Reconfigurable Embedded Control Systems:

Applications for Flexibility and Agility

Mohamed Khalgui
Xidian University, China

Hans-Michael Hanisch
Martin Luther University, Germany

INFORMATION SCIENCE REFERENCE
Hershey · New York

Director of Editorial Content:	Kristin Klinger
Director of Book Publications:	Julia Mosemann
Acquisitions Editor:	Lindsay Johnston
Development Editor:	Joel Gamon
Publishing Assistant:	Casey Conapitski
Typesetter:	Milan Vracarich Jr.
Production Editor:	Jamie Snavely
Cover Design:	Lisa Tosheff

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Reconfigurable embedded control systems : applications for flexibility and
agility / Mohamed Khalgui and Hans-Michael Hanisch, editors.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-60960-086-0 (hardcover) -- ISBN 978-1-60960-088-4 (ebook) 1.
Programmable controllers. 2. Embedded computer systems. 3. Digital control
systems. I. Khalgui, Mohamed. II. Hanisch, Hans-Michael.
TJ223.P76R43 2011
629.8'95--dc22
2010042276

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 12
FPGA-Based Accelerators for Bioinformatics Applications

Alba Cristina Magalhaes Alves de Melo
University of Brasilia, Brazil

Nahri Moreano
Federal University of Mato Grosso do Sul, Brazil

ABSTRACT

The recent and astonishing advances in Molecular Biology, which led to the sequencing of an unprecedented number of genomes, including the human, would not have been possible without the help of Bioinformatics. Bioinformatics can be defined as a research area where computational tools and algorithms are developed to help biologists in the task of understanding the organisms. Some Bioinformatics applications, such as pairwise and sequence-profile comparison, require a huge amount of computing power and, therefore, are excellent candidates to run in FPGA platforms. This chapter discusses in detail several recent proposals on FPGA-based accelerators for these two Bioinformatics applications, highlighting the similarities and differences among them. At the end of the chapter, research tendencies and open questions are presented.

1. INTRODUCTION

Bioinformatics is an interdisciplinary field that involves computer science, biology, mathematics and statistics (Mount 2004). Its main goal is to analyze biological sequence data and genome content in order to obtain the function/structure of the sequences as well as evolutionary information.

Once a new biological sequence is discovered, its functional/structural characteristics must be established. In order to do that, the newly discovered sequence is compared against the sequences that compose biological databases, in search of similarities. Sequence comparison is, therefore, one of the most basic operations in Bioinformatics. A sequence can be compared to another sequence (pairwise comparison), to a profile that describes a family of sequences (sequence-profile comparison) or to a set of sequences (multiple sequence comparison). The most accurate algorithms to execute pairwise and sequence-profile compari-

DOI: 10.4018/978-1-60960-086-0.ch012

sons are usually based on dynamic programming, with quadratic time and space complexity. This can easily lead to extremely high execution times and huge memory requirements, since biological databases are growing exponentially and biological sequences usually are extremely long.

Parallel processing can be used to produce results faster, reducing significantly the time needed to obtain results with dynamic programming-based algorithms. However, using parallel processing in Bioinformatics is not straightforward since the problems are often solved by complex methods, with a great amount of data dependencies. Moreover, it has been observed that, for some problems that involve really huge data sets, software-only parallel processing techniques are not able to reduce the execution time to reasonable limits. For instance, in order to use the dynamic programming exact method to compare the human chromosome 21 and the ape chromosome 22, with sizes 46 Million Base Pairs (MBP) and 34 MBP, respectively, it is estimated that a cluster with 2048 processors would take approximately one day (Batista et al., 2008). This shows clearly that new approaches and technologies must be used to further reduce such high execution times.

Designing specific hardware for such algorithms is a very attractive alternative to software-only solutions. Since the hardware will be tailored to execute a specific algorithm, drastic performance improvements can be achieved. This observation was made by many researchers in Bioinformatics in the 1990s, that led to a great variety of hardware-based solutions. Due to its flexibility, FPGAs (Field Programmable Gate Arrays) are the natural choice for these designs. In the last years, FPGAs have intensively been used to build specific circuits, especially targeted to accelerate compute-intensive Bioinformatics applications.

The goal of this chapter is to discuss in detail and compare the recent advances in FPGA-based accelerators for some classes of Bioinformatics applications. The problems discussed are two types of Biological Sequence Comparison: pairwise sequence comparison and sequence-profile comparison. The first one is widely used all over the world as a first step in the solution of complex problems such as the determination of the evolutionary history of the species. The second one is extremely important since it is used to decide if a recently sequenced protein belongs or not to a particular protein family/superfamily. For both problems, many FPGA-based accelerators have been proposed that obtained impressive speedups over the sequential and software-only parallel implementations.

The structure of this chapter is the following. In Section 2, we will introduce the Biological Sequence Comparison problem and present the most widely used algorithms to solve it. In Section 3, we discuss several state-of-the-art FPGA accelerators that tackle the pairwise sequence comparison problem. In Section 4, several state-of-the-art FPGA-based accelerators for the sequence-profile comparison problem are discussed. Finally, Section 5 concludes the chapter, presenting the future tendencies in this research area.

2. PAIRWISE COMPARISON AND SEQUENCE-PROFILE ALIGNMENT

2.1. Alignment and Score

A biological sequence is a single and continuous molecule of nucleic acid or protein. It is represented by a linear list of residues, which are nucleotide bases (for DNA sequences, for instance) or amino acids (for protein sequences). To compare biological sequences, we need to find the best alignment between them, which is to place one sequence above the other making clear the correspondence between similar residues from the sequences (Durbin et al., 1998).

Given an alignment between two sequences *s* and *t*, a score can be associated for it as follows. For each two residues in the same column, we associate (a) a punctuation *ma,* if the two characters are identical (*match*); or (b) a penalty *mi,* if the characters are different (*mismatch*); or (c) a penalty *g,* if one of the characters is a space (*gap*). The score is the sum of the values computed for each column. The maximal score is called the similarity between the sequences. Figure 1 illustrates an alignment.

If proteins are being compared, a substitution matrix of size 20 × 20 is used to store the match/mismatch punctuation. The most commonly used substitution matrices are PAM and BLOSUM (Durbin et al., 1998).

2.2. Pairwise Sequence Comparison

2.2.1. Needleman-Wunsh Algorithm (NW)

One of the first exact methods to globally align two sequences was NW (Needleman & Wunsh, 1970). It is based on dynamic programming and calculates a similarity matrix of size $m \times n$, where m and n are the sizes of sequences s and t, respectively. This algorithm has $O(mn)$ time and space complexity. It is divided in two phases: create the similarity matrix and obtain the best global alignment.

The first phase receives input sequences s and t. The notation used to represent the i-th character of a sequence *seq* is *seq*[*i*] and, to represent a prefix with i characters, we use *seq*[1..*i*]. The similarity matrix is denoted D, where $D(i, j)$ contains the similarity score between prefixes s[1..*i*] and t[1..*j*].

At the beginning, the first row and column are filled with the values $g \times i$, where i is the size of the non-empty subsequence and g is the gap penalty. This represents the cost of aligning a non-empty subsequence with an empty one. Note that $D(0,0) = 0$. The remaining elements of D are obtained from Equation (1). In this equation, $p(i, j) = ma$ if $s[i] = t[j]$ (*match*) or *mi* otherwise *(mismatch).* The similarity score between sequences s and t is the value contained in cell $D(m,n)$.

Figure 1. An alignment between sequences s = TTGGTGG and t = TTGTCGAGG, with score=+1. Values for matches, mismatches and gaps are ma=+1, mi=–1, g=–2, respectively

T	T	G	-	-	G	T	G	G	
T	T	G	T	C	G	A	G	G	**score**
+1	+1	+1	–2	–2	+1	–1	+1	+1	**+1**

$$d(i,j) = \max \begin{cases} D(i, j-1) = g \\ D(i-1, j-1) + p(i,j) \\ D(i-1, j) + g \end{cases} \tag{1}$$

Figure 2 presents the NW similarity matrix between sequences s = TTGGTGG and t = TTGTCGAGG. The arrows indicate the cell from where the value was obtained.

Phase 2 is responsible to obtain the best global alignment. In order to do that, the algorithm starts from cell $D(m,n)$ and follows the arrows until cell $D(0,0)$ is reached. A left arrow in $D(i, j)$ (Figure 2) indicates the alignment of $s[i]$ with a gap in t. An up arrow represents the alignment of $t[j]$ with a gap in s. Finally, an arrow on the diagonal indicates that $s[i]$ is aligned with $t[j]$. For the similarity matrix D illustrated in Figure 2, the global alignment retrieved is the one shown in Figure 1.

Note that many best global alignments may exist, since many arrows can exist in the same cell $D(i, j)$, indicating that the score value was produced from more than one cell. In Figure 2, we did not represent these multiple arrows because, in this case, we assumed that preference is given to the match/mismatch (diagonal).

