Exploring Modern FPGA Platforms for Faster Phylogeny Reconstruction with RAxML

Pavlos Malakonakis
Electrical and Computer Engineering
Technical University of Crete
Chania, Greece
pmalakonakis@mhl.tuc.gr

Andreas Brokalakis
Electrical and Computer Engineering
Technical University of Crete
Chania, Greece
abrokalakis@mhl.tuc.gr

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

Evripides Sotiriades
Electrical and Computer Engineering
Technical University of Crete
Chania, Greece
esot@mhl.tuc.gr

Apostolos Dollas
Electrical and Computer Engineering
Technical University of Crete
Chania, Greece
dollas@ece.tuc.gr

***Abstract*—The Phylogenetic Likelihood Function (PLF) is one of the cornerstone functions in most phylogenetic inference tools; its execution represents the majority of time required to complete an analysis. This work proposes the acceleration of this function using reconfigurable hardware accelerators, focusing on system-on-chips that integrate Field Programmable Gate Array (FPGA) resources as well as traditional High Performance Computing (HPC) systems that use FPGA-based accelerator cards. Taking into account the specific properties of each platform in order to exploit their processing capabilities, the proposed solutions provide significant performance gains. The measured acceleration of PLF function is up to 8x while the overall time to complete a phylogenetic analysis using the popular RAxML software can be reduced up to 3.2 times (with respect to a pure software implementation on a high-end server processor). Compared to other similar solutions proposed in literature, our systems perform up to 65% faster.**

***Index Terms*—phylogenetic inference, phylogenetic tree, RAxML, PLF, phylogenetic likelihood, FPGA, Reconfigurable hardware, Amazon AWS F1**

I. Introduction

Phylogenetic inference is used in computational biology to construct phylogenetic trees, or phylogenies, based on molecular sequence data. A phylogeny is a tree structure that represents the evolutionary history of a set of species. The input to a phylogenetic analysis is a multiple sequence alignment (MSA) that is produced by a pre-processing step and determines which nucleotides share a common evolutionary history. An MSA is a $k \times m$ matrix that comprises k DNA sequences of m nucleotides each. The analysis can be conducted by applying character-based inference methods such as Maximum Parsimony [1] or Maximum Likelihood (ML) [2] on the aligned sequences.

The output of a phylogenetic analysis is an unrooted binary tree topology. The extant species (for which DNA data can be sequenced and is available) are assigned to the leaves (taxa) of such a tree, whereas the inner nodes represent hypothetical extinct common ancestors. In general, a scoring function and a tree-search strategy are required to reconstruct a phylogenetic tree from an MSA. Finding the best-scoring tree under Maximum Likelihood is known to be NP-hard [3]. The scoring criteria are used to assess how well a specific tree topology explains the underlying molecular sequence data.

The Phylogenetic Likelihood Function (PLF) represents one of the most widely used optimality criteria to score and thus choose among distinct evolutionary scenarios (phylogenetic trees). Numerous phylogenetic inference tools implement the PLF, either for standard ML-based optimization (RAxML [4], GARLI [5], PHYML [6]) or for Bayesian phylogenetic inference (MrBayes [7], PhyloBayes [8]). In all these tools, the PLF dominates the execution time, reaching up to 95% of the total analysis time. Consequently, acceleration of the PLF function is important to the bioinformatics community.

In this paper we propose the use of reconfigurable hardware (FPGAs) in order to accelerate the execution of the PLF function, and thus the overall phylogenetic analysis. The focus is on the PLF implementation of the widely employed phylogenetic inference software RAxML (Randomized Axelerated Maximum Likelihood) [4]. We propose two systems that execute the RAxML software on general purpose processors that are tightly coupled with FPGA fabric. The PLF is implemented as an FPGA-based co-processor. The first system is realized on a Xilinx ZCU102 development board [9], which incorporates a system-on-chip that tightly couples reconfigurable logic with four general purpose ARM A53 processors. The second platform employed in this study is the Amazon AWS F1 cloud instance [10], where FPGA accelerator boards are connected to Xeon E5v4 processors through PCIe.

In the remainder of this paper, Section II describes the mathematical foundation to compute the PLF. Section III discusses relevant previous works. Section IV presents the proposed system architectures of the two platforms, and Section V has the associated experimental results and performance evaluation. Finally, Section VI concludes this work.

2471-7819/20/$31.00 ©2020 IEEE
DOI 10.1109/BIBE50027.2020.00024

II. Phylogenetic Likelihood Function

RAxML employs the Phylogenetic Likelihood Function (PLF) as the scoring function for maximum likelihood estimation. The Felsenstein pruning algorithm [2] is the standard method used for computing the PLF and the overall log likelihood score of a tree topology. The rest of this section provides an abstract description of this algorithm.

The algorithm begins by identifying a pair of child nodes i and j in the given tree for which the likelihood vector at the common ancestor $k(1 \leq i, j, k \leq 2n-2)$ has not been computed yet. Given nodes i and j, the second step is the calculation of the ancestral probability vector entries at k and to subsequently prune the child nodes from the tree. These steps are executed recursively (by means of a post-order tree traversal) until the probability vector at the virtual root vr is calculated. At that point, the pruning process has transformed the initial tree to only one node (ancestral probability vector) that is located at the virtual root of the tree. Phylogenetic trees under ML are unrooted for mathematical and computational reasons [2], but a virtual root vr can be placed into any branch of the tree to evaluate its likelihood score.

