Preliminary results in accelerating profile HMM search on FPGAs

Arpith Jacob, Joseph Lancaster,
Jeremy Buhler, and Roger D. Chamberlain

Arpith Jacob, Joseph Lancaster, Jeremy Buhler, and Roger D. Chamberlain, “Preliminary results in accelerating profile HMM search on FPGAs,” in *Proc. of 6th IEEE International Workshop on High Performance Computational Biology,* March 2007.

Dept. of Computer Science and Engineering
Washington University
Campus Box 1045
One Brookings Dr.
St. Louis, MO 63130

BECS Technology, Inc.
9487 Dielman Rock Island Industrial Dr.
St. Louis, MO 63132
R.D. Chamberlain is a principle in BECS Technology, Inc.

Preliminary results in accelerating profile HMM search on FPGAs

Arpith C. Jacob, Joseph M. Lancaster, Jeremy D. Buhler, and Roger D. Chamberlain

Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, Missouri 63130–4899
{jarpith, jmlancas, jbuhler, roger}@cse.wustl.edu

Abstract

Comparison between biosequences and probabilistic models is an increasingly important part of modern DNA and protein sequence analysis. The large and growing number of such models in today's databases demands computational approaches to searching these databases faster, while maintaining high sensitivity to biologically meaningful similarities. This work[1] *describes an FPGA-based accelerator for comparing proteins to Hidden Markov Models of the type used to represent protein motifs in the popular HMMER motif finder. Our engine combines a systolic array design with enhancements to pipeline the complex Viterbi calculation that forms the core of the comparison, and to support coarse-grained parallelism and streaming of multiple sequences within one FPGA. Performance estimates based on a functioning VHDL realisation of our design show a* $190\times$ *speedup over the same computation in optimised software on a modern general-purpose CPU.*

1. Introduction

Biosequence comparison reveals the biological functions of a DNA or protein sequence by discovering similarities to other sequences of known function. As the size and diversity of biosequence databases has grown, it has become common to group a family of related sequences and summarise their shared features using a probabilistic model, such as a Hidden Markov Model (HMM) [8]. Comparing new sequences to the model, rather than to its constituent sequences, provides a more sensitive and more nuanced measure of how well a new sequence matches others in the family.

[1]This research was supported by NIH/NGHRI grant 1 R42 HG003225-01 and NSF grants CCF-0427794 and DBI-0237902.

The exponential growth of biosequence databases over the last two decades [1] has led to ever larger databases of sequence family models. As of this writing, the PFAM model database [3] contains over $195,000$ such models, with a doubling time of roughly five years. Comparing bacterial and human proteomes against PFAM require 1 to 500 days of CPU time. To keep up with growth in model databases, biologists have sought ways to search them more efficiently, including dividing the database over a large computing cluster or devising special hardware [2, 6, 16] and software [12, 15] to accelerate the search.

Field Programmable Gate Arrays (FPGAs) are one form of specialised hardware that has proven especially useful as a platform for biological applications. FPGAs are hardware devices that can be programmed at the basic logic level to directly implement the core, computationally intensive blocks of an algorithm. These devices exploit the parallelism of their target applications to achieve order-of-magnitude speedups relative to a general-purpose CPU. FPGA-based accelerators require less power and maintenance than a large computing cluster, and their ability to be reprogrammed on the fly allows them to implement multiple applications, unlike an ASIC. Although they require more hardware expertise to develop, advances in tools have made them more accessible. This accessibility, coupled with recent rapid growth in the sizes of FPGA parts, have enabled acceleration of increasingly complex and resource-hungry applications.

In the domain of computational biology, FPGAs have traditionally been used to accelerate the highly regular structure of dynamic programming algorithms for sequence comparison, reporting several orders of magnitude speedup [17]. More recently, other bioinformatic tasks such as the pattern-matching heuristics of BLAST [7] have also been accelerated. By combining FPGAs' traditional strength in dynamic programming with new implementation techniques and high I/O bandwidth into the accelerator, it is possible to achieve large speedups for complex tasks

Protein	Motif Sequence
RA25_SCHPO	RENSVYL**AKL**A**EQAERY**EEMVEN**M**KK**V**ACSND..K**L**SV**E**
BMH1_YEAST	REDSVYL**AKL**A**EQAERY**EEMVEN**M**KT**V**ASSGQ..E**L**SV**E**
1434_LYCES	REENVYL**AKL**A**EQAERY**EEMIEF**M**EK**V**AKTADVEE**L**TV**E**
143T_HUMAN	KTELIQK**AKL**A**EQAERY**DDMATC**M**KA**V**TEQGA..E**L**SN**E**
1433_XENLA	.......**AKL**S**EQAERY**DDMAAS**M**KA**V**TELGA..E**L**SN**E**

Figure 1. A protein motif in five different proteins. Bold-faced residues are invariant across all instances; a dot indicates a missing residue in an instance.

