Accelerating the Viterbi Algorithm for Profile Hidden Markov Models Using Reconfigurable Hardware

Timothy F. Oliver, Bertil Schmidt, Yanto Jakop, and Douglas L. Maskell

School of Computer Engineering, Nanyang Technological University, Singapore 639798
{yanto_jakop, tim.oliver}@pmail.ntu.edu.sg,
{asbschmidt, asdouglas}@ntu.edu.sg

Abstract. Profile Hidden Markov Models (PHMMs) are used as a popular tool in bioinformatics for probabilistic sequence database searching. The search operation consists of computing the Viterbi score for each sequence in the database with respect to a given query PHMM. Because of the rapid growth of biological sequence databases, finding fast solutions is of highest importance to research in this area. Unfortunately, the required scan times of currently available sequential software implementations are very high. In this paper we show how reconfigurable hardware can be used as a computational platform to accelerate this application by two orders of magnitude.

1 Introduction

Profile Hidden Markov Models (PHMMs) have been introduced to molecular biology to statistically describe protein families [6,8]. This statistical description can be used for sensitive and selective protein sequence database searching [5]. The scan operation consists of aligning each sequence in the database to a query PHMM using the well-known Viterbi algorithm [12]. This type of database search is widely used in biological research. Examples include searching for trans-membrane domains [9] and Spin/SSTY homologues [11], to name just a few. However, due to the quadratic time complexity of the Viterbi algorithm this search can take hours or even days depending on the database size, query PHMM length, and hardware used. Therefore, several parallel solutions for PHMM database searching have been developed on coarse-grained architectures such as clusters [13] and grids [2] as well as on fine-grained architectures such as SIMD boards [3,10] and graphics cards [7].

In this paper we show how re-configurable field-programmable gate array (FPGA)-based hardware platforms can be used to accelerate PHMM database scanning by two orders of magnitude. Since there is a large overall FPGA market, this approach has a relatively small price/unit and also facilitates upgrading to FPGAs based on state-of-the-art technology. We present a high-speed implementation on a Virtex II XC2V6000. The implementation is also portable to other FPGAs.

This paper is organised as follows. In Section 2, we introduce the Viterbi algorithm used to align a PHMM to a sequence. The parallel algorithm and its mapping onto a reconfigurable platform are explained in Section 3. The performance is evaluated and compared to previous implementations in Section 4. Section 5 concludes the paper with an outlook to further research topics.

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part I, LNCS 3991, pp. 522 – 529, 2006.

2 Viterbi Algorithm for Profile Hidden Markov Models

Biologists have characterized a growing resource of protein families that share common function and evolutionary ancestry. PHMMs have been identified as a suitable mathematical tool to statistically describe such families and PHMM databases such as PFAM [1] have been created. The general transisiton structure of a PHMM is shown in Figure 1. It consists of a linear sequence of nodes. Each node has a match (*M*), insert (*I*) and delete state (*D*). Between the nodes are transitions with associated probabilities. Each match state and insert state also contains a position-specific table with probabilities for emitting a particular amino acid. Both transition and emission probabilities can be generated from a multiple sequence alignment of a protein family.

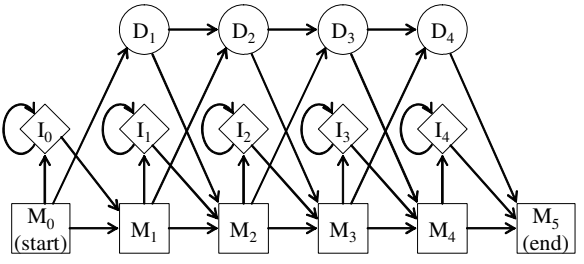

Fig. 1. The transition structure of a PHMM of length 4. Squares represent match states, circles represent delete states and diamonds represent insertions.