To compute the PLF on a given tree, apart from the tree shape itself, one also has to compute the branch lengths and the parameters of the statistical nucleotide substitution model. For DNA data, a model of nucleotide substitution is provided by a 4×4 matrix (a 20×20 matrix is used for protein data) that is usually denoted as Q matrix. The Q matrix contains the instantaneous transition probabilities (for infinitesimal relative evolutionary time dt) of a nucleotide A to change into a nucleotide A, C, G, or T etc. To compute the nucleotide substitution probabilities for a given branch length t (t represents the evolutionary time between two nodes in the tree), one has to compute the following:

$$P(t) = e^{Qt} \tag{1}$$

This is usually implemented via an eigenvalue/eigenvector decomposition. Thus, to compute the likelihood on a fixed tree with given branch lengths and model parameters, one initially needs to apply the Felsenstein pruning algorithm and subsequently compute the overall likelihood score of the tree based on the ancestral probability vector at the virtual root.

Every probability vector entry $\vec{L(c)}$ at a position c, with $c = 1...m$, contains the four probabilities $P(A)$, $P(C)$, $P(G)$, $P(T)$ of observing a nucleotide A, C, G, or T at a specific site c of the input alignment. The probabilities at the leaves of the tree for which observed data (DNA sequences) is available are set to 1.0 for the observed nucleotide character at the respective position c (e.g., for the nucleotide A: $\vec{L(c)} = (1.0, 0.0, 0.0, 0.0)$).

Given probability vectors $\vec{L_i}$ and $\vec{L_j}$ of child nodes i and j, respectively, each of the four probability values $\vec{L}^u_C(i)$, $u \in N$ and $N = \{A, C, G, T\}$, at location c of the probability vector $\vec{L_C}$ that describes the immediate common ancestor C is computed using the following equation:

$$\vec{L}^u_C(c) = \big(\sum_{s \in N} P_{u \to s}(t_l) \times \vec{L}^s_i(c)\big) \times \big(\sum_{s \in N} P_{u \to s}(t_r) \times \vec{L}^s_j(c)\big), \tag{2}$$

where t_l and t_r are the lengths of the branches that connect parent node C with child nodes i and j, respectively, while $P_{u \to s}(t)$ is the nucleotide substitution probability for a nucleotide u to mutate to a nucleotide s given a branch length t. The nucleotide substitution probabilities are computed using Equation (1).

In real-world analyses, the statistical model of nucleotide substitution is extended by additional parameters to account for rate heterogeneity among alignment sites, i.e., the biological fact that genes evolve at different rates. A widely used model to describe rate variation among sites in phylogenetic inference is the Γ model [12], which assumes that rates over sites are random variables drawn from a Γ distribution, integrating the log-likelihood over the Γ function. To yield a computationally tractable solution, this integral is approximated by discretizing the Γ function into typically 4 or 8 discrete rates. Computing the PLF under the Γ model entails the calculation of Equation (2) independently for each discrete rate, resulting to the calculation of a probability vector $\vec{L_x}$ per discrete rate per inner node x. When N discrete Γ rates are used for phylogenetic inference based on DNA sequences, each comprehensive probability vector entry $\vec{L_x}(c)$ contains a total of $N \times 4$ probability values per genomic location.

When the branch with the virtual root is reached, following a number of FPA steps, a likelihood score $l(c)$ per site c is computed based on the probability vector $\vec{L_{vr}}$ at the virtual root using (3):

$$l(c) = \sum_{s \in N} \pi_s \times \vec{L}^s_{vr}(c), \tag{3}$$

where π_s, $s \in N$ and $N = \{A, C, G, T\}$, are the prior probabilities (typically referred to as base frequencies) of observing nucleotides A, C, G, and T at the virtual root. The final likelihood score of the tree is computed as the sum of the logarithm of the per-site likelihood scores using Equation 4:

$$LH = \sum_{c=1}^{m} log(l(c)). \tag{4}$$

It should be noted that state-of-the-art ML inference programs, such as RAxML [4] (used in this work), deploy a Newton-Raphson iterative procedure to optimize the branch lengths and improve the final likelihood score given the tree topology and the nucleotide substitution model.

III. Related Work

There exist multiple works on FPGA-based accelerators for PLF implementation. More specifically, in [13] and [14] Mak and Lam map a PLF implementation with reduced floating-point precision to reconfigurable logic. The performance tests reported in [13] and [14] have been conducted on trees with only 4 leaves (4 input sequences). Hence, scalability beyond 4 species trees is not addressed.

In [15] Alachiotis *et al.* present the implementation of an FPGA-based architecture of the phylogenetic maximum likelihood (ML) function and compare the performance to an efficient software (C) implementation on a high-end multi-core CPU with 16 cores. The initial implementation of the ML function for trees comprising 4 up to 512 sequences on an FPGA yields average speedups of a factor 8.3 vs. execution on a single-core, and is faster than the OpenMP-based parallel implementation on up to 16 cores. Consequently, Alachiotis *et al.*, in [16], present an updated architecture that benefits from the large number of DSP modules available in the FPGA in order to map the PLF. They validate and assess performance against a highly optimized and parallelized software implementation of the PLF that is based on RAxML. Both software and hardware implementations use double precision floating point arithmetic. The updated architecture achieves speedups ranging from 1.6 up to 7.2 compared to a dual-core processor running the OpenMP-based multi-threaded version of the PLF.

