





# Accelerating Phylogenetics Using FPGAs in the Cloud

Nikolaos Alachiotis , University of Twente, 7522 NB, Enschede, The Netherlands

Andreas Brokalakis , Vasilis Amourgianos , Sotiris Ioannidis, Pavlos Malakonakis , and Tasos Bokalis , Technical University of Crete, Chania, 731 00, Greece

*Phylogenetics study the evolutionary history of organisms using an iterative process of creating and evaluating phylogenetic trees. This process is very computationally intensive; constructing a large phylogenetic tree requires hundreds to thousands of CPU hours. In this article, we describe an FPGA-based system that can be deployed on AWS EC2 F1 cloud instances to accelerate phylogenetic analyses by boosting performance of the phylogenetic likelihood function, i.e., a widely employed tree-evaluation function that accounts for up to 95% of the overall analysis time. We exploit domain-specific knowledge to reduce the amount of transferred data that limits overall system performance. Our proof-of-concept implementation reveals that the effective accelerator throughput nearly quadruples with optimized data movement, reaching up to 75% of its theoretical peak and nearly  $10\times$  faster processing than a CPU using AVX2 extensions.*

Phylogenetic inference methods reconstruct the evolutionary history of a collection of organisms based on molecular genetic data (DNA or protein sequences). An evolutionary history is represented by a phylogeny (also referred to as phylogenetic tree), i.e., a binary-tree topology where the organisms under investigation (taxa) are located at the leaves, the inner nodes represent extinct common ancestors, and the length of each branch represents the evolutionary time between two nodes. An example is shown in Figure 1.

Phylogenetic trees find practical application in various scientific and industrial fields, such as conservation biology,<sup>1</sup> epidemiology,<sup>2</sup> forensics,<sup>3</sup> and drug development.<sup>4</sup> They are constructed through an iterative process of creating and evaluating tree topologies until one that best explains the available molecular data is found. However, a fundamental problem in phylogenetics lies in the overwhelming number of alternative tree topologies to be evaluated, which increases exponentially with the number of organisms. For 50 organisms, for instance, there already exist  $2.84 \times 10^{76}$

alternative trees, and although computationally inexpensive clustering methods exist, considerably more advanced ones such as maximum likelihood estimation (MLE) and Bayesian inference (BI) are preferred in real-world analyses due to providing a stronger statistical foundation.<sup>5</sup> Inevitably, practical studies resort to approximations and heuristics for constructing alternative tree topologies (also known as tree-space search strategies), which are then ranked using an evaluation function.

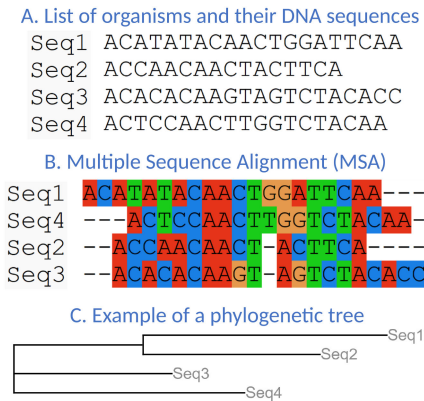
A fundamental phylogeny-evaluation function that is employed both in MLE and BI analyses is the phylogenetic likelihood function (PLF),<sup>6</sup> which calculates a likelihood score for a given tree topology. The PLF takes between 85% and 95% of the execution time of widely used phylogenetics tools, e.g., RAxML<sup>7</sup> and MrBayes,<sup>8</sup> which severely restricts how thoroughly the tree space can be searched in order to obtain results in a reasonable time frame. A recent study by Morel *et al.*<sup>9</sup> concluded 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 authors applied the PLF through RAxML, but the analysis was restricted to just 5% of the parameter space possibly

