qLD: High-performance Computation of Linkage Disequilibrium on CPU and GPU

Charalampos Theodoris*, Nikolaos Alachiotis§, Tze Meng Low‖ and Pavlos Pavlidis¶
* Technical University of Crete, Greece
§ University of Twente, The Netherlands
‖ Carnegie Mellon University, USA
¶ Foundation for Reseach and Technology-Hellas, Greece
Email: *ctheodoris@isc.tuc.gr, §n.alachiotis@utwente.nl, ‖lowt@andrew.cmu.edu, ¶pavlidis@ics.forth.gr

***Abstract*—Linkage disequilibrium (LD) is the non-random association between alleles at different loci. Assessing LD in thousands of genomes and/or millions of single-nucleotide polymorphisms (SNPs) exhibits excessive time and memory requirements that can potentially hinder future large-scale genomic analyses. To this end, we introduce qLD (quickLD) (*https://github.com/StrayLamb2/qLD*), a highly optimized open-source software that assesses LD based on Pearson's correlation coefficient. qLD exploits the fact that the computational kernel for calculating LD can be cast in terms of dense linear algebra operations. In addition, the software employs memory-aware techniques to lower memory requirements, and parallel GPU architectures to further shorten analysis times. qLD delivers up to 5x faster processing than the current state-of-the-art software implementation when run on the same CPU, and up to 29x when computation is offloaded to a GPU. Furthermore, the software is designed to quantify allele associations between arbitrarily distant loci in a time- and memory-efficient way, thereby facilitating the evaluation of long-range LD and the detection of co-evolved genes. We showcase qLD on the analysis of 22,554 complete SARS-CoV-2 genomes.**

***Index Terms*—Linkage disequilibrium, Software, GPU**

I. Introduction

Genomic dataset sizes currently grow at an unprecedented pace, yielding the deployment of memory- and performance-optimized computational approaches a prerequisite for the analysis of future large-scale datasets. In population genetics, linkage disequilibrium (LD), defined as the non-random association between alleles at different loci, has several practical applications. LD is used to identify interactions among co-evolved genes by identifying complementary mutations [1], or to search for traces of positive selection by revealing particular patterns in subgenomic regions [2]. In genome-wide association studies (GWAS), LD facilitates the detection of polymorphisms of interest, e.g., associated with human diseases [3], thereby contributing to the design of more effective drug treatments [4]. Recently, LD was used to locate recombination hotspots in the SARS-CoV-2 genome by assessing the reduction of association between mutations with an increasing genomic distance [5].

The preliminary steps of an LD study include a) DNA sequencing for a set of individuals of interest and b) short-read mapping to a reference genome to create a multiple-sequence alignment (MSA). These are followed by a so-called SNP calling step that identifies the polymorphic sites in the MSA, which are commonly referred to as single-nucleotide polymorphisms (SNPs). Computing LD requires the calculation of allele and haplotype frequencies per SNP and pair of SNPs, respectively. Thus, compute and memory requirements increase linearly with the number of genomes (sample size) and quadratically with the number of SNPs. While the number of SNPs is limited by the chromosomal length, sample sizes continue to increase rapidly, fueled by advances in DNA sequencing technologies that have improved accuracy and throughput and reduced costs. To put the sample-size growth into perspective, the 1000Genomes [6] project that was launched in January 2008 sequenced 2,504 human genomes in an 8-year span [6], while well over 50,000 SARS-CoV-2 complete genomes are already available on GISAID [7] since the beginning of the ongoing coronavirus disease 2019 (COVID-19) pandemic (December 2019).

High-performance software implementations that employ the underlying hardware efficiently and scale well with an increasing number of samples are required to ensure that future scientific discoveries in the fields of population genetics and computational biology will not be obstructed by computational inefficiencies and/or excessive memory requirements. To this end, this work presents qLD (quickLD), an open-source software that couples highly optimized kernels for modern microprocessor [8] and GPU architectures [9] with a custom LD-specific compressed file format, which collectively allow large-scale LD analyses to be conducted on off-the-shelf workstations in a fraction of the time required by state-of-the-art software. qLD outperforms the widely used software PLINK 1.9 [10] as the sample size increases, achieving up to 5x faster processing when the two CPU implementations are compared. When qLD offloads the compute-intensive task of calculating haplotype frequencies to a GPU, analyses complete up to 29x faster than PLINK 1.9, which does not have the capacity to employ a GPU.

The remainder of this paper is organized as follows. Section II provides the mathematical background, while Section III presents related work on computing LD on various platforms. Section IV describes the design and use of qLD, while Section V evaluates performance and scalability against the state of the art. Finally, we conclude in Section VII.

2471-7819/20/$31.00 ©2020 IEEE
DOI 10.1109/BIBE50027.2020.00019

II. Linkage Disequilibrium (LD)

Population genetics employ LD as a statistical measure to identify mutated alleles that are co-inherited more frequently that one would expect if these alleles were inherited independently. From a computational point of view, the input to an LD study is either a multiple sequence alignment (MSA), i.e., a $n \times m$ matrix that comprises n rows (one row per genome) of m columns each (also referred to as alignment sites), or a file that only comprises sites of interest, e.g., SNPs in a Variant Call Format (VCF [11]) file. A SNP is essentially an alignment site with two or more DNA states, i.e., at least one mutation has occurred at that site, whereas monomorphic sites are non-informative for computing LD and therefore are discarded.