Bakos *et al.*, in [17], propose a co-processor architecture to accelerate median-based phylogenetic reconstruction for gene-rearrangement data. They integrated the hardware-based median computation into the GRAPPA toolset and achieved an average speedup of 189 over the entire phylogenetic reconstruction procedure. Another work by Zierke and Bakos presented in [18], uses the MrBayes 3 tool as a framework and utilizes an FPGA-based co-processor that executes the PLF. For large datasets, the authors estimate that the accelerated MrBayes, if run on a current-generation FPGA, achieves a 10× speedup relative to software running on a state-of-the-art server-class microprocessor. The FPGA-based implementation achieves its performance by deeply pipelining the likelihood computations, performing multiple floating-point operations in parallel, and through a natural log approximation that is chosen specifically to leverage a deeply pipelined custom architecture.

In [19] Berger *et al.* presented an accelerator architecture particularly optimized for 4-state input data (i.e., nucleotide characters) and described how to efficiently handle n-state data, with $n > 4$, as for example in the case of protein sequences or RNA secondary-structure data. The authors also proposed a flexible communication mechanism to enable the widely used software RAxML [4] to offload computation to the FPGA accelerator, reporting up to 4.3x faster processing vs. a 256-bit wide AVX-based (Advanced Vector Extensions) heavily optimized sequential implementation.

A more recent work by Alachiotis *et al.* [20] utilizes near-memory computation units (NMUs) within a FPGA-based computing environment with disaggregated memory to alleviate the data movement problem and improve performance and energy efficiency when inferring large-scale phylogenies. They report up to 22x better FLOPS performance and 13x higher power efficiency (FLOPS/Watt) over the standard accelerator-as-a-coprocessor model with explicit remote data transfers.

IV. Implementation on FPGA platforms

In our work we consider two different FPGA platforms. The first one is based on Zynq UltraScale MPSoC devices [11] that integrate in the same chip general purpose CPU cores and FPGA fabric. Although such devices are primarily aimed at embedded systems and related applications, recent research on heterogeneous High Performance Computing systems [21] proposes their adoption in such environments because of their high performance to energy efficiency. The second platform that we examine is a traditional HPC accelerator, i.e., a high-capacity, high-performance FPGA device with direct-attached memory in a PCIe accelerator card (in the same manner as widely used compute accelerators such as GPUs), connected to a server-grade high performance x86 CPU. The specific platform that we employ is available through the Amazon AWS cloud as part of their EC2 F1 instances. In the subsections that follow, we present the two different platforms in more detail and outline the specific implementations that take advantage of the characteristics of each one.

A. Zynq UltraScale+ MPSoC Platform

The first platform is based on a Zynq Ultrascale+ MPSoC. The specific board used is the ZCU102 Evaluation Board [9] that incorporates a Xilinx Zynq Ultrascale+ MPSoC (XCZU9EG) featuring four Arm Cortex-A53 64-bit processors tightly coupled with UltraScale+ reconfigurable fabric. It should be noted that the Zynq MPSoC also contains a Cortex-R5 dual-core real-time processor and a Mali-400 graphics processing unit (GPU), however, for the current work these resources are not employed. The Cortex-A53 processors plus the static logic on the chip are referred to as the Processing System (PS) while the reconfigurable region is called Programmable Logic (PL). The board has 4GB of DDR4 memory connected to the PS and another 512MB directly connected to the PL.

As mentioned above, the PLF dominates the execution time of the RAxML tool (up to 95% of the total execution time). As such, this makes it a natural target to try to accelerate through custom hardware on the PL. The rest of the RAxML application is executed by the ARM Cortex-A53 processor.

One of the most distinct characteristics of the Zynq MPSoC is the high degree of integration between the general purpose CPUs and the accelerators on the device. As such, the FPGA fabric is connected with up to six high-throughput AXI interfaces with the four A53 CPUs and the memory connected to the PS. Two of these interfaces can be coherent with the caches of the processors, while the other ports can be connected to Direct Memory Access (DMA) engines that control data flow between memory and the PL without requiring processor intervention. Each port can be configured to be up to 128-bits wide and streaming modes are also supported. Thus the effective throughput between main memory/CPU cores and the programmable logic can be massive while keeping the communication latency at very low levels. The main advantages of this platform are twofold. First, the data stored in the main memory are accessible by both the PS and the PL removing the necessity of multiple copies and data transfers. Second, the wide stream interfaces allow the implementation of a wide-datapath parallel stream accelerator architecture since data can be provided in such a way. The main disadvantage of course is

