Parallelizing HMMER for Hardware Acceleration on FPGAs

Steven Derrien
IRISA, Université de Rennes 1
Campus de Beaulieu
35000 Rennes
sderrien@irisa.fr

Patrice Quinton
IRISA, ENS Cachan
Campus de Ker Lann
35000 Bruz
quinton@irisa.fr

Abstract

Profile based Hidden Markov Model is a widely used tool in bioinformatics. While being very valuable to biologists, it is extremely compute intensive and suffers from prohibitive execution time. We propose an original parallelization scheme of the `hmmsearch` *tool for FPGA technology. We show how to derive a flexible and generic hardware architecture which accelerates the* `hmmsearch` *main kernel by two orders of magnitude without modifying its original algorithm.*

1 Introduction

Over the last few years, reconfigurable computing has proved to be a very attractive solution for implementing compute intensive bio-computing algorithms. FPGA implementations of Smith and Waterman [15] or BLAST [1] algorithms have exhibited impressive speed-up factors, making them a very viable alternative to more expensive supercomputing infrastructures such as vector computers or PC clusters.

Profile based Hidden Markov Models (HMM) have been recently used by biologists to predict the structure and function of a protein directly from its representation as an amino-acid sequence [8]. Among existing software implementations of this model, the HMMER software package is one of the most widely used.

We propose to speed-up in hardware the most time consuming routine of the `hmmsearch` tool [4] of HMMER, namely the `P7Viterbi` procedure which computes the score between a HMM and an observed sequence. We do so by using powerful linear space-time mappings based on the so-called *polyedral model*. This leads to a flexible parallel architecture template which handles the feedback loop present in the Plan7 HMM model used by HMMER.

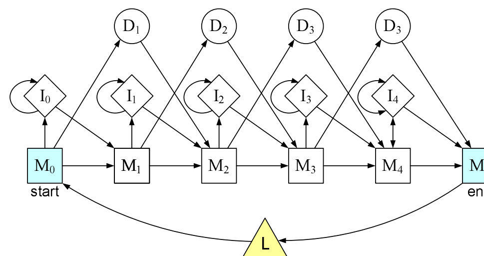

Figure 1. Structure of a Plan7 HMM

This paper is organized as follows. Section 2 briefly introduces the principles of the profiling based HMM algorithm and surveys related work on accelerating this algorithm both in hardware and software. Section 3 presents the principles of our parallelization schemes, and Section 4 details the space-time mapping refinements which lead to the final hardware architecture. Section 5 gives experimental results in terms of resource usage and performance improvement. Section 6 concludes and presents future work directions.

2 Background

2.1 Principles

Fig. 1 depicts the Plan7 HMM used in HMMER. It is a probabilistic model of a family of sequences (see [8] for more details) which contains three types of states: match states (square nodes), insertion states (diamond nodes) and deletion states (circle nodes). Running the `hmmsearch` tool consists in matching a single profile HMM against a large number of input sequences, and in finding the sequences having high similarity with this HMM.

The most time consuming routine in this program is the `P7Viterbi` kernel. This routine computes a similarity score between the target HMM model and the sequence at hand using dynamic programming. A simplified version of `P7Viterbi` is given in Fig. 2, where

seq[] holds the query sequence, x[][] holds the dynamic programming matrix, TAB[] is a cost funcion realized as a look-up table and f1() and f2() are expressions involving *add*, *sub* and *max* operations.

Matching a single HMM against a protein database is a very time consuming process which is repeated many times during intensive comparisons. Profiling shows that the P7Viterbi kernel accounts for more than 97 % of the execution time. It is therefore a perfect candidate for hardware acceleration.

2.2 Related Work

There have been several attempts to accelerate HMMER using SIMD features of modern CPUs[9], parallel machines [16], GPUs [7] and even Network Processors [17].

Some authors have proposed to speed-up HMMER by using reconfigurable hardware. Maddimsetty et al. [10] have studied different hardware-software partitioning schemes, but they do not present actual hardware implementations. Oliver et al. [12] propose an FPGA implementation of an HMM profile application and claim speed-up factors up to two orders of magnitude. But both [10, 12] consider a simplified version of the HMM model without feedback loop, in order to simplify the parallelization of the algorithm. We believe that such an approach is very unlikely to match biologists needs, since this simplified HMM might miss some potentially insteresting match. Besides, experience has shown that whenever an algorithm emerges as a standard (as it is now the case for HMMER), it is very difficult to convince its users community to accept algorithmic variations even though they run much faster.

