High Performance Database Searching with HMMer on FPGAs

Tim Oliver[1], Leow Yuan Yeow[1], and Bertil Schmidt[2]

[1]*Progeniq Pte Ltd.*
8 Prince George's Park, Singapore 118407
{tim,yuanyeow}@progeniq.com

[2]*UNSW Asia*
1 Kay Siang Road, Singapore 248922
Bertil.Schmidt@unswasia.edu.sg

Abstract

Profile Hidden Markov Models (profile HMMs) are used as a popular bioinformatics tool for sensitive database searching, e.g. a set of not annotated protein sequences is compared to a database of profile HMMs to detect functional similarities. HMMer is a commonly used package for profile HMM-based methods. However, searching large databases with HMMer suffers from long runtimes on traditional computer architectures. These runtime requirements are likely to become even more severe due to the rapid growth in size of both sequence and model databases. In this paper, we present a new reconfigurable architecture to accelerate HMMer database searching. It is described how this leads to significant runtime savings on off-the-shelf field-programmable gate arrays (FPGAs).

1. Introduction

These Profile Hidden Markov Models (profile HMMs) are frequently used in molecular biology to statistically model the primary structure consensus of a family of protein sequences [7, 9, 11]. HMMer [8] is an open-source implementation of profile HMM algorithms, which is widely adopted for large-scale database searching. There are two important search procedures in HMMer called *hmmsearch* and *hmmpfam* (see Table 1). Both search procedures use the comparison of a sequence to a profile HMM as a basic building block. This comparison determines the probability that the given sequence is generated by the given profile HMM using the Viterbi algorithm [15]. Due to the quadratic time complexity of the Viterbi algorithm the search procedure can take hours or even days depending on database size, query size, and hardware used. Examples searches include searching of protein sequences against the Pfam database [5] and searching profile HMMs against the Swissprot/TrEMBL sequence databases [2].

Table 1. HMMer programs for database searching

HMMer procedure	Description	Application
hmmpfam	Search a set of query sequences against an HMM database	Annotate various kinds of domains in the query sequence
hmmsearch	Search a sequence database with a query profile HMM	Find additional homologues of a modeled family

Consequently several parallel solutions for HMMer database searching have been developed on coarse-grained architectures, such as clusters [4, 17], as well as on fine-grained architectures, such as network processors [16] and graphics cards [10]. In this paper we show how reconfigurable field-programmable gate array (FPGA)-based hardware platforms can be used to accelerate HMMer database searching by one to two orders of magnitude. Since there is a large overall FPGA market, this approach has a relatively small price/unit and also facilitates regular upgrading to FPGAs based on state-of-the-art technology. Another FPGA-based solution has recently been presented in [12]. Unfortunately, the design has only been simulated for *hmmsearch*, while our design can be applied for both *hmmsearch* and *hmmpfam*. Furthermore, the performance evaluation is only based on estimation rather than real implementation.

This paper is organized as follows. In Section 2, we introduce the profile HMM architecture used in HMMer. The Viterbi algorithm for comparing a sequence to a profile HMM is described in Section 3. Our reconfigurable hardware design is presented in Section 4 and its performance is evaluated in Section 5. Finally, Section 6 concludes the paper.

1-4244-0910-1/07/$20.00 ©2007 IEEE.