2.2.2. Smith-Waterman Algorithm (SW)

To obtain the similarity between parts of the sequences, local alignment must be used. The exact algorithm that is most widely used in this case is the SW algorithm proposed by Smith &

Figure 2. Similarity matrix D to globally align sequences s=TTGGTGG and t=TTGTCGAGG, with score=+1. Values for matches, mismatches and gaps are ma=+1, mi=–1, g=–2, respectively

		T	T	G	G	T	G	G
	0	–2	–4	–6	–8	–10	–12	–14
T	–2	**↖1**	←0	←–2	←–4	←–6	←–8	←–10
T	–4	↖–1	**↖2**	←0	←–2	←–4	←–6	←–8
G	–6	↖–3	↑0	**↖3**	↖1	←–1	↖–3	↖–5
T	–8	↖–5	↖–2	**↑1**	↖2	↖2	↖0	←–2
C	–10	↑–7	↑–4	**↑–1**	↖0	↖1	↖1	↖–1
G	–12	↑–9	↑–6	↖–3	**↖0**	↖–1	↖2	↖2
A	–14	↑–11	↑–8	↑–5	↑–2	**↖–1**	↑0	↖1
G	–16	↑–13	↑–10	↖–7	↖–4	↖–3	**↖0**	↖1
G	–18	↑–15	↑–12	↖–9	↖–6	↖–5	↖–2	**↖1**

Waterman (1981). Like NW, SW is also based in dynamic programming with *O(mn)* time and space complexity. However, there are three basic differences between them.

The first difference is on the initialization of the first row and column, which are filled with zeros in SW. In this way, gaps do not receive penalty if they are at the beginning of the alignment. The second difference involves the equation used to calculate the remaining cells since, in SW, no negative values are allowed. In order to do that, the value zero is included in Equation (1), generating Equation (2).

$$D(i,j) = \max \begin{cases} D(i,j-1)+g \\ D(i-1,j-1)+p(i,j) \\ D(i-1,j)+g \\ 0 \end{cases} \tag{2}$$

The third difference concerns the cell used to start the traceback process. To obtain the best local alignment, the SW algorithm starts from the cell which has the highest value, following the arrows until a zero-valued cell is reached.

Figure 3 presents the SW similarity matrix between sequences s = TTGGTGG and t = TTGTCGAGG.

2.2.3. Affine-gap Model and Divergence

The algorithms NW and SW assign a constant value to gaps. However, keeping gaps together generates more significant results, in a biological perspective (Durbin et al., 1998). For this reason, the opening of a gap must have a greater penalty than its extension. Based on this observation, Gotoh (1982) proposed an algorithm where the gap penalty is calculated with Equation (3), where k is the number of consecutive gaps, v is the penalty for opening a gap and u is the penalty for extending it (affine gap model).

$$w(k) = u \times k + v,\ k \geq 1 \tag{3}$$

In order to calculate gaps according to Equation (3), two matrices are needed (E and F), in addition to the similarity matrix D. These additional matrices are used to compute the cost of a set of gaps in sequences s and t, respectively. Even with this, time complexity remains O(mn) (Gotoh, 1982). Equations (4), (5) and (6) present the recurrence relations of the Gotoh algorithm.

Figure 3. Similarity matrix D to locally align sequences s = TTGGTGG and t = TTGTCGAGG, with score=+3. Values for matches, mismatches and gaps are ma=+1, mi=–1, g=–2, respectively

		T	T	G	G	T	G	G
	0	0	0	0	0	0	0	0
T	0	**↖1**	↖1	0	0	↖1	0	0
T	0	↖1	**↖2**	↖0	0	↖1	↖0	0
G	0	0	↖0	**↖3**	↖1	0	↖2	↖1
T	0	↖1	↖1	↑1	↖2	↖2	0	0
C	0	0	0	↖0	↖0	↖0	0	0
G	0	0	0	↖1	↖1	0	↖1	↖1
A	0	0	0	0	0	0	0	0
G	0	0	0	↖1	↖1	0	↖1	↖1
G	0	0	0	↖1	↖2	0	↖1	↖2

$$D(i,j) = \max \begin{cases} D(i,j-1)+g \\ E(i,j) \\ D(i-1,j-1)+p(i,j) \\ F(i,j) \\ 0 \end{cases} \tag{4}$$

$$E(i,j) = \max \begin{cases} E(i,j-1)+gap\_extension \\ D(i,j-1)+gap\_opening \end{cases} \tag{5}$$

$$F(i,j) = \max \begin{cases} F(i-1,j)+gap+extension \\ D(i-1,j)+gap\_opening \end{cases} \tag{6}$$

The method proposed by Fickett (1984) is adequate to globally align similar sequences. It is based on the following observation: to follow the main diagonal is to align both sequences without gaps. As long as gaps are introduced, the alignment leaves the main diagonal. If the sequences are very similar, the alignment between them is near the main diagonal. Thus, in order to compute the best global alignment(s), it is sufficient to calculate only a small band (*k*-band) near the main diagonal.

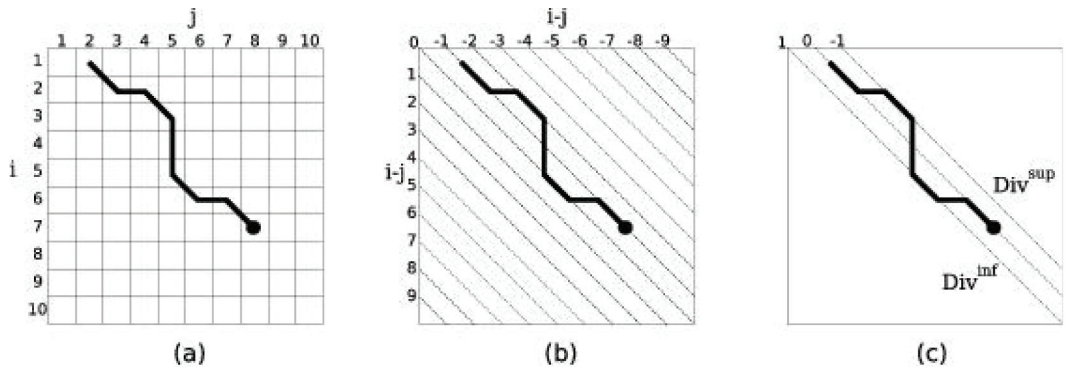
This algorithm has time and space complexity *O(kn)* and works as follows. First, the *k*-band is estimated and the algorithm is executed for this band. If the optimal alignment is outside the band, a larger *k*-band is estimated, until the alignment is entirely retrieved.

Fickett´s algorithm (Fickett, 1984) can be applied to a local alignment problem. To do that, it is sufficient to transform the global alignment problem into a local alignment one. First, the entire similarity matrices are calculated and the similarity score as well as the coordinates that determine the ending position of the optimal alignment ($D(i_{end}, j_{end})$) are obtained. After that, the similarity matrices are re-calculated over the reverses of the sequences, in order to obtain the coordinates where the optimal alignment starts ($D(i_{start}, j_{start})$). Having the coordinates $D(i_{start}, j_{start})$ and $D(i_{end}, j_{end})$, any algorithm designed for global alignment can be applied (Gusfield 1997).

Knowing that the entire matrices will be processed twice, as discussed in the above paragraph, Z-align (Batista et al., 2008) proposed the divergence concept, which enables the algorithm to obtain the exact *k*-band that contains the optimal alignment.

During the matrices calculation, it is also calculated how each possible alignment diverges from its beginning coordinates. As long as there are only matches or mismatches, there is no divergence and the alignment follows its main diagonal. When a gap is introduced, the alignment diverges from the diagonal. This divergence can be superior (DIV^{sup}) or inferior (DIV^{inf}), depending on the sequence where the gap is inserted. Therefore, we need

Figure 4. (a) A possible alignment between sequences s and t. (b) The divergences are initially calculated taking the main diagonal of the similarity matrix D as a basis. (c) Finally, the values DIV^{sup} and DIV^{inf} are adjusted to refer to the beginning of the alignment



two additional dynamic programming matrices (DIV^{sup} and DIV^{inf}). At the end of the similarity matrix computation, the highest score and its coordinates (i,j) are obtained. The divergences for the optimal alignment can be found in $DIV^{sup}(i, j)$ and $DIV^{inf}(i, j)$, respectively. Figure 4 illustrates the divergence concept.

2.2.4. The DIALIGN Algorithm

DIALIGN (DIAGonal ALIGNment) (Morgenstern et al., 1998) is a method for sequence alignment that searches for *fragments* (or *diagonals*) that have no gaps and aligns them. In DIALIGN, a pairwise alignment is defined to be a chain of fragments. By doing that, it avoids the overhead of computing gap scores and discards low similarity regions out of diagonals. When compared to Smith-Waterman (Section 2.2.2), DIALIGN provides better results if there are more than one distinct regions with high similarity (Mount, 2004).

The algorithm works as follows. For each pairwise alignment, many diagonals can be found. Thus, it is necessary to calculate the relevance E of each diagonal before attempting to align them. This is done calculating the probability $P(l,sm)$ of a diagonal D of size l have at least sm matches and then using the negative natural logarithm of it ($E(l,sm) = -ln(P(l,sm))$) (Morgenstern et al., 1998).

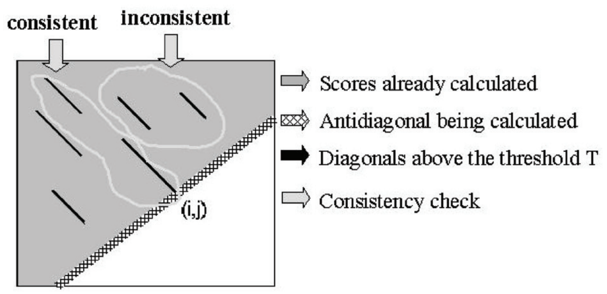
For each candidate diagonal D, a weight $w(D)$ is assigned as $E(l,sm)$ if $E(l,sm)$ is above a given threshold T and 0, otherwise. A high $w(D)$ means that the probability of diagonal D occurrence by chance is small.

When the algorithm obtains a new significant diagonal, it tries to align it consistently with other previously calculated significant diagonals (Morgenstern et al., 1998). Figure 5 illustrates the cases when fragments can and cannot be consistently aligned.

