A RECONFIGURABLE ARCHITECTURE FOR THE PHYLOGENETIC LIKELIHOOD FUNCTION

Nikolaos Alachiotis, Alexandros Stamatakis [*]

The Exelixis Lab
Dept. of Computer Science
Technische Universität München
email: {alachiot,stamatak}@in.tum.de

Euripides Sotiriades, Apostolos Dollas

Microprocessor and Hardware Lab
Dept. of Electronic and Computer Engineering
Technical University of Crete
email: {esot,dollas}@mhl.tuc.gr

ABSTRACT

As FPGA devices become larger, the trend is to have more coarse-grain modules coupled with large scale reconfigurable fabric, thus enabling new classes of applications to run efficiently compared to a general-purpose computer. This paper presents an architecture that benefits from the large number of DSP modules in Xilinx technology to implement massive floating point arithmetic. Our architecture computes the Phylogenetic Likelihood Function (PLF) which is an important bioinformatics kernel. The PLF accounts for approximately 95% of total execution time in all state-of-the-art Maximum Likelihood (ML) based programs for reconstruction of evolutionary relationships. We validate and assess performance of our architecture using a highly optimized and parallelized SW implementation of the PLF that is based on RAxML, which is considered to be one of the fastest and most accurate programs for phylogenetic inference. Both software and hardware implementations use double precision floating point arithmetics. The new architecture achieves speed ups ranging from 1.6 up to 7.2 compared to a high-end 8-way dual-core general-purpose computer running the aforementioned highly optimized OpenMP-based multi-threaded version of the PLF.

1. INTRODUCTION

Several reconfigurable logic-based solutions for various bioinformatics algorithms and applications came to light in recent years. Bioinformatics algorithms that solve the DNA sequence matching problem such as Smith Waterman [1, 2] and BLAST [3, 4, 5] have frequently been mapped to FPGAs in the past. From a computer architecture point of view, these problems deal with data streaming as well as character matching issues and exhibit similar characteristics as applications from other domains, such as network processor and intrusion detection systems [6, 7]. While the reconfigurable architecture community has been involved with bioinformatics, several interesting problems characterized by great computational demands came to light. New FPGA devices offer hardware resources that can be used to build powerful floating point arithmetic architectures, which until recent years represented a weak point of reconfigurable technology. This feature now allows to deploy new generation FPGAs for novel classes of bioinformatics problems. One such challenging problem is the evaluation of the Phylogenetic Likelihood Function (PLF). Previous efforts to design reconfigurable architectures for this problem have been reported [8, 9] but due to the wide number of methods available and the biological data flood that is driven by new wet-lab technologies, the efficient computation of the PLF remains an open challenge.

The PLF is the most computationally intensive part of the RAxML algorithm [10] and a plethora of other PLF-based codes such as GARLI, MrBayes, PAML, or PAUP*. Those programs are widely used by biologists (the most popular ones have accumulated over 20,000 citations according to Google Scholar) to reconstruct the evolutionary history for a group of species by using the DNA sequences of the species under study. The input for PLF-based programs is a multiple sequence alignment, essentially an nxm data matrix, that contains n DNA sequences which all have a lenght of m nucleotide characters (m columns). More than 95% of overall execution time is spent to compute the PLF in the aforementioned programs. A phylogeny or phylogenetic tree is a binary tree structure that represents the evolutionary relationships among species. The tips (also called leaves or taxa) of the tree represent species alive today in contrast to internal (ancestral) nodes that represent species that have become extinct.