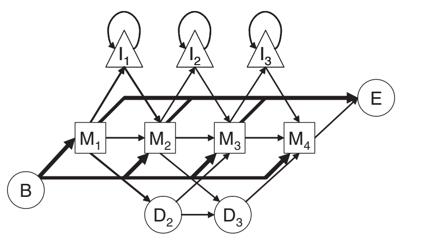

Figure 2. Simplified Plan7 HMM structure for a motif of length $m = 4$.

such as alignment of sequences to probabilistic sequence models.

In this work, we present preliminary results on a hardware design to accelerate the Viterbi decoding algorithm [13] at the core of the popular HMMER software package [5], which compares protein sequences against conserved sequence patterns, or *motifs*, modeled by HMMs. Building on previous work in this area [10], our accelerator acts as a filter that eliminates the vast majority of potential matches between proteins and HMMs, leaving only a small fraction to be validated in software. We describe a systolic array architecture and methods to handle the large amounts of data involved in modeling the motifs. We introduce a novel pipelining technique to exploit coarse-grained parallelism in individual processing elements, while satisfying cell data dependencies. Our design, which has been implemented in the VHDL language, synthesised, and simulated, provides an estimated speedup for the Viterbi computation of 190x over a single modern general-purpose CPU.

2. Related Work

There have been limited algorithmic improvements to the HMMER computation. Heuristic approaches using a seeding strategy such as HMMERHEAD [12] achieve modest improvements in speed. Recently, a pattern matching based prefilter stage [15] has been employed to realise up to 8× speedups without a loss in sensitivity. These approaches typically operate as prefilters, reducing the data stream into the full Viterbi computation of HMMER. While we have choosen to accelerate the Viterbi computation in hardware, heuristic approaches complement our work, and can be used as a hardware prefilter stage prior to full Viterbi.

Most efforts in accelerating HMMER have concentrated on exploiting coarse-grained parallelism, for example using workstation clusters. Clusters typically have high acquisition, maintenance, and energy costs as compared to single-node solutions.

JackHMMer [16] is a Network Processor based implementation that distributes the search process over a number of *Microengines*. On an Intel IXP 2850, they achieve a speedup of approximately 1.8× over a Pentium 4 based workstation. *ClawHMMER* [6] uses commodity graphics processors to achieve speedups of up to 25× over a Pentium 4 Xeon workstation.

Single Instruction Multiple Data (SIMD) instructions on general-purpose processors have been used to exploit fine-grained parallelism. Lindhal reports an implementation on the Altivec G5 [9] running up to 8× faster than an integer implementation. However, the same technique has failed to produce considerable speedup on x86 processors due to the lack of critical SIMD instructions.

A commercial FPGA implementation of the HMMER computation is available from TimeLogic [2]. Using several PCI FPGA cards and a multi-processor host system, they claim a speedup of 2600× over a Pentium III processor. However, no published information exists to perform a fair comparison with our architecture. Oliver et al. [11] report an architecture for accelerating the simple Plan 7 HMMER computation. They report a throughput of 5.3 GCUPS, approximately half our throughput on the same FPGA device.

3. Background: Detecting Motifs via Hidden Markov Models

In this section, we briefly review the structure of the HMMs used in protein motif finding, including key simplifications assumed by our implementation. We also review the core Viterbi algorithm used to determine whether the motif described by an HMM occurs in a given protein sequence. We assume that the reader has basic familiarity with HMMs; a more detailed description of these models and their use in similarity search may be found in [4].

3.1. HMMs as a Model of Protein Motifs

The core problem addressed in this work is recognition of a *motif*, which is a conserved pattern of amino acids, or *residues*, that occurs in multiple protein sequences. Figure 1 shows a motif as it appears in five different proteins. A motif consists of a series of conserved positions, each of which has one or more characteristic residues that occur in that position with high frequency. Not all positions need be present in every instance of a motif, and positions may