0272-1732 © 2021 IEEE

Digital Object Identifier 10.1109/MM.2021.3075848

Date of publication 27 April 2021; date of current version

2 July 2021.

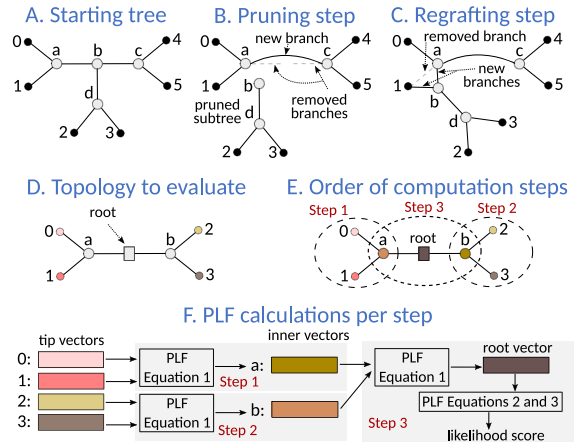


**FIGURE 1.** List of organisms and their DNA sequences (A) are aligned to create a multiple sequence alignment (B) that is used to find the phylogenetic tree that best fits the data (C).

due to the heightened urgency to reach conclusions because of the pandemic.

Evidently, it is of paramount importance to enable phylogenetic tools to search the tree space more thoroughly without being limited by the computational requirements of the PLF. To this end, this article initially presents a generic FPGA-accelerator architecture for the PLF. Motivated by the apparent need to ease the adoption of an accelerated solution in computational phylogenetics, we target datacenter-grade FPGA accelerator cards such as the Xilinx Alveo series, which are readily available for deployment in the cloud, e.g., through Amazon AWS EC2 F1 instances.

Previous efforts that employed FPGAs to accelerate the PLF<sup>10,11</sup> focused on exploiting parallelism to maximize throughput. The primary factor that limits PLF performance in current accelerator-rich servers, however, is not the computational capabilities of the available accelerators, but rather the data movement between the host CPU and discrete accelerator nodes. Deploying an optimized for throughput PLF accelerator on a EC2 F1 instance revealed that moving data to and from the FPGA card takes up to three times longer than the accelerator processing time; in other words, the custom hardware remains idle 75% of the time, waiting for data to become available. Interestingly, the PLF data-movement problem can be alleviated by exploiting the way tree-space search algorithms are practically applied in large-scale phylogenetics for the construction of phylogenetic trees with thousands of organisms. This article also describes a systematic way to achieve this, which reduces the amount of data that need to be transferred, and allows to deliver up to 75% of the



**FIGURE 2.** In a SPR move, a given tree (A) is first pruned (B), and the detached subtree is reattached to another branch to create a new tree (C). The likelihood of a tree (D) is computed through multiple PLF calls, starting at the tips and proceeding toward the root (E). Each step computes probability vectors of inner nodes (1) until the root vector is computed, which is then used (2) and (3) to calculate the likelihood of the tree (F).

theoretical peak performance of a throughput-optimized accelerator in the cloud.

## TREE SEARCH AND THE PLF

The starting point for a phylogenetic analysis is a list of organisms and their DNA or protein sequences [see Figure 1(A)]. Since these sequences can differ in length, a multiple sequence alignment (MSA) is initially created; it is a  $n \times m$  matrix of  $n$  sequences/rows and  $m$  DNA characters each, with gaps inserted into the sequences in such a way that each column (also referred to as alignment site) conveys information about the history of the organisms under study [see Figure 1(B)]. Thereafter, a tree topology is rapidly created (based on sequence similarities) and used as the starting tree to search the tree space [see Figure 1(C)].

A commonly used method for searching the tree space is illustrated in Figures 2(A)–2(C). It is a two-step algorithm that rearranges a given tree topology by first selecting and detaching a subtree from the given tree (pruning step), and then attaching it to another branch (regrafting step), thereby creating a new tree to evaluate. The aforementioned pair of topology-rearrangement steps is referred to as a subtree-pruning-and-regrafting (SPR) move, and the PLF is applied after every SPR move to calculate the likelihood score for the resulting tree topology. If the new

tree topology is found to have a higher likelihood than the pre-SPR-move one, the next SPR move is applied on the new topology. This iterative process of searching the tree space by applying SPR moves and evaluating the resulting tree topologies is repeated until no significant further likelihood improvements are obtained.

