2016 IEEE International Parallel and Distributed Processing Symposium Workshops

Efficient Computation of Linkage Disequilibria as Dense Linear Algebra Operations

Nikolaos Alachiotis, Thom Popovici, Tze Meng Low
Carnegie Mellon University
{*nalachio, lowt, dpopovic*}*@andrew.cmu.edu*

***Abstract*—Genomic datasets are steadily growing in size as more genomes are sequenced and new genetic variants are discovered. Datasets that comprise thousands of genomes and millions of single-nucleotide polymorphisms (SNPs), exhibit excessive computational demands that can lead to prohibitively long analyses, yielding the deployment of high-performance computational approaches a prerequisite for the thorough analysis of current and future large-scale datasets. In this work, we demonstrate that the computational kernel for calculating linkage disequilibria (LD) in genomes, i.e., the non-random associations between alleles at different loci, can be cast in terms of dense linear algebra (DLA) operations, leveraging the collective knowledge in the DLA community in developing high-performance implementations for various microprocessor architectures. The proposed approach for computing LD achieves between 84% and 95% of the theoretical peak performance of the machine, and is up to 17X faster than existing LD kernel implementations. Furthermore, we argue that, the current trend of increasing the SIMD (Single Instruction Multiple Data) register width in microprocessors yields minor benefits for assessing LD, resulting in an increasing gap between performance attainable by LD computations and the theoretical peak of the microprocessor architecture, suggesting the need for hardware support.**

***Keywords*-linkage disequilibrium; population genetics; dense linear algebra; matrix multiplication;**

I. Introduction

Linkage Disequilibrium (LD) [1] describes the non-random association between alleles at different loci. It is a statistical measure that quantifies the existence of associated alleles in a population when the detected allele associations differ from what one would expect if the alleles were inherited independently. Prior to conducting any LD computation, a typical workflow involves the preliminary steps of sequencing the genetic material of a set of individuals under investigation, and the mapping of the generated short reads to a reference genome to create a multiple-sequence alignment (MSA). Thereafter, a so-called SNP calling step identifies variable sites, such as single-nucleotide polymorphisms (SNPs). The computation of LD can only take place after a SNP map of the population under study has been generated, since alignment sites without mutations, i.e., monomorphic sites, are non-informative for LD.

LD has several applications in population genetics studies. The identification of coevolving interacting genes [2], for instance, relies on LD in order to detect regions that undergo complementary mutations that maintain the gene interaction. Searching for traces of positive selection in a population also relies on LD, since, according to selective sweep theory [3], low LD is found on different sides of a positively selected site, whereas high LD is expected across a positively selected site. In genome-wide association studies (GWAS), LD is deployed to identify SNPs associated with cetrain traits of interest, such as human diseases [4], paving the way for more effective and personalized drug treatments [5].

LD calculations entail the extraction of allele frequencies per SNP, and haplotype frequencies per pair of SNPs in a dataset. Thus, the computational demands increase linearly with an increasing sample size (number of individuals), and quadtratically with an increasing number of SNPs. As more and more genomes are sequenced, driven by the continuous advances in DNA sequencing technologies, genomic datasets suitable for population-based association studies grow both in sample size as well as number of SNPs. Thus, high-performance implementations are required to enable the efficient analysis of current and future large-scale datasets on modern microprocessor architectures, as well as to facilitate more thorough LD-based analyses.

In this work, we make the observation that LD computations can be cast in terms of a series of dense linear algebra (DLA)-like operations, which are well-studied by the high performance computing (HPC) community. Casting LD computations in terms of DLA operations allows the bioinformatics and computational biology fields to leverage the collective knowledge in the HPC community to develop high-performance LD implementations for various architectures. We describe LD computations as a series of Basic Linear Algebra Subprograms [6], [7], [8] (BLAS) operations, and show how high-performance LD can be efficiently implemented using HPC techniques.

To facilitate the required allele and haplotype frequency computations, alleles are typically encoded as one- or two-bit entities, based on the assumption of the–widely adopted–infinite sites model [9], or a finite site model (e.g., the general time reversible model of DNA substitution [10], [11]). The performance bottleneck for computing allele/haplotype frequencies is the enumeration of states in SNPs and pairs of states in SNP pairs, which heavily relies on population

/16 $31.00 © 2016 IEEE
DOI 10.1109/IPDPSW.2016.80

count operations, i.e., counting of set bits in a word. While the current trend in microprocessor design is to increase the SIMD (Single Instruction Multiple Data) register width, allowing optimized software codes to exploit this increased computational capacity via the use of vector intrinsics, modern microprocessors do not currently exhibit a vectorized population count operator. Consequently, applications that rely on population count operations do not benefit from the increased computational capacity in microprocessors. We provide an analytical argument that increasing SIMD width yields little benefit for LD, indicating the need for a vectorized population count operator.

The remainder of this paper is organized as follows. In Section II, we describe the mathematical operations and concepts required to compute LD as a series of DLA operations. Section III outlines the GotoBLAS approach for high-performance matrix multiplication, as can be applied for computing LD. Thereafter, we provide implementation details and performance of the required LD micro-kernel (Section IV), and highlight the need for a vectorized popcount operator (Section V). We provide a performance comparison with existing LD implementations in Section VI, and discuss current limitations and future extensions of our approach in Section VII. Finally, we review related work in Section VIII, and conclude in Section IX.

II. Linkage Disequilibrium (LD)

Linkage disequilibirum (LD) essentially is a test to determine if mutations in different SNPs are independent. Two SNPs (s_i and s_j) are said to be in linkage disequilibrium if the mutations in the two SNPs are not independent. Mathematically, two events (A and B) are independent if and only if

$$P(AB) = P(A)P(B),$$

where $P(A)$ and $P(B)$ are the probabilities that event A and event B occur, and $P(AB)$ is the joint probability of both events occuring. Alternatively, events A and B are not independent if the value D, given by

$$D = P(AB) - P(A)P(B), \tag{1}$$

is statistically different than zero.

Given SNPs s_i and s_j, the LD between these SNPs can be computed using Equation 1, where events A and B are the probabilities that a mutation has occured at locations i and j, respectively. In other words, $P(AB)$ is the haplotype frequency, whereas $P(A)$ and $P(B)$ are the allele frequencies. When $D = 0$, s_i and s_j are in linkage equilibrium, i.e., mutations in s_i and s_j occur independently of each other, whereas $D \neq 0$ suggests that the two SNPs are in linkage disequilibrium.

