Exploring FPGAs for Accelerating the Phylogenetic Likelihood Function

N. Alachiotis[1,2], E. Sotiriades[1], A. Dollas[1]
Department of Electronic and Computer Engineering[1]
Technical University of Crete
Chania, Crete, Greece
Email: alachiot@in.tum.de, {esot,dollas}@mhl.tuc.gr

A. Stamatakis[2]
The Exelixis Lab, Department of Computer Science[2]
Technische Universität München
Munich,Germany
Email: stamatak@in.tum.de

Abstract

Driven by novel biological wet lab techniques such as pyrosequencing there has been an unprecedented molecular data explosion over the last 2-3 years. The growth of biological sequence data has significantly out-paced Moore's law. This development also poses new computational and architectural challenges for the field of phylogenetic inference, i.e., the reconstruction of evolutionary histories (trees) for a set of organisms which are represented by respective molecular sequences. Phylogenetic trees are currently increasingly reconstructed from multiple genes or even whole genomes. The recently introduced term "phylogenomics" reflects this development. Hence, there is an urgent need to deploy and develop new techniques and computational solutions to calculate the computationally intensive scoring functions for phylogenetic trees.

In this paper, we propose a dedicated computer architecture to compute the phylogenetic Maximum Likelihood (ML) function. The ML criterion represents one of the most accurate statistical models for phylogenetic inference and accounts for 85% to 95% of total execution time in all state-of-the-art ML-based phylogenetic inference programs. We present the implementation of our architecture on an FPGA (Field Programmable Gate Array) and compare the performance to an efficient C implementation of the ML function on a high-end multi-core architecture with 16 cores.

Our results are two-fold: (i) the initial exploratory implementation of the ML function for trees comprising 4 up to 512 sequences on an FPGA yields speedups of a factor 8.3 on average compared to execution on a single-core and is faster than the OpenMP-based parallel implementation on up to 16 cores in all but one case; and (ii) we are able to show that current FPGAs are capable to efficiently execute floating point intensive computational kernels.

1. Introduction

Significant advances in biological sequencing techniques, such as pyrosequencing [1], in combination with technological advances in the area of computer architectures pose new challenges for parallel computing in Bioinformatics. The field of Bioinformatics currently faces an unprecedented data flood that needs to be analyzed.

In this paper, we assess how the compute- and memory-intensive Phylogenetic Likelihood Function (PLF [2], henceforth we use ML and PLF as equivalent terms) can be mapped onto dedicated hardware and what the expected performance gains of such a specialized PLF architecture are.

The field of phylogenetic inference deals with the reconstruction of the evolutionary history for a set of organisms, in form of an unrooted binary tree, based on a multiple sequence alignment of molecular DNA or protein sequence data from these organisms. An example of a small alignment of DNA sequences for the Human, the Mouse, the Cow, and the Chicken is provided below:

```
Cow       ATGGCATATCCCA-ACAACTAGGATTCCAA
Chicken   ATGGCCAACCACTCCCAACTAGGCTTTC-A
Human     ATGGCACAT---GCGCAAGTAGGTCTAC-A
Mouse     ATGG----CCCATTCCAACTTGGTCTACAA
```

The PLF is one of the most widely used optimality criteria to score and thus chose among distinct evolutionary scenarios (phylogenetic trees). The underlying optimization problem, that consists in conducting a tree search to find the best-scoring ML tree is NP-hard [3]. Phylogenetic trees have many important applications in medical and biological research (see [4]).

While there are many program packages available that implement the PLF, either for standard Maximum Likelihood based optimization (e.g. RAxML [5], GARLI [6]) or to conduct Bayesian phylogenetic inference (MrBayes [7]), the underlying computational problems are identical: all PLF-based phylogenetic inference programs spend the largest part of the overall run time, typically between 85% and 95% in the computation of the PLF [5]. It is thus important to assess and devise architectural solutions for this widely used and challenging Bioinformatics function. The need to introduce HPC expertise into the field of Bioinformatics and phylogenetics in particular is underlined by the significantly higher pace at which biological data accumulates compared to the increase in transistor count, i.e., Moore's law (see

Figure 1 in [8] for a respective plot).

In this paper we exclusively focus on exploitation of fine-grained instruction-level parallelism in the PLF, despite the fact that, both ML and Bayesian phylogenetic inference also provide a source of coarse-grained parallelism. In standard ML analyses embarrassing parallelism is implemented via independent tree searches on bootstrap replicates [9] or through independent parallel searches for the best-scoring ML tree on distinct starting trees [5]. The main reason to focus on fine-grained parallelism is the current trend in systematics towards analyses of phylogenomic alignments. As already mentioned, such phylogenomic alignments consist of a large number of concatenated genes —a current study with collaborators from Biology consists of almost 1,500 genes and required 2.25 million CPU hours on an IBM BlueGene/L supercomputer— and are hence extremely memory- and compute-intensive. Therefore, a large amount of computational resources needs to be allocated for concurrently computing the likelihood on *a single* large tree topology. Generally, fine-grained loop-level parallelism fits well to current trends in Systematics and super-linear speedups (because of increased cache efficiency) can be achieved on large phylogenomic datasets [10], [11]. Finally, parallelization at a fine grained level naturally maps better to a FPGA architecture, while coarse grained parallelism can be exploited at a higher level, e.g., by a cluster of CPU-FPGA nodes.