In a recent work, Oliver at al. [13] propose an architecture which handles the P7Viterbi feedback loop. Their approach is quite similar to the *wavefront* schedule we detail in Section 4.2. However, instead of duplicating the look-up table memory as we do, they use a crossbar network architecture to distribute the look-up memory among the PEs. This approach is not scalable with the number of processors and is therefore limited to a small number of PEs.

In addition to adhere strictly to the original algorithm, the hardware realization that we target should also be *scalable* so as to be easily retargetable to different FPGA hardware platforms with varying amount of resources such as logic cells and memory blocks. Such constraints can hardly be handled by a fixed manual design, and this is why we propose instead to use an *architecture template* which can be seamlessly specialized to suit the user's needs. To derive this template

```
1  int P7Virterbi(char seq[]) {
2    int i,k;
3    for (i=0; i<=L; i++) x[i][0] = -INFTY;
4    for (k=1; k<=M; k++) x[0][k] = -INFTY;
5
6    for (i=1; i<=L; i++) {
7      for (k=1; k<=M; k++) {
8        x[i][k] = f1(
9            TAB[seq[i]][k], hmm[k], y[i-1],
10           x[i-1][k-1], x[i-1][k], x[i][k-1]);
11       if (k==M)
12           y[i] = f2(x[i][M],hmm[k]);
13     }
14   }
15   return x[L][M];
16 }
```

Figure 2. Simplified P7Viterbi code kernel

we rely on well-known parallelization techniques based on the so-called polyedral model which we introduce in the following section.

3 Parallelizing the hmmsearch Software

The P7Viterbi kernel (Fig. 2) consists of a double nested loop that implements the Viterbi dynamic programming algorithm. The loop carried dependencies in the inner loop prevents from parallelizing the execution using classical loop unrolling techniques, as noticed in [10]. Dependencies in the outer loop prevent loop interchange, which in turn forbids the parallel execution of the outer loop. Although it is still possible to take advantage of the instruction level parallelism available within the loop body, this would expose to few parallelism: in practice, FPGAs must be able to run several hundreds of operations per cycle to outperform CPUs.

For the sake of conciseness, we will use in Section 3 and 4 our compacted version of the P7Viterbi loop nest which contains exactly the same features – data-dependencies, indirect addressing, and existence of a feedback loop – as the original algorithm: the following results can therefore be applied to the original algorithm without loss of generality. The remaining of this Section shows how, by expressing the loop nest as a System of Affine Recurrent Equations (SARE), it is possible to derive an efficient parallel realization of the algorithm.

3.1 Expressing P7Viterbi as a SARE

The first step toward parallelization is to express the loop nest as a System of Affine Recurrent Equations, an intermediate representation that exposes data de-

pendences at the loop level. Let L denote the protein sequence length and M the HMM profile length. The SARE corresponding to our compacted `P7Viterbi` algorithm is given below:

$$\left| \begin{array}{lcl} x_{i,k} & = & f_1\,(y_{i-1,M}, hmm_k, \\ & & \quad x_{i-1,k}, \mathrm{TAB}(seq_i, k), \\ & & \quad x_{i,k-1}, x_{i-1,k-1}) \\ & & \\ y_{i,M} & = & f_2(hmm_M, x_{i,M}) \end{array} \right. \quad (1)$$

This SARE is defined over the domain $\mathcal{D}$ given by

$$\mathcal{D} = \{(i,k) \mid 1 \leq i \leq L,\ 1 \leq k \leq M\} \quad (2)$$

and we let $x_{i,k} = -\infty$ and $y_{i,k} = -\infty$ for $i = 0$ and/or $j = 0$. From (1), we can retrieve data dependencies between indexed variables. Denote $p1\ \delta\ p2$ if point $p2$ depends on point $p1$. These dependencies are summarized below:

$$\begin{array}{lcl} (i-1,k-1) & \delta & (i,k) \\ (i-1,k) & \delta & (i,k) \\ (i,k-1) & \delta & (i,k) \\ (i-1,M) & \delta & (i,k) \end{array}$$

3.2 Managing Arbitrary Sized Problems

We observe that several calls to `P7Viterbi` using the same HMM profile are done by the `hmmsearch` tool and we thus propose to merge all the sequences into a single macro sequence. Each sequence is delimited by a special character, the role of which is to reset the matching scores to their initial $-\infty$ value and to indicate that a new Viterbi algorithm instance is to take place.

On the other hand, even though the length of the HMM profile M remains constant for a given execution of `hmmsearch`, it can be different for each `hmmsearch` execution instance (`hmmsearch` motif length vary between 50 and 650, with an average size of 200). We must therefore design an architecture capable of handling arbitrary sized HMM motifs. This can be done by simply inserting idle states in the HMM motif. The role of these idle states is to propagate the scores of the last non-idle state until it reaches the feedback loop. In the rest of the paper, we will therefore consider that constant M denotes the maximum allowed model size of our architecture.