It should be noted that this formulation of LD is not commonly used. A more commonly used measure of LD is the squared Pearson coefficient

$$\begin{aligned} r^2 &= \frac{(P(AB) - P(A)P(B))^2}{P(A)P(B)(1-P(A))(1-P(B))} \\ &= \frac{D^2}{P(A)P(B)(1-P(A))(1-P(B))}, \end{aligned} \tag{2}$$

which has the advantage that all r^2 values remain between 0.0 and 1.0.

A. Allele and Haplotype Frequencies

Population genetics studies typically adopt the infinite sites model (ISM) [9], which assumes that an infinite number of sites exist, thus every new mutation always occurs at a site where no mutation has previously occured. As a consequence, the number of allelic states per SNP is limited to two, i.e., the ancestral state (before the mutation) and the derived state (after the mutation). In this paper, we use 0 and 1 to represent the ancestral and derived states, respectively. The limited number of states in ISM allows us to reformulate the computation of allele and haplotype frequencies using linear algebra, which allows to eventually leverage the collective knowledge of the linear algebra community to produce a high-performance LD implementation.

Given a sample size of N_{seq} sequences, the allele frequency of SNP s_i (P_i) can be computed by counting the number of derived states (mutations) in s_i and dividing that by N_{seq}. This can easily be computed using a linear algebra operation as follows:

$$P_i = \frac{s_i^T s_i}{N_{seq}}. \tag{3}$$

Note that, because s_i contains only ones and zeros, the inner/dot product ($s_i^T s_i$) computes the number of ones in s_i, which is exactly the number of mutations in the SNP.

Similarly, the haplotype frequency $P_{i,j}$ in a pair of SNPs, s_i and s_j, can be obtained by computing the inner product first and then dividing the result by the sample size N_{seq}, as described below:

$$P_{i,j} = \frac{s_i^T s_j}{N_{seq}}. \tag{4}$$

Using Equations 1, 3, and 4, the LD value, $D_{i,j}$, between SNPs s_i and s_j is given by:

$$\begin{aligned} D_{i,j} &= P_{i,j} - P_i P_j \\ &= \frac{1}{N_{seq}}(s_i^T s_j) - \frac{1}{N_{seq}^2}(s_i^T s_i)(s_j^T s_j). \end{aligned} \tag{5}$$

B. Computing LD as Matrix Multiplication

In practice, we want to compute $D_{i,j}$ for all possible pairs of SNPs, s_i and s_j, in a region of n SNPs. Thus, a simple way of computing this is with the following pseudocode:

```
for (i = 0 to n-1)
  for (j = 0 to n-1)
     compute D_{i,j}
```

The problem with this approach is that in the above formulation of $D_{i,j}$, each SNP is treated as a column vector, and the required computations for all LD values are cast in terms of vector operations. This approach is highly inefficient, which is why the dense linear algebra community has developed the Level 3 Basic Linear Algebra Subprograms (BLAS3) [8] operations, which are essentially matrix multiplications of different forms, that are more efficient on modern day processors with hierarchy of caches for memory. Similarly, we note that every dataset that comprises more than one SNPs can be regarded as a genomic matrix G, where each column in the matrix is a SNP. Therefore, it would be beneficial to reformulate LD computations in a way that maximizes the amount of operations that can be cast as matrix multiplication. Reformulating LD computations in terms of matrix multiplication yields the following sequence of dense linear algebra operations:

$$\begin{aligned} H &= \tfrac{1}{N_{seq}} G^T G \\ D &= H - pp^T, \end{aligned}$$

where the first operation computes all the haplotype frequencies, $P_{i,j}$, and stores them in a matrix H, while the second operation subtracts the product of the allele frequencies from H. Here, p is a vector of the per-SNP allele frequencies.

Using this formulation, it becomes obvious that computing the matrix of haplotype frequencies (H) is an $O(n^3)$ operation as it is simply a matrix multiplication. In addition, the subtraction of the allele frequencies is an $O(n^2)$ operation as it is an outer product of two vectors. Therefore, as the number of SNPs gets larger, the cost of computing H dominates the overall required computations, and thus efforts to optimize LD computations should focus on optimizing the computation of H. Hence, the rest of the paper will focus on optimizing the computation of the haplotype frequency matrix H.

III. High Performance LD as GEMM

Using the BLAS naming convention, the computation of the haplotype frequency matrix, H, is in essense the general matrix multiplication operation (GEMM). In this section, we show that a high-performance scalar implementation of the LD computation can indeed be implemented using the GotoBLAS [12] approach for high-performance GEMM.

A. GotoBLAS Approach to GEMM

The GotoBLAS approach (now maintained as OpenBLAS [13]) is a widely adopted approach for implementing high-performance Level 3 BLAS operations on modern architectures with a cache-based memory hierarchy. At the heart of the GotoBLAS approach is a highly optimized GEMM kernel of a particular shape and implementation. For completeness, we provide an overview of the GotoBLAS approach, but the interested reader is recommended to review [12].

Given matrices A, B, and C, the GEMM operation computes

$$C = \alpha AB + \beta C,$$

where A, B, and C are of dimensions $m \times k$, $k \times n$, and $m \times n$, respectively. Using the GotoBLAS approach, the input matrices are partitioned in the k dimension, exposing a loop around a smaller GEMM operation whose inputs are of sizes $m \times k_c$, $k_c \times n$, and $m \times n$, where $m, n \gg k_c$. This smaller GEMM operation, also known as the rank-k update, is the GEMM kernel in the GotoBLAS approach[1]. Using the GotoBLAS approach, high-performance GEMM implementations are obtained when the GEMM kernel is implemented efficiently for the given machine architecture. In particular, a highly efficient GEMM kernel can usually yield GEMM performance that is close to the peak performance of the machine ($\approx 90\%$).

