2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 978-1-6654-9747-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/IPDPSW55747.2022.00044

Accelerated LD-based selective sweep detection using GPUs and FPGAs

Reinout Corts
Faculty of EEMCS
University of Twente
Enschede, The Netherlands
r.corts@student.utwente.nl

Niek Sterenborg
Faculty of EEMCS
University of Twente
Enschede, The Netherlands
n.e.sterenborg@student.utwente.nl

Nikolaos Alachiotis
Faculty of EEMCS
University of Twente
Enschede, The Netherlands
n.alachiotis@utwente.nl

***Abstract*—Selective sweep detection carries theoretical significance and has several practical implications, from explaining the adaptive evolution of a species in an environment to understanding the emergence of viruses from animals, such as SARS-CoV-2, and their transmission from human to human. The plethora of available genomic data for population genetic analyses, however, poses various computational challenges to existing methods and tools, leading to prohibitively long analysis times. In this work, we accelerate LD (Linkage Disequilibrium) - based selective sweep detection using GPUs and FPGAs on personal computers and datacenter infrastructures. LD has been previously efficiently accelerated with both GPUs and FPGAs. However, LD alone cannot serve as an indicator of selective sweeps. Here, we complement previous research with dedicated accelerators for the ω statistic, which is a direct indicator of a selective sweep. We evaluate performance of our accelerator solutions for computing the ω statistic and for a complete sweep detection method, as implemented by the open-source software OmegaPlus. In comparison with a single CPU core, the FPGA accelerator delivers up to 57.1x and 61.7x faster computation of the ω statistic and the complete sweep detection analysis, respectively. The respective attained speedups by the GPU-accelerated version of OmegaPlus are 2.9x and 12.9x. The GPU-accelerated implementation is available for download here: https://github.com/MrKzn/omegaplus.git.**

***Index Terms*—Linkage disequilibrium, selective sweep, positive selection, OmegaPlus, GPU, FPGA, hardware accelerator**

I. Introduction

Natural selection is the process that explains how species adapt to a certain environment and evolve. A mutation in the genome of an individual can be positive, negative, or neural, thereby resulting in a different chance of survival and reproduction for the individual. If the mutation is advantageous, its frequency will increase in the population, whereas if the mutation is deleterious, its frequency will decrease. A selective sweep [1] describes the process in which a beneficial mutation increases in frequency in a population and becomes fixed, i.e., all individuals in the population eventually have the mutation.

Detecting selective sweeps has theoretical and practical significance. Selective sweep detection can help us understand the forces behind adaptive evolution [2]. Furthermore, whole-genome scans for selective sweeps can improve the design of drug treatments [3] and reveal reasons for treatment failures [4]. A recent study by Vasilarou et al. [5] used population genetics methods, including selective sweep detection, to investigate the spread and epidemics of SARS-CoV-2. Kang et al. [6] found signatures of a selective sweep in the spike protein of the SARS-CoV-2 genome, suggesting that this could have contributed to the emergence of the virus from animals or enabled human-to-human transmission.

According to the selective sweep theory [1], a selective sweep leaves three signatures in the genomic region surrounding the beneficial mutation: a) reduced level of genetic variation [1], b) a shift in the site frequency spectrum (SFS) toward low and high-frequency derived variants [7], and c) a distinct pattern of linkage disequilibrium (LD) levels that is characterized by high LD on each side of the beneficial mutation and low LD between loci on different sides of the beneficial mutation [8]. Crisci et al. [9], [10] evaluated several software tools that detect the aforementioned selective sweep signatures under equilibrium and non-equilibrium evolutionary scenarios, demonstrating the higher effectiveness of LD-based over SFS-based methods. The authors evaluated the LD-based OmegaPlus [11] and iHS [12] and the SFS-based SweepFinder [13] and SweeD [14], concluding that "the LD-based OmegaPlus performs best in terms of power to reject the neutral model under both equilibrium and non-equilibrium conditions". Motivated by the need to boost performance of sweep detection tools to cope with the increasing accumulation of molecular data in public and private databases [15], [16], this work explores the potential of GPU and FPGA hardware platforms for the acceleration of LD-based selective sweep detection, employing OmegaPlus [11] as reference for the design and evaluation of the hardware accelerators.

OmegaPlus computes the ω statistic [8], which quantifies how closely the LD patterns of a genomic region resemble those expected to have been generated by a selective sweep. Based on our profiling results with a varying number of samples and polymorphic sites, we observed that computing LD and ω values collectively consume over 98% of the tool's total execution time, with LD computation becoming the execution bottleneck when the number of samples increases, and ω computation dominating the execution time when a small number of sequences that contain a large number of polymorphic sites is analyzed. LD computation has been previously accelerated using GPUs [17], [18] and FPGAs [19], [20]. LD alone, however, cannot be used as an indicator of the

978-1-6654-9747-3/22/$31.00 ©2022 IEEE
DOI 10.1109/IPDPSW55747.2022.00044

presence of selective sweeps. Here, we complement previous research outcome by Binder et al. [17] and Alachiotis and Weisz [19] by designing hardware accelerators for computing the ω statistic, thereby allowing to boost performance of the complete sweep detection process. The contributions of this work are the following:

- We present and evaluate a custom floating-point pipeline architecture for computing the ω statistic on embedded and datacenter FPGA accelerator cards, achieving up to 62x faster execution than a CPU core.
- We describe a throughput-optimized GPU implementation for the ω statistic, which was evaluated on a desktop GPU and a datacenter GPU, delivering up to 2.9x faster execution than a CPU core.
- We estimate overall performance of an FPGA-accelerated system for the complete LD-based selective sweep detection, observing up to 57.1x faster execution than a CPU core.
- We integrate the LD GPU accelerator proposed by Binder et al. [17] and our ω GPU accelerator into OmegaPlus, and release and evaluate a readily available GPU-accelerated version (available at https://github.com/MrKzn/omegaplus.git) that delivers up to 24x faster execution than a CPU core and outperforms a 4-core CPU.

The remainder of this paper is organized as follows. Section II provides background information about selective sweeps and LD-based selective sweep detection. Section III reviews related work on GPU and FPGA accelerators. Section IV presents our GPU implementation of the ω statistic, and Section V describes a processing pipeline for FPGAs. Section VI evaluates performance. We conclude in Section VII.

II. Background

A. Selective sweep

A selective sweep occurs when a beneficial mutation appears in an individual at a certain point in time. Due to the higher chance of reproduction of that specific individual over the rest of the population, this mutation will increase in frequency in the population until all individuals in the population have it, i.e., the mutation has swept the population. Fig. 1 illustrates the process. The phenomena shown in Fig. 1.3-1.4 are caused by the "hitchhiking effect" [1]: neutral mutations around the beneficial mutation also increase in frequency. Because of this sweep-induced reduction of genetic variation, a subgenomic region with a reduced number of single-nucleotide polymorphisms (SNPs) in comparison with the rest of the chromosome emerges.