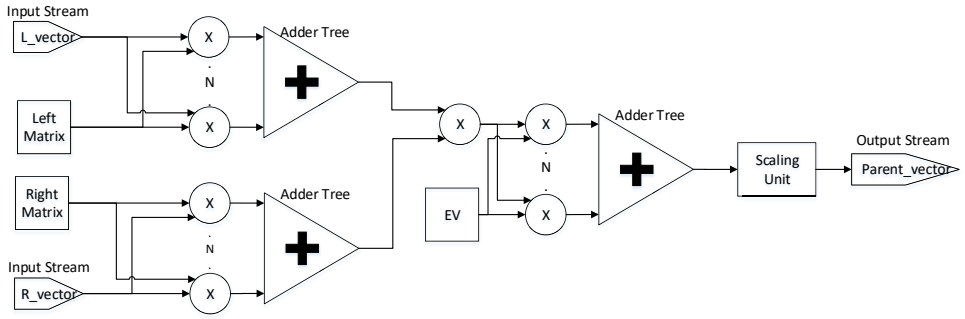

Fig. 1. Architecture of the PLF core

the ARM Cortex-A53 processor as the processing unit for all other tasks, as it is about eight times slower than a state of the art Intel Xeon processor when executing the RAxML software (per thread). It should be noted, however, that this drawback only arises on corner cases of data sets whose processing does not involve the extended levels of 70% to 95% PLF usage which are typically observed.

The PLF core was designed using the Vivado HLS 2019.2 tool. The architecture of the PLF core is shown in Fig 1. The PLF pipeline uses arrays of double-precision floating-point multipliers that perform all required operations per received datum in parallel, and relies on logarithmic adder trees to yield a pipelined datapath. Each multiplier array consists of N multipliers, where N is the alphabet size (N=4 for DNA). Initially the interface is automatically set to 64 bits as the input/output arguments are double precision floating point numbers. By providing the HLS pipeline pragma, each core has a deep pipeline which produces a result every 16 clock cycles, after 114 cycles to produce the first result. This is because the core requires 16 input values per result, with one double precision value being read during every clock cycle. Streaming interfaces are utilized to read/write the data to the DDR memory. More specifically, two DMA controllers are utilized, as the core uses two input streams for the left (L_vector) and right (R_vector) child node vectors and one output stream for the result parent vector. Also, a memory mapped interface is used to provide the core with the values for the left/right probability matrices as well as the eigenvector (EV), all of which are initialized prior to the streaming phase.

In order to increase the performance, the streaming interface was widened to 128 bits from 64 bits. This allows the utilization of the full width of the PS-PL ports, which required changes to the software and hardware so that the split/merge of the data could be done. This way, once full after 92 cycles (vs. 114 cycles in the 64-bit version), the pipeline can produce a result in 8 (rather than 16) clock cycles, doubling the processing throughput of the accelerator core.

Initial resource utilization results allow the placement of four PLF cores inside the PL. The limiting factor though proved to be the PS-PL bandwidth. As mentioned above,

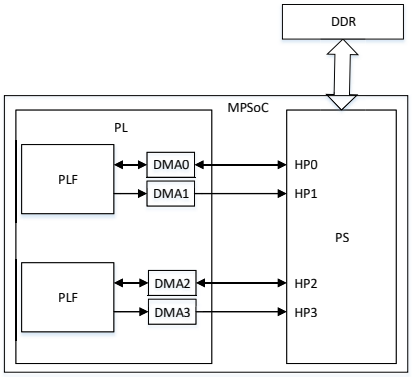

Fig. 2. Block Diagram of two PLF cores on the ZCU102 MPSoC

TABLE I
RESOURCE UTILIZATION OF TWO PLF CORES ON THE ZCU102 PLATFORM.

	Block RAM	DSP48E	FF	LUT
Utilized	19	734	143128	87812
Available	912	2520	548160	274080
Utilization (%)	2	29	26	32

each PLF core requires two DMA controllers in order to read the data from the DDR memory. The bandwidth required by each core is at about 7GB/s. The timing results show that two PLF cores consume the available PS-PL bandwidth when running at 250MHz. As a result, even though the resource utilization allows the placement of more accelerators it is meaningless as the bottleneck is the I/O bandwidth. The top level block diagram of the system is shown in Fig 2. The resource utilization for the two PLF cores, along with the DMA controllers, is presented in Table I.

B. Amazon AWS F1 Instance

The Amazon AWS F1 instances [10] provide remote access to servers that couple high-performance Intel Xeon processors with up to eight FPGA accelerator cards connected via PCIe.

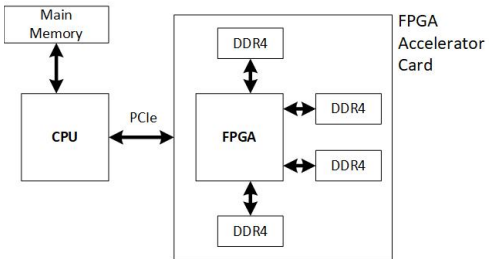

Fig. 3. Simplified view of the topology of the AWS F1 FPGA platform.

The specific instance that we have employed (f1.2xlarge) offers 8 vCPUs (Xeon E5-2686v4) coupled with 122GB of main memory and 470GB SSD storage and is connected to a single FPGA board through x16 PCIe-3.0. The FPGA board hosts a Xilinx Virtex UltraScale+ VU9P FPGA with 64GB of on-board DDR4 memory arranged in four independent memory banks that can be accessed in parallel.