A. Genomic Data Representation

A widely adopted evolutionary model in real-world analyses as well as *in silico* simulations is the infinite-site model (ISM) [12]. It assumes an infinite number of possible genomic locations where a mutation can occur, which leads to at most one mutation per site. In other words, every mutation appears on a site where no mutation has previously ocurred. The ISM allows SNPs to be represented by binary vectors, where a '0' describes the allele state prior to a mutation (ancestral state) while '1' indicates the new state after a mutation (derived state). To reduce the amount of memory accesses, we store each SNP as a group of N_{int} w-bit-long unsigned integers with N_{int} defined as follows:

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

with zero padding if $N_{seq} \mod w \neq 0$, and $w = 64$. The entire set of SNPs that collectively describe a genomic region of interest for computing LD is represented by a $(k \times w) \times n$ matrix, G, where $k = N_{int}$. An example of the matrix G, which we henceforth refer to as the genomic matrix, is shown in Figure 1. The genomic matrix G exclusively comprises SNPs. All monomorphic sites are already discarded during a preceding format conversion step, discussed in detail in Section IV. For clarity reasons, Figure 1 does not show the site locations, which are stored in a separate memory space, but note that adjacent SNPs in G can be thousands of sites apart in the genome. We henceforth represent a SNP as a column vector s.

B. Computing LD using Pearson's correlation coefficient

LD deals with the probability of independent events. The event that two mutations appear at different loci in the same sequence is said to be not independent, or in other words the corresponding pair of SNPs are in linkage disequilibrium, when the probability of the two mutations occurring at different loci in the same sequence is not the same as the product of the probabilities of these mutations occurring independently. Therefore, we compute

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

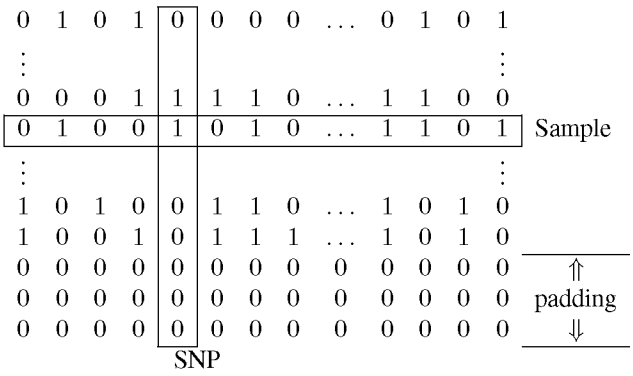

Fig. 1: Pictorial representation of the $(k \times w) \times n$ genomic matrix G. The rows represent samples and the columns represent SNPs at different genomic locations. Adapted from [13].

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

For N_{seq} number of genomes, the probability that a mutation occurs in a SNP s_x, denoted as P_x, can be obtained with the following equation:

$$P_x = \frac{s_x^T s_x}{N_{seq}}, \tag{2}$$

which counts the number of '1's in s_x and then divides it with the total number of bits in the s_x. This is practically the derived allele frequency in SNP s_x.

$P_{i,j}$, which is the haplotype frequency, is computed by counting the number of samples that have mutations in both SNPs, s_i and s_j, and then dividing that number by N_{seq}, as follows:

$$P_{i,j} = \frac{s_i^T s_j}{N_{seq}} \tag{3}$$

Using Equations 2 and 3, we can compute $D_{i,j}$ for all possible pairs of SNPs s_i and s_j in the following manner:

$$\begin{aligned} D_{i,j} &= P_{i,j} - P_i P_j \\ &= \frac{1}{N_{seq}}(s_i^T s_j) - \frac{1}{N_{seq}^2}(s_i^T s_i)(s_j^T s_j) \end{aligned} \tag{4}$$

The LD formulation in Equation 1 is not widely employed because the sign and range of $D_{i,j}$ vary with the frequency at which different mutations occur, which hinders $D_{i,j}$ comparisons across different SNP pairs. Therefore, several standardization methods for D have been proposed. To the best of the

authors' knowledge, the most commonly used measure is the squared Pearson coefficient r_{ij}^2:

$$r_{ij}^2 = \frac{(P_{i,j} - P_i P_j)^2}{P_i P_j (1-P_i)(1-P_j)} = \frac{D_{i,j}^2}{P_i P_j (1-P_i)(1-P_j)} \quad (5)$$

It has the advantage that all r_{ij}^2 values are between 0 and 1, with higher values suggesting stronger association. Regardless of the employed measure, note that the cost of computing $r_{i,j}^2$ values for all pairs of SNPs is dominated by the cost of D.

III. Related Work

Next-generation sequencing technologies currently generate a plethora of DNA data for population genomics, gradually establishing the need for high-performance tools that are capable of efficiently conducting large-scale analyses.

Pfeifer et al. [14] released PopGenome, an R package for population genetic analyses that can compute a wide range of statistics, including LD. Yet, the deployed LD kernel does not exploit the cache hierarchy. Alachiotis et al. [15] released OmegaPlus, which computes the squared Pearson coefficient as a measure of LD, and deploys an intrinsic popcount instruction supported in hardware to count the number of derived alleles per SNP and SNP pair. Similarly to PopGenome, OmegaPlus does not fully exploit the cache hierarchy either.