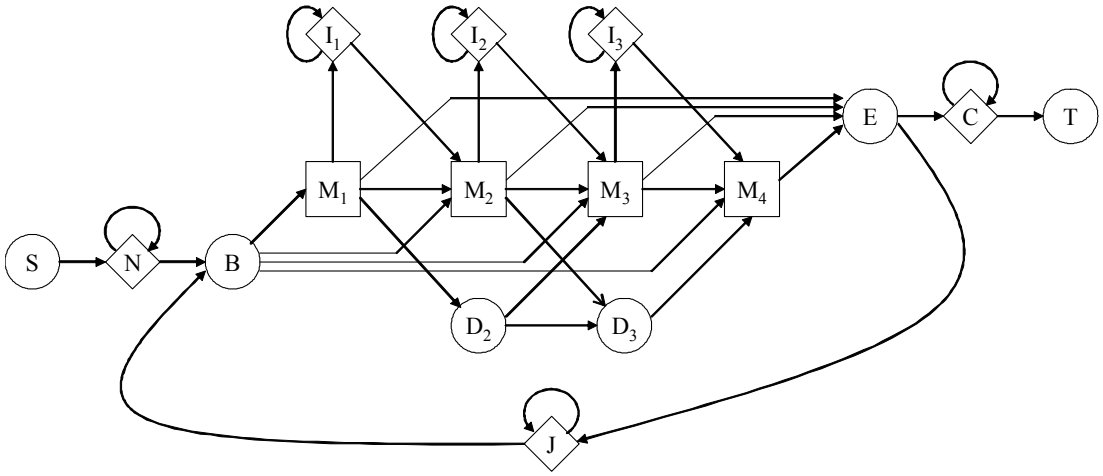

Figure 1. The Plan7 architecture for a profile HMM of length 4.

2. Profile HMMs and Plan7 Architecture

Profile HMMs are statistical models of multiple sequence alignments. They capture position-specific information about how conserved each column of the alignment is, and which residues are likely. Profile HMMs use *position-specific* scores for amino acids and position-specific penalties for opening and extending an insertion or deletion. Traditional pairwise alignment techniques such as the Smith-Waterman [14] algorithm and BLAST [1] use only *position-independent* scoring parameters; i.e. substitution matrix and gap penalties are fixed for all positions. This property of profile HMMs captures important information about the degree of conservation at various positions in the multiple alignments, and the varying degree to which gaps and insertions are permitted. As a consequence, databases of thousands of profile HMMs have been built and applied extensively to whole genome analysis. One such database is Pfam [3]. Pfam covers many common protein domains and families and currently contains 8296 entries (v.20.0). The construction and use of Pfam is tied to the HMMER software package [8].

HMMer uses a profile HMM architecture called Plan7 (see Figure 1). It consists of a linear sequence of nodes. Each node k consists of a match (M_k), insert (I_k), and delete state (D_k). Between the states are arrows called state transitions with associated probabilities. Each *M*-state emits a single residue, with a probability score that is determined by the frequency that residues have been observed in the corresponding column of the multiple sequence alignment. Each *M*-state (and also each *I*-state) therefore carries a vector of 20 probabilities; one for each amino acid. Insertions and deletions are modeled using *I*-states and *D*-states. Transitions are arranged so that at each node, either the *M*-state or the *D*-state is used. *I*-states have a self-transition, allowing one or more inserted residues between to occur between consensus columns.

The linear sequence of nodes is flanked by a begin state (*B*) and an end-state (*E*). Furthermore, there are the special states: *S*, *N*, *C*, *T*, and *J*. They control alignment specific features of the model; e.g. how likely the model is to generate various sorts of global, local or even multi-hit alignments.

3. Viterbi Algorithm

One of the major bioinformatics applications of profile HMMs is database searching. The search operation either consists of searching a profile HMM database with a query protein sequence (*hmmpfam*) or searching a protein sequence database with a query profile HMM (*hmmsearch*). In both cases, the similarity score $sim(H,S)$ of a profile HMM H and a protein sequence S is used to rank all sequences/HMMs in the queried database. The highest ranked sequences/HMMs with corresponding alignments are then displayed to the user as top hits identified by the database search. The similarity score $sim(H,S)$ is usually determined by calculating Viterbi score H of and S. The Viterbi score is defined as the most probable path through H that generates a sequence equal to S. The well-known Viterbi algorithm [15] can compute this score by dynamic programming. The Viterbi dynamic programming (DP) algorithm for Plan7 profile HMMs is given in Algorithm 1.

