A Survey of Processing Systems for Phylogenetics and Population Genetics

REINOUT CORTS and NIKOLAOS ALACHIOTIS, University of Twente, The Netherlands

The COVID-19 pandemic brought Bioinformatics into the spotlight, revealing that several existing methods, algorithms, and tools were not well prepared to handle large amounts of genomic data efficiently. This led to prohibitively long execution times and the need to reduce the extent of analyses to obtain results in a reasonable amount of time. In this survey, we review available high-performance computing and hardware-accelerated systems based on FPGA and GPU technology. Optimized and hardware-accelerated systems can conduct more thorough analyses considerably faster than pure software implementations, allowing to reach important conclusions in a timely manner to drive scientific discoveries. We discuss the reasons that are currently hindering high-performance solutions from being widely deployed in real-world biological analyses and describe a research direction that can pave the way to enable this.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → **Special purpose systems**; **Parallel architectures**; **Reconfigurable computing**; • **Applied computing** → **Bioinformatics;**

Additional Key Words and Phrases: Phylogenetics, population genetics

ACM Reference format:
Reinout Corts and Nikolaos Alachiotis. 2023. A Survey of Processing Systems for Phylogenetics and Population Genetics. *ACM Trans. Reconfig. Technol. Syst.* 16, 3, Article 35 (June 2023), 27 pages.
https://doi.org/10.1145/3588033

1 INTRODUCTION

Bioinformatics research provides researchers in life and health sciences with the necessary toolset for processing molecular data to answer important biological questions, ranging from untangling the evolutionary history of organisms [11] to identifying clinically meaningful genes for cancer risk assessment, diagnosis, prognosis, and treatment [80]. Advances in DNA sequencing technologies in the past years, however, increased sequencing throughput while reducing sequencing costs, leading to an ever-increasing accumulation of whole genomes in private and public databases such as GenBank [79], GISAID [81], ENCODE [25], and the Cancer Genome Atlas [97]. As a result, studies today include a constantly growing number of organisms (sample size) in an effort to improve statistical power and thus accuracy and reliability of the outcome [20]. This has, inevitably, transformed Bioinformatics to a computational discipline that increasingly requires scalable algorithms and high-performance processing systems.

Authors' address: R. Corts and N. Alachiotis, University of Twente, Enschede, 7500, The Netherlands; emails: r.corts@student.utwente.nl, n.alachiotis@utwente.nl.

1936-7406/2023/06-ART35 $15.00
https://doi.org/10.1145/3588033

Furthermore, the COVID-19 pandemic brought Bioinformatics into the spotlight, exposing weaknesses in existing methods and tools with respect to handling large datasets efficiently. For instance, in April 2020, the authors of the widely used software MAFFT [53] for aligning multiple protein/DNA sequences released an experimental implementation only suitable for SARS-CoV-2. This implementation exploits certain characteristics of the viral genome (e.g., the high similarity between sequences due to the rapid spread of the virus) to approximate the alignment process in an effort to prevent prohibitively long processing times as the number of available SARS-CoV-2 genomes continue to increase rapidly. A phylogenetics study by Morel et al. [59] concludes that one of the two reasons why constructing the phylogenetic tree of 75,000 SARS-CoV-2 genomes is "difficult" is the excessive computational requirements due to the large number of sequences (the other reason is the small number of mutations due to the rapid spread of the virus). The phylogenetic analysis was restricted to just 5% of the parameter space, possibly due to the heightened urgency to reach conclusions because of the pandemic.

Over the years, the computing landscape has become highly heterogeneous, with modern platforms currently integrating multi-core CPUs with GPUs and/or FPGAs. Modern GPUs exhibit massively parallel computer architectures with thousands of processing cores. FPGAs, however, do not have a fixed processing architecture but can implement any specialized computer architecture by configuring and interconnecting millions of fine-grained logic elements. Exploiting hardware acceleration and **high-performance computing (HPC)** solutions in Bioinformatics is of paramount importance today more than ever, since HPC systems will enable faster and more thorough biological analyses.

Many widely used algorithms have been previously accelerated using advanced, low-level software solutions and/or hardware accelerators. Yet, despite their demonstrated performance potential, hardware accelerators are less frequently deployed in the field than general-purpose processors. To this end, this survey reviews literature on high-performance solutions targeting CPUs, GPUs, and FPGAs for important computational problems in Bioinformatics with the aim to shed light into the performance potential of these technologies. We also analyze the reasons that are currently hindering high-performance solutions from being actively deployed in real-world analyses and discuss a research direction that can pave the way to enable this.

Related surveys have previously focused mostly on sequence alignment [10, 44, 60]. Here, we discuss solutions that boost performance of widely used compute-intensive kernels in phylogenetics and population genetics, which can have direct, profound impact in future pandemics by enabling faster, more thorough analyses to detect diseases and design drugs. A study by Turakhia et al. [90], for instance, shows the potential of phylogenetics for genomic tracing, which can determine genetic similarities between viral genomes isolated from different hosts that carry information about the transmission dynamics of the virus. Such information can be used to determine, for instance, the number of unique introductions of the viral genome in a given area, thereby identifying transmission chains among seemingly unrelated infections. A study by Vasilarou et al. [93] elaborates on the significance of using population genetics to provide insights into the spread and epidemics of SARS-CoV-2. Population genetics offers the tools to estimate parameters such as the mutation and the recombination rate, which are important for understanding the evolution and thus the management of viral diseases. Kang et al. [48] report 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.

This survey focuses on widely used functions for scoring phylogenetics trees (the **phylogenetic parsimony function (PPF)** and the **phylogenetic likelihood function (PLF)**) and important association mapping measures in population genetics (**linkage disequilibrium (LD)** and epistasis). From a processing standpoint, the selected functions/kernels pose computational

challenges that are addressed more or less efficiently by each of the computer architectures considered here, thereby providing a representative range of performance potential of the processing solutions for Bionformatics problems. For example, both the PLF and epistasis computation rely on floating-point arithmetic that is more efficiently implemented on a GPU but is very resource costly on an FPGA. The PPF and LD computation, however, are based on discrete arithmetic that is more efficiently mapped to FPGAs than GPUs. The phylogenetic scoring functions benefit the most from deep custom pipelined architectures that can be implemented on FPGAs, whereas LD and epistasis computation are memory-bound and can better exploit the high memory bandwidth offered by GPUs.

The rest of the article is organized as follows: Section 2 describes the processing platforms (CPU, GPU, and FPGA). Section 3 focuses on phylogenetics and discusses solutions for the two main scoring functions of phylogenetic trees, i.e., the phylogenetic parsimony function (Section 3.1) and the phylogenetic likelihood function (Section 3.2). Section 4 focuses on population genetics and discusses solutions for the calculation of linkage disequilibrium (Section 4.1), pairwsie epistasis (Section 4.2), and third-order epistasis (Section 4.3). Section 5 reviews energy consumption, while Section 6 discusses performance of unconventional processing systems such as processing in memory and quantum computing. Section 7 provides a discussion and concludes this survey.

2 PROCESSING ARCHITECTURES

This section describes the processing platforms and their respective advantages and disadvantages.

CPU

A **Central Processing Unit (CPU)** has a fixed architecture designed to achieve high performance for the general case. This is achieved by implementing and using caches, on-chip memory, multiple cores/threads, instruction-level and data-level parallelism, among other optimization techniques. The utilization of caches, on-chip memory, and instruction-level parallelism is generally taken care of by the processor itself, whereas the utilization of multiple cores/threads and data-level parallelism has to be managed by the developer. OpenMP is an **Application Programming Interface (API)** that enables multi-threading on a CPU in which tasks can be divided over multiple threads that run on multiple cores. Data-level parallelism can be implemented using vector intrinsics such as the **SSE (Streaming SIMD Extensions)** and **AVX (Advanced Vector Extensions)** instructions, which enable **Single Instruction Multiple Data (SIMD)** execution. Figure 1 illustrates an example of the implementation and usage of a single **Arithmetic Logic Unit (ALU)** and a vector of ALUs that execute the same instruction (e.g., an addition) on multiple data elements in parallel, resulting in higher throughput performance.

GPU

A **Graphics Processing Unit (GPU)** also has a fixed architecture but differs from a CPU through the implementation of a massively parallel architecture. A CPU nowadays can contain tens of cores, whereas a GPU can consist of thousands of cores, which, however, have a more basic processing architecture and thus lower computing power than CPU cores. A GPU implements a **Single Instruction Multiple Threads (SIMT)** execution model that allows multiple threads to execute the same instruction on multiple data. This execution model effectively is a multi-threaded SIMD execution model. This limits the types of applications that can efficiently be mapped on a GPU architecture, since it is more suitable for applications that process large amounts of data with no data dependencies. GPUs have smaller cache memories than CPUs and require targeted optimizations to achieve high performance. The GPU processing cores are called **Stream Processors (SPs)** in

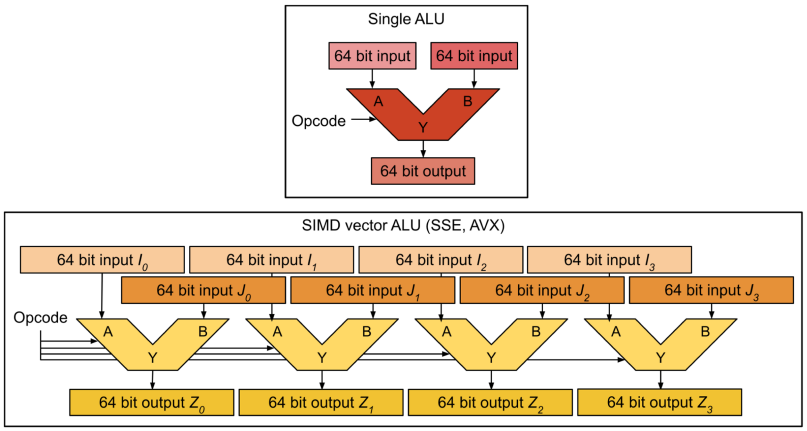

Fig. 1. Example of vector processing with a SIMD vector ALU as used with the SSE and AVX instruction sets.

AMD terminology and reside in SIMD Cores. All SPs in a SIMD Core perform the same instruction simultaneously on multiple data. A **Compute Unit (CU)** consists of several SIMD Cores. Figure 2 depicts an abstract view of the GPU architecture.

FPGA

A **Field-Programmable Gate Array (FPGA)** is an integrated circuit that consists of programmable logic hardware blocks and programmable interconnect, collectively called the fabric. It can be configured to implement application-specific computer architectures. The logic hardware blocks are called **Logic Elements (LEs)** and consist of various basic hardware elements that are combined upon device configuration to implement the desired functionality. The reconfigurable nature of the FPGA enables higher performance for specific applications types than CPUs and GPUs, but FPGA devices have limited on-chip resources and limited bandwidth that reduce performance. Figure 3 shows an abstract view of a generic FPGA architecture and a simplified view of a **Logic Element (LE)**.

3 PHYLOGENETICS

Phylogenetics reconstruct the evolutionary history of organisms (taxa) based on shared ancestry. After a **Multiple Sequence Alignment (MSA)** has been created, phylogenetic inference methods can construct a phylogenetic tree given an optimality criterion. Due to advances in DNA sequencing technologies, the field of phylogenetics is currently in need for accelerated solutions; tree reconstruction increasingly becomes more computationally intensive with an increasing number of taxa. The search for the maximum likelihood phylogenetic tree is NP-hard [23], which is why heuristics are inevitably employed to search the tree space for the optimal tree (given some optimality criterion).

Tree-scoring functions are used for evaluating phylogenetic trees. One method for this purpose is the **Phylogenetic Parsimony Function (PPF)** [26], which is a discrete function that aims to find the phylogenetic tree based on the least number of mutations among taxa. Another method for evaluating phylogenetic trees is the **Phylogenetic Likelihood Function (PLF)** [30]. The PLF is the most widely used method for tree construction and is employed in several phylogenetics tools like RAxML [85], GARLI [110], and MrBayes [74]. Yang and Rannala [103] provide a detailed review of the major methods and principles in phylogenetics.