The likelihood of a phylogenetic tree is computed by recursively invoking the PLF (1), starting at the tips and proceeding toward the root, as depicted in Figures 2(D)–2(F). Every tree node  $x$  is represented by a probability vector  $\vec{L}_x$  that comprises  $m$  entries, where  $m$  is the number of alignment sites. A probability vector entry  $\vec{L}_x(i)$ ,  $i=1..m$  at position  $i$  of an inner node  $x$  contains the probabilities  $L^A$ ,  $L^C$ ,  $L^G$ , and  $L^T$  of observing nucleotides A, C, G, and T, respectively, given the nucleotides present in MSA site  $i$ . When  $x$  is a tip, the respective probability values are either 0.0 or 1.0, since the corresponding sequence is in the MSA.

Given probability vectors  $\vec{L}_A$  and  $\vec{L}_B$  of nodes A and B, the probabilities  $\vec{L}_C^u(i)$ ,  $u \in N$  and  $N = \{A, C, G, T\}$ , at position  $i$  of vector  $L_C$  that describes the common ancestor C are calculated using

$$\vec{L}_C^u(i) = \left( \sum_{s \in N} P_{u \rightarrow s}(t_l) \times \vec{L}_A^s(i) \right) \times \left( \sum_{s \in N} P_{u \rightarrow s}(t_r) \times \vec{L}_B^s(i) \right) \quad (1)$$

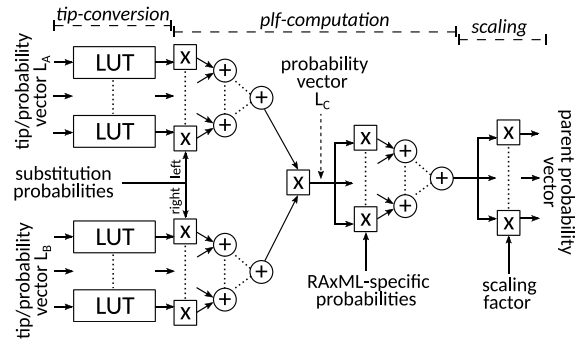
where  $t_l$  and  $t_r$  are the lengths of the branches that connect node C with its child nodes A and B, respectively, while  $P_{u \rightarrow s}(t)$  is the probability for a nucleotide  $u$  to be substituted by a nucleotide  $s$  given a branch length  $t$ . In practice, each probability vector entry consists of  $S \times C$  probabilities, where  $S$  is the number of DNA or protein states (4 or 20, respectively), and  $C$  is a constant (typically 4 or 8) that depends on certain evolutionary assumptions. Thus, (1) is computed  $m \times S \times C$  times for each tree node, where  $m$  is the number of alignment sites.

As previously mentioned, PLF computations start at the tips and proceed toward the root. Once the probability vector  $\vec{L}_{vr}$  of the root node  $vr$  is calculated, the final likelihood  $LH$  of the tree is computed in two steps. First, a likelihood score  $l(i)$  per position  $i$ ,  $i = 1..m$ , is calculated using

$$l(i) = \sum_{s \in N} \pi_s \times \vec{L}_{vr}^s(i) \quad (2)$$

where  $\pi_s$ ,  $s \in N$  and  $N = \{A, C, G, T\}$  are the prior probabilities of observing nucleotides A, C, G, and T at the root (computed from the MSA). Then, the final likelihood score of the tree is computed as the sum of the

### PLF accelerator arithmetic pipeline



**FIGURE 3.** Arithmetic pipeline is logically divided into three stages: (a) tip-conversion (binary data at the tips to probabilities), (b) plf-computation [(1) and RAXML-specific matrix-vector multiplication], and (c) scaling to prevent arithmetic underflow.

logarithm of the per-site likelihood scores using

$$LH = \sum_{i=1}^m \log(l(i)). \quad (3)$$

## ACCELERATOR ARCHITECTURE

Profiling RAXML, a state-of-the-art phylogenetic tool extensively optimized to reduce runtime, showed that over 95% of the PLF time is spent on calculating common-ancestor probability vectors. Hence, an FPGA accelerator for computing (1) was created using high-level synthesis with the Xilinx Vitis 2020.1 unified software platform. The PLF implementation of RAXML was used as a reference. Figure 3 illustrates the arithmetic pipeline, which is based on double-precision floating-point units and is logically divided into three stages: a) *tip-conversion*, b) *plf-computation*, and c) *scaling*.