In Algorithm 1, there are three two-dimensional matrices: M, I, and D. $M(i,j)$ denotes the score of the best path emitting the subsequence $S[1...i]$ of S ending with $S[i]$ being emitted in state M_j. Similarly, $I(i,j)$ is the score of the best path ending with $S[i]$ being emitted in by state I_j, and, $D(i,j)$ for the best path ending in state D_j. Furthermore, there are five one-dimensional matrices: XN, XE, XJ, XB, and XC. $XN(i)$, $XJ(i)$, and $XC(i)$ denotes the

score of the best path emitting S[1...i] ending with S[i] being emitted in special state N, J, and C, respectively. $XE(i)$ and $XB(i)$ denotes the score of the best path emitting S[1...i] ending in E and B, respectively. Finally, the score of the best path emitting the complete sequence S is determined by $XC(n) + tr(C,T)$. The actual path leading to this score can be calculated by a subsequent traceback procedure

Algorithm 1: Viterbi Plan7 DP Algorithm

Input: A profile HMM H of length k in Plan7 format (see Fig. 1) and a protein sequence S of length n. H is described in terms of its transitions ($tr(State1,State2)$ denotes the transition score from *State1* to *State2*) and emissions ($e(State1,s)$ denotes the score of emitting amino acid s at *State1*).

Output: $sim(H,S)$.

Initial Conditions:

$M(0,j) = I(0,j) = D(0,j) = -\infty$ for $j = 1 \ldots k$
$M(i,0) = I(i,0) = D(i,0) = -\infty$ for $i = 1 \ldots n$
$XN(0) = 0$
$XB(0) = tr(\mathrm{N,B})$
$XE(0) = XJ(0) = XC(0) = -\infty$

Recurrence Relation:

for $i = 1 \ldots n$ {
  for $j = 1 \ldots$k {

$$M(i,j) = e(M_j, S[i]) + \max \begin{cases} M(i-1,j-1) + tr(M_{j-1}, M_j) \\ I(i-1,j-1) + tr(I_{j-1}, M_j) \\ D(i-1,j-1) + tr(D_{j-1}, M_j) \\ XB(i-1) + tr(B, M_j) \end{cases}$$

$$I(i,j) = e(I_j, S[i]) + \max \begin{cases} M(i-1,j) + tr(M_j, I_j) \\ I(i-1,j) + tr(I_j, I_j) \end{cases}$$

$$D(i,j) = \max \begin{cases} M(i,j-1) + tr(M_{j-1}, D_j) \\ D(i,j-1) + tr(D_{j-1}, D_j) \end{cases}$$

  }

$$XN(i) = XN(i-1) + tr(N,N)$$
$$XE(i) = \max_{1 \le j \le k} \{M(i,j) + tr(M_j, E)\}$$
$$XJ(i) = \max \begin{cases} XJ(i-1) + tr(J,J) \\ XE(i) + tr(E,J) \end{cases}$$
$$XB(i) = \max \begin{cases} XN(i) + tr(N,B) \\ XJ(i) + tr(J,B) \end{cases}$$
$$XC(i) = \max \begin{cases} XC(i-1) + tr(C,C) \\ XE(i) \end{cases}$$

}

Final score: $sim(H,S) = XC(n) + tr(C,T)$

4. Mapping onto an FPGA Platform

In order to develop a parallel architecture for Algorithm 1, we first analyze to data dependencies in the DP matrices. Figure 2 shows the data dependencies for computing the cell (i,j) in DP matrices M, I, D. It can be seen that the computation of this cell requires the left, upper, and upper-left neighbor. Additionally, it depends on $XB(i-1)$. This values depends on $XJ(i-1)$, which in turn depends on $XE(i-1)$. $XE(i-1)$ then depends on all cells in row $i-1$ in matrix M. Hence, to satisfy all dependencies the two-dimensional matrices M, I, and D must be filled one cell at a time, in row-major order. Hence, computing several DP matrix cells in parallel is not possible for a Plan7 Viterbi score calculation due to the feedback loop induced by the J-state.

Previous solutions to parallelize the Plan7 Viterbi algorithm on FPGAs therefore eliminate the J-state ([12, 13]). This allows the calculations of the values of diagonally arranged cells parallel to the minor diagonal simultaneously; leading to an efficient fine-grained parallel architecture. However, the drawback of this solution is that it cannot find multi-hit alignments; i.e. repeat matches of subsequences of S to subsections of H. This can in turn result in a severe loss of sensitivity for database searching with HMMer. In this paper we present an FPGA solution that uses a full Plan7 model.