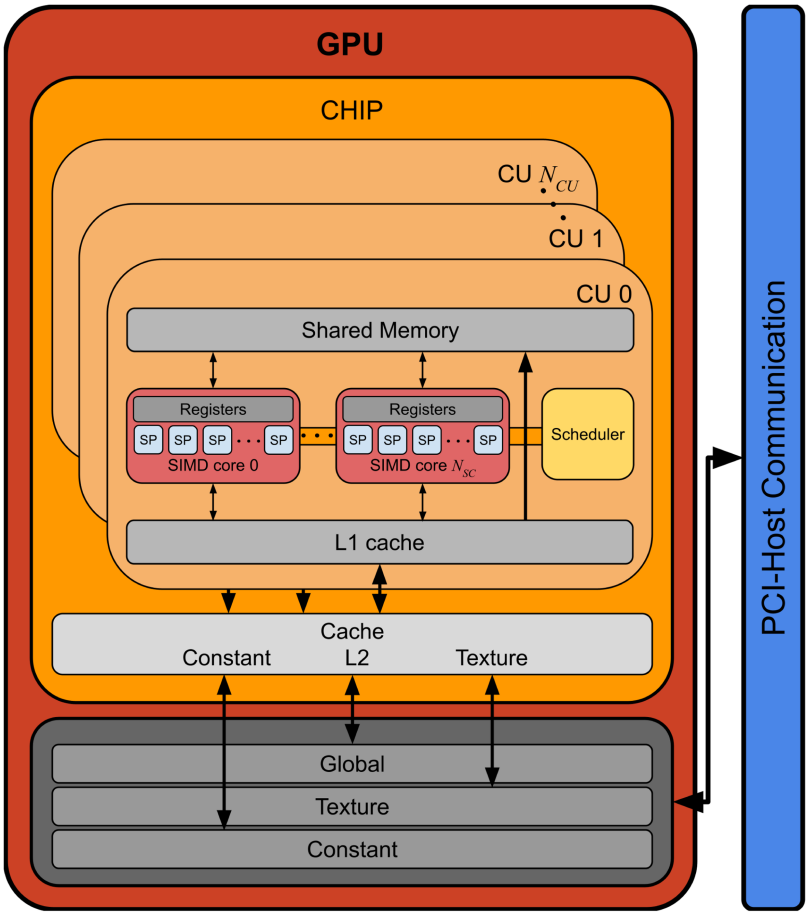

Fig. 2. Abstract view of the GPU architecture with several Stream Processors (SPs) on which threads are scheduled according to the SIMT execution model.

3.1 Phylogenetic Parsimony Function (PPF)

The PPF is used when the main objective is to find the tree topology that requires the least number of mutations to explain the data. Figure 4 depicts two possible topologies and the required mutations to explain the observed data at the tips; the tree that requires only one character change is the most parsimonious one. In comparison with the PLF, which relies on likelihood calculation (discussed in the next section), the PPF is considerably less compute- and memory-intensive [36] due to a simpler scoring function that produces an integer output representing the number of evolutionary changes. Because of this, the PPF is highly suitable for large-scale analyses. Various studies have already focused on optimizing the calculation of the PPF. Two algorithms can be used to implement the PPF, i.e., Fitch's algorithm [31] and Sankoff's algorithm [75], with Fitch's algorithm being more commonly used because of lower computational complexity [18].

Sankoff's algorithm is a dynamic programming algorithm that finds the most parsimonious topology for a given tree by counting the smallest number of possible (weighted) state changes. The algorithm initializes the leaves of the tree with the following scoring scheme:

$$s_v(k) = \begin{cases} 0 & \texttt{if leaf } v \texttt{ has state } k \\ \infty & \texttt{otherwise.} \end{cases} \tag{1}$$

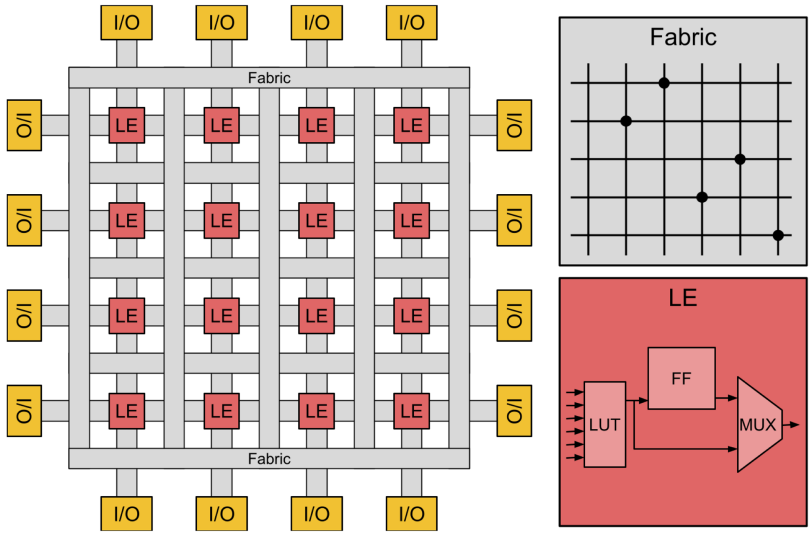

Fig. 3. Abstract view of the FPGA architecture with several LEs placed in the fabric, which can be configured for a specific application. Each LE consists of lookup tables (LUT), basic storage elements such as flip flops (FF), and multiplexers (MUX).

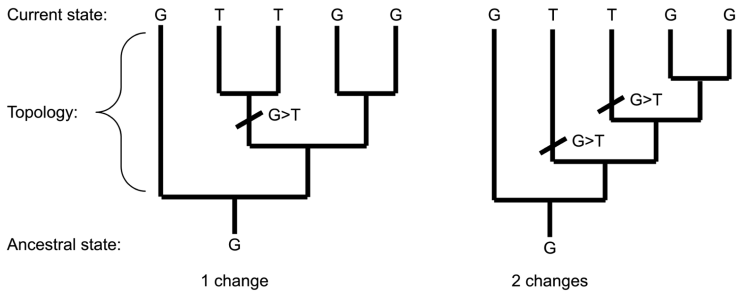

Fig. 4. Parsimony tree scoring visualization: Less changes in sequence data from the ancestral state to the current state result in a higher parsimony score and thus a more parsimonious topology.

Given an inner node v and its two children u and w, the score for v is computed using a $k \times k$ scoring matrix δ, with k being the number of possible characters per state and $\delta_{i,t}$ being the cost to go from state i to state t:

$$s_v(t) = \min_i\{s_u(i) + \delta_{i,t}\} + \min_j\{s_w(j) + \delta_{j,t}\}. \tag{2}$$

When all nodes are scored, a trace-back step is performed to find the state changes with the optimal cost and thus the most parsimonious topology.

Fitch's algorithm initializes each leaf l of the tree with the observed character, i.e., $S(l) =$ `observed character`. For every inner node v with children u and w, a set of $S(v)$ characters are assigned as follows:

$$S(v) = \begin{cases} S(u) \cap S(w) & \texttt{if non-empty intersection} \\ S(u) \cup S(w) & \texttt{otherwise.} \end{cases} \tag{3}$$

Then, the algorithm traverses from root to leaf. The root, r, is assigned an an arbitrary character from its set $S(r)$. The following nodes, v, are assigned:

- the character in the parent node if it is in its set $S(v)$
- an arbitrary letter from the set S(v) otherwise.

This will result in the tree topology with the least character changes.

CPU. Alachiotis and Stamatakis [7] employ an optimized, in-house parallel implementation, dubbed *parsimonator*, as the reference software for evaluating performance of a hardware accelerator (discussed in a following section). This reference software implementation deploys vector instructions such as the Intel SSE (128-bit streaming SIMD extensions) and AVX (256-bit advanced vector extensions) intrinsic instructions. The study shows (based on execution time comparisons) that the AVX implementation is 1.14× to 1.73× faster than the SSE implementation, yet between 5.6× and 9.65× slower than an FPGA accelerator.

Santander et al. [77] present an optimized solution for CPU, which is used for performance comparisons with their hardware-accelerated PPF implementation. The **software (SW)** implementation is parallelized and optimized using the OpenCL framework in combination with SIMD instructions. The solution is compared to an unoptimized serial implementation and a newer version of the *parsimonator* (1.0.2) (available at: https://cme.h-its.org/exelixis/web/software/parsimonator/index.html) tool that is parallelized using OpenMP. All performance values are acquired using two Intel Xeon E5-2630v3 CPUs. The optimized parallel solution is between 8.7× and 77× faster than the serial implementation and between 1.1× and 9.3× faster than the parallel AVX implementation of *parsimonator*.

Block and Maruyama [16] present a CPU-optimized solution developed using C++, which is used as reference for their hardware-accelerated PPF implementation (discussed in a following section). The C++ solution implements a local search algorithm that relies on the **Progressive Neighborhood (PN)** [33] search method. A PN search starts with a large neighborhood and takes more topologies into account to ensure a more intensive search. The C++ software solution is compared with the respective FPGA implementation and with TNT [37], a highly optimized tool for parsimony analysis. Using an Intel core i7-860 CPU running at 2.8 GHz as test platform, the software solution is considerably slower than both TNT and the FPGA accelerator, with TNT achieving, on average, comparable execution times with the FPGA-accelerated implementation (inferred from Block and Maruyama [16], Table 3).

FPGA. Kasap and Benkrid [50, 51] present an FPGA implementation that is based on a systolic array architecture for the acceleration of the PPF using Sankoff's algorithm [75]. The authors applied fine- and coarse-grained parallelism on multiple FPGAs, with each device hosting a systolic array with 20 processing elements. The accelerated system supported up to 12 taxa, which are transferred from a host processor to the FPGA through PCI-X connectivity while the ancestral sequences reside in on-chip memory. To evaluate performance, the authors used the Maxwell supercomputer [14], which contains Intel Xeon processors running at 2.8 GHz and 64× Virtex-4 FPGAs. The FPGA design was clocked at 70 MHz. Speedups between 5× and 21,606× were reported based on comparisons with PAUP [87], a software tool for phylogenetics analyses that was executed on an Intel Centrino Duo CPU running at 2.2 GHz. The speedups reported are only relative speedups with respect to the parsimony implementation in PAUP and not with respect to the fastest known (at the time of publication) implementation of parsimony in the TNT software [37]. Furthermore, the authors implemented an exhaustive search that yields high arithmetic intensity and large speedups but does not scale to many taxa or different tree search strategies, thereby limiting the applicability of the proposed solution in real-world studies.

Alachiotis and Stamatakis [7] accelerate the PPF using a vector-like pipelined architecture on an FPGA. The size of the vector of processing elements can be adjusted based on the available device resources. Input data (tips) are transferred to the FPGA using a custom Gigabit Ethernet communication framework [1, 2] and are stored in on-chip memory (block RAM) along with ancestral states. Unlike the approach of Kasap and Benkrid, this accelerator computes one PPF call per hardware invocation, i.e., it computes the ancestral state vector of the common ancestor of a pair of organisms (tree nodes/tips). As previously mentioned, the authors employ an optimized software implementation as a reference for assessing performance. Profiling reveals that ancestral-vector computations and final-score computations at the virtual root account for 99% of the total execution time, which are subsequently accelerated. In comparison with the AVX-based implementation, the reported speedups range between 5.6× and 9.6× using a Xilinx Virtex-6 FPGA operating at 188 MHz.

Block and Maruyama [16] also employ FPGAs for the PPF, accelerating the complete tree-search algorithm using a versatile approach that is not limited by the number of taxa. As with the SW solution previously mentioned, the accelerated solution uses a local search algorithm that relies on the PN [33] search method. Every algorithmic step is translated into a dedicated hardware unit, thereby allowing them to operate in parallel in a pipelined architecture. The solution is implemented on a Xilinx Kintex-7 FPGA operating at 157 MHz. Tip and ancestral vectors are stored in on-chip memory (block RAM) while host-FPGA communication is not discussed. The performance comparison, already presented in Section 3.1, shows that the FPGA accelerator achieves speedups in the order of thousands over the respective unoptimized software solution, but only matches the execution times of TNT, even though TNT constructs and evaluates 5× more trees than the FPGA implementation.

The more recent works of Block and Maruyama [17, 18] extend the accelerator design to also implement the Indirect Calculation of Tree Lengths [35] search method. When using the Indirect Calculation of Tree Lengths, the time required to visit all the internal nodes of a tree is fixed at $1/T$, where T is the number of taxa. The benefit of this search method is that the time required for this search does not increase with T [35]. The same Kintex-7 FPGA is used running at 156.25 MHz. When compared to the previous work, the approach achieves speedups between 34× and 45× per tree, and between 2× and 6× for the whole local search. When compared with TNT (version 1.1), the speedups per tree range from 2× to 4×, and from 18× to 112× for the whole local search.

GPU. Santander et al. [76] employ a GPU to accelerate phylogenetic tree inference based on the PPF. The authors use OpenCL [86], a framework for parallel programming of heterogeneous systems, and assess performance on various GPUs. In a subsequent study, Santander et al. [77] present a comparative assessment of various parallel-programming frameworks (OpenCL [86], CUDA [62] and OpenACC [101]) on different GPU architectures. The PPF is computed by organizing the entire computational task into independent sub-tasks that can be processed in parallel; each sub-task corresponds to a partial calculation of the PPF. A tree topology is stored on the GPU's constant memory, since this data structure has to be read only, while the sequences are stored in global memory due to their size. A row-major layout is adopted to ensure coalesced memory accesses. The performance results of the GPU-accelerated solution are compared with a sequential software implementation. Of the three GPUs used for evaluation, the Nvidia GeForce GTX TITAN X achieves the highest performance with speedups ranging from 8.8× to 324×. The lowest performance gain is measured on a dataset with short sequences (759 characters), where the memory transfer overhead is large. Santander et al. [77] shows that implementing the solution using the CUDA toolkit outperforms the OpenCL implementations, while both CUDA and OpenCL considerably outperform OpenACC, with OpenCL being between 2.3× to 3× faster than OpenAcc.

