input ports dictates the size of the M-to-1 adder trees in BC Stage 2. The number of adder trees, however, as well as the number of accumulators in the AHF stage is driven by the data type. As the present work focuses on DNA data, we can assume a data type that consists of $S = 4$ states. In this case, an equal number of S adder trees and S accumulators is required for each of the two input SNPs, as well as S^2 trees and accumulators for all possible combinations of states for the two SNPs. The additional adder tree and accumulator pertaining to the validation bits is only required if the user opts to account for missing data and/or base miss-calls in the dataset. If this is the case, this solution, which is specific to the proposed architecture, trades resources for a smaller memory footprint and shorter data retrieval times, as the alternative would require the retrieval of the validation bit vector from memory. Considering the accumulator, its width should be sufficiently large to accommodate the maximum possible number of mutated alleles/states for a sample size N. Since non-polymorphic sites are discarded prior to an analysis and, consequently, alignment sites with 0 or N set bits do not exist in the data to be processed, the accumulator width must present the capability to count up to $N - 1$.

In the final COR stage, the number of COR units operating in parallel may be chosen more liberally than that of the units of the previous stages, because it does not directly affect performance. However, instantiating more COR units than the S^2 pairwise allele/state computations would not improve performance, as the excess units would remain idle. If exactly S^2 COR units are deployed, then the COR stage is capable of fully absorbing the data produced by the previous stage, while instantiating less usually does not affect performance due to the expectedly large sample sizes, which shift the computational bottleneck to the enumeration of alleles and haplotypes. If the number of samples is small enough to permit the previous AHF stage to provide input to the COR stage before the previous inputs have begun processing, only then does the COR stage become the bottleneck. However, this is not the average use-case for which the current architecture is designed. Nevertheless, such small sample sizes are accommodated correctly, albeit suboptimally, through the deployment of zero padding.

Chapter 5

Full System Implementation

To evaluate the correct operation and performance of the proposed architecture we paired its implementation with a high-end hybrid computing system with increased performance capabilities. In this chapter, we attempt to present an overview of the Convey HC-2ex hybrid computing platform, on which our accelerator is implemented, and subsequently, proceed to describe the instantiation of the design based on the specific features of the target platform.

5.1 The Convey HC-2ex Platform

The Convey Hybrid Core platform is a heterogeneous computing system, efficiently pairing traditional general purpose CPUs with reconfigurable hardware co-processor units, in order to increase application performance beyond what is typically possible in a standard x86 system. This is achieved by implementing a custom hardware architecture to be routed on the FPGA co-processor, along with the corresponding set of instructions, the so-called $personality$, and combined with the necessary software running on the CPU addressing all requirements of the custom application. While some basic personalities are provided by Convey itself, a framework to facilitate the implementation of personalities that require specialized functionalities depending on the needs of the application is also provided. The Personality Development Kit (PDK) [12] provides all the hardware, software and simulation interfaces necessary to implement a custom personality on the Convey family of products.

The system itself, as illustrated in Figure 5.1, consists of two interconnected processor units mounted on the two sockets of a commodity motherboard. The first socket accommodates a proprietary Intel x86 host processor along with the standard Intel I/O chipset, while in the second resides a reconfigurable co-processor based on FPGA technology. The co-processor includes its own high-bandwidth memory subsystem that is incorporated into the Intel global memory space, creating the Hybrid-Core Globally Shared Memory (HCGSM). The HC-2ex system, specifically, utilizes an Intel Xeon E5-2642 6-core as its host processor, four Xilinx Virtex LX760 FPGAs on its co-processor unit and a total of 64 GB DDR2 shared memory capable of providing up to 80 GB/s of memory bandwidth when all four co-processor FPGAs are deployed.

5.1.1 Co-processor Architecture

The co-processor has three major sets of components, referred to as the Application Engine Hub (AEH), the Memory Controllers (MCs), and the Application Engines (AEs), depicted in Figure 5.2. The functionality of the AEH is that of a central hub for the co-processor. It acts as an interface to the host processor and the I/O chipset and is responsible for the retrieval and decoding of instructions, as well as

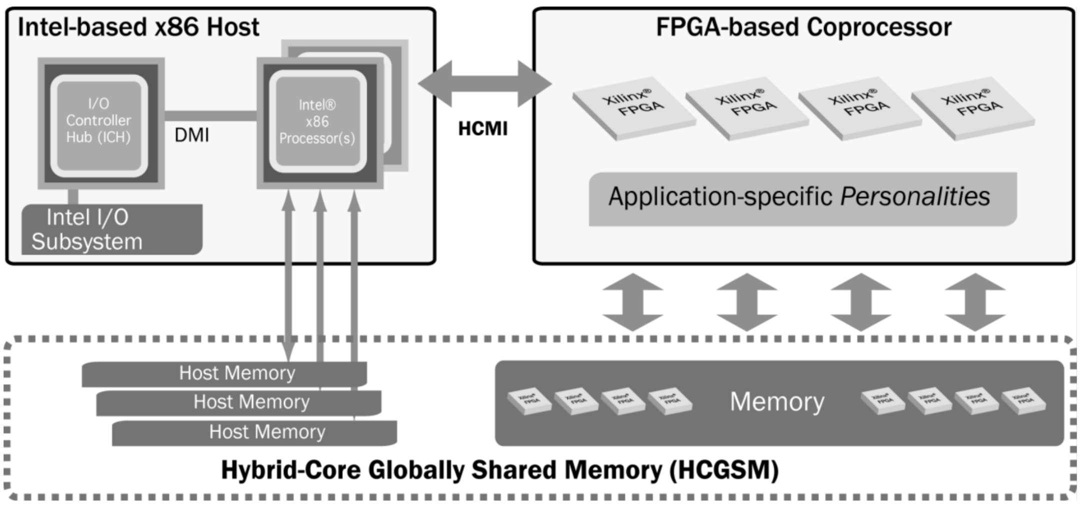


FIGURE 5.1: Diagram of the Convey Hybrid Core System. The diagram (borrowed from the Convey Reference Manual [13]) depicts the host processor, the FPGA co-processor and the shared memory between the two.

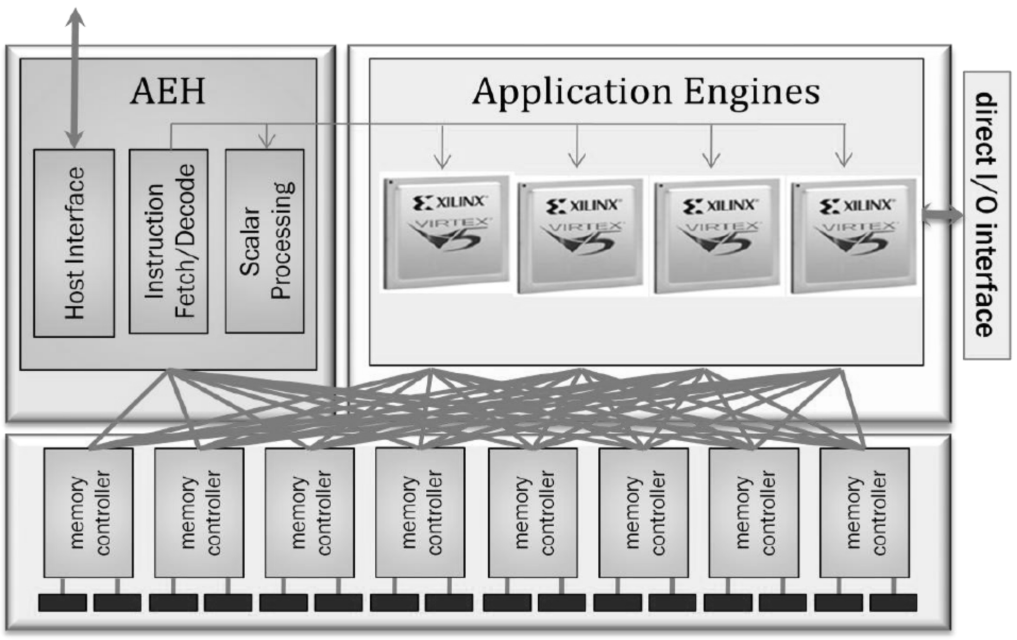


FIGURE 5.2: Diagram of the co-processor. The diagram (source: [13]) depicts the three subsystems of the FPGA co-processor, the Application Engine Hub, Memory Controllers and Application Engines.

the execution of the scalar ones. Furthermore, it processes coherence and data requests from the host processor while, simultaneously, routing requests for addresses in the co-processor memory to the MCs. The 8 MCs of the co-processor support a total of 16 DDR2 memory channels by translating virtual to physical memory addresses on behalf of the AEs. Together with the AEH, the MCs implement features of general purpose, regardless of the personality, in an effort to ensure that important features, such as memory protection or the communication between the host and co-processor, are always available.

