A HMMER Hardware Accelerator using Divergences

Juan Fernando Eusse Giraldo
Department of Electrical Engineering
University of Brasilia
Brasília/DF, Brazil
eusse@microe.udea.edu.co

Nahri Moreano
School of Computing
University of Mato Grosso do Sul
Campo Grande/MS, Brazil
nahri@facom.ufms.br

Ricardo Pezzuol Jacobi
Department of Computer Science
University of Brasilia
Brasília/DF, Brazil
jacobi@unb.br

Alba Cristina Magalhaes Alves de Melo
Department of Computer Science
University of Brasilia
Brasília/DF, Brazil
albamm@cic.unb.br

Abstract - As new protein sequences are discovered on an everyday basis and protein databases continue to grow exponentially with time, computational tools take more and more time to search protein databases to discover the common ancestors of them. HMMER is among the most used tools in protein search and comparison and multiple efforts have been made to accelerate its execution by using dedicated hardware prototyped on FPGAs. In this paper we introduce a novel algorithm called the Divergence Algorithm, which not only enables the FPGA accelerator to reduce execution time, but also enables further acceleration of the alignment generation algorithm of the HMMER programs by reducing the number of cells of the Dynamic Programming matrices it has to calculate. We also propose a more accurate performance measurement strategy that considers all the execution times while doing protein searches and alignments, while other works only consider hardware execution times and do not include alignment generation times. Using our proposed hardware accelerator and the Divergence Algorithm, we were able to achieve gains up to 182x when compared to the unaccelerated HMMER software running on a general purpose CPU.

Keywords – Bioinformatics; Hidden Markov Models; HMMER; Hardware accelerator; FPGA.

I. INTRODUCTION

Due to the constant increase in the size of bio-sequence databases [1-2], a lot of effort has been put into optimizing and accelerating biosequence analysis algorithms. In the field of protein comparison, the Plan7-HMM Viterbi algorithm and the HMMER software developed by Eddy et al. [3] are among the most used tools for profile Hidden Markov Model (HMM) database search. Several optimization strategies for HMMER have been proposed over the years. MPI-HMMER [4] explores parallel execution in a cluster as well as software optimizations via the Intel-SSE2 instruction set. Other approaches like SledgeHMMER [5] and "HMMER on the Sun Grid" [6] provide web based search interfaces to either an optimized version of HMMER running on a web server or the Sun Grid, respectively. Other approaches such as ClawHMMER [7] and GPU-HMMER [8] implement GPU parallelization of the Viterbi algorithm, while achieving a better cost/benefit relation than the cluster approach.

Studies have shown that most of the processing time of the HMMER software is spent into processing non significant sequences [9]. Therefore, most authors have found useful to apply a first phase filter in order to discard poor scoring sequences prior to full processing. Some works apply heuristics, but the mainstream focuses in the use of FPGA-based accelerators [9-14] as this first phase filter. The filter retrieves the sequence score and, if it is acceptable, full reprocessing of the sequence is done in software.

Our work proposes further acceleration of the algorithm by using the concept of divergence in which full reprocessing of the sequence after the FPGA accelerator phase is not needed, since the alignment only appears in specific parts of both the profile HMM model and the sequence. The proposed accelerator outputs the similarity score and the limits of the area of the Dynamic Programming (DP) matrices that contains the optimal alignment. The software then calculates only that small area of the DP matrices for the Viterbi algorithm and returns the same alignment as the unaccelerated software.

The rest of this work is organized as follows. In Section 2 we introduce the concept of protein families, profile HMMs, the Viterbi algorithm and the HMMER suite. Section 3 shows the related work in FPGA accelerators for the HMMER suite. In Section 4, we present the concept of divergences and explain its two main phases. Section 5 presents the proposed hardware architecture. In Section 6, we explain the VHDL implementation of the accelerator. In Section 7 we show the synthesis and performance results of the accelerator. Finally, in Section 8 we summarize the results of our work and present some future works.