Figure 3 shows our design for each individual PE. It contains registers to store the following temporary DP matrix values: $M(i-1,j-1)$, $I(i-1,j-1)$, $D(i-1,j-1)$, $M(i,j-1)$, $I(i,j-1)$, and $D(i,j-1)$. The DP matrix values $M(i,j)$, $I(i,j)$, and $D(i,j)$ are not stored explicitly, instead they are the inputs to the $M(i,j-1)$, $I(i,j-1)$, and $D(i,j-1)$ registers respectively. The PE gets the emission ($e(M_j,s_i)$ and $e(I_j,s_i)$) and transition probabilities $tr(M_{j-1},M_j)$, $tr(I_{j-1},M_j)$, $tr(D_{j-1},M_j)$, $tr(I_j,M_j)$, $tr(I_j,M_j)$, $tr(M_{j-1},D_j)$, $tr(D_{j-1},M_j)$, and $tr(M_j,E)$ from the internal FPGA RAM (Block RAM). The transition probabilities $tr(B,M_j)$, $tr(N,N)$, $tr(E,J)$, $tr(J,J)$, $tr(J,B)$, $tr(N,B)$, $tr(C,C)$, and $tr(C,T)$ are stored in registers in the PE. The PE has a four stage pipeline: Fetch, Compute1, Compute2, and Store. In the Fetch stage transition, emissions and intermediate DP matrix values are read from the Block RAM. All necessary computations are performed in the two compute stages. Finally, results are written to the Block RAM in the store stage. The computation of the special state matrices uses intermediate values for $XE(i)$ which are computes as

$$XE(i,j) = \max\{XE(i,j-1),\ M(i,j) + tr(M_j,E))\}.$$

The updating of XN, XJ, XB, and XC is only performed at the end of the DP matrix row; i.e. if $j=k$.

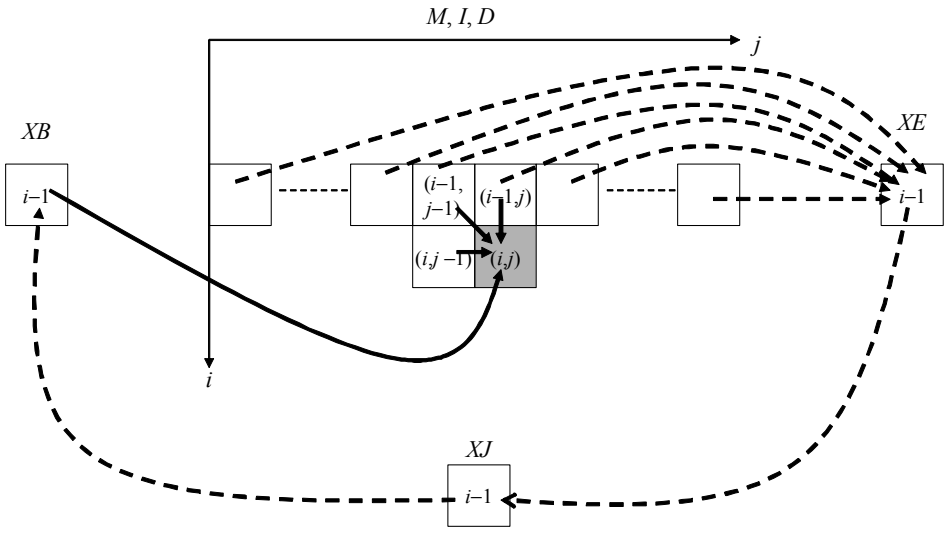

Figure 2: Data dependencies for computing the values *M*(*i*,*j*), *I*(*i*,*j*), and *D*(*i*,*j*) in Algorithm 1. Solid lines are used for direct dependencies and dashed lines for indirect dependencies.

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 2 and 3. 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.

As mentioned above, the Plan7 Viterbi algorithm does not allow computing several cells in parallel. Instead of computing the Viterbi algorithm on one database subject at a time, we align different query/subject pairs independently in separate PEs. Our system design with 4 PEs is shown in Figure 4. Each PE has its own intermediate value storage (IVS). The IVS needs to store one row of previously computed results of the matrices *M*, *I*, and *D*. The PEs are connected to an emission and transition storage. Our design assumes that the same profile HMM has to be aligned to different sequences. All PEs are synchronized to process the same HMM state in every clock cycle. Therefore, the bandwidth requirement to access the transition storage is reduced to a single state. Score collect and score buffer are designed to handle cases where PEs produce results in the same clock cycle. The HMM loader transfers emission and transition values into their respective storage. The sequence loader fetches sequence elements from external memory and forwards them to the emission selection multiplexers. The system is connected to the HMMer software running on the host system via the host interface.