On the other hand, the AEs implement the extended instructions and the custom architecture that deliver performance for a personality. The connection between the AEs and the AEH is established through a command bus that transfers opcodes and scalar operands, while the connection between the AEs and the MCs is implemented via a network of point-to-point links. The AE instruction is passed to all four AEs which in turn process the instruction according to the implemented personality.

5.1.2 Personalities

A personality defines the set of instructions supported by the co-processor and their behavior. It includes the loadable bit files that implement a co-processor instruction set, a list of the instructions supported by that personality, and an ID used by the application to load the correct image at runtime. A personality implements two types of instructions. The scalar instructions are common among all personalities and ensure that all personalities share the same basic functionalities, while the extended instructions provide customization for different applications and workloads.

All personalities share a set of common elements. The host interface that dispatches the co-processor instructions, which also initiate and control the execution of the personality on the co-processor, and the return status are the same for all personalities. Furthermore, the address translation from physical to virtual addresses is compatible with the Intel 64 specification in order for instructions to coherently share the memory with the host processor and to ensure protection for process address spaces, since the host processor and the I/O system are able to access co-processor memory and vice versa.

5.1.3 Memory Controller Interface

The Memory Controller interface provides the AEs with direct access to the co-processor memory. All 4 AEs are connected to all 8 MCs, as illustrated in Figure 5.3, each via a 300 MHz DDR interface designed by the vendor, while each of the 8 interfaces is directly connected to a single MC, which, in turn, physically connects to one-eighth of the co-processor memory. The latter corresponds to two memory DIMMs, thus the AE personality must decode the virtual memory address in a manner that ensures the dispatch of requests to the particular MC attached to their corresponding memory block. Furthermore, to ease timing in the FPGA, the 300 MHz interface is converted into two 150MHz memory ports to and from the AE personality. These ports are denoted as *odd* and *even* respectively and are multiplexed onto the same 300 MHz request channel in the MC interface.

A memory request is sent from the AE to an MC interface in one of the two half-rate ports. The platform implements 4 types of requests, namely load, store, fence, and stall. For store operations, the write data is stored in a first-in, first-out buffer until it is sent through the AE-MC link, while no response is returned to the AE personality. In load operations, however, a write data bus is used to store read return control information until the load request is sent out. When the request is sent to the MC, the read request data are transferred from the write data buffer to a read control buffer based on a transaction ID that is subsequently used to lookup the read control information when the data requested are returned from the MC. Finally, fence requests are sent by the AEH to enforce ordering of loads and stores to memory, while stall requests impede the personality from sending any other requests for two clock cycles to avoid overflowing the MC interface buffers.

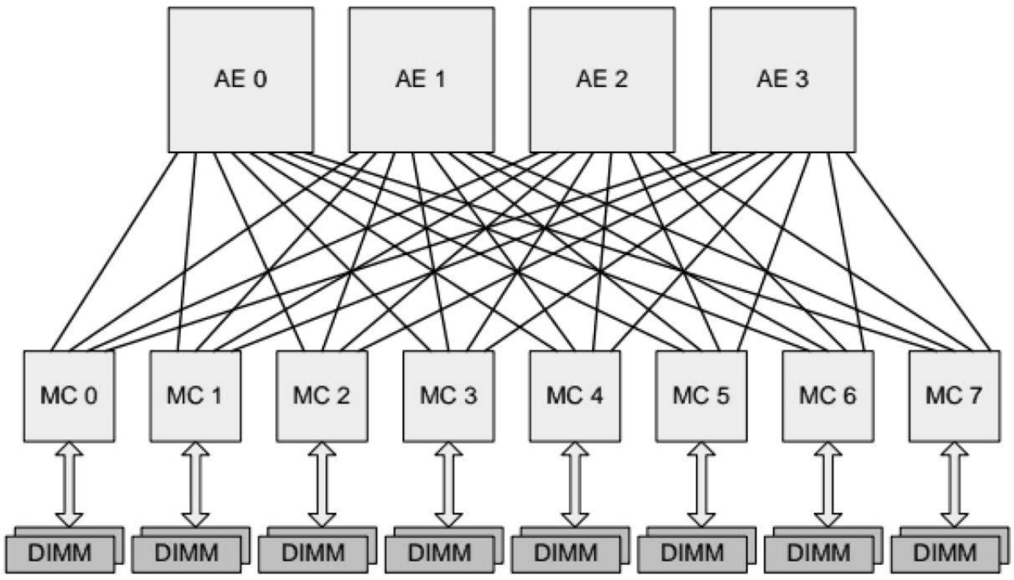


FIGURE 5.3: Diagram of the Connections between the co-processor AEs and memory (source: [13]).

5.2 LD Accelerator Instantiation

The instantiation of the hardware LD accelerator is comprised of two distinct phases. The first phase entails the data preparation on the host processor along with the function call to the co-processor and the instructions that kickstart the execution of the custom personality on the co-processor. The second phase begins with evaluating the number of individual components of the design, as described in Chapter 4, and proceeds with mapping the instantiated architecture onto the AE FPGAs.

5.2.1 Pre-processing on the Host Processor

A C-based application running on the host processor is responsible for handling the raw genomic data and realigning the dataset into the memory layout described in Section 4.1. Then, a memory block of the same size as the transformed dataset is allocated on the shared co-processor memory and the dataset is subsequently copied there, while, depending on the number of pairwise LD correlations, an equivalent memory space for the results is allocated on both the host and co-processor memory. To make the function call to the co-processor the following information must be passed along to the AEs:

- A pointer to the first address of the contiguous memory space that the input data are located.
- A pointer to the first address of the contiguous memory space that the output data will be stored.
- The number of SNP pairs that need to be correlated.
- The number of words in memory that comprise the portion of the data of one SNP that are assigned to one AHF unit.

- By extension of the two previous values, the total SNP size of all the SNP data assigned to one AHF unit, which corresponds to the total number of read operations for each MC.
- A numeric value that represents the excess bits, if present, at the final word of each SNP.
- The total number of SNPs in the dataset.

The two first variables are the pointers to the data structure that our input and output data will reside in the co-processor memory. The rest of the values are constants for each unique execution of the application and they do not need to be retrieved by the MCs, rather they are passed as values directly to the architecture. At this point the software has gathered all the necessary information required for the co-processor to start processing the dataset, so the former implements a function call to the latter through the convey-specific $copcall\_fmt()$ function. Since the number of attributes that the copcall function may take is limited to 15 64-bit values, which will be a hindrance later, the 5 constants required for the execution of the application are passed to the co-processor through 3 64-bit variables, namely Var_1, Var_2 and Var_3, and the data are organized as follows:

- Var$_1$ with bits 0. . . 56 holds the number of read operations.
- Var$_1$ with bits 57. . . 63 holds the number excess bits.
- Var$_2$ with bits 0. . . 31 holds the number of SNP pairs.
- Var$_2$ with bits 32. . . 63 holds the SNP size.
- Var$_3$ holds the total number of SNPs.

After the host has requested the start of the execution by the co-processor and before the latter is able to start processing, the correct initial addresses for each MC are assigned through a short assembly script, which also routes the constants to the corresponding ports of the architecture. Since only one pointer is provided for the input data memory space, this script implements scalar instructions to determine the correct initial address for each of the two ports of each MC through the use of a predefined step depending on the dataset size. The script ends by notifying the co-processor that all data and initial addresses are in order, thus the AEs can start processing the dataset. Note that each assembly instruction may be formed to refer to any number of AEs, therefore the user may choose how many AEs will share the workload.

Up to this point, we described the pre-processing of the dataset for the deployment of a single AE. When multiple AEs are deployed the same process is followed, albeit with some minor differences. The first issue that needs to be addressed when deploying multiple parallel AEs for the processing of a single dataset is the workload distribution between them. Ideally, if m AEs are operating, each AE should compute $1/m^{th}$ of the total pairwise LD scores. However, since each calculation of the LD scores of a diagonal in the correlation matrix is followed by a memory rollback, resulting in a multiple cycle delay of the arrival of the initial data of the next diagonal, and since diagonals vary on the number of LD scores they contain, a simple division would not suffice. For this reason, the workload is distributed through whole diagonals and not individual LD scores. The first step of the process is to find the $1/m^{th}$ value that will be used as a bound for the distribution. Then, starting from the first diagonal, the number of diagonals that contain that bound is calculated and assigned to each AE. Since the upper diagonals contain more LD scores than the lower ones, fewer diagonals will be assigned to the first AE than the last one. However, they are essentially assigned approximately the same number of LD scores, with the first AEs having a slightly higher workload that is balanced out by the lower memory overhead during rollbacks as a result of the reduced number of diagonals they are assigned to process.

