A GENERIC AND VERSATILE ARCHITECTURE FOR INFERENCE OF EVOLUTIONARY TREES UNDER MAXIMUM LIKELIHOOD

Nikolaos Alachiotis, Alexandros Stamatakis

The Exelixis Lab, Scientific Computing Group
Heidelberg Institute for Theoretical Studies
email: {Nikolaos.Alachiotis,Alexandros.Stamatakis}@h-its.org

ABSTRACT

Likelihood-based reconstruction of phylogenetic (evolutionary) trees from molecular sequence data exhibits extreme resource requirements because of the high computational cost of the phylogenetic likelihood function. We propose a dedicated computer architecture for the inference of phylogenies under the maximum likelihood criterion. Our design is sufficiently generic to support any possible input data type, that is, DNA, RNA secondary structure, or protein data. Furthermore, the architecture is able to calculate log-likelihood scores and perform numerical scaling to maintain numerical stability on large datasets. It can also optimize the branch lengths of tree topologies and calculate transition probability matrices. We used FPGA technology to verify the correctness of our architecture.

1. INTRODUCTION

Phylogenetics is a field in Bioinformatics that deals with the reconstruction of evolutionary trees from molecular data. Tree inference programs are characterized by long execution times due to computationally intensive kernels that are deployed. In addition, significant advances in wet-lab sequencing techniques are generating an unprecedented amount of molecular data. Hence, phylogenetic inference programs need to be scaled to handle those new, large, and challenging datasets.

The Phylogenetic Likelihood Function (PLF) [1] is the computational kernel that all likelihood-based (Maximum Likelihood or Bayesian) phylogenetic inference programs need to execute. Applications such as RAxML [2], MrBayes [3], or GARLI [4] spend between 85% and 98% of total execution time in the PLF. Therefore, efforts have been undertaken to orchestrate the PLF on a large variety of parallel architectures, ranging from Field-Programmable Gate Arrays (FPGA) to the IBM BlueGene/L supercomputer [5].

PART OF THIS WORK IS FUNDED UNDER THE AUSPICES OF THE EMMY-NOETHER PROGRAM BY THE GERMAN SCIENCE FOUNDATION (DFG)

Here, we present a dedicated computer architecture for a PLF co-processor, that is sufficiently generic to compute likelihood scores on all possible biological data types, that is, morphological (binary), DNA, RNA secondary structure, and protein data. RAxML [2], a widely-used program for phylogenetic inference under Maximum Likelihood (available as open source code at `http://wwwkramer.in.tum.de/exelixis/software.html`), was used as reference implementation for designing the PLF co-processor and to verify the correctness of our architecture.

A significant improvement over previous designs [6, 7, 8, 9, 10] is, that we accommodate rate heterogeneity. Rate heterogeneity models accommodate the biological fact that, different alignments columns/sites in the input alignment evolve at different speeds. The branch length optimization process in RAxML is carried out using the Newton-Raphson procedure (see [11] for a summary). The computationally challenging part of branch length optimization consists of calculating the first and second derivative of the likelihood function. Branch length optimization accounts for approximately 30% of the total execution time of RAxML. To this end, a dedicated architectural component has been designed and integrated into the co-processor for computing the derivatives of the PLF. Moreover, the architecture is also able to calculate transition probability matrices P (see [5] for details) for a given branch length b. Finally, a resource-efficient LAU (Logarithm Approximation Unit [12]) is used to calculate log-likelihood scores and an optimized numerical scaling unit has been integrated to avoid numerical underflow in the PLF on large trees with many organisms.

The architecture we present here is the third generation architecture for the PLF on reconfigurable logic. The 1st [9] and 2nd [10] generation were only able to execute a small fraction of the PLF routines required for a phylogenetic analysis with RAxML. This 3rd generation design, now provides all the functionality that can be offloaded to a co-processor by a real-world Maximum Likelihood program.

