Accelerated Inference of Positive Selection on Whole Genomes

Nikolaos Alachiotis[1], Charalampos Vatsolakis[2], Grigorios Chrysos[2], and Dionisios Pnevmatikatos[1,2]

[1]Institute of Computer Science, Foundation for Research and Technology - Hellas, Heraklion 700 13, Greece
[2]School of Electrical and Computer Engineering, Technical University of Crete, Chania 731 00, Greece

Abstract*—Positive selection is the tendency of beneficial traits to increase in prevalence in a population. Its detection carries theoretical significance and has practical applications, from shedding light on the forces that drive adaptive evolution to identifying drug-resistant mutations in pathogens. With next-generation sequencing producing a plethora of genomic data for population genetic analyses, the increased computational complexity of existing methods and/or inefficient memory management hinders the efficient analysis of large-scale datasets. To this end, we devise a system-level solution that couples a generic out-of-core algorithm for parsing genomic data with a decoupled access/execute accelerator architecture, thereby providing a method-independent infrastructure for the rapid and scalable inference of positive selection. We employ a novel detection mechanism that mostly relies on integer arithmetic operations, which fit well to FPGA fabric, while yielding qualitatively superior results than current state-of-the-art methods. We deploy a high-end system that pairs Hybrid Memory Cube with a mid-range FPGA, forming a high-throughput streaming accelerator that achieves 751x, 62x, and 20x faster analyses of simulated genomes than the widely used software tools SweepFinder2 (1 thread), OmegaPlus (40 threads), and SweeD (40 threads), respectively. Importantly, our solution can scan thousands of human genomes and millions of genetic polymorphisms (1000 Genomes dataset, 5,008 samples) in a matter of hours, requiring between 4 and 22 minutes per autosome, depending on the chromosomal length.

***Keywords*-decoupled access/execute architecture; hardware accelerator; population genomics; selective sweep detection**

I. Introduction

Positive (or directional) selection is the most extensively studied form of selection, occurring when an allele (a variant of a gene) is favored by natural selection. The favored/beneficial allele increases in frequency over time and eventually becomes fixed, substituting completely the non-beneficial one in a population. In the process, genetic variation (mutations) in the neighborhood of the beneficial allele diminishes, creating a so-called selective sweep [1]. Genetic variation is typically observed in the form of single-nucleotide polymorphisms (SNPs), i.e., single base-pair changes in a DNA sequence. Thus, the sweep-induced reduction of genetic variation triggers the emergence of a subgenomic region that exhibits a reduced number of SNPs in comparison with the rest of the chromosome.

Propelled by advances in DNA sequencing technologies, genomic data are currently accumulated at an unprecedented pace, allowing for more accurate sweep detection analyses [2]. Large-scale sequencing efforts worldwide ([3], [4]) lead to ever-growing numbers of sequenced genomes from species of flora and fauna, enabling the discovery of new genetic variants and consequently paving the way for advancements in personalized medicine and the discovery of rare diseases [5]. Whole-genome scans for selective sweeps lead to the design of more effective drug treatments [6] and reveal reasons for treatment failures [7]. Furthermore, they can shed light on the forces that drive adaptive evolution [8]. However, compute and memory demands rise abruptly, since the complexity of selective sweep detection increases linearly with the experimental sample size (number of sequences) and quadratically with the number of SNPs, yielding high-performance and memory-aware processing a prerequisite for future large-scale analyses.

To this end, we propose a system-level solution for inferring positive selection from large-scale SNP data that efficiently addresses both the need for high performance and the need for reduced memory requirements. The former is driven by a particular method-level insufficiency that current population genetics studies suboptimally overcome by analyzing a dataset multiple times under different assumptions (discussed in Sec. II). The latter derives from the fact that existing tools either allocate memory on an as-needed basis [9], driven by the dataset size, or pre-allocate a fraction of the total memory capacity of the system regardless of the amount of data to be processed [10]. Evidently, both approaches implicitly impose an upper bound on the size of the dataset that can be processed on a given workstation.

In this work, we make the following contributions:

- We describe an out-of-core (OoC) algorithm that parses SNP data from storage space in parameterized-size chunks. It employs a highly compact data representation scheme that relies on SNP patterns, which, in combination with the chunk-based parsing mechanism, yields the analysis of immensely large datasets feasible by allocating just a few MB of main memory. For instance, only 4 MB are required to analyze chromosome 1 of 2,504 humans (64 GB in VCF [11] file format).
- We implement a streaming algorithm for the recently introduced μ statistic [12], and extend the basic OoC data representation by storing metadata per SNP. This allows to reduce computations and eliminates the need for processing the same data more than once. The em-

ployed statistic yields qualitatively *superior* results and requires significantly fewer floating-point operations than widely used methods, such as SweepFinder [13], while mostly relying on integer arithmetic operations, which map well to FPGA fabric.

- We pair the OoC algorithm with the DAER (Decoupled Access/Execute Reconfigurable [14]) architecture, creating a method-independent accelerator infrastructure. To efficiently conduct sliding-window operations on streams of SNP data, we modify/improve DAER to support the interconnection of arbitrary numbers of access and execute units to form pipelines with varying degrees of parallelism per stage, as well as to allow one or more pairs of access/execute units per pipeline stage.