Table 1. Overview of High-performance Computational Solutions for the Phylogenetic Parsimony Function (PPF)

Work by	Year	System details	Achieved speedup/Reference
Satander et al. [78]	2020	CPU + GPU	22 × −299× / CPU (single-core)
Satander et al. [77]	2019	CPU + GPU	8.8 × −324× / CPU (single-core)
Satander et al. [77]	2019	CPU	8.7 × −77× / CPU (single-core)
Block and Maruyama [17, 18]	2017	CPU + FPGA	18 × −112× / CPU (multi-core)
Block and Maruyama [16]	2013	CPU	N/A
Block and Maruyama [16]	2013	CPU + FPGA	1 × / CPU (multi-core)
Alachiotis and Stamatakis [7]	2011	CPU	1.14 × −1.73× / CPU (SSE, multi-core)
Alachiotis and Stamatakis [7]	2011	CPU + FPGA	5.6 × −9.6× / CPU (AVX, multi-core)
Kasap and Benkrid [50, 51]	2010	CPU + FPGA	5 × −21,606× / CPU (single-core)

Santander et al. [78] present a new solution that solely focuses on processing protein sequences, which results in higher complexity due to the larger number of states (20 amino acids). This work also presents a comparative view on multiple GPU architectures and different implementation frameworks. Profiling the accelerated design revealed that nearly 40% of the execution time is spent on data transfers and pre-processing tasks. By overlapping data transfers with pre-processing tasks, this time can be reduced. Overall, the CUDA implementation on the Nvidia GeForce RTX 2080 Ti achieves the highest performance, with speedups ranging from 22× to 299× when compared to a sequential software implementation.

A summarized overview of the performance-driven solutions for the phylogenetic parsimony function that were reviewed in this section is provided in Table 1.

3.2 Phylogenetic Likelihood Function (PLF)

The PLF is used by Maximum Likelihood and Bayesian inference tools such as MrBayes [74] and RAxML [84] to evaluate phylogenetic trees by calculating the likelihood of the tree. The calculation of the PLF is both computationally and memory-intensive and takes approximately between 85% and 95% of the runtime [84]. This amounts to several CPU hours and is therefore of great importance to be accelerated. The PLF is recursively applied, starting from the tips and proceeding toward a virtual root, calculating likelihood vectors at the inner nodes of the tree topology.

The PLF for a phylogenetic tree is computed as follows: Every node x is represented by a probability vector $\overrightarrow{L}_x$ with m entries (m is the MSA sequence length) and each entry containing the probabilities L^A, L^C, L^G, and L^T of observing nucleotides A, C, G, and T, respectively. If x is a tip, then all probabilities are either 1.0 or 0.0, since the corresponding sequence is in the MSA. Given probability vectors $\overrightarrow{L}^A$ and $\overrightarrow{L}^B$, the probabilities of vector $\overrightarrow{L}^u_C(i)$, $i = 1..m$, $u \in N$, and $N = \{A, C, G, T\}$ at position i of the vector that describes the common ancestor C, are calculated as follows:

$$L^u_C(i) = \left(\sum_{s \in N} P_{u \rightarrow s}(t_l) * \overrightarrow{L}^s_A(i)\right) * \left(\sum_{s \in N} P_{u \rightarrow s}(t_r) * \overrightarrow{L}^s_B(i)\right), \tag{4}$$

with t_l and t_r being the lengths of the branches that connect node C with the child nodes A and B, respectively. $P_{u \rightarrow s}(t)$ is the probability for a nucleotide u to be substituted by a nucleotide s given a branch length t.

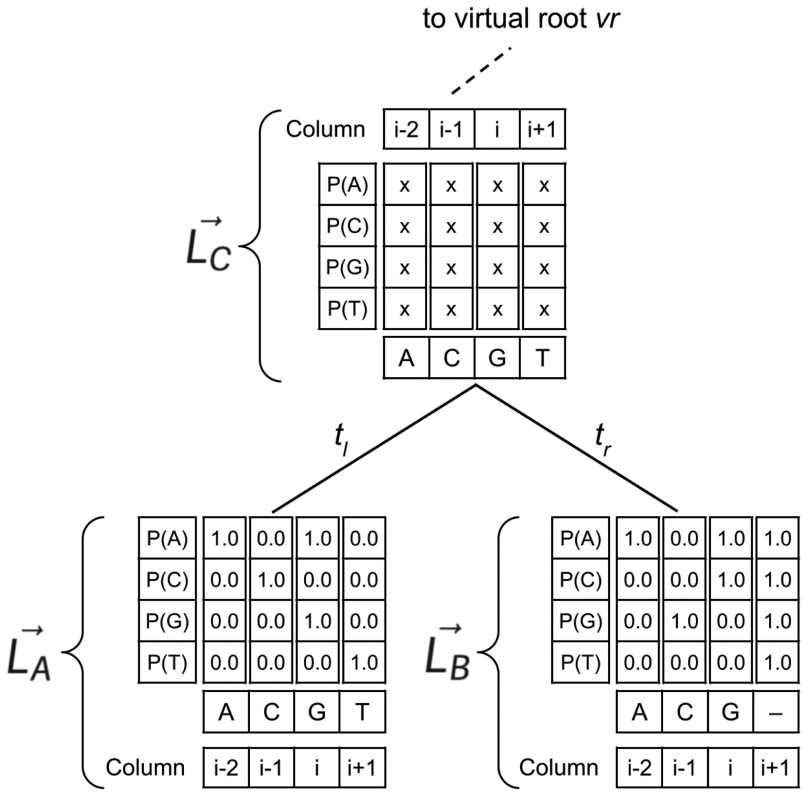

Fig. 5. Illustration of the computation of a likelihood vector $\vec{L}_C$ of a common ancestor C with two child nodes A and B with likelihood vectors $\vec{L}_A$ and $\vec{L}_B$, respectively.

Once the probability vector of the root node, $\vec{L}_{vr}$, is calculated, the final likelihood of the tree is computed by first calculating one likelihood score, $l(i)$, per position i, $i = 1..m$ using:

$$l(i) = \sum_{s \in N} \pi_s * \vec{L}^s_{vr}(i), \tag{5}$$

with π_s, $s \in N$ and $N = \{A, C, G, T\}$ being the prior probabilities of observing nucleotides A, C, G, and T at the root. Thereafter, the final likelihood score of the tree is computed as the sum of the logarithm of the per-site likelihood scores:

$$LH = \sum_{i=1}^{m} log(l(i)). \tag{6}$$

Figure 5 shows an illustration of the computation of the likelihood vectors of two child nodes and a common ancestor as a part of a complete tree. To calculate the likelihood score of a tree, Equation (4) is repeatedly performing matrix-matrix multiplications to compute ancestral state vectors at inner nodes, while Equations (5) and (6) are only applied at the virtual root. The PLF is typically applied on DNA or protein data. When processing DNA, Equation (4) multiplies 4×4 transition probability matrices with $4 \times m$ probability vectors (m is the alignment length), while when protein data is processed, 20×20 transition probability matrices are multiplied with $20 \times m$ probability vectors. A detailed explanation of the PLF is provided by Malakonakis et al. [58].

CPU. Pratas et al. [68] present solutions improving performance of the PLF on a CPU, a Cell Broadband Engine, and a GPU. The authors developed an OpenMP implementation and parallelized the outermost loop of the PLF to minimize synchronization overheads. The solution is implemented on three different systems with multiple CPUs: 2× Intel Xeon Quad-core CPUs (System A), 4× AMD Opteron Quad-core CPUs (System B), and 8× AMD Opteron Dual-core CPUs (System C), and performance comparisons are performed with sequential execution on an Intel Core2 Duo E8400 CPU. Overall, System B achieves the highest speedups, ranging from 4× to 11×, depending on the dataset. System C achieves the same maximum speedups but lower on-average performance. System A delivered lower performance than the other two systems under test (speedups between 6× and 7×), but performance was more consistent over varying dataset sizes.

Flouri et al. [32] present the **Phylogenetic Likelihood Library (PLL)**, a highly optimized application programming interface for developing likelihood-based phylogenetic inference and post-analysis software. Similarly, Ayres et al. [12, 13] present BEAGLE, an optimized software library that implements both likelihood-based and Bayesian-based development. BEAGLE implements solely the likelihood calculation, while the PLL also implements the tree data structure. The PLL implementation employs Intel 128-bit SSE and 256-bit AVX intrinsic instructions, whereas BEAGLE only employs 128-bit SSE instructions. Both libraries support parallel processing, with the PLL relying on Posix threads, while BEAGLE uses OpenMP. In the latest work of Ayres et al. [12], a performance comparison is presented between the AVX PLL (Version 2) and SSE BEAGLE (Version 3.1.2) implementations. Both solutions are executed on a single thread of an Intel Core i7-2600 CPU, where the PLL achieves speedups of up to 3.1× over BEAGLE.

FPGA. Alachiotis et al. [6] implement a subset of the PLF functions required to conduct a full real-world tree search on an FPGA, limiting functionality to fixed tree topologies. The solution is evaluated using trees with between 4 and 512 taxa and compared with an efficient parallel C code for multicore CPUs (a stripped-down version of RAxML). A Xilinx Virtex-5 SX240T was used as the target FPGA, which achieved speedups between 3× and 13.5× faster than a single CPU core. When compared with 8 CPU cores, the FPGA implementation led to a slowdown of 0.96× for the 16-taxon tree and speedups up to 5.08× for the rest of the tree sizes.

Zierke and Bakos [108] present an FPGA-accelerated solution based on MrBayes [74] using the Bayesian Metropolis-Coupled **Markov Chain Monte Carlo (MC^3)** method. In addition to the PLF, the normalisation and log-likelihood steps of MrBayes are also accelerated. For likelihood calculations, the internal nodes of the tree are processed via a post-order traversal with minimal intervention from the host to reduce CPU-FPGA communication overheads. The solution utilizes on-board memory to cache the output vectors of the computations to minimize host-FPGA communication. A deep pipeline architecture is devised, and the solution is implemented on a Xilinx Virtex-6 SX475 FPGA running at 310 MHz, and a Xeon 5500-series CPU is used as the host processor. For performance evaluation, the solution is compared with the software implementation of MrBayes executed on the same Xeon CPU, resulting in speedups between 4.7× and 8.7×.

In a recent study, Malakonakis et al. [58] implement the complete RAxML algorithm on a hybrid system. The calculation of the PLF is done on an FPGA while the rest of the algorithm runs on the host CPU. Two different target systems are expored: a Xilinx ZCU102 development board, which consists of a system-on-chip that combines reconfigurable logic with a quad-core ARM A53 general purpose processor, and a cloud-based Amazon AWS EC2 F1 instance that hosts multiple FPGAs connected to Intel Xeon E5-v4 CPUs through PCIe. The first system deploys at most two PLF accelerators at a frequency of 250 MHz, due to memory bandwidth limitations. This implementation is 7.7× faster than a pure software implementation run on an AWS EC2 F1 instance. The AWS-based accelerated system is about 5.2× faster than the software

implementation. In comparison with previous work by Alachiotis et al. [6], the implementation on the Xilinx development board is about 2.35× faster.

Alachiotis et al. [3] propose a systematic way to exploit the way tree-search algorithms work in practice to optimize data movement for PCI-attached accelerators for the PLF. The authors implemented a software cache controller that can exploit the data dependence between subsequent PLF calls in-between topology rearrangements to cache probability vectors that will be reused shortly on the accelerator card. In combination with double buffering, the software cache approach increased the perceived performance of an FPGA accelerator by nearly 4×. Executing the complete RAxML algorithm on an AWS EC2 F1 instance, the authors observed up to 9.2× faster analyses of protein data than a Xeon processor running at 2.7 GHz in the same cloud environment.

GPU. Pratas et al. [68] accelerate the calculation of the PLF within MrBayes on a GPU. A fine-grained architecture is adopted where parallelization is done at the likelihood vector entry level. Calculation of each vector entry is assigned to one independent thread to minimize thread synchronization. To improve performance, data partitioning is done on three levels: a) global partitions are created when the data is larger than the GPU's global memory, (b) block partitioning is used to distribute the likelihood array elements among processing engines that are processed independently, and (c) thread partitioning for a set of computations. For every call to the PLF, the input data is transferred to the GPU's global memory, and the results are returned when the computation is finished. The percentage of time spent on calculating the PLF is reduced from more than 90% to 5%–10%. The CPU-GPU communication, however, severely limits overall performance. The resulting speedup, when executed on an Nvidia GeForce GTX 285, is approximately 1.5× over the respective single-thread CPU implementation executed on an Intel Core2 Duo E8400. The presented speedup is scaled with respect to the clock frequencies of the GPU and CPU, which are 1.48 GHz and 3.0 GHz, respectively.