The remainder of this paper is organized as follows: Section 2 describes the basic mathematical operations and concepts that are required to implement the PLF. In Section 3

we discuss related work on FPGA implementations for phylogenetic inference. In the following Section 4, we present each of the individual processing units of the co-processor as well as the overall architecture. Section 5 provides implementation and verification results. We conclude in Section 6 and address directions of future work.

2. PHYLOGENETIC LIKELIHOOD FUNCTION

In the following, we briefly outline the basic mathematical operations of the PLF. Figure 1 depicts the operations for calculating the ancestral probability vector in a tree for DNA data from the two respective child nodes.

In order to compute the maximum log likelihood value on a given, fixed, tree topology one needs to optimize the branch lengths and the parameters of the statistical nucleotide, or, for instance, protein substitution model. For DNA data, a model of nucleotide substitution is provided by a 4×4 matrix (for protein data by a 20×20 matrix) that is usually denoted as Q matrix. The Q matrix contains the instantaneous transition probabilities (for relative evolutionary time dt) of a nucleotide `A` to change into a nucleotide `A`, `C`, `G`, or `T` etc. To compute 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 analyses this model is extended by additional model parameters to accommodate, for instance, rate heterogeneity among sites. The branch lengths t_i and t_j in Figure 1) and all branch lengths in the tree for that matter, need to be optimized to obtain the maximum log likelihood value for a given tree and given substitution model parameters.

To compute the log likelihood of a fixed *unrooted* tree topology with given branch lengths and model parameters, one initially needs to compute the entries for all ancestral probability vectors which are located at the inner nodes of the tree. Given, a parent node k and two child nodes i and j, their probability vectors $\vec{L}^{(i)}$ and $\vec{L}^{(j)}$, the respective branch lengths leading to the children t_i and t_j, and the transition probability matrices $P(t\_i)$, $P(t\_j)$, the probability of observing an A at position c of the ancestral (parent) vector $\vec{L}_A^{(k)}(c)$ is computed as follows:

$$\vec{L}_A^{(k)}(c) = \Big(\sum_{S=A}^{T} P_{AS}(t\_i)\vec{L}_S^{(i)}(c) \Big) \Big(\sum_{S=A}^{T} P_{AS}(t\_j)\vec{L}_S^{(j)}(c) \Big)$$

An abstract representation of the above equation is provided in Figure 1. An ancestral probability vector entry at a position c contains the four probabilities $P(A)$, $P(C)$, $P(G)$, $P(T)$ of observing a nucleotide `A`,`C`,`G`, or `T` at an ancestral (internal) node for a column c 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 (e.g., for a nucleotide `A` we set $P(A) := 1.0$ and $P(C) := 0.0$, $P(G) := 0.0$, $P(T) := 0.0$). The probability vectors at the tips and ancestral nodes have the same number of columns/sites as the sequences in the input alignment.

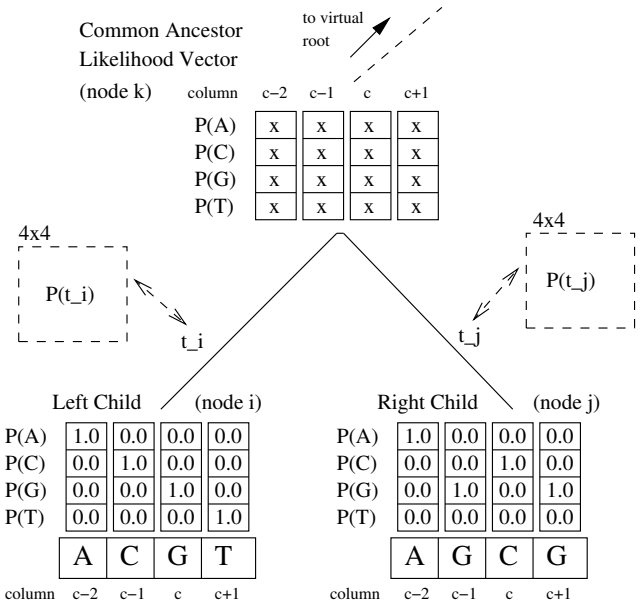

Fig. 1. The Felsenstein pruning algorithm.