In an alignment of k diagonals $D_1, D_2, \ldots, D_k$, the total score S is given by the addition of all weights $w(D_i)$, from D_1 to D_k. To discover the score S, a dynamic programming based strategy is used. Consider two sequences s and t, having sizes m and n, respectively. For each pair (i,j), it will be determined all integers k with $k \leq \min(i,j)$ where the diagonal $(s_{i-k}t_{i-k}, \ldots, s_i t_j)$ beginning at position $(i-k, j-k)$ and ending in position (i, j) has a positive weight w. For each position (i, j) is defined a $score(i, j)$ for the alignment (chain of diagonals $D_1, D_2, \ldots, D_k$) in the prefixes $s[1..i]$ and $t[1..j]$.

According to Morgenstern et al. (1998), the last diagonal D_k aligned in position (i, j) is recovered by function $prec(i, j)$ (Equation (8)). The score is calculated as in Equation (7), where $\sigma(D_{i,j})$ is

Figure 5. DIALIGN consistency check: for each new significant fragment (or diagonal) calculated, a consistency check is made where the ending coordinates of a diagonal cannot overlap with the beginning coordinates of the next one


consistent
inconsistent
Scores already calculated
Antidiagonal being calculated
Diagonals above the threshold T
Consistency check
(i,j)

defined as the chain of diagonals ending at point (*i, j*) that has the highest score.

$$score(i,j) = \max\begin{cases} score(i-1,j) \\ score(i,j-1) \\ \sigma(D_{ij}) \end{cases} \tag{7}$$

$$prec(i,j) = \max\begin{cases} pre(i,j-1),\ \text{if } score(i,j) = \\ score(i,j-1) \\ prec(i-1,j), \text{if } score(i,j-1) < \\ score(i,j) = score(i-1,j) \\ D_{ij}, \text{if } score(i,j-1), score(i-1,j) < \\ score(i,j) = \sigma(D_{ij}) \end{cases} \tag{8}$$

Two dynamic programming matrices are calculated, one for scores (Equation (7)) and the other for the preceding diagonal (Equation (8)). Once these matrices are calculated, the reverse path on the *prec* matrix gives the alignment. One example of such alignment is given in Figure 6. In Figure 6(a), the subsequences belonging to diagonals are shown in gray and the aligned diagonals are shown as lines. Figure 6(b) shows the final alignment.

2.3. Sequence-Profile Alignment

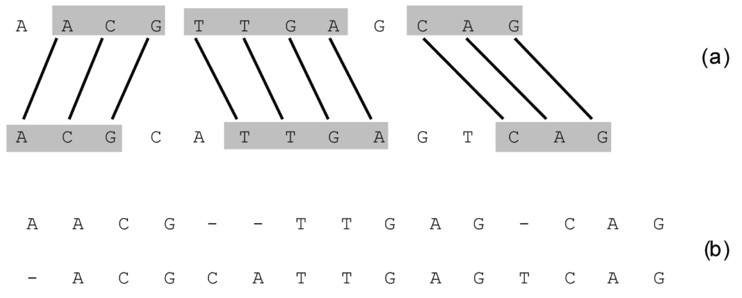
Profile analysis complements standard pairwise comparison methods for large-scale analysis of families of sequences. A sequence family is defined to be a set of sequences with similar function, similar 2D/3D structure, or a common evolutionary history (Hunter, 1993). Therefore, a newly identified sequence is often compared to several known families, in search of similarities. The comparison usually aligns the sequence to the representation of the family. This representation can be a profile, a consensus sequence or a signature (Gusfield, 1997).

We can build a multiple alignment of the sequences in a family and, based on the alignment, produce a profile representing the family. This profile keeps statistical information about the family, recording the residue distribution at each position of the multiple alignment, and describing statistically the variations in the family.

Given a sequence of interest and the profile modeling a sequence family, if the sequence-profile comparison results in high similarity, the sequence is usually identified to be member of the family. This identification is a very important step towards determining the function or/and structure of the sequence.

For instance, the PFAM database (Finn at al., 2008) is a huge collection of protein families represented as multiple sequence alignments, available via WWW servers that enable the comparison of protein sequences of interest against

Figure 6. Example of a DIALIGN alignment


A A C G T T G A G C A G
A C G C A T T G A G T C A G
(a)
A A C G - - T T G A G - C A G
- A C G C A T T G A G T C A G
(b)

protein families, in search for additional remotely homologous sequences.

Our discussion will focus on profile representations of protein families, which are based on multiple sequence alignments. Nevertheless, the same techniques are also applicable to DNA sequences.

2.3.1. Hidden Markov Models

One of the most accepted probabilistic models to do sequence-profile comparisons is based on Hidden Markov Models (HMMs). HMMs, a statistical model based on Markov processes, have been applied to other research areas, such as speech recognition (Rabiner, 1989).

A profile HMM models a sequence family, representing the common similarities among the sequences in the family as discrete states, each one corresponding to an evolutionary possibility such as residue insertions or deletions, or matches between them.

Once a profile HMM is built to model a given sequence family (for instance, from the multiple alignment of the sequences in the family), it is commonly used to score how well a new sequence fits the family profile. For example, one could build a model for a number of proteins in a family, and then match sequences in a database to that model in order to try to find other family members.

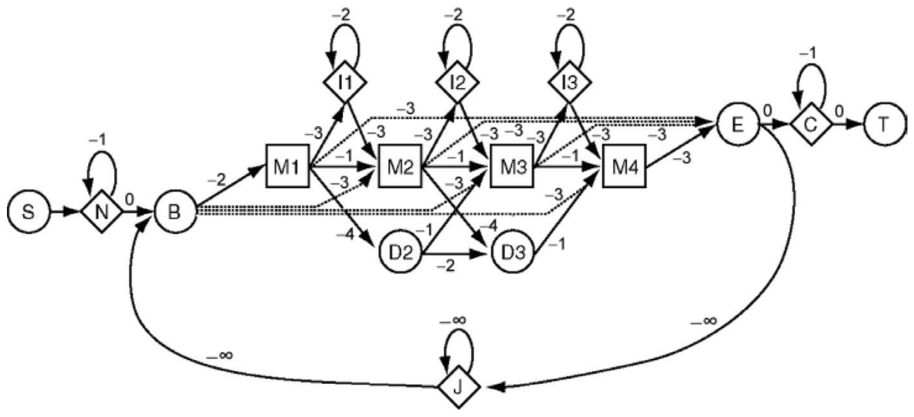
Krogh et al. (1994) proposed a profile HMM structure for modeling protein families, consisting of match (M), insert (I), and delete (D) states. This HMM usually has one state M for each consensus position in the multiple alignment of the sequences in the family. Each M state aligns to (emits) a single residue, with a probability score that is determined by the frequency that residues have been observed in the corresponding position of the multiple alignment. Therefore, each M state has 20 probabilities for scoring the 20 amino acids.

The states I and D model gapped alignments, that is, alignments including residue insertions and deletions. Each I state also emits a residue and has 20 probabilities for scoring the 20 amino acids. The D states do not emit any residues.

Besides the emission probabilities, there are transition probabilities associated to each transition from one state to another, that are set to capture specific information about each position in the multiple alignment of the family. The profile HMM allows position dependent insertion and deletion penalties, with different probabilities for entering different I and D states, modeling that certain regions of the multiple alignment tolerate insertions and deletions more than others. Also, we may have different transition probabilities for entering a particular insert state for the first time vs. staying in it, modeling affine gap penalties.

The Plan7 architecture (Eddy, 2003) is a modified profile HMM structure that augmented the simple model of Krogh at al. (1994) in order to enable global or local alignments, with respect to the model or to the sequence, and also multiple

Figure 7. A profile HMM with 4 nodes (with the Plan7 structure) and the transition probabilities



hit alignments. In addition to the original M, I and D states, the Plan7 HMM includes a set of flanking states.

The group of M, I, and D states corresponding to the same position in the multiple alignment is called a node of the HMM. The structure of a Plan7 HMM with four nodes is illustrated in Figure 7. The labels in the state transitions represent transition probabilities.

Local alignments with respect to the profile HMM are allowed by non-zero state transition probabilities from state B to internal M states, and from these states to state E. Local alignments with respect to the sequence are allowed by non-zero state transitions on the special insert states N and C. More than one hit to the HMM per sequence is allowed by transitions through the special insert state J.

2.3.2. Viterbi Algorithm

Given a query sequence and a profile HMM modeling a protein family, an alignment of the sequence to the model is an assignment of states to each residue in the sequence (represented as a path of the sequence over the states). There are many such alignments for a given sequence. Therefore, we need to find the best alignment of the sequence to the HMM, i.e., the one with the highest probability. This can be done efficiently by a dynamic programming algorithm called the Viterbi algorithm (Rabiner, 1989). The algorithm finds the best alignment for that sequence and gives its probability.

The probability of an alignment is calculated by multiplying the transition and emission probabilities at each state in the path, and can be interpreted as a similarity score. If this score is sufficiently good, we can conclude that the sequence belongs to the family.

The probability parameters in a profile HMM are usually converted to additive log-odds scores. The transition and emission probabilities are converted into logarithms, and the log-odds score of the alignment is calculated by adding up these numbers, instead of multiplying the probabilities.

The Viterbi algorithm applied to Plan7 HMMs for aligning a sequence s of length n to a profile HMM with k nodes is shown in Equations (9), in the form of a set of recurrence equations. It calculates a set of score matrices of size $n \times k$ (corresponding to states M, I, and D) and vectors of size n (corresponding to states N, B, E, J, and C).(see Figure 8).