Since all probabilities at the tips are exclusively 0.0 or 1.0, each of them is represented in memory by a single bit. When the probability vector of the common ancestor of tips is computed, the *tip-conversion* stage performs the necessary conversion of bits to double-precision values using multiple lookup tables for throughput. Thereafter, the *plf-computation* stage, which comprises three arrays of multipliers and an equal number of sum reduction trees, implements (1) and a RAXML-specific matrix-vector multiplication that is related to the way the nucleotide substitution probabilities  $[P_{u \rightarrow s}(t)$  in (1)] are calculated. The final *scaling* stage prevents arithmetic underflow when the PLF is applied on phylogenetic trees with many organisms/tips. It performs numerical scaling per probability vector entry by multiplying its probabilities by a large

constant if any of them is found to be below a certain threshold after the *plf-computation* stage.

The accelerator architecture is described by a series of parameters that define the number of lookup tables as well as the size of the multiplier arrays and the sum reduction trees. They can be set according to the memory subsystem and the available FPGA resources to create accelerators optimized for throughput by increasing parallelism and reducing the initiation interval.

We initially created a PLF accelerator for processing DNA data. It fully exploits the available memory bandwidth of the AWS EC2 F1 instance and 28% of the available FPGA resources to compute eight probabilities per clock cycle at 225 MHz. Thus, a probability vector entry is computed in two clock cycles since there are 16 probabilities per vector entry ( $S \times C = 4 \times 4 = 16$  because  $C = 4$  in RAxML).

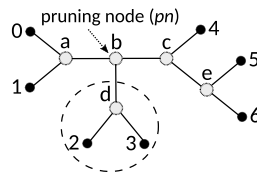
When protein sequences are processed, each probability vector entry comprises 80 probabilities ( $S = 20$  states,  $C = 4$ ). The available memory bandwidth of the AWS EC2 F1 instance, would, in theory, allow us to create an accelerator that calculates a probability vector entry in ten clock cycles. This, however, led to a resource-intensive design that could not be implemented due to routing congestion. Thus, we deployed a PLF accelerator that computes a probability vector entry in 20 clock cycles while occupying 51.5% of the device resources.

### OPTIMIZING DATA MOVEMENT

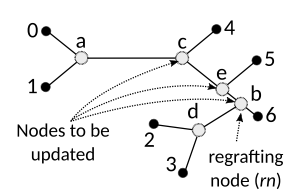
As previously mentioned, accelerator performance is limited by data movement. Computing the PLF requires the probability vectors of two child nodes (input) to be transferred to the accelerator card before processing, and the probability vector of the parent node (output) to be transferred back to the host for the tree-space search algorithm to continue. For both DNA and protein data, moving probability vectors to and from the accelerator reduces the effective throughput (seen by the host CPU) to as low as 18% of its theoretical peak. Note that, nucleotide substitution probabilities also need to be transferred to the accelerator in every call (128 probabilities for DNA, 3,200 for proteins), but the respective transfer time becomes negligible as the probability vector size increases. Thus, we focus next on optimizing the movement of probability vectors, which is the bottleneck in accelerating the PLF.

In practice, the pruned subtree in a SPR move is reattached to all nearby branches given a maximum distance  $d_{max}$  that indicates the number of inner

#### A. Before SPR move



#### B. After SPR move

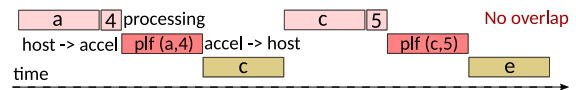


#### C. Partial Traversal Descriptor (PTD)

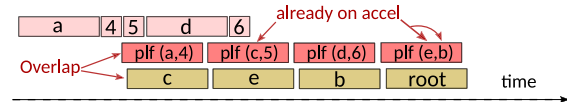
{plf(a,4) = c}, {plf(c,5) = e}, {plf(d,6) = b}, {plf(e,b) = root}

reuse of probability vectors already residing on the accelerator card

#### D. Sequential execution mode



#### E. Optimized data movement



**FIGURE 4.** SPR move (A-B) is followed by a partial traversal descriptor (C) that contains an ordered sequence of required PLF calls (the root is assumed on the branch connecting nodes e and b after regrafting). Exploiting data dependencies within the PTD in combination with double-buffering lead to a data-movement-optimized execution mode (E) instead of sequential execution (D).

nodes within the path between the pruning node and the regrafting node. Figure 4(A)–(B) illustrates an example of an SPR move where the pruned tree is reattached to a branch at a distance  $d_{pp \rightarrow rp} = 2$ , where  $pn$  and  $rn$  denote the pruning node and the regrafting node, respectively. After the SPR move, the nodes within the path between the pruning node and the regrafting node are updated via PLF (1) to reflect the topological changes introduced by the SPR move. At this point, we can exploit the data dependence between subsequent PLF calls in-between SPR moves to cache on the accelerator card probability vectors that will be reused shortly.