A PHMM can be aligned to a given protein sequence to determine the probability that the sequence belongs to the modeled protein family. The most probable path through the PHMM generating a sequence equal to the given sequence determines a similarity score. The well-known Viterbi algorithm can compute this score by dynamic programming (DP). The computation is given by the following recurrence relations.

$$
\begin{aligned}
M(i,j) &= e(M_j, s_i) + \max\begin{cases} M(i-1,j-1) + tr(M_{j-1}, M_j) \\ I(i-1,j-1) + tr(I_{j-1}, M_j) \\ D(i-1,j-1) + tr(D_{j-1}, M_j) \end{cases} \\
I(i,j) &= e(I_j, s_i) + \max\begin{cases} M(i-1,j) + tr(M_j, I_j) \\ I(i-1,j) + tr(I_j, I_j) \end{cases} \\
D(i,j) &= \max\begin{cases} M(i,j-1) + tr(M_{j-1}, D_j) \\ D(i,j-1) + tr(D_{j-1}, D_j) \end{cases}
\end{aligned}
$$

where $tr(state1, state2)$ is the transition cost from *state1* to *state2* and $e(M_j, s_i)$ is the emission cost of amino acid s_i at state M_j. $M(i,j)$ denotes the score of the best path matching subsequence $s_1 \ldots s_i$ to the submodel up to state j, ending with s_i being emitted by state M_j. Similarly $I(i,j)$ is the score of the best path ending in s_i being

emitted by I_j, and, $D(i,j)$ for the best path ending in state D_j. Initialization and termination are given by $M(0,0)=0$ and $M(n+1,m+1)$ for a sequence of length n and a PHMM of length m. By adding jump-in/out costs, null model transitions and null model emission costs the equation can easily be extended to implement Viterbi local scoring (see e.g. [4]).

An alignment example is illustrated in Figures 2, 3, and 4. A PHMM with transition scores is given in Figure 2. The emission scores of the M-states are given in Figure 3. The I-states emission scores in this example are set to zero, i.e. $e(I_j,s_i) = 0$ for all i ,j. The Viterbi DP matrix for computing the global alignment score of the sequence `HEIKQ` and the given PHMM is shown in Figure 4. The three values M, I, D at each position are displayed as ${}_D M^I$. A traceback procedure starting at $M(6,5)$ and ending at $M(0,0)$ (shaded cells in Figure 4) delivers the optimal path through the given PHMM emitting the sequence `HEIKQ`.

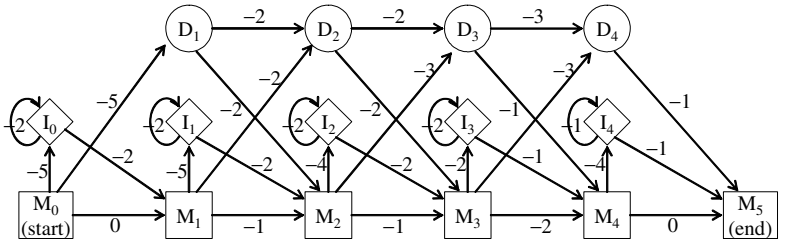

Fig. 2. The given PHMM of length 4 with transition scores

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
M_1	−1	−1	−1	−1	1	−1	3	1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
M_2	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1
M_3	2	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M_4	−1	−1	1	−1	−1	−1	1	−1	2	−1	−1	−1	−1	−1	−1	−1	1	−1	−1	−1