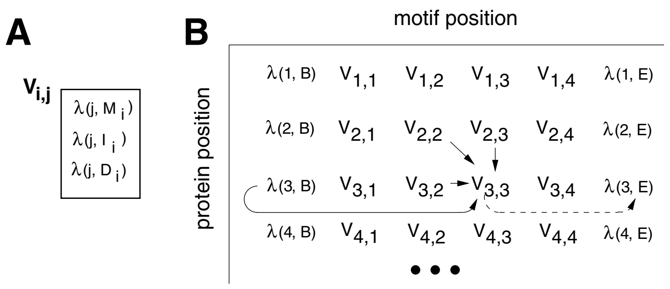
A
B
motif position
protein position
$V_{i,j}$
λ(j, M i)
λ(j, I i)
λ(j, D i)
λ(1, B) V1,1 V1,2 V1,3 V1,4 λ(1, E)
λ(2, B) V2,1 V2,2 V2,3 V2,4 λ(2, E)
λ(3, B) V3,1 V3,2 V3,3 V3,4 λ(3, E)
λ(4, B) V4,1 V4,2 V4,3 V4,4 λ(4, E)

Figure 3. Organisation of the Plan7 Viterbi recurrence as a dynamic programming matrix. (A) values grouped into one matrix cell $V_{i,j}$; (B) data dependencies between cells.

occasionally be separated by non-conserved "background" residues.

In this work, the variations observed across the instances of a motif, are modeled by HMMs structured according to a simplified version of the "Plan7" schema, as defined by the HMMER motif-finding software [5]. Figure 2 shows the structure of one such HMM. A motif model of length m ($m = 4$ in the figure) contains m "match states" $M_1 \ldots M_m$, where M_i emits the residue for the motif's i^{th} conserved position. A parallel sequence of non-emitting "deletion states" states $D_2 \ldots D_{m-1}$ allows any substring of motif positions to be skipped, while another parallel set of "insertion states" $I_1 \ldots I_{m-1}$ can emit background residues between any two motif positions. The non-emitting states B and E act as the model's initial and final states; each is connected directly to every other match state, allowing emission of partial motifs starting or ending at any position.

Any path through a motif's HMM emits a sequence of residues comprising one instance. To distinguish between sequences that are more or less likely to be observed, an emitted sequence is associated with a *score*, derived as follows. Each pair of HMM states (q_i, q_j) connected by a transition is associated with a *transition score* $\tau(q_j \mid q_i)$, while each emitting state q_i is associated with a set of *emission scores* $\epsilon(a \mid q_i)$ for each possible residue a[2]. Suppose a sequence s is emitted along a path of states $q_0 \ldots q_n$ (where $q_0 = B$ and $q_n = E$), such that residue $s[j]$ is emitted by state q_{i_j}. Then the total score of this sequence given this path is

$$\sum_{i=2}^{n} \tau(q_i \mid q_{i-1}) + \sum_{j} \epsilon(s[j] \mid q_{i_j}).$$

Higher-scoring paths correspond to motif instances that are more likely to appear in real proteins.

[2]A probabilistic interpretation of these scores is given in [4]; in this work, we treat them as given for any motif model.

3.2. Recognising Motifs with the Viterbi Algorithm

To determine whether a protein sequence s contains a motif that matches an HMM $\mathcal{M}$, a motif-finding tool finds the highest-scoring path through $\mathcal{M}$ that emits s. If the score $\mathrm{L}(s, \mathcal{M})$ of this path exceeds some user-defined threshold ρ, then $\mathcal{M}$ is said to *hit* s, and the path indicates which residue of the protein (if any) correspond to each position of the motif.

The Viterbi algorithm [13] calculates $\mathrm{L}(s, \mathcal{M})$ by the following dynamic programming recurrence. Let $\lambda(j, q)$ be the highest score for any path through $\mathcal{M}$ from initial state $q_0 = B$ to a later state q that emits the string of residues $s[1..j]$. Then for any j and q,

$$\lambda(j, q) = \begin{cases} P_e(j, q) & \text{if } q \text{ is emitting} \\ P_n(j, q) & \text{otherwise.} \end{cases} \tag{1}$$

where

$$\begin{aligned} P_e(j, q) &= \max_{q' \in \mathcal{M}} \left[\lambda(j-1, q') + \tau(q \mid q') + \epsilon(s[j] \mid q)\right] \\ P_n(j, q) &= \max_{q' \in \mathcal{M}} \left[\lambda(j, q') + \tau(q \mid q')\right]. \end{aligned}$$

Any path through $\mathcal{M}$ must end at the unique end state E of the model after emitting all of s; hence, $\mathrm{L}(s, \mathcal{M}) = \lambda(|s|, E)$ is the score of the best path through $\mathcal{M}$ emitting s.