B. LD-based detection of selective sweeps with OmegaPlus

Kim and Nielsen [8] observed that a selective sweep can be detected by examining the Linkage Disequilibrium (LD) pattern around the position of the beneficial mutation. LD is the non-random association of alleles at different positions in the genome [21]. As previously mentioned, other mutations hitchhike along with the beneficial mutation. This causes

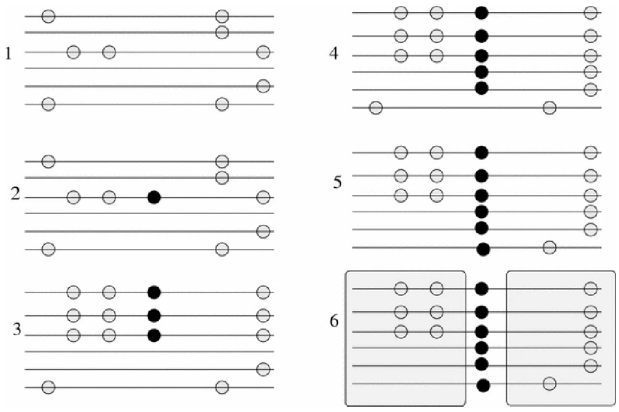

Fig. 1. Illustration of a selective sweep (adapted from [22]). (1) The population before the beneficial mutation. Each line represents an individual and the circles are mutations. (2) A beneficial mutation (black circle) occurs. (3-4) The beneficial mutation increases in frequency while the frequencies of other mutations in the proximity of the beneficial mutation change (hitchhiking effect). (5-6) The mutation is fixed (complete selective sweep). The LD pattern that emerges is used for selective sweep detection.

elevated LD on each side of the beneficial mutation, and decreased LD between polymorphic sites that are on opposite sides of the mutation. The most commonly used measure of LD relies on Pearson's correlation coefficient, r^2, as provided for the calculation of LD between SNPs i and j in Equation (1):

$$r_{ij}^2 = \frac{p_{ij} - p_i p_j{}^2}{p_i(1-p_i)p_B(1-p_j)}, \tag{1}$$

where p_i and p_j are the frequencies of the derived allele at SNPs i and j, respectively, and p_{ij} is the frequency of occurrence of the derived allele at both SNPs i and j.

Kim and Nielsen [8] proposed the ω-statistic to detect the expected LD pattern of a selective sweep. Given a genomic region with W SNPs that is split into a left L and a right R subgenomic regions that consist of l and $W-l$ SNPs, respectively, the ω statistic is computed at the junction of the two subregions as follows using Equation 2:

$$\omega = \frac{\left(\binom{l}{2} + \binom{W-l}{2}\right)^{-1}\left(\sum_{i,j\in L} r_{ij}^2 + \sum_{i,j\in R} r_{ij}^2\right)}{\left(l(W-l)\right)^{-1}\sum_{i\in L, j\in R} r_{ij}^2}. \tag{2}$$

High values indicate candidate regions since the total LD in the numerator is elevated within the L and R subgenomic regions but decreased between sites that are in different regions.

OmegaPlus evaluates a user-defined number of ω locations across a genome. Every such location is assumed at the centre of the genomic region being evaluated and all possible combinations of varying-sized subwindows are scored. Fig. 2 shows how OmegaPlus evaluates multiple positions with varying windows along the input dataset.

The computation of the maximum ω-statistic at the centre of the genomic region is implemented using a nested for loop. The number of iterations increases with two factors:

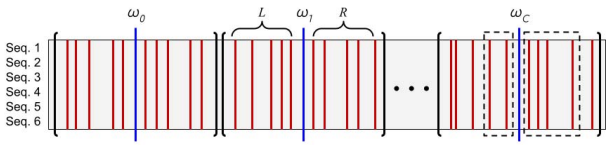

Fig. 2. Multiple ω-statistic computations along a genome, as implemented by OmegaPlus. The ω statistic is computed at equidistant positions ω_0, ω_1, ..., ω_c. For each ω position, a maximum window (in bp) is specified by the user (denoted by the vertical black lines). The ω statistic is computed using Equation 2 for all possible combinations of different-sized L and R subwindows, as shown by the dashed rectangles. The maximum ω corresponds to the ω position.

the genome length, with longer genomes requiring more ω-statistic locations to be evaluated, and the number of SNPs per region, which affects the number of ω-statistic values that need to be evaluated per genomic location to find the maximum value. Because finding a selective sweep relies on calculating the window length and combination of subwindows that maximizes the ω-statistic, OmegaPlus evaluates all possible window sizes using a sliding window algorithm to find the highest ω score.

All possible sums of correlation values (see Equation (2)) are computed using a dynamic programming algorithm described below:

$$M_{i,j} = \begin{cases} 0 & 1 \leq i \leq W, j = i \\ r_{ij}^2 & 2 \leq i \leq W, j = i-1 \\ M_{i,j+1} + M_{i-1,j} - & \\ M_{i-1,j+1} + r_{ij}^2 & 3 \leq i < W, i-1 > j \geq 0 \end{cases}, \tag{3}$$

OmegaPlus implements a data-reuse optimization: it reuses already calculated correlation values in matrix M when consecutive genomic regions partially overlap. This is achieved by relocating values in matrix M. Correlation values for sites in non-overlapping regions are computed for each new grid position and stored in M using the dynamic programming approach. An abstract view of the OmegaPlus workflow is shown in Fig. 3 (CPU side).

III. Related Work

This section reviews related work on accelerating LD computation and selective sweep detection with GPUs and FPGAs.

Binder et al. [17] present a portable framework for performing CPU-based SNP comparison algorithms on a GPU. Comparing SNPs is the core of LD calculations. The implementation of LD maps the BLIS [23] framework for high-performance dense linear algebra onto the GPU. The SNP-comparison framework is evaluated on a Nvidia TITAN V GPU, a GeForce GTX 980 GPU, and an AMD Radeon Vega GPU, and performance comparisons with a BLIS-based CPU implementation [24] are performed. The authors report that the GPU implementation is up to 7.8x faster than the CPU implementation on an Intel Xeon E5-2620v2 6-core CPU running at 2.10 GHz.

Theodoris et al. [18] present quickLD, a software for computing various LD statistics using a CPU or a GPU. quickLD extends the approach presented by Binder et al. [17] and handles large-scale datasets using a two-step process that separates parsing from processing to more flexibly perform computations between distant SNPs without increasing memory requirements. quickLD is compared with PLINK1.9 [25] on two different computing systems: a) a personal computer with an Intel Core i5-8300H 2.3 GHz CPU and a Nvidia GeForce GTX 1050-M GPU, and b) the Aris supercomputer (*https://hpc.grnet.gr/en/*) with two Intel Xeon E5-2660v3 2.6 GHz CPU and a Nvidia Tesla K40 GPU per node. Using datasets with up to 100,000 samples and 10,000 SNPs on the supercomputer, the authors report up to 29x faster processing than PLINK1.9 (20 threads).