We introduced a series of accelerator runtime library routines into the source code of RAxML to control data movement based on the partial traversal descriptor (PTD, RAxML terminology) that is produced after every SPR move, i.e., an ordered sequence of PLF calls. Figure 4(C) shows the data dependencies among the four accelerator calls in the PTD that follows the SPR move shown in Figure 4(A)–(B). The PTD size can vary between 1 and  $n$  PLF calls, where  $n$  is the number of sequences. We observed, however, that over 90% of

the PTDs comprise between 2 and 8 PLF calls, which allows for the extensive reuse of data within a PTD, and across subsequent PTDs when the memory size on the accelerator card is sufficiently large.

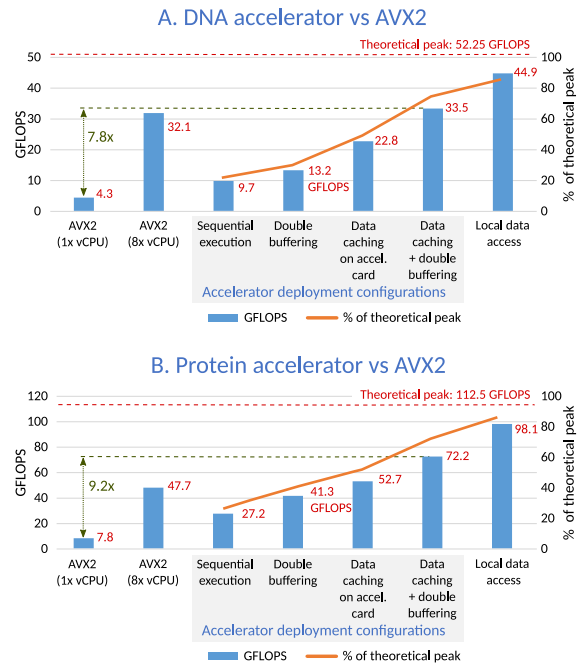
The PTD-driven movement of probability vectors reduces the overall amount of data that need to be transferred from the host to the accelerator. Since data coherency between the host and the accelerator is required for correct RAxML execution, each output probability vector is also transferred back to the host as soon as it is computed. We hide most of the time required for transferring data from the accelerator to the host with double-buffering; it allows to overlap the accelerator execution and data movement. This is depicted in Figure 4(E) in comparison with the sequential execution mode of Figure 4(D) (no data transfer/execution overlap).

### PERFORMANCE ASSESSMENT

To assess the performance, we run our modified version of standard RAxML (v.8.2) on a AWS EC2 F1 instance with a Xeon E5-2686v4 processor running at 2.7 GHz. The PLF accelerators were implemented on a Xilinx accelerator card with a Virtex UltraScale+ VU9P FPGA and 64 GB of DDR4 memory, organized into four 16-GB banks. We compared various accelerator deployment configurations with the PLF implementation in RAxML that employs Advanced Vector Extensions 2 (AVX2) and POSIX threads, and report throughput performance in floating-point operations per second. Figure 5(A) and (B) summarizes the results for DNA and protein processing, respectively.

