FPGA Optimizations for a Pipelined Floating-Point Exponential Unit

Nikolaos Alachiotis and Alexandros Stamatakis

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

Abstract. The large number of available DSP slices on new-generation FPGAs allows for efficient mapping and acceleration of floating-point intensive codes. Numerous scientific codes heavily rely on executing the exponential function. To this end, we present the design and implementation of a pipelined CORDIC/TD-based (COrdinate Rotation DIgital Computer/Table Driven) Exponential Approximation Unit (EAU) that will be made freely available for download (including the hardware description). The EAU supports single and double precision arithmetics and we provide appropriate configurations for Virtex2, Virtex4, and Virtex5 FPGAs. The architecture has been verified via simulations and by testing on a real FPGA. The implementation achieves the highest clock frequency reported in literature to date. Moreover, the EAU only occupies 5% of hardware resources on a medium-size FPGA such as the Virtex 5 SX95T. In addition, a general framework for safely conducting application-specific optimizations of floating-point operators on FPGAs is presented. We apply this framework to a bioinformatics application and optimize the EAU architecture using width-reduced floating-point operators and application-specific performance tuning. The optimized application-specific EAU occupies approximately 70% less hardware resources than the initial single precision implementation.

Keywords: floating point, exponential, FPGA, CORDIC, Table-Driven

1 Introduction

Field Programmable Gate Arrays (FPGAs) are increasingly being used as accelerators for floating-point intensive scientific applications which suffer from long execution times. Furthermore, the unprecedented growth of FPGAs in terms of reconfigurable resources, in particular with respect to the number DSP slices and memory blocks available, has facilitated their deployment as accelerator devices for floating-point intensive codes. Because of their inherent complexity, widely used functions such as the exponential function require an excessive amount of reconfigurable resources when both good performance *and* high accuracy are desired.

Our research focuses on developing a reconfigurable phylogenetic co-processor [1, 2] for RAxML [3], a Bioinformatics program, which evaluates the Maximum

Likelihood function on evolutionary trees. The likelihood computations require frequent evaluation of the exponential function.

There already exist several implementations of the exponential function on FPGAs [4–10]. While most of these implementations provide high numerical accuracy—that may however not always be required—only one is freely available for download. This open-source exponential unit can be generated using the FloPoCo tool suite [9].

Here, we present and make available (`http://wwwkramer.in.tum.de/exelixis/expFPGA.tar.bz2`) a single precision pipelined floating-point Exponential Approximation Unit (EAU) that is based on similar design principles as our recently released Logarithm Approximation Unit (LAU) [11]. Moreover, we apply a RAxML-specific optimization/adaptation process to further improve the performance of the EAU co-processor and to reduce the amount of hardware resources that are required for the calculation of the exponential function. Based upon this optimized single precision EAU configuration, we have developed a double precision EAU that will also become available for download.

The EAU architecture is based on a TD (Table-Driven) implementation of the CORDIC (COrdinate Rotation DIgital Computer) algorithm. CORDIC (also known as Volder's algorithm [12]) is a digit-by-digit method that relies on additions, shift operations, and read-only memory. The original algorithm that focused on calculating trigonometric functions was later extended by Walther [13] to compute functions like the logarithm and the exponential. This iterative algorithm generally requires resource-intensive hardware implementations to attain sufficient accuracy levels coupled with high performance. However, accuracy and performance requirements are determined by the application at hand. Thus, it may be desirable to sacrifice a certain amount of accuracy and/or speed for saving hardware resources and thereby make more resources available to the larger, potentially slower, and more complex overall hardware design (e.g., RAxML) into which an exponential unit is embedded.

Based upon this rationale, the EAU, can be optimized and adapted with respect to the architecture that will be using it, in our specific case, the phylogenetic co-processor. Design and optimization decisions have thus been made to generate a unit that is both resource-efficient and does not decrease the average performance of the entire co-processor.

The remainder of this paper is organized as follows: In Section 2 we review previous work on FPGA-based exponential units. Section 3 introduces the EAU architecture. The application-specific optimization framework that was applied to the single precision EAU is described in Section 4. In Section 5 we assess EAU performance and conduct a performance comparison with competing implementations. We conclude in Section 6.