II. PROFILE HMMS AND THE PLAN7-VITERBI ALGORITHM

As explained in [15], databases usually group similar proteins into protein families. These families have structural and/or functional similarities and are used to construct probabilistic models for the family in order to be able to identify and align new members to it. One of the most accepted probabilistic models is based on HMMs. It is called profile HMM because it groups the evolutionary statistics for all the family members "profiling" it. A profile HMM models the common similarities among all the sequences in a protein family as discrete states, each one corresponding to an evolutionary possibility such as amino acid insertions, deletions or matches between them. The traditional profile HMM architecture proposed by Krogh et. al. [15] only consisted of Insert (I), Delete (D) and Match (M) states with transition probabilities between states and emission probabilities for each amino acid element of the sequence. HMMER [3] uses a modified architecture that in addition to the traditional M, I and D states includes flanking states that enable the algorithm to produce global or local alignments, with respect to the model or to the sequence, and also multiple hit alignments. The Plan7 architecture used by HMMER is shown in Figure 1.

978-3-9810801-6-2/DATE10 © 2010 EDAA

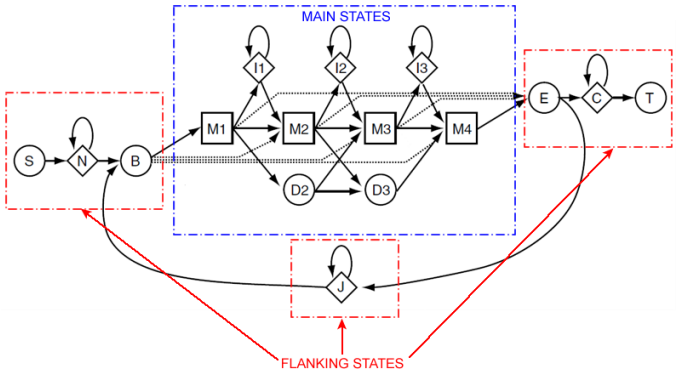

Figure 1. Plan7 architecture used by HMMER [3]

Given a HMM modeling a protein family and a query sequence, HMMER computes the probability that the sequence belongs to the family, as a similarity score, and generates the resulting alignment in case the score is sufficiently good. To do so, it implements a well-known DP algorithm called the Viterbi algorithm [16]. This algorithm calculates a set of DP matrices (corresponding to states M, I, and D) and vectors (corresponding to states N, B, E, C, and J) by means of a set of recurrence equations found in [10,12,13] and shown in Figure 2. Each state has transition probabilities to other states and the M and I states have emission probabilities in order to model the statistical behavior of the protein family. As a result, it finds the best (most probable) alignment and its score for the query sequence with the given model.

$$M(i,0) = I(i,0) = D(i,0) = -\infty \quad \forall\, 1 \le i \le n$$
$$M(0,j) = I(0,j) = D(0,j) = -\infty \quad \forall\, 0 \le j \le k$$

$$M(i,j) = em(M_j, s_i) + max \begin{cases} M(i-1,j-1) + tr(M_{j-1}, M_j) \\ I(i-1,j-1) + tr(I_{j-1}, M_j) \\ D(i-1,j-1) + tr(D_{j-1}, M_j) \\ B(i-1) + tr(B, M_j) \end{cases} \quad \forall\, 1 \le i \le n,$$
$$I(i,j) = em(I_j, s_i) + max \begin{cases} M(i-1,j) + tr(M_j, I_j) \\ I(i-1,j) + tr(I_j, I_j) \end{cases} \quad \forall\, 1 \le j \le k$$
$$D(i,j) = max \begin{cases} M(i,j-1) + tr(M_{j-1}, D_j) \\ D(i,j-1) + tr(D_{j-1}, D_j) \end{cases}$$

$$N(0) = 0$$
$$N(i) = N(i-1) + tr(N,N) \quad \forall\, 1 \le i \le n$$
$$B(0) = tr(N,B)$$
$$B(i) = max \begin{cases} N(i) + tr(N,B) \\ J(i) + tr(J,B) \end{cases} \quad \forall\ 1 \le i \le n$$
$$E(i) = max_{1 \le j \le k} (M(i,j) + tr(M_j, E)) \quad \forall\ 1 \le i \le n$$
$$J(0) = -\infty$$
$$J(i) = max \begin{cases} J(i-1) + tr(J,J) \\ E(i) + tr(E,J) \end{cases} \quad \forall\ 1 \le i \le n$$
$$C(0) = -\infty$$
$$C(i) = max \begin{cases} C(i-1) + tr(C,C) \\ E(i) + tr(E,C) \end{cases} \quad \forall\ 1 \le i \le n$$

$$similarity\_score = C(n) + tr(C,T)$$

Figure 2. Plan7-Viterbi algorithm recurrence equations, for a profile HMM with k nodes and sequence s of length n. In the figure *tr(state1,state2)* corresponds to the transition probability from state1 to state 2 and em($state_j$,S_i) corresponds to the probability of amino acid S_i to appear in $state_j$.