Alachiotis et al. [19] present a FPGA accelerator for computing Pearson's correlation coefficient as a measure of LD. The hardware architecture is automatically generated based on a number of parameters that were used to explore the accelerator design space. A host CPU runs an iterative algorithm that schedules execution on the accelerator hardware based on the available number of accelerator instances on the FPGA. To evaluate performance, the proposed solution is mapped to a Xilinx Virtex-7 VX980T-2 FPGA with a clock frequency of 137Mhz. When compared with PLINK1.9 [25] running on a workstation with an Intel Xeon E5-2630 hexa-core 2.6 GHz CPU, the FPGA achieves 50x faster processing than 12 CPU threads, and 200x faster processing than 1 CPU thread.

Bozikas et al. [20] also implement the Pearson's correlation coefficient as a measure of LD. An accelerator architecture that supports any number of samples is presented and mapped to a system with four FPGAs. The authors observe that transferring SNPs to the FPGAs is limiting performance, and propose a memory layout that facilitates the parallel retrieval of SNPs through multiple memory controllers. The Convey HC-2ex platform with 4 Xilinx Virtex-6 LX760 FPGAs is used. When compared with PLINK1.9 running on an Intel Xeon E5-2630 CPU at 2.3 GHz, one FPGA is 4.7x faster than 12 CPU threads, whereas processing becomes up to 12.7x faster than 12 CPU threads when all 4 FPGAs are used.

Alachiotis et al. [26] proposed an integer-based method to detect selective sweeps on FPGAs. Observing that integer arithmetic and discrete operations are more efficiently implemented on reconfigurable logic than floating-point arithmetic, the authors devised a SNP-comparison-based method to achieve high performance on FPGAs, reporting up to 62x faster execution than OmegaPlus on a 20-core CPU. Note that the implemented method is inherently different than the actual operations performed by OmegaPlus, and as such, the reported performance improvement does not represent the actual performance potential of FPGAs over multicore CPUs.

To the best of the authors' knowledge, the present work is the first that explores the potential of GPUs and FPGAs in implementing the exact computations required by OmegaPlus for LD-based selective sweep detection using the ω statistic.

IV. GPU Implementation

Our acceleration efforts focus on the ω statistic. To perform selective sweep detection using the ω statistic, the computation of LD is also required. For this purpose, the work of Theodoris et al. [27] is adapted for computing LD as required by OmegaPlus. Thus, we also exploit here the fact that the computation of LD can be cast into Dense Linear Algebra (DLA) operations [24] using the BLIS [23] framework mapped onto the GPU presented by Binder et al. [17]. This approach computes LD based on a general matrix multiplication operation (GEMM).

A. Dynamic two-kernel deployment

Due to the non-uniform distribution of SNPs along the genome, the workload per grid position can vary considerably. To address this efficiently, we implemented two GPU kernels, one optimized for small computational loads (small number of SNPs, henceforth referred to as Kernel I) and one optimized for large computational loads (large number of SNPs, henceforth referred to as Kernel II). The two kernels are dynamically deployed per grid position based on the following workload threshold for the total number of required ω calculations for each position:

$$N_{thr} = N_{CU} * W_s * 32, \tag{4}$$

where N_{CU} is the number of Compute Units (CUs) or Streaming Multiprocessors (SMs) on the GPU, and W_s is the wavefront or warp size. This threshold allows for high hardware utilization for both kernels since 32 wavefronts/warps per CU/SM is the upper limit for optimal device occupancy that is specified by both AMD [28] and Nvidia [29]. Fig. 3 illustrates the design and task distribution of the GPU-accelerated OmegaPlus implementation to the CPU and the GPU.

B. Kernel I: Optimized for low computational loads

Kernel I is employed when the total number of ω-statistic computations for a specific grid position is lower than N_{thr}. This kernel effectively mimics the behaviour of the OmegaPlus sliding-window nested loop by each work-item performing a single ω-score computation. In the standard OmegaPlus source code, the inner loop traverses the right subgenomic region performing l iterations (l is the number of SNPs in the right region), while the outer loop traverses the left subgenomic region performing $W - l$ iterations (W is the total number of SNPs in the region surrounding the grid position). Due to the non-uniform SNP distribution, the left and right regions commonly exhibit a different number of SNPs. To improve performance, the subregion (per grid position) with most SNPs is always processed by the inner loop of the GPU kernel irrespective of its actual position in the genome. This dynamic sub-region order-switch optimization when processed by the GPU improves memory accessing by ensuring that the maximum possible number of coalesced memory accesses are performed for each grid position. Fig. 4 illustrates the processing model of Kernel I.

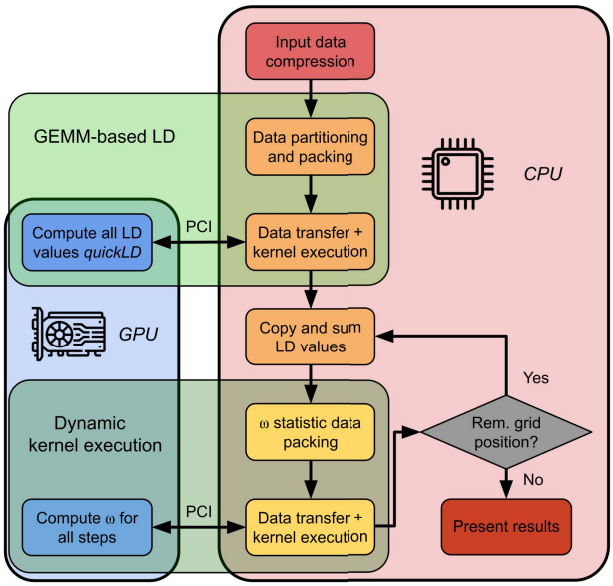

Fig. 3. The OmegaPlus workflow showing all computational steps: data-reuse optimization, LD computation, dynamic programming update of M, and ω-statistic computation. These steps repeat for every remaining grid position. This illustration explains the general design flow chart where the data is first compressed on the CPU. Then, during the GEMM-based LD implementation, data is partitioned and packed before transferred to the GPU for the execution of the BLIS kernel. After computing all LD sums, data is packed per grid position and transferred to the GPU for the ω-statistic kernel execution.