2 Related Work

There already exist various alternative implementations of floating-point exponential units for FPGAs [4–10, 16].

Boudadous [16] presented a CORDIC algorithm on a Xilinx VirtexE FPGA. However, as discussed in the evaluation section of [16] this implementation suffers from a comparatively high relative error rate.

The implementations presented in [8] and [9, 10] only address SP (single precision) exponential units, while those presented in [6, 7] and [4, 5] focus on DP (double precision). Our EAU architecture can accomodate both SP (SP-EAU) and DP (DP-EAU). The number of DSP slices and memory blocks used is slightly higher for the DP-EAU. Since the amount of DSP slices and memory blocks increases significantly with each FPGA generation, the EAU architecture is thus well-suited for new-generation FPGAs (e.g., Virtex 5 and 6 families).

With respect to SP, Doss *et. al* [8] presented a pipelined table-driven approach that occupies as much as 5,564 slices on a Virtex II FPGA and operates at a maximum clock frequency of 85MHz. Detrey *et. al* [10, 9] presented an alternative table-driven approach which is significantly more efficient than the implementation by Doss [8] in terms of resources (only 948 slices are required). At the same time the implementation by Detrey exhibits a higher clock frequency (100MHz) when mapped to a FPGA of the same family (Virtex II). While it is not entirely clear from the paper what the accuracy of the implementation by Doss is, the core by Detrey *et. al* offers last-bit accuracy. The SP-EAU does not attain this level of accuracy, since the optimization stategy (see Section 4) that has been adopted for the design of the EAU targets a specific application (RAxML), that does not require such a high degree of accuracy for numerical stability. Furthermore, the SP-EAU occupies less resources than [8] but is not as resource-efficient as [10, 9]. Nonetheless, the SP-EAU clearly outperforms both aforementioned implementations in terms of maximum clock frequency (168 MHz on a Virtex II).

Regarding DP, Jamro *et. al* [6, 7] also presented a table-driven implementation that occupies approximately 5,000 slices on a Virtex 4 FPGA. It exhibits a very low latency (27 clock cycles), high accuracy (to meet precision requirements of a quantum chemistry application), and has the second-highest clock frequency (166MHz) after the DP-EAU. Pottathuparambil *et. al* [5] presented, and recently improved [4], a CORDIC implementation which was also mapped to a Virtex 4 FPGA. The most recent paper [4] introduced a more efficient implementation than [6, 7] in terms of resources, a latency of 258 cycles, and a clock frequency of 100 MHz. The implementation is partially pipelined, but due to the iterative procedure, the pipeline needs to be flushed after every iteration. In every iteration the computation of 5 DP values can be accomodated since this corresponds to the pipeline length of the iterative part. The RAxML-optimized DP-EAU architecture is partially pipelined as well and the computation of 11 DP values can be accomodated during each iteration. To allow for a fair performance comparison, the DP-EAU was also mapped to a Virtex 4 FPGA. The implementations by Jamro [6, 7] occupy more FPGA slices than the DP-EAU, while the one presented in [4, 5] is more resource-efficient than both the Jamro as well as the DP-EAU implementations.

The EAU significantly outperforms both [6, 7] and [5, 4] implementations in terms of maximum clock frequency (252 MHz). The EAU architecture can support both SP and DP and has been designed for applications that do not require maximum accuracy. It offers a fully pipelined and also a partially pipelined mechanism and represents a sufficiently accurate implementation at the highest maximum clock frequency reported in literature to date.

3 The EAU Architecture

The EAU architecture represents a one-to-one transformation into VHDL of the exponential function implemenetation in C which forms part of the most recent release of RAxML (v7.2.8, `http://wwwkramer.in.tum.de/exelixis/software.html`). The RAxML C exponential function implementation is based on the CORDIC C++ library by Burkardt [14].

Throughout the paper, we denote the C++ exponential function of the CORDIC library [14] as *EXP_CORDIC*. The TD extension of *EXP_CORDIC* that was integrated into the RAxML C code is denoted as *EXP_FPGA*. It also represents an *exact* software model of the EAU hardware architecture. Furthermore, by SP, DP, and FP we denote IEEE-754 single precision arithmetics, IEEE-754 double precision arithmetics, and floating-point representations respectively.