Implementing the μ statistic on the DAER architecture, and coupling the resulting hardware accelerator with the OoC algorithm forms a complete system to conduct sweep detection in a scalable and timely manner. Employing a high-end system that couples the high-bandwidth Hybrid Memory Cube with a mid-range Kintex Ultrascale FPGA, we observed 62 and 20 times faster processing than the parallel tools OmegaPlus [9] and SweeD [2], respectively, when 40 threads are launched on 20 CPU cores, and up to 751 times faster processing than the widely used, sequential software SweepFinder2 [15]. In addition, the proposed system can analyze the 2,504 human genomes of the 1000 Genomes project [4] (5,008 sequences due to ploidy, and approximately 78×10^6 SNPs excluding the sex chromosomes) in slightly longer than 4 hours, thereby demonstrating the capacity of our solution to efficiently analyze real-world datasets with thousands of whole genomes.

II. Positive Selection Inference

In the following section, we describe a selective sweep and its signatures, as well as the μ statistic [12].

A. Selective sweep and sweep signatures

As already mentioned, genetic variation is observed in the form of single-nucleotide polymorphisms (SNPs). A SNP results from one or more mutations at the same genetic location in a genome. Prior to an analysis, a multiple sequence alignment (MSA) is constructed, i.e., a two-dimensional matrix of S rows and l columns that comprises S DNA sequences (one per individual under investigation) of length l nucleotide bases, as shown below.

seq_1 ATC**A**TACC**C**CT-CCAA**C**TAGG**A**TT**C**C
seq_2 ATC**C**TACC**A**CTCCCAA**C**TAGG**C**TT**C**C
seq_3 ATC**A**TAC-**C**-TCCCAA**G**TAGG**T**TT**T**C

Thereafter, a SNP calling tool, e.g., SAMtools [11], is used to extract the SNPs from the MSA, facilitating that way the execution of selection inference tools since monomorphic sites are not informative for sweep detection. The above MSA example contains 5 SNPs, at columns 3, 8, 16, 21, and 24 (indexing starts at 0).

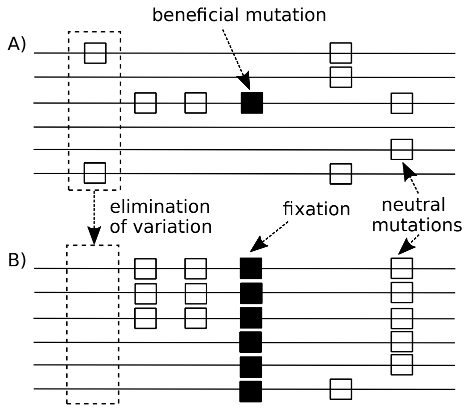

Figure 1. A population at two different junctures: A) when the beneficial mutation (solid square) first occurs, and B) after the beneficial mutation is fixed. Note the elimination of neutral variation due to the selective sweep.

A selective sweep is the reduction or elimination of polymorphisms/mutations in a subgenomic region as a result of positive selection. When a beneficial mutation occurs, its frequency increases in the population and eventually reaches fixation (all individuals carry the beneficial mutation). This is due to the fact that a beneficial mutation improves the chances of survival and reproduction for the individuals carrying it over the rest of the population. Due to the existence of additional evolutionary forces that form the genetic composition of a population, neighboring neutral variation (mutations that do not affect survival/reproduction) that is linked to the beneficial mutation diminishes in the proximity of the selected locus, thereby creating a selective sweep. Figure 1 illustrates two instances of the same population at different junctures: Figure 1A depicts the population when a beneficial mutation first occurs (prior to a sweep), while Figure 1B shows the expected reduction of polymorphisms when the beneficial mutation is fixed (after the sweep).

A selective sweep leaves three distinct signatures in genomes as a result of the aforementioned increase of the frequency of the beneficial mutation and additional evolutionary forces, facilitating the detection of loci that have undergone positive selection. Studies typically rely on a combination of sweep signatures (tests), under the rationale that the more tests agreeing on an outcome, the more likely the outcome. According to the selective sweep theory [1], the first signature is a localized reduction of the polymorphism level (genetic variation), which is observed as a subgenomic region with a reduced number of SNPs in comparison with the rest of the chromosome.

The second sweep signature is a particular shift in the site frequency spectrum (SFS) toward low- and high-frequency derived variants [16]. A SNP consists of two allele types, the derived and the ancestral. While the former is the result of a mutation, the latter is any allele that is not derived. A selective sweep therefore creates subgenomic regions with the majority of the SNPs comprising either a low number of derived alleles, e.g., one or two, or a high number of derived ones, e.g., $S-1$ or $S-2$, where S is the total number of sequences under investigation (sample size).

The third sweep signature is a localized pattern of linkage disequilibrium (LD) levels. LD is the non-random association between alleles at different loci, and is used to quantify the existence of associated alleles when the observed allele associations differ from what one would expect if the alleles were inherited independently. Kim and Nielsen [17] showed that increased LD levels (summed over all SNP pairs in a region) are observed on each side of a beneficial mutation, whereas the amount of LD between loci that are located on different sides of the beneficial allele remains low.

B. Detection method

The μ statistic [12] considers all three sweep signatures simultaneously, eliminating the need to analyze a dataset more than once, and consequently the need for the post-execution combination of the results. It detects sweep signatures by relying on the enumeration of occurrences of SNPs, rather than compute-intensive tests [13]. This alleviates the need for increased compute and memory capacity, which state-of-the-art software tools ([9], [15]) typically exhibit, and predominantly relies on integer arithmetic, allowing to reduce hardware resources and increase parallelism on an FPGA.