Zhou et al. [106] propose an improvement over the work by Pratas et al [68]. While the work by Pratas et al. mainly focuses on the GPU-side computations, this work adopts a more hybrid approach where the CPU performs computations in parallel with the GPU. Besides that, the authors employ pipelining to reduce the idle time of both platforms and overlap CPU-GPU communication with computations. To further improve performance, shared memory is exploited and thread idle operations are reduced to the minimum. When compared with the fastest—at the time of publication—CPU multi-core implementation, the solution achieves speedups between 0.9× to 5.4×. In comparison with the work by Pratas et al., the performance gain is between 7.5× and 12.6×.

Ayres et al. [12, 13] also include GPU acceleration in BEAGLE and present various optimizations in version 3.1.2 [12] to further improve GPU acceleration. Both CUDA and OpenCL are used in different implementations to target a wide range of GPUs, as well as solutions implementing single- and double-precision arithmetic. Fine-grained parallelization of the likelihood calculation is applied. BEAGLE version 3.1.2 optimized thread utilization by identifying additional opportunities for parallelization to prevent thread idle operations. Data partitioning is improved to prevent sequential execution of BEAGLE instances on the GPU. Overall, the optimizations in version 3.1.2 focus on higher utilization of the massively parallel architecture of the GPU. When executed on an Nvidia Tesla P100 GPU and two Intel Xeon E5-2690v4 CPUs as host, the solution achieves a 32× speedup over the single-thread PLL [32] software implementation on the same CPU.

A summarized overview of the performance-driven solutions for the phylogenetic likelihood function that were reviewed in this section is provided in Table 2.

4 POPULATION GENETICS

Population genetics study the genetic variation within one and among different populations. This includes the detection and understanding of footprints caused by evolutionary phenomena such

Table 2. Overview of High-performance Computational Solutions for the Phylogenetic Likelihood Function (PLF)

Work by	Year	System details	Achieved speedup/Reference
Alachiotis et al. [3]	2021	CPU + FPGA	9.2× / CPU (single-core)
Malakonakis et al. [58]	2020	CPU + FPGA	7.7× / CPU (multi-core)
Ayres et al. [12, 13]	2019	CPU + GPU	32× / CPU (AVX, single-core)
Flouri et al. [32]	2015	CPU	3.1× / CPU (SSE, single-core)
Zhou et al. [106]	2011	CPU + GPU	0.9 × −5.4× / CPU (multi-core)
Zierke and Bakos [108]	2010	CPU + FPGA	4.7 × −8.7× / CPU (multi-core)
Pratas et al. [68]	2009	CPU	4 × −11× / CPU (single-core)
Alachiotis et al. [6]	2009	CPU + FPGA	3 × −13.5× / CPU (single-core)
Pratas et al. [68]	2009	CPU + GPU	1.5× / CPU (single-core)

as positive selection, epistasis, recombination, linkage disequilibrium, and genetic drift, among others, which can explain changes in the frequencies of genes over space and time. The current section focuses on computational solutions for linkage disequilibrium and pairwise epistasis.

4.1 Linkage Disequilibrium

Linkage Disequlibrium (LD) is the non-random association between alleles at different loci in a population. The levels and patterns of LD in a population are affected by the action of positive selection, as predicted by the genetic hitchhiking model [82]. As the number of sequenced genomes increases and more genetic variation is discovered, the calculation of LD becomes increasingly compute- and memory-intensive. Computational and memory requirements increase quadratically with the number of **Single-Nucleotide Polymorphisms (SNPs)**, while computational requirements also increase linearly with the number of genomes (sample size).

The most commonly used measure of LD is r^2, which is equivalent to Pearson's correlation coefficient applied on binary data [69]. The calculation of LD between SNPs A and B is provided in Equation (7):

$$r^2_{AB} = \frac{D^2_{AB}}{p_A(1-p_A)p_B(1-p_B)}, \tag{7}$$

where p_A and p_B are the frequencies of the derived allele at SNPs A and B, respectively, while D_{AB} is the coefficient of linkage disequilibrium, defined as follows:

$$D_{AB} = p_{AB} - p_A p_B, \tag{8}$$

where p_{AB} is the frequency of occurrence of the derived allele at both SNPs A and B. Under the widely adopted Infinite Sites Model [55], which assumes that there is an infinite number of sites and consequently each new mutation appears on a site where previously no mutation has occurred, SNPs can be represented by binary vectors; an unset bit ("0") represents the initial—before mutation—state (the ancestral state) and a set bit ("1") indicates a new—after mutation—state (the derived state). Thus, allele frequencies are computed by dividing the number of set bits (population count operation) in SNPs and SNP pairs by the sample size.

The significance of LD, as can been observed from titles of population genetics papers declaring "Population genomics: Linkage disequilibrium holds the key" [34], has motivated various studies for high-performance LD computation.

CPU. Chang et al. [22] present an optimized version of PLINK [71], which is a widely used tool for whole-genome association studies and population genetics. The optimized version (PLINK1.9)

calculates both Pearson's correlation coefficient and D'[1] as measures of LD. PLINK1.9 implements various improvements, such as bit-level parallelism, vector instructions, and higher memory efficiency than its predecessor. For pairwise LD computations, PLINK1.9 is between 754× and 8,450× faster than PLINK1.07 (initial release) as a result of exploiting bit-level parallelism in combination of several algorithmic improvements, as can be inferred from the reported execution times by the authors (Chang et al. [22], Table 5).

Tang et al. [88] present LDkit, a parallel computing toolkit for linkage disequilibrium analysis. The tool implements both Pearson's correlation coefficient and D'. Using task-level parallelism to deploy multiple threads/cores, the authors report speedups of up to 12.8× (over single-thread execution) with 32 threads. Performance comparisons with other tools reveal than LDkit does not outperform PLINK1.9 [22], which is between 1.3× and 25× faster. LDkit, however, has a user-friendly graphical user interface.

Zhang et al. [105] present PopLDdecay, a C++ tool for LD decay analysis that can be used to study the rate of recombination in a population. Similarly to the previous tools, PopLDdecay implements both Pearson's correlation coefficient and D'. PopLDdecay does not outperform the second-generation of PLINK, which is approximately 2.7× faster, but it achieves higher memory efficiency, utilizing up to 12× less memory than PLINK, on average.

Alachiotis and Pavlidis [4] present a series of parallelization strategies to overcome the problem of load imbalance when computing LD on multi-core processors. A fine-grained parallelization approach is suitable for large sample sizes, achieving up to 11.1× speedup with 16 threads/cores, whereas a coarse-grained approach is proposed for better parallel performance on long genomes. Because the coarse-grained approach is particularly sensitive to load imbalance (varying SNP density along the genome), a generic algorithm is proposed that achieves up to 2.5× faster processing than the coarse-grained approach on 16 threads/cores. All parallelization alternatives are implemented in the open-source software OmegaPlus [8].

Alachiotis et al. [5] demonstrate that the calculation of LD can be cast in terms of **Dense Linear Algebra (DLA)** operations. The authors describe the caclulation of LD as a series of **Basic Linear Algebra Subprograms (BLAS)** [28, 43, 56] operations and show that the GotoBLAS [41] approach (now maintained as OpenBLAS [63]) can be used to compute LD as a high-performance **General Matrix Multiplication (GEMM)** operation . The proposed approach is implemented based on the **BLAS-Like Instantiation Software (BLIS)** [92]), i.e., a high-performance framework for rapidly implementing DLA operations using the GotoBLAS approach, and is up to 17× and 6.7× faster than PLINK1.9 [22] and OmegaPlus [8], respectively.

FPGA. Alachiotis et al. [9] present an FPGA accelerator for computing Pearson's correlation coefficient as a measure of LD, using the **Infinite Sites Model (ISM)** [55]. The hardware architecture is automatically generated based on a number of parameters that were used to explore the accelerator design space. The study reports that throughput improves when a moderate amount of wide, pipelined population count[2] operators are used instead of a larger number of narrow operators. 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 on a Xilinx Virtex-7 VX980T-2 FPGA with a clock frequency of 137 MHz. When compared with PLINK1.9 [22] running on a workstation with an Intel Xeon E5-2630 hexa-core 2.6 GHz CPU, the FPGA achieves 50× faster processing than 12 CPU threads and 200× faster processing than 1 CPU thread.

[1] D' is the normalized D (Equation (8)) by the maximum value it can obtain.
[2] Population counting describes the operation of counting the number of set bits ("1") in a computer register.

Bozikas et al. [19] also implement the Pearson's correlation coefficient as a measure of LD, with the architecture supporting the more compute-intensive **Finite Sites Model (FSM)**.[3] 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 four 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 4.7× faster than 12 CPU threads, whereas processing becomes up to 12.7× faster than 12 CPU threads when all four FPGAs are used. Despite using FPGA technology as well, the speedups by Bozikas et al. [19] are lower than the 50× speedup previously achieved by Alachiotis et al. [9] because of the additional support of the FSM model that requires more computations and hardware resources.

GPU. Xian et al. [102] present a GPU-accelerated solution for LD, computing Pearson's correlation coefficient under the ISM model. The authors employ the *__popc* instruction of the CUDA Toolkit API [62] for faster bit counting (population count operation). Furthermore, a data reorganization scheme and atomic instructions are used for reducing memory footprint and latency. The proposed solution is implemented on a cluster of Nvidia Tesla C2075 GPUs (two GPUs per node), achieving speedups between 906× and 1,589× in comparison with a sequential software implementation on an Intel Xeon E5410 quad-core CPU running at 2.33 GHz. The overall processing capacity of the cluster (number of nodes, CPU cores, and GPUs) is not specified.

Theodoris et al. [89] present quickLD, an optimized software for computing LD statistics using either a CPU or a GPU. The GPU implementation is based on the OpenCL framework. The authors focus on handling large-scale datasets by introducing a two-step process that separates parsing from processing. This allows for more flexibility in scheduling computation between distant SNPs without increasing memory requirements. For performance evaluation, quickLD is compared with PLINK1.9 on two different computing systems: (a) a personal computer with an Intel Core i5-8300H 2.3 GHz CPU and an 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 CPUs and an 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 29× faster processing than PLINK1.9 (20 threads).

Binder et al. [15] present a portable framework for performing CPU-based SNP comparison algorithms on a GPU. Comparing SNPs is the core of LD calculations. For portability, the implementation of LD is based on OpenCL [86] and maps the BLIS [92] framework onto the GPU. The SNP-comparison framework is evaluated on an Nvidia TITAN V GPU, a GeForce GTX 980 GPU, and an AMD Radeon Vega GPU, and performance comparisons with a BLIS-based CPU implementation [5] are performed. The authors report that the GPU implementation is up to 7.8× faster than the CPU implementation on an Intel Xeon E5-2620v2 6-core CPU running at 2.10 GHz.

A summarized overview of the performance-driven solutions for computing linkage disequilibrium statistic that were reviewed in this section is provided in Table 3.

4.2 Pairwise Epistasis

Epistasis is the phenomenon where interaction between different genes is antagonistic in such a way that one gene overrules or interferes with the expression of another gene. This section focuses on pairwise epistasis (direct gene-gene interaction). An example of pairwise epistasis is the interaction between genes that control hair color and genes responsible for total baldness. The

[3]Under FSM, several mutations can appear at the same genomic location; thus, alleles are represented by 2 to 5 bits, depending on the data type (DNA, protein) requiring more arithmetic operations than ISM due to the longer alphabet length.

Table 3. Overview of High-performance Computational Solutions for Computing Linkage Disequilibrium (LD) Statistics

Work by	Year	System details	Achieved speedup / Reference
Tang et al. [88]	2020	CPU	12.8× / CPU (single-core)
Theodoris et al. [89]	2020	CPU + GPU	20× / CPU (SSE, multi-core)
Zhang et al. [105]	2019	CPU	N/A / N/A
Binder et al. [15]	2019	CPU + GPU	7.8× / CPU (multi-core)
Bozikas et al. [19]	2017	CPU + FPGA	4.7 × −12.7× / CPU (SSE, multi-core)
Alachiotis et al. [4]	2016	CPU	2.5 × −11× / CPU (multi-core)
Alachiotis et al. [5]	2016	CPU	17× / CPU (SSE, multi-core)
Alachiotis et al. [9]	2016	CPU + FPGA	50× / CPU (SSE, multi-core)
Chang et al. [22]	2015	CPU	754 × −8,450× / CPU (multi-core)
Xian et al. [102]	2013	CPU + GPU	906 × −1,589× / CPU (single-core)

gene that is responsible for total baldness is epistatic to the gene that controls hair color because the gene for total baldness supersedes the effect of the gene that controls hair color. The gene that controls hair color is called hypostatic to the gene for total baldness.