The model schema of Figure 2 exhibits two important simplifications compared to the full Plan7 schema used by HMMER. Firstly, the full Plan7 schema contains a *feedback loop* from state E to state B. This loop enables the Viterbi algorithm to recognise multiple copies of a motif in a single protein, yielding a total score which is effectively the sum of scores for all motif copies. In contrast, our simplified schema lacks this feedback loop; hence, the Viterbi algorithm with our models will match only the single best copy of the motif in a protein.

While full Plan7 models are better able to detect weak motifs that appear in multiple copies per protein, the data dependencies induced by the feedback loop greatly impede parallelisation of the Viterbi algorithm. We previously explored this issue in [10] and showed that the sensitivity of full Plan7 models can be matched by our simplified models, at a manageable increase in computational cost, by reducing the hit threshold ρ for detecting a motif in a protein.

The second simplification of our schema versus full Plan7 is that we do not explicitly model the background sequences flanking an instance of the motif. In HMMER, these flanking sequences contribute an amount proportional to their lengths to the total Viterbi score of a protein s against a model $\mathcal{M}$, independent of their amino acid content. While we do not currently include this contribution in our scoring, it could easily be computed given the starting and ending points of a motif instance and the total protein length.

3.3. High-Performance Viterbi for Motifs

The special structure of motif HMMs suggests a fruitful approach to accelerating the Viterbi algorithm. We organise the values to be computed into a dynamic programming matrix V. Rows of the matrix correspond to amino acid positions i in a protein, while columns correspond to positions j in a motif. Figure 3 illustrates this matrix for the example model of Figure 2.

For a motif of length m and a sequence s of length ℓ, the matrix V contains $m\ell$ *cells*. As shown in Figure 3A, each cell $V_{i,j}$ holds the values $\lambda(i, M_j)$, $\lambda(i, I_j)$ and $\lambda(i, D_j)$ computed by the Viterbi algorithm. The structure of the Plan7 model and Equation (1) together imply that the values in $V_{i,j}$ can be computed given the value $\lambda(i-1, B)$ and the three cells $V_{i-1,j-1}$, $V_{i,j-1}$, and $V_{i-1,j}$. The global dependencies can be converted to a local dependency from the left cell. These four dependencies are shown for cell $V_{3,3}$ in Figure 3B by solid arrows flowing into it.

For local alignment, the values $\lambda(i, B)$ of a single motif instance to a protein are zero for all i. The remaining data dependencies are all downwards and to the right. This highly localised dependency structure, analogous to that of the well-known Smith-Waterman algorithm [14] for pairwise sequence alignment, permits simultaneous computation of an entire band of cells at once, as follows. The first step of computation computes $V_{1,1}$; the second computes both $V_{1,2}$ and $V_{2,1}$; the third computes all of $V_{1,3}$, $V_{2,2}$, and $V_{3,1}$, and so forth, with the dth step computing $V_{i,d-i+1}$ for all rows i. The band of cells computed in each step is called an *anti-diagonal* of the matrix V.

In addition to the local dependencies described above, the output of each cell $V_{i,j}$ also contributes to the score $\lambda(i, E)$, as shown by the dashed line in Figure 3B. This incremental contribution can be computed as part of the work for a cell and so adds only constant overhead per cell. The best score obtained by local alignment is $\max_i \lambda(i, E)$.

We emphasise that the HMMER accelerator described in this paper computes the score of a protein sequence against a model. It does not however retrieve the most likely motif. That is, the HMMER accelerator acts as a filter to the input stream. The predicted motif can be retrieved by maintaining path information during the Viterbi algorithm, run on an attached general-purpose processor (typically 1% of the execution time). In practice, only a small fraction of input sequences are passed to this stage, and so its execution time is dominated by that of the FPGA hardware.

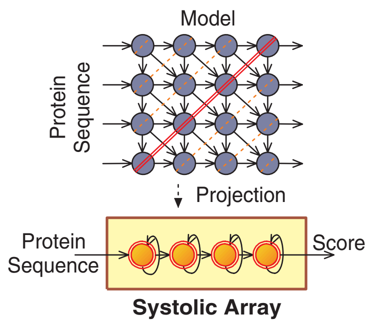

Figure 4. Projection of the Plan7 Viterbi matrix to a systolic PE array. The array is shown computing cell values in the anti-diagonal.

4. FPGAs as a Platform for High Throughput Sequence Analysis

Field Programmable Gate Arrays (FPGAs) are special-purpose hardware programmable devices used to accelerate computationally bound applications. An FPGA consists of programmable logic blocks called lookup tables (*LUTs*) with interconnects between them. LUTs can be programmed to function as basic logic gates, thereby realising digital circuits. A number of LUTs (typically two) are grouped into a *slice* and associated with a clocked flip-flop. Limited on-chip memory is also available as *block RAMs*. FPGA designs are clocked at a much lower frequency than ASICs, but their advantage is in the parallel computing and memory resources available. Such designs typically exploit fine-grained parallelism in algorithms to realise several orders of magnitude speed-up over general-purpose processors.