To compute the μ statistic, assume a two-dimensional $D_{sz} \times S$ MSA-like structure D that consists solely of SNPs, where D_{sz} is the number of SNPs and S is the sample size. Let D_{ln} be the length of the genomic region in nucleotide bases that corresponds to the D_{sz} SNPs, and W_{sz} be the size of a window W in SNPs. Additionally, let s_i denote SNP i in D, and l_i denote its location. The final statistic value is computed as follows:

$$\mu = D_{ln} \times (\mu^{\text{VAR}} \times \mu^{\text{SFS}} \times \mu^{\text{LD}}), \tag{1}$$

based on three factors, one for each sweep signature. Genetic variation in W is measured as follows:

$$\mu^{\text{VAR}} = \frac{l_{W_{sz}-1} - l_0}{D_{ln} \times W_{sz}}, \tag{2}$$

where l_0 and $l_{W_{sz}-1}$ are the locations of the first and the last SNPs in the window, respectively. Equation 2 assumes high values, indicating reduced genetic variation, when the W_{sz} SNPs in W correspond to a large genomic region.

The expected shift in the SFS is computed as follows:

$$\mu^{\text{SFS}} = \frac{\sum_{s_i \in W}[M(s_i) = 1] + \sum_{s_i \in W}[M(s_i) = S-1]}{W_{sz}}, \tag{3}$$

where [ ] is the Iverson bracket notation (it returns 1 if the logical proposition expressed by the statement in the brackets is true; otherwise returns 0) and $M(s_i)$ is the number of derived alleles at SNP s_i. Equation 3 assumes high values with an increased total number of SNPs that comprise one or $S-1$ derived alleles.

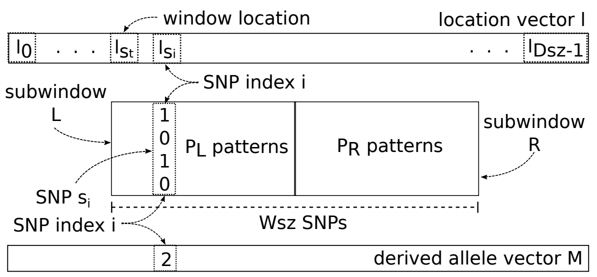

Figure 2. The window W is organized into two subwindows, L and R, to facilitate cross-LD evaluation based on the P_L and P_R SNP patterns.

The expected pattern of LD among SNPs is calculated based on the rationale that a reduced number of different SNPs is expected in a region of high LD, and vice versa. To account for the expected low LD across the beneficial mutation, assume that W is split into two subwindows, L and R, with $W_{sz}/2$ SNPs each, as illustrated in Figure 2. The third sweep signature is therefore captured as follows:

$$\mu^{\text{LD}} = \frac{\sum_{s_i \in W_L}[s_i \in P_L][s_i \notin P_R] + \sum_{s_i \in W_R}[s_i \notin P_L][s_i \in P_R]}{\sum_{s_i \in W_L}[s_i \in P_L] \times \sum_{s_i \in W_R}[s_i \in P_R]}, \tag{4}$$

where P_L and P_R are the sets of SNP patterns in the L and R subwindows, respectively. In Equation 4, the numerator assumes high values with an increasing number of exclusive SNP patterns per subwindow, while the denominator assumes low values with a reduced number of SNP patterns in each of the subwindows.

III. Related Work

In this section, we review selection inference approaches reported to scale to a large number of whole genomes and SNPs, and point out disadvantages that can potentially deem them to be unsuitable for future dataset sizes [3].

Nielsen et al. [13] released SweepFinder, an SFS-based software that implements a composite likelihood ratio (CLR) test where the numerator is the likelihood of a sweep and the denominator accounts for neutrality. It can analyze entire genomes while maintaining low memory requirements. Yet, it suffers from long execution times due to the fact that it heavily relies on floating-point operations and only employs a single core. Furthermore, SweepFinder is reported to be numerically unstable when the sample size exceeds 1,027 sequences [2], due to unhandled floating-point exceptions.

DeGiorgio et al. [15] recently released SweepFinder2, which employs the same statistical framework as SweepFinder and exhibits increased sensitivity. Particular code modifications yield a significantly more stable solution for computing likelihood scores, but performance is not improved as the software still deploys a single processing core. Pavlidis et al. [2] released SweeD, a software that also relies on the aforementioned statistical framework. SweeD, however, avoids the numerical stability issues via a modified set of mathematical operations that allow to correctly analyze thousands of whole genomes. Furthermore, it employs multiple CPU cores to reduce execution time. Yet, the excessive computational requirements, due to the implemented floating-point-intensive CLR test, remain.

Alachiotis et al. [9] proposed an efficient algorithm for the ω statistic [17], and released the parallel software OmegaPlus, which relies on the LD signature of a selective sweep. LD computations are the limiting factor for performance, due to the–typically–high number of pairwise calculations in whole-genome scans. Furthermore, OmegaPlus exhibits prohibitively large memory footprints because the entire dataset is loaded to memory prior to processing.