Since the workload slightly varies between AEs, all constants except the number of SNPs must be individually computed and passed to each AE. Furthermore, since multiple AEs are producing results simultaneously, four extra constants are necessary in order to dictate the initial address of the result memory space for each AE. These facts elevate the number of the copcall function's attributes to the maximum allowed value of 15, which reinforces the decision to concatenate multiple constants into single 64-bit quantities. The assembly script is structured in the same fashion for multi-FPGA instantiations as for single-FPGA ones, with the difference that each AE is assigned different addresses and constants, depending on the outcome of the aforementioned workload distribution procedure.

5.2.2 Hardware Architecture Instantiation

Before the LD accelerator can be mapped on the AE FPGAs, the generic design described in Chapter 4 must be instantiated by defining the number of MCs that will be used and, consequently, the number of ports, AHF units, and COR units that the implemented design will feature, as well as the width of the adder trees on the BC stage 2. Since each AE processes a different SNP pair, the same bitfile must be downloaded on all FPGAs. Thus, we will attempt to describe the instantiation process for one FPGA, which remains the same for all AEs regardless of the number of FPGAs deployed.

Each application engine has access to 16 total memory controller ports. Our implementation utilizes an odd number of ports, two for each MCP and one for the output specifically. For that reason, out of the 16 ports, the 15 are utilized delivering to the accelerator 14 total input and one output port. To completely absorb the bandwidth of the 14 input ports, 7 AHF units are instantiated, since a pair of input ports is tasked with retrieving a portion of the data of a SNP pair. Consequently, the width of the 25 adder trees of BC stage 2 is set to 7 in order to accumulate all partial allele, haplotype and valid counts of the 7 AHF units of the previous stage. The number of COR units, on the other hand, is arbitrary and unrelated to the number of ports provided. We chose to implement 4 COR units operating in parallel with the reasons being the extended support of moderately small FSM datasets.

At this point the accelerator is ready to be mapped onto the FPGAs. The proposed architecture is described using VHDL and developed with the Xilinx ISE 12.4 interface, while the floating-point cores and memory elements are generated by the Xilinx Core Generator. Table 5.1 details the resources utilized when the accelerator is deployed as a single instance onto one Virtex6 LX760 FPGA compared to the resources utilized when the complete system is deployed. The latter includes the implementation of the accelerator itself, as well as the vendor-provided infrastructure of the Convey co-processor. As clearly denoted by the table above, only a small fraction of the available resources of the FPGA are utilized for the implementation of the architecture. This is a direct consequence of the design decision to support arbitrarily large datasets. Thus, any attempt to implement the proposed design is

TABLE 5.1: Occupied resources on a Virtex6 LX760 FPGA by a single accelerator instance, as well as the fully functional system.

	Single Instance		Full System	
Resources	Amount	Occupied	Amount	Occupied
Occupied Slices	11,778	9%	43,969	37%
Slice Registers	42,453	4%	150,932	15%
Slice LUTs	36,717	7%	124,909	26%
BRAMs (18Kb)	175	12%	437	30%
DSPs	96	11%	96	11%

bound first by the available memory bandwidth and, if the latter is large enough, only then will the available resources become a hindrance.

The throughput of the accelerator under the FSM when 1 FPGA is utilized can be described theoretically by a series of equations. Given the number of SNP pairs P that need to be correlated, then the throughput, measure in LD scores per second, is given by the following formulation:

$$throughput = \frac{P}{t}, \tag{5.1}$$

where t refers to the time required for retrieval, processing and storage of the dataset. The time required for I/O purposes can be calculated by means of the number of sequences in the dataset (S), the number of nucleotide bases (b), the utilized memory bandwidth (B), measured in bits/s, and the word size (W) as:

$$t_{IO} = \frac{2bSP + WP}{B}. \tag{5.2}$$

The processing time needs to be accounted for only once, since the design is fully pipelined and presents the ability to fully absorb the available memory bandwidth. Therefore, the design latency can be denoted as l_{design} and amounts to a total of 131 clock cycles. If we account for the initial overhead until the MCs start operating (t_{setup}) and the total time spent on retrieving unbuffered data from memory ($t_{rollback}$), then the total time t required for the processing of a dataset by 1 FPGA under the FSM assumption is:

$$t = t_{setup} + t_{IO} + t_{rollback} + \frac{l_{design}}{F}, \tag{5.3}$$

where F denotes the clock frequency under which the architecture operates.

Chapter 6

System Evaluation

With the conclusion of the instantiation process of the design, the architecture needs to be properly vetted for correct operation and, eventually, evaluated concerning its performance in comparison with reference software when processing a wide array of datasets. Hereby we describe the verification process followed during the implementation of the accelerator and present a series of performance comparisons between the full accelerator system and two state-of-the-art high-performance software for LD analyses, OmegaPlus and PLINK.

6.1 System Verification

The development process of the accelerator design was accompanied by continuous simulations that verified the correct operation of each individual module before it was connected to the rest of the architecture. All individual modules and stages of the architecture implementation before the full system was completed were tested through waveforms using the Xilinx ISim 12.4 simulator. For the sake of verifying the system's operation, a spreadsheet modeling the individual stages described in Chapter 2 given minimum datasets was developed. This model allowed the cross-reference between the theoretical results and the ones produced by the implemented architecture at all intermediate stages of the design, therefore enabling the verification of the accelerator's operation at every point on the pipeline. The fully implemented system's operation was at first verified using ModelSim SE-64 6.5c, along with a simulation infrastructure provided by Convey Computers, since the full operation of the implementation on the target platform needed to be simulated. By performing extensive simulations of the system and its components, it's correct and synchronized operation was assured before attempting to map it onto the actual platform, thus minimizing the chance of the latter malfunctioning.

After concluding with certainty that the simulations of the architecture confirm the proper operation of the design in regards to result correctness, it was ready to be mapped onto the AE FPGAs. In order to reduce the risk of malfunction due to incorrect operation of the full system, the final verification process was subdivided into 3 stages. The first stage entails the verification of the software running on the host processor without a function call to the co-processor. It is an important step which ensures that the dataset is handled correctly and the correct constants are calculated before attempting to pass them to the co-processor for processing. Furthermore, it is essential that enough memory space has been allocated in both the main and the shared memory, in order to accommodate all input and output data before proceeding with their processing. The second verification stage addressed the proper retrieval and storing of data to and from the AEs. Instead of the full architecture, a small design implementing a few simple computations was mapped onto the FPGAs in order to verify the correct operation of the memory controllers and the FSMs dictating their functionality. This step was necessary as erroneous data management could very easily lead to incorrect behavior from the accelerator and, possibly, even system-wide crashes. The full system was finally implemented and tested during the third stage, where its complete operation was verified by cross-referencing the results produced by small datasets with their corresponding known expected values calculated through the means of the theoretical methods detailed in Chapter 2 of this document. Finally, with a working prototype in place, small-scale optimization improvements were implementing, attempting to reach higher operating clock frequencies. The achieved stable clock frequency is 149.9 MHz, with 150 MHz being an upper bound imposed by the Convey platform.

The Convey platform consists a complicated system since it is a hybrid computing machine pairing traditional high-end CPUs with vendor-specific multi-FPGA-based co-processing in the form of both the MC interface and the AEs themselves. Factoring in the scale of complexity that encompasses LD calculations, as well as the relatively short amount of experience of the author concerning such intricate systems, each stage of the verification process, from the simplest module to the whole architecture mapped on multiple AE FPGAs, needed to be extensive. The process described in this section of the present document ensured that, by verifying each part of the architecture as it was being implemented, fewer errors were introduced in the later stages of its development, which would otherwise be harder to address because of the increasing complexity of the design as it was approaching completeness.

6.2 Performance Evaluation

The problem of computing LD scores, as mentioned multiple times in this document, is memory-bound. For this reason, the Convey platform was specifically chosen for its high-speed memory interface, providing a bandwidth of up to 80 GB/s. Pairing this feature with an efficient highly parallelized architecture for LD analysis is theorized to lead to increased throughput when processing very large datasets. To support this hypothesis, the accelerator system is evaluated through the performance comparison with 2 efficient parallel algorithms for LD analyses, namely OmegaPlus and PLINK. In addition, the architecture is mapped to 1,2 and 4 AE FPGAs in order to demonstrate the scalability of the design when deployed on multiple parallel machines.