As the routine that implements the Viterbi algorithm in HMMER is the most time consuming of the search and comparison programs, most works [9-14,17] focus in accelerating its execution by proposing a pre-filter phase which only calculates the similarity score for the algorithm. Then, if the similarity score is good, the entire query sequence is reprocessed by the software to produce the alignment. This paper presents first the related work in order to introduce the advances made in such acceleration and then proposes a new technique which not only accelerates the execution by pre-filtering the query sequences, but also reduces the number of calculations required to generate the alignment of the query sequence to the profile HMM.

III. RELATED WORK

In general, FPGA-based accelerators for the Viterbi algorithm are composed of processing elements (PEs), connected together in a systolic array to exploit the algorithm parallelism, by eliminating the J state of the Plan7. As the typical profile HMMs would be too long for its implementation to fit into an FPGA, the entire sequence processing is divided into several passes [9-11,14]. First-In First-Out memories are included inside the FPGA implementation to store the necessary intermediary data between passes. Transition and emission probabilities for all the passes of the HMM are pre-loaded into block memories inside the FPGA to hide model turn around (transition probabilities reloading) when switching between passes. These memory requirements impose restrictions on the maximum number of PEs that can fit into the device, the maximum HMM size and the maximum sequence size.

Benkrid et al. [10] propose an array of 90 PEs, capable of comparing a 1024 element sequence with a 1440 nodes profile HMM. They eliminate the J state dependencies in order to exploit the dynamic programming parallelism and report a maximum performance of 9 GCUPS (Giga Cell Updates per Second). Maddimsetty et al. [9] enhances accuracy by reducing the precision error induced by the elimination of the J state, and proposes a two-pass architecture to detect and correct false negatives. Based on technology assumptions, they report an estimated maximum size of 50 PEs at an estimated clock frequency of 200MHz and a performance of 5 to 20 GCUPS. Jacob et al. [14] divide the PE into 4 pipeline stages estimating a maximum clock frequency of 180MHz and a throughput of 10 GCUPS. Their work also eliminates the J state. The proposed architecture was implemented in a Virtex 2 6000 FPGA and supports up to 68 PEs, a HMM with maximum length of 544 nodes and a maximum sequence size of 1024 amino acids. In Derrien et al. [13], a methodology to implement a pipeline inside the PE is outlined, based on the mathematical representation of the algorithm. Then a design space exploration is made for a Xilinx Spartan 3 4000, with maximum PE clock frequency of 66MHz and a maximum performance of about 1.3 GCUPS. Oliver et al. [11] implement the typical PE array without taking into account the J and the B state when calculating the score. They obtain an array of 72 PEs working at a clock rate of 74MHz, and an estimated performance of 3.95 GCUPS. In [17] a special functional unit is introduced to detect when the J state feedback loop is taken. Then a control unit updates the value for state B and instructs

the PEs to recalculate the inaccurate values. The implementation was made in a Xilinx Virtex 5 110-T FPGA with a maximum of 25 PEs and operating at 130MHz. The reported performance is 3.2GCUPS. No maximum HMM length or pass number is reported in the article.

Walters et al. [12] implement a complete Plan7-Viterbi algorithm (including the J State) in hardware, by exploiting the inherent parallelism in processing different sequences against the same HMM at the same time. They include hardware acceleration into a cluster of computers, in order to further enhance the speedup. The implementation was made in a Xilinx Spartan 3 1500 board with a maximum of 10 PEs per node and a maximum HMM profile length of 256. The maximum clock speed for each PE is 70MHz and the complete system yields a performance of 700 MCUPS per cluster node.

Like all the designs discussed in this section, our design does not calculate the alignment in hardware, providing the score as output. Nevertheless, unlike the previous FPGA proposals, our design also provides information that can be used by the software to significantly reduce the number of cells contained in the DP matrices that need to be recalculated. Therefore, besides the score, our design outputs also the Divergence Algorithm information, that will be used by the software to determine a band in the DP matrices where the actual alignment occurs. In this way, the software reprocessing time can be reduced and better overall speedups can be attained. This work also proposes the use of a more accurate performance measurement that includes not only the time spent calculating the sequence score and divergence, but also takes into account the time spent while reprocessing the sequences of interest found, which gives a clearer idea of the real gain achieved when integrating the accelerator to HMMER.