Detecting pairwise epistasis consists of two stages: (a) creation of contingency tables that contain the (multivariate) frequency distribution of the variables, and (b) statistical testing of each created table. A contingency table is created for every SNP pair, which leads to excessive compute and memory requirements. Because of this, approximate statistical tests [91, 96] have been proposed to shorten analysis times when conducting **Genome-Wide Association Studies (GWAS)**. Commonly used tools for epistasis detection are BOOST [94], MB-MDR [21], and iLOCi [65]. The most widely used method for pairwise epistasis relies on the maths behind BOOST, which is described in detail by Wan et al. [94]. **BOOST (BOolean Operation-based Screening and Testing)** consists of two stages, a screening stage and a testing stage. In the screening stage, the Kirkwood superposition approximation is used to evaluate all pairwise SNP interactions based on the contingency table collected by using Boolean operations, and a user-defined threshold is applied to determine whether the SNP pair will be used in the testing stage. In the testing stage, a likelihood ratio is computed for each pair to assess the interaction effect, and a statistical test is applied to determine whether the interaction is significant. Cordell [24] provides a detailed explanation of epistasis and related statistical methods.

CPU. Wienbrandt et al. [100] present an optimized implementation of the BOOST [94] algorithm, which performs an exhaustive pairwise analysis using statistical regression and is used by PLINK [71]. To improve performance, sample covariance is not supported, and a logistic regression test based on contingency tables is used. This optimization reduces the computational complexity from $O(NT)$ to $O(N + T)$, where N is the number of samples and T is the number of iterations required for a single test. When executed on a system with two Octa-core Intel Xeon E5-2667v4 3.2 GHz CPUs, the optimized version is between 10× and 15× faster than the original PLINK BOOST implementation.

González-Domínguez et al. [38] also optimize the PLINK BOOST algorithm using logistic regression, targeting the Intel Xeon Phi 5110P co-processor with between 57 and 61 simplified Intel CPU cores running at 1.0–1.2 GHz. Optimizations are mainly focused on exploiting the available 512-bit-wide vector instructions, including the *popcount* instruction. In addition, an embarrassingly parallel workflow is adopted to employ the underlying many-core architecture. Moreover, the authors present a heterogeneous CPU/GPU implementation that additionally deploys an Nvidia Tesla K20m GPU. This heterogeneous implementation is between 8× and 33× faster than

the Phi-only software implementation, as can be inferred from the reported execution times by the authors (González-Domínguez et al. [38], Table 3).

FPGA. Wienbrandt et al. [99] present an FPGA-accelerated GWAS epistasis detection tool. The solution combines fine- with coarse-grained parallelism through systolic arrays on multiple FPGAs, resulting in a large number of **Processing Engines (PEs)** operating in parallel. The systolic array architecture is used for both the creation of large contingency tables and the application of a statistical test that is adopted from iLOCi [65]. The authors introduce a nearly redundant-free SNP pairing scheme while maintaining load balance among a large number of FPGAs. The proposed solution is implemented on the RIVYERA [64] system that features 128 Xilinx Spartan 6-LX150 FPGAs and two Intel Xeon E5-2620 CPUs as host processor. All FPGAs run at a clock frequency of 150 MHz, and each one of them implements 100 PEs. The accelerated implementation achieves up to 285× faster processing than the iLOCi software executed on two Intel Xeon quad-core 2.4 GHz CPUs.

González-Domínguez et al. [40] also target the RIVYERA [64] system for implementation of the commonly used BOOST [94] algorithm, including its statistical tests. A similar systolic architecture as in the work by Wienbrandt et al. [99] is used for large-scale parallel pairwise contingency table creation and preparation, followed by the statistical tests. A total of 128 FPGAs running at 133 MHz are deployed, with each device hosting 56 processing engines. For performance evaluation purposes, the authors created an optimized parallel software implementation using PThreads. The FPGA solution achieves a speedup of 190× when the software implementation is run on an Intel Core i7-3930K using 12 threads.

GPU. Hemani et al. [45] propose a GPU-accelerated pairwise epistasis analysis tool called epiGPU, which uses the OpenCL framework [86]. The solution performs an exhaustive pairwise analysis where each SNP is statistically tested against all other SNPs, resulting in a two-dimensional search grid with calculations distributed over the massive parallel architecture of the GPU. To increase performance, the program does not consider the effect of covariates in the analysis. Also, the authors observe that the slow access to global memory limits performance considerably, and they introduce optimizations to the regression algorithm to achieve higher utilization of the faster shared memory on the GPU. Moreover, the CPU-GPU communication overhead is minimized by using bit-packed compression, while the memory access time is minimized by using coalesced memory accesses. The optimizations result in a 15× speedup over the unoptimized implementation. For performance evaluation, the Nvidia GeForce GTX 580 was used with an Intel Core i7-970 CPU as host processor. In comparison with the respective parallel software implementation running on the host CPU (six CPU cores), the accelerated epiGPU solution is 15.7× faster.

Yung et al. [104] present GBOOST, a GPU-accelerated implementation of BOOST [94] using the CUDA toolkit. The creation and preparation of the contingency tables as well as the statistical test are performed on the GPU. The log-linear filter, however, is executed on the CPU to avoid thread divergence. The performance of the statistical test calculation is improved through coalesced memory accesses to the global memory. The solution omits the effects of covariates. For performance comparison GBOOST is tested on an Nvidia GeForce GTX 285 and compared to BOOST, which is executed on an unknown CPU running at 3 GHz. GBOOST achieves a 40× speedup compared with BOOST.

Wang et al. [95] present an optimized version of GBOOST [104] called GBOOST 2.0. The solution achieves higher true positive rates than GBOOST through the implementation and consideration of covariates. Performance comparison is performed using the Nvidia GeForce GTX 285, but no host CPU is reported. The authors report a speedup of 1.5× speedup of GBOOST 2.0 over GBOOST.

González-Domínguez et al. [40], in addition to the FPGA implementation previously discussed, also present a hybrid CPU-GPU implementation of the BOOST algorithm. The same contingency

Table 4. Overview of High-performance Computational Solutions for Calculating Pairwise Epistasis

Work by	Year	System details	Achieved speedup / Reference
Wienbrandt et al. [100]	2019	CPU	10 × −15× / CPU (multi-core)
Wang et al. [95]	2016	CPU + GPU	1.5× / GPU
González-Domínguez et al. [38]	2015	CPU	N/A
González-Domínguez et al. [40]	2015	CPU + FPGA	190× / CPU (multi-core)
González-Domínguez et al. [40]	2015	CPU + GPU	31× / CPU (multi-core)
Wienbrandt et al. [99]	2014	CPU + FPGA	285× / CPU (multi-core)
Hemani et al. [45]	2011	CPU + GPU	15.7× / CPU (multi-core)
Yung et al. [104]	2011	CPU + GPU	40× / CPU (single-core)

table creation and statistical test calculation steps are performed as with the FPGA implementation. The GPU solution, however, uses a single kernel to perform the whole analysis on a batch of SNP-pairs. This improves performance, since large tables do not need to be stored in global memory. The resulting SNP information is stored in binary form in global memory before being transferred back to the host. As with the FPGA implementation, the GPU implementation is also compared with the same optimized software solution that employs PThreads, executed on an Intel Core i7-3930K using 12 threads. The GPU solution is also compared with BOOST [94] executed on the same CPU but using a single thread. Using an Nvidia GeForce GTX TITAN GPU, speedups of 269× and 31× are achieved over BOOST and the PThreads implementation, respectively.

A summarized overview of the performance-driven solutions for epistasis evaluation that were reviewed in this section is provided in Table 4.

4.3 Third-order Epistasis

Third-order epistasis is the epistatic interaction between three genes. It is used to explain complex traits such as Alzheimer's disease [109] or breast cancer [73]. Three-way gene interaction increases the number of required tests from $n(n-1)/2$ to $n(n-1)(n-2)/6$, where n is the number of SNPs in the dataset, thus massively increasing the computation and memory complexity of the problem.

CPU. Ponte-Fernández et al. [66] present MPI3SNP, a tool that implements an exhaustive third-order epistasis examination on CPU cluster systems. Unlike other works, this solution does not make any compromises to keep execution times feasible, e.g., limiting the number of SNPs. The solution is a hybrid implementation that combines **Message Passing Interface (MPI)** [42] processes and threads to utilize the available cores on every node of a cluster. A workload distribution scheme is implemented to prevent high synchronization and communication times.

A more recent study by Ponte-Fernández et al. [67] extends the MPI3SNP solution with different functions implemented according to the SIMD execution model using 512-bit AVX vector intrinsics. The vector implementation of the algorithm is between 5.0× and 15.0× faster than the initial solution when both algorithms are executed on an Intel Xeon Gold 6240 using 18 cores.

FPGA. Kässens et al. [52] present an FPGA-accelerated third-order epistasis analysis tool for exhaustive GWAS. The solution extends an earlier work by Wienbrandt et al. [99] and applies fine-grained parallelism using a systolic array architecture. Due to the increased memory complexity of third-order epistasis, the solution implements multiple systolic arrays that perform parts of the whole computation to achieve high performance and to prevent data transfer stalls. To further increase performance, resource usage is minimized to allow to increase parallelism of the systolic array architecture. A total of 102 PEs, divided into 6 arrays, are implemented on a Xilinx Kintex

Table 5. Overview of High-performance Computational Solutions for Calculating Third-order Epistasis

Work by	Year	System details	Achieved speedup/Reference
Ponte-Fernández et al. [67]	2022	CPU	5 × -15× / CPU (multi-core)
Ribeiro et al. [72]	2021	CPU + FPGA	6.0× / FPGA (Kässens et al. [52])
Ponte-Fernández et al. [66]	2020	CPU + GPU	126.3× / CPU (single-core)
Nobre et al. [61]	2020	CPU + GPU	3.3 × -4.4× / GPU
Kässens et al. [52]	2015	CPU + FPGA	182× / CPU (multi-core)

7-K325T FPGA running at a frequency of 250 MHz. The FPGA solution achieves a speedup of 182× when compared to an equivalent, optimized parallel, software implementation using six cores on an Intel Core i7 Sandy Bridge 3.20 GHz CPU.

Ribeiro et al. [72] present an FPGA-accelerated high-order exhaustive epistasis analysis tool. A novel parameterizable architecture is proposed that can perform any order of epistasis interaction with any dataset size. The developed architecture is parameterizeable to generate specialized accelerators for any epistatic interaction order. Resource optimizations are applied to a systolic array architecture, e.g., algorithmic optimizations for binary encoded input data are applied as well as SNP processing decoupling among several specialized specific function units for different stages of the epistasis detection. To maximize memory bandwidth and minimize data traffic, the data layout is reorganized and the data transfer size is tuned to the number of PEs. The solution is implemented on a Xilinx Virtex 7-690T running at a frequency of 250 MHz with 204 PEs. Using a simulated dataset with the same number of SNPs and samples as the evaluation setup of Kässens et al. [52], this solution outperforms the solution of Kässens by 6.0×.

GPU. The initial work by Ponte-Fernández et al. [66] also presents a GPU cluster system implementation of MPI3SNP to perform a third-order exhaustive search. A hybrid approach is implemented that utilizes both MPI [42] and CUDA [62]. MPI is used to transfer the workload to the GPUs and to call the correct kernel. The CUDA kernels are adopted from an earlier work by González-Domínguez and Schmidt [39]. The two steps of epistasis analysis are implemented using two kernels to minimize thread divergence. When using a single Nvidia Tesla K80 GPU, which is effectively the solution by González-Domínguez and Schmidt [39], a speedup of 126.3× is achieved over a sequential CPU implementation using an Intel Xeon E5-2680v3. Using eight Tesla K80 GPUs, a speedup of 947× is achieved over the sequential version.

Nobre et al. [61] propose a highly optimized hybrid CPU-GPU solution. The solution aims to fully exploit the computational capabilities of both the CPU and the GPU by scheduling and distributing parts of the algorithm suited for each of the architectures. A multi-core CPU generates the SNP vectors, as this task requires complex control and synchronization primitives. The analysis of the SNP vectors takes place on the GPU, since it is a highly data-parallel task: It consists of the computation of contingency tables and statistical testing to determine the local optimal result. The process of generating and analyzing the SNP vectors is split up into multiple scheduling rounds until all SNP combinations are examined. When compared to the GPU implementation of MPI3SNP, speedups between 3.35× and 4.39× are achieved using an Intel Core i7-5960X and an Nvidia Titan X.