Chang et al. [16] released a comprehensive update to the widely used PLINK software [10] for whole-genome association and population-based linkage analyses. The updated implementation (PLINK 1.9/2.0) exhibits significant performance and scalability improvements over the initial software. It relies on bitwise operations, multithreading, and high-level algorithmic improvements for the most compute-demanding functions, such as distance-based clustering and LD-based pruning. PLINK 1.9 implements the squared Pearson coefficient as a measure of LD and deploys the SSE2-based Lauradoux/Walish popcount algorithm to achieve high performance.

Alachiotis et al. [8] observed that the computational kernel for calculating LD can be cast in terms of dense linear algebra (DLA) operations. This allowed to leverage the collective knowledge in the DLA community in developing high-performance implementations for various microprocessor architectures, leading to the design of a highly efficient CPU kernel for LD that achieves between between 84% and 95% of the theoretical peak performance of the machine.

Building upon the aforementioned DLA-based approach, which targeted modern CPU architectures, Binder et al. [9] presented a generic SNP-comparison framework that calculates LD on GPU architectures. The authors ported the proposed framework onto a variety of GPU platforms from different vendors, reporting between 55% and 97% of the theoretical peak throughput of each specific GPU architecture.

Both of the aforementioned DLA-based approaches for computing LD on CPUs [8] and GPUs [9] solely focused on the LD kernel itself, thereby requiring significant development effort to allow their deployment in real-world population genetics analyses. qLD eliminates this requirement by coupling specially adapted versions of the two kernels with a memory efficient mechanism for parsing SNP data in the widely used VCF file format, thereby yielding a high-performance ready-to-use software implementation for large-scale LD studies.

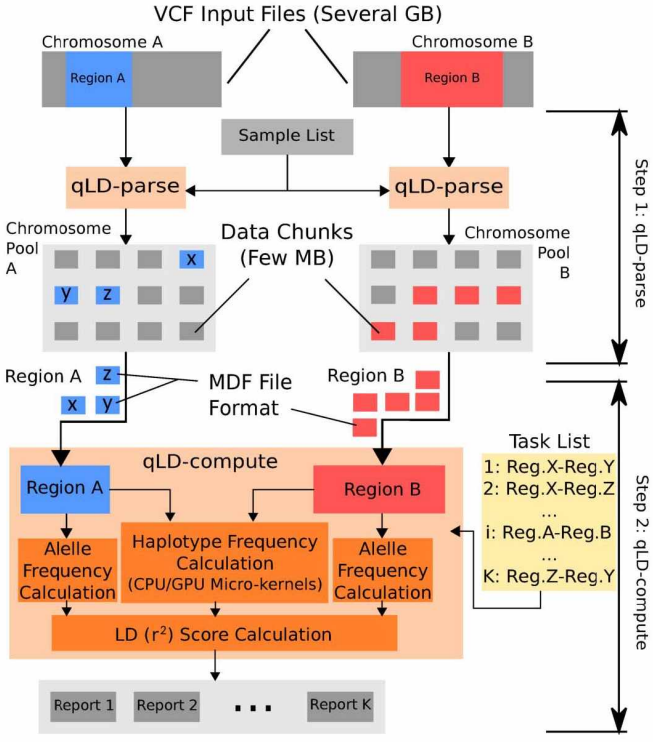

Fig. 2: The qLD workflow: Step 1 includes parsing of VCF files and generation of pools of MDF files, while Step 2 includes processing of several compute tasks (region pairs) using an optimized LD micro-kernel within the BLIS framework.

IV. Design and Usage

A. Computational workflow

In addition to performance, a major concern in designing an efficient software for large-scale LD studies is the memory management, since requirements grow quadratically with the number of SNPs. For this reason, qLD implements a two-step process that separates parsing from processing, which allows pairwise LD calculations between arbitrarily distant SNPs to be conducted without increasing the memory space. We henceforth refer to the parsing and the processing modes of qLD as `qLD-parse` and `qLD-compute`, respectively.

Figure 2 illustrates the qLD workflow for calculating all pairwise LD scores between pairs of subgenomic regions on different chromosomes. The first step (`qLD-parse` in the figure) focuses on converting each–potentially large–VCF input file to an intermediate data representation that allows faster subsequent parsing and processing. The conversion process is performed once per VCF file and list of samples of interest. During this step, each VCF is split into a series of fixed-size

files (Chromosome Pool in the figure) in a custom, LD-specific data format dubbed Matrix Data Format (MDF, see Listing 1). The genomic coordinates (start and end position) of the subgenomic region stored in each MDF file are part of the filename. This convenient naming convention facilitates backward mapping of MDF files to the chromosomal region of the input VCF. Each MDF file is typically a few MB in size, regardless of the number of samples or SNPs in the VCF, with larger sample sizes leading to fewer SNPs per file.

```
1   ##fileformat=VCF
2   ... smp0 smp1 smp2 smp3 smp4 smp5
3   ...  1    1    0    0    1    1
4   ...  1    0    1    0    1    1
5   ...  0    0    0    1    0    1

1   ##fileformat=MDF
2   ... Bitcount packed0
3   ...    4      15
4   ...    3      11
5   ...    1      1
```

Listing 1: VCF-to-MDF conversion using the sample list "smp0 smp1 smp4 smp5" and assuming 4-bit MDF words for convenience. LD-parse-2MDF parses the VCF file and converts the valid samples (in the sample list) into 4-bit unsigned words. A larger number of samples leads to a larger sequence of 4-bit words per MDF row. Note that MDF files to be processed using qLD contain 64-bit words. Each line also contains the total bitcount per SNP (MDF row).