In these equations, $M(i, j)$ is the score of the best path aligning the subsequence $s[1..i]$ to the model up to state M_j, ending with s_i being emitted by state M_j. Similarly, $I(i,j)$ is the score of the best

Figure 8. Equation 9

$$
\begin{aligned}
&M(i,0) = I(i,0) = D(i,0) = -\infty \quad \forall 1 \le i \le n\\
&M(0,j) = I(0,j) = D(0,j) = -\infty \quad \forall 1 \le j \le k\\
&M(i,j) = em(\mathrm{M}_j,s_i) + \max\begin{cases} M(i-1,j-1) + tr(\mathrm{M}_{j-1},\mathrm{M}_j)\\ I(i-1,j-1) + tr(\mathrm{I}_{j-1},\mathrm{M}_j)\\ D(i-1,j-1) + tr(\mathrm{D}_{j-1},\mathrm{M}_j)\\ B(i-1) + tr(\mathrm{B},\mathrm{M}_j)\end{cases} \quad \forall 1 \le i \le n, \forall 1 \le j \le k\\
&I(i,j) = em(\mathrm{I}_j,s_i) + \max\begin{cases} M(i-1,j) + tr(\mathrm{M}_j,\mathrm{I}_j)\\ I(i-1,j) + tr(\mathrm{I}_j,\mathrm{I}_j)\end{cases} \quad \forall 1 \le i \le n, \forall 1 \le j \le k\\
&D(i,j) = \max\begin{cases} M(i,j-1) + tr(\mathrm{M}_{j-1},\mathrm{D}_j)\\ D(i,j-1) + tr(\mathrm{D}_{j-1},\mathrm{D}_j)\end{cases} \quad \forall 1 \le i \le n, \forall 1 \le j \le k\\
&N(0) = 0\\
&N(i) = N(i-1) + tr(\mathrm{N},\mathrm{N}) \quad \forall 1 \le i \le n\\
&B(0) = tr(\mathrm{N},\mathrm{B})\\
&B(i) = \max\begin{cases} N(i) + tr(\mathrm{N},\mathrm{B}) \quad \forall 1 \le i \le n\\ J(i) + tr(\mathrm{J},\mathrm{B})\end{cases}\\
&E(i) = \max_{\forall 1 \le j \le k}\{\, M(i,j) + tr(\mathrm{M}_j,\mathrm{E}) \,\} \quad \forall 1 \le i \le n\\
&J(0) = -\infty\\
&J(i) = \max\begin{cases} J(i-1) + tr(\mathrm{J},\mathrm{J}) \quad \forall 1 \le i \le n\\ E(i) + tr(\mathrm{E},\mathrm{J})\end{cases}\\
&C(0) = -\infty\\
&C(i) = \max\begin{cases} C(i-1) + tr(\mathrm{C},\mathrm{C}) \quad \forall 1 \le i \le n\\ E(i) + tr(\mathrm{E},\mathrm{C})\end{cases}\\
&\text{similarity\_score} = C(n) + tr(\mathrm{C},\mathrm{T})
\end{aligned}
\tag{9}
$$

path ending with s_i being emitted by I_j, and $D(i,j)$ is the score of the best path ending in state D_j. The emission probability of the residue s_i at state$_1$ is denoted by $em(state_1,s_i)$, while $tr(state_1,state_2)$ represents the transition cost from state$_1$ to state$_2$.

As a result, the Viterbi algorithm finds the best (most probable) alignment and its score for the query sequence with the given model. The best alignment is represented as a path through the HMM, with a sequence of states that maximizes the probability of the query sequence being observed. The similarity score of the best alignment is given by $C(n) + tr(C,T)$. The best alignment itself can be obtained by tracking back on the Viterbi variables. This alignment can be used to add the sequence into the multiple alignment of the family.

This algorithm has $O(nk)$ time complexity. However, from Equations (9) we can see that it requires more computations per cell of the dynamic programming matrices than the pairwise sequence comparison algorithms.

Given the profile HMM of Figure 7, the transition scores are shown in the figure, labeling the state transitions, and the emission scores for the M and I states are shown in Table 1. Figure 9 shows the score matrices and vectors computed by the Viterbi algorithm, when comparing the query sequence ACYDE to that profile HMM ($n = 5$ and $k = 4$). The best alignment has the similarity score of 22 and corresponds to the path (S,−)→(N,−)→(B,−)→(M1,A)→(M2,C)→ (I2,Y)→(M3,D)→(M4,E)→(E,−)→(C,−)→(T,−), which is shaded in Figure 9.

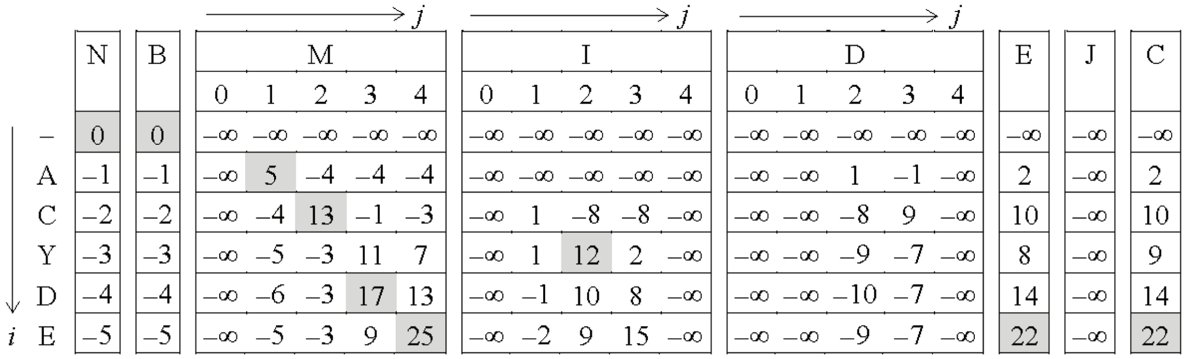
2.3.3. HMMER

The HMMER program suite developed by Eddy (2003) is a widely used software implementation of profile HMMs for biological sequence analysis. It is composed of several programs and can be

Table 1. Emission scores of amino acids for M and I states of profile HMM of Figure 7

State	A	C	D	E	F, I, L, M, V, W	G, K, N, P, S	H, Q, R, T	Y
M1	7	−1	−1	1	−1	2	1	−1
M2	−1	9	−1	1	−1	2	1	−1
M3	−1	−1	8	2	−1	2	1	−1
M4	−1	−1	3	9	−1	2	1	−1
I1	−1	−1	0	1	−1	0	1	2
I2	−1	−1	0	1	−1	0	1	2
I3	−1	−1	0	1	−1	0	1	2

Figure 9. Score matrices and vectors of the Viterbi algorithm for the comparison of the sequence ACYDE against the profile HMM of Figure 7: shaded cells represent the best alignment



	N	B	M					I					D					E	J	C
			0	1	2	3	4	0	1	2	3	4	0	1	2	3	4			
–	0	0	–∞	–∞	–∞	–∞	–∞	–∞	–∞	–∞	–∞	–∞	–∞	–∞	–∞	–∞	–∞	–∞	–∞	–∞
A	–1	–1	–∞	5	–4	–4	–4	–∞	–∞	–∞	–∞	–∞	–∞	–∞	1	–1	–∞	2	–∞	2
C	–2	–2	–∞	–4	13	–1	–3	–∞	1	–8	–8	–∞	–∞	–∞	–8	9	–∞	10	–∞	10
Y	–3	–3	–∞	–5	–3	11	7	–∞	1	12	2	–∞	–∞	–∞	–9	–7	–∞	8	–∞	9
D	–4	–4	–∞	–6	–3	17	13	–∞	–1	10	8	–∞	–∞	–∞	–10	–7	–∞	14	–∞	14
E	–5	–5	–∞	–5	–3	9	25	–∞	–2	9	15	–∞	–∞	–∞	–9	–7	–∞	22	–∞	22

used to build database search models from pre-existing alignments. HMMER software uses the Plan7 HMM structure in order to produce global or local alignments, with respect to the model or to the sequence, and also multiple hit alignments.

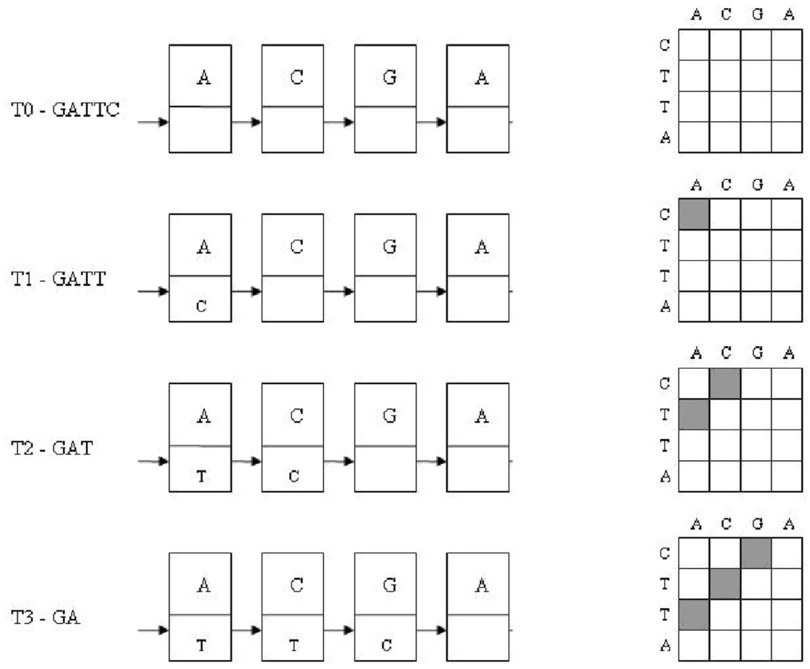
HMMER can take multiple alignments of sequence families and build the profile HMMs representing them. In particular, the program *hmmsearch* searches a sequence database for matches to a profile HMM, while the program *hmmpfam* searches a HMM database for matches to a query sequence. Both programs use the Viterbi algorithm to perform the sequence-profile comparison and generate the alignment of the sequence to the HMM and the corresponding similarity score. Experiments have shown that the function that implements the Viterbi algorithm is the most time consuming kernel of these programs (Sun et al., 2009).