Phylogenetic trees have many important applications in medical and biological research (see [11] for a summary). In this paper we present a new architecture which significantly extends an initial proof-of-concept design [11] (preprint at http://wwwkramer.in.tum.de/exelixis/HICOMB2009.pdf). The initial architecture was only able to compute the PLF

[*]Part of this work is funded under the auspices of the Emmy-Noether program by the German Science Foundation (DFG)

on fully balanced trees, but already yielded a significant performance boost. The major extension in the more versatile architecture presented here is that it can evaluate the PLF for any given tree topology at the same speed as the previous architecture, it exploits the intrinsic parallelism of the PLF in a more flexible and scalable way, and that it is able to conduct so-called partial tree traversals which represent a fundamental mechanism in the design of current tree search algorithms. The proposed architecture yields exactly the same likelihood scores as the reference software. It has been fully post place and route simulated and executes several dozens of double-precision floating point operations during every clock cycle. Input/output issues have been taken into account to allow for mapping to a modern platform. An accelereated HW solution, can save valuable time, since current large-scale phylogenetic analyses projects with RAxML in collaboration with biologists require up to 2.25 million CPU hours on an IBM BlueGene/L supercomputer and up to 89GB of main memory [12].

2. COMPUTING THE PHYLOGENETIC LIKELIHOOD FUNCTION

Felsenstein's pruning algorithm [13] is the standard method to compute the PLF and hence the likelihood score for a given tree topology. In the following we provide an abstract description of this algorithm. The first step consists in tracking down 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 already been computed. The second step is to calculate the likelihood vector entries of the common ancestor (ancestral likelihood vector at k) and prune out the child nodes. These steps are executed recursively until the likelihood vector at the virtual root vr has been calculated and thus the pruning process has transformed the initial tree to only one node that is located at the virtual root. Phylogenetic trees under ML are unrooted for mathematical and computational reasons (see [13] for details), but a virtual root vr can be placed into any branch of the tree to evaluate its likelihood score.

In order to compute the PLF on a given, fixed, tree topology one also requires the branch lengths and the parameters of the statistical model of nucleotide substitution $P(b)$ which is a 4x4 matrix that provides the transition probabilities between nucleotide states A, C, G, T given a branch b. To compute the likelihood on a fixed tree with given branch lengths and model parameters, one initially needs to compute the entries for all ancestral likelihood vectors which are located at the inner nodes of the tree bottom up from the tips towards the virtual root. Every likelihood vector entry at position c $(c = 1...m)$ $\vec{L}(c)$ at the tips and at the inner nodes contains the four probabilities P(A), P(C), P(G), P(T) of observing a nucleotide A, C, G, or T at a specific column c of

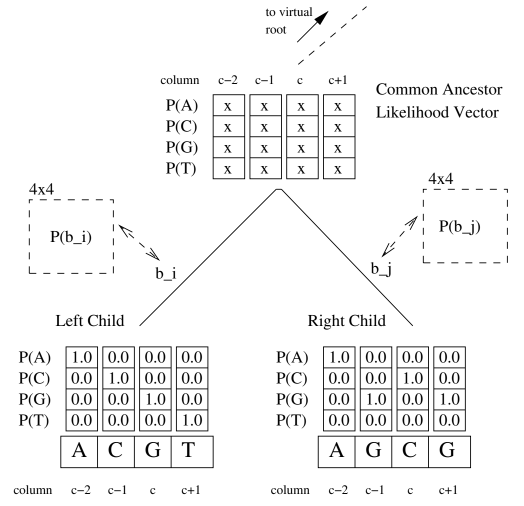


Fig. 1. Computation of ancestral likelihood vector entries.

the input alignment. The probabilities at the tips (leaves) of the tree for which observed data (the DNA sequences of the organisms under study) *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, a parent node k and two child nodes i and j, their likelihood vectors $\vec{L}^{(i)}$ and $\vec{L}^{(j)}$, the respective branch lengths leading to the childern b_i and b_j and the transition probability matrices $P(b_i), P(b_j)$, the likelihood of observing an A at position c of the ancestral (parent) vector $\vec{L}_A^{(k)}(c)$ is computed as follows:

$$\vec{L}_A^{(k)}(c) = \big(\sum_{S=A}^{T} P_{AS}(b_i)\vec{L}_S^{(i)}(c)\big)\big(\sum_{S=A}^{T} P_{AS}(b_j)\vec{L}_S^{(j)}(c)\big) \quad (1)$$

A schematic representation of this procedure is outlined in Figure 1. When the procedure reaches the root, the column likelihood $l(c)$ is computed as follows using the likelihood vector $\vec{L}^{(vr)}$ of the virtual root:

$$l(c) = \sum_{S=A}^{T} \pi_S {L_S}^{(vr)}(c) \quad (2)$$

The probabilities π_A through π_T are the prior probabilities (also called base frequencies) for observing A, C, G, or T at vr and are typically drawn empirically from the input data. To compute the overall likelihood we then compute the product over all $l(c)$.