The second step (qLD-compute in the figure) focuses on computing all LD scores between SNPs in pairs of genomic regions. A region pair, e.g., regions A and B in Figure 2, represents a compute task. qLD relies on the genomic coordinates that appear in the MDF filenames to parse the right subsets of MDF files and load the two chromosomal regions of a compute task to main memory. For region A in Figure 2, for instance, only the MDF files x, y, and z are parsed. When the region pair is represented in main memory, a highly optimized LD micro-kernel is employed within the BLAS-Like Instantiation Software (BLIS) [17], [18] framework to compute haplotype frequencies on the CPU, as described by Alachiotis et al. [8]. The BLIS framework is also employed for SNP processing on the GPU, as described by Binder et al. [9]. Thereafter, allele frequencies and the final r^2 scores are computed according to Equations 2 and 5, respectively. qLD produces a separate LD report per compute task.

B. Usage

The following listings provide the required command lines for using qLD. A single asterisk indicates a required argument, while double asterisks indicate mutually exclusive arguments.

a) qLD-parse: The first step of the qLD workflow is implemented through qLD-parse, which itself consists of two subfunctions: qLD-parse-VCF (splits a VCF to chunks) and qLD-parse-2MDF (converts each chunk to an MDF file). Listings 2 and 3 provide the basic input arguments.

```
./bin/qLD-parse-VCF
-inputList $STRING    **
-input     $STRING    **
-output    $STRING    *
-size      $INTEGER   *
```

Listing 2: qLD-parse-VCF command line

```
./bin/qLD-parse-2MDF
-input       $STRING  *
-output      $STRING  *
-sampleList  $STRING
```

Listing 3: qLD-parse-2MDF command line

qLD-parse-VCF receives an input file that is either a single VCF (**-input**) or a list of several VCF files (**-inputList**) and creates a directory (**-output**) that contains VCF chunks of fixed size in MB (**-size**). The path to the produced output folder is input to qLD-parse-2MDF (**-input**), along with the list of samples of interest (**-sampleList**), which stores the generated MDF files in a user-given directory (**-output**).

b) qLD-compute: In the second step, qLD-compute is launched to conduct the required LD calculations. Listing 4 provides the basic input arguments.

```
./bin/qLD-compute
-input          $STRING   **
-input2         $STRING   **
-inputList      $STRING   **
-output         $STRING   *
-ploidy         $STRING   *
-r2limit        $FLOAT
-mdf
-gpu
```

Listing 4: qLD-compute command line

Using VCF chunks or, preferably, MDF files (**-mdf**) as input, qLD-compute generates LD reports. Similarly to qLD-parse-VCF, either a pair of inputs (**-input,-input2**) are provided, in which case a single report is produced, or a list of several tasks (**-inputList**), in which case a report per task is stored to the output directory (**-output**). The ploidy is also required (**-ploidy**). Optional parameters allow to deploy a GPU (**-gpu**), and/or apply a threshold to the output (**-r2limit**).

V. Performance Evaluation

A. Experimental setup

For evaluation purposes, we performed experiments on an off-the-shelf personal laptop and the ARIS supercomputer (*https://hpc.grnet.gr/en/*). Table I provides the specifications of the employed test platforms. We compare performance with the widely used software PLINK 1.9 [16], both in terms of execution times and throughput, based on the analysis of simulated datasets with an increasing number of SNPs and

samples. All runs were performed using the default r^2 limit of PLINK 1.9 ($r^2_{limit} = 0.2$).

TABLE I: System Specifications

	System 1	***System 2***
Description	Off-the-shelf laptop	Aris supercomputer
CPU Model	Core i5-8300H	Xeon E5-2660v3
Microarchitecture	Coffee Lake	Haswell
Nominal Frequency	2.3 GHz	2.6 GHz
Max. Turbo Frequency	4.0 GHz	3.3 GHz
Processors	1	2
Cores/Processor	4	10
Total Cores	4	20
Memory	8 GB	32 GB
GPU Model	GTX 1050-M	Tesla K40
Streaming Multiprocessors	5	15
Cuda Cores	640	2880
GPU Memory	4 GB	12 GB

B. Execution time comparison

Figure 3 illustrates execution times of qLD and PLINK 1.9 when the sample size increases up to 100k (Fig. 3A), and the number of SNPs increases up to 10k for fixed sample sizes of 2,5k (Fig. 3B), 10k (Fig. 3C), and 100k (Fig. 3D). Expectedly, execution times increase linearly with the number of samples and quadratically with the number of SNPs. We can also observe that System 1 (off-the-shelf laptop) outperforms System 2 (ARIS supercomputer), exhibiting shorter execution times over all runs. This is expected because qLD is a sequential software (future work will focus on parallel processing on multiple CPUs and GPUs), and the operating frequency of System 1 (3.8 GHz turbo frequency) is about 40% higher than the nominal frequency of System 2 (2.6 GHz). Tables II and III summarize these results in terms of speedup (discussed in detail in Section V-C).