3.3 Linear Space-Time Mappings

Given a system of recurrent equations similar to the one presented in Equ. (1), we want to derive a space-time mapping that is a linear transformation which gives, for any indexed variable in the SARE:

- A *logical execution* time instant, in the form of a linear function of the index (which we call *schedule*), written as $s(i_0,\ldots,i_m) = s_0 i_0 + \ldots + s_m i_m$.
- A *physical location*, i.e. coordinates in a processor space. This location is also expressed in the form of a linear function of the index (that we call *allocation function*). In our case we are only interested in linear arrays. We therefore write $p(i_0,\ldots,i_m) = \alpha_0 i_0 + \ldots + \alpha_m i_m$.

Of course, this space-time mapping must satisfy several conditions.

- First, the chosen schedule must enforce all data dependencies present in the SARE. For $u, v \in \mathcal{D}$ with $u\ \delta\ v$, the schedule function must guarantee that $s(v) > s(u)$.
- Then, the space-time mapping must be conflict-free: there must be no u and v in $\mathcal{D}$ s.t. $v \neq u$, $s(u) = s(v)$ and $p(u) = p(v)$.

Proving that a schedule is conflict-free when each PE executes computations in a one dimensional domain is relatively straightforward (see [14]), but this is more involved for higher dimensional domains. To solve this problem, we use Darte et al. [2] results on *juggling schedules*, which can be summarized as follows. Given a rectangular domain $\mathcal{P}$ in $\mathbb{Z}^{m+1}$ defined as :

$$\mathcal{P} = \{i_0,\ldots,i_m \,|0 \leq i_0 < N_0,\ldots,0 \leq i_m < N_m\} \quad ,$$

a schedule is said to be *juggling* if it has the following properties:

- It is *conflict-free*, i.e. there are no two distinct iterations $(i_0,\ldots,i_m)$ and $(i'_0,\ldots,i'_m)$ in $\mathcal{P}$ which are scheduled at the same time instant.
- It is *dense*, i.e. the number of schedule steps separating the execution of iterations $(i_0, i_1,\ldots,i_m)$ and $(i_0+1, i_1,\ldots,i_m)$ is $\prod_{k=1}^{m} N_k$.

Darte and al. have shown that juggling schedules are of the form below (up to a permutation of index i_k) with $pgcd(\lambda_k, N_k) = 1$:

$$s(i_0,\ldots,i_m) = \left(\prod_{k=1}^{m} N_k\right) i_0 + \lambda_1 \left(\prod_{k=2}^{m} N_k\right) i_1 + \ldots + \lambda_m i_m \quad (3)$$

Notice that we may have used well established results on *multi-dimensional schedules* [5, 6] to find conflict-free schedules, but this would lead to more complex proofs in our case where the domains are hyperparallelepipeds.

In the next Section, we evaluate and refine several space-time mappings in order to obtain the best possible implementation.

4 Design Space Exploration

4.1 Adding a Dimension to the SARE

Although loop level parallel schedules cannot be found in the `P7Viterbi` routine, we observe that running the `hmmsearch` tool consists in completely independent matchings of a single HMM against a large number of input sequences: these matchings are completely independent and can therefore be run in parallel.

We model this additional parallelism by adding to the SARE a new index which identifies instances of the kernel running in parallel. Call j this additional index. The new iteration domain $\mathcal{D}'$ is then:

$$\mathcal{D}' = \{i, j, k \mid 0 \leq i < L,\ 0 \leq j < N,\ 0 \leq k \leq M\} \quad (4)$$

where N stands for the number of sequences that are matched in parallel during each realization of the modified SARE, which is shown below:

$$\left| \begin{array}{lcl} x_{i,j,k} & = & f_1\,(y_{i-1,j,M}, hmm_k, \\ & & \quad x_{i-1,j,k}, \mathrm{TAB}(seq_{i,j}, k), \\ & & \quad x_{i,j,k-1}, x_{i-1,j,k-1}) \\ & & \\ y_{i,j,M} & = & f_2(hmm_k, x_{i,j,M})) \end{array} \right. \quad (5)$$

Denote

$$s(i, j, k) = s_0 i + s_1 k + s_2 j \quad (6)$$

the scheduling function of this new SARE. The intrinsic data-dependencies remain unchanged by the transformation (all `P7Viterbi` instance are independent), therefore we can write the constraints on the scheduling function as:

$$\begin{array}{llcl} (i-1, k-1) & \delta(i, k) & \Rightarrow & s_0 + s_1 \geq 1 \\ (i-1, k) & \delta(i, k) & \Rightarrow & s_0 \geq 1 \\ (i, k-1) & \delta(i, k) & \Rightarrow & s_1 \geq 1 \\ (i-1, M) & \delta(i, 1) & \Rightarrow & s_0 \geq M \end{array}$$

This lead to many possible schedules, among which only two really deserve attention: a *wavefront* and an *interlaced* schedule.