To evaluate the performance of the accelerator under the FSM model, OmegaPlus 3.0.0 [2] was used as the reference software. Table 6.1 details the execution times and throughput of the hardware implementation when the accelerator is deployed for LD analyses of datasets consisting of up to 1,000,000 sequences. Similar data are presented in Table 6.3 concerning analyses of the same datasets conducted with the fine grain version of OmegaPlus, as well as the optimized fine grain version of the software (Opt.), which implements the Intel intrinsic population count instruction *_mm_popcnt_u64*, while multiple threads are deployed. For the purpose of the software execution, the testing workstation deployed was comprised by two Intel Xeon E5-2620 6-core processors running at 2.00 GHz and 24 GB of main memory.

From the data presented in the aforementioned tables, several conclusions can be deduced concerning the performance of the accelerator system. Concerning the scaling of the performance when multiple FPGAs are deployed, a linear increase

TABLE 6.1: Performance of the FPGA accelerator for FSM datasets consisting of 10,000 SNPs and 1,000, 100,000 and 1,000,000 sequences by deploying 1,2 and 4 FPGAs.

	LD exec. time (s)			Throughput (kLD/s)		
FPGAs	10^3	10^5	10^6	10^3	10^5	10^6
1	4.23	298.74	2964.85	11829.62	167.45	16.86
2	2.23	152.72	1482.26	22428.30	327.37	33.73
4	3.37	111.59	842.25	14816.03	448.02	59.36

TABLE 6.2: Scaling of performance of the FPGA accelerator for FSM datasets consisting of 10,000 SNPs and 1,000, 100,000 and 1,000,000 sequences when increasing the number of parallel FPGAs.

	Speedup (X)		
FPGAs	10^3	10^5	10^6
1->2	1.90	1.96	2.00
2->4	0.66	1.37	1.76
1->4	1.25	2.68	3.52

of throughput was expected to accompany the increase of parallel FPGAs processing the same dataset. However, Table 6.2 presents two discrete deviations from the expected behavior of the system. Firstly, while in small datasets an increase of the number of FPGAs utilized from 1 to 2 is followed by a 1.9x speedup, this is not the case when an equivalent increase from 2 to 4 FPGAs is implemented. On the contrary, a decrease of performance is observed with the throughput plummeting from approximately 22.4 million LD scores per second to just 14.8 million. This behavior can only be attributed to the increased synchronization overhead required to utilize 4 AEs instead of 2 in conjunction with the very low execution times that are required for analyses of small sample sizes, amounting to just a few seconds. With the gradual increase of the sample size, however, it is safe to conclude that more devices translate to improved performance, since the computation-to-synchronization ratio becomes favorable due to excessive computational requirements.

The second observation that can be made is that the overall performance gain from deploying multiple FPGAs is not as high as expected. As was previously the case, performance for analyses of smaller datasets seems to benefit less from the deployment of multiple parallel machines due to the increased setup and synchronization overhead. On the other hand, when processing larger datasets, the speedup gained by utilizing a larger number of AEs tends to match the theoretical expected values, reaching a 2x and 3.52x speedup when utilizing 2 and 4 FPGAs instead of 1 respectively. Besides the static temporal overhead needed to setup and synchronize multiple machines, the increase of relative performance when analyzing larger datasets and the decrease of the former when deploying multiple FPGAs is caused by a common factor, namely a mediocre optimization of the method that distributes the workload between the AEs. This is better illustrated in Figure 6.4, were the execution times and throughput of the accelerator system is showcased for datasets comprised of a fixed number of sequences and varying number of SNPs. In this case, while the overall execution time is expected to increase with increasing sample sizes, the throughput of the system should remain the same when the same number

TABLE 6.3: Performance of OmegaPlus 3.0.0 for FSM datasets consisting of 10,000 SNPs and 1,000, 100,000 and 1,000,000 sequences (Opt. = intrinsic population counter implemented).

	OmegaPlus 3.0.0					
	LD exec. time (s)			Throughput (kLD/s)		
threads	10^3	10^5	10^6	10^3	10^5	10^6
1	151.11	11307.24	113647.76	330.84	4.42	0.44
2	80.98	5824.79	56736.29	617.40	8.58	0.88
4	42.47	2921.36	29176.21	1177.04	17.11	1.71
8	21.57	1528.04	14883.85	2317.43	32.72	3.37
12	22.64	1689.48	11920.92	2208.58	29.59	4.19

	OmegaPlus 3.0.0 Opt.					
	LD exec. time (s)			Throughput (kLD/s)		
threads	10^3	10^5	10^6	10^3	10^5	10^6
1	72.71	3960.23	39818.63	687.61	12.62	1.26
2	41.92	2048.73	19813.02	1192.57	24.40	2.52
4	21.64	1080.31	10000.93	2310.09	46.28	5.00
8	15.67	707.69	5351.34	3191.08	70.65	9.34
12	10.56	594.61	4303.89	4734.14	84.08	11.62

TABLE 6.4: Performance of the accelerator for datasets of 5,008 sequences and a varying number of SNPs.

		LD exec. time (s)			Throughput (mLD/s)		
Number of SNPs	Number of Pairs	1	2	4	1	2	4
1000	499500	0.16	0.08	0.05	3.05	5.98	10.14
5000	12497500	4.08	2.07	1.19	3.06	6.04	10.50
10000	49995000	16.30	8.26	4.70	3.07	6.05	10.64
20000	199990000	65.22	32.98	20.41	3.07	6.06	9.80

of AEs are utilized. This is better explained when we consider that the throughput is defined as the number of LD scores the system can produce in 1 second. Since the relation of the number of SNPs in the dataset to the processing time of the data is linear, provided that the number of sequences remains fixed, then the ratio of LD scores per time unit should remain fixed. However, as the experimental results suggest, there are minor fluctuations on the throughput measurements, even among analyses with the same number of sequences and utilized AEs. This is a result of the presence of various synchronization and initialization overheads, such as the time needed to process and decode the initial instructions for the function call to the co-processor, the increasing number of memory leaps that are larger than a single memory page when the number of SNPs increases, and the time required to copy datasets of varying size from the main to the shared memory and vice versa, which is included in our measurements. A more significant observation is that the deviance of the throughput from an average value grows as more FPGAs are utilized. This is a direct result of a poor optimization of the workload distribution among the AEs. However, it still happens to a minor degree and is not harmful to the overall performance of the accelerator.

TABLE 6.5: Performance evaluation of the FPGA accelerator vs OmegaPlus 3.0.0 for FSM datasets consisting of 10,000 SNPs and 1,000, 100,000 and 1,000,000 sequences when increasing the number of parallel FPGAs (Opt. = intrinsic population counter implemented).

		Speedup (X) vs OmegaPlus 3.0.0 threads				
sample size	FPGAs	1	2	4	8	12
	1	35.76	19.16	10.05	5.11	5.36
10^3	2	67.79	36.33	19.05	9.68	10.16
	4	44.78	24.00	12.59	6.39	6.71
	1	37.85	19.50	9.78	5.12	5.66
10^5	2	74.03	38.14	19.13	10.01	11.06
	4	101.32	52.20	26.18	13.69	15.14
	1	38.33	19.14	9.84	5.00	4.02
10^6	2	76.67	38.28	19.68	10.01	8.04
	4	134.93	67.38	34.63	17.61	14.15

		Speedup (X) vs OmegaPlus 3.0.0 Opt. threads				
sample size	FPGAs	1	2	4	8	12
	1	17.20	9.92	5.12	3.71	2.50
10^3	2	32.62	18.81	9.71	7.02	4.73
	4	21.55	12.42	6.41	4.64	3.13
	1	13.26	6.86	3.62	2.37	1.99
10^5	2	24.59	13.42	7.07	4.64	3.89
	4	33.65	18.36	9.68	6.34	5.33
	1	13.43	6.68	3.37	1.80	1.80
10^6	2	26.85	13.37	6.75	3.61	3.61
	4	47.26	23.53	11.87	6.35	6.35