C. Throughput comparison

A major difference between qLD and PLINK 1.9 is the way pairwise LD scores are computed. As previously mentioned, qLD relies on the BLIS framework and exploits the observation that pairwise LD computations can be cast as a matrix-multiply operation. Computing all pairwise LD scores between all SNPs in a single region using BLIS results in the calculation of a symmetric output matrix, thus having evaluated the same scores twice. PLINK 1.9, on the other hand, only processes a single file/region and calculates a diagonal matrix with all the pairwise LD scores. Because of this, when a single region is processed, qLD computes twice the amount of scores that PLINK 1.9 computes. To perform a fair throughput comparison, since qLD can not compute only the diagonal matrix when a single region is processed, and PLINK 1.9 does not support two different regions as input, we report effective throughput performance for an increasing number of samples (Table II) and SNPs (Table III), distinguishing between processing a single region and a pair of regions of

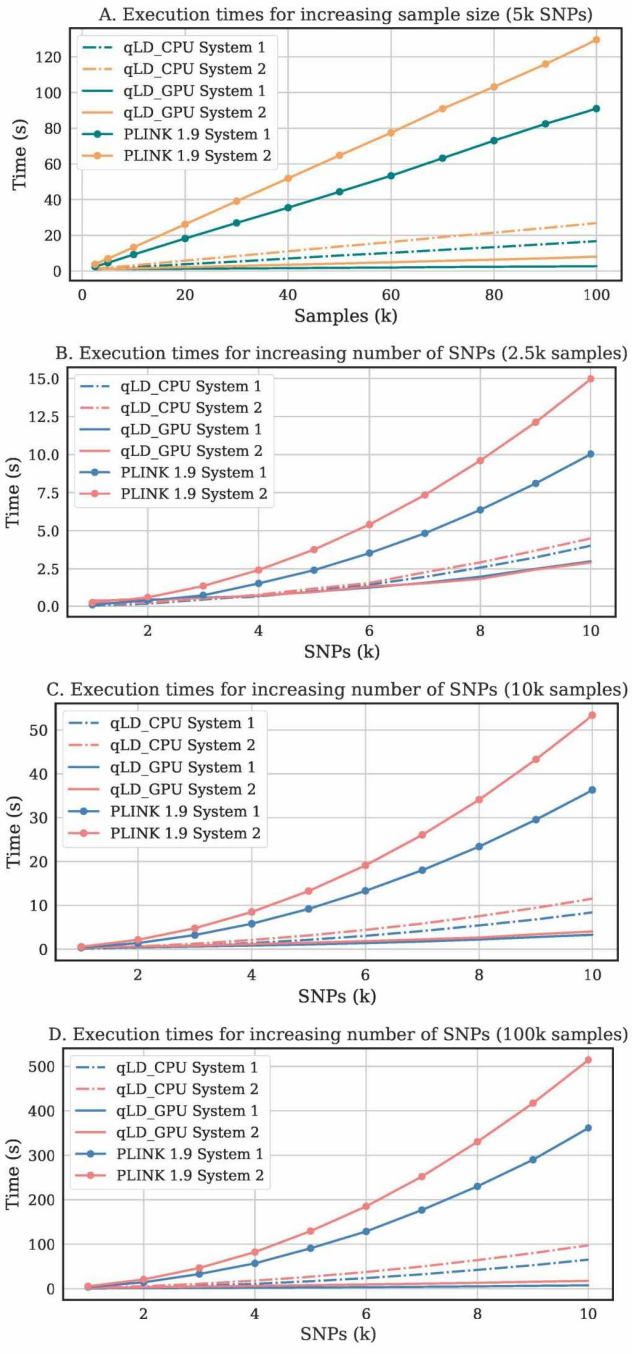

Fig. 3: Execution times on System 2 for increasing number of samples (from 2,500 to 100,000 sequences, Fig. 3A) and increasing numbers of SNPs (from 1,000 to 10,000 SNPs) when the sample size is 2,500 sequences (Fig. 3B), 10,000 sequences (Fig. 3C), and 100,000 sequences (Fig. 3D). We observe linear and quadratic increase with the number of samples and the number of SNPs, respectively. qLD_CPU and qLD_GPU are up to 5.3x and 29.3x faster than PLINK 1.9 running on the CPU, respectively.

TABLE II: Throughput performance on System 2 for increasing number of samples when a single SNP region and a pair of regions are processed by qLD_CPU, qLD_GPU, and PLINK 1.9. The table also provides the observed speedups of qLD_CPU and qLD_GPU versus PLINK 1.9. The region size is 5,000 SNPs, thus 12.5×10^6 and 25×10^6 LD scores are computed when a single SNP region and a pair of regions are processed, respectively.