IV. DIVERGENCE ALGORITHM

The divergence concept was first introduced in [18] for pairwise DNA alignments in order to both enhance processing speed and reduce memory requirements. It reduces the number of DP cells that the software has to calculate by computing limits that instruct the algorithm where the alignment starts and ends. We adapted the divergence concept to the Plan7-Viterbi algorithm. Our Divergence Algorithm (DA) works in two main phases. The first phase is inserted into a simplified version of the Viterbi algorithm which eliminates data dependencies induced by the J state. We calculate the limits of the alignment expressed by its initial and final rows and superior and inferior divergences (IL, FL, SD and ID, respectively). These limits are computed as new DP matrices, by means of a new set of recurrence equations, for the M, I, D, E, and C states of the Viterbi algorithm. Figure 3 shows the main idea behind the divergence concept. The superior and inferior divergences represent how far the alignment departs from the main diagonal, in up and down directions, respectively. The DA calculates the limits of the alignment while it computes the similarity score leaving no space to error and generating the same results as the unbound calculations.

The DA's first phase was thought to be implemented in hardware because its implementation in software would increase the memory requirements and processing time as it introduces new DP matrices. Besides, the alignment limits computation does not create new data dependencies inside the Viterbi algorithm and can be performed in parallel with the similarity score calculation.The second phase takes the output data coming from the first phase. If the sequence's score is significant enough, it produces the alignment by processing only a small portion of the DP matrices, called the alignment region (AR), saving memory space and processing time. It is necessary to initialize only the cells above and to the left of the AR and to calculate only the cells inside it. Figure 3 shows the initialization process and the calculated cells inside the AR. In the next section we propose a hardware implementation of the first phase of DA. This implementation not only speeds-up the execution of the Viterbi algorithm by acting as a pre-filtering phase, but also minimizes the alignment generation time by reducing the number of DP cells that the software has to calculate for significant sequences as it returns the alignment limits for them to the second phase of the DA. The second phase of the DA is implemented as a modification inside HMMER's hmmpfam and hmmsearch programs.

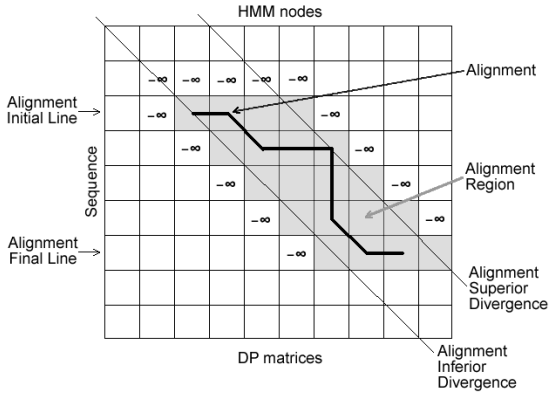

Figure 3. Divergence concept: alignment limits, initialized cells (with -∞) and calculated cells (AR)

V. PROPOSED ARCHITECTURE

In this section we describe the architecture and implementation of a simplified Plan7 Viterbi algorithm, without the J state, and the insertion of the DA's first phase in order to further speed up the whole system when compared to the unaccelerated software. The architecture consists of an array of interconnected PEs, each one calculating one column of the dynamic programming matrices. The system also has RAM memories to store emission and transition probabilities and FIFOs to store intermediate results when dealing with long profile HMMs similar to those analyzed in Section 3. Figure 4 shows the main building blocks of the system.

The PE consists of a score stage which calculates the M, I, D and E scores, and a divergence stage which calculates the alignment limits for the current sequence. A pass controller [14] is included inside each PE, programmed to recognize two especial characters (one for a pass end and another for a sequence end) and increment or clear a pass register. Additional modules are included for the B and C vector calculations as well. Block RAM memories are inserted to store emission probabilities, transition probabilities and intermediate results between passes. A pass controller was included inside each memory in order to control address offsets and to keep track of the current pass.