4.2 Wavefront Space-Time Mapping

In this approach we use the space-time mapping given by

$$\begin{array}{lcl} s(i, j, k) & = & Mi + k \\ p(i, j, k) & = & j \end{array} \quad (7)$$

which obviously enforces data dependencies and is conflict-free. It is illustrated (for $M = 4$, $N = 4$ and $L = 5$) in Fig. 3, in which blue dashed arrows represent the iteration execution order on a single PE, while red lines show which iterations are actually executed in parallel.

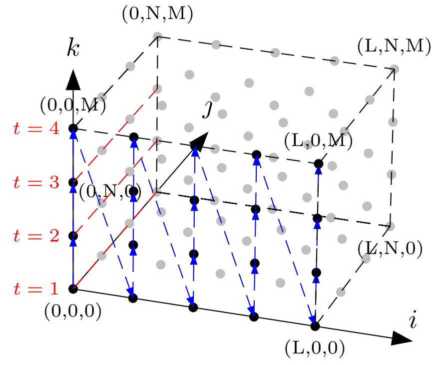

Figure 3. Wave-front space-time mapping.

The wavefront space-time mapping is a very natural parallelization scheme in which each PE executes a distinct `P7Viterbi` kernel instance, and the chosen value for parameter N directly controls the number of PEs in the architecture. Although these `P7Viterbi` instances share the same HMM parameters values, we must account for the fact that variable `TAB` in (5) is used as a lookup table with $seq_{i,j}$ and k as indices. At a given time instant t, all PEs share the same value of i, k, therefore they must access the same subset `TAB[*][k]`. Moreover, each PE accesses the whole table across time. As a consequence, all PEs must either hold a copy of the `TAB` variable, or they must access a shared copy of it. In the full `P7Viterbi` kernel the size of `TAB` is quite large: for an average HMM model size of 250, each processor requires a 10000×32 lookup table: this would severely limit the number of PEs that can be implemented.

One solution to this problem is to use a crossbar-like interconnect structure to distribute the table content (see subsection 2.2). Instead, we propose another space-time mapping which allows the `TAB` variable to be distributed among the PEs, thereby reducing the architecture memory footprint to its minimum, while avoiding the need for a complex crossbar interconnect structure (as opposed to Oliver et al. [13])

4.3 Interlaced Space-Time Mapping

This improved space-time mapping is given in Equ. (8), and is illustrated in Fig. 4:

$$\begin{array}{lcl} s(i, j, k) & = & Mi + j + k \\ p(i, j, k) & = & k \end{array} \quad . \quad (8)$$

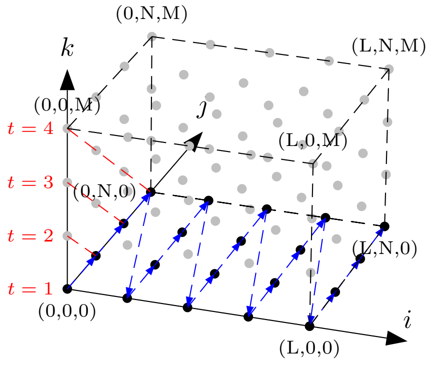

Figure 4. Interlaced space-time mapping.

The reader can check that this mapping enforces data-dependencies and is conflict free if and only if $M = N$. This results in a mapping in which $N = M$ PEs are running in parallel. Since PE_p only executes iterations for which $k = p$, the hardware cost of the lookup table implementation is much lower: PE_p only accesses the subset `TAB[*][p]`. It is thus possible to distribute the content of the lookup table among the PEs, each PE holding $1/M^{th}$ of it. The memory cost of a single PE is reduced by a factor of M compared to the approach of subsection 4.2.

On the other hand, we have no control over the resource usage of our architecture, since the number of PEs must be equal to the HMM model size M. This is again a severe limitation, since it is very unlikely that such architecture can be implemented on a FPGA even for moderate size model ($M \approx 50$).