The GEMM kernel is implemented in a carefully layered manner where each matrix is further partitioned in a particular manner to maximize data reuse in all cache levels. In particular, each GEMM kernel is implemented as a series of block-panel multiplications. The inputs to each block-panel multiplication are then packed into contiguous memory, and the resulting matrix multiplication is implemented as a blocked-dot product. Pictorially, this layering of the different matrix multiplications required to compute the GEMM kernel is shown in Figure 1, where the bottom layer shows the GEMM kernel matrix being partitioned into a series of blocked-panel matrix multiplications (denoted by the lighter shades). The layers above the bottom layer show how each resulting matrix multiplication at the different layers is implemented in terms of simpler and smaller blocked matrix multiplication operations.

B. A "Future-proof" Approach

Apart from being a highly efficient approach to implementing the GEMM operation, the GotoBLAS approach is well-suited for computing LD, now and in the future. Typically, the number of SNPs is larger than the number of available samples/sequences. This means that the genomic matrix G has a dimension of $k \times n$ where $n \gg k$. In addition, the operation for computing the haplotype frequency matrix, H, is

$$H = \frac{1}{N_{seq}} G^T G.$$

As $m = n \gg k$, the computation of the haplotype frequency matrix H is already a rank-k update, which means that the input and output matrices are already of the shapes optimized by the GotoBLAS approach. Hence, the haplotype

[1]Larger input matrices are first partitioned in the m and n dimensions, before being partitioned to expose the rank-k updates.

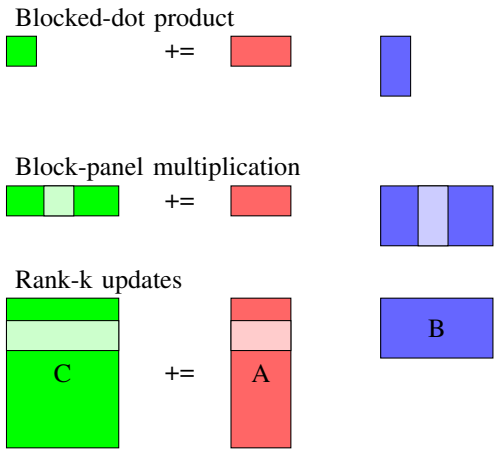

Figure 1. GotoBLAS layered approach to implementing a GEMM (rank-k) kernel on cache-based architectures. General matrix dimensions are first partitioned into rank-k kernels (bottom layer), which in turn are implemented as block-panel matrix multiplications (middle layer). These block-panel matrix multiplications are then implemented as blocked-dot products (top layer).

frequency computation is already optimized with a highly optimized GEMM kernel.

In addition, as DNA sequencing technology improves, the number of sequences in genomic datasets increases. This implies that the k dimension of the genomic matrix, representing the number of samples, is increasing. Recall that the GotoBLAS approach partitions the input matrices in the k dimension. Thus, increasing the k dimension of the genomic matrix requires no change to the implemented algorithm. More importantly, the GotoBLAS approach is already optimized for this increase in the k dimension. Thus the GotoBLAS approach can be considered a "future"-proof approach for computing LD.

IV. A Scalar LD Micro-kernel

Apart from leveraging the GotoBLAS approach, we utilized the BLAS-Like Instantiation Software (BLIS) [14], a framework for rapidly implementing DLA operations using the GotoBLAS approach. The BLIS framework represents the accumulated knowledge of how DLA libraries are built using the GotoBLAS approach. Using the BLIS framework, the developer only needs to implement a highly efficient micro-kernel, a much smaller GEMM operation than the GotoBLAS GEMM kernel, in order to obtain a high-performance GEMM implementation. Furthermore, employing this framework allows to leverage existing efficient parallelization schemes.

A. The LD Micro-kernel in BLIS

At the heart of the BLIS framework is a highly efficient micro-kernel that computes

$$C = \alpha AB + \beta C,$$

where the dimensions of the matrices A, B, and C, are $m_r \times k_c$, $k_c \times n_r$, and $m_r \times n_r$, respectively, and $k_c \gg m_r, n_r$[2]. A highly tuned micro-kernel will yield a highly efficient GEMM implementation. The interested reader is pointed to [15] for details on how a BLIS micro-kernel is implemented and the performance achievable using the BLIS framework. In the rest of this section, we describe the changes required to implement the LD micro-kernel within the BLIS framework. No attempt was made to tune the parameters within BLIS to obtain an optimized LD kernel.

A key difference in implementing a micro-kernel for LD is that the LD micro-kernel operates on binary data. Recall that our genomic matrix is a binary matrix, whose elements can be effectively stored by using one bit per element. However, the use of a binary matrix is not supported by BLIS or BLAS in general. However, note that the size of a double-precision floating-point value is the same as the size of an unsigned long integer (64 bits). This implies that, as long as the same 64 bits in memory are loaded into the registers, we can interpret them either as unsigned long integers or as double-precision floating-point numbers. Therefore, the following steps are performed:

- The genomic matrix G is stored as an array of unsigned long integers.
- The pointer to the genomic matrix G is typecast into a pointer to a double-precision floating-point matrix, before being passed to the BLIS framework.
- Within the micro-kernel, the pointers to the input matrices are recast back into pointers to unsigned long integer matrices, thus allowing to correctly retrieve the binary data.

The net result is that the genomic matrix G is stored in the same storage scheme as described in [16], where each SNP is stored as consecutive unsigned long integers. When the number of sequences is not a multiple of 64, each SNP is padded with zeros to make the number of sequences a multiple of 64. A pictorial description of the data layout introduced in [16] is shown in Figure 2.

A second difference is that computing with binary data simplifies the computations that need to be performed. Recall that each haplotype frequency, $P_{i,j}$, is computed as follows:

$$P_{i,j} = \frac{s_i^T s_j}{N_{seq}}.$$

Since the elements in SNPs s_i and s_j are binary, the multiplication of each pair of elements from s_i and s_j will be one if and only if both elements are ones. Hence, the multiplication of the elements of s_i and s_j can be simplified to performing an `and` operation (&). Similarly, the addition operation for binary data is a bit-counting/population count

[2]The subscripts r and c stand for registers and cache, respectively.