For DNA data, every vector entry c contains 4 floating-point numbers to store the respective probabilities. However, for the commonly used, more realistic models that incorporate the Γ [13] model of rate heterogeneity, 16 or even 32 floating-point numbers need to be stored per vector entry c. To compute the integral of the likelihood over the Γ function at each site, it is discretized into, for instance, 4 discrete rates: $r_0, ..., r_3$. For each discrete rate one then needs to compute the corresponding ancestral probability vector entries, that is, 4×4 (4 discrete Γ rates) or 8×4 (8 discrete rates) values for each ancestral probability column. The entries of the ancestral probability vectors are computed (filled) bottom-up from the tips towards a virtual root (see below). This procedure is known as the Felsenstein pruning algorithm [1].

Under certain standard model restrictions, namely time reversibility of the substitution model, that is, in essence a symmetric Q matrix, the log likelihood score will be the same regardless of the placement of the virtual root. This means that the virtual root can be placed into an arbitrary position of an arbitrary branch of the tree to then apply the Felsenstein pruning algorithm.

An important practical implementation issue is, that the ancestral probability vector entries need to be scaled in order to avoid numerical underflow because of very small probability values. Once all probability vector entries in the tree have been computed, the log likelihood value can be calculated by essentially summing up over the ancestral probabil-

ity vector values to the left and right of the virtual root.

3. RELATED WORK

While there exists numerous tools and several methods for phylogenetic inference, only a few have been mapped to hardware.

Mak and Lam [6, 7] mapped a reduced floating point precision PLF of the Jukes-Cantor (JC69 [14]) model of DNA substitution to FPGAs. The Jukes-Cantor model is the most simple statistical model of DNA substitution and is rarely used in present-day biological analyses [15]. All performance tests reported in [6] and [7] have been conducted on trees with only 4 leaves (4 input sequences) and scalability issues are not being addressed.

Davis et al. [16] presented an implementation of the UPGMA method (Unweighted Pair Group Method with Arithmetic Mean) which is a very simple tree reconstruction algorithm and is practically not used for real-world analyses any more.

In [17, 18], Bakos *et al.* focused on tree reconstruction using gene order input data, that is, the order of corresponding genes in the genomes of different organisms is used to reconstruct trees. Bakos *et al.* mapped GRAPPA [19], an open-source implementation for gene order based phylogenetic inference onto FPGAs. The main difference to ML-based phylogenetic inference is that, the kernel function used in gene order analyses is discrete. In other words, only few floating point operations need to be performed to reconstruct a phylogeny.

Kasap and Benkrid [20] presented a FPGA design of the Maximum Parsimony method for phylogenetic inference and assessed performance on a FPGA supercomputer. The implementation is limited to trees with a maximum of 12 organisms, which are very small by todays standards; the largest published parsimony-based tree has 73,060 taxa [21]. They report speedups between a factor of 5 and up to a factor of 32,414 for utilizing 1, 2, 4, and 8 nodes (each node is equipped with a Xilinx Virtex4 FX100 FPGA) on the Maxwell system compared to a 2.2GHz Intel Centrino Duo processor. However, the speedups reported are only relative speedups with respect to the parsimony implementation in PAUP* [22] and not with respect to the fastest-known implementation of parsimony in the TNT program package used in [21].

Recently, Zierke and Bakos [8] presented a FPGA accelerator for the PLF as deployed for Bayesian MCMC-based (Markov Chain Monte Carlo) inference methods. The authors mapped the PLF for DNA data as implemented in MrBayes [3] to their co-processor. Numerical scaling is provided as well as the calculation of log likelihood scores. The speed up estimates (based on the largest available Virtex 6 SX FPGA) reported in the paper vary between 2.5X and 8.7X compared to a single state-of-the-art Intel Xeon 5500-series server processor. Note that, Bayesian inference programs do not require functions for branch length optimization since the MCMC procedure is used to integrate over branch length distributions. Hence the PLF as used in Bayesian inference programs is less complex than for ML programs.