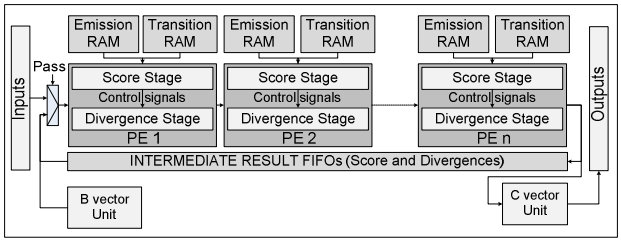

Figure 4. Block diagram of the proposed system: inputs are the protein sequence elements and outputs are the alignment score and limits.

A. SCORE STAGE

This stage calculates the scores for the simplified Viterbi algorithm. Each PE represents an individual HMM node, and calculates the scores as each element of the sequence passes through. The PE's inputs are the scores calculated for the current element in a previous HMM node, and the PE's outputs are the scores for the current sequence element in the current node. Figure 5 shows the score calculation stage of the PE. The signals ctlM, ctlI, ctlD and ctlXE produced in this stage are used in the divergence stage and represent which input was chosen by the maximum operator in the M, I, D and E score calculations, respectively.

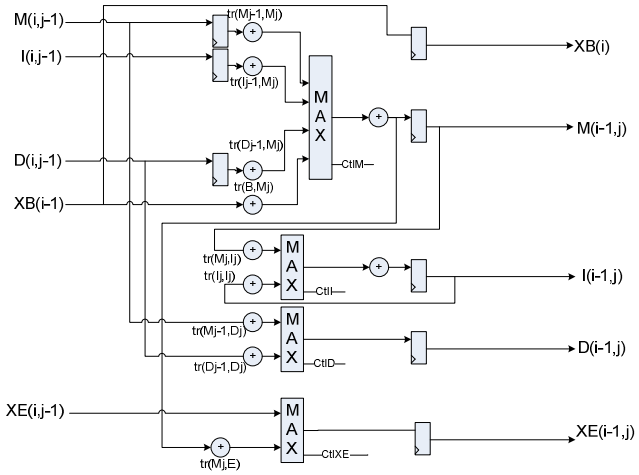

Figure 5. Score calculating stage of the PE

B. DIVERGENCE STAGE

This stage calculates the alignment limits for the current sequence element. The stage inputs are the previous node alignment limits for the current sequence element and the outputs are the calculated alignment limits for the current element. The outputs depend directly on the score stage of the PE and are controlled by the ctlM, ctlI, ctlD and ctlXE signals. The divergence stage also requires the current sequence element index, in order to calculate the alignment limits. Figure 6 shows the DA implementation for the M and E states. Figure 7 show the DA implementation for the I state. The DA implementation for the D state is similar. The Base parameter is the position of the PE in the systolic array and the #PE parameter is the total number of PEs in the current system implementation. These parameters are used to initialize the divergence stage registers according to the current pass and ensure the limits are calculated correctly.

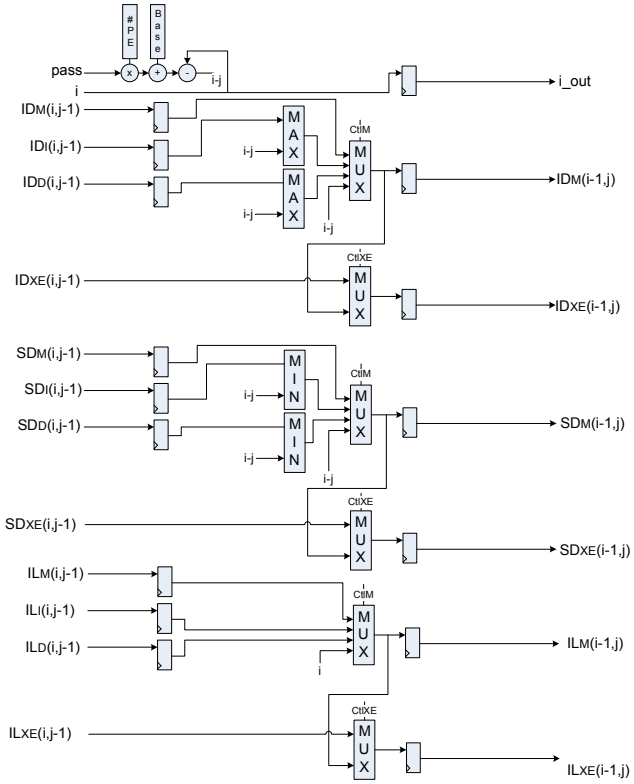

Figure 6. Divergence calculating stage for M and E states

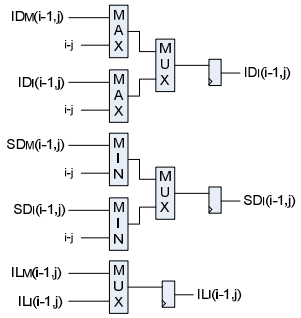

Figure 7. Divergence calculating stage for I state

The C vector module must include the computation of the output alignment limits that will feed the second phase of the DA. The C module is shown in Figure 8.