The *EXP_CORDIC* code performs 4 operations denoted by the author as: *Determine Weights*, *Calculate Products*, *Perform Residual Multiplication*, and *Account for factor EXP (X_INP)*, where *X_INP = floor (INPUT)*. The first two operations execute *for-loops* that perform a predetermined—fixed—number of iterations. The third operation is a static mathematical function that combines the results of the first two fields. The last operation, which calculates the factor *EXP(X_INP)*, is implemented as an iterative process using multiplications/divisions. The number of iterations and hence the execution time for this operation are not known *a priori*, since they are determined by the absolute value of *X_INP*. Such an unpredictable behavior is problematic for a pipelined hardware architecture, since results can be produced at unexpected clock cycles. In order to overcome this drawback of *EXP_CORDIC* in the EAU, the respective *Account for factor EXP(X_INP)* operation of *EXP_FPGA* deploys a lookup table.

Figure 1 depicts the fully pipelined general EAU architecture (left) and the optimized application-specific unit for RAxML (right). The design consists of three components denoted as *PRE_ITER*, *ITER* and *AFT_ITER*.

The *PRE_ITER* component splits the input value into respective integer and decimal values. The *floor()* function (see above) has been implemented using a subtracter and a Xilinx Floating-Point Operator (FPO) [15] configured for float-to-fixed operations. The subtracter is used to execute the operation: $input\_number - 0.5$. This is necessary because the default (and sole available) rounding mode of the Xilinx FPO is "Round to Nearest", as defined by the IEEE-754 Standard.

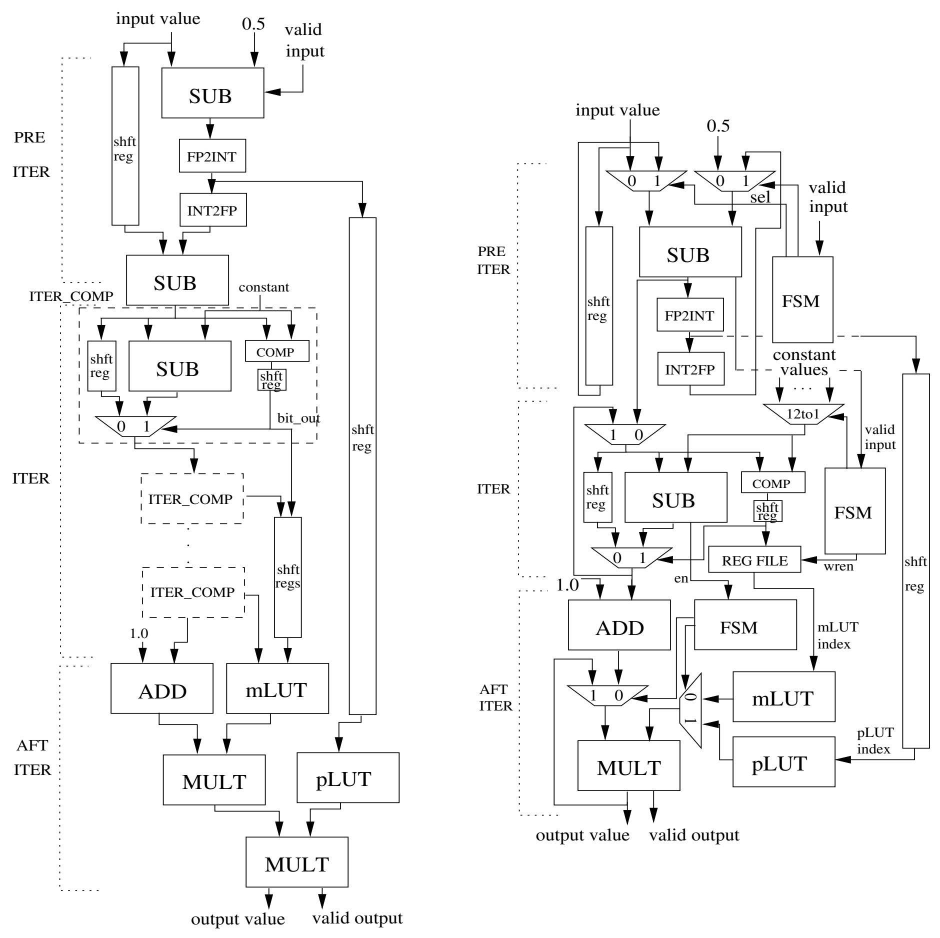

Fig. 1. Block diagram of the general EAU architecture (left) and the optimized, application-specific EAU architecture (right).