The basic design of an FPGA accelerator for the Viterbi algorithm is similar to one for Smith-Waterman. However, the more resource intensive computation in each cell of the matrix, coupled with the large number of parameters in a motif model make it a challenging task to implement efficiently. FPGA devices have only recently become large enough to accommodate such designs.

5. System Architecture

In the following sections we describe a hardware architecture to implement the Viterbi computation of the HMMER model. We describe a processing element, the basic computational block of the Viterbi algorithm. We then detail the data flow design of the profile parameters. Finally, we describe the overall HMMER engine.

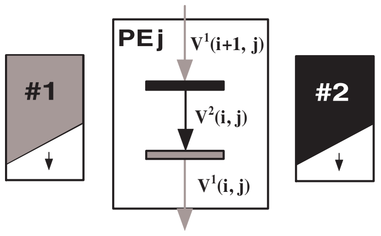

Figure 5. Coarse-grained scheduling in a pipelined, two-stage PE where the computation of cells is interleaved between two protein sequences. Our implementation uses a four-stage PE.

5.1. Processing Elements

The local data dependencies of the single motif-matching model in the Viterbi algorithm naturally lead to a systolic array architecture. Each cell $V_{i,j}$ of the Viterbi matrix is mapped to a processing element PE_j, where PE_j computes cells for the j^{th} position of the model. A linear array of such PEs computes the score of cells in an entire anti-diagonal.

Figure 4 shows the projection of the Viterbi matrix to the PE array. The protein sequence flows left to right through the PE array, one residue per time step. The data dependency structure of the Viterbi algorithm restricts PE communication to its two adjacent neighbours. Each processing element computes the score of the M, I, and D states at a particular position in the model as described in the previous section. In addition, the contribution of each cell to the single E state at the end of the model is also computed, and the best value in the row so far is forwarded to the next PE. As the computation proceeds vertically down in anti-diagonal sweeps, the score of the E state for each row is emitted from the final PE. The score of the Viterbi computation is the maximum of these scores emitted for each row.

A cell's computation is implemented using parallel signed adders and maximisers. A HMMER PE uses substantial hardware resources for transition and emission parameters at the PE's model position which must be made available synchronously with the addition operations. Previous work [10] has determined that signed 16-bit values are sufficient to reduce underflow/overflow errors over a wide range of input sequences.

The addition of two signed n-bit values produces, in general, an $(n + 1)$-bit result. Consequently, the result of each operation has to be saturated to the maximum or minimum representable value on an overflow or underflow, respectively. The latter is especially common in the HMMER computation since "impossible" state transitions or emissions are modeled with a transition score of negative infinity, i.e. the lowest possible score value. We increase a

Table 1. Variation of throughput with pipelined stages in a PE.

Stages	LUTs	Est. Freq. (MHz)
1	402	70
4	437	180

PE's datapath internally to 18 bits by sign extension. Two extra bits are sufficient to hold the result of two addition operations without an overflow or underflow. Maximum and minimum operators are then applied at the end of the final addition operation.

Coarse-grained Parallelism in PEs: The PE computation as outlined emits the set of state scores for a cell at each time step, i.e. the computation for a cell is performed in a single clock cycle. A HMMER PE, however, has a large logic propagation delay due to its computational complexity and the relatively large datawidth. An FPGA HMMER accelerator using an array of such PEs would be limited in its clock speed, and hence in its overall throughput, by this logic path.

The throughput of FPGA designs can be increased by pipelining the combinational logic with the longest propagation delay. This increased throughput comes at the expense of increased latency. FPGAs are specifically designed with pipelining in mind; every logic slice includes a flip-flop for this purpose. Pipelining results in little to no increase in the area of the PE, since the flip-flops would otherwise remain unused.

Pipelining makes effective use of hardware resources by time-multiplexing them with other cell computations. The challenge is to schedule cells of the Viterbi matrix to a PE at each clock period. For example, for an array of PEs each with a latency of two clock cycles (i.e. a two-stage PE), assume that anti-diagonal d_i is scheduled to the first stage at time t_k. At time t_{k+1}, computation for cells in anti-diagonal d_i proceeds to the second stage of the PE. At the same time step, the anti-diagonal d_{i+1} cannot be scheduled to the first stage, since it depends on the as-yet-unavailable cells in d_i. The data dependencies of the Viterbi algorithm make it impossible to time-multiplex multiple cells of the matrix in a single PE.