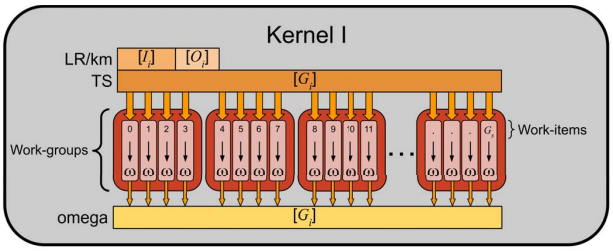

Fig. 4. Abstract view of the design of Kernel I. Each of the four work-groups contains four work-items that are scheduled on the Stream Processors (SPs)/CUDA cores in a CU/SM. The figure shows G_s work-items, which is equal to the number of computation steps and ω values that need to be computed for a given grid position. The figure makes the abstraction that the number of computation steps is an integer multiple of the work-group size, L_s. In reality, padding is applied to ensure that this is the case. Every work-item reads the required elements from the input sub-buffers in `LR` and `km` and from the input buffer `TS` to compute a single ω value. The computed ω values together are written to the `omega` buffer in a coalesced way.

C. Kernel II: Optimized for high computational loads

Kernel II is employed when the number of ω-statistic computations for a grid position exceeds N_{thr}. As previously described, Kernel I is optimized for grid positions with less than N_{thr} computational steps by exploiting the optimal occupancy limit on the GPU. Kernel II is designed to operate efficiently for more than N_{thr} calculations by

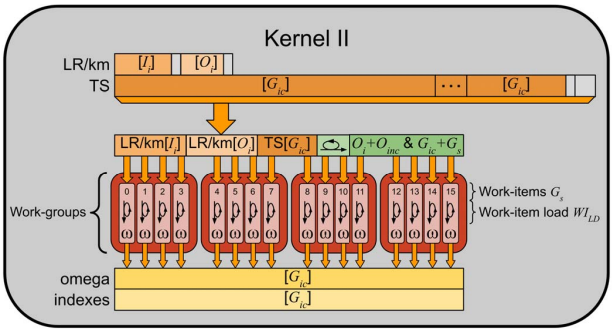

Fig. 5. Abstract view of the design of Kernel II. Each of the four work-groups contain four work-items that are scheduled on the SPs/CUDA cores in a CU/SM. 15 work-items are scheduled which can be seen as the set number of work-items G_s, which is near the indicative value set by the user. Padding of the input buffers is also shown, where `TS` is made up of WI_{LD} sections of the global size, G_s. The additional buffer shown above the work-groups indicates which elements are read to describe and accommodate the work-item load for multiple iterations. The green part of this buffer indicates the memory access pattern increment for every work-item load iteration. Every work-item reads a single value from `LR` and `km` using the inner loop index value I_i, and WI_{LD} values from `LR`, `km`, and `TS` using the outer loop index value, O_i, and the global index value, G_{ic}. Every work-item computes $WI_{LD} \times \omega$ values and the maximum ω and its global index are written in a coalesced way to the `omega` and `indexes` buffers, respectively.

scheduling multiple computational steps per work-item. The number of scheduled work-items per position (work-item load) is a variable. It is initialized with an empirically determined constant, and fluctuates around this value depending on the number of inner loop iterations along the grid positions. The work-item load is transferred to the GPU to determine the required number of for-loop iterations in the kernel. Loop unrolling is applied using the OpenCL `unroll #pragma` directive, with an unroll factor of 4 (empirically determined).

Due to the far higher number of computational steps performed by Kernel II, the memory access pattern has a more profound effect on performance. All data buffers transferred to the GPU are padded to a size that is multiple of the work-group size. This increases the data transfer time, which, however, is outweighed by the increase of performance due to the better memory access pattern of the input buffers. Fig. 5 illustrates work-groups and work-items accessing memory and executing a nested loop each based on Kernel II.

V. FPGA Accelerator

For the FPGA, the ω calculation is described and optimized using high-level synthesis with Vivado HLS version 2020.1. In order to adjust the amount of hardware resource being utilized by the accelerator to increase/decrease parallelism based on the available resources, the inner for-loop of the sliding-window nested loop is partially unrolled according to a specified unroll factor as shown in Fig. 6. Upon synthesis, a number of accelerator instances equal to the unroll factor are placed in the reconfigurable logic. This allows us to resize the accelerator and adapt resource utilization and parallelism/performance based on the target FPGA. Expectedly, the unroll factor does

```
//left side loop
for (i = leftMaxIndex; i > leftMinIndex; i--)
{
    //right side loop
    for (j = rightMinIndex; j < rightMaxIndex; j++)
    {
        //Calculation using i and j
        AcceleratorPipeline(i, j);
    }
}

//left side loop
for (i = leftMaxIndex; i >= leftMinIndex; i--)
{
    //right side loop
    for (j = rightMinIndex; j < rightMaxIndex; j += UNROLLFACTOR)
    {
        //unroll loop
        for (u = 0; u < UNROLLFACTOR; u++)
        {
            //Calculation using i and (j+u)
            AcceleratorPipeline(i, (j+u));
        }
    }
}
```

Fig. 6. Pseudo code of the nested for-loop implemented by OmegaPlus to process one ω position. The inner-most loop (right-side loop) is unrolled as part of the preparation for high-level synthesis of the hardware accelerator.

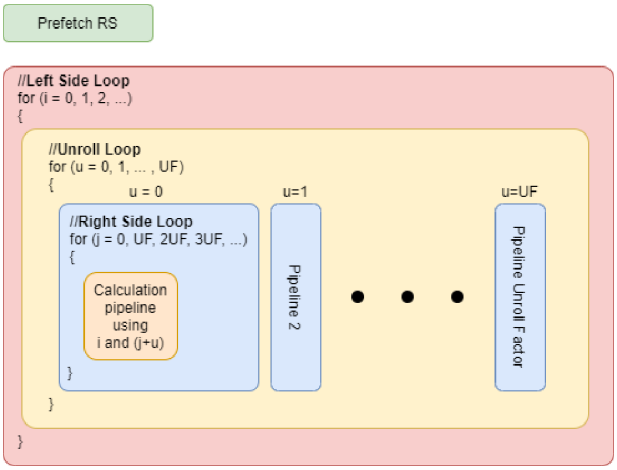

Fig. 7. Overview of the accelerator design with the execution of the multiple loops and the pipeline mapping on the FPGA hardware.

not always divide exactly the number of right side loop iterations. The remaining iterations are executed in software. In the architecture description, the order of the right side loop and the unroll loop is switched. This allows the inner loop to be fully pipelined with initiation interval of one clock cycle, thereby computing an ω score per clock cycle. Fig. 7 depicts the loop reordering and shows the parallel placement/execution of accelerator pipelines on the FPGA hardware.