The *ITER* component corresponds to the *Determine Weights* operation of *EXP_CORDIC*. This component calculates an array of FP values and reduces the decimal part of the input argument by using a parameter that is divided by two in each iteration. The array of FP values is used by the iterative *Calculate Products* operation to compute a base value which is then provided as input to the *Perform Residual Multiplication* field. In order to further optimize this part of the algorithm, *EXP_FPGA* (the C code as well as the hardware implementation) calculates a bit-vector (instead of a FP array) during the iterative part. This bit-vector is then used as an index for a second lookup table that contains values that would otherwise need to be calculated by the *Calculate Products* field.

Finally, the *AFT_ITER* component implements the remaining operations: *Calculate Products*, *Perform Residual Multiplication*, and *Account for factor EXP(X_INP)*. As already mentioned, the *Calculate Products* and *Account for factor EXP(X_INP)* fields are calculated via lookup-tables (*m_LUT and p_LUT* in Figure 1) while the *Perform Residual Multiplication* operation only requires an addition and a multiplication.

4 Application-Specific Optimization

The EAU architecture has been extensively optimized. The optimizations aim to reduce the hardware resources occupied by the unit, while not affecting the overall accuracy of the phylogenetic coprocessor into which the EAU is embedded. In order to identify the type of optimizations that can be applied, a set of experiments were conducted. We tested the behavior of RAxML in software for various accuracy levels of the exponential function and analyzed the exponential function call pattern, that is, at what frequency the specific function is invoked and with which *load* each time. Here, the term *load* refers to the number of consecutive exponentials the program needs to calculate without other intervening arithmetic operations, that is, *for-loops* that only contain calls to the exponential function can be used for determining the (maximum) exponential function *load* of RAxML.

The accuracy required by the exponential function in the application allowed for further reducing the number of iterations in *EXP_CORDIC*. *EXP_FPGA* only executes 12 iterations without significantly (in a statistical sense) affecting the behavior of RAxML. The next step consists of quantifying the maximum relative error of *EXP_FPGA* versus the GNU DP EXP (GNU C Library [17]) function that is used by RAxML. This information allowed to safely apply the reduce-width technique [6, 7] for floating-point operators, that is, the use of FP operators that do not comply with the IEEE-754 standard for SP/DP representations. When the width is reduced, the exponent and mantissa fields only contain as many bits as necessary to attain the required arithmetic range and precision. Furthermore, we also appropriately decreased the width of the floating-point representations in the lookup tables to reduce the number of required memory blocks.

The application of the reduce-width technique alone was not sufficient to decrease the EAU hardware resource requirements to an acceptable level, since not all operators of the EAU can be width-reduced. In order to determine if an operator can be reduced or not, we simulated the behavior of width-reduced operators in *EXP_FPGA* by reducing the width of the input and output arguments of the operator.

Finally, the largest reduction in EAU resource utilization was achieved by analyzing the exponential function call pattern and respective *load* in RAxML. A thorough analysis of the source code revealed that the exponential function is called a specific number of times that is determined by the *load*. In between such sequences of exponential invocations, a huge number of multiplications and additions is carried out that perform other parts of the likelihood calculation.

The time window, during which only multiplications and additions are carried out, allows for further optimization of the EAU with respect to hardware resource utilization, by reducing the unit's pipelining capability. The rationale is that, a pipelined EAU architecture which can accomodate more exponential calls than the maximum *load* of the application using it, will exhibit exactly the same performance as an EAU whose pipeline matches the *load*.

Thus, the limit-pipeline optimization approach allowed us to further reduce the hardware footprint of the EAU by utilizing only a single width-reduced subtracter for the *ITER* component. As already mentioned in Section 2 the latency of the iterative part of the EAU determines the maximum number of input values that can be accomodated at each invocation of the exponential function. The current EAU configuration utilizes a minimal amount of hardware by occupying only one subtracter in the respective *ITER* component. Thus, only 11 exponentials can be calculated between pipeline-flushes. Note that, this configuration is not sufficient to accomodate the maximum exponential *load* of RAxML, which can amount to 40 consecutive exponential calculations when protein data is analyzed. It can fully accomodate the *load* of DNA analyses though. Therefore, further tuning is required to determine the optimal number of input values that the EAU should be able to accomodate at each call and/or whether two or more partially pipelined EAUs are more efficient than a single, fully pipelined, one.