4.4 Managing Resource Constraints

To overcome this new problem, we propose to use LSPG (Locally Parallel Globally Sequential) partitioning [11]. (LPGS – Local Parallel Globally Sequential, – partitioning [2] is of no interest here because of the feed-back loop: it would not allow partitioning along dimension k in order to reduce the number of processors in the architecture.) The LSGP partitioning transformation consists of two steps. We first consider the processors of the initial space-time mapping as being *virtual*, and we tile the virtual processor space. Each tile of the virtual processor space is then mapped to a single *physical* processor which executes in turn the calculations associated with all virtual processors of the tile.

As far as the SARE is concerned, tiling the virtual processor domain amounts to replacing the processor space index k by two new indexes (v, p) defined by the equality $k = \sigma p + v$. Parameter σ is called the *tile width*, and we have $0 \leq v < \sigma$ and $0 \leq p < M'$, with $M' = \lceil \frac{M}{\sigma} \rceil$ and $0 \leq \sigma p + v < M$. The resulting domain becomes:

$$\mathcal{D}" = \left\{ i, j, v, p \,|\, 1 \leq i < L,\ 1 \leq j < N,\ 1 \leq v < \sigma,\ 1 \leq p < M' \right\} \quad (9)$$

To cope with this transformation, the SARE is rewritten by using indices (i, j, v, p) instead of (i, j, k) and by modifying the data-dependencies accordingly.

For example, data dependency between $(i-1, j, k)$ and (i, j, k) now holds between points $(i-1, j, v, p)$ and (i, j, v, p) and is thus still uniform. When a dependency spans the k index, it leads to more complex situation. For example, consider the dependency between $(i, j, k-1)$ and (i, j, k). Depending on whether the values of k and $k-1$ belongs to the same tile (i.e lead to the same value of p), the original dependency is then transformed into two distinct data dependencies:

$$\begin{cases} (i, j, v{-}1, p)\ \delta\ (i, j, v, p) & \textit{when } v > 1 \\ (i, j, \sigma, p\text{-}1)\ \delta\ (i, j, 1, p) & \textit{when } v = 1 \end{cases} \quad (10)$$

The partitioned SARE thus becomes:

$$\left| \begin{array}{lcl} z_{i,j,v,p} & = & \begin{cases} x_{i,j,v-1,p} & \textit{when} \quad v > 1 \\ x_{i,j,\sigma,p-1} & \textit{when} \quad v = 1 \end{cases} \\ w_{i,j,v,p} & = & \begin{cases} x_{i-1,j,v-1,p} & \textit{when} \quad v > 1 \\ x_{i-1,j,\sigma,p-1} & \textit{when} \quad v = 1 \end{cases} \\ x_{i,j,v,p} & = & f_1\,(y_{i-1,j,M}, hmm_{\sigma p+v}, \\ & & \quad z_{i,j,v,p},\ x_{i-1,j,v,p}, \\ & & \quad w_{i,j,v,p}, \mathrm{TAB}(seq_{i,j}, v + \sigma p)) \\ y_{i,j,\sigma,M'} & = & f_2(hmm_M, x_{i,j,\sigma,M'}) \end{array} \right. \quad (11)$$

We now can use the following space-time mapping:

$$\begin{array}{lcl} s(i, j, v, p) & = & \sigma M' i + \sigma j + v + \sigma p \\ p(i, j, v, p) & = & p \end{array} \quad . \quad (12)$$

We can show that this mapping juggles for the domain $\mathcal{D}"$ if we choose $N = M'$. It is illustrated in Fig. 5 where the black dots correspond to the iteration subspace allocated to the first PE. Here $\sigma = 2$, and therefore $\lceil \frac{M}{\sigma} \rceil = 2$ iterations are executed at a given time step.

4.5 Pipelining the PE Datapath

Although the architecture of Section 4.4 allows for a resource constrained implementation, it still requires that a whole loop iteration be executed within a single clock cycle. Given that the *complete* `P7Viterbi` loop body contains more than 20 arithmetic operations among which 7 lie in the critical path, the final maximum clock frequency of our design is likely to be disappointing.

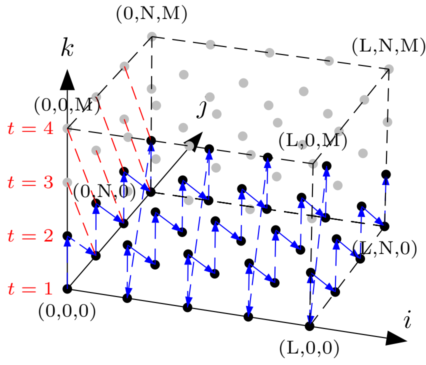

Figure 5. Partitioned space-time mapping.