Compared to the Zynq-based platform, in this case the topology follows a more typical organization (Fig. 3), where the CPU is connected to the main memory and through PCIe to the FPGA accelerator card. The CPU has to access data and orchestrate explicit data transfers to/from the FPGA accelerator. Data can be placed on any of the four available memory banks and the FPGA can access those memory resources in parallel. The communication between the CPU and the FPGA is orchestrated through a runtime environment (Xilinx XRT) and is programmed using OpenCL calls.

The architecture of the PLF accelerator we designed is based on the Decoupled Access Execute design paradigm [23], [24]. The main execution pipeline (Execution Unit - EX) is similar to the one presented in Fig 1, however the inputs and outputs are handled by separate units, called Access Units (AU) - the decoupled access-execute conceptual architecture. The overall system functions as a dataflow streaming pipeline. In total seven Access Units are employed (two units for the two input vectors, three units for the R, L and EV matrices, one unit for the weight scaling vector and one output unit for the result matrix) and a single wide Execution Unit.

An invocation of the accelerator consists of two steps. First, the Left- and Right-matrix AUs retrieve the left and right probability matrices, and the EV AU retrieves the inverted eigenvector (RAxML computes P(t) matrices based on eigenvector/eigenvalue decomposition [4]) from memory and store them into register files. Then, the two FIFO-based AUs fetch the APV vectors L_vector and R_vector that correspond to the left and right child nodes, and stream them through the PLF datapath (Fig. 1). This computes Eq. 2, performs multiplication with the inverted eigenvector, and scales up the results if needed. The output Parent_vector is stored in memory through a FIFO-based AU. The AUs that prefetch data into register files do not contain FIFOs to lower resource utilization.

As mentioned above, the FPGA device on the accelerator card has access to four independent channels of DDR4 memory. Each memory channel can be accessed by a separate memory controller, thus enabling the simultaneous accesses to all memory channels. The memory controllers allow up to 512bits wide connections to the FPGA compute resources through an AXI stream interface. Our design makes use of two such interfaces in order to transfer the two matrices from memory to the EX unit and another one to transfer the results back to memory. A fourth interface to the remaining memory channel (64bits wide) is used for the R, L and EV vectors as well as the scaling factors.

Compared to the Zynq MPSoC implementation, the initiation interval for the execution pipeline is much shorter, at two cycles, thus making the actual accelerator a very high performance one. However, since all data have to be transferred from the host to the accelerator card before being processed (and similarly results have to be transferred from the accelerator card to the host after the computation), significant time is required for I/O operations. Typically, our experiments demonstrated that the processing time is about half (or even less) vs. the time required to perform data transfers.

In order to reduce the data transfer to computation time ratio, we employed a double buffering mechanism taking advantage of the support provided by the OpenCL and Xilinx XRT runtime support - thus it does not require any significant hardware changes, however proper computation scheduling and synchronization mechanisms have to be implemented in software. We divide the data that have to be processed in smaller groups and transfer them to the accelerator group-by-group. The accelerator processes each data group once all data belonging to that group are made available. During computation time, the next group of data is transferred to the accelerator and thus data transfers and computation overlap. On the best case scenarios, this approach enables up to almost 40% reduction in the overall execution time of the PLF as viewed by the user. A more detailed analysis of the performance will be provided in the next section.

In terms of hardware resources required, Table II provides an overview of the device resource utilization. It should be noted that Table II presents the resources required by our accelerator and the resources required by the Amazon Shell (that is, the resources occupied by the Amazon F1 platform including control logic, communication through the PCIe interface, clocking resources, etc. [22]). The accelerator operates at a clock frequency of 222MHz.

The resource utilization numbers in Table II indicate the potential to use more accelerator instances as in the case of the design in the Zynq MPSoC. However, as mentioned above, the computation is I/O bound, and this approach would be of no (further) benefit.

V. Performance Evaluation of FPGA Platforms

In this section we provide an analysis of the measured performance of the PLF function in both platforms considered. Our results are compared with the PLF executed in software on the application processors of both platforms. The latter form the baseline with which we evaluate the effectiveness of our proposed solutions. Finally, we highlight the similarities and

TABLE II
RESOURCE UTILIZATION OF PLF ON THE VIRTEX ULTRASCALE+ VU9P FPGA.

	Block RAM	DSP48E	FF	LUT
Utilized (Platform)	339	9	330466	222466
Utilized (Accelerator)	30	1448	205274	99104
Available	2160	6840	2363536	1181768
Overall Utilization (%)	17.09%	21.3%	22.67%	27.21%

TABLE III
EXECUTION TIME OF THE PLF FUNCTION ON THE ZCU102 BOARD. SOFTWARE IS EXECUTED ON THE ARM CORTEX-A53 PROCESSOR. N REFERS TO THE NUMBER OF ELEMENTS OF THE LEFT AND RIGHT PROBABILITY VECTORS. ALL TIMES REPORTED ARE IN SECONDS.