Bozikas et al. [18] described a multi-FPGA system that is capable of accurately computing LD on millions of sequences. This was achieved, however, by avoiding on-chip buffering of SNP data, inevitably yielding an I/O-bound architecture. Alachiotis and Weisz [19] presented an LD accelerator that relies on prefetching and the caching of entire SNPs on chip to deliver high performance, implicitly imposing an upper bound on the sample size. While both LD acceleration approaches employ FPGA technology to boost performance, LD alone can not serve as a test for neutrality. Thus, the aforementioned architectures can not be used as-is for positive selection inference.

To the best of the author's knowledge, the work presented here is the first to address the problem of inferring positive selection from SNP data at the hardware level, as well as the first to accelerate a Bioinformatics problem with a decoupled access/execute architecture.

IV. System Design

A. Out-of-Core (OoC) algorithm

The primary operation of the host processor is to fetch input data from storage space in parameterized-size chunks. This ensures that the memory footprint does not increase with the number of samples or SNPs, while the parameterized size allows to adapt the algorithm to various processor architectures and/or accelerator platforms. A SNP chunk of size 1 MB, for instance, can reduce cache misses, or allow to store an entire SNP chunk on FPGA on-chip memory throughout execution. More importantly, the described OoC approach is both application- and method-agnostic, and can be deployed to maintain the memory footprint of tools that process large-scale SNP data below a user-defined threshold.

The OoC algorithm is depicted in Figure 3. The input dataset is loaded from storage space on a SNP-by-SNP basis. SNPs that are stored in row-major order (VCF [11] format) are fetched with minimal parsing overhead, relying on an iterative routine that sequentially loads one byte/character per iteration until a complete SNP is parsed. When SNPs are stored in column-major order (ms [20] / FASTA formats), however, the next allele in the SNP is placed several bytes far from the one loaded last. The exact byte distance between same-SNP alleles depends on the file format and the number

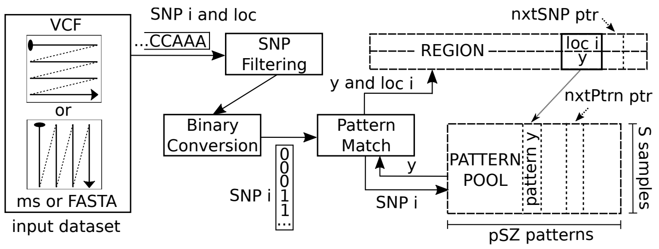

Figure 3. The OoC algorithm for parsing large-scale SNP data. Patterns are stored in their entirety in the `PATTERN POOL` data structure. An accurate description of the genomic region is maintained through the `REGION` structure, which comprises pointers to SNP patterns in the pool.

of SNPs. Hence, prior to loading the next character, the file position indicator needs to be set accordingly.

When an entire SNP is loaded, e.g., SNP i in Figure 3, a filtering step (`SNP Filtering`) applies a series of checks to verify correctness and quality. The verified SNP is then converted to a compact binary representation (`Binary Conversion`) that reduces memory footprint and facilitates processing. The allele encoding scheme is dictated by the mutation model, which can either be the Infinite Sites Model [21] or a Finite Sites Model [22]. The former assumes at most one mutation per site (1 bit per state), whereas the latter allows all possible DNA states at every site (4 bits per state). Upon conversion to binary, a pattern matching routine (`Pattern Match`) compares the verified SNP with all existing SNP vector patterns in the `PATTERN POOL` data structure. In the case that the entire pool is scanned and no matching pattern is found, the incoming SNP is stored in the pool as a new pattern. In any case, the pattern index y is stored along with the corresponding SNP location $loc\ i$ in the `REGION` data structure, which describes the genomic region to be processed. Both structures maintain a nxt pointer that indicates the next available position for a new entry. The `PATTERN POOL` data structure exhibits increased versatility and bounded memory allocation through a flexible management mechanism that adapts the pSZ number of patterns that the pool can accommodate to the desired memory footprint for the sample size S.

The OoC algorithm yields a generic solution that is applicable to a variety of SNP analyses. We observe, however, that memory requirements can be further reduced when the OoC algorithm is used for selective sweep detection without inducing an overhead. This is due to the fact that the number of derived alleles that is required by Equation 3 (see also Figure 2) is already calculated at the `SNP Filtering` stage, which enumerates the number of derived alleles per SNP as part of its correctness checks, and thus it can be stored as an additional field in the `REGION` structure.

B. Decoupled Access/Execute Reconfigurable architecture

The DAER [14] accelerator architecture (Figure 4) adopts a decoupled access/execute paradigm [23], [24] that facilitates system design, as well as code orchestration and development. It exhibits pairs of FETCH and PROCESS units that operate in tandem to implement a wide range of applications. A series of code preparation steps are required to decouple memory accesses from processing, and resolve dependencies. DAER resolves dependencies using a distributed memory-access scheme that serves requests concurrently through a network of FIFO-based interconnected FETCH units. Coupling the FETCH-unit network with multiple PROCESS units yields a dataflow engine that accommodates task-based applications with streaming and/or arbitrary access patterns.

FETCH units communicate with a host processor to load execution parameters and memory traces, as well as with external on-board memory to retrieve/store data. PROCESS units receive input from external memory through FIFO-based links, and implement the required logic and/or arithmetic operations. FIFO-based links additionally facilitate the passing of intermediate results between neighboring processing units in a pipeline, as well as the synchronization among PROCESS units operating concurrently. The aforementioned code preparation steps can be applied on any algorithm that is implemented in a high-level programming language. The source code is primarily translated to a dataflow description, followed by an annotation step that employs high-level synthesis directives for interfacing and hardware generation.