In regards to the performance of the accelerator compared to the OmegaPlus software when deployed for analyses of gradually increasing sample sizes, the FPGA implementation achieves higher bit-count capacity and, therefore, better overall performance, as detailed in Table 6.5. The table is comprised by the speedup achieved when the accelerator is mapped on 1, 2 and 4 FPGAs compared to the normal and optimized fine-grain version OmegaPlus 3.0.0 software. The performance gain, while not particularly impressive (approximately 5-6X when comparing the fastest versions of both implementations for large datasets), they are within the expected range. The design decision to support arbitrarily large sample sizes forbids any further optimization of the architecture, thus establishing the current design as optimal, since it absorbs the maximum amount of memory bandwidth possible for the target platform. For this reason, in order to achieve higher execution times, either the available memory bandwidth should be increased, followed by an equivalent increase of the parallelism in the accelerator's pipeline, or the whole design perspective should undergo a focus shift. Furthermore, when examining analyses of a wider range of sample sizes, as illustrated in Figure 6.1, an increase in sample size slowly but surely is followed by a stabilization of the performance gain over the reference software. This is expected as both the software and the hardware accelerator exhibit a linear relation between the dataset size and its processing time, while on datasets whose size does not permit efficient use of the cache hierarchy on the commercial CPUs,

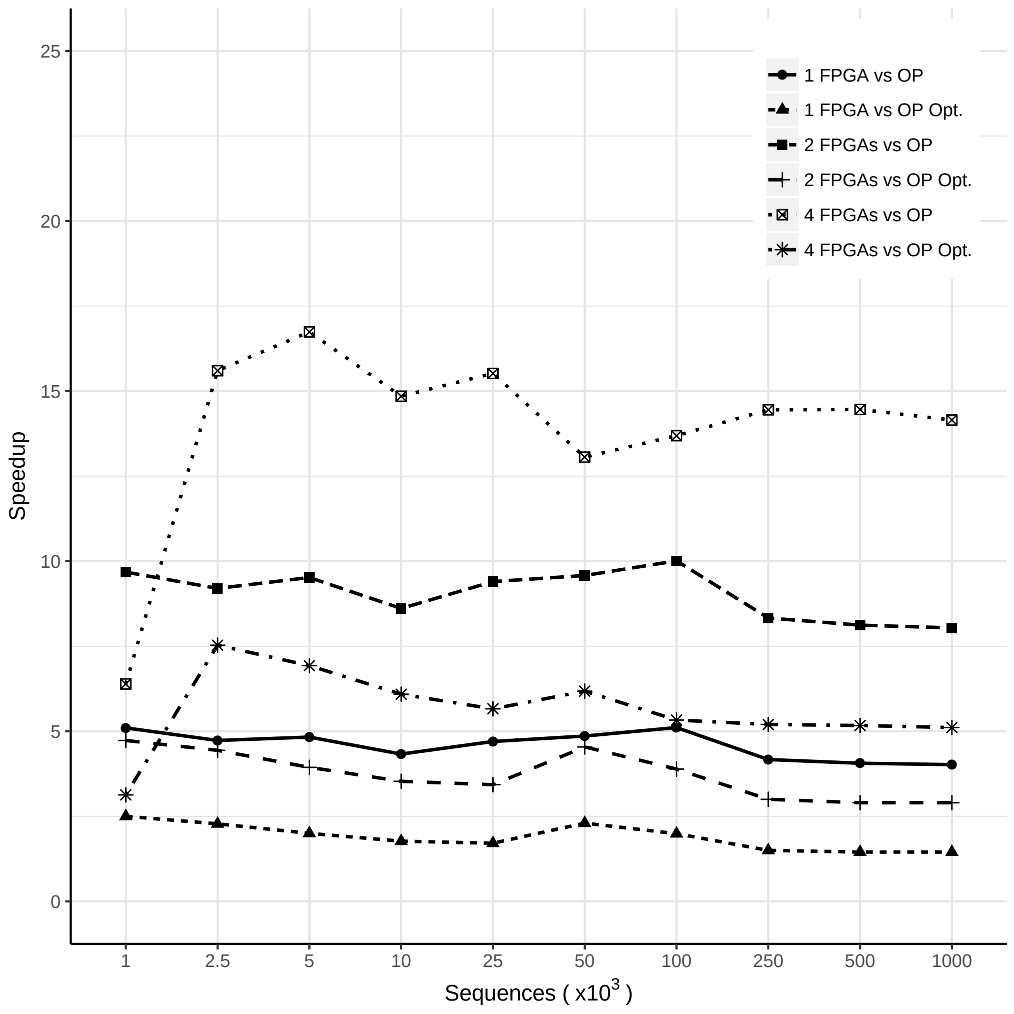


FIGURE 6.1: Speedup of the FPGA accelerator deployed on 1,2 and 4 FPGAs vs the normal and optimized OmegaPlus 3.0.0 reference software for datasets consisting of 10,000 SNPs and 1,000 to 1,000,000 sequences. Additional data supporting the diagram can be found in Appendix A.

FPGAs appear to gain an advantage due to their performance stability.

For the performance evaluation of our architecture would it be deployed to process datasets according to the ISM model, PLINK 1.9 [9] was chosen over OmegaPlus 3.0.0 since it outperforms the latter when more than 4 CPU cores are deployed [3]. However PLINK computes LD based on genotypes and not alleles, thus exhibiting increased complexity in comparison with the allele-based OmegaPlus implementation under the FSM. Note, however, that the speedup of our accelerator system reported in this case refers to mere estimates of its performance under the ISM model, based on previous analyses conducted under the FSM model. To ensure a fair comparison with the ISM implementations, we include the evaluation results of the previous hardware implementation by Alachiotis and Weisz [3] as reported by the authors (Table 6.6). The test platforms for the ISM evaluation were a workbench featuring an Intel Xeon E5-2630 6-core processor running at 2.60 GHz with 32 GB of main memory for the PLINK 1.9 measurements, while the LD accelerator [3] was mapped to a Virtex7 VX980T FPGA.

The reported evaluation scores between the 3 ISM implementations clearly lean

TABLE 6.6: Estimated performance comparison for analyzing 10,000 SNPs and 100,000 sequences based on ISM.

	PLINK 1.9		LD Accel. [3]	HW speedup	
threads	Exec. Time	kLD/s	1 FPGA	1 FPGA	4 FPGAs
1	389.1	128	159.3	20.9	56.0
2	297.6	168	121.4	15.9	42.7
4	180.2	277	73.6	9.7	25.9
8	109.4	456	44.7	5.9	15.7
12	88.3	566	36.0	4.7	12.7

in favor of the LD accelerator presented by Alachiotis and Weisz [3], as it achieves a speedup of 36X versus PLINK operating on 12 threads, while our accelerator potentially achieves a mere 4.7X speedup utilizing a single FPGA versus the same software. However, the design point of the two hardware accelerators differs to a significant degree. On the one hand, the work of Alachiotis and Weisz is focused on maximizing throughput performance for moderate sample sizes, while only assuming complete ISM data. In our work the focus is to support arbitrarily large sample sizes while considering missing data and alignment gaps to allow potential deployment in future real-world large-scale analyses. Furthermore, our architecture is optimized for accommodating primarily FSM datasets, with the support for ISM data being a potential extra functionality.

To this end, we report expected performance gains from the potential deployment of the multi-FPGA accelerator system in a real-world analysis for the detection of positive selection on the 22nd human chromosome. The dataset from the 1,000 genomes project [56] containing that information comprises 5,008 sequences and 1,055,736 SNPs. The generic version of OmegaPlus 3.0.0 with the Intel intrinsic population counter implemented operating on 12 cores achieves a throughput of 1.88 million LD/s. As illustrated in Table 6.4, the hardware accelerator system achieves throughputs between 9.8 and 10.64 million LD/s when all 4 FPGAs are utilized, yielding a speedup of approximately 5X compared to the aforementioned software.

Chapter 7

Conclusions and Future Work

Linkage disequilibrium computations constitute a subject that is characterized by complex calculations. However it is of high significance to the biological and Bioinformatics community due to the genetic information it represents and the wide range of further analyses it can enable. In a landscape of ever-increasing genomic dataset sizes, developing tools that are able to efficiently process sample sizes numbering millions of DNA sequences presents a noble challenge to computer engineers worldwide. By concluding this work, we attempt to further the search of a highly efficient tool for genomic association analyses through the means of developing custom computing machines.

7.1 Conclusions

With conventional CPUs falling out of favor due to their limitations concerning the disproportion between Moore's law and the rate that DNA data are being sequenced, as well as the absence of custom instructions, such as a vectorized population counter, a custom architecture based on reconfigurable logic would provide a more efficient alternative through the exploitation of multiple degrees of parallel operations combined with a high bandwidth memory interface for the retrieval and analysis of large-scale sample sizes.

The current work attempted to design and develop such a system and deploy it for analyses based on simulated data, while evaluating its potential real-world application. The implementation process entailed the gradual design of the architecture following a top-down approach. The first stages were focused on detecting the individual key calculations comprising LD and which of them would constitute the bottlenecks of the accelerator. Each stage ventured deeper in the pipeline, describing its computational elements with higher detail. Parallelized components were introduced in key junctions of the pipeline where the dataflow would come to a halt due to bottlenecks, i.e. population count etc. Only then was the accelerator ready to be instantiated following a bottom-up approach for easier debugging and faster development.