Fig. 8 depicts the custom accelerator pipeline that is fully pipelined and computes a single ω value per clock cycle for each grid location. To practically approach the maximum theoretical performance of the accelerator pipeline, i.e., one

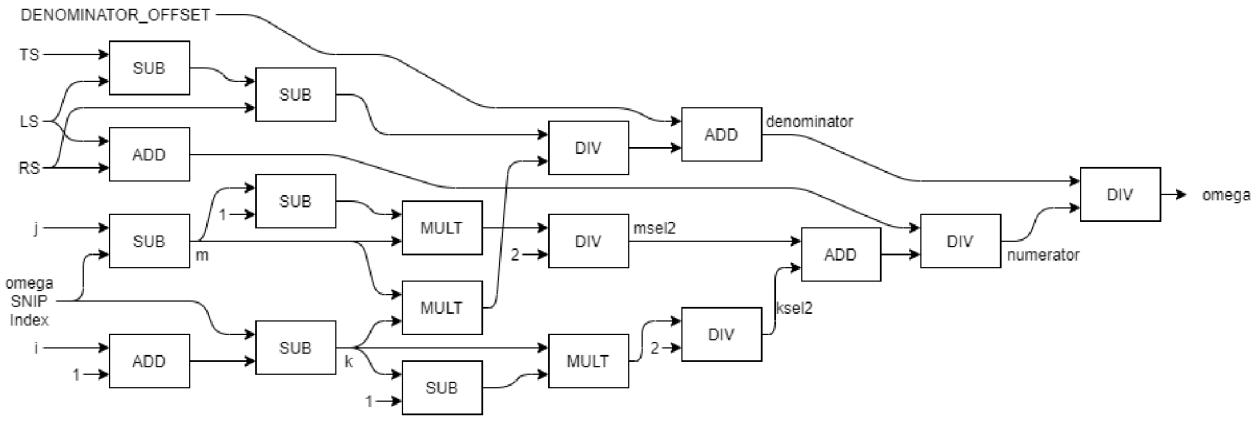

Fig. 8. Block diagram of the custom processing pipeline for the computation of the ω statistic. It produces one ω score per clock cycle.

ω score per clock cycle, input data need to be provided at the same rate. For this purpose, the memory layout of OmegaPlus was appropriately transformed to allow values TS, LS, and RS (see Fig. 8), which correspond to sums $\sum_{i\in L, j\in R} r_{ij}^2$, $\sum_{i,j\in L} r_{ij}^2$, and $\sum_{i,j\in R} r_{ij}^2$, respectively, to be efficiently fetched from matrix M that resides in external memory. Figure 9 demonstrates how matrix M is accessed. Indices i and j select a column and a row in M, respectively. Assume processing is at a certain i and j that is denoted by the dark red circle in Fig. 9. The next iteration of j moves down one row to the lighter red circle. The next iteration of i moves one column to the left, i.e., from the red cicle to the yellow one. This means that for every i, the accelerator needs a new array of TS values. It can be observed, however, that the blue column remains the same when i changes. Therefore, to optimize memory accessing, RS values can be prefetched and reused throughout multiple iterations of i. Furthermore, we store matrix M in a column-major order since we need two columns per iteration of i.

VI. Performance Evaluation

A. Experimental setup

In this section, we evaluate the performance of the proposed solutions. We initially describe the experimental setup, which is different for each accelerator technology. We employ throughput as a common performance metric for both accelerators to facilitate a direct comparison between the two processing architectures. Design choices differ per platform: the FPGA implementation unrolls and accelerates the innermost nested for-loop, while the GPU implementation accelerates the entire nested for-loop. To obtain insights into the performance potential of each solution, we measure throughput with respect to the factor/parameter that most profoundly affects performance of each implementation, i.e., the unroll factor for the FPGA, and the number of SNPs (workload size) for the GPU. We generated simulated datasets using Hudson's ms (Hudson [30]).

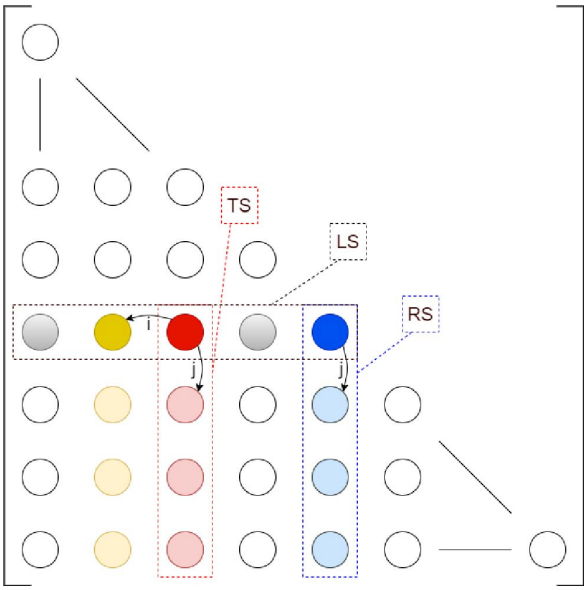

Fig. 9. Explanation on how matrix M is accessed by OmegaPlus to calculate all possible ω scores at one ω position along the genome.

FPGA: Because the FPGA solution is focused on accelerating the innermost for-loop, the number of iterations of the outermost for-loop does not affect performance. We therefore vary only the total number of innermost iterations, which we henceforth refer to as the right-side loop. Two FPGAs have been targeted for performance evaluation: the Zynq UltraScale+ ZCU102 Evaluation Board, which is a development board for embedded solutions, and the considerably more powerful Alveo U200 Data Center Accelerator Card. The unroll factors that allow the accelerators to utilize the available bandwidth of each target platform are 4 for the ZCU102 and 32 for the Alveo U200. The FPGA design is evaluated with ms datasets that required up to 4,500 right-side loop iterations on the ZCU102, and up to 30,500 right-side loop iterations on

TABLE I
RESOURCE UTILIZATION OF THE FPGA ACCELERATORS ON THE ZCU102 AND THE ALVEO U200.

	System I: ZCU102	**System II: Alveo U200**
Description	Zynq UltraScale+	Alveo U200
Logic Cells (k)	600	892
Unroll Factor	4	32
BRAM_8K	36/1824 (1.97%)	40/4320 (0.93%)
DSP48E	48/2520 (1.90%)	215/6840 (3.14%)
FF	12003/0.55$\times 10^6$ (2.19%)	50841/2.4$\times 10^6$ (2.15%)
LUT	12847/0.27$\times 10^6$ (4.69%)	50584/1.2$\times 10^6$ (4.28%)
Frequency	100 MHz	250 Mhz

TABLE II
OVERVIEW OF THE PLATFORM SPECIFICATIONS OF THE GPU SYSTEMS (* THE CORRESPONDING DEVICE HAS MORE CORES/THREADS AVAILABLE BUT THE NUMBER IS RESTRICTED WITHIN GOOGLE COLABORATORY).

	System I	**System II**
Description	off-the-shelf laptop	Google Colab
CPU Model	AMD A10-5757M	Intel Xeon E5-2699 v3
Base Freq.	2.5 GHz	2.3 GHz
Cores/Processor	4	2*
Threads/Processor	1	1*
GPU Model	Radeon HD8750M	NVIDIA Tesla K80
Compute Units	6	13
Stream Processors	384	2496

the Alveo. Tab. I provides resource utilization of the hardware accelerators on the target devices.