Samples	Single Region					Pair of regions	
	Throughput ($LD \times 10^6/sec$)			Speedup (x) over PLINK 1.9		Throughput ($LD \times 10^6/sec$)	
($\times 10^3$)	qLD_CPU	qLD_GPU	PLINK 1.9	qLD_CPU	qLD_GPU	qLD_CPU	qLD_GPU
2.5	8.679	11.466	3.306	2.625	3.468	17.361	22.936
10.0	3.967	8.560	0.942	4.210	9.082	7.937	17.123
20.0	2.155	5.813	0.478	4.510	12.167	4.310	11.628
30.0	1.497	4.416	0.319	4.685	13.823	2.994	8.834
40.0	1.133	3.581	0.241	4.706	14.874	2.267	7.163
50.0	0.912	2.983	0.193	4.726	15.465	1.823	5.967
60.0	0.770	2.572	0.161	4.773	15.940	1.540	5.144
70.0	0.657	2.200	0.137	4.782	16.012	1.314	4.401
80.0	0.583	1.965	0.121	4.812	16.228	1.166	3.931
90.0	0.518	1.758	0.108	4.804	16.312	1.036	3.516
100.0	0.465	1.558	0.096	4.822	16.157	0.930	3.117

TABLE III: Throughput performance on System 2 for increasing region size (number of SNPs) when a single SNP region and a pair of regions are processed by qLD_CPU, qLD_GPU, and PLINK 1.9. The table also provides the observed speedups of qLD_CPU and qLD_GPU versus PLINK 1.9. The sample size is 100,000 sequences.

Region size (SNPs)	Single Region						Pair of regions		
	number of LD scores	Throughput ($LD \times 10^6/sec$)			Speedup (x) over PLINK 1.9		number of LD scores	Throughput ($LD \times 10^6/sec$)	
($\times 10^3$)	($\times 10^6$)	qLD_CPU	qLD_GPU	PLINK 1.9	qLD_CPU	qLD_GPU	($\times 10^3$)	qLD_CPU	qLD_GPU
1.0	0.500	0.257	0.284	0.095	2.701	2.977	1.000	0.515	0.568
2.0	1.999	0.359	0.645	0.097	3.713	6.671	4.000	0.718	1.290
3.0	4.499	0.414	0.953	0.097	4.279	9.845	9.000	0.829	1.907
4.0	7.998	0.444	1.303	0.097	4.568	13.414	16.000	0.887	2.606
5.0	12.498	0.466	1.580	0.097	4.822	16.363	25.000	0.931	3.161
6.0	17.997	0.480	1.902	0.097	4.934	19.562	36.000	0.960	3.805
7.0	24.497	0.492	2.181	0.097	5.060	22.443	49.000	0.984	4.363
8.0	31.996	0.498	2.439	0.097	5.140	25.176	64.000	0.996	4.878
9.0	40.496	0.507	2.616	0.097	5.220	26.951	81.000	1.013	5.233
10.0	49.995	0.512	2.842	0.097	5.272	29.254	100.000	1.024	5.685

the same size in terms of SNPs. As can be observed in the tables, qLD is between 2.63x and 4.82x faster than PLINK 1.9 when the sample sizes increases from 2,500 sequences to 100,000 sequences, and between 2,70x and 5,27x faster when the region size increases from 1,000 SNPs to 10,000 SNPs. When qLD offloads computations to a GPU, qLD is between 3.47x and 16.31x faster than PLINK 1.9 (running on the CPU) when the sample size increases, and between 2.98x and 29.25x faster when the number of SNPs increases.

D. Execution time breakdown

Figure 4 presents execution time breakdowns for the processing step of qLD when the number of samples and the number of SNPs increase. `qLD-compute` consists of 4 main stages that collectively contribute to its total execution time:

- **Memory Layout Transformation (MLT):** transposition of one of the two input genomic regions, as required by BLIS for computing LD as a marix-multiply operation. This stage is not performance-critical as it only relocates SNP data in memory.
- **General Matrix-Multiply (GEMM):** Invocation of the CPU/GPU LD micro-kernel through BLIS for computing haplotype frequencies as a matrix-multiply operation. This stage is performance-critical and dominates the total execution time in all CPU runs.
- **Linkage Disequilibrium (LD) score computation:** Calculation of the allele frequencies for all SNPs in the two regions and the final LD scores. This stage heavily relies on floating-point operations, but the amount of time spent on floating-point operations for computing LD scores becomes negligible as the sample sizes grow because the overall execution time is dominated by GEMM. When qLD deploys a GPU, however, the GPU-based GEMM stage is between 2 and 20 times faster than the CPU one, which leads to the LD time dominating execution times for sample sizes as low as 10,000 sequences.