5 ENERGY CONSUMPTION

Energy efficiency is a major concern in high-performance computing This section summarizes energy-consumption evaluation results from the previously discussed works. Only a limited number of studies provide energy consumption measurements: two studies that implement the

PPF [77, 78], two studies that compute pairwise epistasis [40, 99], and three studies that compute third-order epistasis [61, 66, 72].

Phylogenetic Parsimony Function. Santander et al. [77] implement the PPF for phylogenetic tree inference using the GPU architecture. Various parallel programming frameworks and GPUs are used to collect performance values. The Nvidia GeForce GTX TITAN X achieved the highest performance (in terms of execution time) using 224.7 J to process a dataset that consists of 1,495 sequences and 8,610 sites. The Nvidia Tesla K40 and K20Xm use 256.9 J and 251.3 J to examine the same dataset, respectively. The energy consumption values of the parallel-programming frameworks (OpenCL [86], CUDA [62], and OpenACC [101]) are in line with the computational performance of each framework, as discussed in Section 3.1. This results in the CUDA implementation consuming the least amount of energy, closely followed by OpenCL, while both frameworks consume far less than OpenACC.

Santander et al. [78] also compute the PPF, solely focusing on processing protein sequences. This work presents energy consumption values using the Nvidia Tesla V100 and Nvidia GeForce RTX 2080 Ti utilizing the CUDA toolkit. To examine a dataset with 720 sequences and 5,873 sites, 51.4 J and 75.8 J are consumed using the V100 and RTX 2080 Ti, respectively. The RTX 2080 Ti achieves the best energy efficiency based on this dataset.

Pairwise epistasis. González-Domínguez et al. [40] accelerate the BOOST algorithm for pairwise epistasis with a hybrid CPU-GPU implementation utilizing the CUDA toolkit with the Nvidia Tesla K20 and GTX TITAN GPUs. To perform an analysis of a dataset with 5,009 sequences and 500,568 SNPs, the Tesla K20 and TITAN GPUs consume 648 kJ and 540 kJ, respectively. The authors also present an FPGA implementation of the BOOST algorithm using the RIVYERA [64] system. To analyze the same dataset with 5,009 sequences and 500,568 SNPs, the FPGAs running at a clock frequency of 133 MHz outperform the GPU implementation presented in the work and consume a total of 252 kJ.

Wienbrandt et al. [99] present an FPGA-accelerated GWAS epistasis detection tool using the RIVYERA [64] system that features 128 Xilinx Spartan 6-LX150 FPGAs. Multiple FPGAs are used in the fine- and coarse-grained parallel solution running at a clock frequency of 150 MHz. The RIVYERA system consumes 180 kJ to examine a dataset with 5,000 sequences and 500,000 SNPs.

Third-order epistasis. Ribeiro et al. [72] present an FPGA-accelerated high-order exhaustive epistasis analysis tool implemented on the Xilinx Virtex 7-690T FPGA and Zync-7000 and Zync-Ultrascale+ development boards. The reconfigurable logic on the Zync-7000 is clocked at a frequency of 250 MHz and consumes 1,156 kJ for performing a third-order analysis on a dataset consisting of 6,400 sequences and 40,000 SNPs. Note that the authors, in addition to energy consumption values of their own work, present energy consumption measurements for the works of Ponte-Fernández et al. [66] and Nobre et al. [61], which are presented in the following paragraphs.

Ponte-Fernández et al. [66] present a GPU-accelerated third-order exhaustive epistasis analysis for GPU clusters. The solution is tested on a heterogeneous system consisting of an Intel Core i9-7900K and an Nvidia TITAN V. The same dataset consisting of 6,400 sequences and 40,000 SNPs is analyzed, consuming 24,840 kJ. This measurement is reported by Ribeiro et al. [72].

Nobre et al. [61] also present a GPU-accelerated third-order exhaustive epistasis analysis tool but optimized for a single GPU. The same dataset and system are used as with the energy consumption evaluation of the work by Ponte-Fernández et al. [66]. The third-order analysis of the 6,400 sequences and 40,000 SNPs consumes 82,084 kJ. This measurement is reported by Ribeiro et al. [72].

Table 6. Overview of Energy Consumption of Discussed Works

Work by	Year	System details	Algorithm	Energy (J)	Dataset (seq./SNPs)	Power (W)
Ribeiro et al. [72]	2021	CPU + FPGA	3Epi.	1,156 k	6,400/40,000	14.73
Ponte-Fernández et al. [66]	2020	CPU + GPU	3Epi.	24,840 k	6,400/40,000	250 (Max)
Nobre et al. [61]	2020	CPU + GPU	3Epi.	82,084 k	6,400/40,000	250 (Max)
Santander et al. [78]	2020	CPU + GPU	PPF	75.8	720/5,873	137.7
Santander et al. [77]	2019	CPU + GPU	PPF	224.7	1,495/8,610	123.9
González-Domínguez et al. [40]	2015	CPU + GPU	Epi.	540 k	5,009/500,568	–
González-Domínguez et al. [40]	2015	CPU + FPGA	Epi.	252 k	5,009/500,568	–
Wienbrandt et al. [99]	2014	CPU + FPGA	Epi.	180 k	5,000/500,000	780 (Max)

A summary of the energy-consumption results that were reviewed in this section is provided in Table 6.

6 UNCONVENTIONAL ARCHITECTURES

The processing platforms (CPU, GPU, FPGA) reviewed so far are load-store architectures: data need to be loaded from main memory to the processing units and then stored back to main memory when computation is over. This data movement restricts system performance, and this problem intensifies in Bioinformatics with the large volumes of data to be processed. To overcome this performance limitation, some more advanced, unconventional architectures have already been proposed, which we briefly review in this section.

Huanfu et al. [46] present RADAR, a 3D-ReRAM-based DNA alignment accelerator architecture. RADAR is a **Processing-In-Memory (PIM)** architecture that utilizes **Resistive Content Addressable Memory (ReCAM)** with the aid of **Application-Specific Integrated Circuits (ASICs)** to accelerate BLASTN, a widely used nucleotide sequence alignment algorithm. RADAR is between 53× and 189× faster than multi-core CPUs, FPGA-, and GPU-based accelerators.

A study by Joardar et al. [47] demonstrates the potential of using a Network-on-Chip enabled co-design framework for counting *k-mers* (substrings of fixed length k) in DNA and protein sequences. A PIM design is proposed, which offers in-memory computation options to the software. The system outperforms a state-of-the-art software implementation that utilizes **Hybrid Memory Cube (HMC)** memory by up to 7.14×.

Prousalis and Konofaos [70] improve performance of dot matrix sequence alignment through the use of quantum computing for pattern recognition. The proposed method is partially based on the quantum search algorithm presented by Zhou and Ding [107]. Prousalis and Konofaos [70] organize the dot matrix data in a way that allows implementation and execution of the adopted quantum search algorithm. The work does not present any speedup values but shows the effectiveness of the proposed method through an example application.

Ellinas and Jarvis [29] also explore quantum computing for Bioinformatics algorithms. More specifically, the study describes how quantum computing can be used to compute probability vectors of phylogenetic tree nodes. The authors simulate a variety of standard phylogenetic branching models applied on trees of various topologies using appropriate decoherent quantum circuits.

Kim et al. [54] propose a novel seed location filtering algorithm called GRIM-Filter. The GRIM-Filter is optimized to exploit 3D-stacked memory systems that perform processing-in-memory by integrating computation within a logic layer stacked under memory layers.

Kaplan et al. [49] present a novel **Processing-In-Storage (PIS)** architecture called PRinS. The architecture employs to accelerate the Smith-Waterman sequence alignment algorithm [83]. The architecture is capable of general-purpose associative processing and can also be applied to machine learning tasks. High performance is achieved by simultaneously using the proposed

architecture for data storage and processing, implementing a massively parallel SIMD accelerator. The system is at least 4.7× faster than implementations using a Xeon Phi [57], an FPGA [98], and a GPU cluster [27].

7 DISCUSSION AND CONCLUSION

Due to advances in DNA sequencing technologies in the past years, the domain of Bioinformatics has been gradually transformed into a computational discipline that requires scalable algorithms and high-performance processing systems. The use of hardware acceleration and high-performance computing is found to be a viable solution for this trend. In this study, we reviewed heterogeneous FPGA-/GPU-based systems and CPU-based algorithmic optimizations that boost performance of compute-intensive kernels in the fields of phylogenetics and population genetics, providing insights into the performance and energy-efficiency potential of these processing technologies.

CPU optimizations and hardware-accelerated solutions empowered by FPGAs and GPUs are capable of reducing analysis times by two to three orders of magnitude in comparison with unoptimized software implementations. Despite the well-demonstrated potential of such technologies, however, we observe a lack of widespread adoption of hardware accelerators by the computational biology and bioinformatics community. This can be explained, in part, by the high acquisition costs of more advanced, specialized hardware like FPGAs, which, unlike CPUs and GPUs, require low-level technical expertise to set up a working system. Interestingly, there are several on-demand cloud computing platforms available today that offer pay-per-use access to GPU and FPGA hardware-accelerated servers, thereby mitigating acquisition and setup costs.

While on-demand cloud computing represents a cost-effective option for end-users, one more general challenge facing hardware accelerators, which possibly hinders their widespread adoption further, is the limited I/O and external memory bandwidth; when I/O- or memory-bound tasks are offloaded to accelerator hardware, the effective accelerator performance perceived by the host application can be considerably lower than the accelerator's theoretical peak performance if the system cannot satisfy the accelerator's bandwidth requirements. Hardware accelerators for phylogenetics are generally more prone to performance degradation (due to system-level bandwidth capacity limitations) than accelerators for population genetics. Because phylogenetics investigate interspecies relationships, accelerators for tasks that usually dominate analysis times, such as the PLF and the PPF, operate on a species pair at a time to calculate a representation of the common ancestor (an inner-node vector). On the one hand, offloading pair-based computations to accelerators leads to hardware solutions that can work in tandem with any tree search strategy (as they are independent of the tree structure [3]) and avoids tree-size limitations that can make the solutions impractical for real-wold, many-species phylogenies [50]. On the other hand, the workload of a topology-unaware hardware accelerator becomes a function of the alignment length alone. This reduces arithmetic intensity, because the inner-node vector calculation, which is an embarrassingly parallel task with no cross-site dependencies, leaves no opportunities for data reuse. Therefore, the negative effect that insufficient system bandwidth has on the accelerator's perceived performance is exacerbated, suggesting that boosting performance of phylogenetic tree reconstruction using specialized hardware is first about optimizing the data movement to/from the accelerator.

Hardware accelerators for population genetics are less likely to be bound by the system bandwidth as they investigate interactions between genomic locations across all available samples of a species. Tasks that frequently dominate analysis times calculate pairwise statistics to quantify concepts such as LD and epistasis in an all-to-all fashion over all locations in one or more genomic regions of interest. This offers several opportunities for data reuse and increased arithmetic intensity that is now limited by the computational capacity of the processing/acceleration hardware [5].

As genomic datasets continue to grow in size, we expect to see more real-world bioinformatics analyses conducted on cloud computing infrastructures in the future. This will be enabled by a shift of engineering efforts toward devising programming models and frameworks that alleviate the data-movement problem, since hardware accelerators in cloud infrastructures require explicit data movement from a host processor to the accelerator card (GPU or FPGA), which can easily dominate execution times and dismiss compute-time improvements. Luckily, the concerns are common for FPGA and GPU acceleration, and we expect to witness a convergence of engineering efforts in addressing the common data-movement challenge, thereby opening up the path for existing research outcomes on optimized processing solutions to be even more frequently deployed by computational biologists and bioinformaticians.

REFERENCES

[1] Nikolaos Alachiotis, Simon A. Berger, and Alexandros Stamatakis. 2010. Efficient PC-FPGA communication over gigabit ethernet. In *10th IEEE International Conference on Computer and Information Technology*. IEEE, 1727–1734.

[2] Nikolaos Alachiotis, Simon A. Berger, and Alexandros Stamatakis. 2012. A versatile UDP/IP based PC FPGA communication platform. In *International Conference on Reconfigurable Computing and FPGAs*. IEEE, 1–6.

[3] Nikolaos Alachiotis, Andreas Brokalakis, Vasilis Amourgianos, Sotiris Ioannidis, Pavlos Malakonakis, and Tasos Bokalidis. 2021. Accelerating phylogenetics using FPGAs in the cloud. *IEEE Micro* 41, 4 (2021), 24–30.

[4] Nikolaos Alachiotis and Pavlos Pavlidis. 2016. Scalable linkage-disequilibrium-based selective sweep detection: A performance guide. *GigaScience* 5, 1 (2016), s13742–016.

[5] Nikolaos Alachiotis, Thom Popovici, and Tze Meng Low. 2016. Efficient computation of linkage disequilibria as dense linear algebra operations. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 418–427.