Table 2: 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 3: 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

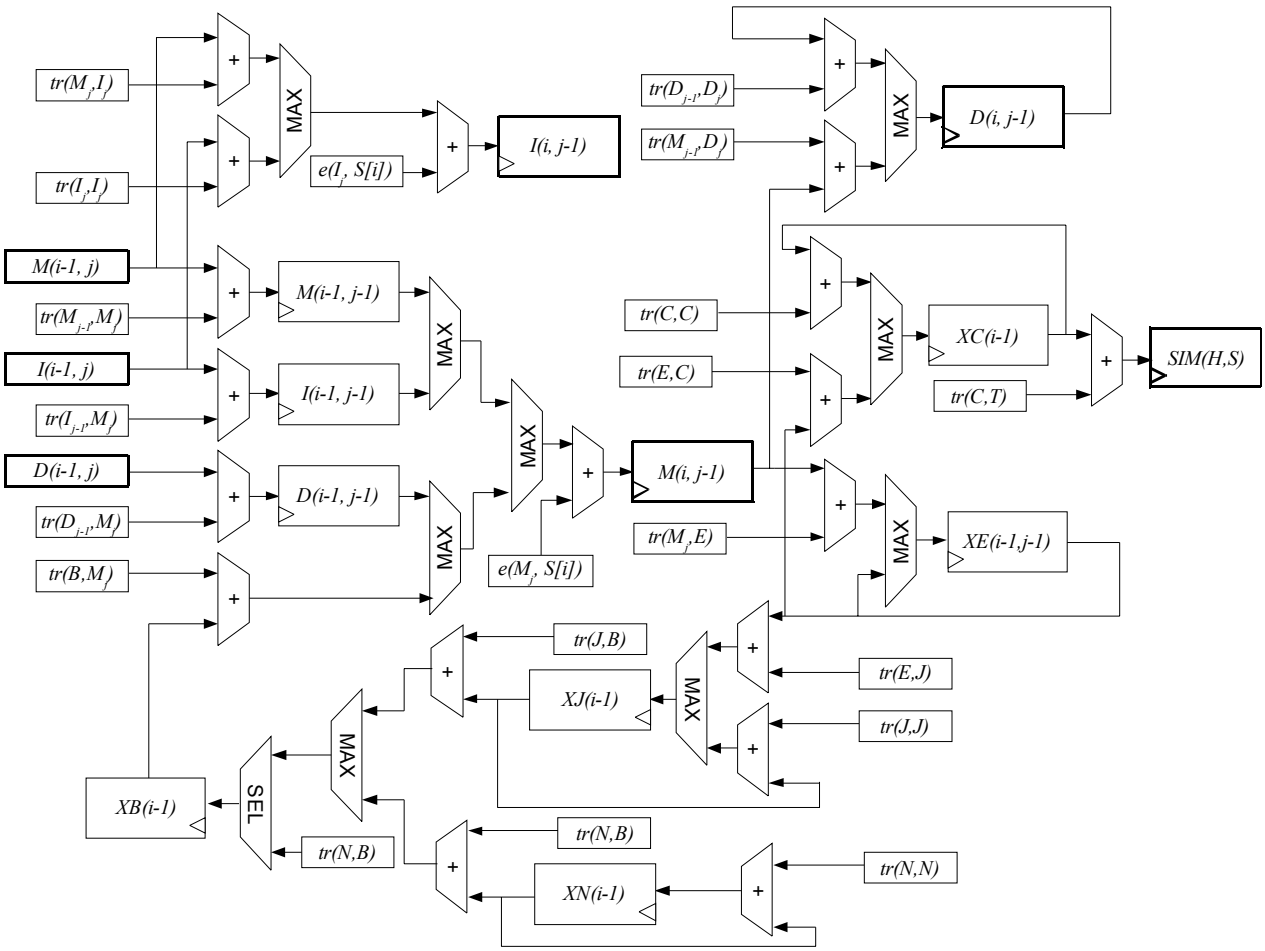

Figure 3: HMM Processing Element (PE) design

5. Performance Evaluation

We have described our PE design in Verilog and targeted it to the low-cost Xilinx Spartan-3 XC3S1500 architecture. We have used Xilinx ISE 8.2i for synthesis, mapping, placement, and routing. The size of one PE is 451 logic slices. The amount of memory required is 50 RAM entries per HMM state, comprising 42 emissions and 8 transitions. Furthermore, there are 3 entries per HMM state for each PE's IVS. Thus, both the size of the HMM and the number of PEs that we are able to support is limited by the number of Block RAM in the target FPGA. Using all 32 Block RAMs on the XC3S1500 we are able to fit 10 PEs and support a profile HMM of up to 256 states.

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 theoretical peak CUPS performance of our implementations can be measured by multiplying number of PEs times the clock frequency: 70 MHz × 10 PEs = 700 Mega CUPS.