We introduce a novel approach to exploit coarse-grained parallelism in the PEs. The pipelined array of PEs in our design simultaneously computes the Viterbi matrix cell values of multiple protein sequences against the same motif model. From our previous illustration, anti-diagonal d_i^j is scheduled at time t_k, and anti-diagonal d_i^{j+1} at time t_{k+1}. Here, j is the protein sequence being compared, with there being as many sequences as is the latency of the PE. This approach

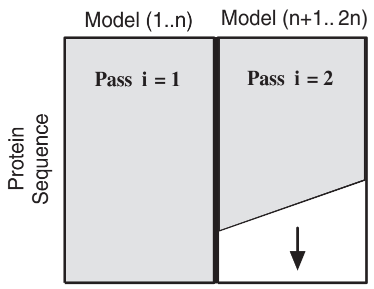

Figure 6. Supporting large models by a multi-pass vertical sweep of the Viterbi matrix

is successful due to the lack of dependencies between two independent protein sequences. The residue stream can be modified to contain multiple protein sequences interleaved with each other, to support coarse-grained parallelism. No further hardware support is required to enable this design.

Figure 5 shows the interleaved computation of two Viterbi matrices V^1 and V^2 in a two-stage PE. Cells $V^1_{i,j}$ and $V^2_{i,j}$ from the two protein sequences can be scheduled one after the other due to their complete independence. Table 1 shows the results of pipelining the PE on the throughput of our design. A four-stage pipeline increases the LUT count by just 9% while increasing the expected clock frequency by more than $2.5\times$. Clearly, the hardware resource increase is negligible compared to the gain in throughput provided by a pipelined PE design.

5.2. Model Parameters

The main feature distinguishing profile-HMM search from the simpler Smith-Waterman algorithm is the data requirements of the motif model. Each PE must have concurrent access to nine transition scores and fourty emission scores for its position in the motif model. To support an efficient design, these parameters must be stored using on-chip resources.

We use nine 16-bit registers for the transition parameters and a dual-ported block RAM to store the emission tables of the match and insert states for every position in the model. The block RAM is organised as 32-bit words, with each word containing the emission scores of the M and I states. A lookup is performed every clock cycle using the residue that is streamed into a PE. Careful scheduling is necessary to ensure that the scores are available at the appropriate time-point in the PE. Note that each dual-ported block RAM may be addressed independently from two sources and so can be shared by two adjacent PEs. A single such *model position block* (*MPB*) is associated with each PE, with the total design using half as many block RAMs as the number of PEs.

Another challenging task is the loading of parameters during the setup of a search. A motif model is collected by a top-level controller and must be distributed to the various MPBs. Having point-to-point links from the controller to every parameter requires a large amount of routing resources on the FPGA, which results in an inefficient implementation. Our design uses a simple bus architecture. The controller acts as the master and initiates parameter transfers to the MPBs. A 32-bit data bus is capable of transferring up to two parameter values, with a 5-bit address bus denoting the transition score being written, or the residue of the emission score. A parameter select line is used to distinguish between a transition and emission parameter transfer. Finally, an MPB select bus indicates the model position being written. A model of length m requires $25m$ clock cycles to be loaded into the MPBs before a search task can be executed. However, the load time is easily amortised over the entire search.

5.2.1. Supporting larger models

The average HMM model in the Pfam-A database is of length $m = 170$, with many models larger than 1000 positions. The model size on the HMMER accelerator is limited by the number of PEs, n, that can be implemented on an FPGA using available resources. On currently available FPGAs, this is typically too small to support all models in the Pfam-A database. Our solution is to divide the Viterbi matrix into vertical bands. Viterbi decoding is done in multiple passes $1 \le i \le \lceil m/n \rceil$ over the protein sequence, each pass decoding a fixed-size fraction of the model. To support multiple passes, the HMM models are padded to be of size equal to a multiple of n, with the query sequence being streamed on each pass. Figure 6 illustrates this process.

Two important design enhancements are required to support multiple passes: cell store-and-forward logic, and a quick model position turn-around time. Due to the cell dependency structure, cells at the first model position in pass i depend on cells at the final model position of pass $i-1$. To satisfy this dependency, we implement a store-and-forward logic block designed to store the cell value (scores of states M, I, D, and E) of the final PE for each protein residue during pass $i-1$. The forward logic initialises the first PE during pass i using this information. Two on-chip dual-ported block RAMs are sufficient to implement store-and-forward logic, supporting a sequence length of up to 1024.