[6] Nikolaos Alachiotis, Euripides Sotiriades, Apostolos Dollas, and Alexandros Stamatakis. 2009. Exploring FPGAs for accelerating the phylogenetic likelihood function. In *IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 1–8.

[7] Nikolaos Alachiotis and Alexandros Stamatakis. 2011. FPGA acceleration of the phylogenetic parsimony kernel? In *21st International Conference on Field Programmable Logic and Applications*. IEEE, 417–422.

[8] Nikolaos Alachiotis, Alexandros Stamatakis, and Pavlos Pavlidis. 2012. OmegaPlus: A scalable tool for rapid detection of selective sweeps in whole-genome datasets. *Bioinformatics* 28, 17 (2012), 2274–2275.

[9] Nikolaos Alachiotis and Gabriel Weisz. 2016. High performance linkage disequilibrium: FPGAs hold the key. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 118–127.

[10] Srinivas Aluru and Nagakishore Jammula. 2013. A review of hardware acceleration for computational genomics. *IEEE Des. Test* 31, 1 (2013), 19–30.

[11] Ryan M. Ames, Daniel Money, Vikramsinh P. Ghatge, Simon Whelan, and Simon C. Lovell. 2012. Determining the evolutionary history of gene families. *Bioinformatics* 28, 1 (2012), 48–55.

[12] Daniel L. Ayres, Michael P. Cummings, Guy Baele, Aaron E. Darling, Paul O. Lewis, David L. Swofford, John P. Huelsenbeck, Philippe Lemey, Andrew Rambaut, and Marc A. Suchard. 2019. BEAGLE 3: Improved performance, scaling, and usability for a high-performance computing library for statistical phylogenetics. *System. Biol.* 68, 6 (2019), 1052–1061.

[13] Daniel L. Ayres, Aaron Darling, Derrick J. Zwickl, Peter Beerli, Mark T. Holder, Paul O. Lewis, John P. Huelsenbeck, Fredrik Ronquist, David L. Swofford, Michael P. Cummings, et al. 2012. BEAGLE: An application programming interface and high-performance computing library for statistical phylogenetics. *System. Biol.* 61, 1 (2012), 170–173.

[14] Rob Baxter, Stephen Booth, Mark Bull, Geoff Cawood, James Perry, Mark Parsons, Alan Simpson, Arthur Trew, Andrew McCormick, Graham Smart, et al. 2007. Maxwell—A 64 FPGA supercomputer. In *2nd NASA/ESA Conference on Adaptive Hardware and Systems (AHS'07)*. IEEE, 287–294.

[15] Elliott Binder, Tze Meng Low, and Doru Thom Popovici. 2019. A portable GPU framework for SNP comparisons. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 199–208.

[16] Henry Block and Tsutomu Maruyama. 2013. A hardware acceleration of a phylogenetic tree reconstruction with maximum parsimony algorithm using FPGA. In *International Conference on Field-Programmable Technology (FPT)*. IEEE, 318–321.

[17] Henry Block and Tsutomu Maruyama. 2014. An FPGA hardware acceleration of the indirect calculation of tree lengths method for phylogenetic tree reconstruction. In *24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–4.

[18] Henry Block and Tsutomu Maruyama. 2017. FPGA hardware acceleration of a phylogenetic tree reconstruction with maximum parsimony algorithm. *IEICE Trans. Inf. Syst.* 100, 2 (2017), 256–264.
[19] Dimitrios Bozikas, Nikolaos Alachiotis, Pavlos Pavlidis, Evripides Sotiriades, and Apostolos Dollas. 2017. Deploying FPGAs to future-proof genome-wide analyses based on linkage disequilibrium. In *27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–8.
[20] Katherine S. Button, John P. A. Ioannidis, Claire Mokrysz, Brian A. Nosek, Jonathan Flint, Emma S. J. Robinson, and Marcus R. Munafò. 2013. Power failure: Why small sample size undermines the reliability of neuroscience. *Nat. Rev. Neurosci.* 14, 5 (2013), 365–376.
[21] M. Luz Calle, Víctor Urrea Gales, Núria Malats i Riera, Kristel Van Steen, et al. 2008. MB-MDR: Model-based multifactor dimensionality reduction for detecting interactions in high-dimensional genomic data. (2008). http://hdl.handle.net/10854/408. Accessed May 4, 2022.
[22] Christopher C. Chang, Carson C. Chow, Laurent C. A. M. Tellier, Shashaank Vattikuti, Shaun M. Purcell, and James J. Lee. 2015. Second-generation PLINK: Rising to the challenge of larger and richer datasets. *Gigascience* 4, 1 (2015), s13742–015.
[23] Benny Chor and Tamir Tuller. 2005. Maximum likelihood of evolutionary trees: Hardness and approximation. *Bioinformatics* 21, suppl_1 (2005), i97–i106.
[24] Heather J. Cordell. 2002. Epistasis: What it means, what it doesn't mean, and statistical methods to detect it in humans. *Hum. Molec. Genet.* 11, 20 (2002), 2463–2468.
[25] Carrie A. Davis, Benjamin C. Hitz, Cricket A. Sloan, Esther T. Chan, Jean M. Davidson, Idan Gabdank, Jason A. Hilton, Kriti Jain, Ulugbek K. Baymuradov, Aditi K. Narayanan, et al. 2018. The Encyclopedia of DNA elements (ENCODE): Data portal update. *Nucleic Acids Res.* 46, D1 (2018), D794–D801.
[26] William H. E. Day, David S. Johnson, and David Sankoff. 1986. The computational complexity of inferring rooted phylogenies by parsimony. *Math. Biosci.* 81, 1 (1986), 33–42.
[27] Edans Flavius de Oliveira Sandes, Guillermo Miranda, Xavier Martorell, Eduard Ayguade, George Teodoro, and Alba Cristina Magalhaes Melo. 2016. CUDAlign 4.0: Incremental speculative traceback for exact chromosome-wide alignment in GPU clusters. *IEEE Trans. Parallel Distrib. Syst.* 27, 10 (2016), 2838–2850.
[28] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain S. Duff. 1990. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.* 16, 1 (1990), 1–17.
[29] Demosthenes Ellinas and Peter D. Jarvis. 2019. Quantum channel simulation of phylogenetic branching models. *J. Phys. A: Math. Theoret.* 52, 11 (2019), 115601.
[30] Joseph Felsenstein. 1981. Evolutionary trees from DNA sequences: A maximum likelihood approach. *J. Molec. Evolut.* 17, 6 (1981), 368–376.
[31] Walter M. Fitch. 1971. Toward defining the course of evolution: Minimum change for a specific tree topology. *System. Biol.* 20, 4 (1971), 406–416.
[32] Tomas Flouri, F. Izquierdo-Carrasco, Diego Darriba, Andre J. Aberer, L.-T. Nguyen, B. Q. Minh, Arndt Von Haeseler, and Alexandros Stamatakis. 2015. The phylogenetic likelihood library. *System. Biol.* 64, 2 (2015), 356–362.
[33] Adrien Goeffon, Jean-Michel Richer, and Jin-Kao Hao. 2008. Progressive tree neighborhood applied to the maximum parsimony problem. *IEEE/ACM Trans. Computat. Biol. Bioinf.* 5, 1 (2008), 136–145.
[34] David B. Goldstein and Michael E. Weale. 2001. Population genomics: Linkage disequilibrium holds the key. *Curr. Biol.* 11, 14 (2001), R576–R579.
[35] Pablo A. Goloboff. 1996. Methods for faster parsimony analysis. *Cladistics* 12, 3 (1996), 199–220.
[36] Pablo A. Goloboff, Santiago A. Catalano, J. Marcos Mirande, Claudia A. Szumik, J. Salvador Arias, Mari Källersjö, and James S. Farris. 2009. Phylogenetic analysis of 73 060 taxa corroborates major eukaryotic groups. *Cladistics* 25, 3 (2009), 211–230.
[37] Pablo A. Goloboff, James S. Farris, and Kevin C. Nixon. 2008. TNT, a free program for phylogenetic analysis. *Cladistics* 24, 5 (2008), 774–786.
[38] Jorge González-Domínguez, Sabela Ramos, Juan Touriño, and Bertil Schmidt. 2015. Parallel pairwise epistasis detection on heterogeneous computing architectures. *IEEE Trans. Parallel Distrib. Syst.* 27, 8 (2015), 2329–2340.
[39] Jorge González-Domínguez and Bertil Schmidt. 2015. GPU-accelerated exhaustive search for third-order epistatic interactions in case–control studies. *J. Computat. Sci.* 8 (2015), 93–100.
[40] Jorge González-Domínguez, Lars Wienbrandt, Jan Christian Kässens, David Ellinghaus, Manfred Schimmler, and Bertil Schmidt. 2015. Parallelizing epistasis detection in GWAS on FPGA and GPU-accelerated computing systems. *IEEE/ACM Trans. Computat. Biol. Bioinf.* 12, 5 (2015), 982–994.
[41] Kazushige Goto and Robert A. van de Geijn. 2008. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.* 34, 3 (2008), 1–25.
[42] B. Gropp. 2022. MPI. Retrieved from https://www.mpi-forum.org/.

[43] S. Hammarling, J. Dongarra, J. Du Croz, and R. Hanson. 1988. An extended set of Fortran basic linear algebra subprograms. *ACM Trans. Math. Softw.* 14, 1 (1988), 1–32.
[44] Laiq Hasan and Zaid Al-Ars. 2011. An overview of hardware-based acceleration of biological sequence alignment. *Computat. Biol. Appl. Bioinf.* Chapter 9 (2011), 187–202.
[45] Gibran Hemani, Athanasios Theocharidis, Wenhua Wei, and Chris Haley. 2011. EpiGPU: Exhaustive pairwise epistasis scans parallelized on consumer level graphics cards. *Bioinformatics* 27, 11 (2011), 1462–1465.
[46] Wenqin Huangfu, Shuangchen Li, Xing Hu, and Yuan Xie. 2018. Radar: A 3D-ReRAM based DNA alignment accelerator architecture. In *55th Annual Design Automation Conference*. 1–6.
[47] Biresh Kumar Joardar, Priyanka Ghosh, Partha Pratim Pande, Ananth Kalyanaraman, and Sriram Krishnamoorthy. 2019. NoC-enabled software/hardware co-design framework for accelerating k-mer counting. In *13th IEEE/ACM International Symposium on Networks-on-Chip*. 1–8.
[48] Lin Kang, Guijuan He, Amanda K. Sharp, Xiaofeng Wang, Anne M. Brown, Pawel Michalak, and James Weger-Lucarelli. 2021. A selective sweep in the Spike gene has driven SARS-CoV-2 human adaptation. *Cell* 184, 17 (2021), 4392–4400.
[49] Roman Kaplan, Leonid Yavits, Ran Ginosar, and Uri Weiser. 2017. A resistive cam processing-in-storage architecture for DNA sequence alignment. *IEEE Micro* 37, 4 (2017), 20–28.
[50] Server Kasap and Khaled Benkrid. 2009. A high performance FPGA-based core for phylogenetic analysis with Maximum Parsimony method. In *International Conference on Field-Programmable Technology*. IEEE, 271–277.
[51] Server Kasap and Khaled Benkrid. 2010. High performance phylogenetic analysis with maximum parsimony on reconfigurable hardware. *IEEE Trans. Very Large Scale Integ. Systems* 19, 5 (2010), 796–808.
[52] Jan Christian Kässens, Lars Wienbrandt, Jorge González-Domínguez, Bertil Schmidt, and Manfred Schimmler. 2015. High-speed exhaustive 3-locus interaction epistasis analysis on FPGAs. *J. Computat. Sci.* 9 (2015), 131–136.
[53] Kazutaka Katoh and Daron M. Standley. 2013. MAFFT multiple sequence alignment software version 7: Improvements in performance and usability. *Molec. Biol. Evolut.* 30, 4 (2013), 772–780.
[54] Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. 2018. GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies. *BMC Genom.* 19, 2 (2018), 23–40.
[55] Motoo Kimura. 1969. The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations. *Genetics* 61, 4 (1969), 893.
[56] Chuck L. Lawson, Richard J. Hanson, David R. Kincaid, and Fred T. Krogh. 1979. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw.* 5, 3 (1979), 308–323.
[57] Yongchao Liu and Bertil Schmidt. 2014. SWAPHI: Smith-Waterman protein database search on Xeon Phi coprocessors. In *IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, 184–185.
[58] Pavlos Malakonakis, Andreas Brokalakis, Nikolaos Alachiotis, Evripides Sotiriades, and Apostolos Dollas. 2020. Exploring modern FPGA platforms for faster phylogeny reconstruction with RAxML. In *IEEE 20th International Conference on Bioinformatics and Bioengineering (BIBE)*. IEEE, 97–104.
[59] Benoit Morel, Pierre Barbera, Lucas Czech, Ben Bettisworth, Lukas Hübner, Sarah Lutteropp, Dora Serdari, Evangelia-Georgia Kostaki, Ioannis Mamais, Alexey Kozlov, et al. 2021. Phylogenetic analysis of SARS-CoV-2 data is difficult. *Molecular Biology and Evolution* 38, 5 (2021), 1777–1791.
[60] Ho-Cheung Ng, Shuanglong Liu, and Wayne Luk. 2017. Reconfigurable acceleration of genetic sequence alignment: A survey of two decades of efforts. In *27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–8.
[61] Ricardo Nobre, Sergio Santander-Jiménez, Leonel Sousa, and Aleksandar Ilic. 2020. Accelerating 3-way epistasis detection with CPU+ GPU processing. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 106–126.
[62] NVIDIA, Péter Vingelmann, and Frank H. P. Fitzek. 2021. CUDA. Retrieved from https://developer.nvidia.com/cuda-toolkit.
[63] OpenBLAS Contributors. 2021. OpenBLAS. Retrieved from https://www.openblas.net.
[64] Gerd Pfeiffer, Stefan Baumgart, Jan Schröder, and Manfred Schimmler. 2009. A massively parallel architecture for bioinformatics. In *International Conference on Computational Science*. Springer, 994–1003.
[65] Jittima Piriyapongsa, Chumpol Ngamphiw, Apichart Intarapanich, Supasak Kulawonganunchai, Anunchai Assawamakin, Chaiwat Bootchai, Philip J. Shaw, and Sissades Tongsima. 2012. iLOCi: A SNP interaction prioritization technique for detecting epistasis in genome-wide association studies. In *BMC Genomics*, Vol. 13. Springer, 1–15.
[66] Christian Ponte-Fernández, Jorge González-Domínguez, and María J. Martín. 2020. Fast search of third-order epistatic interactions on CPU and GPU clusters. *Int. J. High Perform. Comput. Applic.* 34, 1 (2020), 20–29.
[67] Christian Ponte-Fernández, Jorge González-Domínguez, and María J. Martín. 2022. A SIMD algorithm for the detection of epistatic interactions of any order. *arXiv preprint arXiv:2201.02460* (2022).