GPU: The workload of the GPU solution is affected by the total number of iterations in both the outer and inner for-loops. To evaluate GPU performance, we used datasets with a varying number of SNPs and a constant number of sequences, since the amount of ω calculations does not depend on the sample size. The simulated datasets consist of 50 sequences and a varying number of SNPs from 1,000 to 20,000. The ω-statistic was computed at 1,000 equidistant locations along the datasets (grid size), with a maximum window size of 20,000 SNPs and minimum window size of 1,000 SNPs. The grid size input argument corresponds to realistic analyses that typically evaluate thousands of positions along the genome [14], while the minimum and maximum window sizes allow to exhaustively analyze every grid position. Tab. II provides the specification of the evaluated GPU platforms, i.e., a desktop GPU and a datacenter GPU.

B. FPGA accelerator performance

Figs. 10 and 11 show throughput performance for the ZCU102 and the Alveo U200 card, respectively, for the maximum unroll factor per device. It can be observed that, on both platforms, the throughput approaches a maximum value based on the number of right-side loop iterations. The accelerator performs considerably worse in runs with a small number of iterations in the right-side loop, while performance increases with an increasing number of iterations. The horizontal dashed lines in both figures correspond to 90% of the theoretical maximum throughput of each design. This theoretical maximum is the ideal execution scenario where an infinitely large number of iterations are performed, thereby

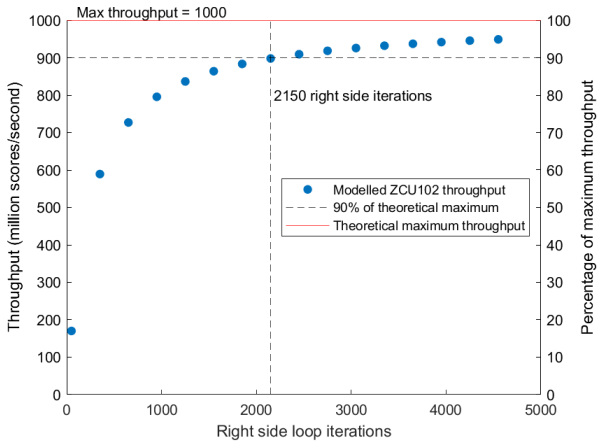

Fig. 10. Throughput as a function of right side loop iterations for the ZCU102 targeted implementation

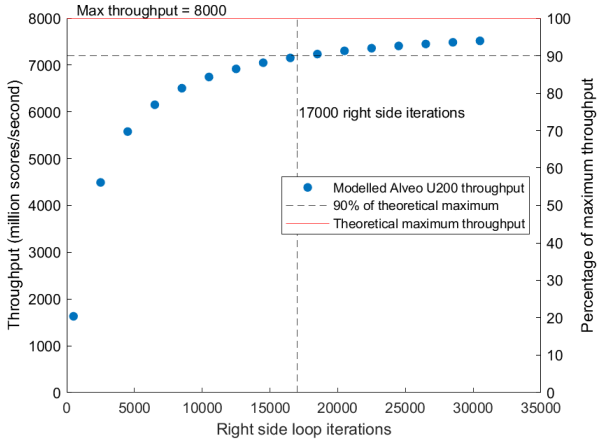

Fig. 11. Throughput as a function of right side loop iterations for the Alveo U200 targeted implementation

yielding the pipeline latency negligible. With a sufficiently large number of iterations for the right-side loop, the pipeline latency will be considerably lower than the number of clock cycles required to produce all ω scores. An accelerator pipeline computes one output score per clock cycle. In practice, this maximum theoretical throughput cannot be reached because of the latency of the pipeline and the time required to prefetch data (see Fig. 7, buffer RS). The 90% lines represent a realistic average-case operational region for the accelerator.

C. GPU performance

Fig. 12 shows the attained throughput performance of the GPU kernels. Datasets with a relatively low number of SNPs ($\leq$ 2,000) require a low workload per grid position and show the effectiveness of kernel I. With 1,000 SNPs, kernel I is 10% faster than kernel II on both systems. When the number of SNPs increase, however, kernel I shows a plateau at around

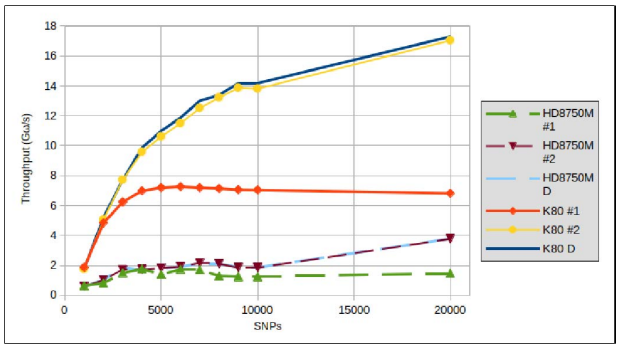

Fig. 12. Throughput values (Gω/s) of the GPU kernels on the test systems for increasing number of SNPs (from 1,000 to 20,000) and 50 sequences/samples. #1 and #2 denote the two GPU kernels (one optimized for low computational loads and one optimized for high computational loads), while *D* indicates the dynamic two-kernel deployment.

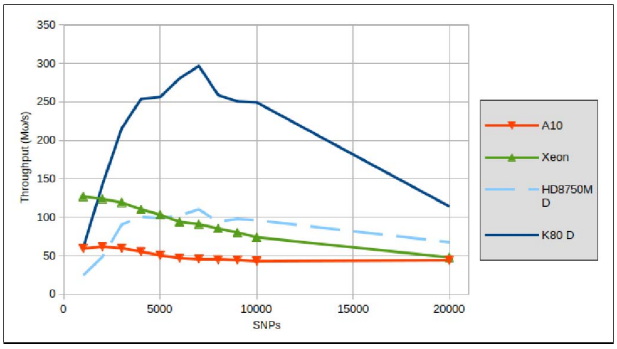

Fig. 13. Throughput values (Mω/s) of the complete GPU-accelerated ω-statistic computation on both systems and architectures using the dynamic kernel for increasing number of SNPs (from 1,000 to 20,000) and 50 sequences/samples. #1 and #2 denote the two GPU kernels (one optimized for low computational loads and one optimized for high computational loads), while *D* indicates the dynamic two-kernel deployment.

7 Gω/s (i.e., 10^9 ω-score calculations per second). For datasets with a higher number of SNPs and a higher workload per grid position, the effectiveness of kernel II cab be observed. The dynamic kernel-switching optimization performs similarly to kernel II because only a small number of grid positions exhibit a low number of computation steps and thus workload. The dynamic kernel execution on the K80 is up to 14% faster than kernel II alone. When the number of SNPs varies from 2,000 to 20,000 SNPs, the dynamic kernel execution is 1.25x to 2.59x and 1.08x to 2.54x faster than kernel I on Systems I and II, respectively. Kernel II on the K80 datacenter GPU delivers up to 17.3 Gω/s.