3. FPGA-BASED ACCELERATORS FOR PAIRWISE COMPARISON

3.1. General Overview of the Proposed Solutions

In the literature, there are many proposals of FPGA-based architectures to accelerate pairwise sequence comparison applications by calculating the similarity matrix anti-diagonals in hardware, taking full advantage of the parallelism inherent in this computation. In this approach, an array processor with *N* elements is used, where each

Figure 10. A 4-element systolic array to calculate the dynamic programming matrix



element is capable of calculating one matrix score per turn. Thus, when the wavefront is being processed by all elements, one processor array with *N* elements can generate *N* scores at a time.

Usually, systolic processor arrays (Kung, 1982) are used, where the processing elements operate in a lock-step basis. The seminal work of Lipton & Lopresti (1985) proposed a successful implementation of Smith-Waterman in a systolic array. Since then, variations of this design have been used in several different proposals. However, the *O(mn)* space complexity of the SW algorithm is a great restriction. For this reason, most of the hardware solutions do not store the entire similarity matrix, obtaining only the similarity score.

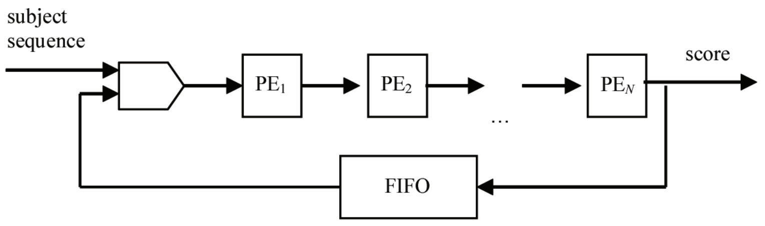
Usually, the smallest sequence being aligned, called query sequence, is stored on the computing elements. The other sequence, called database sequence, can be of any size, since it "passes" through the FPGA.

At time T0 (Figure 10), the elements of the query sequence (A, C, G, and A) are already stored in the processing elements (PEs). At T1, the first processing element compares A to C, using, for instance, the recurrence relation (2). At T2, two comparisons are made simultaneously (A to T and C to C) by the first and second processing elements, respectively. At T3, three cells are calculated in parallel, and so on. The wavefront is said to be at full processing when all PEs are processing simultaneously.

Frequently, the size of the query sequence is greater than the number of PEs contained in the FPGA. In this case, a partitioning technique is needed. To break query sequences, it is necessary to keep the scores of the last calculated column in the FPGA memory (usually a FIFO) to allow new scores to be calculated. Figure 11 illustrates the systolic array design with partitioning.

Some designs avoid the use of a FIFO by putting many query bases in the same computing element. The drawback of this approach is that to put more bases in each cell requires more registers per element and thus decreases the maximum

Figure 11. Systolic array composed by N PEs, with partitioning. The last column of each phase is stored in a FIFO and provided to the PE array as input to the next step



number of computing elements in the systolic array.

3.2. Some Early Approaches

In the last two decades, many proposals were made to execute pairwise sequence comparison applications in specific hardware. In this section, we will present briefly some of the relevant proposals made from 1992 to 2003.

A bidirectional systolic array was proposed by Hoang & Lopresti (1992). In this design, the database and query sequences are fed at the same time into the processing element array, in opposite sides and directions. Computation starts when the first characters of both sequences meet in the middle of the systolic array and continues until every character is compared. The sequences are processed twice and the alignment is output as a set of 2-bit representations that indicate (a) the alignment of both characters, (b) a gap in the first sequence or (c) a gap in the second sequence. Ten thousand alignments of 100-nucleotide sequences were performed in the SPLASH board, that is composed of 32 FPGA Xilinx XC3090. The total of 248 PEs was obtained.

Yamagutchi et al. (2002) proposed a unidirectional systolic array to compute the Smith-Waterman score and retrieve the alignment (Section 2.2.2), using the basic design illustrated in Figure 10. Sequence partitioning is achieved by a pipelined approach that calculates each diagonal in two steps: superior and inferior part. The similarity matrix is stored in the FPGA and it is used to retrieve the alignment. Using the FPGA Xilinx XCV2000E, the alignment between sequences of 1KBP and 4KBP was obtained by 144 PEs. Speedups of 50 and 330 were obtained for the alignment retrieval and the score computation, respectively, when compared to a software implementation.

A systolic array architecture to run the SW algorithm at the OSIRIS board was proposed by Puttegowda et al. (2003). This board contains two FPGAs: one interfaces with the host machine whereas the other executes user-programmed hardware designs. Reconfiguration is used to generate custom circuit logic with the dynamic programming parameters and the query sequence. Four systolic arrays were implemented in the same FPGA, which enabled four comparisons to be made simultaneously. Sequence partitioning is achieved by using more than one PE array. In this design, the similarity matrix is calculated but the edit distance is obtained, instead of the SW score. A speedup of 500 was obtained when comparing sequences with up to 1750 nucleotides with a genomic database, with respect to a software solution.

3.3. Customized Smith-Waterman Accelerator

Oliver et al. (2005) proposed the use of a linear array of PEs to solve the pairwise SW-based local alignment problem with linear or affine gap functions. The proposed architecture computes the whole similarity matrix and outputs the highest score. Switching between the gap functions is done by static reconfiguration. Using reconfiguration, the size of the operands and other system parameters can also be specified for optimized circuit generation.

Moreover, the systolic array PEs are placed in the FPGA according to a zig-zag pattern, which provides a very good space occupation. Depending on the sizes of the sequences being compared, the FPGA can be reconfigured to compare more than one sequence at a time.

Several elements that compose the query sequence can be stored in the same PE. A FIFO is designed to accommodate the query sequence elements, as well as the substitution matrix column and the database sequence itself.

The architecture was programmed in Verilog and synthesized for the Virtex II XC2V6000 FPGA board. Query sequences of size that range from 1 to 1004 amino acids were compared with the Swiss-Prot database on a FPGA prototype with 252 PEs. Speedups of approximately 170 and 125 were achieved for the linear and affine gap functions, respectively, when compared to an optimized software program. This design imposes restrictions on the size of the query and database sequence. For the targeted FPGA, only sequences with less than 8192 amino acids can be compared.

3.4. Accelerator to Retrieve Highest Score and its Coordinates

Boukerche et al. (2007a) proposed a systolic array that computes the Smith-Waterman dynamic programming matrix (Section 2.2.2) for DNA sequences in linear space. A linear gap model is employed. The basic design discussed in Section 3.1 was used, where each PE calculates one column of the dynamic programming matrix. If the size of the query sequence is greater than the total number of PEs, sequence splitting is done and the last column calculated is stored in the FPGA memory (Figure 11).

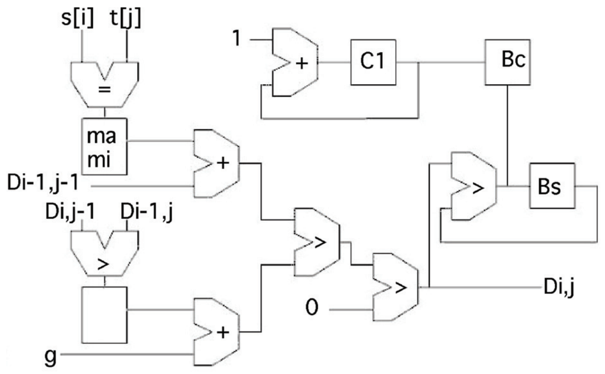
The particularity of this proposal is that it provides as output not only the highest score but also the coordinates in the dynamic programming matrix where the score occurs. These coordinates determine the position where the optimal score ends. In many parallel software-only approaches (Batista et al., 2008), this is the first phase of the SW algorithm. Having the highest score and its ending position, reprocessing can be done over the reverses of the sequences, in order to retrieve the beginning coordinates and the actual optimal alignment itself (Section 2.2.3).

Figure 12 presents the circuit executed in each PE. With the exception of the top right elements (*Bc, C1* and *"+"*), slight variations of this design are employed in most of the FPGA accelerators that implement SW.

To calculate each cell *Di,j* in the matrix, three cells are needed (Equation (2)): *Di,j–1, Di–1,j* and *Di–1,j–1* (Figure 12). In each cycle, values *Di-1,j-1* and *Di,j-1*, which are stored in registers, and value *Di-1,j*, which is transmitted from the left PE, are used to calculate *Di,j*. Two bases are compared: a query base *s[i]*, that is fixed in the PE and a database base *t[j]* that comes from the left PE. If the bases are equal, it means that a match occurred and the coincidence punctuation *ma* is used. Otherwise, it is a mismatch and the substitution penalty *mi* is used. In parallel, values *Di,j-1* and *Di-1,j* are compared and the greater one is added to the gap penalty *g*. These values are compared and the greater one is compared to zero, as shown in Equation (2).