[68] Frederico Pratas, Pedro Trancoso, Alexandros Stamatakis, and Leonel Sousa. 2009. Fine-grain parallelism using multi-core, cell/BE, and GPU systems: Accelerating the phylogenetic likelihood function. In *International Conference on Parallel Processing*. IEEE, 9–17.

[69] Jonathan K. Pritchard and Molly Przeworski. 2001. Linkage disequilibrium in humans: Models and data. *Amer. J. Hum. Genet.* 69, 1 (2001), 1–14.

[70] Konstantinos Prousalis and Nikos Konofaos. 2018. Improving the sequence alignment method by quantum multi-pattern recognition. In *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*. 1–4.

[71] Shaun Purcell, Benjamin Neale, Kathe Todd-Brown, Lori Thomas, Manuel A. R. Ferreira, David Bender, Julian Maller, Pamela Sklar, Paul I. W. De Bakker, Mark J. Daly, et al. 2007. PLINK: A tool set for whole-genome association and population-based linkage analyses. *Amer. J. Hum. Genet.* 81, 3 (2007), 559–575.

[72] Gaspar Ribeiro, Nuno Neves, Sergio Santander-Jiménez, and Aleksandar Ilic. 2021. HEDAcc: FPGA-based accelerator for high-order epistasis detection. In *IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 124–132.

[73] Marylyn D. Ritchie, Lance W. Hahn, Nady Roodi, L. Renee Bailey, William D. Dupont, Fritz F. Parl, and Jason H. Moore. 2001. Multifactor-dimensionality reduction reveals high-order interactions among estrogen-metabolism genes in sporadic breast cancer. *Amer. J. Hum. Genet.* 69, 1 (2001), 138–147.

[74] Fredrik Ronquist and John P. Huelsenbeck. 2003. MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics* 19, 12 (2003), 1572–1574.

[75] David Sankoff and Pascale Rousseau. 1975. Locating the vertices of a Steiner tree in an arbitrary metric space. *Math. Program.* 9, 1 (1975), 240–246.

[76] Sergio Santander-Jiménez, Aleksandar Ilic, Leonel Sousa, and Miguel A. Vega-Rodríguez. 2017. Accelerating the phylogenetic parsimony function on heterogeneous systems. *Concurr. Computat.: Pract. Exper.* 29, 8 (2017), e4046.

[77] Sergio Santander-Jiménez, Miguel A. Vega-Rodríguez, Jorge Vicente-Viola, and Leonel Sousa. 2019. Comparative assessment of GPGPU technologies to accelerate objective functions: A case study on parsimony. *J. Parallel Distrib. Comput.* 126 (2019), 67–81.

[78] Sergio Santander-Jiménez, Miguel A. Vega-Rodríguez, Antonio Zahinos-Márquez, and Leonel Sousa. 2020. GPU acceleration of Fitch's parsimony on protein data: From Kepler to Turing. *J. Supercomput.* (2020), 1–27.

[79] Eric W. Sayers, Mark Cavanaugh, Karen Clark, Kim D. Pruitt, Conrad L. Schoch, Stephen T. Sherry, and Ilene Karsch-Mizrachi. 2021. GenBank. *Nucleic Acids Res.* 49, D1 (2021), D92–D96.

[80] Arlina Shen, Han Fu, Kevin He, and Hui Jiang. 2019. False discovery rate control in cancer biomarker selection using knockoffs. *Cancers* 11, 6 (2019), 744.

[81] Yuelong Shu and John McCauley. 2017. GISAID: Global initiative on sharing all influenza data—From vision to reality. *Eurosurveillance* 22, 13 (2017), 30494.

[82] John Maynard Smith and John Haigh. 1974. The hitch-hiking effect of a favourable gene. *Genet. Res.* 23, 1 (1974), 23–35.

[83] T. F. Smith and M. S. Waterman. 1981. Identification of common molecular subsequences. *J. Molec. Biol.* 147, 1 (1981), 195–197. DOI: https://doi.org/10.1016/0022-2836(81)90087-5

[84] Alexandros Stamatakis. 2006. RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics* 22, 21 (2006), 2688–2690.

[85] Alexandros Stamatakis. 2014. RAxML version 8: A tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics* 30, 9 (2014), 1312–1313.

[86] John E. Stone, David Gohara, and Guochun Shi. 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. *Comput. Sci. Eng.* 12, 3 (2010), 66–73.

[87] D. Swofford. 2002. *PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4.0b10.* Vol. Version 4.0. DOI: https://doi.org/10.1111/j.0014-3820.2002.tb00191.x

[88] You Tang, Zhuo Li, Chao Wang, Yuxin Liu, Helong Yu, Aoxue Wang, and Yao Zhou. 2020. LDkit: A parallel computing toolkit for linkage disequilibrium analysis. *BMC Bioinf.* 21, 1 (2020), 1–8.

[89] Charalampos Theodoris, Nikolaos Alachiotis, Tze Meng Low, and Pavlos Pavlidis. 2020. qLD: High-performance computation of linkage disequilibrium on CPU and GPU. In *IEEE 20th International Conference on Bioinformatics and Bioengineering (BIBE)*. IEEE, 65–72.

[90] Yatish Turakhia, Bryan Thornlow, Angie S. Hinrichs, Nicola De Maio, Landen Gozashti, Robert Lanfear, David Haussler, and Russell Corbett-Detig. 2021. Ultrafast Sample placement on Existing tRees (UShER) enables real-time phylogenetics for the SARS-CoV-2 pandemic. *Nat. Genet.* 53, 6 (2021), 809–816.

[91] Kristel Van Steen. 2012. Travelling the world of gene–gene interactions. *Brief. Bioinf.* 13, 1 (2012), 1–19.

[92] Field G. Van Zee and Robert A. Van De Geijn. 2015. BLIS: A framework for rapidly instantiating BLAS functionality. *ACM Trans. Math. Softw.* 41, 3 (2015), 1–33.

[93] Maria Vasilarou, Nikolaos Alachiotis, Joanna Garefalaki, Apostolos Beloukas, and Pavlos Pavlidis. 2021. Population genomics insights into the first wave of COVID-19. *Life* 11, 2 (2021), 129.

[94] Xiang Wan, Can Yang, Qiang Yang, Hong Xue, Xiaodan Fan, Nelson L. S. Tang, and Weichuan Yu. 2010. BOOST: A fast approach to detecting gene-gene interactions in genome-wide case-control studies. *Amer. J. Hum. Genet.* 87, 3 (2010), 325–340.

[95] Meng Wang, Wei Jiang, Ronald Ching Wan Ma, and Weichuan Yu. 2016. GBOOST 2.0: A GPU-based tool for detecting gene-gene interactions with covariates adjustment in genome-wide association studies. In *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 1437–1437.

[96] Yue Wang, Guimei Liu, Mengling Feng, and Limsoon Wong. 2011. An empirical comparison of several recent epistatic interaction detection methods. *Bioinformatics* 27, 21 (2011), 2936–2943.

[97] John N. Weinstein, Eric A. Collisson, Gordon B. Mills, Kenna R. Mills Shaw, Brad A. Ozenberger, Kyle Ellrott, Ilya Shmulevich, Chris Sander, and Joshua M. Stuart. 2013. The cancer genome atlas pan-cancer analysis project. *Nat. Genet.* 45, 10 (2013), 1113–1120.

[98] Lars Wienbrandt. 2014. The FPGA-based high-performance computer RIVYERA for applications in bioinformatics. In *Conference on Computability in Europe*. Springer, 383–392.

[99] Lars Wienbrandt, Jan Christian Kässens, Jorge González-Domínguez, Bertil Schmidt, David Ellinghaus, and Manfred Schimmler. 2014. FPGA-based acceleration of detecting statistical epistasis in GWAS. *Proced. Comput. Sci.* 29 (2014), 220–230.

[100] Lars Wienbrandt, Jan Christian Kässens, Matthias Hübenthal, and David Ellinghaus. 2019. 1000× faster than PLINK: Combined FPGA and GPU accelerators for logistic regression-based detection of epistasis. *J. Computat. Sci.* 30 (2019), 183–193.

[101] Sandra Wienke, Paul Springer, Christian Terboven, and Dieter an Mey. 2012. OpenACC: First experiences with real-world applications. In *18th International Conference on Parallel Processing (Euro-Par'12)*. Springer-Verlag, Berlin, 859–870. DOI: https://doi.org/10.1007/978-3-642-32820-6_85

[102] Fang Liu, Jue Wang, Xian-Yu Lang, Chi-Xue Bin, Haipeng Zhao, and Jin-Sheng Lai. 2013. Fast computing of linkage disequilibrium on GPU. In *GPU Technology Conference*. Citeseer.

[103] Ziheng Yang and Bruce Rannala. 2012. Molecular phylogenetics: Principles and practice. *Nat. Rev. Genet.* 13, 5 (2012), 303–314.

[104] Ling Sing Yung, Can Yang, Xiang Wan, and Weichuan Yu. 2011. GBOOST: A GPU-based tool for detecting gene–gene interactions in genome–wide case control studies. *Bioinformatics* 27, 9 (2011), 1309–1310.

[105] Chi Zhang, Shan-Shan Dong, Jun-Yang Xu, Wei-Ming He, and Tie-Lin Yang. 2019. PopLDdecay: A fast and effective tool for linkage disequilibrium decay analysis based on variant call format files. *Bioinformatics* 35, 10 (2019), 1786–1788.

[106] Jianfu Zhou, Xiaoguang Liu, Douglas S. Stones, Qiang Xie, and Gang Wang. 2011. MrBayes on a graphics processing unit. *Bioinformatics* 27, 9 (2011), 1255–1261.

[107] Rigui Zhou and Qiulin Ding. 2008. Quantum pattern recognition with probability of 100%. *Int. J. Theoret. Phys.* 47, 5 (2008), 1278–1285.

[108] Stephanie Zierke and Jason D. Bakos. 2010. FPGA acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods. *BMC Bioinf.* 11, 1 (2010), 1–12.

[109] George S. Zubenko, Hugh B. Hughes III, and Wendy N. Zubenko. 2010. D10S1423 identifies a susceptibility locus for Alzheimer's disease (AD7) in a prospective, longitudinal, double-blind study of asymptomatic individuals: Results at 14 years. *Amer. J. Medic. Genet. Part B: Neuropsych. Genet.* 153, 2 (2010), 359–364.

[110] Derrick Joel Zwickl. 2006. *Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion*. Ph.D. Dissertation. University of Texas at Austin.

Received 26 May 2022; revised 26 December 2022; accepted 7 March 2023