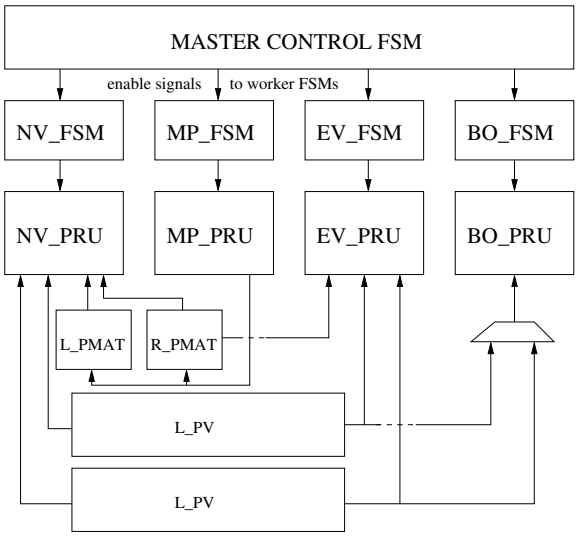

Fig. 2. Top-level design of the co-processor's architecture.

4. THE CO-PROCESSOR ARCHITECTURE

In the following we describe the architecture of our generic, reconfigurable PLF co-processor. Figure 2 depicts an abstract view of the top-level design. As already mentioned, RAxML [2] was used as reference for the design of the architecture. Thus, each processing unit (PRU) represents a hardware-optimized implementation of a RAxML likelihood function.

The *MASTER CONROL FSM* is located at the top of Figure 2. This FSM is vital, because it is the only component that can enable or terminate the operation of the co-processor. The master FSM directly communicates with the worker FSMs, which are placed below the *MASTER CONROL FSM* in Figure 2 and coordinate the correct operation of the PRUs. Each PRU is controlled by one worker FSM. The worker FSMs control the components (counters and comparators) that generate read and write addresses for the memory blocks of the co-processor memory subsystem. In addition, each worker FSM generates appropriate control signals for synchronizing the operation of the respective PRU with the data-fetch and write-back operations.

The co-processor architecture comprises the following processing units: *NV_PRU* (calculation of the ancestral probability vector), *MP_PRU* (calculation of the transition prob-

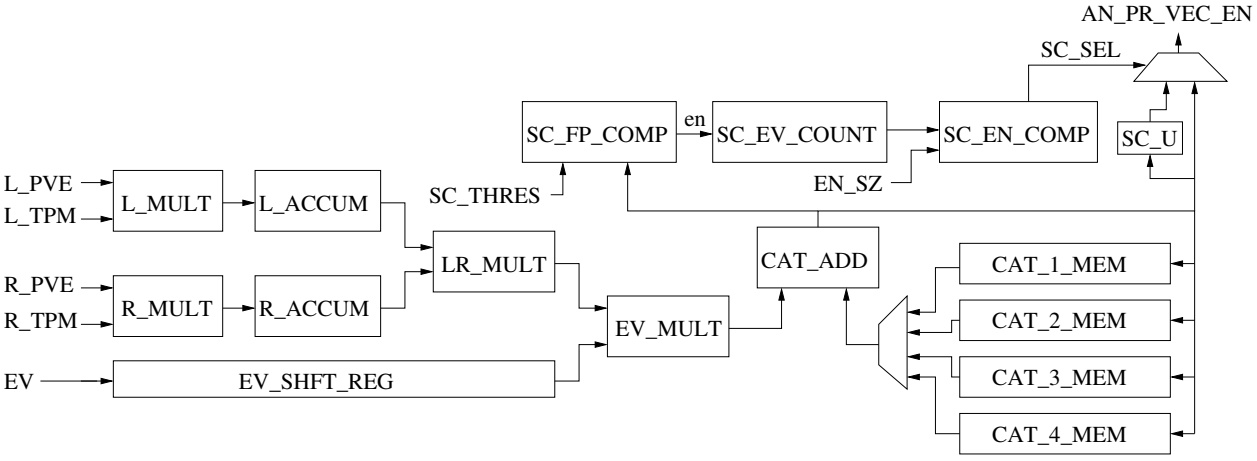

Fig. 3. Block diagram of the NV_PRU processing unit.