HMMer [8] is a widely used open source implementation of profile HMM 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 profile HMM to all protein sequences of a given database using the Viterbi algorithm as described in Sections 2 and 3. We have developed a version of *hmmsearch* that replaces the Viterbi algorithm with our FPGA accelerator. Since the software still performs the traceback for the identified top hits, the overall speedup is reduced from the theoretical peak performance.

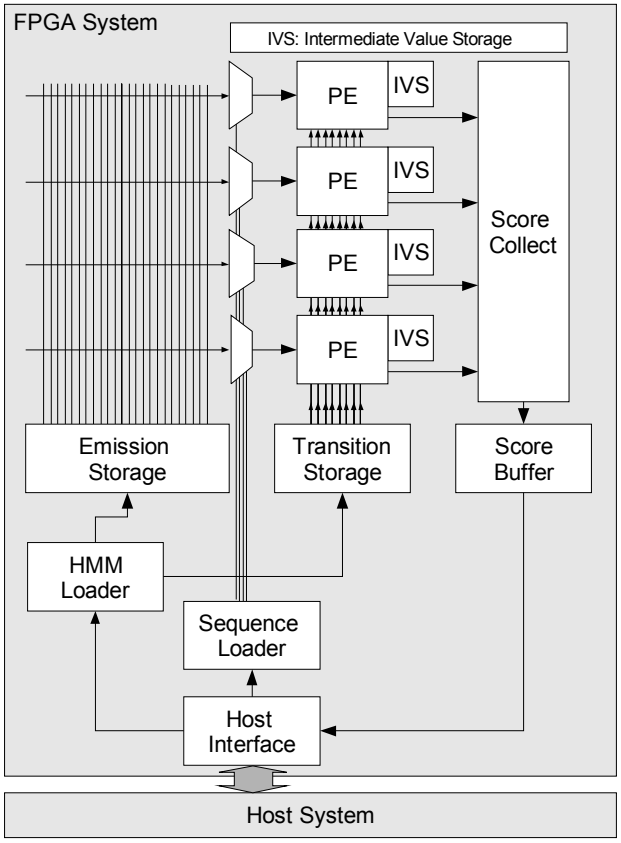

Figure 4: Architecture of the HMM system implemented on FPGA.

For measuring the performance of *hmmsearch* a database of 65535 amino-acid sequences was searched using a profile HMM with 236 states. The total number of cells processed in this problem is 5,471,693,984. We used a Pentium 4 3GHz with 1GB RAM running Linux 2.6.11 and a single processor on the Power Mac Dual G5 2.5GHz with 512MB of RAM running Mac OS 10.4 (Tiger). The results of the performance comparison are shown in Table 4. The HMMer 2.3.4 package is able to take advantage of the Altivec extensions available in the G5 processor, for a fair comparison we have included this optimization in Table 4.