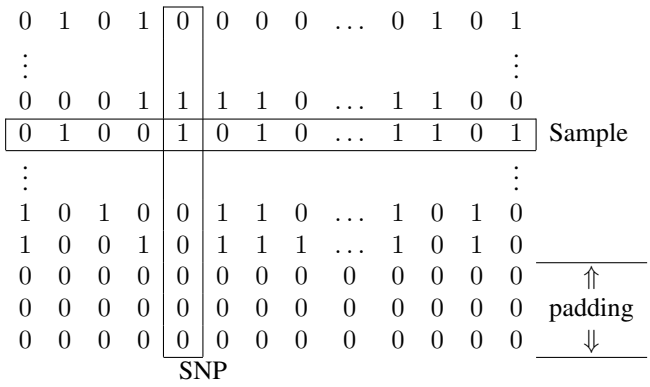

Figure 2. A genomic matrix G. Each row represents a sample, while every column represents a SNP (source: [16]).

(POPCNT) operation. Therefore, the computation of $P_{i,j}$ can be performed in the following manner:

$$P_{i,j} = \frac{1}{N_{seq}} \texttt{POPCNT}(s_i \ \& \ s_j).$$

However, recall that within the micro-kernel, binary data is being retrieved as unsigned long integers of 64 bits. Hence, the computation of $P_{i,j}$ in the micro-kernel is given by

$$P_{i,j} = \frac{1}{N_{seq}} \sum_{k=0}^{N_{int}} \texttt{POPCNT}(s_i^k \& s_j^k),$$

where N_{int} is the number of unsigned long integers required to store a single SNP vector, and s_x^k is the k^{th} unsigned long integer of SNP s_x.

On modern x86 architectures (2007 onwards), the population count operation can be effectively implemented with the intrinsic POPCNT instruction[3]. There are different efforts in implementing efficient population counters in software, and a summary of these methods can be found in [17]. However, these software implementations have been shown to attain lower performance than simply using the POPCNT instruction [18]. As such, we opted to use the POPCNT instruction in our implementation of the micro-kernel.

B. A Measure of Performance

In order to determine if an implementation of LD is high performance, a metric for determining the theoretical peak performance for computing the haplotype frequency matrix D is required. Using execution time and/or number of LD values computed per second is not a good measure as both measures are highly dependent on the shape and size of the input genomic matrices. We, instead, examine how the HPC community determines the theoretical peak for GEMM.

[3]The POPCNT instruction comes in a 32-bit and a 64-bit variant. The 64-bit variant was chosen as it reduces the number of operations required to compute the haplotype frequency.

GEMM is computed using multiplications and additions. In each clock cycle, v multiplications and v addition instructions can be issued in parallel. On machines with SIMD or vector registers, v is typically the number of elements in each SIMD/vector register. Therefore, a total of $v + v = 2v$ floating-point operations (FLOP) required for computing GEMM can be performed in each cycle. Hence, the theoretical peak of GEMM is $2v$ FLOP per clock cycle.

To determine the theoretical peak of LD computations, note that three separate operations are required. Initially, the haplotypes are constructed using an and operation to identify samples that exhibit mutations in SNPs s_i and s_j. Thereafter, the number of such samples is counted using the POPCNT intrinsic operation, and the total number of samples is computed via an accumulation (add operator). On current x86 architectures, only one POPCNT instruction, computing the number of set bits in 64 bits, can be issued in each cycle, i.e., $v = 1$. However, all three (and, POPCNT, and add) instructions can be issued in the same clock cycle. This implies that for these architectures, the theoretical peak for computing LD is three operations per clock cycle. Note that these three operations are scalar operations (i.e., $v = 1$).

C. Performance

In Figure 3, we report the performance of our haplotype frequency implementation with scalar instructions for different sizes of the haplotype frequency matrix H, where $m = n$, on an Intel Haswell architecture, 3.5 GHz machine. In each test, we measured the performance attained as a percentage of the theoretical peak as we varied the k dimension of the genomic matrix. The top of the y-axis represents the theoretical peak of the scalar instructions of 3 operations per clock cycle. Measuring the performance as the k dimension increases, which represents increasing sample size, is important because it simulates the effect of improvements in DNA sequencing technologies that lead to more and more genomes being sequenced.

In all these tests, even though the parameters used were optimized for double-precision GEMM, our scalar implementation achieves between 84% and 90% of the theoretical peak performance as the k dimension increases. This variation is expected, and is also observed in many regular GEMM implementations. The variation can be attributed to the size of the matrix being not a multiple of the different cache sizes. When the matrix is not a multiple of the cache sizes, computations need to be performed on data that do not fit in cache. As such, these data has to be brought into the cache before computation can proceed. As the matrix size increases, more of the cache is filled. This translates to increased performance. It should be noted that, this performance variation could be smoothen, but is not considered in this implementation.

It is important to note that even as the k dimension increases, the performance attained by our implementations

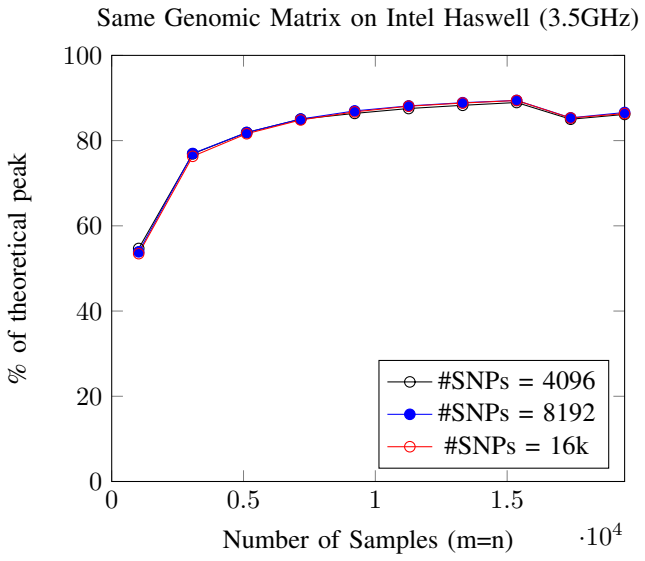

Figure 3. Performance attained on Intel Haswell, 3.5GHz with scalar implementation.