Fig. 3. Emission scores of the M-states for the PHMM in Figure 2

	0	1	2	3	4	5
∅	$*0^{*}$	$_{-5}-\infty^{-\infty}$	$_{-7}-\infty^{-\infty}$	$_{-9}-\infty^{-\infty}$	$_{-12}-\infty^{-\infty}$	
H	$_{-\infty}-\infty^{-5}$	$_{-\infty}3^{-7}$	$_{1}-7^{-\infty}$	$_{-1}-9^{-\infty}$	$_{-4}-9^{-\infty}$	
E	$_{-\infty}-\infty^{-7}$	$_{-\infty}-8^{-2}$	$_{-4}3^{-2}$	$_{0}-1^{-4}$	$_{-3}-3^{-6}$	
I	$_{-\infty}-\infty^{-9}$	$_{-\infty}-6^{-4}$	$_{-6}-4^{-1}$	$_{-3}2^{-3}$	$_{-1}-2^{-5}$	
K	$_{-\infty}-\infty^{-11}$	$_{-\infty}-10^{-6}$	$_{-8}-5^{-3}$	$_{-5}-3^{0}$	$_{-4}2^{-3}$	
Q	$_{-\infty}-\infty^{-13}$	$_{-\infty}-12^{-8}$	$_{-10}-8^{-5}$	$_{-7}-5^{-2}$	$_{-6}-2^{-2}$	
						−2

Fig. 4. The Viterbi DP matrix for computing the global alignment score of the protein sequence `HEIKQ` and the given PHMM

3 Mapping onto a Reconfigurable Platform

The three values of I, D, and M of any cell in the Viterbi DP matrix can only be computed if the values of all cells to the left and above have been computed. But the calculations of the values of diagonally arranged cells parallel to the minor diagonal are independent and can be done simultaneously. Assuming we want to align a subject sequence to a query PHMM on a linear array of processing elements (*PEs*) this parallelization is achieved by mapping the Viterbi calculation as follows: one PE is assigned to each node of the query PHMM. The subject sequence is then shifted through the linear chain of PEs (see Fig. 5). If l_1 is the length of the subject sequence and l_2 is the length of the query PHMM, the comparison is performed in l_1+l_2-1 steps on l_1 PEs, instead of the $l_1 \times l_2$ steps required on a sequential processor.

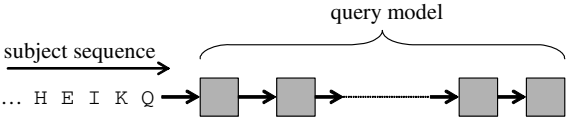

Fig. 5. Systolic sequence comparison on a linear processor array

Figure 6 shows our design for each individual PE. It contains registers to store the following temporary DP matrix values: $M(i-1,j-1)$, $I(i-1,j-1)$, $D(i-1,j-1)$ (upper-left cells) and $M(i-1,j)$, $I(i-1,j)$, $D(i-1,j)$ (upper cells). The values $M(i,j-1)$, $I(i,j-1)$, and $D(i,j-1)$ are stored in the left neighbour. Each PE holds the emission probabilities $e(M_j,s_i)$ and $e(I_j,s_i)$ for the corresponding PHMM node in two look-up-tables (LUTs). The look-ups of $e(M_j,s_i)$ and $e(I_j,s_i)$ and their addition are done in one clock cycle. The results are then passed to the next PE in the array together with the sequence character.

The data width (*dw*) is scaled to the required precision (usually $dw = 24$ bits is sufficient). The LUT depth is scaled to hold the number of emission scores per node (usually 20 for aminoacid sequences). The emission width (*ew*) is scaled to accommodate the dynamic range required by the emission score (usually $ew=16$ is sufficient). The look-up address width (*lw*) is scaled in relation to the LUT depth. All numbers are represented in 2-complement form. Furthermore, the adders in our PE design use saturation arithmetic.

In order to achieve high clock frequencies fast saturation arithmetic is crucial to our design. Therefore, we have added two tag bits to our number representation. These two tags encode the following cases: number (00), +max (01), −max (10), and not-a-number (NaN) (11). The tags of the result of an addition and maximum operation are calculated according to Table 1 and 2. Our representation has the advantage that result tags can be computed in a very simple and efficient way: if any of the operand's tags is set in an addition, a simple bit-wise OR operation suffices. Otherwise, the tags will be set according to the overflow bit of the performed addition.