The difference between these previous SIMD approaches (such Kestrel [6]) and ours is that FPGAs allow easy upgrading. This, and our previous work, shows the portability inherent in FPGA technology. Our previous HMM accelerator [13] was targeted at the Virtex-II architecture but did not support the full Plan-7 Viterbi algorithm. Our new architecture to support the full Plan-7 Viterbi algorithm could easily be ported to the higher performance Virtex-4 architecture or the newer Virtex-5 architecture (65nm) for a higher computing performance (due to larger number of logic slices and more Block RAM). However, we have opted for the Spartan-3 architecture due to its better price/performance ratio. In addition, available Spartan-3 boards have small form factors and low power consumption, making our design applicable to mobile applications. Furthermore, the low cost of the Spartan-3 devices makes it possible to put an FPGA accelerator in several nodes within a cluster for an even more performance.

Table 4: Performance comparison of different *hmmsearch* implementations

Machine	*Optimization*	*Time taken (s)*	*Processing rate (MCUPS)*	*Speedup*
Pentium4	None	246.62	22.19	1
Apple G5	None	220.55	24.91	1.1
Apple G5	Altivec	61.79	88.54	4.0
Pentium4	FPGA	9.558	572.35	25.8

6. Conclusion

In this paper we have demonstrated that re-configurable hardware platforms provide a cost-effective solution to high performance biological sequence database searching with HMMer. We have described a PE design to implement database scanning using the full Plan-7 Viterbi algorithm. Our strategy outperforms available sequential desktop implementations for *hmmsearch* by one order of magnitude. Our design can also be applied to *hmmpfam* using a set of protein sequences as queries to an HMM database. The corresponding performance evaluation is currently ongoing. Our future work also includes making our implementation available to biologists as a web server.

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W, Lipman, D.J.: Basic local alignment search tool, *J. Mol. Biol.*, 215 (1990) 403-410.

[2] Anand, B., Verma, S.K., Prakash, B.: Structural stabilization of GTP-binding domains in circularly permuted GTPases: Implications for RNA binding, *Nucleic Acids Research* 34 (8) 2196–2205 (2006)

[3] Bateman, A., et al: The PFAM Protein Families Database, *Nucleic Acid Research*, 32: 138-141 (2004)

[4] Chukkapalli, G., Guda, C., Subramaniam, S.: SledgeHMMER: a web server for batch searching the pfam database, *Nucleic Acid Research* 32 (July), W542-544 (2004)

[5] Guda, C., Fahy, E., Subramaniam, S.: MITOPRED: A genome-scale method for prediction of nucleus-encoded mitochondial proteins, *Bioinformatics* 20 (11) 1785-94 (2004)

[6] Di Blas, A. et al: The UCSC Kestrel Parallel Processor, *IEEE Transactions on Parallel and Distributed Systems* 16 (1) 80-92 (2005)

[7] Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biologcial Sequence Analysis, Probabilistic models of proteins and nucleic acids, *Cambridge University Press* (1998)

[8] Eddy, S.R.: HMMER: Profile HMMs for protein sequence analysis, http://hmmer.wustl.edu (2004)

[9] Eddy, S.R.: Profile Hidden Markov Models, *Bioinformatics* 14, 755-763 (1998)

[10] Horn, D.R., Houston, M., Hanrahan, P.: ClawHMMER: A Streaming HMMer-Search Implementation, *ACM/IEEE Conference on Supercomputing* (2005)

[11] 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)

[12] Maddimsetty, R.P., Buhler, J., Chamberlain, R., Franklin, M., Harris, B.: Accelerator design for protein sequence HMM search, *Proc. 20th ACM International Conference on Supercomputing (ICS06)* 288-296, 2006

[13] Oliver, T.F., Schmidt, B., Yanto, J. and Maskell, D.L., "Accelerating the Viterbi Algorithm for Profile Hidden Markov Models using Reconfigurable Hardware", *Lecture Notes in Computer Science*, Springer, Vol. 3991, 522-529 (2006)

[14] Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences, *J. Mol. Biol.* 147 (1981) 195-197.

[15] Viterbi, A.J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Transactions on Information Theory* 13, 2, 260-269 (1967)

[16] Wun, B., Buhler, J., Crowley, P.: Exploiting coarse-grained parallelism to accelerate protein motif finding with a network processor" *Proc. 14th Int. Conf. on Parallel Architectures and Compilation Techniques (PACT 2005)*, 173-184 (2005)

[17] 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)