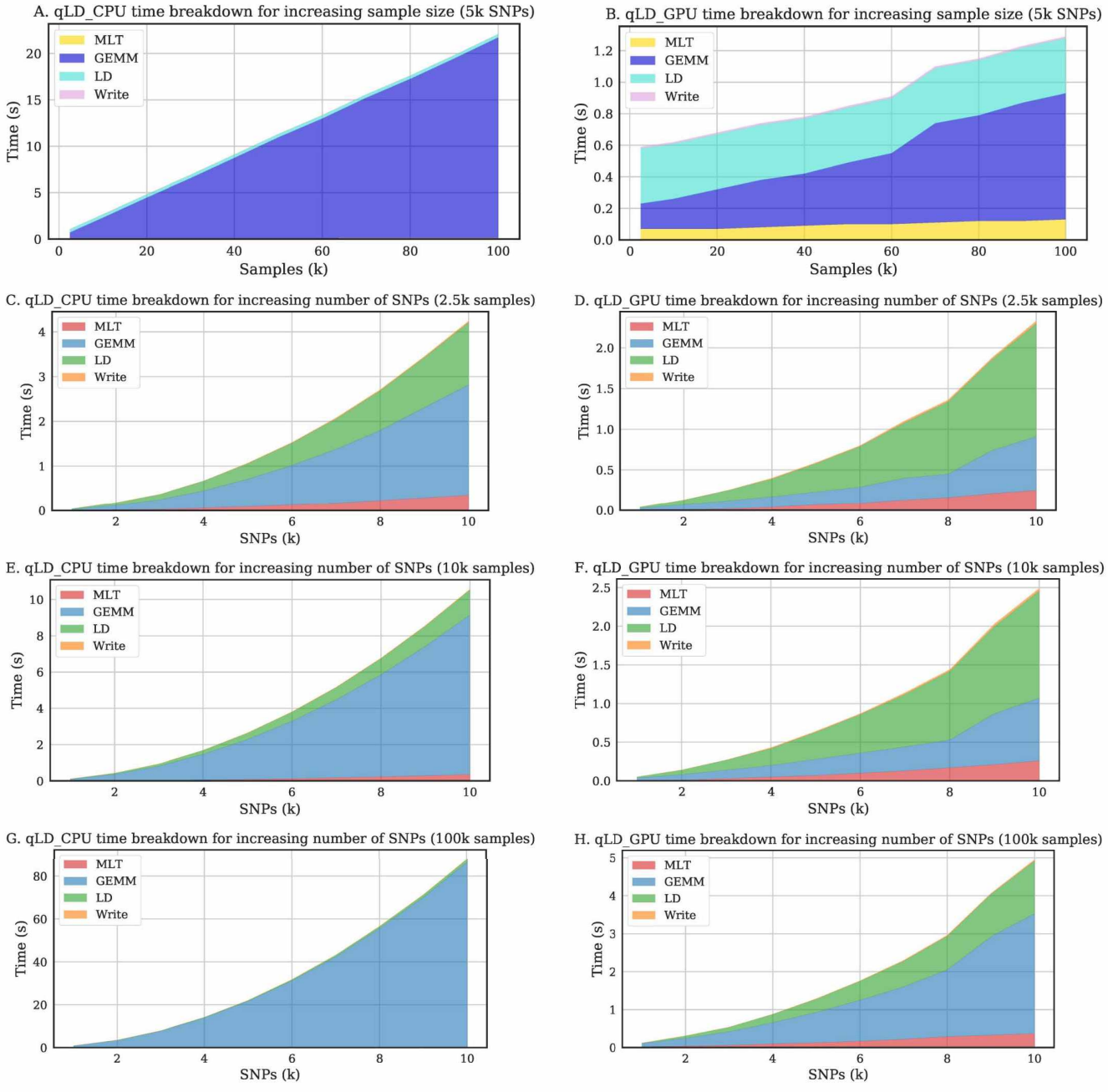

Fig. 4: Execution time breakdown of qLD-compute when executing on a CPU (left column) and a GPU (right column) on System 2. The total processing time is spent on four discrete stages: a) Memory Layout Transposition (MLT), b) General Matrix-Multiply (GEMM), c) Linkage Disequilibrium (LD) score calculation, and d) Output generation (Write).

- **Write Output:** Storing LD scores in a text file. Note that we apply the default cutoff threshold for LD scores ($r^2_{limit} = 0.2$) to discard very low scores and prevent output reports from exploding in size. Since disk input/output is critical to performance for large-scale LD studies, we devised a simple lookup-table-based approach for printing. Based on the fact that the output is dominated by floating-point values (FPVs), we map each FPV to a character array from '0' to '9', where each digit points to its representation. This simple optimization achieves up to 1.7x faster printing, with 10^6 FPVs with 9-digit precision stored in a file in 0.123 seconds, while requiring 0.208 seconds when the standard C library function `fprintf` is used.

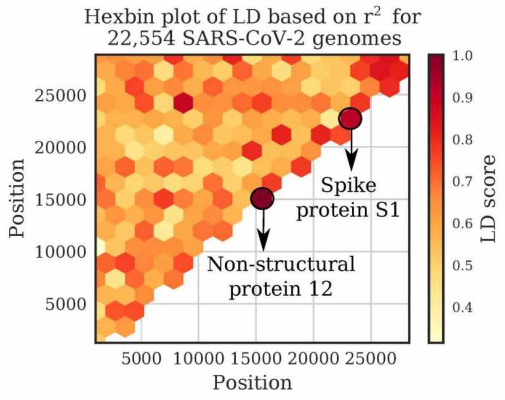

Fig. 5: Hexbin plot (gridsize=16) of LD scores calculated using qLD on System 1 (laptop) in 18 seconds.