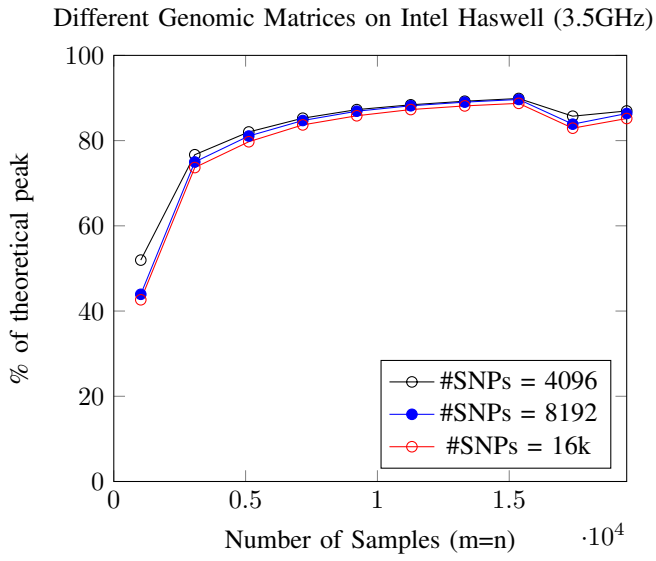

Figure 4. Performance attained on Intel Haswell, 3.5GHz when computing with two different genomic matrices.

remains relatively steady, thereby demonstrating that an implementation of LD using the GotoBLAS approach remains a high-performance algorithm even as the number of samples increases. Furthermore, we show that the algorithm is agnostic to the number of SNPs being computed. As we increase the number of SNPs in the genomic matrix from 4096 to 16384, the peak performance attained still remains between 84% and 90% of the theoretical peak. Even though increasing the number of samples could potentially increase the number of SNPs identified, the performance of our computation remains relatively stable across increased number of SNPs.

Similar performance (Figure 4) is observed when two different genomic matrices are used as inputs. In this test, the haplotype frequencies between the SNPs in two different genomic matrices of sizes $m \times k$ and $k \times n$ are computed. The difference between this set of tests and the previous results shown in Figure 3 is that all $m \times n$ haplotype frequencies must be computed. Notice that, despite having to compute approximately twice as many output values, the attained performance from our implementation remains consistent between 84% and 90%. More importantly, this demonstrates that our proposed GEMM-based LD computational approach can be deployed for association studies between distant genes, as well as long-range LD calculations.

V. In Need For SIMD Hardware Support

Given that we can achieve good scalar performance with no modifications to the GotoBLAS approach, one would not be remissed if one assumes that almost linear speed up is achievable by switching to SIMD instructions, such as 128-bit-wide SSE (Streaming SIMD Extensions) or 256-bit-wide AVX (Advanced Vector Extensions). This is because high-performance matrix multiplication invariantly is implemented with SIMD instructions in order to attain near peak performance. In addition, the current trend in hardware architecture is to increase the length of SIMD instructions allowing more computations to be performed in parallel[4]. Recall that, on current x86 architectures, the `POPCNT` instruction enumerates the number of set bits in a chunk of 32 or 64 bits, which means that `POPCNT` is a scalar operation. For a SIMD register containing v scalar elements, each of the v scalar values must be extracted from the SIMD register before the `POPCNT` operation is performed on the extracted value. In addition, after all v `POPCNT` operations have been performed, all v results must be stored back into a SIMD register before a parallel SIMD addition is performed for the accumulation step. We analyze the changes to performance in switching to SIMD instructions in the subsequent section.

A. Analysis of SIMD Benefit for LD

Recall that in computing the theoretical peak for the construction of the haplotype frequency matrix, we assumed that all three scalar operations (`and`, `POPCNT`, and `add`) can be issued in parallel in one clock cycle. For simplicity, we also assume that every instruction takes one clock cycle.

Mathematically, we can represent the time required to compute the haplotype frequency matrix as

$$\begin{aligned} T &= mn(\max(T_{\texttt{add}}, T_{\texttt{and}}, T_{\texttt{POPCNT}})) \\ &= mnT_{\texttt{add}} = mnT_{\texttt{and}} = mnT_{\texttt{POPCNT}}. \end{aligned}$$

[4]512-bit SIMD instructions are already being introduced in the latest architecture.

With the use of SIMD instructions, $T_{\texttt{add}}$ and $T_{\texttt{and}}$ are reduced to $T_{\texttt{add}}/v$ and $T_{\texttt{and}}/v$, respectively, where v is the number of elements in the SIMD register. However, because POPCNT is sequential, there is no change to $T_{\texttt{POPCNT}}$. Thus the time for computing the haplotype frequency matrix is:

$$\begin{aligned} T_{SIMD} &= mn(\max(\tfrac{T_{\texttt{add}}}{v}, \tfrac{T_{\texttt{and}}}{v}, T_{\texttt{POPCNT}})) \\ &= mnT_{\texttt{POPCNT}}, \end{aligned}$$

which suggests no benefit for using SIMD instructions under the best scenario.

However, recall that extraction and insertion operations have to be performed in order to exploit the POPCNT instruction. If extractions and insertions can be performed in parallel with all three instructions, then the time to compute the haplotype frequency matrix is $mnT_{\texttt{POPCNT}}$. In practice, extractions and insertions cannot be performed in parallel as they require the same hardware resources. This means that when an insertion has to happen, no extraction instruction can be executed. In turn, because there is a stall in the extraction of values from the SIMD registers, there is a stall in the execution of the POPCNT instruction. Hence, $T_{\texttt{POPCNT}}$ will increase. Thus, there potentially could be a decrease in performance in moving to SIMD instructions for LD.

B. A Need for Vectorized POPCNT

Assuming that a vectorized POPCNT implementation in hardware is available, this implies that all three instructions required to calculate LD can be parallelized, and the expected time taken is

$$\begin{aligned} T_{HW} &= mn(\max(\tfrac{T_{\texttt{add}}}{v}, \tfrac{T_{\texttt{and}}}{v}, \tfrac{T_{\texttt{POPCNT}}}{v})) \\ &= mn\tfrac{T_{\texttt{add}}}{v} = mn\tfrac{T_{\texttt{and}}}{v} = mn\tfrac{T_{\texttt{POPCNT}}}{v}. \end{aligned}$$

The difference between T_{SIMD} and T_{HW} is that a vectorized POPCNT instruction eliminates the need for extractions and insertions, and more importantly parallelizes the sequential POPCNT instruction. This eliminates any sequential segments of code in the computation of the haplotype frequency matrix, thus parallelizing the entire operation.