C. Application-specific DAER accelerator

Employing a decoupled access/execute paradigm yields a distinct separation between parsing SNP data and applying SNP-based statistics. The resulting accelerator architecture exhibits a well-defined interface between the OoC algorithm and the PROCESS units via FETCH-unit-controlled FIFO-based links. Thus, repurposing the entire system for a different SNP-based statistic only requires to adapt the PROCESS units, whereas the rest of the system remains largely intact.

Because of the modifications/improvements that we introduced in DAER, all three factors that account for the different sweep signatures (Eq. 2, 3, and 4) can now be evaluated in parallel by different PROCESS units since they exhibit no dependencies. This allows to exploit parallelism within a single window evaluation, and facilitates computations at subsequent locations of the sliding window through localized, per-signature data buffering, as described below.

Equation 2 is computed by `PROCESS T1`, which operates on the input data (window borders l_0 and l_{SZ-1}) fetched from the `REGION` data structure by `FETCH T1`. Similarly, the unit pair `FETCH T2` and `PROCESS T2` calculate Equation 3 by consecutively parsing the derived-allele vector in the `REGION` structure and updating internal counters per window shift to account for the SNP differences between neighboring windows, thus eliminating the need for window-wide summations (see numerator of Eq. 3). This is feasible due to the fact that the step of the sliding-window algorithm is 1 SNP, which limits the amount of on-chip memory that is required for buffering to a couple of registers (one for the removed SNP and one for the new SNP in the next window).

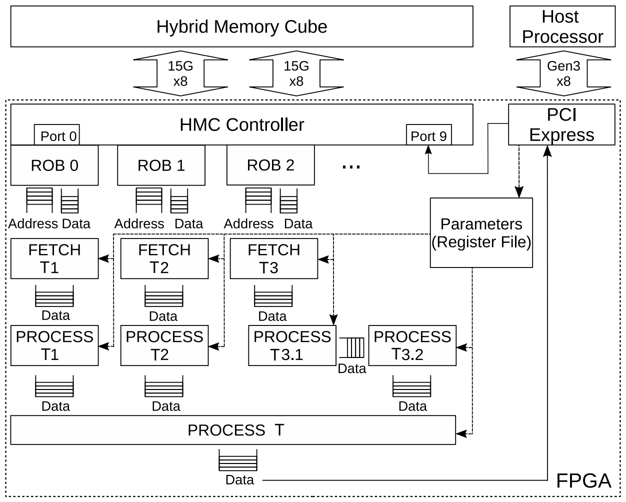

Figure 4. System overview. Compressed SNPs are initially accumulated on the HMC. These are then retrieved by FETCH units and forwarded to PROCESS units in streams. Results are directly transferred to the processor.

The third sweep signature, i.e., the emergence of specific LD patterns in a region, exhibits significantly higher complexity. This is due to the fact that Equation 4 relies on two non-overlapping subwindows to capture the expected low LD level across a beneficial mutation. While unit `FETCH T3` parses the vector of pattern indices in the `REGION` structure consecutively, the processing step requires on-chip memory blocks to capture the window shift and perform the required summations efficiently (see Eq. 4). For this reason, we organize the respective processing routine into two submodules, one to host the required data structures on on-chip memory blocks and perform the required updates in parallel (`PROCESS T3.1`), and one to conduct the required enumeration of common and exclusive patterns in the subwindows (`PROCESS T3.2`). The final statistic value (Eq. 1) is computed by `PROCESS T`. Note the absence of a corresponding FETCH unit in Figure 4, since no access to external memory or data permutations are required.

V. Implementation

Development: We initially developed a proof-of-concept C code that parses SNPs using the OoC algorithm, and computes μ statistic values. We applied a series of code transformations, dictated by DAER (Sec. IV-B), to prepare the code for hardware generation. A high-level synthesis tool (Xilinx Vivado HLS) was employed to obtain the hardware description for the custom accelerator.

Target platform: We used a Linux-based desktop PC, the SC6-mini by Pico Computing, for implementation and verification purposes. It exhibits an Intel Core i7-5930K 6-core CPU running at 3.5GHz (host processor), 32GB of DDR4 main memory, and an AC-510 SuperProcessor module with 4GB of Hybrid Memory Cube (HMC) and

Table I
RESOURCE UTILIZATION AND PERFORMANCE OF THE FULL SYSTEM WITH ONE AND THREE OPTIMIZED ACCELERATOR INSTANCES.