*WE CAN OBSERVE THAT THE DATA TRANSFER TIME IN THE SEQUENTIAL EXECUTION CONFIGURATION REDUCES PLF ACCELERATOR PERFORMANCE TO AS LOW AS 18% AND 24% OF ITS THEORETICAL PEAK FOR DNA AND PROTEIN DATA, RESPECTIVELY.*

We can observe that the data transfer time in the sequential execution configuration reduces PLF accelerator performance to as low as 18% and 24% of its theoretical peak for DNA and protein data, respectively. Using double buffering to overlap data movement with execution improves throughput by 36% and 51% for DNA and protein data, respectively. However,



**FIGURE 5.** Various accelerator deployment configurations were compared with the AVX2 PLF implementation in RAxML in terms of throughput (GFLOPS) for DNA (A) and proteins (B). The “Local data access” bar provides the maximum practically attainable throughput (the accelerator is computing exclusively with local data stored in the onboard DDR4 memory), whereas the accelerator deployment configurations include all necessary data movement between the host and the accelerator node per case.

when probability vectors are cached on the accelerator node (up to 12 GB in our runs) and combined with double buffering, the effective accelerator performance reaches up to 75% of its practical maximum in both cases. Note that an inevitable performance loss (between 12% and 14.5% of the theoretical peak) was observed in all execution configurations due to the communication overhead for initiating the operation of the accelerator (transferring arguments) and filling its pipeline.

Furthermore, we can observe that the fully optimized accelerator deployment configurations, which couple data caching with double buffering, deliver, on average, 7.8x and 9.2x faster computation of PLF (1) than the respective AVX2 implementations for DNA and proteins on one vCPU (virtual CPU in AWS terminology, one vCPU is a thread of a CPU core). This led to an overall performance boost of up to 2x and 5x for the complete phylogenetic analysis of DNA and

protein sequences, respectively, using RAxML on the Amazon AWS cloud (EC2 F1).

Related work by Izquierdo-Carrasco *et al.*<sup>12</sup> explored the potential of GPUs for accelerating the PLF implementation of RAxML, supporting the processing of DNA sequences only. The study compared performance of a NVIDIA Tesla C2075 GPU with an Intel i5-3550 CPU, reporting 2× faster PLF execution on the GPU in comparison with the AVX implementation on the CPU. The overall performance boost for complete phylogenetic analyses ranged between 1.3× and 2.3×, which was achieved by offloading to the GPU not only the calculation of common-ancestor probability vectors but also the calculation of the final likelihood score (2) and (3).

---

IN THIS ARTICLE, WE DESCRIBED A  
GENERIC ACCELERATOR  
ARCHITECTURE FOR COMPUTING THE  
PLF USING DATACENTER-GRADE  
FPGAs IN THE CLOUD.

---

## CONCLUSION

In this article, we described a generic accelerator architecture for computing the PLF using datacenter-grade FPGAs in the cloud. We integrated throughput-optimized accelerators for DNA and protein sequences with the widely used software RAxML, and assessed performance on an Amazon EC2 F1 instance. We observed that data movement between the host and the accelerator node considerably deteriorates accelerator performance, and proposed a systematic way to exploit domain-specific knowledge (the way tree-search algorithms work in practice) to alleviate the data-movement problem, which led to 3.6× higher accelerator performance.

As future work, we intend to devise a scale-out solution to deploy multiple FPGA accelerators. Furthermore, we plan to explore the potential of custom-precision floating-point arithmetic to further boost accelerator performance. Lower precision arithmetic will increase scaling frequency, which yields such an approach impractical in software. On an FPGA, however, custom-precision arithmetic

cores will occupy less resources, thereby paving the way for more parallelism, while the scaling overhead is completely hidden since scaling is already in the pipeline.