To calculate the best score, the new *Di,j* score is compared to the value stored in register *Bs* (Best score in that column so far). If the *Di,j* score is greater than *Bs*, it replaces the old value. To recover the anti-diagonal and therefore the row where the best score occurs, register *C1* is utilized. It is incremented by one each time a new score is calculated. If the current score is greater than the one stored in *Bs*, the writing on register *Bc* is

Figure 12. Circuit executed by each PE including the dynamic programming cell calculation and the highest score and its coordinates computation (Adapted from Boukerche et al., 2007a)



enabled and its value is replaced by the current value of *C1*. Therefore, *C1* stores the row where *Bs* occurs and, with the relative position of the PE itself, the column is obtained.

A prototype of this architecture was synthesized for the FPGA Xilinx XC2VP70. With this prototype, a speedup of 246.9 was achieved over an optimized C program, when comparing synthetic sequences of 10MBP (Millions of Base Pairs) and 10KBP (10 Kilo Base Pairs).

3.5. Pipelined Accelerator with Uneven Stages

Zhang et al. (2007) proposed the use of a unidimensional systolic array to execute the SW algorithm for protein and DNA sequences with both linear and affine gap function, i.e., SW and Gotoh gap penalties, respectively (Sections 2.2.2 and 2.2.3). As a result, the highest score is output. A partition technique is also used that compares parts of the query sequence with the entire database sequence in each phase. The elements of the last column computed in each phase are stored in the FPGA internal memory, as illustrated in Figure 11.

In order to achieve better performance, the authors propose a pipelined control mechanism, with uneven stage latencies. Four clocks with the same frequency are used, but there are different phase delays between clocks. Also, intra-PE optimizations were proposed to reduce the number of *max* operations and to reduce the space needed to store the substitution matrix containing the mismatch penalties between every residue pair.

The proposed architecture was synthesized, targeted to run in the XD1000 platform. In this platform, the main processor is an AMD64 Opteron that is connected through HyperTransport to an Altera Stratix II FPGA. The main processor and the FPGA are located in the same board and the FPGA acts as a co-processor. A PE array of 384 elements that work at 66.7 MHz was implemented. Compared to an optimized C program, the proposed architecture achieved a speedup of 249.52 when comparing two sequences of 65,536 amino acids.

3.6. DIALIGN Accelerator

Boukerche et al. (2007b) proposed an FPGA-based accelerator for the DIALIGN algorithm (Section 2.2.4). The basic design discussed in Section 3.1 was used. If the query sequence has more elements than the PE array, the last column of each phase is stored in the FPGA internal memory (Figure 11).

The main difference between this proposal and the Smith-Waterman accelerators discussed so far is that the design of a PE that executes DIALIGN is totally different from the design of a PE that executes Smith-Waterman, with or without affine gap. This happens because the DIALIGN recurrence relations (Equations (7) and (8)) are more complex since they include conditional statements.

In order to obtain a high performance implementation of the DIALIGN algorithm in a processor array, some simplifications were made. First, the natural logarithm, that is used in the original DIALIGN equation, was replaced by log2. Second, the fragments are now defined to contain only matches. As soon as a mismatch is detected, the fragment is ended. This second simplification eliminates the dependence on the *k* last elements and, thus, makes it possible to use the generic processor array shown in Figure 10, in the DIALIGN implementation.

Also, this design includes a handshaking protocol between every two neighbor processing elements. Therefore, an asynchronous design is used instead of the traditional systolic approach. Inside the PEs that implement the DIALIGN recurrence relations, the size of the possible paths varies considerably and the path taken depends on the values being compared and on the best chain of fragments obtained so far. In this scenario, a systolic approach would have been targeted to the size of the largest path, limiting considerably the performance gains.

The proposed design was implemented in SystemC and compiled to Verilog by Forte (www.forteds.com). A processor array of 200 PEs was synthesized for the FPGA Altera Stratix 2 EP2S180F1508I4. A maximum speedup of 383.41 was achieved, when compared to an optimized software implementation.

3.7. Parameterized FPGA-Based Skeleton

Observing that there are many FPGA-based accelerators for pairwise sequence comparison that present the same general structure but differ in some characteristics such as type of sequence compared, recurrence relation and/or output produced, Benkrid et al. (2009) proposed a parameterized skeleton for sequence comparison accelerators. In this work, the user can specify some parameters and a program is generated in Handel-C that executes the target architecture. The accelerators generated are systolic processor arrays that calculate the anti-diagonals in parallel (Figure 10).

Seven parameters do exist: sequence type (DNA, RNA or protein), query sequence length, maximum database sequence length, substitution matrix, gap penalty (linear or affine), matching algorithm (global, local or overlapped matching) and match score threshold. The query sequence length is used to determine the number of PEs. If this length is greater than a given value, a partition technique is implemented, where the last column is stored in the FPGA memory (Figure 11).

Using the proposed skeleton, an implementation was made of the SW with affine gap algorithm (Section 2.2.3), comparing proteins with the BLOSUM50 substitution matrix. The FPGA used was the XC2VP100-6 Virtex-II Pro. In this implementation, the systolic array had 135 processing elements, running at a clock frequency of 40MHz. When comparing a query sequence of 362 residues with a subset of the Swiss-Prot database, a speedup of 62 was achieved.

3.8. Comparative Overview

Table 2 presents a comparative overview of the proposals discussed in Sections 3.2 to 3.7.

The pairwise sequence comparison algorithm is shown in the second column (Table 2). As it can be seen, most of the papers implement the Smith-Waterman (SW) algorithm (Section 2.2.2). The more recent papers implement also the Gotoh algorithm (Section 2.2.3). Zhang et al. (2007), Oliver et al. (2005), and Benkrid et al. (2009) provide some kind of framework where different dynamic programming algorithms for pairwise sequence comparison can be implemented in a

Table 2. Comparative overview of the pairwise sequence comparison accelerators

Paper	Algorithm	Number of PEs	Clock rate (MHz)	Max query size compared	Speedup reported	Output
Hoang et al. (1992)	SW	248	Not reported	100	290.0	alignment
Yamagutchi et al (2002)	SW	144	40.0	4,096	330.0 / 50.0	score, alignment
Puttegowda et al (2003)	SW (edit distance)	4×1750	180.0	1,750	500.0	edit distance
Oliver et al. (2005)	SW/Gotoh	252	55.5	1,428	170.0	score
Boukerche et al (2007a)	SW	100	174.7	10,000	246.9	score, coordinates
Zhang et al. (2007)	SW/Gotoh	384	66.7	65,536	249.5	score
Boukerche et al (2007b)	DIALIGN	200	74.4	169,786	383.4	score
Benkrid et al. (2009)	SW/Gotoh/ NW/ overlap	135	40.0	362	62.0	score

simpler way. Only the accelerator proposed by Boukerche et al (2007b) implements the algorithm DIALIGN (Section 2.2.4).

The number of PEs (column 3 in Table 2) varies a lot and depends on both the complexity of each processing element and the capacity of the FPGA board used. In these designs, the number of PEs range from 100 to 384. An impressive result is obtained by Puttegowda et al. (2003), that were able to generate four systolic arrays with 1750 PEs each. This huge number of PEs can be explained by the algorithm implemented (edit distance), which is simpler than SW. Also, the best score was not stored, simplifying further the design of the PEs.

Column 4 presents the clock rate of each accelerator. With the exception of Boukerche et al. (2007a) and Puttegowda et al. (2003), the clock rates range from 40.0 to 74.4 MHz. It is surprising to see very similar clock rates in these accelerators since there is a gap of 27 years between the first accelerator discussed in this section and the last one. It must be noted, however, that, even though the capability of the FPGA boards have increased substantially from 1992 to 2009, the complexity of each PE has also increased. For instance, the designs of Hoang et al. (1992), Yamagutchi et al. (2002), and Puttegowda et al. (2003) compare DNA sequences and the other SW designs, with the exception of Boukerche et al. (2007a), compare proteins. This makes a big difference in the design of each PE since a substitution matrix of size 20 × 20 must be stored, for the protein case. Also, most of the recent designs implement the Gotoh algorithm, that calculates three similarity matrices, instead of the one traditional SW matrix. This, of course, adds complexity to the design of the PE, increasing its size. In the same way, the simplicity in the design of each PE was a very important factor that determined the higher clock rates achieved by Puttegowda et al. (2003) and Boukerche et al. (2007a).

The size of the largest query sequence compared is illustrated in the fifth column. With the exception of Hoang & Lopresti (1992), all accelerators employ optimized partition techniques. This allowed query sequences of more than 10,000 elements to be compared in Boukerche et al. (2007b) and Zhang et al. (2007). Even though Benkrid et al. (2009) does use query sequence partition, their results used this feature in a restricted way, comparing small sequences of up to 362 amino acids.

Column 6 of Table 2 presents the speedup reported by each paper. In all cases, the FPGA implementations were compared to optimized software-only programs. Since different FPGA boards and desktops were used, a direct comparison among the accelerators cannot be made. But the speedup is an important indicator of the potential gains of the accelerators. The reported speedups range from 62 to 500 and, with the exception of Benkrid et al. (2009), all the accelerators were able to achieve speedups higher than 100, when compared to software. Impressive speedups, higher than 200, were achieved for most accelerators. In Benkrid et al. (2009), flexibility was obtained at the expense of performance. Even in this case, a speedup of 62 can be considered a good cost/benefit trade-off.