Our initial PLF architecture only covers a subset of the operations and functions required to conduct a full real-world tree search as implemented in RAxML. However, the subset of computations we consider here, which corresponds to conducting a full tree traversal to score a given fixed tree topology using Felsenstein's pruning algorithm (see [2] and Section 2), adequately represents the characteristics of the required computations. We hence explore the potential speedups that can be obtained by a one to one mapping of the computations required for full tree traversals from an efficient C source code on a high-end multi-core machine to an FPGA on trees comprising 4 up to 512 taxa. Compared to execution on a single core, FPGA speedups for a full tree traversal range from factor 3.07 up to 13.68 (average: 8.3) on the largest input dataset with 512 taxa (taxa is used as a synonym for sequences/organisms). With respect to the fastest execution time achieved by an OpenMP-based parallel SW implementation on 2, 4, 8, and 16 cores the FPGA is still faster in all but one case (16 taxa on 8 cores).

The remainder of this paper is organized as follows: In Section 2 we describe the basic mathematical operations and concepts required to implement the PLF kernel. Thereafter, (Section 3) we discuss related work on exploiting FPGAs for phylogenetic inference under Maximum Likelihood and other models. In Section 4 we propose an architecture for computing the PLF. In the following Section 5 we present experimental results for the FPGA implementation compared to a high-end 16-core Sun x4600 system. We conclude in Section 6 and address areas of future work.

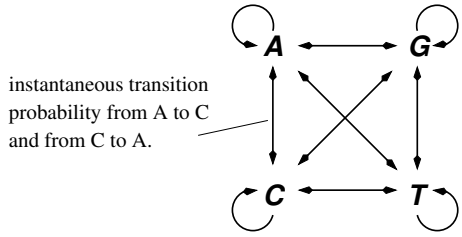

Figure 1. Statistical nucleotide substitution model for DNA data.

2. Computation of the Phylogenetic Likelihood Function

As already mentioned the computational kernel of all current ML based implementations consists of the Phylogenetic Likelihood Function (PLF). To compute the PLF on a given fixed tree topology, i.e., we do not consider the problem of tree search algorithms here, one needs to estimate the branch lengths and the parameters of the statistical model of nucleotide substitution. In the case of DNA data, an appropriate statistical model of nucleotide substitution is provided by a 4x4 matrix usually denoted as Q, that contains the instantaneous transition probabilities for a nucleotide `A` to mutate to a nucleotide `A`, `C`, `G`, or `T`, within time dt (see Figure 1). To compute the nucleotide substitution probabilities given a branch length t (t essentially represents the evolutionary time between two nodes in the tree) one has to compute $P(t) = e^{Qt}$. In typical real world analysis this model is extended by additional ML model parameters to accommodate for rate heterogeneity, i.e., the biological fact that different columns of nucleotides in the alignment evolve at different speeds. The branch lengths are optimized in order to improve the likelihood given the tree and the substitution model.

To compute the likelihood of a fixed *unrooted* tree topology with given branch lengths and model parameters, one initially needs to compute the entries for all internal likelihood vectors which are located at the inner nodes of the tree. They contain the four probabilities $P(A), P(C), P(G), P(T)$ of observing a nucleotide `A`,`C`,`G`, or `T` at an inner node for each column of the input alignment. The probabilities at the tips of the tree for which observed data is available are set to 1.0 for the observed nucleotide character, e.g., for the nucleotide `A`: $P(A) = 1.0$ and $P(C) = P(G) = P(T) = 0.0$ for all remaining characters.Apart from `A`,`C`,`G`,`T` there also exist ambiguous characters like, e.g., `M` that stands for `A` or `C` and is thus represented by $P(A) = 1.0, P(C) = 1.0, P(G) = P(T) = 0.0$ at the tip vector level. Hence, the likelihood vectors at the