Resources (Total)	Accelerator (ACC)		Full System (Opt. ACC)	
/ Performance	Unopt.	Opt.	ACC $\times$ 1	ACC $\times$ 3
LUTs (331,680)	4.46%	7.84%	25.86%	41.79%
FFs (663,360)	3.47%	4.59%	20.23%	30.93%
BRAMs(1,080)	2.69%	1.85%	23.19%	33.47%
DSPs (2,760)	0.58%	0.58%	0.58%	1.74%
I/O BW (MB/sec)	125	214	214	642
Latency (clk cycles)	124	187	187	187
Throughput (res/clk)	1/24	1/14	1/14	3/14

a Xilinx Kintex UltraScale XCVU060 FPGA (hardware platform). The maximum theoretical bandwidth that the HMC interface exposes to the user logic on the FPGA is 30GB/sec per direction, through 10$\times$ 128-bit-wide memory ports operating at 187.5MHz. Communication between the host processor and the FPGA, and subsequently the HMC, is achieved through PCI Express, occupying one of the HMC ports, as depicted in Figure 4. Reorder buffers (ROB), implemented in Verilog HDL, are inserted between HMC ports and FETCH units to efficiently perform data reordering and word trimming since the HMC controller serves requests out of order whereas DAER FETCH units issue requests and expect data in order.

Performance system: Table I provides resource utilization and performance (datapath latency in clock cycles, and throughput in results/cycle) for two accelerator instances: a) the initial/unoptimized accelerator, which was derived directly from a straightforward application of the DAER-specific code preparation steps to the aforementioned proof-of-concept software implementation, and b) a performance-tuned/optimized accelerator instance, using particular HLS-based code modifications, as discussed in the following paragraphs. The table also provides resource utilization and performance for the complete FPGA system, i.e., including the supportive infrastructure (controllers) for the HMC and PCI Express, with 1 and 3 accelerator instances (ACC).

The initial accelerator occupied 5 HMC ports, thus preventing the instantiation of additional accelerators (only 9 HMC memory ports are available after occupying one for communication with the host processor). To reduce the number of required ports per instance, we assign one port to each FETCH unit and rely on in-unit SRAM memory blocks and additional control logic for performing necessary data permutations prior to providing input to the PROCESS units for the next window iteration. Address generation for-loops were fully unrolled, issuing a new read request per cycle, which is generally applicable for sliding-window applications with a fixed window width throughout execution. Loops in PROCESS units were also fully unrolled.

After reducing the number of HMC ports to 1 per FETCH unit, which now allows to instantiate up to 3 accelerator instances, computing Equation 4 required a series of for-loops with branches, introducing a latency overhead. To improve throughput, the for-loops in the respective PROCESS unit were rewritten as multiple inter-dependent and branch-free for-loops, leading to the optimized and final version of our custom accelerator architecture (see Table I, column 'Opt.'). In addition, we employed a dual-clock-frequency approach to further reduce latency of the aforementioned PROCESS unit, which was clocked at 250MHz, whereas the rest of the FETCH and PROCESS units operate at 187.5MHz. In DAER, communication between modules operating at different clock frequencies is established through dual-clocked, multi-ported FIFO-based links.

VI. Performance Evaluation

A. Experimental setup

To assess performance, we compare the FPGA-accelerated system with a series of state-of-the-art software tools: i) the parallel tool OmegaPlus [9], serving as the reference implementation for the LD-based sweep signature, and ii) the sequential tool SweepFinder2 [15] and the parallel tool SweeD [2], for the SFS-based one. As a test platform for the software benchmarks, we deploy a Dell PowerEdge R530 rack server with two 10-core Intel Xeon E5-2630v4 CPUs (20 threads per CPU) running at 2.2GHz (base), and 128GB of DDR4 main memory. For all execution scenarios, we report total execution times per software or hardware solution, including disk access, file parsing and SNP fetching, processing, and generation/storing of output reports in a file. Note that, although this does not represent a direct comparison of processing capacity between the involved technologies (multi-core CPUs versus FPGA+HMC), it provides a realistic notion of the expected performance and scalability gains from the deployment of a complete FPGA-accelerated solution in population genetics analyses.

Prior to the runs, we considered the deployment of HMC for boosting performance of the software tools through in-memory computations, since OmegaPlus relies on integer and bitwise operations that are supported by the latest HMC specification. However, they only operate on a single memory segment, which provides the first operand and serves as the destination for the result, whereas the second operand is always a constant. This does not serve the tool's required pairwise computations, since operations between different memory locations are not supported. SweepFinder2 and SweeD, which do operate on single SNPs, require floating-point operations that cannot be conducted in memory.

B. Evaluation on simulated data

We initially assess performance based on a series of simulated datasets with increasing sample size (20, 50, and 100 sequences) and a varying number of SNPs (in the order of some thousands[1]). We employ the ms [20] software tool

[1]The exact number is determined by the simulation software and depends on the evolutionary history of the population and the strength of selection.

Table II
EXECUTION TIMES (SEQUENTIAL) AND DETECTION ACCURACY. (SF2:SWEEPFINDER2, SD: SWEED, OP: OMEGAPLUS)

Sample	Execution times (secs)				Detection accuracy (%)			
size	SF2	SD	OP	DAER	SF2	SD	OP	DAER
20	24,571	14,721	17,807	32.7	29.7	29.8	48.8	62.6
50	29,814	14,733	27,567	61.7	42.3	42.5	67.0	74.1
100	61,905	15,758	41,144	104.8	45.8	46.1	75.6	75.9

for neutral simulations, and the mssel tool (kindly provided by R.R. Hudson) for datasets with selection. To resemble realistic execution scenarios, which typically require a large number of software invocations to conduct a statistical analysis and calculate a cut-off threshold to distinguish between selection and neutrality, each dataset comprises 1,000 neutral sets of SNPs, and 1,000 sets of SNPs with selection at the center of the genomic region they simulate.