VI. Performance Comparison

To compare the performance of our proposed approach with existing LD implementations, we consider two optimized software codes, PLINK 1.9 [19] (a performance update to the widely used PLINK software [20] for whole-genome association and population-based linkage analyses), and OmegaPlus [21], [22] (a high-performance implementation[5] of the ω statistic [23] for selective sweep detection that relies on LD). We used a workstation with two Intel Xeon E5-2620 v2 (Ivy Bridge) 6-core processors (running at 2.10 GHz, 128 GBs main memory), as a test platform.

[5]We further improved performance by employing the same 64-bit popcount intrinsic instruction used in the GEMM-based LD implementation.

We conduct performance comparisons in terms of execution time and number of LD values per second. While execution time/number of LDs per second is not a good measure of performance (see Section IV-B), we unavoidably conduct such comparisons here to eliminate the risk of miscalculating the theoretical peak performance for the other software, as it requires low-level understanding not only of the employed algorithms, but also the implementation itself. We consider three datasets, A, B, and C, with same number of SNPs (10,000) and varying sample sizes. Dataset A represents a small subset of variants from the first chromosome of the human genome (available from the 1000 Genomes project, *http://www.1000genomes.org*), with a sample size of 2,504 sequences. Datasets B and C are simulated and comprise 10,000 and 100,000 sequences, respectively.

Due to different scope/concerns of the three LD implementations under comparison (PLINK 1.9 vs OmegaPlus vs GEMM-based LD), it is important to note the following: i) While all three assume the ISM, the focus of PLINK 1.9 is on genotypes, whereas the focus of OmegaPlus and GEMM is on alleles. ii) PLINK 1.9 and GEMM compute all $N(N+1)/2$ LD values in a region of N SNPs, whereas OmegaPlus computes only the LD values required for the ω statistic calculations. For this reason, PLINK 1.9 and GEMM conducted 50M pairwise LD calculations for all three datasets, whereas OmegaPlus conducted 49.4M pairwise LD calculations for Dataset A, and 49.9M LD calculations for Datasets B and C.

Tables I, II, and III provide the results of the comparisons for the three datasets. As can be observed, the GEMM-based LD approach outperforms both the PLINK 1.9 and OmegaPlus implementations for all dataset sizes and different numbers of threads. Figure 5 illustrates a performance comparison based on Dataset C (10,000 SNPs and 100,000 sequences) as the number of threads increases until no more performance gains are observed by any of the implementations, which extends beyond the number of physical cores on the test platform (12 cores). As can be observed, the GEMM-based performance immediately diminishes when the number of threads is higher than 12, because each thread is already achieving near peak core performance, whereas both OmegaPlus and PLINK 1.9 performances improve further, suggesting the underutilization of each core when a small number of threads is launched.

VII. Discussion

It should be pointed out that in this work we focus solely on computing LD under the assumption of the infinite sites model. From a computational standpoint, this requires the simplest LD kernel implementation, since a single bit is sufficient to accurately represent each allelic state in a SNP. The computational demands and complexity of the LD kernel will increase when a finite sites model is assumed, or alignment gaps and ambiguous characters are considered.

Table I
PERFORMANCE COMPARISON BASED ON THE TEST DATASET THAT COMPRISES 10,000 SNPS FROM THE GENOMES OF 2,504 HUMANS.

Threads	Execution time (seconds)			LDs per second ($\times 10^6$)			GEMM Speedup (X) vs	
	PLINK 1.9	OmegaPlus	GEMM	PLINK 1.9	OmegaPlus	GEMM	PLINK 1.9	OmegaPlus
1	14.18	7.04	1.89	3.52	7.10	26.36	7.48	3.71
2	12.02	6.72	1.36	4.15	7.43	36.75	8.85	4.94
4	8.21	6.02	1.11	6.09	8.29	44.87	7.36	5.41
8	5.88	4.56	0.73	8.49	10.94	68.34	8.05	6.25
12	5.29	4.21	0.62	9.44	11.85	79.58	8.43	6.72

Table II
PERFORMANCE COMPARISON BASED ON A SIMULATED DATASET THAT COMPRISES 10,000 SNPS AND 10,000 SEQUENCES.

Threads	Execution time (seconds)			LDs per second ($\times 10^6$)			GEMM Speedup (X) vs	
	PLINK 1.9	OmegaPlus	GEMM	PLINK 1.9	OmegaPlus	GEMM	PLINK 1.9	OmegaPlus
1	49.20	23.71	5.36	1.01	2.10	9.31	9.22	4.43
2	39.11	14.32	3.16	1.27	3.49	15.81	12.45	4.53
4	23.98	7.79	2.01	2.08	6.41	24.85	11.94	3.87
8	13.60	5.34	1.44	3.67	9.35	34.64	9.44	3.70
12	9.78	4.67	1.17	5.11	10.70	42.37	8.29	3.96

Table III
PERFORMANCE COMPARISON BASED ON A SIMULATED DATASET THAT COMPRISES 10,000 SNPS AND 100,000 SEQUENCES.

Threads	Execution time (seconds)			LDs per second ($\times 10^6$)			GEMM Speedup (X) vs	
	PLINK 1.9	OmegaPlus	GEMM	PLINK 1.9	OmegaPlus	GEMM	PLINK 1.9	OmegaPlus
1	465.99	222.54	48.09	0.10	0.22	1.03	10.3	4.68
2	364.96	114.50	25.07	0.13	0.43	1.99	15.31	4.63
4	210.64	60.31	13.54	0.23	0.82	3.69	16.04	4.50
8	120.81	31.08	7.37	0.41	1.60	6.78	16.54	4.24
12	88.37	20.95	5.21	0.56	2.38	9.59	17.13	4.01

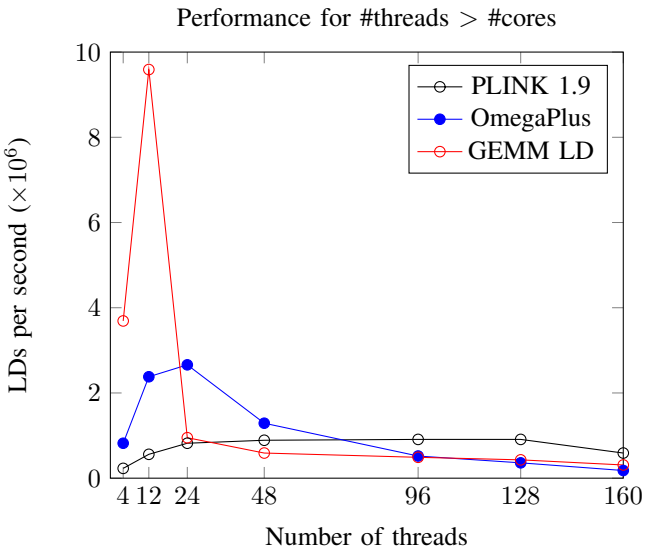

Figure 5. Performance comparison based on Dataset C (10,000 SNPs, 100,000 sequences) as the number of threads increases beyond the number of physical cores (12 cores on the test platform).