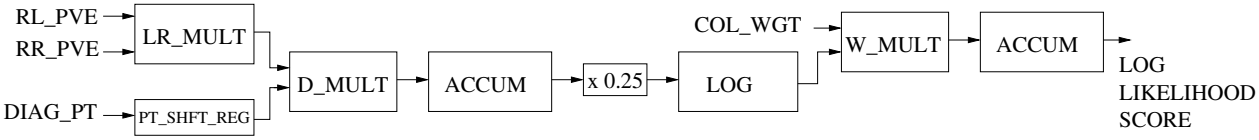

Fig. 4. Block diagram of the EV_PRU processing unit.

ability matrices), *EV_PRU* (calculation of the log-likelihood score), and *BO_PRU* (calculation of the PLF's first and second derivatives). The *NV_PRU* (Figure 3) executes the variable size (variable sizes are required for different input data types, see below) matrix-vector multiplication (see Equation 2) for computing ancestral probability vectors (including rate heterogeneity and numerical scaling). In Figure 3, the left and right child vector entries are denoted as *L_PVE* and *R_PVE*, while the respective left and right P matrices are denoted as *L_TPM* and *R_TPM*. The two *MULT - ACCUM* components implement the generic PLF architecture, that is, an architecture that can compute the likelihood on morphological, DNA, RNA secondary-structure, and protein data. The *CAT_ADD* adder and the 4 *CAT_X_MEM* memory components allow for repeatedly using the pipelined datapath. When a Γ model with 4 discrete rate categories is used, the datapath is traversed 4 times for each discrete rate (see Section 1). Finally, the numerical scaling subsystem is depicted at the top right of Figure 3. The comparator and counter components, denoted as *SC_FP_COMP* and *SC_EV_COUNT*, count the number of values (values smaller than a scaling threshold *SC_THRES*) that need to be scaled in the probability vector entry. Then, the *SC_EN_COMP* comparator decides whether the entire probability vector entry should be scaled before the write back operation. The *SC_U* unit scales all entries by multiplying double precision machine numbers with a constant that is a power of 2 (implemented by an addition in the exponent field). The final 2to1 multiplexer then selects the scaled or unscaled probability vector entry according to the signal from the *SC_EN_COMP* comparator.

When the *NV_PRU* has completed the PLF computations, the *EV_FSM* can trigger the operation of the *EV_PRU* (Figure 4) which calculates the overall log likelihood score for the tree at the virtual root. *RL_PVE* and *RR_PVE* are the probability vectors to the left and the right of the branch where the virtual root is located. The accumulator calculates the sum of the likelihoods of the 4 discrete Γ rates. The *x 0.25* multiplier that is connected to the accumulator's output bus is required because the Γ model assumes that all 4 discrete rates are equally probable. The *LOG* unit calculates the log likelihood score of the specific column/site. Because alignment columns with equal site patterns can be compressed into a single column, the log likelihood of each column is multiplied by an integer weight (*COL_WGT* in Figure 4) that correponds to the number of identical site patterns that have been compressed into a single column. Finally, the accumulator sums over the per-site log likelihoods to obtain the overall score for the tree.

The values of the transition probability matrices P, that is, *L_TPM* and *R_TPM* in Figure 3, are calculated by the *MP_PRU* (see Figure 5). The dimensions of the P matrices are not constant since they depend on the data type being analyzed; their size can vary between 2×2 (morphological data) to 20×20 (protein data).