3. RELATED WORK

While there exist many tools and methods for phylogeny reconstruction, only few of them have been mapped to hardware. Bakos *et al* [14] map GRAPPA [15] to an FPGA which is based on gene order input data. Phylogenetic inference using gene order data is mainly a discrete problem and therefore only requires few floating point operations. Phylogenetic analyses that use gene order instead of DNA sequence input data are rarely used for real-world analyses at present. Also, a simple sequence-based method called UPGMA (Unweighted Pair Group Method with Arithmetic Mean) has been mapped to HW [16]. UPGMA is one of the most simple tree reconstruction methods and is currently not used for real-world phylogenetic analyses. Mak and Lam [8, 9] report the mapping of a reduced floating point precision PLF implementation to FPGAs that is based on the simple Jukes-Cantor (JC69 [17]) model of nucleotide substitution. The most commonly used and most complex model is the GTR (General Time Reversible) model of nucleotide substitution which we also implement in our architecture.

4. ARCHITECTURE

We propose a master-worker architecture that consists of two main units: the Target Pair Unit (*TPU*, master) and the Computational Basic Core (*CBC*, worker). The *TPU* (master) performs the first of the previously described algorithmic steps while the *CBC* (worker) unit performs the second. The *TPU* executes the tree traversal steps of the pruning algorithm on the contents of a local memory which is used to hold information about the tree nodes and tips, i.e., the tree structure. The *TPU* tracks down which tips, inner nodes or which combination thereof should be combined to compute the entries of an ancestral likelihood vector. The information needed by the worker to locate the tip sequences or the likelihood vectors in the external or internal memories in order to start calculating the ancestral likelihood vector is provided via shared registers. Both the *TPU* as well as the *CBC* have access to these registers. Once the *TPU* has written the required addresses and selection bits, it sends a start signal to the *CBC* which indicates that there are valid data available in the registers. This means that the *CBC* can start its calculation process. At the same time the *TPU* switches to stand-by mode. The *TPU* now waits for the *CBC* to calculate the ancestral likelihood vector and write back (to the shared registers) the address and selection bits of the memory position where the newly computed ancestral likelihood vector has been stored. The *CBC* then announces that the calculation process has been completed by sending a respective signal to the *TPU*. The *TPU* will then update its local memory accordingly using the information from the shared registers and will determine the next pair of tips and/or nodes for which an ancestral vector needs to be computed.

We denote this design as master-worker architecture because only the *TPU* can initiate computations on the *CBC*. The general architectural scheme of the proposed design is illustrated in Figure 2.

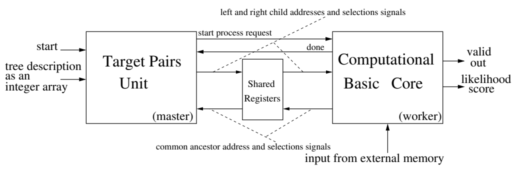

left and right child addresses and selections signals
start process request
done
start
tree description as an integer array
Target Pairs Unit
(master)
Shared Registers
Computational Basic Core
(worker)
valid out
likelihood score
common ancestor address and selections signals
input from external memory

Fig. 2. Architecture overview.

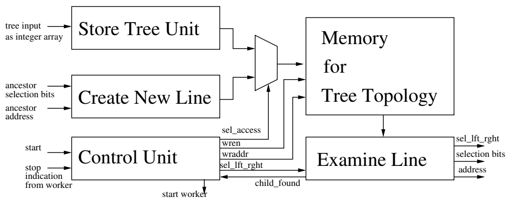


Fig. 3. Architecture of the Target Pair Unit (TPU).