Table 1. Computation of result tags in the case of an addition

add	number (00)	+max (01)	−max (10)	NaN (11)
number (00)	00[(a)]	01	10	11
+max (01)	01	01	11	11
−max (10)	10	11	10	11
NaN (11)	11	11	11	11

[(a)]except the case that the result produces an overflow, then the result tag is 01 (if MSB is set) or 10 (if MSB is not set)

Table 2. Computation of result tags in the case of a maximum operation

max	number (00)	+max (01)	−max (10)	NaN (11)
number (00)	00	01	00	11
+max (01)	01	01	01	11
−max (10)	00	01	10	11
NaN (11)	11	11	11	11

Assuming, we are aligning the subject sequence $S = s_1 \ldots s_M$ of length M to a query PHMM of length K on a linear processor array of size K using the Viterbi algorithm. As a preprocessing step, the transition and emission probabilities of states M_j, I_j, and D_j are loaded into PE j, $1 \le j \le K$. S is then completely shifted through the array in $M+K-1$ steps as displayed in Figure 5. In iteration step k, $1 \le k \le M+K-1$, the values $M(i,j)$, $I(i,j)$, and $D(i,j)$ for all i, j with $1 \le i \le M$, $1 \le j \le K$ and $k=i+j-1$ are computed in parallel in all PEs within a single clock cycle. For this calculation, PE j, $2 \le j \le K$, receives the values $M(i,j-1)$, $I(i,j-1)$, $D(i,j-1)$ and s_i from its left neighbor $j-1$, while all other required values are stored locally. Thus, it takes $M+K-1$ steps to compute the alignment score with the Viterbi algorithm. However, notice that after the last character of S enters the array, the first character of a new subject sequence can be input for the next iteration step. Thus, all subject sequences of the database can be pipelined with only one-step delay between two different sequences.

So far we have assumed a processor array equal in size of the query model length. In practice, this rarely happens. Since the length of the HMMs may vary, the computation must be partitioned on the fixed size processor array. The query model is usually larger than the processor array. For sake of clarity we firstly assume a query sequence of length K and a processor array of size N where K is a multiple of N, i.e. $K=p \cdot N$ where $p \ge 1$ is an integer. A possible solution is to split the computation into p passes:

The first N nodes of the query model are assigned to the processor array and the corresponding emission and transition scores are loaded. A number of database sequences to be aligned to the query model then cross the array; the M-, I-, and D-value computed in PE N in each iteration step are output. In the next pass the following N nodes of the query model are loaded into the array. The data stored previously is loaded together with the corresponding subject sequences and sent again through the processor array. The process is iterated until the end of the query model is reached.