A high-end hybrid computing system with increased computational capacity was selected to accommodate the architecture in order to fully exploit its capabilities for the parallelization of the operations required for the estimation of LD. The combination of a high-bandwidth memory interface with an architecture featuring multiple levels of computational parallelism (population count, allele/haplotype frequencies, multiple FPGAs) proved to perform better than state-of-the-art software running on high-end CPUs. Even thought the design was not tailored for extreme performance efficiency, but rather focused on a combination of high performance and the support of sample sizes that will become staple for genomic analyses over the next few years, it still managed to process large datasets 5 times faster than its current software counterparts.

In addition, both the user-oriented nature of reconfigurable machines, as well as the overall modular structure of the design encourage further design space exploration of the LD architecture. Since the LD statistic is so widely used, presenting a design that could be altered and improved upon on the fly by a knowledgeable user is considered an important feature of the accelerator. This approach was only possible due to the FPGA-specific feature of sharing the modularity of general purpose processors and implementing it on a hardware level while contesting the efficiency and performance of ASICs without inheriting their finality.

Thus, the most important conclusion of this work would have to be that analyses, such as LD and a lot of other biological computations, that require complex algorithms, yet when broken down, consisting of a large amount of repetitive simple calculations, highly benefit from the development of custom application-specific pipelines that exploit the capacity of reconfigurable devices for cheap and efficient parallel implementations with custom-width arithmetic addressing exactly the application's requirements. Both the previous work of Alchiotis and Weisz [3] and the current work represent, in our opinion, important steps towards the exploration of custom accelerators in LD analyses. However, we don't consider this work to be the ultimate tool of such analyses, but rather a small contribution to the fields of reconfigurable computing and Bioinformatics that is to be improved upon, as hybrid computing seems to be entering its most fruitful era yet.

7.2 Future Work

The work described in this manuscript, while novel to the field of Bioinformatics, since only a few FPGA-based implementations of LD architectures have been presented thus far, is not considered as the penultimate research upon the subject. Several propositions for future continuation of our architecture can be proposed, which, despite being out of the scope of the current study, could constitute interesting research subjects and even lead to significant advances in Bioinformatics.

First and foremost, the presented accelerator can be improved upon in a multitude of different ways. Similarly to the previous work of Alachiotis and Weisz [3], a memory grid can be inserted between the MCs and the AEs, which will be used to store whole SNPs, while additional LD cores can be instantiated per FPGA for the efficient processing of multiple SNP pairs simultaneously. The supported dataset size will be significantly reduced, even more so in the case of FSM-based analyses, however, the throughput of the accelerator is expected to be significantly increased. Nevertheless, depending on the scale of the dataset, the user would be presented with the option to choose between the implementation that best suits his analysis.

Additionally, the LD accelerator can be used as a basis for the further implementation of additional population genomics statistics. The implementation of the ω statistic implemented in OmegaPlus is, possibly, the most obvious example. Given the platform that our accelerator was instantiated on, there two routes may be followed towards that end. The one entails the implementation of the ω statistic and the dynamic programming algorithm applied on the correlation matrix on hardware. This is expected to lead to a slight increase in performance for datasets smaller than the size of the shared memory, since an analysis of this magnitude would require only one function call to the co-processor. For larger analyses, an argument for another parallel approach could be made. By organizing the computations into compute lists, the software could handle the calculation of the ω statistics referring to one compute list, while the co-processor calculated the LD scores of the next. This could be achieved by utilizing multi-threading in a fashion that allows for one thread to facilitate the function call to the AEs, while the rest work in parallel towards the calculation of the ω values.

As a next step in the evolution of the architecture, several of the aforementioned features can be combined into one design implementing multiple statistics. However, due to the nature of different statistics and various variables dependent on the dataset, such a multi-purpose tool would present significant fluctuation in the performance of different parts of the architecture and, possibly, shifting the bottleneck between them as different analyses are conducted. To address such issues, an interesting and ambitious proposition would entail an architecture that would be able to adapt on demand through partial reconfiguration, in order to meet the computational demands of different analyses and varying dataset sizes.

Lastly, the LD accelerator could be implemented on different heterogeneous platforms. Due to the need of a large amount of relatively simple computations during the calculation of allele and haplotype frequencies, a platform providing a great number of processing cores, such as GPUs, even if they are running at a lower speed than conventional CPUs, could lead to a significant performance gain. This argument can only be strengthened by the release of High Bandwidth Memory technology for GPUs, theorized to provide a bandwidth of up to 512 GB/s until 2020.

Concluding this study, we can only be hopeful for the future of hardware design, as more interesting advances in high-performance computing are underway. The progress in heterogeneous architectures and the efficient pairing of CPUs and FPGAs, that this work has barely grazed upon, can only lead to a blissful yearning for further experimentation around novel ideas and designs.

Appendix A

Additional Data

TABLE A.1: Performance of the FPGA accelerator for FSM datasets consisting of 10,000 SNPs and 1,000 - 1,000,000 sequences by deploying 1,2 and 4 FPGAs.

	FPGA Performance					
	LD exec. time (s)			Throughput (kLD/s)		
sequences (x10^3)	1	2	4	1	2	4
1	4.23	2.23	3.37	11829.62	22428.30	14816.03
2.5	8.27	4.25	2.51	6044.89	11759.72	19928.46
5	16.30	8.26	4.70	3067.16	6050.54	10639.62
10	30.80	15.51	8.99	1623.14	3223.71	5561.19
25	75.23	37.59	22.77	664.58	1329.89	2195.60
50	149.49	75.82	55.63	334.44	659.41	898.76
100	298.74	152.72	111.59	167.35	327.37	448.02
250	742.42	371.22	213.96	67.34	134.68	233.66
500	1483.54	741.69	416.16	33.70	67.41	120.13
1000	2964.85	1482.26	842.25	16.86	33.73	59.36

TABLE A.2: OmegaPlus 12-threaded execution times and throughput for a dataset consisting of 1,055,736 SNPs and 5,008 sequences (22nd human chromosome).

software version	number of pairs	LD exec. time (s)	Throughput (mLD/s)
fine-grain	1610858086	1648.68	0.98
fine-grain Opt.	1610858086	1041.61	1.55
generic	1643288612	2287.02	0.72
generic Opt.	1744192036	927.23	1.88

TABLE A.3: Speedup of the accelerator vs. various OmegaPlus 3.0.0 software versions for a dataset of 10,000 SNPs and 1,000 sequences (Opt. = intrinsic population counter implemented).

			Throughput	Speedup of accelerator for		
software version	threads	LD exec. time (s)	(kLD/s)	4 FPGA	2 FPGAs	1 FPGAs
sequential	1	151.12	330.84	44.78	67.79	35.76
sequential Opt.	1	72.71	687.62	21.55	32.62	17.20
fine-grain	2	80.98	617.40	24.00	36.33	19.16
	4	42.47	1177.05	12.59	19.05	10.05
	8	21.57	2317.43	6.39	9.68	5.10
	12	22.64	2208.58	6.71	10.16	5.36
fine-grain Opt.	2	41.92	1192.57	12.42	18.81	9.92
	4	21.64	2310.09	6.41	9.71	5.12
	8	15.67	3191.08	4.64	7.03	3.71
	12	10.56	4734.14	3.13	4.74	2.50
generic	2	125.12	399.57	37.08	56.13	29.61
	4	65.17	767.12	19.31	29.24	15.42
	8	35.31	1415.70	10.47	15.84	8.36
	12	33.65	1485.95	9.97	15.09	7.96
generic Opt.	2	41.83	1195.22	12.40	18.77	9.90
	4	22.19	2253.31	6.58	9.95	5.25
	8	12.73	3927.34	3.77	5.71	3.01
	12	13.26	3769.91	3.93	5.95	3.14

TABLE A.4: Speedup of the accelerator vs. various OmegaPlus 3.0.0 software versions for a dataset of 10,000 SNPs and 100,000 sequences (Opt. = intrinsic population counter implemented).