4.1. The Target Pair Unit (TPU)

As already mentioned the *TPU* executes Felsenstein's pruning algorithm on the contents of its local memory. The top-level architecture of this unit is outlined in Figure 3. The unit consists of three structural components, a *Control Unit* implemented as FSM (Finite State Machine) and a local memory. The component *Examine Line* in coordination with the *Control Unit* execute the algorithm to determine the pairs of tips and/or nodes to be combined.

The *Store Tree* unit is used to initialize the contents of the local memory. It reads in the input tree to be evaluated that is encoded as an integer array. The integer array that describes the tree topology contains the node depths (distances from the virtual root) in depth-first order from the virtual root. For example, the tree illustrated in Figure 5 (top right) is given by the integer array: 1 (depth of A), 2 (depth of D), 3 (depth of C), 3 (depth of B). The *Create New Line* unit is used to back-transfer the information that has been written to the shared registers by the *CBC* which describes the ancestral vector that has just been calculated. The fields in the memory lines of the local memory which holds the tree structure are shown in Figure 4. The valid field indicates whether the memory line contains useful information for the remainder of the pruning process. The depth field contains the depth of the tip or node that the line describes. As already mentioned, the depth represents the depth-first node distance to the virtual root, e.g., the two child nodes of the virtual root have depth one. The remaining fields (address, ext_int_sel and lft_rght_sel) hold the necessary information for addressing the nucleotide sequences at the tips or the ancestral likelihood vectors. The address field is an index to the memory line that contains the first nucleotide of a DNA sequence (if the line denotes a tip) or the first likelihood vector entry (if the line represents an inner node). Both nucleotide sequences and ancestral likelihood vectors are

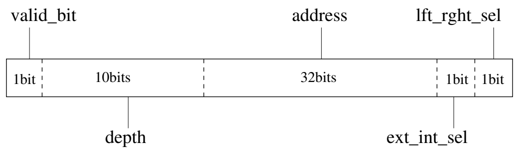
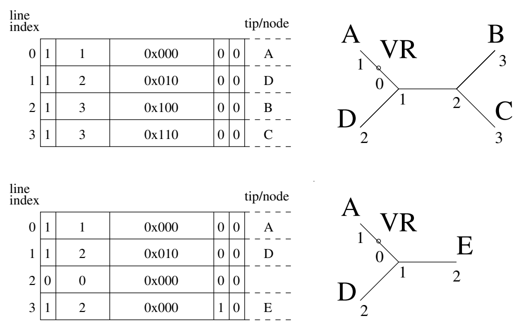


Fig. 4. Layout of a tree topology memory line.



line index						tip/node
0	1	1	0x000	0	0	A
1	1	2	0x010	0	0	D
2	1	3	0x100	0	0	B
3	1	3	0x110	0	0	C

line index						tip/node
0	1	1	0x000	0	0	A
1	1	2	0x010	0	0	D
2	0	0	0x000	0	0	
3	1	2	0x000	1	0	E

Fig. 5. Two consecutive snapshots of the tree topology memory and the respective trees.

stored contiguously in the external and the internal memory respectively. The address field contains a memory address but it has not yet been specified, whether this address refers to internal or external memory. This information is provided by the ext_int_sel (external/internal selection) field. If this bit is set, the address refers to the internal memory. Internal memory is organized into two big parts, thus the lft_rght_sel (left/right selection) field is used to select between these two parts. If this bit is set, then the address refers to the right part of internal memory. Each valid line represents a tip or a node vector of the tree. Two snapshots of the local memory that illustrate how the *TPU* works are depicted in Figure 5. For the four-taxon tree shown in the figure (top right), the master unit initializes the memory as shown by the table in the top left corner. Every tip of the tree has been labeled with a capital letter. The virtual root has been placed into the branch that connects A to the rest of the tree. There is also one number for every node (tips as well as internal nodes) of the tree that indicates the distance to the virtual root. The component *Examine Line* reads the memory from the top and stores the contents of the fields: address, ext_int_sel and lft_rght_sel to the shared registers. Then the *Control Unit* triggers the *CBC* to start the calculation. When the *CBC* has calculated the respective likelihood vectors it updates the shared registers with the address and selection bits of the ancestral vector which has been labeled by E in the second tree of Figure 5 (lower part). Then, the unit *Create New Line* updates the memory and the contents of it are available to the next step of the pruning algorithm. The memory update sequence is thus equivalent to the pruning steps in the tree.