5 Evaluation & Performance

Dataset	RAxML with GNU EXP	RAxML with EXP_FPGA
44-355	-11231.33	-11231.32
90-1524	-54074.89	-54077.97
150-1130	-39606.93	-39611.42
218-1846	-134160.35	-134166.02
140-1041	-120849.31	-120849.41

Table 1. Log-likelihood score deviation of RAxML with EXP_FPGA.

Initially, we verified the functionality of the EAU architecture (Section 5.1) and examined the overall behavior of the application (Section 5.2). A detailed resource usage, accuracy, and performance analysis on the same FPGA device that

	SP-EAU	DP-EAU
RESOURCES-Total		
slice registers-58,800	1792	3131
slice LUTs-58,800	1669	2909
occupied slices-14,720	724	1085
# 36k blockRAM-244	2	6
# DSP48Es-640	6	15
ACCURACY		
Max RE*	$0.83 * 2^{-17}$	$0.86 * 2^{-23}$
Mean RE*	$0.66 * 2^{-19}$	$0.59 * 2^{-24}$
PERFORMANCE		
Clock Frequency(MHz)	317,6	307,9
Latency(# clock cycles)	212	222

Table 2. Resource Usage, Accuracy and Performance of the SP-/DP-EAUs on a Virtex 5 SX95T-2 FPGA. (* Relative Error)

was used for verification can be found in Section 5.3. Section 5.4 describes how each step of the optimization process affected the efficiency of the architecture. Finally, Section 5.5 provides a comparison with other hardware and software implementations.

5.1 Hardware Verification

The EAU was implemented in VHDL, using the Xilinx ISE Suite 10.1 for synthesis and post place and route. In order to verify correctness of the proposed architecture, we conducted extensive post place and route simulations as well as tests on an actual FPGA. As simulation tool we used Modelsim 6.3f by Mentor Graphics. For hardware verification we used the HTG-V5-PCIE development platform equipped with a Xilinx Virtex 5 SX95T-1 FPGA. The advanced verification tool Chipscope Pro Analyzer was used to monitor the output port of the SP- and DP-EAUs and the expected signals for given input numbers were tracked.

5.2 Application-Level Error Analysis

Due to the application-specific optimization process described in Section 4, the accuracy of the EAU is inferior compared to other existing implementations. Nonetheless, the accuracy is sufficient for our target application. Table 1 shows the log-likelihood score deviation of RAxML with and without using the EAU for the exponential on various real-world biological datasets. Based on standard statistical significance tests (as implemented in the CONSEL package [21]), that are commonly used in phylogenetics (see [20] for a review), the differences (induced by the approximation of the exponential function) of log-likelihood scores between trees is not statistically significant. Therefore, an EAU with 12 iterations in the *ITER* component provides sufficient accuracy for RAxML.

5.3 EAU Architecture Evaluation

Table 2 contains the resource utilization report for the EAU after the post place and route process. Accuracy results (maximum and mean relative error) as well as performance (maximum clock frequency and latency) for the SP and DP configurations are also provided. Benchmarks with 10^8 random numbers were used to measure the maximum and mean relative error of the implementations.

5.4 Optimization Techniques Evaluation

	initial	width-reduced	pipeline-limited
slice registers-58,800	5964	6853	1792
slice LUTs-58,800	5396	6433	1669
occupied slices-14,720	2020	2274	724
# 36k blockRAMs-244	2	2	2
# DSP48Es-640	34	10	6
max frequency(MHz)	305,9	325,4	317,6
latency(# clock cycles)	194	194	212

Table 3. The effect of each optimization step on the SP-EAU when mapped on a Virtex 5 SX95T-2 FPGA.

The initial SP-EAU implementation executes 12 iterations and is fully pipelined. The optimization techniques described in Section 4 were then applied step-by-step to this basic design. Table 3 shows the effect of each application-specific optimization step on hardware resources, clock frequency, and latency.