Finally, the *BO_PRU* (Figure 6) is used to compute the

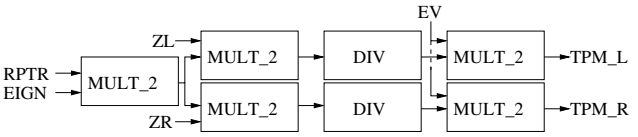

Fig. 5. Block diagram of the MP_PRU processing unit.

first and second derivative of the likelihood function. The derivatives are required for the Newton-Raphson procedure that optimizes branch lengths.

5. SYSTEM IMPLEMENTATION & EVALUATION

To verify the functionality of the PLF co-processor architecture, we implemented it in VHDL and mapped it onto a Xilinx Virtex 5 SX95T-1 FPGA. We used Modelsim 6.3f by Mentor Graphics as simulation tool and conducted extensive post place and route simulations for each processing unit. For testing, we executed RAxML on real-world datasets and created testbenches for the FPGA implementation by storing the input arguments for the functions implemented in the co-processor. Then, we compared the results calculated by the co-processor with those of the corresponding RAxML functions.

The majority of floating-point operations is executed on the FPGA by components that have been generated using the Xilinx Floating-Point Operator (Xilinx Floating Point Operator v4.0, `http://www.xilinx.com/support/ip_documentation/floating_point_ds335`, accessed July 2009). However, the accumulator and exponential operators have been generated using FloPoCo [23]. For computing logarithms, we used the Logarithmic Approximation Unit (LAU [12]).

Table 1 provides resource usage and performance data for each PRU as well as for a light-weight system instantiation that contains only one PRU of each type. As shown by Table 1, the entire co-processor design occupies only a fraction of the available reconfigurable resources on the chip. We did not attempt to exploit the full HW capacity of the FPGA, since our objective is to present and verify a general computer architecture for the PLF.

The performance of this generic co-processor architecture that can operate on all types of biological input data is affected by two factors. Firstly, the use of reconfigurable —flexible— hardware and secondly, the design of a generic, flexible, but nonetheless static design on programmable hardware. Our vision was to create a prototype design for an ASIC, thus a static architecture that can handle all types of biological data is required. To this end, the current FPGA implementation of the co-processor is between 2.8 and 4.5 times slower than the highly optimized SSE3-vectorized PLF RAxML implementation on an Intel processor running at 2.4 GHz. However, a dedicated, optimized accelerator-oriented version of our co-processor that only accomodates DNA data showed a speed-up of factor 6X on a Xilinx Virtex 5 SX240T. To allow for reproducing our results the VHDL source code of the co-processor is available at `http://wwwkramer.in.tum.de/exelixis/genericPLF_FPGA.tar.bz2`.

A question that may arise is why runtime reconfiguration is not deployed to improve performance. An initial assessment indicates that, because of the high frequency of reconfigurations required to make available optimized, data-type specific PRUs on the co-processor, execution times will be largely dominated by reconfiguration times.

For instance, RAxML requires 6,743 seconds to complete a tree search on a real-world dataset comprising 714 sequences (organisms) with a length of 1,231 nucleotides each. The total number of PLF function invocations that will be offloaded to the co-processor during the complete tree search amounts to 61,092,096, which corresponds to 9,060 reconfigurations per second. We used the Partial Reconfiguration Cost Calculator (PRCC, `http://users.isc.tuc.gr/~kpapadimitriou/prcc.html`, accessed October 2010) to estimate the reconfiguration time for the entire chip (based on the framework presented in [24]), assuming an average-size FPGA like the Virtex5 FX200T and a PowerPC as reconfiguration controller. The PRCC tool calculated a reconfiguration time of 576 milliseconds which means that 9,060 reconfigurations (the number of reconfigurations that would be required per second) will require approximately 5,218 seconds.