			Throughput	Speedup of accelerator for		
software version	threads	LD exec. time (s)	(kLD/s)	4 FPGA	2 FPGAs	1 FPGAs
sequential	1	11307.24	4.42	101.32	74.03	37.85
sequential Opt.	1	3960.23	12.62	35.49	25.93	13.26
fine-grain	2	5824.79	8.58	52.20	38.14	19.50
	4	2921.36	17.11	26.18	19.13	9.78
	8	1528.04	32.72	13.69	10.01	5.12
	12	1689.48	29.59	15.14	11.06	5.66
fine-grain Opt.	2	2048.73	24.40	18.36	13.42	6.86
	4	1080.31	46.28	9.68	7.07	3.62
	8	707.69	70.65	6.34	4.63	2.37
	12	594.61	84.08	5.33	3.89	1.99
generic	2	10332.57	4.84	92.59	67.65	34.58
	4	5374.83	9.30	48.16	35.19	17.99
	8	2831.49	17.66	25.37	18.54	9.48
	12	2265.21	22.07	20.30	14.83	7.58
generic Opt.	2	2103.77	23.77	18.85	13.78	7.04
	4	1118.27	44.71	10.02	7.32	3.74
	8	686.11	72.87	6.15	4.49	2.30
	12	640.47	78.06	5.74	4.19	2.14

TABLE A.5: Speedup of the accelerator vs. various OmegaPlus 3.0.0 software versions for a dataset of 10,000 SNPs and 1,000,000 sequences (Opt. = intrinsic population counter implemented).

software version	threads	LD exec. time (s)	Throughput (kLD/s)	Speedup of accelerator for 4 FPGA	2 FPGAs	1 FPGAs
sequential	1	113647.77	0.44	134.93	76.67	38.33
sequential Opt.	1	39818.63	1.26	47.26	26.85	13.43
fine-grain	2	56736.29	0.88	67.38	38.28	19.14
	4	29176.21	1.71	34.63	19.68	9.84
	8	14883.85	3.37	17.61	10.01	5.00
	12	11920.92	4.19	14.15	8.04	4.02
fine-grain Opt.	2	19813.02	2.52	23.53	13.37	6.68
	4	10000.93	5.00	11.87	6.75	3.37
	8	5351.34	9.34	6.35	3.61	1.80
	12	4303.89	11.62	5.11	2.90	1.45
generic	2	103219.19	0.48	122.64	69.69	34.84
	4	52442.28	0.95	62.29	35.39	17.69
	8	27544.45	1.82	32.70	18.58	9.29
	12	20051.70	2.49	23.81	13.53	6.76
generic Opt.	2	20560.45	2.43	24.41	13.87	6.93
	4	10986.13	4.55	13.04	7.41	3.71
	8	5775.98	8.66	6.86	3.90	1.95
	12	4974.38	10.05	5.91	3.36	1.68

TABLE A.6: Speedup of the accelerator vs. the fine-grain version of OmegaPlus 3.0.0 operating on 12 cores for datasets of 10,000 SNPs (Opt. = intrinsic population counter implemented).

sequences (x10^3)	software version	LD exec. time (s)	Throughput (kLD/s)	Speedup of accelerator for 4 FPGA	2 FPGAs	1 FPGAs
2.5	fine-grain	39.13	1277.75	15.60	9.20	4.73
	fine-grain Opt.	18.89	2646.02	7.53	4.44	2.28
5	fine-grain	78.67	635.53	16.74	9.52	4.83
	fine-grain Opt.	32.59	1534.23	6.93	3.94	2.00
10	fine-grain	133.50	374.48	14.85	8.61	4.33
	fine-grain Opt.	54.74	913.36	6.09	3.53	1.78
25	fine-grain	353.41	141.46	15.52	9.40	4.70
	fine-grain Opt.	128.86	387.99	5.66	3.43	1.71
50	fine-grain	726.63	68.80	13.06	9.58	4.86
	fine-grain Opt.	343.87	145.39	6.18	4.54	2.30
250	fine-grain	3092.87	16.17	14.45	8.33	4.17
	fine-grain Opt.	1112.45	44.94	5.20	3.00	1.50
500	fine-grain	6019.10	8.31	14.46	8.12	4.06
	fine-grain Opt.	2149.91	23.25	5.17	2.90	1.45

Bibliography

[1] Nikolaos Alachiotis, Thom Popovici, and Tze Meng Low. "Efficient Computation of Linkage Disequilibria as Dense Linear Algebra Operations". In: *High Performance Computational Biology (HICOMB)* (2016).

[2] Nikolaos Alachiotis, Alexandros Stamatakis, and Pavlos Pavlidis. "OmegaPlus: a scalable tool for rapid detection of selective sweeps in whole-genome datasets". In: *Bioinformatics* 28.17 (2012), pp. 2274–2275.

[3] Nikolaos Alachiotis and Gabriel Weisz. "High Performance Linkage Disequilibrium: FPGAs Hold the Key". In: *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM. 2016, pp. 118–127.

[4] Md Tauqeer Alam et al. "Selective sweeps and genetic lineages of Plasmodium falciparum drug -resistant alleles in Ghana." In: *The Journal of infectious diseases* 203.2 (Jan. 2011), pp. 220–7. ISSN: 1537-6613. DOI: 10.1093/infdis/jiq038. URL: http://jid.oxfordjournals.org/content/203/2/220.long.

[5] Jeffrey C Barrett et al. "Haploview: analysis and visualization of LD and haplotype maps". In: *Bioinformatics* 21.2 (2005), pp. 263–265.

[6] Jonathan P Beauchamp. "Genetic evidence for natural selection in humans in the contemporary United States". In: *Proceedings of the National Academy of Sciences* 113.28 (2016), pp. 7774–7779.

[7] Penelope E Bonnen et al. "Haplotype and linkage disequilibrium architecture for human cancer-associated genes". In: *Genome research* 12.12 (2002), pp. 1846–1853.

[8] Fabio Cancare, Alessandro Marin, and Donatella Sciuto. "Dedicated hardware accelerators for the epistatic analysis of human genetic data". In: *Embedded Computer Systems (SAMOS), 2011 International Conference on*. IEEE. 2011, pp. 102–109.

[9] Christopher C Chang et al. "Second-generation PLINK: rising to the challenge of larger and richer datasets". In: *Gigascience* 4.1 (2015), p. 7.

[10] Charles C Chung et al. "Genome-wide association studies in cancer-current and future directions". In: *Carcinogenesis* (2009), bgp273.

[11] Andrew G Clark. "Finding genes underlying risk of complex disease by linkage disequilibrium mapping". In: *Current opinion in genetics & development* 13.3 (2003), pp. 296–302.

[12] Convey Computer Corporation. *Convey PDK Reference Manual v5.2*. 2012. URL: http://www.conveysupport.com/alldocs/ConveyPDKReferenceManual.pdf.

[13] Convey Computer Corporation. *Convey Reference Manual v1.1*. 2012. URL: http://www.conveysupport.com/alldocs/ConveyReferenceManual.pdf.

[14] Olivia Corradin et al. "Combinatorial effects of multiple enhancer variants in linkage disequilibrium dictate levels of gene expression to confer susceptibility to common traits". In: *Genome research* 24.1 (2014), pp. 1–13.

[15] Jessica L Crisci et al. "The impact of equilibrium assumptions on tests of selection". In: *Frontiers in genetics* 4 (2013).

[16] Keyue Ding et al. "LDA—a java-based linkage disequilibrium analyzer". In: *Bioinformatics* 19.16 (2003), pp. 2147–2148.

[17] Laurent Excoffier, Guillaume Laval, and Stefan Schneider. "Arlequin (version 3.0): an integrated software package for population genetics data analysis". In: *Evolutionary bioinformatics* 1 (2005).

[18] Laurent Excoffier and Heidi EL Lischer. "Arlequin suite ver 3.5: a new series of programs to perform population genetics analyses under Linux and Windows". In: *Molecular ecology resources* 10.3 (2010), pp. 564–567.

[19] Genomics England. *The 100,000 genomes project*. 2015. URL: https://www.genomicsengland.co.uk/the-100000-genomes-project/.

[20] Jorge González-Domínguez et al. "Parallel Pairwise Epistasis Detection on Heterogeneous Computing Architectures". In: *IEEE Transactions on Parallel and Distributed Systems* 27.8 (2016), pp. 2329–2340.

[21] Daniel F Gudbjartsson et al. "Large-scale whole-genome sequencing of the Icelandic population". In: *Nature genetics* 47.5 (2015), pp. 435–444.

[22] W. G. Hill and Alan Robertson. "Linkage disequilibrium in finite populations". In: *Theoretical and Applied Genetics* 38.6 (1968), pp. 226–231. ISSN: 1432-2242. DOI: 10.1007/BF01245622. URL: http://dx.doi.org/10.1007/BF01245622.

[23] Stephan Hutter, Albert J Vilella, and Julio Rozas. "Genome-wide DNA polymorphism analyses using VariScan". In: *BMC bioinformatics* 7.1 (2006), p. 409.