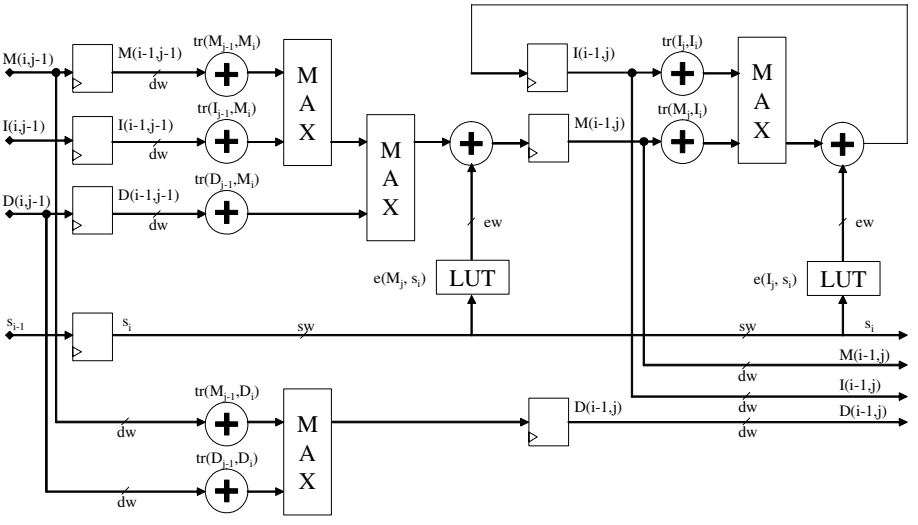

Fig. 6. Schematic diagram of our PE design

The database sequences are passed in from the host one by one through a first-in first-out (FIFO) buffer. The database sequences have been pre-converted to LUT addresses. For query lengths longer than the PE array, the intermediate results are stored in a FIFO. The FIFO depth is sized to hold the longest sequence in the database. The database sequence is also stored in the FIFO. On each consecutive pass an LUT offset is added to address the emission table corresponding to the model of the next iteration step within the PEs. Figure 7 illustrates this solution for 4 PEs.

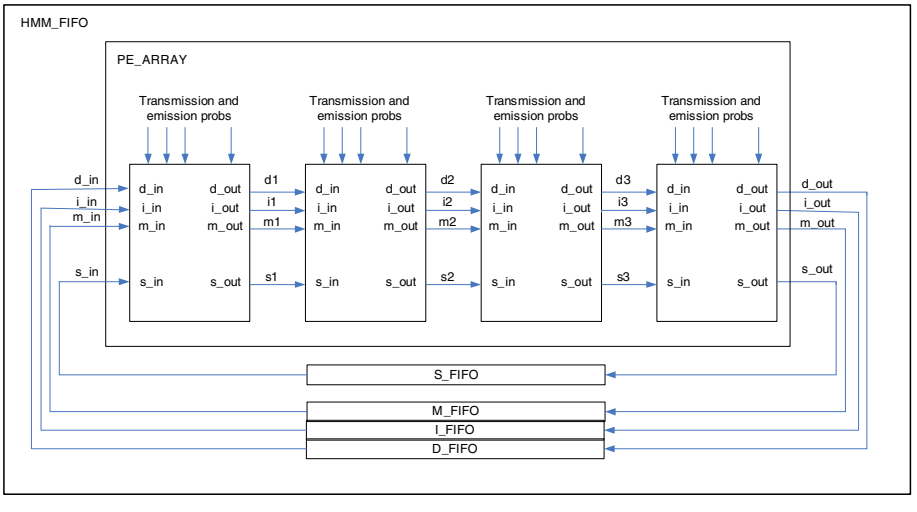

Fig. 7. System implementation

4 Performance Evaluation

We have described our PE design in Verilog and targeted it to the Xilinx Virtex II architecture. We have specified an area constraint for each PE. The linear array is placed in a zigzag pattern as shown in Figure 8. We use on-chip RAM for the partial result FIFO, i.e. one column of block SelectRAM. The host interface also takes up some of the FPGA space in the bottom right-hand corner. Our design has been synthesized with Synplify Pro 7.0. We have used Xilinx ISE 6.3i for mapping, placement and routing.

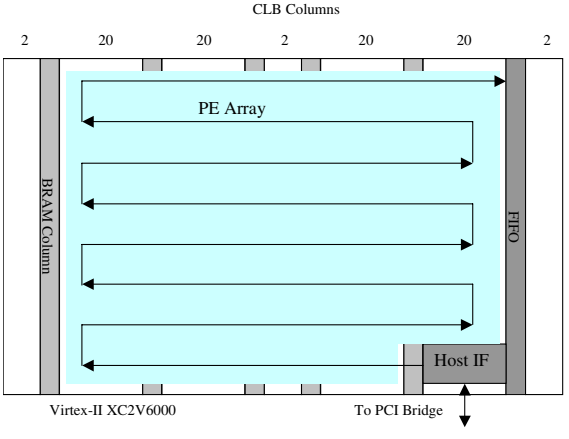

Fig. 8. System Floor plan in the XC2V6000 on the Alpha-Data ADM-XRC-II Board

The size of one PE is 8×14 configurable logic blocks (CLBs). We have implemented a linear array of these PEs. Using all 96×88 CLBs of a Virtex II XC2V6000 on an Alpha-Data ADM-XRC-II PCI board we are able to accommodate (96×88)/(8×14) =12×6 = 72 PEs. The corresponding clock frequency is 74MHz.