tips and inner nodes have the same number of columns as the sequences in the input alignment, e.g., a length of 30 in our small example alignment in Section 1. Every vector entry contains 4 floating-point numbers to store the respective probabilities. However, for the significantly more realistic models, such as the Γ [12] model, that accommodate for rate heterogeneity among columns, 16 floating-point numbers need to be stored per vector entry. This is due to the fact that the Γ function is discretized into, e.g., 4 discrete rates, $r_0, ..., r_3$. Then, for each discrete rate one needs to compute the individual likelihood vector entries, hence 4x4 values for each column of the alignment. The entries of the inner likelihood vectors are computed (filled) bottom-up from the tips towards a virtual root that can be placed into any branch of the tree. Let us consider a small three-node subtree (see Figure 2) with two child nodes (tips in this case) and an inner node at the root of the subtree. The entries of the likelihood vectors of the child nodes are initialized in order to calculate the likelihood vector at the common ancestor. For every child there is one entry for each column of the alignment and for every branch there is one transition probability matrix, $P(t_1)$ and $P(t_2)$. A series of operations (multiplications and additions) are conducted using the entries at position `i` of both likelihood vectors at the child nodes and the two transition probability matrixes $P(t_1)$ and $P(t_2)$ in order to calculate the respective entry at position `i` of the parent likelihood vector.These operations are executed for all columns `i=1...alignmentLength` of the input alignment. The procedure outlined in Figure 2 is known as Felsenstein's pruning algorithm. Under certain standard model restrictions, namely time reversibility of the nucleotide substitution model, i.e., in principle a symmetric nucleotide substitution model, the likelihood will be the same regardless of the placement of the virtual root. This means that the virtual root can be placed into an arbitrary branch of the tree. Once a virtual root has been placed into the tree the pruning algorithm can be applied bottom-up towards this root. An important practical implementation issue is that the likelihood vector entries need to be scaled, in order to avoid numerical underflow because of very small probability values. Once all likelihood vector entries have been computed towards the virtual root, the log likelihood value can then be calculated by essentially summing up over the likelihood vector values to the left and right of the virtual root. Note that, in order to compute the likelihood at the root the prior probabilities $\pi_A, \pi_C, \pi_G, \pi_T$ (nucleotide frequencies) of observing `A,C,G,T` at the virtual root are required. These frequencies can either all be set to 0.25 (as in our architecture), be obtained empirically by counting the occurrences of `A,C,G,T` in the input alignment, or via an ML estimate.

Due to the high complexity of implementing the PLF function on an FPGA our exploratory architecture currently only computes a subset of the PLF. We have made the following simplifications: we only conduct full tree traversals from the leaves (tips) towards the virtual root of the tree to compute the likelihood score on a fixed topology with fixed branch lengths and fixed model parameters. In addition, for the time being we assume that all branch lengths are equal, i.e., the matrix $P(t) = e^{Qt}$, where t is the branch length, is constant across the tree. We have currently not implemented a scaling procedure for very small likelihood vector entries since scaling only affects very large trees with hundreds to thousands of sequences. In addition, we have not implemented a model of rate heterogeneity. However, the Γ model of rate heterogeneity can be accommodated by conducting as many tree traversals as there are discrete Γ rates (typically 4 or 8) with appropriate rate-dependent P matrices. Nonetheless, the architecture is based on the computations required for the most complex general time reversible (GTR [13]) model of nucleotide substitution. Finally, the likelihood score can currently only be computed on fully balanced binary trees of size 4, 8, 16, 32, etc.

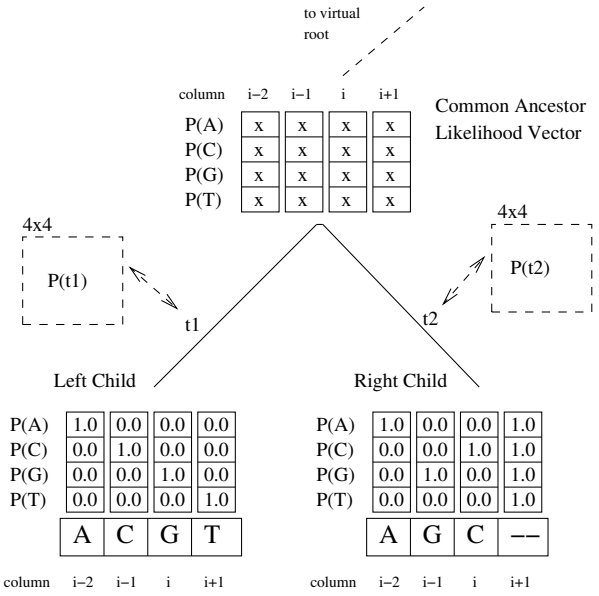

Figure 2. Schematic representation of Felsenstein's pruning algorithm.

While the above restrictions only cover a small subset of the computations required by RAxML or MrBayes for real-world biological analyses, they allow to accurately explore the performance of the computational kernel of the PLF on an FPGA. To ensure a fair comparison, our C reference program (a significantly strapped-down version of RAxML, available as open-source code at http://wwwkramer.in.tum.de/exelixis/software.html) was implemented to conduct *exactly* the same computations as the FPGA, i.e., we removed code for branch length optimization, per-branch substitution matrix computation, model parameter optimization, likeli-