Fig. 13 shows throughput performance of the complete GPU-accelerated ω-statistic computation, i.e., including data preparation and data movement between the host to the GPU card. As can be observed in the figure, despite the fact that the throughput values of kernel II and the dynamic kernel scheme increase with the number of SNPs on both systems, the attained throughput of the complete GPU-accelerated ω-statistic computation decreases when the number of SNPs exceeds 7,000. This is due to an increase in data preparation and data movement overheads caused by the number of SNPs per grid position (larger buffers initialized and transferred per kernel call).

D. Performance of selective sweep detection

This section presents execution time and throughput comparisons of OmegaPlus implemented on a CPU, an FPGA, and a GPU. The total execution time of OmegaPlus depends on the proportion of LD and ω computations. The LD workload increases with the number of sequences while it remains mostly unaffected by the number of SNPs due to the data-reuse optimization. The ω workload on the other hand increases only with the number of SNPs. Therefore, our performance evaluation for the complete sweep detection process is based on three dataset sizes to show the average-case and the two extremes in terms of execution time distribution to LD and ω computation.

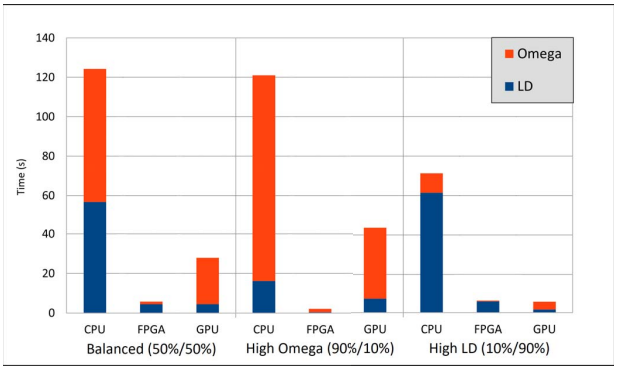

Fig. 14. Execution time distributions of LD and ω-statistic computation for balanced, high ω-statistic, and high LD workloads comparing the CPU, GPU, and FPGA implementations. The GPU execution times include data preprocessing, packing, and data transfer through PCIe communication. Note also that part of the data movement overhead is hidden by overlapping data transfers with kernel execution on the GPU.

For LD, execution times and throughput of the CPU and BLIS-based GEMM GPU implementations are directly measured by running the CPU and GPU versions of OmegaPlus. For the FPGA, performance numbers reported by Bozikas et al. [20] are used to provide an accurate estimate of the overall performance. For the ω statistic, performance numbers are directly measured by executing the CPU and GPU implementations of OmegaPlus, and extracted from post-place-and-route cycle accurate simulations for the FPGA architecture. As already briefly described, to obtain a realistic and complete insight into the performances of all the platforms and their implementations, three different workloads distributions are evaluated with respect to the execution time of LD and the ω-statistic computation on the CPU. The evaluated distributions are: a) a balanced workload ($\approx$50%/50%), b) a high ω-statistic workload ($\approx$90%), and c) a high LD workload ($\approx$90%). For the balanced workload, a dataset with 13,000 SNPs and 7,000 sequences is used. For the high ω-statistic workload, a dataset

TABLE III
THROUGHPUT COMPARISON AND SPEEDUP EVALUATION BETWEEN THE CPU, FPGA AND GPU PLATFORMS WITH THE PRESENTED SOLUTIONS AND DERIVED PERFORMANCE VALUES.

	Throughput (million scores/second)						Speedup to CPU			
Platf.	CPU		FPGA		GPU		FPGA		GPU	
Dist.	ω	LD	ω	LD	ω	LD	ω	LD	ω	LD
50/50	71.26	2.98	3500.00	38.20	206.72	37.14	49.1x	12.8x	2.9x	12.5x
90/10	60.76	13.91	3750.00	535.00	173.26	32.25	61.7x	38.5x	2.9x	2.3x
10/90	72.50	0.41	1500.00	4.50	181.10	15.84	20.7x	11.0x	2.5x	38.9x

TABLE IV
THROUGHPUT (ω SCORES $\times 10^6$ PER SECOND) OF THE GENERIC MULTITHREADED OMEGAPLUS FOR AN INCREASING NUMBER OF THREADS ON A 4-CORE INTEL CORE CPU.

Threads	**Throughput**
1	99.8
2	198.1
3	300.1
4	390.0
8	433.1

with 15,000 SNPs and 500 sequences is used. For the high LD workload, a dataset with 5,000 SNPs and 60,000 sequences is used.

Fig. 14 shows a comparison of the three platforms based on execution times. As can be observed in the figure, the FPGA solution achieves high performance for the ω-statistic computation and outperforms the GPU-accelerated ω-statistic computation. Both platforms however outperform the CPU implementation when considering the complete sweep detection process. The FPGA and GPU implementations achieve 21.4x and 4.5x faster execution than a CPU core for the balanced workload, 57.1x and 2.8x faster execution for the high ω-statistic workload, and 11.8x and 12.9x faster execution for the high LD workload. Overall, the FPGA system performs better with high ω-statistic workloads while the GPU implementation performs better with high LD workloads. Tab. III shows throughput performance of each for the three workload distributions, and the corresponding speedups of the FPGA and GPU platforms over a CPU core.

To evaluate FPGA and GPU performance versus a multi-core CPU, we initially used the generic multi-threaded version of OmegaPlus [31] on a 4-core AMD A10-5757M 2.5GHz processor. The multi-core CPU computes 4×10^6, 41×10^6, and 0.5×10^6 LD scores per second for the balanced, high ω, and high LD workloads, respectively. With all workloads, the multi-core implementation computes between 119×10^6 and 120×10^6 ω scores per second. Except for the high ω workload, which leads to unfavourable execution for both accelerator platforms due to the low number of LD computations, both the FPGA and GPU implementations outperform the 4-core AMD CPU. We further evaluated ω throughput performance of the generic multi-threaded version of OmegaPlus on a 4-core Intel Core i7-6700HQ 2.6GHz CPU, to explore performance for a higher number of threads using hyperthreading (Tab. IV).

Note that, due to the fact that the FPGA LD implementation of Bozikas et al. [20] is not publicly available for download, our sweep-detection FPGA system evaluation partially underestimates the memory access time for SNP data during LD computations. Our GPU evaluation on the other hand includes all data-access times for both ω and LD. Our results highlight the need for optimization of data preparation and data movement for the GPU-accelerated ω-statistic implementation. When comparing the throughput values of only GPU kernel and the FPGA pipeline, the GPU kernel is faster than the FPGA accelerator by 4.3x, 4.2x and 7.4x for the balanced, high ω and high LD workloads, respectively.