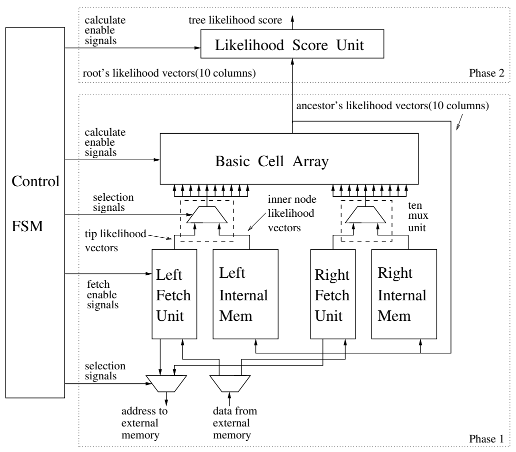


Fig. 6. Overview of the Computational Basic Core (CBC).

4.2. The Computational Basic Core (CBC)

Initially the *CBC* (worker unit) remains idle while the *TPU* determines the nodes to be combined. When the *CBC* is triggered by the *TPU* to start the calculations, it fetches the data at the position provided by the addresses and selection bits in the shared registers and starts the calculation of the likelihood vectors. An overview of the *CBC* architecture is provided in Figure 6. It consists of the Control FSM, the *Basic Cell Array*, the *Fetch Units* (FIFOs), internal memories and the *Likelihood Score Unit*. The *Fetch Units* have been specifically designed to hide the latency of linear external memory acesses to likelihood vectors (evidently the latency can not be hidden for the initial accesses to vector or tip addresses) from the *Basic Cells*. Since the likelihood vectors are long, i.e., typically $m \geq 1,000$, the latency for the access to the first datum of an array is negligible while we can achieve infinitely large burst ability.

The *Basic Cell Array* consists of ten *Basic Cells* which perform the double precision floating point additions and multiplications provided in Equation 1. All *Basic Cells* work in parallel on a different column of the sequence alignment which is stored in external memory. This design can easily be extended to a maximum of m *Basic Cells*, i.e., every *Basic Cell* can work concurrently to compute one of the entries of the respective vector $\vec{L}$ of length m. According to the node pair provided by the *TPU* the appropriate data are prefetched and stored in the *Fetch Units* or accessed directly in internal memories. The resulting ancestral likelihood vector is written to the internal memories. Once the likelihood vector of the virtual root has been computed, the Basic Cells are used again to calculate the per-column likelihood scores (see Equation 2) and the product over the per-column scores $l(c)$ is then computed by the *Likelihood Score Unit*.

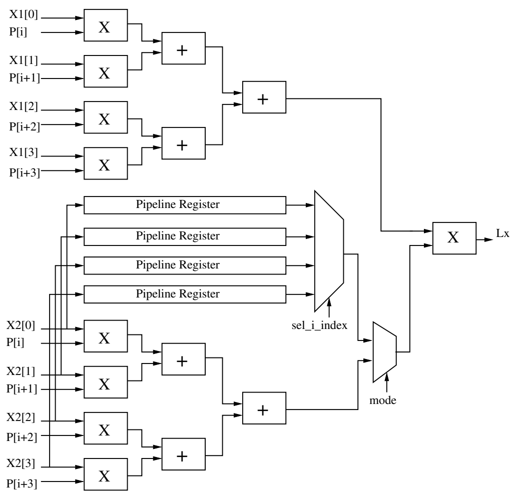


Fig. 7. Basic Cell Architecture.