hood vector entry scaling, and rate heterogeneity. As already mentioned, the Γ model of rate heterogeneity can be implemented using the architecture we propose by repeatedly traversing the tree for the individual discrete Γ rates. This approach will provide savings of factor 4 (if a Γ model with 4 discrete rates is used) with respect to memory requirements as well as transistor count. The computation of full tree traversals, which are not necessary when techniques such as Lazy Subtree Rearrangements (LSRs [5]) are deployed that only require partial tree traversals, i.e., the re-computation of a subset of the likelihood vectors, is however still required during certain phases of all ML programs. When ML model parameters like the substitution rates of the GTR model or the α shape parameter of the Γ model of rate heterogeneity are changed during optimization, full tree traversals are required because the changes in these model parameters affect the entire tree and hence the resulting likelihood score. In programs such as MrBayes or RAxML model parameter optimization phases (as opposed to tree search phases) that require full tree traversals can consume up to 20-30% of overall execution time. In addition, an FPGA implementation that conducts full tree traversals could be useful for the optimization of the 20x20 GTR model substitution matrix for protein data. While so far the ML estimate of a GTR model for protein alignments was considered to be over-parameterizing the model, this has changed with new large phylogenomic datasets. The computational requirements for estimating those 190 GTR rate parameters are enormous, e.g., almost 16 hours are required to optimize GTR rate parameters for a fixed tree topology on a 16-core AMD Barcelona system using RAxML on a dataset with 191 sequences and 17,472 protein data columns. Another potential application scenario for full tree traversals on fixed tree topologies is that of ML-based inference of internal states, i.e., the optimal assignment of internal/ancestral sequences to the inner nodes in order to maximize the likelihood score.

3. Related Work

There is comparatively little related work that deals with the implementation of phylogenetic inference methods on FPGAs. Bakos *et al* [14], [15] focus on evolutionary reconstruction based on gene order input data, i.e., the order or permutation by which corresponding genes are placed within a genome. Corresponding genes with the same function can be placed within different parts of the genome in distinct organisms. Thus, they focus on mapping GRAPPA [16], one of the standard implementations for gene order based phylogenetic inference onto FPGAs. The major difference to ML-based phylogenetic inference in a computational context is that the scoring function used in gene order analyses is mostly discrete, i.e., only few floating point operations need to be performed.

Davis *et al* [17] present an implementation of the UPGMA (Unweighted Pair Group Method with Arithmatic Mean) which is considered to be the most simple tree reconstruction algorithm currently available. It is based on a clustering algorithm that uses a pairwise distance matrix. Due to several methodological concerns the UPGMA method is practically not used any more for real-world phylogenetic studies. The most popular models for tree reconstruction in current biological practice are Maximum Parsimony (MP [18]) and Maximum Likelihood. We are not aware of any work on mapping the MP function to an FPGA. This is surprising, because the MP scoring function is discrete as well as significantly less memory intensive and thus more straightforward to efficiently implement on an FPGA. As for ML, the MP phylogeny problem is NP-hard [19], but there also exist efficient heuristics like those implemented in TNT [20].

Finally, Mak and Lam have published a series of articles that deal with mapping the ML function to FPGAs [21], [22]. However, their architectures only implement the Jukes-Cantor (JC69 [23]) model of nucleotide substitution, which represents the most simple statistical model of DNA substitution and is rarely used in present-day biological analyses [24]. In addition all performance tests have been conducted on 4-taxon trees (trees with 4 leaves) only, and execution times are not explicitly provided for larger trees, such that we do not know if and how their implementation scales to larger tree sizes as well as to longer alignments.

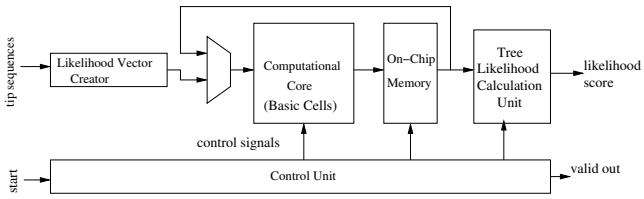

Figure 3. Block diagram of the architecture for the Phylogenetic Likelihood Function

4. An Architecture for the Phylogenetic Likelihood Function

The architecture we propose takes as input the aligned DNA sequences, and then computes and stores the entries of the inner likelihood arrays in the internal (embedded) memory of the FPGA. The values of the tip and inner likelihood arrays are then successively re-used to compute all internal likelihood vectors bottom-up towards the virtual root of the tree. As described in Section 2 we need to devise components that (i) read the sequence data, (ii) compute the entries of the inner likelihood vectors and (iii) compute the likelihood score at the virtual root of the tree. In Figure 4 we depict the block diagram of the proposed architecture.

The unit designated as `Likelihood Vector Creator` reads the sequence data and generates a

likelihood vector for every nucleotide by converting the 4-bit word that encodes a DNA character to four double precision floating point numbers. The conversion is performed via a look-up table which complies with the standard definition for ambiguous DNA character encoding (see http://www.hgu.mrc.ac.uk/Softdata/Misc/ambcode.html and also Section 2). The output of the `Likelihood Vector Creator` is then forewarded to the `Computational Core`. The `Computational Core` consists of seven `Basic Cell (BC)` units (see Figure 4) that can operate in two distinct modes (see below) and are arranged in a binary tree as shown in Figure 5 which also reflects the datapath of the system. Each `BC` unit consists of nine multipliers and six adders that are organized as outlined in Figure 4 and conducts the main bulk of the likelihood computations. The multipliers and adders in a `BC` are pipelined with a throughput of one result per cycle and a latency of fifteen and fourteen cycles respectively. Due to the limited amount of DSP48E slices on the FPGA, that are available to implement floating point operations, several multiplexer units are deployed to optimally exploit the computational resources available. Each `BC` has an input steering signal of 1 bit that is used to select among two basic operations. When a `Basic Cell` works in mode `0`, it calculates the inner likelihood vector from two child nodes that can either be tip vectors or inner vectors respectively. Mode `1` is invoked, when the basic cell needs to calculate the likelihood vector at the virtual root of the tree, which requires distinct arithmetic operations.