N	Hardware PLF		Software PLF
	single core	dual core	
10k	0,0004	0,0002	0,019
20k	0,0007	0,0005	0,038
30k	0,0011	0,0007	0,051
40k	0,0015	0,001	0,071
50k	0,0018	0,0012	0,084
100k	0,0036	0,0025	0,16
200k	0,0072	0,0049	0,31
500k	0,018	0,012	0,78
1M	0,036	0,025	1,5
2M	0,072	0,049	3,1
5M	0,18	0,12	7,8
10M	0,36	0,24	15,5

differences of our solutions with those presented in the related work section and provide a comparison.

A. Zynq UltraScale+ MPSoC Platform

We measured the execution time of the PLF both in software running on the Cortex-A53 processor, and on the hardware accelerators for multiple datasets (ranging from 10k to 10M elements for the left and right probability matrices). The reported execution time for the hardware accelerated function includes all related operations required to compute the PLF, such as initiations of the DMA engines and data transfers.

Table III provides the execution times of the PLF on the ZCU102 platform. It can be observed that execution times increase linearly with respect to the data size. The dual core implementation provides about 1.5x faster execution times than the single core PLF. The reason that the performance does not scale linearly is attributed to the PS-PL I/O capabilities of the device. Each core requires about 7GB/s of bandwidth in order to keep the pipeline full, however the device can provide close to 11GB/s throughput from the four HP ports at 250MHz. Compared to executing the PLF in software on the ARM processor, both hardware implementations are significantly faster, achieving a 43x to 64x acceleration (for the single and dual core implementations respectively).

B. Amazon AWS F1 Instance

In order to evaluate the performance of our implementation of the PLF function we setup a set of experiments. As mentioned above, we employed real-world data for left and right probability vectors ranging from 10k to 10M elements and for each data set we performed 100 runs and calculated the average execution time for the PLF to offset potential interferences with the OpenCL/XRT runtime environments, server load, scheduling etc. In addition, for each data set size, we used a number of different data sets in order to avoid caching effects, so that we have more realistic measurements.

It should be noted that we considered both single and double buffering implementations as mentioned in Section IV.B. For the latter, we experimented with different block sizes (i.e. how the overall data set is split into blocks) ranging from 4k to 128k elements. The software version used as reference comparison is the PLF function implementation in the official RAxML version 8.2 and we employ the same compiler performance flags with that version. The software is also executed on the AWS F1 instance (Intel Xeon E5-2686v4).

Table IV summarizes our results. The execution time measured is the overall execution time observed by the RAxML application when the PLF function is called. This means that it includes all preprocessing steps in order to setup the call of the accelerator, the transfer of data to and from the accelerator card and of course the execution orchestration and actual computation time. As such, the reported times can be considered exactly equivalent with their software counterparts.

Comparing single and double buffering implementations, it can be seen that for small sizes of the input vectors the former provides higher performance. This is because PCIe bandwidth is better utilized through larger transfers and for small input data sizes the data transfer times prevail the computation times. This reverses when input vector sizes become larger and the increased processing time allows to mask the data transfer of the next data set. However, the results demonstrate that there is not a single block size that is able to perform best in all cases. Therefore, in order to achieve the best overall performance in all possible scenarios, we propose that a proper buffer and block selection mechanism is employed in the RAxML application: for small data sizes, single buffering can be used and for larger data sizes double buffering with 16k or 32k element blocks may be employed. It should be noted that all three cases do not require any change in the underlying FPGA hardware. Comparing the execution time of the best accelerator case on the Amazon F1 and the time required to compute the same function in software, it can be seen that the accelerator provides 2.3x to 5.2x better performance. The performance gap widens as the input size increases.

We note from Table IV that the overall execution time of the PLF function seems rather small and therefore it appears to be a bad target for hardware acceleration. However, profiling of actual real-world analysis performed with RAxML reveals that the PLF function is invoked several hundreds of thousands up to several million times per run. Thus, the collective amount of time spent in this function can amount from several minutes up to several hours, and thus the reported speedups may add up to hours of wall clock time, per RAxML run in real-world applications vs. software-only execution.

TABLE IV
EXECUTION TIME OF THE PLF FUNCTION ON THE VIRTEX VU9P FPGA (AWS F1 INSTANCE). SOFTWARE IS EXECUTED ON THE SAME F1 INSTANCE. N REFERS TO THE NUMBER OF ELEMENTS OF THE LEFT AND RIGHT PROBABILITY VECTORS. ALL TIMES REPORTED ARE IN SECONDS.

N	Block Size (Double Buffering)						Single Buffering	Software PLF
	4k	8k	16k	32k	64k	128k		
10k	0,0015	-	-	-	-	-	0,00097	0,0022
20k	0,0022	0,0026	-	-	-	-	0,0016	0,0044
30k	0,0028	0,0031	-	-	-	-	0,0022	0,006
40k	0,0034	0,0036	0,0045	-	-	-	0,0028	0,0082
50k	0,0041	0,0042	0,0051	-	-	-	0,0033	0,0098
100k	0,0074	0,0065	0,0076	0,0095	-	-	0,006	0,019
200k	0,014	0,011	0,011	0,016	0,018	-	0,011	0,037
500k	0,038	0,025	0,023	0,026	0,035	0,047	0,028	0,092
1M	0,098	0,052	0,042	0,042	0,053	0,071	0,055	0,18
2M	0,3	0,11	0,079	0,076	0,085	0,11	0,11	0,36
5M	1,56	0,44	0,21	0,17	0,18	0,22	0,27	0,91
10M	3,25	0,93	0,43	0,35	0,35	0,42	0,55	1,83