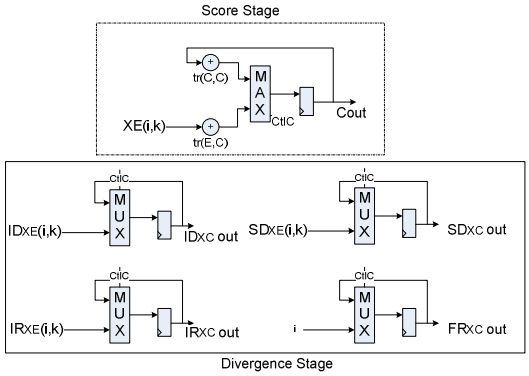

Figure 8. Modified C module to include the DA. The CtlC line is generated by the Score Stage to control the output of the Divergence Stage.

VI. HARDWARE IMPLEMENTATION

The entire system is organized as a linear array of PEs, each one connected to its two immediate neighbors (left and right). Each PE represents one node of the profile HMM and computes the scores for the current sequence element in the current node as well as the alignment limits. Emission and transition probabilities are stored in RAM memories, loaded at the beginning of the execution with the profile HMM we are comparing the query sequence against to. As nine transition probabilities must be available all the time for each PE, the transition memories are connected to a register bank. This register bank has two input ports and nine output ports, and thus can be fully loaded in only 5 clock cycles. Transition memories also have a small controller inside that allows the memory to recognize the end of pass and end of sequence character, and automatically load the register bank with the right transition probabilities for each pass. As intermediate results must be stored to support multiple passes, FIFO memories for each DP matrix are present. The B and C state score units are placed outside from the first and last PEs in order to have an easily modifiable and homogeneous PE.

The complete system was implemented in VHDL and mapped to an Altera Stratix II 180 device. Several configurations were explored to maximize the number of HMM nodes, the number of PEs and the maximum sequence length. In order to do design space exploration, we developed a parametrable VHDL code, in which we can modify the PE word size, the number of PEs of the array and the memories size. For the current implementation, we obtained a maximum frequency of 67MHz after constraining the design time requirements in the Quartus II tool to optimize the synthesis for speed instead of area. Further works will include pipelining the PE to achieve better performance in terms of clock frequency. Table 1 shows the synthesized configurations and their resource utilization.

VII. PERFORMANCE RESULTS

The proposed architecture not only enhances software execution by applying a first phase pre-filter to the HMMER software, but also provides a means to limit data processing to a small portion of the DP matrices. Because of this, the speedup of the solution must be measured taking into account the performance achieved by the hardware pre-filter as well as the saved software processing time by only recalculating the scores inside the alignment region. Execution time is measured separately for hardware by measuring its real throughput rate (including loading time and inter-pass delays), and for software by computing the savings when calculating the scores of the divergence-limited region of the DP matrices. Experimental tests were conducted over all the 10340 profile HMMs for the PFam-A protein database [2]. Searches were made using 4 sets of 2000 randomly sampled protein sequences from the Uniprot.Sprot protein database [1] and only significantly scoring sequences were considered to DA second phase (reprocessing in software). To find out which sequences from the sequence set were significant, we utilized a user defined threshold and relaxed it to include the greatest possible number of sequences. The experiments were done several times to guarantee their repeatability and the stability of the obtained data.

A. HMMER'S PERFORMANCE

To obtain HMMER performance in a typical work environment we used a platform composed of an Intel Centrino Duo processor running at 1.8GHz, 4GB of RAM memory and a 250GB Hard Drive. HMMER was compiled to optimize execution time inside a Kubuntu 8.10 Linux distribution. We also modified the hmmsearch program in order to get the execution times only for the Viterbi algorithm, as our main goal was to accelerate it. From the experiments we obtained a performance of 23.157 MegaCUPS. We also obtained the execution times shown in the fourth and fifth columns of Table 2 for one of the sets of random sequences and 6 PFam-A profile HMMs.