Thus, given the current state of technology, a system supporting runtime reconfiguration will not be able to outperform the static generic design we presented here. Therefore, our generic design can either serve as a prototype architecture for building an ASIC or for deriving accelerator-like FPGA implementations that are optimized for computing the PLF on specific data types, for instance, a dedicated design for DNA data. Alternatively, one could also think of a system comprising several FPGAs that operate in parallel and provide dedicated, optimized PRUs for specific data types.

6. CONCLUSION & FUTURE WORK

We presented a comprehensive and versatile design of a phylogenetic co-processor for ML-based inference of evolutionary trees. The architecture consists of hardware-optimized processing units, that have the same functionality as the corresponding highly optimized likelihood functions in RAxML. Thus, the co-processor, unlike previous designs, can be used for conducting real-world phylogenetic analysis. We used reconfigurable logic to verify functionality and correctness of the co-processor. We find that, present FPGA floating-point components are not capable of accomodating a highly generic implementation of the PLF. Thus, the present work

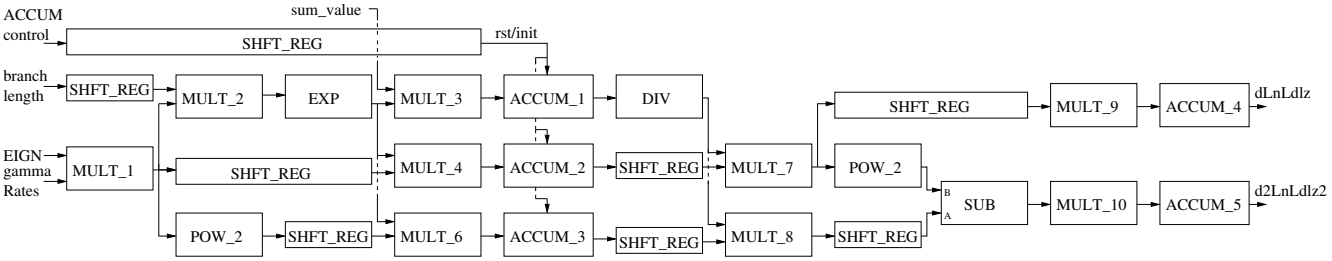

Fig. 6. Block diagram of the BO_PRU processing unit.

Resources-Performance	NV_PRU	MP_PRU	EV_PRU	BO_PRU	SYSTEM
Number of Slice Registers (58,880)	6,147	4,951	5,885	5,110	22,077
Number of Slice LUTs (58,880)	4,271	6,629	4,841	14,040	31,354
Occupied Slices	1,850	2,163	2,087	3,829	10,780
Number of BlockRAM/FIFO(36k)	0	7	3	3	21
Number of BlockRAM/FIFO(18k)	0	0	1	0	1
Number of DSPs	43	68	39	108	258
Maximum Frequency(MHz)	211	104	223	59	61

Table 1. Resource usage and performance report of the co-processor's processing units on a Virtex 5 SX95T FPGA.

can be regarded as proof-of-concept architecture for a potential ASIC or as starting point for building optimized FPGA accelerators for the PLF targeting specific input data types (e.g., DNA or protein data). Future work will address the deployment of FPGAs for such derived data-specific PLF accelerators based as well as the usage of alternative accelerators architectures (e.g., GPUs).

7. REFERENCES

[1] J. Felsenstein, "Evolutionary trees from DNA sequences: a maximum likelihood approach," *J. Mol. Evol.*, vol. 17, pp. 368–376, 1981.

[2] A. Stamatakis, "RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models," *Bioinformatics*, vol. 22, no. 21, pp. 2688–2690, 2006.

[3] F. Ronquist and J. Huelsenbeck, "MrBayes 3: Bayesian phylogenetic inference under mixed models," *Bioinformatics*, vol. 19, no. 12, pp. 1572–1574, 2003.

[4] 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.

[5] A. Stamatakis, *Bioinformatics: High Performance Parallel Computer Architectures*. CRC Press, Taylor & Francis, 2010, ch. Orchestrating the Phylogenetic Likelihood Function on Emerging Parallel Architectures, pp. 85–115.