All blocks of embedded memory are organized in eight parts of size 9,216x256 bits. Each of the eight parts is used to store inner likelihood arrays which may need to be read again—depending on the input tree size—by the `Basic Cells` after several hundreds of cycles. As the computation proceeds bottom-up towards the virtual root those eight parts of embedded RAM are read or written several times depending on the tree depth. Data stored in these memory locations can be overwritten several times but we maintain a distance of 174 positions, which corresponds to the latency of the datapath, between the read index and the write index for each of the eight memory parts. When the likelihood vector at the virtual root has been computed by the `Basic Cells` for a set of alignment columns, the `Tree Likelihood Calculation` unit then computes and combines the per-site (per-column) likelihood scores and returns the overall likelihood score of the tree. The basic components of this unit are illustrated in Figure 6. Finally, the `Control Unit` consists of a hierarchy of five simple FSMs (Finite State Machines) which concurrently control the datapath. We use a master FSM to synchronize the 4 remaining worker FSMs. The worker FSMs are used to generate the required control signals.

Our architecture operates in three phases: During the *first phase*, the system reads the nucleotide sequences at the tips and initially translates them to sequences of likelihood vectors as described in Section 2. Then, it calculates the likelihood vectors of their common ancestors bottom-up, up to two levels above the tips and stores the results in embedded RAM. This *first phase* (computation of inner likelihood vector for 8-taxon trees) is successively executed for the entire input tree in groups of 8 taxa, e.g., 4 times for a 32-taxon tree. The first phase is completed, when the entire input tree and data have been read and the likelihood vectors of the ancestors of the respective 8-taxon trees have been computed (e.g., 4 likelihood vectors in the 32-taxon case).

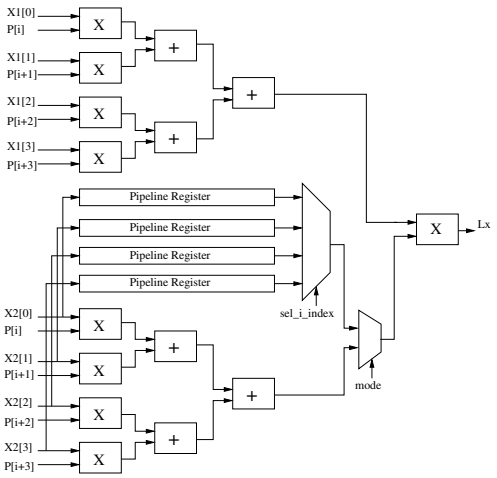

Figure 4. Arrangement of multipliers and adders within a Basic Cell unit. Vectors X1[] and X2[] represent the likelihood vectors entries of the right and left child, and P[] a single row of the transition probability matrix $P_{i,j}(t)$. The mode multiplexer is used to select among functions for computations at the virtual root and and at inner nodes below the virtual root. The output Lx corresponds to one of the 4 probability values P(A), P(C), P(G), P(T) that are written to to the likelihood vector above the input vectors.

The *second phase* is very similar to the initial phase. However, instead of using the DNA sequence information at the tips of the tree as input, it uses the likelihood vectors of the ancestors (previously computed internal nodes of the tree) that have been stored in embedded RAM during the first phase. The results of the computations of the second phase (if the virtual root of the tree has not already been reached) are then stored again in embedded RAM. *Phase 2* is repeated for a maximum of 8 internal nodes at a time (if the number of taxa is a power of eight) until all the internal likelihood vectors at the input level have been used and the inner likelihood vectors two levels above the input level have been computed. At the end of *phase 2* the likelihood vector at the virtual root has been calculated.

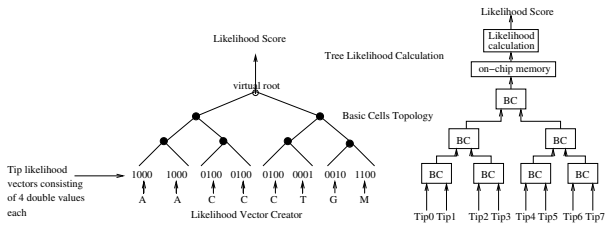

Figure 5. Example for the computation of the likelihood score on a single alignment column for an 8-taxon tree using the Likelihood Vector Creator, Basic Cell, and Tree Likelihood Calculation Units