VII. CONCLUSION

In this paper, we explored the potential of modern GPU and FPGA platforms for the acceleration of LD-based selective sweep detection. We presented a custom processing pipelined for reconfigurable logic, which was mapped to an embedded FPGA platform and a datacenter-grade FPGA accelerator card. We also implemented two GPU kernels for varying ω-statistic computational loads and evaluated their performance on a desktop GPU and a datacenter-grade GPU. We evaluated our accelerated solutions with various dataset sizes to provide a general overview of the performance of each solution with respect to a reference CPU execution. We integrated our solutions with previous works that focused on accelerating LD, thereby providing complete acceleration of selective sweep detection as implemented by the open-source OmegaPlus. The FPGA and GPU implementations achieve 21.4x and 4.5x faster execution than a CPU core for balanced workloads, 57.1x and 2.8x faster execution for high ω workloads, and 11.8x and 12.9x faster execution for high LD workloads. Overall, the FPGA system performs better with high ω-statistic workloads while the GPU implementation performs better with high LD workloads.

As future work, we intend to optimize data movement for the GPU implementation to achieve even higher performance. We separately evaluated the potential of the two-kernel GPU implementation and observed that, despite being faster than a CPU core, a large fraction of the total execution time is spent on data transfers between the host and the GPU. Thus, we are going to explore algorithmic solutions in OmegaPlus to minimize these data transfers and further boost GPU performance.

REFERENCES

[1] J. M. Smith and J. Haigh, "The hitch-hiking effect of a favourable gene," *Genetics Research*, vol. 23, no. 1, pp. 23–35, 1974.

[2] T. Ohta, "The neutral theory is dead. the current significance and standing of neutral and nearly neutral theories," *BioEssays*, vol. 18, no. 8, pp. 673–677, 1996.

[3] N. G. De Groot and R. E. Bontrop, "The hiv-1 pandemic: does the selective sweep in chimpanzees mirror humankind's future?" *Retrovirology*, vol. 10, no. 1, pp. 1–15, 2013.

[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, M. D. Wilson *et al.*, "Selective sweeps and genetic lineages of plasmodium falciparum drug-resistant alleles in ghana," *Journal of Infectious Diseases*, vol. 203, no. 2, pp. 220–227, 2011.

[5] M. Vasilarou, N. Alachiotis, J. Garefalaki, A. Beloukas, and P. Pavlidis, "Population genomics insights into the first wave of covid-19," *Life*, vol. 11, no. 2, p. 129, 2021.

[6] L. Kang, G. He, A. K. Sharp, X. Wang, A. M. Brown, P. Michalak, and J. Weger-Lucarelli, "A selective sweep in the spike gene has driven sars-cov-2 human adaptation," *bioRxiv*, 2021.

[7] J. M. Braverman, R. R. Hudson, N. L. Kaplan, C. H. Langley, and W. Stephan, "The hitchhiking effect on the site frequency spectrum of dna polymorphisms." *Genetics*, vol. 140, no. 2, pp. 783–796, 1995.

[8] Y. Kim and R. Nielsen, "Linkage disequilibrium as a signature of selective sweeps," *Genetics*, vol. 167, no. 3, pp. 1513–1524, 2004.

[9] J. L. Crisci, Y.-P. Poh, S. Mahajan, and J. D. Jensen, "The impact of equilibrium assumptions on tests of selection," *Frontiers in genetics*, vol. 4, p. 235, 2013.

[10] J. L. Crisci, "On identifying signatures of positive selection in human populations: A dissertation," 2013.

[11] 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.

[12] B. F. Voight, S. Kudaravalli, X. Wen, and J. K. Pritchard, "A map of recent positive selection in the human genome," *PLoS biology*, vol. 4, no. 3, p. e72, 2006.

[13] R. Nielsen, S. Williamson, Y. Kim, M. J. Hubisz, A. G. Clark, and C. Bustamante, "Genomic scans for selective sweeps using snp data," *Genome research*, vol. 15, no. 11, pp. 1566–1575, 2005.

[14] P. Pavlidis, D. Živković, A. Stamatakis, and N. Alachiotis, "Sweed: likelihood-based detection of selective sweeps in thousands of genomes," *Molecular biology and evolution*, vol. 30, no. 9, pp. 2224–2234, 2013.

[15] E. W. Sayers, M. Cavanaugh, K. Clark, K. D. Pruitt, C. L. Schoch, S. T. Sherry, and I. Karsch-Mizrachi, "Genbank," *Nucleic acids research*, vol. 49, no. D1, pp. D92–D96, 2021.

[16] Y. Shu and J. McCauley, "Gisaid: Global initiative on sharing all influenza data–from vision to reality," *Eurosurveillance*, vol. 22, no. 13, p. 30494, 2017.

[17] 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 (IPDPSW)*. IEEE, 2019, pp. 199–208.

[18] C. Theodoris, T. M. Low, P. Pavlidis, and N. Alachiotis, "quickld: An efficient software for linkage disequilibrium analyses," *Molecular Ecology Resources*, vol. 21, no. 7, pp. 2580–2587, 2021.

[19] 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*, 2016, pp. 118–127.

[20] D. Bozikas, N. Alachiotis, P. Pavlidis, E. Sotiriades, and A. Dollas, "Deploying fpgas to future-proof genome-wide analyses based on linkage disequilibrium," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

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

[22] P. Pavlidis and N. Alachiotis, "A survey of methods and tools to detect recent and strong positive selection," *Journal of Biological Research-Thessaloniki*, vol. 24, no. 1, pp. 1–17, 2017.

[23] F. G. Van Zee and R. A. Van De Geijn, "Blis: A framework for rapidly instantiating blas functionality," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 3, pp. 1–33, 2015.

[24] 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)*. IEEE, 2016, pp. 418–427.

[25] 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*, vol. 4, no. 1, pp. s13 742–015, 2015.

[26] N. Alachiotis, C. Vatsolakis, G. Chrysos, and D. Pnevmatikatos, "Accelerated inference of positive selection on whole genomes," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 202–2027.

[27] C. Theodoris, N. Alachiotis, T. M. Low, and P. Pavlidis, "qld: High-performance computation of linkage disequilibrium on cpu and gpu," in *2020 IEEE 20th International Conference on Bioinformatics and Bioengineering (BIBE)*. IEEE, 2020, pp. 65–72.

[28] "OPENCL Optimization," OPENCL Optimization - ROCm Documentation 1.0.0, Accessed on: Oct. 3, 2021. [Online]. Available: https://rocmdocs.amd.com/en/latest/Programming_Guides/Opencl-optimization.html

[29] "CUDA C++ Best Practices Guide," CUDA Toolkit Documentation v11.4.2, Accessed on: Oct. 3, 2021. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html

[30] R. R. Hudson, "Generating samples under a wright–fisher neutral model of genetic variation," *Bioinformatics*, vol. 18, no. 2, pp. 337–338, 2002.

[31] N. Alachiotis and P. Pavlidis, "Scalable linkage-disequilibrium-based selective sweep detection: a performance guide," *GigaScience*, vol. 5, no. 1, pp. s13 742–016, 2016.