The output of each accelerator is shown in column 7. As it can be seen, the older approaches (1992 and 2002) provided the alignment as output. Nevertheless, in order to retrieve the alignment, the similarity matrix was stored in the FPGA internal memory and that severely restricted the size of the sequences being compared. Therefore, from 2003 to now, the pairwise sequence comparison accelerators act as filters, providing only the score as output. For the cases where the score is higher than a given threshold, the software will re-calculate the similarity matrix and generate the alignment using a software-only approach. In order to reduce this reprocessing time, the coordinates that determine the end of the optimal alignment are also retrieved by Boukerche et al. (2007a).

4. FPGA-BASED ACCELERATORS FOR SEQUENCE-PROFILE COMPARISON

4.1. General Overview of the Proposed Solutions

Since the Viterbi algorithm is the most time consuming part of the sequence-profile comparison, multiple attempts to optimize it have been made, including the implementation of FPGA-based accelerators. Studies have also shown that most of the execution time of the HMMER suite comparison programs is spent in processing poor scoring, and consequently non significant, sequences (Maddimsetty et al., 2006).

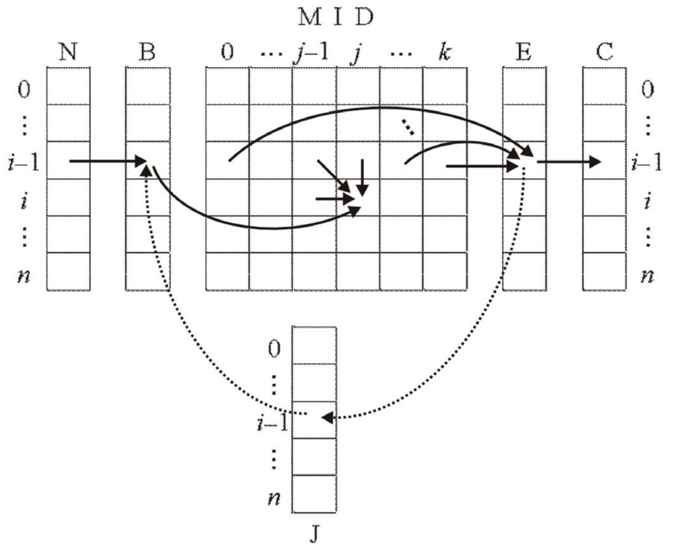
In order to compute only the best alignment score it is not necessary to store the whole Viterbi matrices and vectors. In fact, only two rows of these structures are needed. On the other hand, to produce the best alignment a memory space of size $O(nk)$ is required, for a profile HMM with k nodes and a sequence of length n. This usually prevents the implementation of the alignment generation operation in hardware, given the memory limitations of current FPGAs and the long length of the biological sequences.

Therefore, most sequence-profile comparison solutions apply a first phase filter in order to discard poor scoring sequences, prior to full processing. The filter phase calculates only the similarity score (through the Viterbi algorithm) and is implemented in hardware, using FPGAs. Then, if the score is above a given threshold, the sequence is reprocessed in software in order to generate the corresponding alignment. In this reprocessing, the Viterbi algorithm is executed again, comparing the selected sequence against the profile HMM (using the HMMER software, for instance), but this time keeping information to produce the alignment.

Considering that hardware implementations can exploit the inherent fine- and coarse-grained parallelism of the algorithm and that only a small fraction of the sequences (the ones with high similarity) will be reprocessed in software, the FPGA filter can achieve performance gains when compared to the software-only implementation.

Figure 13 shows (with arrows) the data dependences for computing the cells (i,j) of the M, I, and D matrices in the Viterbi algorithm. These dependences determine which operations of the algorithm can be performed in parallel. The scores $M(i,j)$, $I(i,j)$, and $D(i,j)$ can be computed in parallel, since there are no data dependences between them. State J induces a feedback loop in the HMM (shown with dashed arrows in the figure) and creates data dependences which impose that the

Figure 13. Dependences of the Viterbi algorithm



cells in each matrix must be computed one at a time in row-major order, limiting the parallelism.

In order to increase the available parallelism, most FPGA-based accelerators for the Viterbi algorithm (Benkrid et al., 2008; Oliver et al., 2009; Jacob et al., 2007) eliminate the state J of the Plan7 HMM. With this strategy, all cells in the same anti-diagonal of the Viterbi matrices become independent, yielding significant fine-grained parallelism. Therefore, a systolic array of processing elements (PEs) can be implemented so that each HMM node is mapped to one PE. Each PE calculates the score for the corresponding column of the dynamic programming matrices. At each time step, all PEs compute the cells in an anti-diagonal simultaneously, in a way similar to the approach used in pairwise sequence alignment (Section 3.1). However, the more resource intensive computations for each cell of the matrices, allied to the large number of parameters in a profile HMM, make it a challenging task to efficiently implement the Viterbi algorithm in hardware.

The state J removal strategy eliminates the Plan7 HMM ability to find multiple hit alignments such as several matches of subsequences of the query sequence to the profile HMM. It can result in loss of sensitivity for database searches, since sequences that would produce multiple hits, when limited to only one hit, can obtain a low score and be discarded. In order to match the sensitivity of a full Plan7 HMM with this simplified model, the threshold for accepting an alignment score is relaxed, at the cost of increasing the software reprocessing time.

The solution proposed by Eusse et al. (2009) uses a similar systolic array structure, eliminating the state J. However, besides computing the score, the PEs compute the alignment limits, in order to reduce the software reprocessing time. Maddimsetty et al. (2006) and Sun et al. (2009) also use the systolic array structure, but propose different techniques in order to reduce or correct the error induced by the J state elimination.

A different approach is presented in Oliver et al. (2008), where several sequences are compared to the same profile HMM at the same time, exploiting a coarse-grained parallelism. In this strategy, each PE implements the Viterbi algorithm for a complete Plan7 HMM, including the state J.

Derrien & Quinton (2007) proposed a mathematical model to represent the Viterbi algorithm applied to the comparison of multiple sequences against the same profile HMM, exposing the data dependences this problem presents. Using space-time mappings of the problem on the polyhedral model, they derive different parallelization schemes of the Viterbi algorithm for FPGAs.

The performance of most FPGA-based accelerators for the sequence-profile comparison is reported using the throughput measure CUPS (Cell Updates per Second), which indicates how many cells of the dynamic programming matrices are computed in one second. CUPS represent the peak performance that the hardware accelerator can achieve and usually does not include data communication time or initialization time. Therefore, it should be considered an upper-bound of the actual performance.

In the following sections, we describe in detail some solutions proposed to implement sequence-profile comparison on FPGAs. Section 4.8 presents a comparative overview of these solutions.

4.2. Systolic Array

Oliver et al. (2009) and Benkrid et al. (2008) proposed FPGA-based accelerators for the Viterbi algorithm in order to obtain the best alignment score between a profile HMM and a sequence. Oliver et al. (2009) use the HMM model of Krogh at al. (1994) consisting only of M, I, and D states (Section 2.3.1), while Benkrid et al. (2008) use the Plan7 HMM, but eliminate the state J. Therefore, their designs do not produce multiple hit alignments.

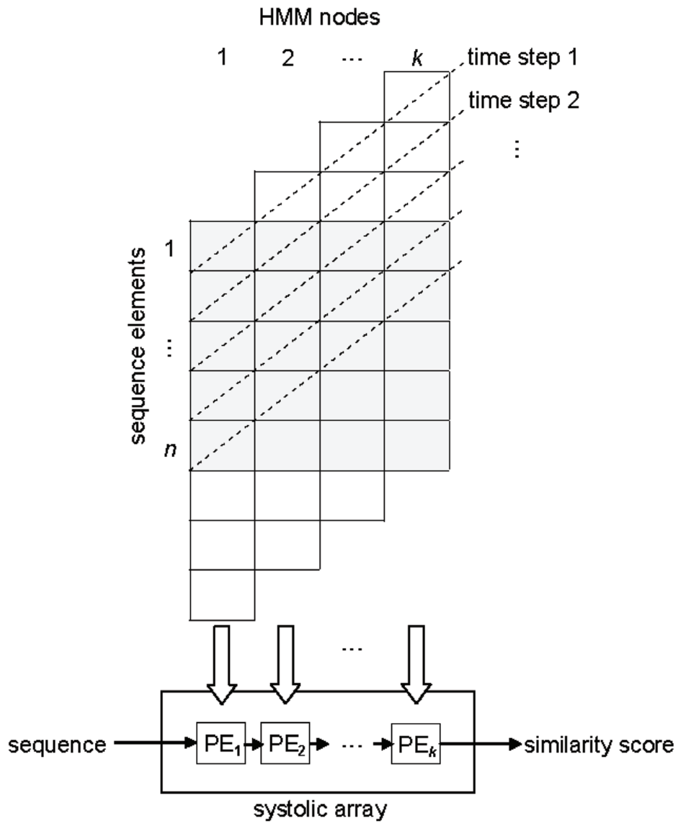
Both accelerators use the systolic array architecture described in Section 4.1, composed of a number of PEs interconnected in a linear structure. Each HMM node is mapped to one PE, which calculates the cells in the corresponding column of the matrices of the Viterbi algorithm. At each time step, each PE calculates one cell, in a way that all cells in an anti-diagonal of the matrices are computed in parallel.

Figure 14 illustrates how the Viterbi matrices columns are mapped to the PEs in the systolic array, considering that the number *k* of profile HMM nodes is equal to the number of implemented PEs. In the figure, each anti-diagonal shows the cells that are computed simultaneously, at each time step. The shaded cells are calculated by their corresponding PEs, while a blank cell corresponds to an idle PE at that time step.