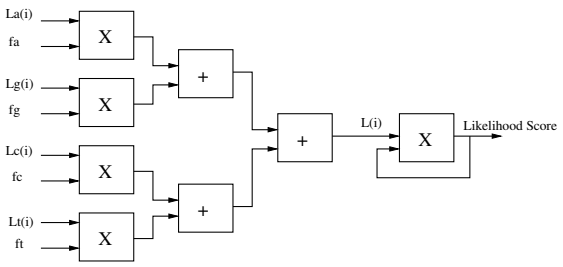

Figure 6. Tree Likelihood Calculation Unit: Likelihood score computation $L(i)$ for a single alignment column at position i and likelihood accumulation over several alignment columns $i = 1...n$. This unit takes as input the inner likelihood vector entries $La(i), Lc(i), Lg(i), Lt(i)$, (denoted as Lx here since they are computed with the Basic Cell in Mode 1) at position i of the virtual root and the base frequencies (prior probabilities) fa, fc, fg, ft (also denoted as $\pi_A, ..., \pi_T$) and multiplies the result $L(i)$ with the product of the previous results $L(0) \cdot L(1) \cdot ... \cdot L(i-1)$ such that $L = L(0) \cdot L(1) \cdot ... \cdot L(i)$.

In the *third* and final *phase* our system loads the inner likelihood vector at the virtual root from embedded RAM and calculates the per-column likelihood score $L(i)$ for every column of the input alignment. A multiplier is used to compute the overall product $L = L(0) \cdot L(1) \cdot ... \cdot L(n-1)$ over all n per-column likelihood scores. Note that the `Tree Likelihood Calculation` unit actually computes the likelihood score and not the more commonly used log likelihood score. The implementation of a `Tree Likelihood Calculation` unit that returns the log likelihood score is subject of future work. If the input data has 8 taxa or less, *phase 2* will be omitted since the internal likelihood vector two levels above the input sequences already *is* the virtual root of the tree. For a 64-taxon input tree, *phase 1* will be executed 8 times and will produce 8 internal likelihood vectors. Then, *phase 2* will be executed only once for the resulting 8 likelihood vectors and yield the likelihood vector at the virtual root. Analogously, for a 512-taxon tree, *phase 1* will be executed 64 times, *phase 2* will be executed 8 times for the first set of internal likelihood vectors two levels above the tips, and 1 more time for the calculation of the likelihood vector at the virtual root.

5. Results

5.1. Experimental Setup

To conduct performance analyses we generated 8 simulated DNA datasets containing 4, 8, 16, 32, 64, 128, 256, and 512 taxa (sequences/organisms) respectively. Every simulated alignment contains 1,000 distinct alignment columns, i.e., the alignment can not be further compressed by merging identical column patterns. In addition, we generated respective fully balanced binary input trees in standard NEWICK format. The datasets are available together with the software implementation at http://wwwkramer.in.tum.de/exelixis/software.html.

As already mentioned the software implementation represents a significantly strapped down version of RAxML that executes exactly the same mathematical operations as our computational kernel on the FPGA to ensure a fair comparison. The sequential as well as OpenMP-based version of the code was compiled using the Intel `icc` compiler (v 10.1, optimization option `-O3`) which generates faster code than `gcc` for RAxML. The loop-level parallelization of RAxML with OpenMP was re-implemented analogously to the description in [11]. In order to obtain accurate software timing results for the very fast tree traversal operation that takes less than 0.015 seconds on all datasets, we measured the time required by a loop that executes 20,000 full tree traversals. As software test platform we used a high-end 8-way dual-core SUN x4600 system equipped with 8 dual-core AMD Opteron processors running at 2.6 GHz with 64 GB of main memory. On long enough input alignments and hence long enough for-loops we have measured super-linear speedups for both Pthreads- and OpenMP-based parallelizations of the standard RAxML implementation running on all 16 cores [11]. The OpenMP-based strapped-down version of RAxML was executed using 2, 4, 8, and 16 cores.

As HW platform for our FPGA implementation we used the Xilinx Virtex 5 SX240T. This FPGA provides a large number of 1,056 DSP48E slices. The DSP48E slices are used to implement the double precision floating point multipliers and adders that form part of the `Basic Cells`. The specific FPGA offers the largest number of embedded memory blocks currently available in commercial FPGAs and is hence well-suited for our purposes.

To validate the correctness of the results (likelihood scores) returned by the FPGA implementation, we deployed an accurate cycle-by-cycle, post place and route simulation for datasets with 8, 64 and 512 taxa and 256 alignment columns each. The FPGA yields *exactly* the same results as