[6] T. Mak and K. Lam, "Embedded computation of maximum-likelihood phylogeny inference using platform FPGA," in *Proceedings of IEEE Computational Systems Bioinformatics Conference (CSB 04)*, 2004, pp. 512–514.

[7] ——, "FPGA-Based Computation for Maximum Likelihood Phylogenetic Tree Evaluation," *Lecture Notes in Computer Science*, pp. 1076–1079, 2004.

[8] S. Zierke and J. Bakos, "FPGA acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods," *BMC bioinformatics*, vol. 11, no. 1, p. 184, 2010.

[9] N. Alachiotis, E. Sotiriades, A. Dollas, and A. Stamatakis, "Exploring FPGAs for Accelerating the Phylogenetic Likelihood Function," in *Proc. of HICOMB2009*, 2009, accepted for publication.

[10] N. Alachiotis, A. Stamatakis, E. Sotiriades, and A. Dollas, "An Architecture for the Phylogenetic Likelihood Function," in *Proceedings of FPL2009*, 2009, accepted for publication.

[11] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, "Numerical recipes in C," *The art of scientific computing (Cambridge: University Press*, vol. 3, no. 2, 1992.

[12] N. Alachiotis and A. Stamatakis, "Efficient Floating-Point Logarithm Unit for FPGAs," in *Proceedings of IPDPS 2010 (RAW Workshop)*, 2010, accepted for publication.

[13] Z. Yang, "Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites," *J. Mol. Evol.*, vol. 39, pp. 306–314, 1994.

[14] T. Jukes and C. Cantor, "Evolution of protein molecules," *Mammalian Protein Metabolism*, vol. 3, pp. 21–132, 1969.

[15] J. Ripplinger and J. Sullivan, "Does Choice in Model Selection Affect Maximum Likelihood Analysis?" *Syst. Biol.*, vol. 57, no. 1, pp. 76–85, 2008.

[16] J. Davis, S. Akella, and P. Waddell, "Accelerating phylogenetics computing on the desktop: experiments with executing UPGMA in programmable logic," in *Proceedings of 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 2, 2004.

[17] J. Bakos, "FPGA Acceleration of Gene Rearrangement Analysis," in *Proceedings of 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2007, pp. 85–94.

[18] J. Bakos, P. Elenis, and J. Tang, "FPGA Acceleration of Phylogeny Reconstruction for Whole Genome Data," in *Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering, 2007*, 2007, pp. 888–895.

[19] B. Moret, J. Tang, L. Wang, and T. Warnow, "Steps toward accurate reconstructions of phylogenies from gene-order data," *Journal of Computer and System Sciences*, vol. 65, no. 3, pp. 508–525, 2002.

[20] S. Kasap and K. Benkrid, "A high performance FPGA-based core for phylogenetic analysis with Maximum Parsimony method," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*. IEEE, 2010, pp. 271–277.

[21] P. A. Goloboff, S. A. Catalano, J. M. Mirande, C. A. Szumik, J. S. Arias, M. Källersjö, and J. S. Farris, "Phylogenetic analysis of 73060 taxa corroborates major eukaryotic groups," *Cladistics*, vol. 25, pp. 1–20, 2009.

[22] D. L. Swofford, *PAUP*: Phylogenetic analysis using parsimony (* and other methods), version 4.0b10*. Sinauer Associates, 2002.

[23] F. De Dinechin, C. Klein, and B. Pasca, "Generating high-performance custom floating-point pipelines," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*. IEEE, 2009, pp. 59–64.

[24] K. Papadimitriou, A. Anyfantis, and A. Dollas, "An effective framework to evaluate dynamic partial reconfiguration in fpga systems," *IEEE Transactions on Instrumentation and Measurement (TIM)*, vol. 59, no. 6, pp. 1642–1651, June 2010.