VI. Application on SARS-CoV-2 Genomes

To showcase qLD, we used 39,941 high-coverage SARS-CoV-2 genomes from the GISAID database (*https://www.gisaid.org/*), downloaded on July 9, 2020. We kept only complete sequences (genomes with base pair lengths greater than 29,000), and trimmed the ambiguous states (Ns) from both the beginning and the end of the genomes. Thereafter, we excluded all sequences that contained Ns, and employed the experimental version of MAFFT [19] for closely-related viral genomes to create a multiple sequence alignment in FASTA format. The final dataset, after discarding the sequences that did not pass the aforementioned filters, comprised 22,554 genomes. We used snp-sites [20] to convert the FASTA to VCF for processing with qLD, invoking the tool's built-in option for omitting columns that did not exclusively contain A, C, G, T.

Figure 5 illustrates qLD scores (r^2) in a hexbin plot with grid=16. The two highest-score bins shown in the figure correspond to regions $15,042-15,517$ and $22,684-23,198$ with scores 1.0 and 0.92, respectively. Both regions are UniProt highlighted regions of interest as shown in the UCSC genome browser view of SARS-CoV-2 genomic datasets (*https://genome.ucsc.edu/cgi-bin/hgTracks?db=wuhCor1*), with the first region found in the non-structural protein 12 and is known to interact with RMP Rendemsivir, while the second region found in the Spike protein S1 and is a motif in the Receptor Binding Domain that binds to human ACE2.

VII. Conclusion

In this paper, we presented a new scalable platform for calculating pairwise LD scores in heterogeneous environments. Using optimized kernels based on the BLIS framework (`qLD_CPU`) and the OpenCL framework (`qLD_GPU`), we achieved speedup in all test-cases (up to 29x) over the widely used state-of-the-art Plink 1.9. Based on HPC practices, the platform works on par with the theoretical performance of these computation kernels with minimal overhead from the rest of the processing steps.

Based on the outcome of our experiments, our future work will focus on parallelization techniques that will lead to even higher performance and scalability. A parallel heterogeneous approach will take advantage of the high CPU core count of modern computers and couple the two (now separate) modes of execution, i.e., `qLD_CPU` and `qLD_GPU`.

VIII. Acknowledgements

This research has been partially funded by a FORTH Synergy Grant (2019) to Pavlos Pavlidis.

References

[1] 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*, 86(5):674–685, 2010.

[2] J. M. Smith and J. Haigh. The hitch-hiking effect of a favourable gene. *Genetics Research*, 23(1):23–35, 1974.

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

[4] M. T. Alam, D. K. De Souza, S. Vinayak, S. M. Griffing, A. C. Poe, N. O. Duah, A. Ghansah, K. Asamoa, L. Slutsker, et al. Selective sweeps and genetic lineages of plasmodium falciparum drug-resistant alleles in ghana. *Journal of Infectious Diseases*, 203(2):220–227, 2011.

[5] M. Vasilarou, N. Alachiotis, J. Garefalaki, A. Beloukas, and P. Pavlidis. Population genomics insights into the recent evolution of sars-cov-2. *BioRxiv*, 2020.

[6] 1000 Genomes Project Consortium and others. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015.

[7] S. Elbe and G. Buckland-Merrett. Data, disease and diplomacy: Gisaid's innovative contribution to global health. *Global Challenges*, 1(1):33–46, 2017.

[8] N. Alachiotis, T. Popovici, and T. M. Low. Efficient computation of linkage disequilibria as dense linear algebra operations. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 418–427. IEEE, 2016.

[9] E. Binder, T. M. Low, and D. T. Popovici. A portable gpu framework for snp comparisons. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops*, pages 199–208. IEEE, 2019.

[10] S. Purcell et al. PLINK: a tool set for whole-genome association and population-based linkage analyses. *The American Journal of Human Genetics*, 81(3):559–575, 2007.

[11] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, et al. The variant call format and vcftools. *Bioinformatics*, 27(15):2156–2158, 2011.

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

[13] N. Alachiotis and G. Weisz. High performance linkage disequilibrium: Fpgas hold the key. In *Proceedings of 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 118–127, 2016.

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

[15] N. Alachiotis et al. OmegaPlus: a scalable tool for rapid detection of selective sweeps in whole-genome datasets. *Bioinf.*, 28(17):2274–2275, 2012.

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

[17] F. G. Van Zee and R. A. Van De Geijn. Blis: A framework for rapidly instantiating blas functionality. *ACM Transactions on Mathematical Software (TOMS)*, 41(3):1–33, 2015.

[18] F. G. V. Zee, T. M. Smith, B. Marker, T. M. Low, R. A. V. D. Geijn, F. D. Igual, M. Smelyanskiy, X. Zhang, M. Kistler, V. Austel, et al. The blis framework: Experiments in portability. *ACM Transactions on Mathematical Software (TOMS)*, 42(2):1–19, 2016.

[19] K. Katoh and D. M. Standley. MAFFT multiple sequence alignment software version 7: improvements in performance and usability. *Molecular biology and evolution*, 30(4):772–780, 2013.

[20] A. J. Page, B. Taylor, A. J. Delaney, J. Soares, T. Seemann, J. A. Keane, and S. R. Harris. Snp-sites: rapid efficient extraction of snps from multi-fasta alignments. *Microbial genomics*, 2(4), 2016.