Secondly, the model parameters for the i^{th} pass must be loaded as quickly as possible to reduce the turn-around time. We enhance the MPB by adding on-chip block RAMs for storage of transition scores. In addition to storing the emission scores at multiple model positions in each emission score table, we store transition parameters for multiple positions in a second dual-ported block RAM. When initialising the next pass, each MPB independently reads the tran-

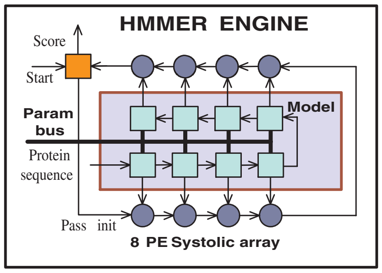

Figure 7. Organisation of the HMMER engine with $n = 8$ PEs. PEs are indicated by the circular blocks, MPBs by the blue boxes, and the store-and-forward logic by the orange box.

sition parameters for the next position, two per clock cycle. The design now uses n block RAMS, but the turn-around time is just 5 clock cycles.

The address for each block RAM is a triple of the form (p, i, a) where $p \in 0, 1$ indicates one of the two shared PEs, i is the pass number (corresponding to the model position) and a is the the transition score being read, or the residue whose emission score is being retrieved. Our implementation supports up to $i = 8$ model positions per MPB.

5.2.2. Protein sequence streaming

The design of the HMMER engine enables comparison of a single model against a database of protein sequences. A model is first loaded in to the FPGA, after which multiple protein sequences are streamed through. Our hardware defines a few special symbols to enable streaming. One of these is the *reset* residue which terminates every protein sequence in the stream. As it propagates through the array, it initialises each PE, with the final one forwarding the E state score of the previous protein sequence comparison. The use of the reset residue has an important consequence for protein sequence interleaving. Rather than considering coarse-grained parallelism as the simultaneous comparison of multiple protein sequences, it is better understood as a comparison of multiple streams against the same model. Each stream in turn is an independent concatentation of protein sequences, separated by reset residues. A simple offline pre-processing step creates the streams from a set of protein sequences, with necessary padding to ensure that the streams have equal length.

5.3. The HMMER engine

Figure 7 shows the block diagram of our FPGA based HMMER engine with $n = 8$ PEs. The systolic array computes the scores of the three states in each cell. Inter-PE communication is restricted to its immediate two neighbours. An equal number of MPBs are also arranged in a linear array, with the residue stream being forwarded through them. Each MPB communicates with its associated PE to make available the various model parameters. Model loading is done via the shared parameter bus, controlled by an external bus master.

The store-and-forward logic block receives cell values from the final PE and initialises the first PE. This block also initiates a search task and reports the score of the best path through a model to the end user.

5.4. Implementation state

The HMMER engine is work in progress. At the time of publication, the entire design except the store-and-forward logic block has been coded in VHDL and simulated. While the current implementation does not yet support multiple passes, the MPB does support loading of multiple models as described for the multi-pass functionality.

The design has been built post-place-and-route on a Xilinx Virtex-II 6000 FPGA with 33,792 slices and 144 18-Kbit on-chip block RAM memory. The device supports a 68 PE HMMER engine running at 180 MHz, processing 4 protein residue streams in parallel. The implementation uses 83% of the logic resources, and 72 block RAMs – 68 of these are used in the MPBs, and 4 for high-level buffering. The availability of logic resources is the critical factor in the number of PEs supported.

Our current implementation supports a model of up to 68 positions in a single pass, and models of up to 544 positions with a turn-around time of 5 clock cycles. Supporting larger models require a more expensive reloading of the MPBs. These numbers are however expected to scale linearly on the latest generation of FPGAs.

6. Performance Analysis

The throughput of the HMMER systolic array is measured by the number of cell updates per second (CUPS). We assume that the set of input protein sequences are packed into multiple streams, and a residue is available each clock cycle. We must account for the setup time spent loading new models, and the model turn-around time when processing large models. Throughput is calculated as:

$$T_{hw} = \frac{|S| \sum_{i=1}^{k} m_i}{\sum_{i=1}^{k} \left[|S| \lceil \frac{m_i}{n} \rceil + 25n \lceil \frac{m_i}{n} \rceil + 5 \lceil \frac{m_i}{n} \rceil \right]} \times f$$

The numerator is the total number of cells to be evaluated, and is the product of the total size of the protein sequences S, and the sum of all k models, each of size m_i. The denominator aggregates the number of clock cycles required by the hardware to compute these cells using n PEs. The first term calculates the number of clock cycles actually required by the hardware to process the models. The final two terms are the model load and turn-around times respectively. The HMMER engine runs at $f = 180$ MHz. Using a comparison of a sample of protein sequences from the Swiss-Prot database and the PFAM models, we estimate throughput to be $10,647$ MCUPS.