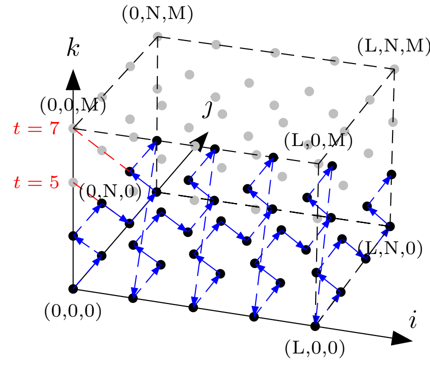

Figure 6. Pipelined space-time mapping

One solution is to modify the scheduling so that the pipelining of the loop body execution becomes possible (see our previous work [3]). This is achieved by applying another tiling transformation along axis j. Let us write $j = \lambda jj + l$, with jj and l being the two new index. The new SARE is given in Equ. (14), and its variables are defined over the domain:

$$\mathcal{D}''' = \{i, jj, l, v, p \mid 1 \le i < L,\ 1 \le \lambda jj + l < N, \\ 1 \le l \le \lambda, 1 \le v < \sigma,\ 1 \le p < M'\} \quad (13)$$

$$\left|\begin{array}{lcl} z_{i,jj,l,v,p} & = & \begin{cases} x_{i,jj,l,v-1,p} & when \quad v > 1 \\ x_{i,jj,l,\sigma,p-1} & when \quad v = 1 \end{cases} \\ w_{i,jj,l,v,p} & = & \begin{cases} x_{i-1,jj,l,v-1,p} & when \quad v > 1 \\ x_{i-1,jj,l,\sigma,p-1} & when \quad v = 1 \end{cases} \\ x_{i,jj,l,v,p} & = & f_1\,(y_{i-1,j}, hmm_{\sigma p+v}, \\ & & \quad w_{i,jj,l,v,p},\ x_{i-1,jj,l,v,p}, \\ & & \quad z_{i,jj,l,v,p},\ \mathrm{TAB}(seq_{i,jj,ll}, v)) \\ y_{i,jj,l,\sigma,M'} & = & f_2(hmm_M, x_{i,jj,l,\sigma,M'}) \end{array}\right. \quad (14)$$

Using the following space-time mapping:

$$\begin{array}{rcl} s(i,jj,l,v,p) & = & \sigma\lambda M' i + \sigma(\lambda jj + j) + \lambda v + \lambda\sigma p \\ p(i,j,v,p) & = & k \end{array} \quad (15)$$

we can show that this mapping juggles iff $0 \le jj < N'$ and $0 \le l < \lambda$ when N' and λ satisfy $\lambda N' = M'$. This space-time mapping is illustrated in Fig. 6 (with $\sigma = 2$ and $\lambda = 2$).

The execution of two dependent iterations is now separated by at least λ cycles (for dependency $(i, jj, l, \sigma, p-1)\ \delta\ (i, jj, l, v, p)$), and by at most $\lambda\sigma M' + 1$ cycles (for dependency $(i-1, jj, l, \sigma, p-1)\ \delta\ (i, jj, l, v, p)$). By carefully selecting λ, it is therefore possible to find the best tradeoff between resource cost (each PE must implement a $\lambda M + 1$ deep delay-line) and the operating frequency (the data-path corresponding to the loop body can be implemented with λ pipeline stages).

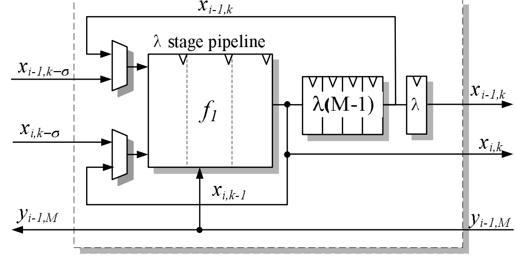

Figure 8. PE internal structure.

4.6 Resulting architecture

Using the mapping of section 4.5, we obtain an architecture which consists of a linear array of M' processors as illustrated in Fig. 7. All PEs in the array have local interconnect except for the last one ($PE_{M'}$) which broadcasts part of its results (this broadcast corresponds to the HMM feedback loop). PEs of this array are active every cycle, except for $PE_{M'}$ in which the variable $y_{i,M}$ is only updated (and broadcasted) every σ cycles.

From the schedule, we can derive the depth of the delay lines associated to each data dependency in a PE. This leads to a PE internal architecture shown in Fig.8, where functions f_1 and f_2 are implemented as a λ stage pipeline datapath (for the sake of readability, the data pipelines for the Amino Acid sequence $seq_{i,j}$ and the HMM parameters hmm_k pipelines are not represented.