The reduce-width optimization yielded a significant reduction of DSP slices, while slightly increasing reconfigurable resource utilization (slice registers, slice LUTs, occupied slices). This redistribution of resources, that is, less DSPs and more slices, is due to different implementation decisions made by the Xilinx FPO operator, when custom-sized floating-point operators are generated (i.e., reconfigurable resources are used instead of DSPs).

The limit-pipeline optimization further reduced the number of DSP slices as well as all other reconfigurable resources used. Clock frequency and latency slightly increased after application of all optimization steps. The increased latency is caused by the width-reduced floating-point operators, that are reconfigured after the second optimization step (limit-pipeline) to obtain a final maximum clock frequency that is roughly equivalent to the initial clock frequency of the EAU.

5.5 Comparison with other implementations

Details	SP-EAU	Doss [8]	Detrey [10]
Style	CORDIC + TD	TD	TD
Error (Max)	$0.83 * 2^{-17}$	NA*	2^{-23}
Slices	2483	5564	948
Max Frequency	168 MHz	85 MHz	100 MHz
Latency	212 cycles	NA*	85ns

Table 4. Resource Usage, Accuracy and Performance comparison of SP implementations. (* Not Available in [8])

Details	DP-EAU	Pottathuparabil [5]	Jamro [6]
Style	CORDIC + TD	CORDIC	TD
Error (Max)	$0.86 * 2^{-23}$	2^{-53}	0.4708 *
Slices	3407	2024	5000
Max Frequency	252 MHz	100 MHz	161 MHz
Latency	224 cycles	258 cycles	27 cycles

Table 5. Resource Usage, Accuracy and Performance comparison of DP implementations. (* This value is *mean* absolute error measured in ULP (Unit in the last place [18]).)

As explained in Section 2 the SP-EAU was also mapped to a Virtex II device in order to conduct a fair comparison between our implementation and those presented in [8] and [10]. Accordingly, the DP-EAU was also mapped to a Virtex 4 device and compared to the respective DP implementations [6, 5]. Tables 4 and 5 provide a summary of these comparisons.

Furthermore, we assessed the performance of the EAU architecture with respect to software implementations: SP-/DP-GNU exponential functions [17] and SP-/DP-MKL (Intel Math Kernel Library [19]) exponential functions. In order to take full advantage of the Intel Core2 Duo T9600 processor@2.8GHz that was used for the comparisons, we used the Intel icc compiler with appropriate optimization flags.

Table 6 provides the execution times of the SP/DP GNU functions (expf,exp) and SP/DP MKL functions (vsExp,vdExp) for 10^8 invocations of the functions. The SP GNU function (expf) is known to be slower than the DP GNU function (exp; see comments in the expf source code).

The fully pipelined SP-EAU implementation is 94 times faster than the GNU expf function and as fast as the MKL vsExp function. As expected, the optimized (width-reduced and pipeline-limited) SP and DP EAUs are slower than the MKL implementations. Nonetheless, the SP-EAU is 7 times faster than the SP GNU function and the DP GNU function is only 1.34 times faster than the DP-EAU.

Implementation	Execution Time	Clock Frequency
expf (gcc)	29,408	2,8 GHz
vsExp (icc)	0,312	2,8 GHz
exp (gcc)	3,268	2,8 GHz
vdExp (icc)	0,632	2,8 GHz
Initial SP-EAU	0,31	305 MHz
Pipeline-Limited SP-EAU	4,2	317 MHz
Pipeline-Limited DP-EAU	4,4	307 MHz

Table 6. Execution times (in seconds) of the GNU, MKL and EAU implementations for 10^8 invocations.

6 Conclusion & Future Work

A new architecture to calculate an approximation of the exponential function in reconfigurable logic under SP and DP was presented. The SP-/DP-EAU configurations for several FPGA families (Virtex2, Virtex4 and Virtex5) will be made freely available for download. The functionality of the EAU architecture was verified on real hardware and both (SP/DP) configurations occupy less than 5% of overall hardware resources on a medium-sized new-generation FPGA like the Virtex5 SX95T.