Table II provides sequential execution times for the analysis of the 2,000 sets of SNPs per dataset using SweepFinder2 (SF2), SweeD (SD), OmegaPlus (OP), and the DAER-accelerated system. As can be observed in the table, DAER is up to 751, 450, and 544 times faster than SweepFinder2, SweeD, and OmegaPlus, respectively. The table also provides a qualitative comparison of the implemented methods, based on detection accuracy, which is measured as the percentage of reported sweep locations (best-score location per SNP set with selection) at a distance less than 1% of the size of the simulated genomic region. All runs assumed a genomic region size of 100,000 nucleotide bases. Thus, selection occurred at genomic location 50,000, with a detection outcome considered accurate if the best-score location is in the range [49,001-50,999]. For the dataset with a sample size of 20 sequences, for instance, DAER accurately reported 626 sweep locations (out of 1,000), whereas SF2, SD, and OP reported 297, 298, and 488, respectively.

When the aggregate processing capacity of modern processors is exploited by SweeD and OmegaPlus, through the deployment of multiple cores (none of the tools employ vector intrinsics), shorter overall execution times are achieved. Figure 5 illustrates how the tools scale (primary vertical axis) with an increasing number of threads (up to 40) on 20 CPU cores. Still, the FPGA-based solution remains 61.9 and 19.5 times (on average over all runs) faster than OmegaPlus and SweeD, respectively, as indicated by the bold continuous lines that provide a performance comparison between the DAER accelerator and the parallel execution of SweeD and OmegaPlus (note the secondary vertical axis).

Note that, all the software tools require a user-defined parameter that determines how exhaustively to scan the dataset. To yield comparable results, each tool evaluated 1,000 genomic locations. Evidently, the computational load increases proportionally with the number of evaluation points, introducing a trade-off between detection accuracy and analysis time, the examination of which is beyond the scope of this paper.

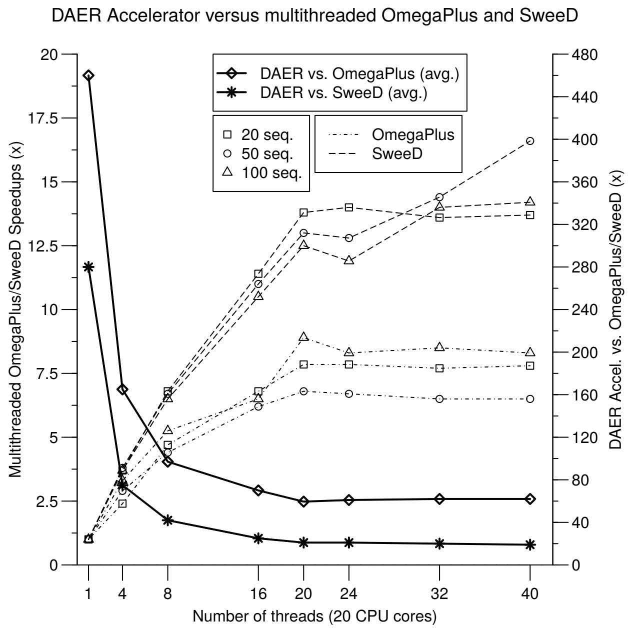

Figure 5. Performance evaluation of DAER, SweeD, and OmegaPlus based on datasets with 20, 50, and 100 sequences (2,000 sets of SNPs per dataset)

C. Scanning the human genome

To assess performance on real data, we also analyzed the human chromosomes 18 to 22, available by the 1000 Genomes project [4], which sequenced 2,504 genomes. Due to the fact that humans are diploid organisms, the experimental sample size for sweep detection is $2,504 \times 2 = 5,008$ sequences. Table III provides the number of SNPs per chromosome, total execution times for OmegaPlus and SweeD when 1 and 40 threads (20 cores) are used, as well as total execution times for the FPGA-based system and the respective speedups. SweepFinder2 failed to analyze the chromosomes due to the large sample size, terminating abruptly on a failing assertion in all execution attempts (SweepFinder2.c:738: get_pstar). The remaining software tools calculated scores at $10,000$ evaluation points, whereas the FPGA system evaluated two orders of magnitude more points along each chromosome. This is due to the sliding-window algorithm, which evaluates a SNP-dependent number of windows, X, with $X = SNPs - W\_sz + 1$, where W_sz is the window size, set to 10 SNPs for all runs. The number of calls to the accelerator (denoted 'HW calls' in Table III) demonstrates the out-of-core operation of the hardware-enhanced solution for the efficient processing of large-scale SNP data, and allows to maintain a negligible memory footprint of less than 2 MB for all chromosomes.

Given the high-quality SNP data that the 1000 Genomes project [4] made available, one can estimate the total required time to scan the entire human genome. Based on the observed execution times and the number of SNPs per chromosome, we caclulate that the FPGA system scans 5,189.9 SNPs per second when the sample size is 5,008

Table III
PERFORMANCE EVALUATION BASED ON THE ANALYSIS OF THE HUMAN AUTOSOMES 18 TO 22 FROM 2,504 INDIVIDUALS [4].