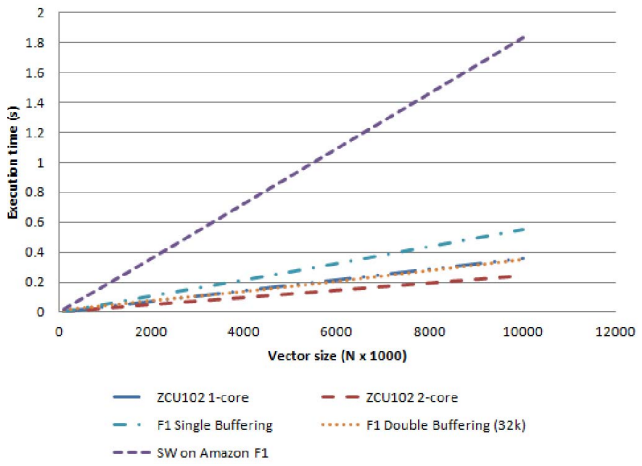

Fig. 4. PLF Execution time Comparison.

C. Performance Results Evaluation and Comparison with Related Works

Figure 4 provides a comparative view of the PLF runtimes for the 100k-10M datasets. The results of the software running on the Amazon F1 instance are compared with the proposed hardware accelerators, namely the accelerators on the F1 instance (using single and double buffering) and the single and dual core implementations on the Zynq UltraScale+.

The accelerator using single buffering on the F1 instance provides 3.3x faster execution times than the software equivalent function, while using double buffering increases the performance up to 5.2x. In our data sets, we measured that the PLF execution time ranges from 50 to 80% of the overall execution time of the RAxML application (while in literature cases that raise this up to 95% are reported [19]). The percentage of the execution time that is spent on the PLF increases with the size (N) of the probability vectors. As such, using our accelerated PLF on the Amazon F1, a complete phylogenetics analysis through the RAxML application is expected to be performed from 1.3 to 3.2 times faster.

On the other hand, the single core accelerator on the ZCU102 board is 5.2x faster than the software PLF and the dual core implementation increases that further up to 7.7x. As mentioned above, the baseline for this performance analysis is the software PLF on the Xeon processor. The reason that the accelerators on the ZCU102 perform better than the ones on the F1 instance is that there is no need for explicit data transfers to/from the accelerator memory, as the PL has direct access to the PS memory. In fact, in terms of theoretical peak processing power, the accelerator within the F1 FPGA is about 2 times faster then the dual core implementation on the ZCU102, as a result of its wider execution unit and significantly lower initiation interval.

It should be noted that the software execution time on the ARM Cortex-A53 processor is not considered. This is because it is about 8.5 times slower vs. the Intel Xeon of the F1 instance, making it an inappropriate target for datasets in which the PLF is not dominant in the overall execution time. Using the dual core implementation and by taking into account that the software part will be executed on the ARM processor, a complete phylogenetics analysis through the RAxML application is expected to be performed from 4.5 to 1.8 times slower than the Intel Xeon processor on the F1. However, in corner cases that the PLF execution covers more than 90% of the overall execution time (such as those reported in [19]), the dual core implementation on the ZCU102 can provide up to 1.8x better performance than a purely software RAxML solution on a high-end Intel processor. In the ZCU102 MPSoC design the bottleneck is the CPU and not the accelerator. This demonstrates the potential of the proposed solution, as future MPSoC implementations will employ higher performance general purpose CPU cores.

Table V provides comparison results based on VEUPS (Vector Entry Updates Per Second), a PLF-specific metric introduced by Berger et al. [19] that represents the number of ancestral probability vector entries that are computed per second. As explained by the authors, VEUPS-based comparisons are only meaningful under the same number of states and rate categories. We therefore adapted the VEUPS

TABLE V
COMPARISON OF CURRENT WORK'S PERFORMANCE WITH PREVIOUS WORK.

Implementation	VEUPS	clock frequency
Alachiotis et al. [15]	17.750	284MHz
Alachiotis et al. [16]	25.250	101MHz
Berger et al. [19]	18.385	167MHz
ZCU102 single core	27.777	250MHz
ZCU102 dual core	41.665	250MHz
F1 AWS instance	32.358	222MHz

performance that Berger *et al.* [19] report, such that it reflects four rate categories instead of one used by the authors, and additionally estimate VEUPS performance for the accelerators presented by Alachiotis *et al.* [15], [16] based on the reported operating clock frequency, pipeline latency, initiation interval, and throughput. The results demonstrate that our solutions provide the highest performance, improving the state-of-the-art by up to 65%.

VI. CONCLUSIONS

In this work we presented two platforms based on reconfigurable hardware that can be used for the acceleration of phylogenetic analyses. We focus on the most computationally intensive function of the widely used software RAxML, i.e., the PLF, and propose different architectures taking into consideration the specifics of each platform.