Considering alignment gaps: Gaps in a MSA can appear due to missing data [24], or can be artifacts of the employed scoring scheme for the construction of the MSA. The proposed framework can be adapted to compute LD between SNPs with alignment gaps by treating them as invalid states. This requires an additional bit vector per SNP, denoted by c, to indicate the existence of valid allelic states in the SNP bit-vector s. For every pair of SNPs i and j, $c_{ij} = c_i \& c_j$ describes all valid pairs of states. Therefore, the inner products for the allele frequencies can be computed as follows:

$$\begin{aligned}(c_{ij}\&s_i)^T(c_{ij}\&s_i) &= \texttt{POPCNT}(c_{ij}\&s_i) \\ &= \sum_{k=0}^{N_{int}} \texttt{POPCNT}(c_{ij}^k\&s_i^k)\end{aligned}$$

for both s_i and s_j, whereas the inner product for the haplotype frequency can be computed as:

$$\begin{aligned}(c_{ij}\&s_i)^T(c_{ij}\&s_j) &= \texttt{POPCNT}(c_{ij}\&s_i\&s_j) \\ &= \sum_{k=0}^{N_{int}} \texttt{POPCNT}(c_{ij}^k\&s_i^k\&s_j^k).\end{aligned}$$

Facilitating finite sites models: Due to the wide acceptance of the infinite sites assumption by population genetists,

our current work already covers a broad range of real-world use-case scenarios. However, the proposed framework can be adapted for other LD use cases that are being increasingly deployed lately, such as finite sites models [25]. As already explained, under the FSM assumption, the number of sites is no longer infinite, which translates to more than two possible states per SNP, requiring a minimum of 2 bits per allelic state per SNP, or more. DNA states (A for adenine, C for cytosine, G for gouanine, and T for thymine) can be encoded by as few as 2 bits, but 4 bits are typically allocated per state to account for ambiguous characters and alignment gaps. Note that, ambiguous DNA characters may appear in the short reads due to sequencing errors and/or insufficient base miscall correction.To facilitate LD computations under the FSM, each SNP is now represented by 4 bit vectors, one for every nucleotide state. Thus, a coefficient-based statistic for LD, denoted T_{ij}, is computed as follows, according to [26]:

$$T_{ij} = \frac{(v_i - 1)(v_j - 1)v_{ij}}{v_i v_j} \sum_{s_i, s_j \in S} r^2_{s_i s_j}, \qquad (6)$$

where $S : \{A, C, G, T\}$, v_i is the number of existing states in SNP i ($v_i \leq 4$), v_j is the number of existing states in SNP j ($v_j \leq 4$), v_{ij} is the number of valid pairs of states (no alignment gaps), and $r^2_{s_i s_j}$ is given by Equation 2. Evidently, the worst case of computing LD under the FSM requires 16 times more computations than the ISM, due to the 4 valid DNA states in each SNP.

Adapting for other domains: The representation of objects by binary vectors to facilitate computations is common in various fields. For instance, in chemical informatics, compounds are represented by binary vectors (typically referred to as 2D fingerprints), which are generated by subgraph isomorphism algorithms that examine atoms and bonds, and set a bit for each different pattern. Given two compounds A and B, with p, q, and x being the numbers of set bits in A, B, and $A \cap B$, respectively, the similarity between A and B can be calculated by several measures, with the most commonly employed being the Tanimoto coefficient [18]:

$$Tanimoto_{AB} = \frac{x}{p + q - x}. \qquad (7)$$

From a computational standpoint, computing the Tanimoto coefficient between compounds represented by 2D fingerprints is similar to computing LD between SNPs under the ISM. Therefore, our approach can be adapted for domain-specific similarity calculations, as long as the objects under comparison can be represented by binary vectors.

VIII. Related Work

Several software tools or packages to compute LD have been released [27], [19]. These implementations either rely on scalar kernels that are not optimized for performance [27] or on vector intrinsics [19] that, as already demonstrated, exhibit poor performance for population count operations. Thus, it is essential to explore how the proposed framework can be exploited to boost performance of existing software tools for LD, or be adapted for more specialized use-cases such as higher-order LD [28] or for selective sweep detection [21].

IX. Conclusion

In this paper, we show that efficient linkage disequilibrium computations are in fact DLA operations in disguise. By employing the GotoBLAS and BLIS approaches from HPC, an efficient LD implementation can be attained. Additionally, the benefit of relying on the GotoBLAS approach for computing LD is that the approach already is future-proof with respect to advances in DNA sequencing technologies that lead to increasing genomic dataset sizes. We also show analytically that the lack of a SIMD popcount instruction severely reduces the performance attainable by a SIMD implementation of LD, resulting in a diverging gap between what needs to be computed, and what can be computed efficiently.

As future work, we intend to explore GPUs for the acceleration of LD calculations. In theory, LD performance can be significantly improved by exploiting the high memory bandwidth that current GPUs offer, since, like matrix multiplication, LD computations are memory-bound. The data access pattern suggests that LD is well-suited for current SIMT (Single Instruction Multiple Threads) GPU architectures. It remains to explore, however, whether the underlying LD arithmetics can be efficiently handled by the ALUs (Arithmetic and Logic Units) on the GPUs.

Acknowledgements

The authors thank Pavlos Pavlidis (FORTH, Greece) for comments on the manuscript. This work was sponsored by the DARPA BRASS program under agreement FA8750-16-2-003. The content, views and conclusions presented in this document do not necessarily reflect the position or the policy of DARPA or the U.S. Government. No official endorsement should be inferred.