A. HARDWARE'S PERFORMANCE

We formulated the exact equation for the proposed accelerator performance, taking into account all possible delays, including systolic array data filling and consuming, profile HMM probabilities loading into RAM memories and probabilities reloading delays when switching between passes.

TABLE 1. AREA AND SPEED SYNTHESIS RESULTS

#PEs	Max passes	Max HMM nodes	Max sequence size	Combinational ALUs	Dedicated registers	Memory bits	% Logic	Max clock frequency (MHz)
25	25	625	8192	31738	18252	2609152	25	71
50	25	1250	8192	59750	35294	3121152	49	71
75	25	1875	8192	93132	52520	3663152	75	69
85	25	2125	8192	103940	59285	3837952	84	67

TABLE 2. SYSTEM PERFORMANCE RESULTS

#PEs	Sequence set elements (entire set)	HMM nodes	HMMER execution time (sec)	Viterbi execution time (sec)	T_{hw} (GigaCUPS)	t_h (sec)	AR's processing time (sec)	DA system execution time (sec)	Performance gain
85	700218	788	24.40	23.37	5.4206	0.0989	0.0537	0.15	159.89
		10	0.42	0.38	0.6877	0.0099	0.0009	0.01	38.88
		226	6.76	6.72	5.1818	0.0297	0.0149	0.04	151.56
		337	10.26	9.84	5.7953	0.0396	0.0226	0.06	164.95
		2295	81.23	62.25	5.8472	0.2671	0.1787	0.44	182.21
		901	27.41	26.33	5.6345	0.1088	0.0603	0.16	162.09

In order to validate this equation we developed a *testbench* to execute all the test sets. Data transmission is not considered into the formula due to the fact that we will be using at a minimum a PCIe interface [19], whose data transmission rates are well above the maximum required for the system (130MBps). Let m_i be the number of nodes of the current HMM, S_j be the size of the sequence being processed, n be the number of PEs in hardware, and f be the maximum system frequency. Then the throughput T_{hw} of the system (measured in CUPS) and the time $t_{h(i,j)}$ the accelerator takes to process one sequence are fully described by Equations (1) and (2), respectively. In these equations, $25n\lceil m_i/n \rceil$ are the number of cycles spent loading the current HMM into memory, $2n$ are the cycles spent filling and emptying the array of PEs, $(S_j+6)\lceil m_i/n \rceil$ are the cycles spent while processing the current sequence, 3 are the cycles spent loading the special transitions, and $S_j m_i$ are the number of cells that the unaccelerated algorithm has to calculate to process the current sequence with the current HMM. The sixth and seventh columns of Table 2 show the obtained throughputs and the execution times for a hardware accelerator with 85 PEs.

$$T_{hw} = \frac{\sum_{i=1}^{\#HMMs}\sum_{j=1}^{\#Seqs} S_j m_i}{\sum_{i=1}^{\#HMMs}\left[\left(\sum_{j=1}^{\#Seqs}(S_j+6)\left\lceil\frac{m_i}{n}\right\rceil\right)+25n\left\lceil\frac{m_i}{n}\right\rceil+3+2n\right]} * f \quad (1)$$

$$t_{h_{(i,j)}} = \frac{(S_j+6)\left\lceil\frac{m_i}{n}\right\rceil+25n\left\lceil\frac{m_i}{n}\right\rceil+3+2n}{f} \quad (2)$$

A. DIVERGENCE'S SECOND PHASE PERFORMANCE

After simulating the hardware accelerator, we fed the modified HMMER software with the score and alignment limits data for the test HMMs and the significant sequences in the sequence sets. The eighth column of Table 2 shows the execution time obtained when only calculating the cells inside the AR and obtaining the resulting alignment.

B. SYSTEM'S PERFORMANCE

To obtain the system performance we added the execution time of the accelerator and the reprocessing time for the alignment regions of the significant sequences and compared it to the total HMMER processing time for the same set. We obtained performances up to 5.8 GigaCUPs when considering only accelerator throughput, which means a gain of up to 254 times the performance of the unaccelerated HMMER. When including also the reprocessing times, we obtained a gain of up to 182 times the performance of the unaccelerated software, which represents a significant reduction in execution time. The last column of Table 2 shows the obtained performance gains.