Table. 1 summarizes the characteristics of the various space-time mappings for the full `P7Viterbi` kernel as a function of the HMM model size (M), the design parameters (σ,λ,N) and the number of distinct bases in the Amino Acid set (here 25).

5 Experimental Results

The hardware resource usage (in terms of logic cells and memory) of our hardware accelerator is highly dependent on the size M of the HMM model at hand, on

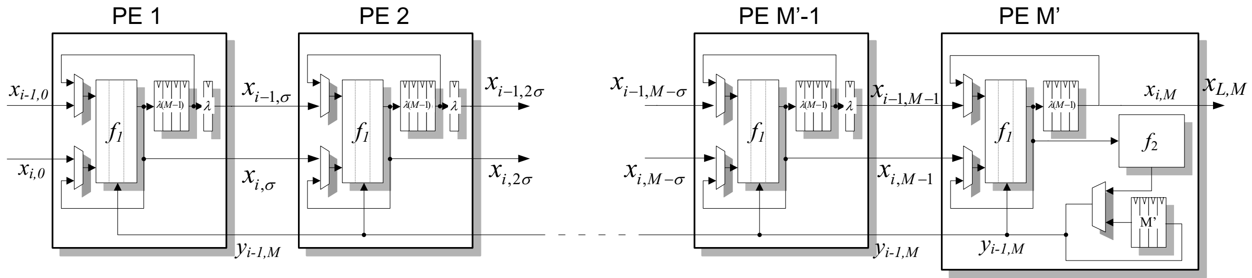

Figure 7. The final processor array architecture for the P7Viterbi routine.

M	#PE	# sequences processed in parallel	Memory cost per PE (in 24 bits words)	# pipeline stages
Wave front ($\sigma = 1, \lambda = 1$)	N	N	$M \times (2 \times 25 + 4)$	1
Interlaced ($\sigma = 1, \lambda = 1$)	M	M	$4 \times M + 2 \times 25$	1
Partitioned ($\lambda = 1$)	$\lceil M/\sigma \rceil$	$\lceil M/\sigma \rceil$	$\sigma \times (4 \times \lceil M/\sigma \rceil + 2\sigma \times \frac{25}{M})$	1
Pipelined	$\lceil M/\sigma \rceil$	$\lceil M/\sigma \rceil \times \lambda$	$\sigma \times (4\lambda \times \lceil M/\sigma \rceil + 2\sigma \times \frac{25}{M})$	λ

Table 1. A summary of the characteristics for the proposed Space-Time Mappings

design parameters such as the pipeline level λ, and on the partitioning factor σ. In order to have a more quantitative view of this resource cost, we implemented on a Xilinx Spartan3-4000 FPGA several configurations of the *complete* `P7Viterbi` kernel (including control), each one with a different set of parameters. To do so we designed a generic architectural VHDL template which can be parameterized at compile time (parameters include σ, λ, M).

Although the original `P7Viterbi` uses 32 bits integer arithmetic, it is possible to reduce the datapath bitwidth to 24 bits without effect on the final results. As a consequence, this benchmark considers scores and parameters encoded as 24-bit integer values.

For a fair comparison, we implemented the architecture corresponding to the *wave-front schedule*. This architecture was obtained by generating an instance of our architecture in which we set $\sigma = M$ and $\lambda = 1$. The results are summarized in Table 2. We can observe that the resource bottleneck is the limited amount of embedded memory blocks on the FPGA, which strongly affects the number of PEs which can actually be implemented on the device, and the level of pipelining in the datapath.

Performance figure for `hmmsearch` are generally measured in Million Cells Update per Second (MCUP/s), where a cell update corresponds to an iteration of the Viterbi algorithm. In our implementation, each PE performs up to 1 iteration per cycle, whith clock frequencies ranging from 33 MHz to 66 MHz. This is to be compared to a reported software performance of 24 MCUP/s on a Intel P4 CPU (see Oliver et al.[12]). In short, an architecture consisting of 10 PEs should outperform a desktop CPU by a factor of almost thirty.

However, this performance gain is balanced by the following observation: when using a HMM profile of size M_0 with an architecure which accomodates a maximum model size of M, there is only a fraction M_0/M of the computations that actually contribute to the matching score. For example for $M_0 = 50$ and $M = 200$, our architecture performs only one useful computation every four cycles. This strongly encourages the use of dynamic reconfiguration, in order to adapt at run-time the architecture to the HMM profile at hand. In our case, this would consists in using an architecture instance with a value of M suited (i.e close) to the size M_0 of the query HMM. This is made possible thanks to our flexible architectural model which can be seamlessly parameterized to obtain a set of hardware configurations with different values for M.