Our solutions achieved significant speedups compared to pure software implementations on high-end Intel Xeon processors on both cases. The proposed hardware architectures were only limited by the I/O bandwidth, thus indicating that future revisions of the platforms can unlock even higher performance potential. Compared to other competing FPGA implementations, all our proposed solutions provide significantly better performance, achieving up to 65% higher VEUPS vs. the highest performing work reported in literature. In the future, we will try to mitigate the I/O limitations, especially on F1-like systems. Possible improvements include integration of additional functionality on the accelerator in order to reduce data transfers and caching of data on the accelerator memory. Also, for platforms that tightly integrate CPU cores and FPGA fabric, the use of parts that employ higher performing general purpose processors should be explored.

ACKNOWLEDGMENT

This work was funded in part by the "Centre for the study and sustainable exploitation of Marine Biological Recourses (CMBR)", which is implemented under the Action "Reinforcement of the Research and Innovation Infrastructure", funded by the Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014-2020, MIS Code 5002670) and co-financed by Greece and the European Union (European Regional Development Fund).

This work was also partially funded by the EDRA Project, funded from the European Union's Horizon 2020 research and innovation programme "FET Innovation Launchpad" under grant agreement No 851631.

REFERENCES

[1] W.Fitch and E. Margoliash, "Construction of phylogenetic trees," Science, vol. 155, no. 3760, pp. 279–284, 1967.
[2] J. Felsenstein, "Evolutionary trees from DNA sequences: a maximum likelihood approach," J. Mol. Evol., vol. 17, pp. 368–376, 1981.
[3] S. Roch, "A short proof that phylogenetic tree reconstruction by maximum likelihood is hard," Computational Biology and Bioinformatics, IEEE/ACM Transactions on, vol. 3, no. 1, pp. 92–94, 2006.
[4] Stamatakis, A. (2014). RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. Bioinformatics, 30(9), 1312-1313.
[5] D. Zwickl, "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, April 2006.
[6] S. Guindon and O. Gascuel, "A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood." Syst. Biol., vol. 52, no. 5, pp. 696–704, 2003.
[7] F. Ronquist and J. Huelsenbeck, "MrBayes 3: Bayesian phylogenetic inference under mixed models," Bioinformatics, vol. 19, no. 12, pp. 1572–1574, 2003.
[8] N. Lartillot, S. Blanquart, and T. Lepage, "PhyloBayes. v2. 3," 2007.
[9] https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/ug1182−zcu102−eval−bd.pdf
[10] Amazon EC2 F1 instances, https://aws.amazon.com/ec2/instance-types/f1/
[11] Xilinx Zynq UltraScale+ Heterogeneous MPSoC devices, https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html
[12] Yang, Z. (1994). Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: approximate methods. Journal of Molecular evolution, 39(3), 306-314.
[13] Mak, T. S., Lam, K. P. (2004, August). Embedded computation of maximum-likelihood phylogeny inference using platform FPGA. In Proceedings. 2004 IEEE Computational Systems Bioinformatics Conference, 2004. CSB 2004. (pp. 512-514). IEEE.
[14] Mak, T. S., Lam, K. P. (2004, August). FPGA-based computation for maximum likelihood phylogenetic tree evaluation. In International Conference on Field Programmable Logic and Applications (pp. 1076-1079). Springer, Berlin, Heidelberg.
[15] Alachiotis, N., Sotiriades, E., Dollas, A., Stamatakis, A. (2009, May). Exploring FPGAs for accelerating the phylogenetic likelihood function. In 2009 IEEE International Symposium on Parallel Distributed Processing (pp. 1-8). IEEE.
[16] Alachiotis, N., Stamatakis, A., Sotiriades, E., Dollas, A. (2009, September). A reconfigurable architecture for the phylogenetic likelihood function. In 2009 International Conference on Field Programmable Logic and Applications (pp. 674-678). IEEE.
[17] Bakos, J. D., Elenis, P. E., Tang, J. (2007, October). FPGA acceleration of phylogeny reconstruction for whole genome data. In 2007 IEEE 7th International Symposium on BioInformatics and BioEngineering (pp. 888-895). IEEE.
[18] Zierke, S., Bakos, J. D. (2010). FPGA acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods. BMC bioinformatics, 11(1), 184.
[19] Berger, S. A., Alachiotis, N., Stamatakis, A. (2012, May). An optimized reconfigurable system for computing the phylogenetic likelihood function on dna data. In 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum (pp. 352-359). IEEE.
[20] Alachiotis, N., Skrimponis, P., Pissadakis, M., Rangan, S., Pnevmatikatos, D. (2020, February). Near-memory Acceleration for Scalable Phylogenetic Inference. In The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (pp. 324-324).
[21] European Exascale System Interconnect and Storage, https://exanest.eu/
[22] AWS Shell Interface Specification, https://github.com/aws/aws-fpga/blob/master/hdk/docs/AWS_Shell_Interface_Specification.md
[23] James E Smith. Decoupled access/execute computer architectures. ACM Transactions on Computer Systems (TOCS), 2(4):289–308, 1984.
[24] Chen T. and G.E. Suh. Efficient data supply for hardware accelerators with prefetching and access/execute decoupling. 49th Annual IEEE/ACM International Symposium on Microarchitecture, page 46. IEEE Press, 2016.