VIII. CONCLUSIONS

In this paper we introduced the Divergence Algorithm for sequence-profile searches that enables the implementation of a hardware accelerator for the *hmmsearch* and *hmmpfam* programs of the HMMER suite. We proposed an accelerator architecture which acts as a pre-filtering phase and uses the divergence concept to avoid the full reprocessing of the sequence in software. We also introduced a more accurate performance measurement strategy when evaluating HMMER hardware accelerators, which not only includes the time spent on the pre-filtering phase or the hardware throughput, but also includes reprocessing times for the significant sequences found in the process. We implemented our accelerator in VHDL, obtaining performance gains of up to 182 times the performance of the HMMER software.

For future works we also intend to implement the Divergence Algorithm in a complete version of the Plan7-Viterbi algorithm, and observe the performance gains.

ACKNOWLEDGMENTS

The authors would like to acknowledge the National Council for Technologic and Scientific development (CNPq), the National Microelectronics Program (PNM), the Studies and Projects Financial Fund (FINEP), the Brazilian Millennium Institute (NAMITEC), CAPES and Fundect-MS for funding this work.

REFERENCES

[1] The Universal Protein Resource (UniProt. http://www.uniprot.org. Last access: June 2009.
[2] Sanger's Institute PFAM Protein Sequence Database. http://pfam.sanger.ac.uk/. Last access: May 2009.
[3] HMMER: Biosequence analysis using profile hidden Markov models. http://hmmer.janelia.org, 2006.
[4] Darole, R., Walters, J. P., and Chaudhary, V. (2008) "Improving MPI-HMMER's Scalability With Parallel I/O." Technical Report. Department of Computer Science and Engineering. University of Buffalo.
[5] Chukkapalli, G., Guda, C., and Subramaniam, S. (2004) "SledgeHMMER: A Web Server for Batch searching the PFam Database", Nucleic Acids Research, 32, 542-544.
[6] HMMER on the Sun Grid Project. https://hmmer.dev.java.net/. Last access, July 2009.
[7] Horn, D. R., Houston, M., and Hanrahan, P. (2005) "ClawHMMER: A Streaming HMMer-Search Implementation." In: Conference on Supercomputing, 11-19.
[8] GPU-HMMER. http://mpihmmer.org/userguideGPUHMMER.htm. Last Access, July 2009.
[9] Maddimsetty, R. P., Buhler, J., Chamberlain, R. D., Franklin, M. A., Harris B. (2006) "Accelerator design for protein sequence HMM search." In: Conference on Supercomputing.
[10] Benkrid, K., Velentzas, P., and Kasap, S. (2008) "A High Performance Reconfigurable Core for Motif Searching Using Profile HMM." In: Conference on Adaptive Hardware Systems, 285-292.
[11] Oliver, T., Schmidt, B., Jakop, Y., and Maskell, D. "High Speed Biological Sequence Analysis with Hidden Markov Models on Reconfigurable Platforms." In: IEEE Transactions on Information Technology in Biomedicine, in press.
[12] Walters, J. P., Meng, X., Chaudhary, V., Oliver, T., Yeow, L. Y., Schmidt, B., Nathan, D., and Landman, J. (2007) "MPI-HMMER-Boost: Distributed FPGA Acceleration." In: Journal of VLSI Signal Processing Systems, 48(3), 223-238.
[13] Derrien, S. and Quinton, P. (2007) "Parallelizing HMMER for Hardware Acceleration on FPGAs." In: Conference on Application-specific Systems, Architectures and Processors, 10-17.
[14] Jacob, A. C., Lancaster, J. M., Buhler, J. D., and Chamberlain, R. D. (2007) "Preliminary results in accelerating profile HMM search on FPGAs." In: International Symposium on Parallel and Distributed Processing.
[15] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (2008) Biological Sequence Analysis Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press.
[16] Rabiner, L. R. (1989) "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". In: Proceedings of the IEEE, 77(2), 257-286.
[17] Sun, Y., Li, P., Gu, G., Wen, Y., Liu, Y., and Liu, D. (2009) "HMMer acceleration using systolic array based reconfigurable architecture." In: Symposium on Field Programmable Gate Arrays.
[18] Batista, R. B., Boukerche, A., and Melo, A. C. (2008) "A parallel strategy for biological sequence alignment in restricted memory space." In: Journal of Parallel and Distribituted Computing, 68(4), 548-561.
[19] Dell PCI Express Technology. http://www.dell.com/content/topics/global.aspx/vectors/en/2004_pciexpress. Last access, July 2009.