Furthermore, the EAU component was optimized for being used as an embedded component in a larger and more complex reconfigurable phylogenetic co-processor that is currently under development. We provide a detailed description of this application-specific optimization process and individually assess the effects of every optimization step.

Future work will focus on the integration of the EAU into the phylogenetic co-processor and on further tuning after integration, that is, to determine the optimal number and EAU(s) configuration in terms of performance, resources, and number of units deployed.

References

1. Alachiotis, N., Sotiriades, E., Dollas, A., Stamatakis, A.: Exploring FPGAs for Accelerating the Phylogenetic Likelihood Function. Proceedings of HICOMB2009, pp. 1–8, Rome, Italy (2009)
2. Alachiotis, N., Stamatakis, A., Sotiriades, E., Dollas, A.: A Reconfigurable Architecture for the Phylogenetic Likelihood Function. Proceedings of FPL 2009, pp. 674–678, Prague, September (2009)

3. Stamatakis, A.: RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. Bioinformatics, vol. 22, no. 21, pp. 2688–2690 (2006)
4. Pottathuparambil, R., Sass, R.: A Parallel/Vectorized Double-Precision Exponential Core to Accelerate Computational Science Applications. Proceedings of FPGA 2009, Monterey, California, USA, pp. 285–285 (2009)
5. Pottathuparambil, R., Sass, R.: Implementation of a CORDIC-based Double-Precision Exponential Core on an FPGA. Proceedings of RSSI 2008, Urbana, Illinois, USA (2008)
6. Jamro, E., Wiatr, K., Wielgosz, M.: FPGA Implementation of 64-bit Exponential Function for HPC. Proceedings of FPL 2007, pp. 718–721 (2007)
7. Wielgosz, M., Jamro, E., Wiatr, K.: Highly Efficient Structure of 64-bit Exponential Function Implemented in FPGAs. Proceedings of ARC 2008, pp. 274–279, London, UK (2008)
8. Doss, C. C., Robert, J., Riley, L.: FPGA-based Implementation of a Robust IEEE-754 Exponential Unit. Proceedings of FCCM 2004, pp. 229–238 (2004)
9. de Dinechin, F., Klein, C., Pasca, B.: Generating High-Performance Custom Floating-Point Pipelines. Proceedings of FPL 2009, Prague (2009)
10. Detrey, J., de Dinechin, F.: Parameterized Floating-Point Logarithm and Exponential Functions for FPGAs. Proceedings of Microprocess. Microsyst., pp. 537–545 (2007)
11. Alachiotis, N., Stamatakis, A.: Efficient Floating-Point Logarithm Unit for FPGAs. Proceedings of RAW2010, pp. 1–8, Atlanta, GA, USA (2010)
12. Volder, J. E.: The CORDIC trigonometric computing technique. Proceedings of IRE Transactions on Electronic Computers, pp. 330–334 (1959)
13. Walther, J. S.: A Unified Algorithm for Elementary Functions. Spring Joint Computer Conference, pp. 379–385 (1971)
14. Burkardt, J.: CORDCIC Approximation of Elementary Functions. http://people.sc.fsu.edu/ burkardt/cpp_src/cordic/cordic.html (last visited: 17-05-2010)
15. Xilinx: Floating Point Operator v.4.0. http://www.xilinx.com/support/documentation/ip_documentation/floating_point_ds335.pdf (last visited: 17-05-2010)
16. Boudabous, A., Ghozzi, F., Kharrat, M., Masmoudi, N.: Implementation of Hyperbolic Functions using CORDIC Algorithm. Proceedings of ICM 2001, pp. 738–741 (2004)
17. McGrath, R.: GNU C Library. http://www.gnu.org/software/libc (last visited: 17-05-2010)
18. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv. pp. 5–48 (1991)
19. Intel: Intel Math Kernel Library Reference Manual. http://www.intel.com/software/products/mkl/docs/ WebHelp/mkl.htm
20. Goldman, N., Anderson, J.P., Rodrigo, A.G.: Likelihood-based tests of topologies in phylogenetics. Systematic Biology, vol. 49, no. 4, pp. 652–670 (2000)
21. Shimodaira, H., Hasegawa, M.: CONSEL: for assessing the confidence of phylogenetic tree selection. Bioinformatics, vol. 17, no. 12, pp. 1246 (2001)