A performance measure commonly used in computational biology is *cell updates per second* (CUPS). A CUPS represents the time for a complete computation of one entry of each of the matrices *M*, *D*, and *I*. The CUPS performance of our implementations can be measured by multiplying number of PEs times the clock frequency: 74 MHz × 72 PEs = 5.3 Giga CUPS.

HMMER [5] is a widely used open source implementation of PHMM algorithms with protein databases written in the C programming language. We have measured the performance of the *hmmsearch* algorithm, which is part of the HMMER 2.3.2 package. *hmmsearch* also aligns a query PHMM to all protein sequences of a given database using the Viterbi algorithm as described in Section 2. The performance of hmmsearch for searching the SwissProt database (release 48.6 containing 201,594 sequences) is around 24 Mega CUPS on a Pentium4 3GHz, 1GB RAM, running Linux 2.6.11. Hence, our FPGA implementation achieves a speedup of 220.

5 Conclusions and Future Work

In this paper we have demonstrated that re-configurable hardware platforms provide a cost-effective solution to high performance biological sequence database searching with PHMMs. We have described a partitioning strategy to implement database scans using the Viterbi algorithm on a fixed-size processor array with varying query model lengths. Our PE design and partitioning strategy outperforms available sequential desktop implementations by two orders of magnitude. Our future work includes extending our design to compute local alignments between a sequence and a PHMM and making our implementation available to biologists as a webserver.

References

1. Bateman, A., et al: The PFAM Protein Families Database, *Nucleic Acid Research*, 32: 138-141 (2004)
2. Chukkapalli, G., Guda, C., Subramaniam, S.: SledgeHMMER: a web server for batch searching the pfam database, *Nucleic Acid Research* 32 (July), W542-544 (2004)
3. Di Blas, A. et al: The UCSC Kestrel Parallel Processor, *IEEE Transactions on Parallel and Distributed Systems* 16 (1) 80-92 (2005)
4. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biologcial Sequence Analysis, Probabilistic models of proteins and nucleic acids, *Cambridge University Press* (1998)
5. Eddy, S.R.: HMMER: Profile HMMs for protein sequence analysis, http://hmmer.wustl.edu (2003)
6. Eddy, S.R.: Profile Hidden Markov Models, *Bioinformatics* 14, 755-763 (1998)
7. Horn, D.R., Houston, M., Hanrahan, P.: ClawHMMER: A Streaming HMMer-Search Implementation, *ACM/IEEE Conference on Supercomputing* (2005)
8. Krogh A., Brown, M., Mian, S., Sjolander, K., Hausler, D.: Hidden Markov Models in computational biology: Applications to protein modeling, *Journal of Molecular Biology* 235, 1501-1531 (1994)
9. Narukawa, K., Kadowaki, T.: Transmembrane regions prediction for G-protein-coupled receptors for hidden markov models, *Proc. 15th Int. Conf. on Genome Informatics* (2004)
10. Schmidt, B., Schröder, H.: Massively Parallel Sequence Analysis with Hidden Markov Models, *International Conference on Scientific & Engineering Computation*, World Scientific, Singapore (2002)
11. Staub, E., Mennerich, D., Rosenthal, A.: The Spin/Ssty repeat: a new motif identified in proteins involved in vertebrate development from gamete to embryo, *Genome Biology* 3, 1 (2001)
12. Viterbi, A.J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Transactions on Information Theory* 13, 2, 260-269 (1967)
13. Zhu, W., Niu, Y., Lu, J., Gao, G.R.: Implementing Parallel Hmm-Pfam on the EARTH Multithreaded Architecture, 2nd *IEEE Computer Society Bioinformatics Conference*, 549-550 (2003)