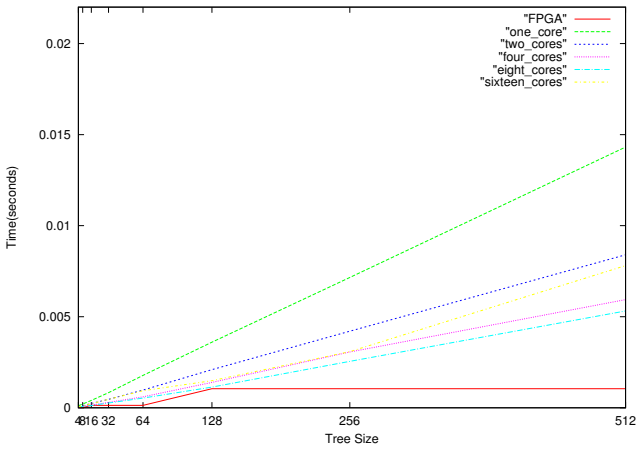

Figure 7. FPGA and multi-core execution times for full tree traversals using Felsenstein's pruning algorithm for trees with 4, 8, 16, 32, 64, 128, 256, and 512 taxa

the software implementation. In order to measure execution times for the FPGA implementation we used the static timing report of the Xilinx tools (ADVANCED 1.53 speed file). The reported clock speed was 284.152 MHz. According to this clock speed and the number of cycles that the system requires to perform likelihood calculations, we determined projected execution times on the FPGA (see Figure 7).

5.2. Experimental Results

The results outlined in Figure 7 depict the execution times of one full tree traversal on the FPGA and of the software implementation on 1, 2, 4, 8, and 16 cores on the Sun x4600 multi-core system. The slowdown that can be observed for the OpenMP version with 16 cores is clearly due to the relatively short alignment length and hence an unfavorable communication to computation ratio. FPGA performance improves slightly with increasing dataset size. As depicted in Figure 7 the execution times on the FPGA for 4-, and 8-taxon trees, for 16-, 32-, and 64-taxon trees as well as for 128-, 256-, and 512-taxon trees are identical. This behavior is related to the degree of utilization of the system. If the number of taxa in the input tree is a power of 8, the `Basic Cells` will be fully utilized at all times during the entire computation until the virtual root is reached. If the number of taxa is, e.g., 4, the same number of cycles as for an 8-taxon tree will be required to reach the virtual root, but 50% of the `Basic Cells` will not be used. Thus, the execution times will remain constant for $8^n < t \leq 8^{n+1}$, where t is the number of taxa in the input tree. Thus, the computational resource utilization in our design is optimal when the input comprises 8, 64, or 512 taxa.

In Table 1 we indicate the speedup values that the FPGA achieved compared to the multi-core system on 1, 2, 4, 8, and 16 cores/threads for all tree sizes. The respective worst speedup obtained by the FPGA is shown in bold letters. In all but one case (16 taxa, 8 cores) the FPGA is faster than a high-end multi-core system executing a highly optimized software implementation of the PLF that has been compiled with the currently most efficient commercial compiler for this type of application. While the worst speedup achieved is 0.96, i.e., the slowdown is negligible, this only occurs in the most unfavorable case—in terms of input tree size—for the FPGA implementation, where a large number of `Basic Cells` remains under-utilized.

# Cores	4 taxa	8 taxa	16 taxa	32 taxa	64 taxa	128 taxa	256 taxa	512 taxa
1	6.81	12.76	3.07	6.30	13.55	3.44	6.83	13.68
2	3.79	7.35	1.79	3.53	7.39	2.00	4.02	8.03
4	2.32	**4.74**	1.15	2.26	4.60	1.33	2.93	5.67
8	**2.09**	4.85	**0.96**	**2.01**	**4.05**	**1.08**	**2.44**	**5.08**
16	3.65	6.82	1.68	3.55	7.14	1.42	2.95	7.46

Table 1. FPGA speedups on 4- up to 512-taxon trees compared to software execution times on 1 up to 16 cores. The worst speedup achieved by the FPGA on the respective datasets is indicated by bold letters

6. Conclusion and Future Work

We have presented an exploratory FPGA design that can conduct full tree traversals using Felsenstein's pruning algorithm on fully balanced binary trees under the realistic GTR model of nucleotide substitution. Our FPGA implementation achieves speedups of up to a factor of 13.68 compared to an efficient software implementation running on a single core and remains competitive when compared to an OpenMP-based parallel implementation running on up to 16 cores on a high-end multi-core system. The speedups achieved are typical for floating point intensive computational kernels on FPGAs. An implementation using single precision floating point arithmetics or specifically adapted floating and/or fixed point number formats (a technique also known as word-length optimization, see, e.g., [25]) might yield substantial additional speedups.

Future work will cover the implementation of the Γ model of rate heterogeneity as well as the CAT approximation of rate heterogeneity [26]. In addition, we will implement additional control units to handle unbalanced trees and enhance the design to explicitly calculate the $P(t)$ transition matrix at each branch individually. In order to devise a more realistic design we also intend to integrate methods for likelihood vector entry scaling and implement models of amino acid substitution. Finally, in order to provide an architecture that will be useful to Biologists we intend to devise an FPGA-based optimization procedure of the GTR substitution parameters for large protein alignments and to develop an FPGA-based heuristic algorithm for ML-based inference of ancestral sequences.

Acknowledgments

Part of this work was funded under the auspices of the Emmy-Noether program by the German Science Foundation (DFG). We would also like to thank Christos Koukouzas for his technical support.