References

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

[2] R. V. Rohlfs, W. J. Swanson, and B. S. Weir, "Detecting coevolution through allelic association between physically unlinked loci," *The American Journal of Human Genetics*, vol. 86, no. 5, pp. 674–685, 2010.

[3] J. Maynard Smith and J. Haigh, "The hitch-hiking effect of a favourable gene." *Genetical research*, vol. 23, no. 1, pp. 23–35, Feb. 1974. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/4407212

[4] D. E. Reich, M. Cargill, S. Bolk, J. Ireland, P. C. Sabeti, D. J. Richter, T. Lavery, R. Kouyoumjian, S. F. Farhadian, R. Ward *et al.*, "Linkage disequilibrium in the human genome," *Nature*, vol. 411, no. 6834, pp. 199–204, 2001.

[5] M. T. Alam, D. K. de Souza, S. Vinayak, S. M. Griffing, A. C. Poe, N. O. Duah, A. Ghansah, K. Asamoa, L. Slutsker, M. D. Wilson, J. W. Barnwell, V. Udhayakumar, and K. A. Koram, "Selective sweeps and genetic lineages of Plasmodium falciparum drug -resistant alleles in Ghana." *The Journal of infectious diseases*, vol. 203, no. 2, pp. 220–7, Jan. 2011. [Online]. Available: http://jid.oxfordjournals.org/content/203/2/220.long

[6] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for fortran usage," *ACM Transactions on Mathematical Software (TOMS)*, vol. 5, no. 3, pp. 308–323, 1979.

[7] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, "An extended set of FORTRAN basic linear algebra subprograms," vol. 14, no. 1, pp. 1–17, Mar. 1988.

[8] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff, "A set of level 3 basic linear algebra subprograms," vol. 16, no. 1, pp. 1–17, Mar. 1990.

[9] M. Kimura, "The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations," *Genetics*, vol. 61, no. 4, p. 893, 1969.

[10] T. H. Jukes and C. R. Cantor, "Evolution of protein molecules," *Mammalian protein metabolism*, vol. 3, pp. 21–132, 1969.

[11] J. Felsenstein, "Evolutionary trees from DNA sequences: a maximum likelihood approach," *Journal of molecular evolution*, vol. 17, no. 6, pp. 368–376, 1981.

[12] K. Goto and R. van de Geijn, "Anatomy of high-performance matrix multiplication," vol. 34, no. 3, pp. 12:1–12:25, May 2008.

[13] http://www.openblas.net, 2015.

[14] F. G. Van Zee and R. A. van de Geijn, "BLIS: A Framework for Rapidly Instantiating BLAS Functionality," *ACM Trans. Math. Softw.*, vol. 41, no. 3, pp. 14:1–14:33, Jun. 2015. [Online]. Available: http://doi.acm.org/10.1145/2764454

[15] F. G. Van Zee, T. Smith, F. D. Igual, M. Smelyanskiy, X. Zhang, M. Kistler, V. Austel, J. Gunnels, T. M. Low, B. Marker, L. Killough, and R. A. van de Geijn, "The BLIS Framework: Experiments in Portability," *ACM Transactions on Mathematical Software*, 2015, accepted.

[16] N. Alachiotis and G. Weisz, "High Performance Linkage Disequilibrium: FPGAs Hold the Key," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. ACM, 2016, pp. 118–127. [Online]. Available: http://doi.acm.org/10.1145/2847263.2847271

[17] Benchmarking crc32 and popcnt instructions. [Online]. Available: http://www.strchr.com/crc32_popcnt

[18] I. S. Haque, V. S. Pande, and W. P. Walters, "Anatomy of high-performance 2d similarity calculations," *Journal of chemical information and modeling*, vol. 51, no. 9, pp. 2345–2351, 2011.

[19] C. C. Chang, C. C. Chow, L. C. Tellier, S. Vattikuti, S. M. Purcell, and J. J. Lee, "Second-generation PLINK: rising to the challenge of larger and richer datasets," *Gigascience*, no. 4, 2015.

[20] S. Purcell, B. Neale, K. Todd-Brown, L. Thomas, M. A. Ferreira, D. Bender, J. Maller, P. Sklar, P. I. De Bakker, M. J. Daly *et al.*, "PLINK: a tool set for whole-genome association and population-based linkage analyses," *The American Journal of Human Genetics*, vol. 81, no. 3, pp. 559–575, 2007.

[21] N. Alachiotis, A. Stamatakis, and P. Pavlidis, "OmegaPlus: a scalable tool for rapid detection of selective sweeps in whole-genome datasets," *Bioinformatics*, vol. 28, no. 17, pp. 2274–2275, 2012.

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

[23] Y. Kim and R. Nielsen, "Linkage disequilibrium as a signature of selective sweeps," *Genetics*, vol. 167, no. 3, pp. 1513–1524, Jul. 2004. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/15280259

[24] A. Stamatakis and N. Alachiotis, "Time and memory efficient likelihood-based tree searches on phylogenomic alignments with missing data," *Bioinformatics*, vol. 26, no. 12, pp. i132–i139, 2010.

[25] L. A. Mathew, P. R. Staab, L. E. Rose, and D. Metzler, "Why to account for finite sites in population genetic studies and how to do this with jaatha 2.0," *Ecology and evolution*, vol. 3, no. 11, pp. 3647–3662, 2013.

[26] D. V. Zaykin, A. Pudovkin, and B. S. Weir, "Correlation-based inference for linkage disequilibrium with multiple alleles." *Genetics*, vol. 180, no. 1, pp. 533–45, Sep. 2008. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2535703\&tool=pmcentrez\&rendertype=abstract

[27] B. Pfeifer, U. Wittelsbürger, S. E. Ramos-Onsins, and M. J. Lercher, "PopGenome: An Efficient Swiss Army Knife for Population Genomic Analyses in R." *Molecular biology and evolution*, vol. 31, no. 7, pp. 1929–36, Jul. 2014. [Online]. Available: http://mbe.oxfordjournals.org/content/31/7/1929

[28] M. Slatkin, "Linkage disequilibrium-understanding the evolutionary past and mapping the medical future," *Nature Reviews Genetics*, vol. 9, no. 6, pp. 477–485, 2008.