6 Conclusion

In this paper, we have presented an original parallelization scheme for the `P7Viterbi` algorithm, which represents the most time consuming kernel of the `hmmsearch` application. This parallelization scheme is based on the polyedral model and allowed us to derive a simple yet flexible parallel architecture for accelerating the `hmmsearch` program. Our ongoing work includes an in depth analysis of the `P7Viterbi` precision requirements, and a system level exploration of the trade-off between performance and Quality of Results, under the constraint that no false negative should be tolerated.

M	50	100	200	400	50	50	100	100	200	200	300	400	600
σ	50	100	200	400	3	3	5	5	10	12	17	27	150
λ	1	1	1	1	1	2	1	2	1	2	1	1	1
N_{PE}	1	1	1	1	22	22	22	22	22	18	18	15	4
Slices	1302	1419	1684	2100	21200	20100	21500	21000	21500	16650	18100	16500	4700
BRAM	8	14	26	51	88	88	88	88	88	90	90	90	81
f_{max}	33	33	33	33	33	55	33	50	33	50	33	33	33
GCUP/s	33	33	33	33	730	1210	730	1210	730	900	590	495	132
speedup	1.4	1.4	1.4	1.4	31	50	22	50	31	37.5	18	15	4

Table 2. Resource usage and performance on a Xilinx Spartan3-4000

References

[1] S.F. Altschul, T.L. Madden, A.A. Schffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: a New Generation of Protein Database Search Programs. *Nucleic Acids Research*, pages 3899–3402, 1997.

[2] A. Darte, R. Schreiber, B. Ramakrishna Rau, and F. Vivien. Constructing and exploiting linear schedules with prescribed parallelism. *ACM Trans. Des. Autom. Electron. Syst.*, 7(1):159–172, 2002.

[3] S. Derrien, S. Rajopadhye, and S. Sur-Kolay. Combining Instruction and Loop Level Parallelism for Array Synthesis on FPGAs. In *International Symposium on System Synthesis (ISSS'01), Montral*, 2001.

[4] S. Eddy. Sequence Analysis Using Profile Hidden Markov Models. Technical report, Washington University at Saint Louis, 2004.

[5] P. Feautrier. Some Efficient Solutions to the Affine Scheduling Problem. Part II. Multidimensional Time. *International Journal of Parallel Programming*, 21(6):389–420, 1992.

[6] A. C. Guillou, P. Quinton, and T. Risset. Hardware Synthesis for Multi-Dimensional Time. In *14th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'03)*, 2003.

[7] D. R. Horn, M. Houston, and P. Hanrahan. ClawHMMER: A Streaming HMMer-Search Implementation. In *SC'05 : Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005.

[8] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden Markov Models in Computational Biology: Applications to Protein Modeling. *Journal Molecular Biology*, 235:1501–1531, 1994.

[9] Eric Lindahl. HMMer Altivec Implementation. http://lindahl.sbc.su.se/software/altivec/altivec-hmmer,-version-2.html, 2005.

[10] R. P. Maddimsetty, J. Buhler, R. D. Chamberlain, M. A. Franklin, and Brandon Harris. Accelerator Design for Protein Sequence HMM Search. In *Proceedings of the ACM International Conference on Supercomputing*, Cairns, Australia, 2006. ACM.

[11] Dan I Moldovan and Jose A. B Fortes. Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays. *IEEE Transactons on Computers*, 35(1):1–12, 1986.

[12] T. Oliver, B. Schmidt, Y. Jakop, and D. L. Maskell. Accelerating the Viterbi Algorithm for Profile Hidden Markov Models Using Reconfigurable Hardware. In *International Conference on Computational Science*, 2006.

[13] T. Oliver, L. Y. Yeow, and B. Schmidt. High Performance Database Searching with HMMer on FPGAs. In *HiCOMB 2007, Sixth IEEE International Workshop on High Performance Computational Biology*, march 2007.

[14] P. Quinton. Automatic Synthesis of Systolic arrays from Recurrent Uniform Equations. In *International Conference on Computer Architecture*, 1984.

[15] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol*, 147:195–197, 1981.

[16] J. P. Walters, B. Qudah, and V. Chaudhary. Accelerating the HMMER Sequence Analysis Suite Using Conventional Processors. In *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*, 2006.

[17] Ben Wun, Jeremy Buhler, and Patrick Crowley. Exploiting Coarse-Grained Parallelism to Accelerate Protein Motif Finding with a Network Processor. In *PACT '05: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, 2005.