Chrom. Num.	18	19	20	21	22
Size (SNPs)	2,171,507	1,747,024	1,733,485	1,049,984	1,051,673
OP-1 thread	3h50m	3h27m	3h8m	2h1m	2h5m
OP-40 threads	32.2m	29.1m	26.8m	16.5m	17.9m
SD-1 thread	3h3m	2h38m	2h36m	1h53m	1h41m
SD-40 threads	31.5m	25.5m	25.7m	17m	16.5m
DAER Acc.	6.9m	5.6m	5.6m	3.4m	3.4m
HW calls	1,104	904	890	541	548
DAER vs. OP-1	33.3x	36.9x	33.6x	35.6x	36.8x
DAER vs. OP-40	4.7x	5.2x	4.8x	4.9x	5.3x
DAER vs. SD-1	26.5x	28.2x	27.9x	33.2x	29.7x
DAER vs. SD-40	4.5x	4.6x	4.6x	5x	4.9x

sequences. For the entire human genome, which comprises 77,832,252 SNPs in the 22 pairs of autosomes (we exclude the sex chromosomes in this calculation), we estimate that the FPGA system will take approximately 4 hours and 10 minutes, whereas the multithreaded execution of SweeD on 20 CPU cores (40 threads), for instance, will require 19 hours and 26 minutes, processing 1,112.1 SNPs per second.

VII. CONCLUSIONS

In this work, we explored the potential of FPGAs to efficiently conduct large-scale analyses of SNP data to detect traces of positive selection. We devised a memory-efficient out-of-core algorithm to parse SNP data and offload computations to an FPGA, and designed a high-throughput decoupled access/execute accelerator for sliding-window computations. We compared our complete system-level solution with state-of-the-art software implementations, observing up to two orders of magnitude faster execution for both simulated and real data. As future work, we intend to explore the potential of in-memory computations of the Hybrid Memory Cube for further performance improvements.

ACKNOWLEDGEMENT

This work was supported in part by EU H2020 FETHPC project EXTRA, contract #671653.

REFERENCES

[1] 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.

[2] P. Pavlidis *et al.*, "SweeD: likelihood-based detection of selective sweeps in thousands of genomes," *Molecular biology and evolution*, p. mst112, 2013.

[3] Genomics England, "The 100,000 genomes project," vol. 100, pp. 0–2, 2016.

[4] P. H. Sudmant *et al.*, "An integrated map of structural variation in 2,504 human genomes," *Nature*, vol. 526, no. 7571, p. 75, 2015.

[5] K. M. Boycott *et al.*, "Rare-disease genetics in the era of next-generation sequencing: discovery to translation," *Nature reviews. Genetics*, vol. 14, no. 10, p. 681, 2013.

[6] 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, p. 53, 2013.

[7] M. T. Alam *et al.*, "Selective sweeps and genetic lineages of Plasmodium falciparum drug-resistant alleles in Ghana," *J. of Infectious Diseases*, vol. 203, no. 2, pp. 220–227, 2011.

[8] 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.

[9] N. Alachiotis *et al.*, "OmegaPlus: a scalable tool for rapid detection of selective sweeps in whole-genome datasets," *Bioinf.*, vol. 28, no. 17, pp. 2274–2275, 2012.

[10] S. Purcell *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.

[11] H. Li *et al.*, "The sequence alignment/map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[12] N. Alachiotis and P. Pavlidis, "RAiSD detects positive selection based on multiple signatures of a selective sweep and SNP vectors," *Communications Biology*, vol. 1, no. 1, p. 79, 2018.

[13] R. Nielsen *et al.*, "Genomic scans for selective sweeps using SNP data," *Genome Research*, vol. 15, no. 11, pp. 1566–1575, Nov. 2005.

[14] G. Charitopoulos, C. Vatsolakis, G. Chrysos, and D. Pnevmatikatos, "A decoupled access-execute architecture for reconfigurable accelerators," in *Proceedings of the Computing Frontiers Conference*. ACM, 2018.

[15] M. DeGiorgio *et al.*, "Sweepfinder2: increased sensitivity, robustness and flexibility," *Bioinformatics*, vol. 32, no. 12, pp. 1895–1897, 2016.

[16] J. M. Braverman *et al.*, "The hitchhiking effect on the site frequency spectrum of DNA polymorphisms." *Genetics*, vol. 140, no. 2, pp. 783–96, Jun. 1995.

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

[18] D. Bozikas *et al.*, "Deploying FPGAs to Future-proof Genome-wide Analyses based on Linkage Disequilibrium," in *FPL2017*. IEEE, 2017, pp. 1–4.

[19] N. Alachiotis and G. Weisz, "High Performance Linkage Disequilibrium: FPGAs Hold the Key," in *ACM/SIGDA FPGA2016*. ACM, 2016, pp. 118–127.

[20] R. R. Hudson, "Generating samples under a Wright-Fisher neutral model of genetic variation." *Bioinformatics (Oxford, England)*, vol. 18, no. 2, pp. 337–8, 2002.

[21] 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.

[22] S. Tavaré, "Some probabilistic and statistical problems in the analysis of dna sequences," *Lectures on mathematics in the life sciences*, vol. 17, no. 2, pp. 57–86, 1986.

[23] J. E. Smith, "Decoupled access/execute computer architectures," in *ACM SIGARCH Computer Architecture News*, vol. 10, no. 3. IEEE Computer Society Press, 1982, pp. 112–119.

[24] T. Chen and G. E. Suh, "Efficient data supply for hardware accelerators with prefetching and access/execute decoupling," in *MICRO2016*. IEEE, 2016, pp. 1–12.