## REFERENCES

1. L. Volkmann *et al.* "Prioritizing populations for conservation using phylogenetic networks," *PLoS One*, vol. 9, no. 2, 2014, Art. no. e88945.
2. T. T.-Y. Lam *et al.* "Use of phylogenetics in the molecular epidemiology and evolutionary studies of viral infections," *Crit. Rev. Clin. Lab. Sci.*, vol. 47, no. 1, pp. 5–49, 2010.
3. A. B. Abecasis *et al.* "Phylogenetic analysis as a forensic tool in HIV transmission investigations," *AIDS*, vol. 32, no. 5, pp. 543–554, 2018.
4. A. D. Yermanos *et al.* "Tracing antibody repertoire evolution by systems phylogeny," *Front. Immunol.*, vol. 9, 2018, Art. no. 2149.
5. M. J. Zvelebil and J. O. Baum, *Understanding Bioinformatics*. New York, NY, USA: Garland Science, 2007.
6. J. Felsenstein, "Evolutionary trees from DNA sequences: A maximum likelihood approach," *J. Molecular Evol.*, vol. 17, no. 6, pp. 368–376, 1981.
7. A. Stamatakis, "RAxML version 8: A tool for phylogenetic analysis and post-analysis of large phylogenies," *Bioinformatics*, vol. 30, no. 9, pp. 1312–1313, 2014.
8. F. Ronquist *et al.* "MrBayes 3.2: Efficient Bayesian phylogenetic inference and model choice across a large model space," *Systematic Biol.*, vol. 61, no. 3, pp. 539–542, 2012.
9. B. Morel *et al.* "Phylogenetic analysis of SARS-Cov-2 data is difficult," *Molecular Biol. Evol.*, vol. 38, pp. 1777–1791, 2021.
10. P. Malakonakis, A. Brokalakis, N. Alachiotis, E. Sotiriades, and A. Dollas, "Exploring modern FPGA platforms for faster phylogeny reconstruction with RAxML," in *Proc. IEEE 20th Int. Conf. Bioinf. Bioeng.*, 2020, pp. 97–104.
11. S. Zierke and J. D. Bakos, "FPGA acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods," *BMC Bioinf.*, vol. 11, no. 1, 2010, Art. no. 184.
12. F. Izquierdo-Carrasco *et al.* "A generic vectorization scheme and a GPU kernel for the phylogenetic likelihood library," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum*, 2013, pp. 530–538.

**NIKOLAOS ALACHIOTIS** is an Assistant Professor with the University of Twente, Enschede, The Netherlands. His research interests are in the areas of computer architecture, reconfigurable computing, HPC, and bioinformatics. Alachiotis received a Ph.D. degree in computer science from the Technische Universitaet Munchen, Munich, Germany. Contact him at [n.alachiotis@utwente.nl](mailto:n.alachiotis@utwente.nl).

**ANDREAS BROKALAKIS** is currently working toward a Ph.D. degree with the School of Electrical and Computer Engineering, Technical University of Crete (TUC), Chania, Greece. His research interests are in the areas of computer architecture for embedded and HPC systems, computer arithmetic, and reconfigurable computing. Brokalakis received a B.Eng. degree in computer engineering and informatics and an M.Eng. degree on integrated software/hardware systems from the University of Patras, Patras, Greece. He is a member of IEEE. Contact him at [abrokalakis@mhl.tuc.gr](mailto:abrokalakis@mhl.tuc.gr).

**VASILIS AMOURGIANOS** is currently a researcher of hardware and network security at Technical University of Crete (TUC), Chania, Greece. His research interests include security in computer architecture, HPC systems, and reconfigurable computing. Amourgianos received an M.Eng. degree in electronics and computer engineering from TUC and is currently working toward an M.Sc. degree with the same university. Contact him at [vamourgianos@isc.tuc.gr](mailto:vamourgianos@isc.tuc.gr).

**SOTIRIS IOANNIDIS** is currently an Associate Professor with Technical University of Crete (TUC), Chania, Greece, and the Director of the Microprocessor and Hardware Laboratory. His research interests include the areas of systems and network security, security policy, privacy, and high-speed networks. Ioannidis received a Ph.D. degree from the University of Pennsylvania, Philadelphia, PA, USA. Contact him at [sotiris@ece.tuc.gr](mailto:sotiris@ece.tuc.gr).

**PAVLOS MALAKONAKIS** is currently working toward a Ph.D. degree with the School of Electrical and Computer Engineering, Technical University of Crete (TUC), Chania, Greece. His research interests are in the areas of reconfigurable computing, computer architecture, embedded systems and FPGA-based HPC systems. Malakonakis received a B.Eng. degree in electronics and computer engineering and an M.Eng. degree in computer engineering from TUC. Contact him at [pmalakonakis@mhl.tuc.gr](mailto:pmalakonakis@mhl.tuc.gr).

**ANASTASIOS BOKALIDIS** is an ECE Engineer. His research interests include the areas of computer architecture, reconfigurable computing, HPC, and embedded systems. Bokalidis received a B.Eng. degree in electronics and computer engineering from the School of Electrical and Computer Engineering, Technical University of Crete (TUC), Chania, Greece. Contact him at [abokalidis@isc.tuc.gr](mailto:abokalidis@isc.tuc.gr).