The throughput of the software system was empirically measured. Note that we did not use stock HMMER, but an optimised version presented in [16] which is $2\times$ faster than the original. On a single 2.8 GHz Pentium 4 workstation with 1 GB of RAM, the throughput of the software was 55 MCUPS. Assuming that the HMMER accelerator is not software bound, this yields a total speedup of over $190\times$.

Since our HMMER accelerator utilises a combination of a traditional CPU and an FPGA, the software must be able to process results of the hardware fast enough so as not to be a bottleneck. The load on the CPU depends on the cut-off threshold e-value used in the FPGA. The software processing times for various cut-off threshold e-values were previously presented in Table 2 of [10]. If the hardware takes less time than the path generation in software, then the software is a bottleneck. For an e-value of 0.001 or lower, the software is not a bottleneck and the full $190\times$ speedup can be realised with one CPU. Otherwise, simply doubling or quadrupling the number of processing cores will allow the accelerator to sustain maximum throughput.

Another possible throughput limitation is the I/O bandwidth available to feed the FPGA. To run at maximum throughput, the HMMER accelerator must be able to ingest data at 120 MB/s which is well below our system limit of 600 MB/s.

7. Conclusion

The rapid growth of computational biology data has greatly increased the need to accelerate profile HMM search. In this work we have described a hardware design to accelerate the Viterbi algorithm, using a systolic array of processing elements. We have exploited coarse-grained parallelism by pipelining our PEs to perform multiple searches in parallel, a technique easily generalised to hardware accelerators for other algorithms such as Smith-Waterman. The model parameters present an additional challenge for the computation, which we deal with effectively by storing them on-chip. Our implementation is expected to run two orders of magnitude faster than a general-purpose workstation. The architecture is also scalable to future FPGA technologies, allowing support for a greater number of PEs.

In the immediate future we intend to test the design on our available FPGA card and integrate it with the HMMER frontend. We also intend to support the original Plan7 model in hardware, to detect multiple copies of a motif in a protein. This will alleviate the software bottleneck described in the previous section.

8. Acknowledgments

The authors would like to thank Rahul Maddimsetty, Brandon Harris, and Eric Tyson for useful discussions during the development.

References

[1] Growth of GenBank. `http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html`.

[2] Timelogic DeCypherHMM solution, 2006.

[3] A. Bateman et al. The Pfam protein families database. *Nucleic Acids Research*, 32:D138–141, 2004.

[4] R. Durbin et al. *Biological Sequence Analysis*. Cambridge University Press, New York, 1998.

[5] S. Eddy. HMMER: Sequence analysis using profile hidden Markov models, 2004. `http://hmmer.janelia.org`.

[6] D. R. Horn et al. ClawHMMER: A streaming HMMer-search implementation. In *Proc. of ACM/IEEE conference on Supercomputing*, 2005.

[7] P. Krishnamurthy et al. Biosequence similarity search on the Mercury system. *To appear in Journal on VLSI Signal Processing*, 2007.

[8] A. Krogh et al. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[9] E. Lindahl. Altivec HMMER, version 2. `http://lindahl.sbc.su.se/software/altivec/altivec-hmmer-version-2.html`, 2005.

[10] R. Maddimsetty et al. Accelerator design for protein sequence HMM search. In *Proc. of ICS06*, June 2006.

[11] T. Oliver et al. Accelerating the Viterbi algorithm for profile Hidden Markov Models using reconfigurable hardware. *Lecture Notes in Computer Science, Springer-Verlag*, 3991:522–529, 2006.

[12] E. Portugaly and M. Ninio. HMMERHEAD accelerating HMM searches on large databases. In *Poster Abstracts from RECOMB*, pages 250–251, 2004.

[13] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. of the IEEE*, 77:257–86, 1989.

[14] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[15] Y. Sun and J. Buhler. Designing patterns for profile HMM search. In *Proc. of ECCB06*.

[16] B. Wun et al. Exploiting coarse-grained parallelism to accelerate protein motif finding with a network processor. In *Proc. 14th Int'l Conf. on PACT*, pages 173–184, 2005.

[17] Y. Yamaguchi et al. High speed homology search with FPGAs. In *Pacific Symposium on Biocomputing*, pages 271–282, 2002.