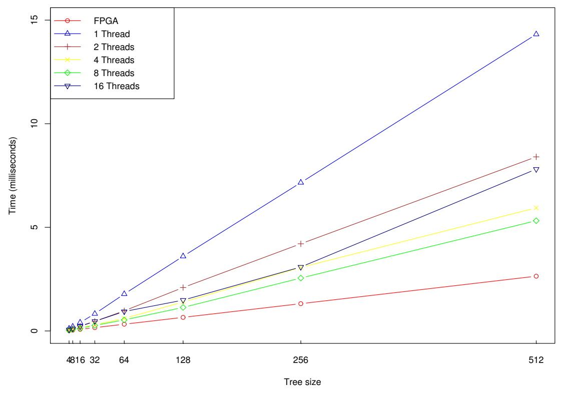


Fig. 8. FPGA versus multi-threaded execution times on the Sun x4600.

4.2.1. The Basic Cell Design

The *Basic Cell Array* consists of 10 *Basic Cells* that work in parallel. Each *Basic Cell* is arranged as a tree of double precision floating point adders and multipliers, which also operate in parallel as shown in Figure 7. The *Basic Cells* are fully pipelined with a total pipeline depth of 58 cycles. Each *Basic Cell* evaluates one likelihood value per cycle yielding one likelihood vector entry every 4 cycles (for all 4 nucleotides). Because the operations required for the calculation of the likelihood vector at the virtual root are slightly different, the *Basic Cell* also contains a vector of pipeline registers and a 4 to 1 multiplexer in order to perform the appropriate operations when the respective *mode* signal is set.

# taxa	1 Thr.	2 Thr.	4 Thr.	8 Thr.	16 Thr.
4	7.04	3.92	2.40	2.16	3.77
8	5.76	3.32	2.14	2.18	3.08
16	5.22	3.04	1.95	1.62	2.84
32	5.19	2.90	1.86	1.66	2.92
64	5.49	3.00	1.86	1.64	2.89
128	5.50	3.20	2.18	1.73	2.27
256	5.44	3.20	2.33	1.94	2.34
512	5.41	3.18	2.28	2.01	2.96

Table 1. Speedups of the hardware design compared to the multi-core software implementation.

5. SYSTEM EVALUATION & PERFORMANCE

Extensive post place and route simulations were conducted to verify the functionality of the proposed architecture. The input data sets contained trees with 4, 8, 16, 32, 64, 128, 256 and 512 taxa (tips/leaves) and a length of 1,000 nucleotides each. The results (likelihood scores of the trees) computed by the new architecture were exactly identical to those obtained by the software implementation. In order to conduct a fair performance comparison between the HW and SW implementations, we designed and optimized (based on 8 years of programming experience with phylogenetic inference software) a light-weight SW version to compute the PLF that omits the overhead of the standard RAxML open-source distribution. We compiled this light-weight version (available at http://wwwkramer.in.tum.de/exelixis/software) using the Intel icc compiler (v 10.1, optimization option -O3) that generates faster code than gcc for RAxML and also parallelized the PLF with OpenMP. We executed the program with 1, 2, 4, 8, and 16 threads on a high-end SUN x4600 system equipped with 8 dual core AMD Opteron processors running at 2.6 GHz and with 64 GB of main memory. In order to obtain accurate software timing results, we measured the time required by a loop that executes 20,000 full tree traversals. The proposed architecture was mapped to a Xilinx V5 SX240T FPGA. The design with 10 parallel basic cells uses 87% of the slice LUTs, 94% of the BRAMs, and 93% of the DSP48Es on this device.

The clock speed that was measured for the design amounts to 101 MHz (static timing report of the Xilinx Tools, ADVANCED 1.53 speed file). Figure 8 provides the projected FPGA execution times and actual software execution times on the Sun x4600 for 1, 2, 8, and 16 threads as a plot over input tree size for the respective data sets. As can be observed in the plot, the execution of the SW implementation with 16-threads is slower than with 8-threads. The reason for this slowdown and the lack of scalability lies in an unfavorable communication to computation ratio. For all test datasets, FPGA performance is better than that of a highly optimized SW implementation running in parallel on a high-end multicore machine. While the execution times are within the millisecond range, in real application scenarios, search algorithms will invoke likelihood computations millions of times to conduct ML estimates of model parameters and to search for the best ML tree topology which is an NP-complete optimization problem. As already mentioned current large-scale analyses can require up to 2.25 million CPU hours and 89GB of main memory. Hence, architectural solutions are urgently required to be able to handle the biological data flood in the near future.