[24] Thomas H Jukes and Charles R Cantor. "Evolution of protein molecules". In: *Mammalian protein metabolism* 3.21 (1969), p. 132.

[25] Tony Kam-Thong et al. "GLIDE: GPU-based linear regression for detection of epistasis". In: *Human heredity* 73.4 (2012), pp. 220–236.

[26] Jan Christian Kässens et al. "High-speed exhaustive 3-locus interaction epistasis analysis on FPGAs". In: *Journal of Computational Science* 9 (2015), pp. 131–136.

[27] David R Kelley, Michael C Schatz, and Steven L Salzberg. "Quake: quality-aware detection and correction of sequencing errors". In: *Genome biology* 11.11 (2010), p. 1.

[28] Yuseob Kim and Rasmus Nielsen. "Linkage disequilibrium as a signature of selective sweeps". In: *Genetics* 167.3 (2004), pp. 1513–1524.

[29] Motoo Kimura. "The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations". In: *Genetics* 61.4 (1969), p. 893.

[30] Rick A Kittles et al. "Extent of linkage disequilibrium between the androgen receptor gene CAG and GGC repeats in human populations: implications for prostate cancer risk". In: *Human genetics* 109.3 (2001), pp. 253–261.

[31] Leonid Kruglyak. "Prospects for whole-genome linkage disequilibrium mapping of common disease genes". In: *Nature genetics* 22.2 (1999), pp. 139–144.

[32] David S Lawrie. "Accelerating Wright-Fisher forward simulations on the graphics processing unit". In: *bioRxiv* (2016), p. 042622.

[33] R. C. Lewontin. "THE INTERACTION OF SELECTION AND LINKAGE. I. GENERAL CONSIDERATIONS; HETEROTIC MODELS". In: *Genetics* 49.1 (1964), pp. 49–67. ISSN: 0016-6731. eprint: `http://www.genetics.org/content/49/1/49.full.pdf`. URL: `http://www.genetics.org/content/49/1/49`.

[34] RC Lewontin and Kenichi Kojima. "The evolutionary dynamics of complex polymorphisms". In: *Evolution* (1960), pp. 458–472.

[35] Isaac TS Li, Warren Shum, and Kevin Truong. "160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA)". In: *BMC bioinformatics* 8.1 (2007), p. 185.

[36] Scott Lloyd and Quinn O Snell. "Accelerated large-scale multiple sequence alignment". In: *BMC bioinformatics* 12.1 (2011), p. 466.

[37] Po-Ru Loh et al. "Inferring admixture histories of human populations using linkage disequilibrium". In: *Genetics* 193.4 (2013), pp. 1233–1254.

[38] Jiachun Lu et al. "Polymorphisms and haplotypes of the NBS1 gene are associated with risk of sporadic breast cancer in non-Hispanic white women< 55 years". In: *Carcinogenesis* 27.11 (2006), pp. 2209–2216.

[39] Atabak Mahram and Martin C Herbordt. "FMSA: FPGA-accelerated ClustalW-based multiple sequence alignment through pipelined prefiltering". In: *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE. 2012, pp. 177–183.

[40] Lisha A Mathew et al. "Why to account for finite sites in population genetic studies and how to do this with jaatha 2.0". In: *Ecology and evolution* 3.11 (2013), pp. 3647–3662.

[41] Grant Morahan et al. "Linkage disequilibrium of a type 1 diabetes susceptibility locus with a regulatory IL12B allele". In: *Nature genetics* 27.2 (2001), pp. 218–221.

[42] Rasmus Nielsen et al. "Genomic scans for selective sweeps using SNP data". In: *Genome research* 15.11 (2005), pp. 1566–1575.

[43] Pamela Orjuela-Sánchez et al. "Single-nucleotide polymorphism, linkage disequilibrium and geographic structure in the malaria parasite Plasmodium vivax: prospects for genome-wide association studies". In: *BMC genetics* 11.1 (2010), p. 65.

[44] Pavlos Pavlidis, Jeffrey D Jensen, and Wolfgang Stephan. "Searching for footprints of positive selection in whole-genome SNP data from nonequilibrium populations". In: *Genetics* 185.3 (2010), pp. 907–922.

[45] Bastian Pfeifer et al. "PopGenome: an efficient Swiss army knife for population genomic analyses in R". In: *Molecular biology and evolution* (2014), msu136.

[46] Gerd Pfeiffer et al. "A massively parallel architecture for bioinformatics". In: *International Conference on Computational Science*. Springer. 2009, pp. 994–1003.

[47] Shaun Purcell et al. "PLINK: a tool set for whole-genome association and population-based linkage analyses". In: *The American Journal of Human Genetics* 81.3 (2007), pp. 559–575.

[48] Michel Raymond and François Rousset. "GENEPOP (version 1.2): population genetics software for exact tests and ecumenicism". In: *Journal of heredity* 86.3 (1995), pp. 248–249.

[49] David E Reich et al. "Linkage disequilibrium in the human genome". In: *Nature* 411.6834 (2001), pp. 199–204.

[50] Rori V Rohlfs, Willie J Swanson, and Bruce S Weir. "Detecting coevolution through allelic association between physically unlinked loci". In: *The American Journal of Human Genetics* 86.5 (2010), pp. 674–685.

[51] Francois Rousset. "genepop'007: a complete re-implementation of the genepop software for Windows and Linux". In: *Molecular ecology resources* 8.1 (2008), pp. 103–106.

[52] Jiawei Shen, Zhiqiang Li, and Yongyong Shi. "SHEsisPCA: A GPU-Based Software to Correct for Population Stratification that Efficiently Accelerates the Process for Handling Genome-Wide Datasets". In: *Journal of Genetics and Genomics* 42.8 (2015), pp. 445–453.

[53] Montgomery Slatkin. "Linkage disequilibrium—understanding the evolutionary past and mapping the medical future". In: *Nature Reviews Genetics* 9.6 (2008), pp. 477–485.

[54] Xavier Solé et al. "SNPStats: a web tool for the analysis of association studies". In: *Bioinformatics* 22.15 (2006), pp. 1928–1929.

[55] Zachary D Stephens et al. "Big data: astronomical or genomical?" In: *PLoS Biol* 13.7 (2015), e1002195.

[56] Peter H Sudmant et al. "An integrated map of structural variation in 2,504 human genomes". In: *Nature* 526.7571 (2015), pp. 75–81.

[57] Joseph D Terwilliger and Kenneth M Weiss. "Linkage disequilibrium mapping of complex disease: fantasy or reality?" In: *Current Opinion in Biotechnology* 9.6 (1998), pp. 578–594.

[58] Sarah A Tishkoff et al. "Haplotype diversity and linkage disequilibrium at human G6PD: recent origin of alleles that confer malarial resistance". In: *Science* 293.5529 (2001), pp. 455–462.

[59] Peter M Visscher et al. "Five years of GWAS discovery". In: *The American Journal of Human Genetics* 90.1 (2012), pp. 7–24.

[60] Robin S Waples and Phillip R England. "Estimating contemporary effective population size on the basis of linkage disequilibrium in the face of migration". In: *Genetics* 189.2 (2011), pp. 633–644.

[61] Lars Wienbrandt et al. "FPGA-based acceleration of detecting statistical epistasis in GWAS". In: *Procedia Computer Science* 29 (2014), pp. 220–230.

[62] Fang Liu1 Jue Wang Xian-Yu, Lang1 Chi-Xue Bin Hai-Nan, and Zhao2 Jin-Sheng Lai. "Fast Computing of Linkage Disequilibrium on GPU". In: ().

[63] Hongyi Xin et al. "Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping". In: *Bioinformatics* (2015), btu856.

[64] Haiming Xu et al. "Detection of epistatic and gene-environment interactions underlying three quality traits in rice using high-throughput genome-wide data". In: *BioMed research international* 2015 (2015).

[65] Jian Yang et al. "GCTA: a tool for genome-wide complex trait analysis". In: *The American Journal of Human Genetics* 88.1 (2011), pp. 76–82.

[66] Jinho Yoo et al. "SNPAnalyzer 2.0: a web-based integrated workbench for linkage disequilibrium analysis and association analysis". In: *BMC bioinformatics* 9.1 (2008), p. 290.

[67] Chunbao Zhou et al. "gPGA: GPU Accelerated Population Genetics Analyses". In: *PloS one* 10.8 (2015), e0135028.

[68] Stephanie Zierke and Jason D Bakos. "FPGA acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods". In: *BMC bioinformatics* 11.1 (2010), p. 184.