References

[1] M. Margulies, M. Egholm, W.E. Altman, S. Attiya, J.S. Bader, L.A. Bemben, J. Berka, M.S. Braverman, Y.J. Chen, Z. Chen, et al. *Genome sequencing in microfabricated high-density picolitre reactors.* Nature, 437:376–380, 2005.

[2] J. Felsenstein. *Evolutionary trees from DNA sequences: a maximum likelihood approach.* Journal of Molecular Evolution, 17: 368–376, 1981.

[3] B. Chor and T. Tuller. *Maximum likelihood of evolutionary trees: hardness and approximation.* Bioinformatics, 21(1):97–106, 2005.

[4] D.A. Bader , B.M.E. Moret, L. Vawter *Industrial applications of high-performance computing for phylogeny reconstruction.* Proc. SPIE Commercial Applications for High-Performance Computing, 4528, 159–168, 2001.

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

[6] D. Zwickl. *Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion.* PhD thesis, University of Texas at Austin, April 2006.

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

[8] N. Goldman and Z. Yang. *Introduction. statistical and computational challenges in molecular phylogenetics and evolution.* Philosophical Transactions of The Royal Society B, 363(1512): 3889–3892, 2008.

[9] J. Felsenstein. *Confidence Limits on Phylogenies: An Approach Using the Bootstrap.* Evolution, 39(4):783–791, 1985.

[10] M. Ott, J. Zola, S. Aluru, and A. Stamatakis. *Large-scale Maximum Likelihood-based Phylogenetic Analysis on the IBM BlueGene/L.* On-Line Proceedings of IEEE/ACM Supercomputing Conference 2007, 2007.

[11] A. Stamatakis and M. Ott *Exploiting Fine-Grained Parallelism in the Phylogenetic Likelihood Function with MPI, Pthreads, and OpenMP: A Performance Study.* In Madhu Chetty, Alioune Ngom, and Shandar Ahmad, editors, PRIB, volume 5265 of Lecture Notes in Computer Science, pages 424–435. Springer, 2008.

[12] Z. Yang. *Maximum likelihood phylogenetic estimation from dna sequences with variable rates over sites.* Journal of Molecular Evolution, 39:306–314, 1994.

[13] S. Tavare. *Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences.* Some Mathematical Questions in Biology: DNA Sequence Analysis, 17, 1986.

[14] J.D. Bakos. *FPGA Acceleration of Gene Rearrangement Analysis.* In 15th Annual Symposium on Field- Programmable Custom Computing Machines (FCCM 2007), pages 85–94, 2007.

[15] J.D. Bakos, P.E. Elenis, and J. Tang. *FPGA Acceleration of Phylogeny Reconstruction for Whole Genome Data.* In Proc. of 7th IEEE International Conference on Bioinformatics and Bioengineering (BIBE 2007), pages 888–895, 2007.

[16] B.M.E. Moret, J. Tang, L.S. Wang, and T. Warnow. *Steps toward accurate reconstructions of phylogenies from gene-order data.* J. of Comp. and Sys. Sci., 65(3):508–525, 2002.

[17] J.P. Davis, S. Akella, and P.H. Waddell. *Accelerating phylogenetics computing on the desktop: experiments with executing UPGMA in programmable logic.* In Engineering in Medicine and Biology Society, 2004. IEMBS04. 26th Annual International Conference of the IEEE, volume 2, pages 2864–2868 2004.

[18] A.W.F. Edwards and L.L. Cavalli-Sforza. *The reconstruction of evolution.* Heredity, 18(533):104–105, 1963.

[19] L.R. Foulds and R.L. Graham. *The Steiner problem in phylogeny is NP-complete.* Advances in Applied Mathematics, volume 3, pages 43–49, 1982.

[20] P.A. Goloboff, J.S. Farris, and K. Nixon. *TNT: Tree Analysis Using New Technology. Program and documentation, version, 1,* 2000.

[21] T.S.T. Mak and K.P. Lam. *Embedded computation of maximum-likelihood phylogeny inference using platform FPGA.* In Proceedings of IEEE Computational Systems Bioinformatics Conference (CSB 04), pages 512–514, 2004.

[22] T.S.T. Mak and K.P. Lam. *FPGA-Based Computation for Maximum Likelihood Phylogenetic Tree Evaluation.* In Proceedings of IEEE Computational Systems Bioinformatics Conference (CSB 04),volume 3203 of Lecture Notes in Computer Science, pages 1076–1079, 2004.

[23] T.H. Jukes and C.R. Cantor. *Evolution of protein molecules.* Mammalian Protein Metabolism, 3:21–132, 1969.

[24] J. Ripplinger and Jack Sullivan. *Does Choice in Model Selection Affect Maximum Likelihood Analysis?* Systematic Biology, 57(1):76–85, 2008.

[25] G.A. Constantinides. *Perturbation analysis for word-length optimization.* In 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2003), pages 81–90, 2003.

[26] A. Stamatakis. *Phylogenetic models of rate heterogeneity: A high performance computing perspective.* Proceedings of 20th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS2006), 2006.