6. CONCLUSION & FUTURE WORK

A new architecture which was jointly developed by a high performance computing bioinformatics group and a reconfigurable hardware group was presented. It yields exactly identical results (likelihood scores) as the equivalent software implementation. The speedups of 1.6–7.04 are modest, though within the typical order of magnitude for double precision floating point kernel implementations on FPGAs. Moreover, performance is compared in a fair way to a highly optimized and OpenMP-parallelized code on a high-end multi-core machine. Thus, the results presented here are encouraging and the insights and experience that has been gained can be used to further improve this new architecture. We will work towards redesigning the critical path in order to increase the clock speed, and plan for mapping the design to actual hardware and integrate it with software for phylogenetic inference. We will further extend the current hardware design by modules to calculate the likelihood score of very large trees (with respect to the number of taxa n), which require a scaling mechanism to avoid numerical underflow, as well as by modules for supporting the branch length optimization process which is conducted via an iterative Newton-Raphson procedure.

7. REFERENCES

[1] K. Puttegowda, W. Worek, N. Pappas, A. Dandapani, P. Athanas, and A. Dickerman, "A run-time reconfigurable system for gene-sequence searching," in *Proc. 16th Int. Conf. on VLSI Design*, 2003, pp. 561–566.

[2] S. Guccione, E. Keller, "Gene Matching Using JBits," in *Proc. 12th Int. Conf. on Field-Programmable Logic and Applications , Lecture Notes In Computer Science*, 2002, pp. 1168–1171.

[3] M. Herbordt, J. Model, Y. Gu, B. Sukhwani, and T. VanCourt, "Single pass, BLAST-like, approximate string matching on FPGAs," in *Proc. of 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006, pp. 217–226.

[4] A. Jacob, J. Lancaster, J. Buhler, and R. Chamberlain, "FPGA-accelerated seed generation in Mercury BLASTP," in *Proc. 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2007, pp. 95–106.

[5] F. Xia, Y. Dou, and J. Xu, "FPGA-Based Accelerators for BLAST Families with Multi-Seeds Detection and Parallel Extension," in *Proc. of 2nd Int. Conf. on Bioinformatics and Biomedical Engineering*, 2008, pp. 58–62.

[6] A. Dollas, D. Pnevmatikatos, N. Aslanides, S. Kavvadias, E. Sotiriades, S. Zogopoulos, K. Papademetriou, N. Chrysos, K. Harteros, E. Antonidakis, *et al.*, "Architecture and Application of PLATO, A Reconfigurable Active Network Platform," in *Proc. 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2001, pp. 101–110.

[7] I. Sourdis and D. Pnevmatikatos, "Pre-decoded CAMs for efficient and high-speed NIDS pattern matching," in *Proc. 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2004, pp. 258–267.

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

[9] T. Mak and K. Lam, "FPGA-Based Computation for Maximum Likelihood Phylogenetic Tree Evaluation," *Lecture Notes in Computer Science*, pp. 1076–1079, 2004.

[10] 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.

[11] 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.

[12] M. Ott, J. Zola, S. Aluru, A.D. Johnson, D. Janies, and A. Stamatakis, "Large-scale phylogenetic analysis on current HPC architectures," in *Scientific Programming*, 2008, pp. 255–270.

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

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

[15] B. Moret, J. Tang, L. Wang, and T. Warnow, "Steps toward accurate reconstructions of phylogenies from gene-order data," *J. Comp. Syst. Sci.*, vol. 65, no. 3, pp. 508–525, 2002.

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

[17] T. Jukes and C. Cantor, *Evolution of protein molecules.* Academic Press, New York, 1969, ch. III, pp. 21–132.