The systolic array is filled gradually as the query sequence elements are inserted until there are no idle PEs left. The sequence is processed, flowing through the systolic array, from left to right, one residue at each time step. The systolic array is also emptied gradually, until there are no more cells to compute, and then, it outputs the best alignment score. Assume we are aligning a sequence of length n to a profile HMM with k nodes on a systolic array of k PEs. Then, it takes $n+k-1$ time steps to completely shift the sequence through the array and to compute the alignment score with the Viterbi algorithm. Note that the structure of this design is very similar to the one presented for the pairwise sequence comparison accelerators (Section 3.1).

The parameters that define the particular profile HMM configuration are loaded in the systolic array, as a preprocessing step: the transition and emission probabilities of states M_j, I_j, and D_j are stored into PE_j, using memories or lookup tables of the FPGA. Each PE only interacts with its two adjacent neighbors. PE_j receives the values $M(i, j-1)$, $I(i, j-1)$, $D(i, j-1)$ and the residue s_i from its left neighbor PE_{j-1}, and sends the values $M(i, j)$, $I(i, j)$, $D(i, j)$ it calculated and the input residue to the right neighbor PE_{j+1}. All other required values are stored locally. The general structure of a PE (corresponding to HMM node j) is illustrated in Figure 15, with adders and maximum operators for computing the scores corresponding to states M, I, and D, according to the Viterbi algorithm (shown in Equations (9)). The design of the PE

Figure 14. Mapping of the columns of the dynamic programming matrices to the PEs in the systolic array

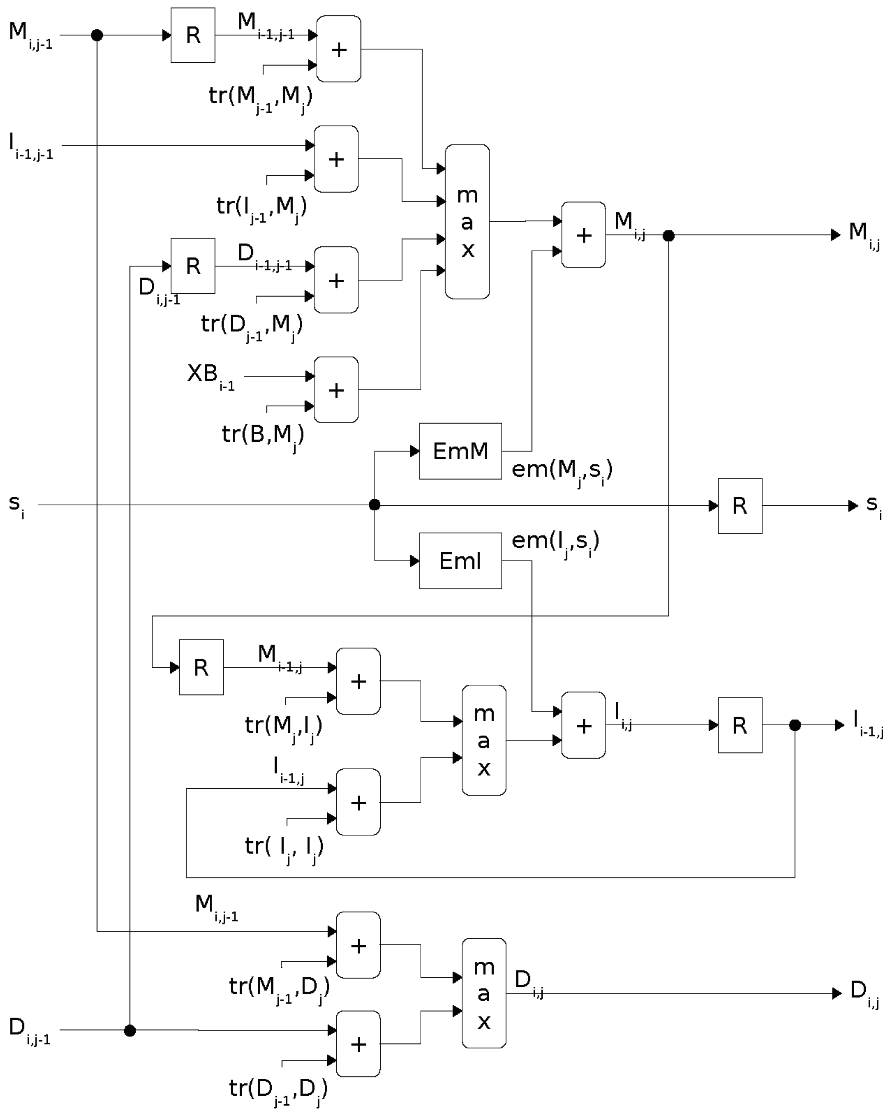


may also include the computation of the scores corresponding to states B and E.

Ideally, there would be one PE in the systolic array for each HMM node. In practice, this rarely happens, since the size of commercial FPGAs is limited and it is not practical to implement a system with a number of PEs that is equal to the number of nodes of the longest profile HMM in the databases. Since the length of the profile HMMs may vary, the computation must be partitioned to fit the fixed size systolic array.

The solution is to split the computation into several passes, similarly to the strategy used in pairwise sequence alignment (Section 3.1). Each pass computes a set of *N* adjacent columns of the dynamic programming matrices, where *N* is the maximum number of PEs that fits into the target FPGA. In the first pass, the first *N* nodes of the profile HMM are assigned to the systolic array and the corresponding emission and transition probabilities are loaded. The entire query sequence is shifted through the array of PEs and the scores are calculated for the first set of adjacent columns. The *M*, *I*, and *D* scores computed by the last PE in each time step are output and stored, since they are the input to the next pass and will be consumed by the first PE. First-In First-Out (FIFO) memories are included in the implementation to store these partial results between passes. In the second pass, the next *N* nodes of the profile HMM are loaded into the systolic array. The sequence is shifted again through the array of PEs and the scores are computed for the second set of adjacent columns. This process is repeated until the end of the profile HMM is reached. Figure 16 shows the

Figure 15. Basic structure of a PE from the systolic array: em(state$_1$,s$_i$) is the emission probability of amino acid s$_i$ at state$_1$ and tr(state$_1$,state$_2$) is the transition cost from state$_1$ to state$_2$

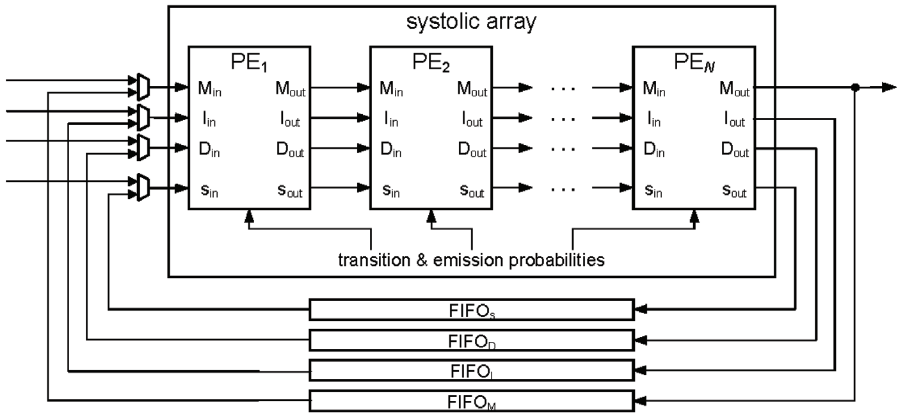


systolic array structure with *N* PEs and the FIFO memories that enable multiple passes.

The memory requirements for storing the transition and emission scores and implementing the FIFOs impose restrictions on the maximum number of PEs that can fit into the device, the maximum HMM length and the maximum sequence size.

Both Benkrid et al. (2008) and Oliver et al. (2009) implementations permit supplying the systolic array with a continuous stream of query sequences to be aligned to the profile HMM. The sequences are separated by a delimiter symbol that does not belong to the amino-acid alphabet. The purpose of this symbol is to define the start and finish boundaries of the sequences, so that each PE knows when it has to initialize its internal

Figure 16. Systolic array with N PEs and FIFOs to enable multiple passes



structures. After the last residue of a sequence enters the array, the first residue of the following sequence can be input in the next time step, after the delimiter symbol. Thus, all subject sequences of the database can be pipelined with only one time step delay between two consecutive sequences.

The systolic array proposed by Benkrid et al. (2008) has 90 PEs and is capable of comparing a sequence with 1024 residues against a 1440 nodes profile HMM. This solution was designed using Verilog and synthesized into a Virtex-II Pro 2VP100 FPGA with a 100MHz clock frequency, yielding a maximum performance of 5.2 GCUPS (GigaCUPS). Oliver et al. (2009) implemented their accelerator in Verilog, targeting the Xilinx Virtex-II XC2V6000 FPGA with a clock frequency of 74MHz. Their array has 72 PEs to process profile HMMs with up to 236 nodes against sequences of 8192 residues, achieving a performance of 3.95 GCUPS.

4.3. Systolic Array with Pipelined PE and Interleaved Sequences

Jacob et al. (2007) proposed a hardware accelerator for the Viterbi algorithm, also using the systolic array structure and eliminating the state J, as the works described in the previous section. The main contribution of their approach is that they divide each PE into pipeline stages, in order to increase the clock frequency of the design and, consequently, improve the performance of the accelerator.

The throughput of a FPGA-based implementation can be increased by pipelining its datapath, in order to reduce the clock period of the design. However, the data dependences of the Viterbi algorithm make it impossible to pipeline the PEs, while processing the cells of a single sequence-profile comparison. For example, assume the systolic array has pipelined PEs, each with several stages. If the anti-diagonal d enters the first stage of the PEs at cycle c, then at cycle $c + 1$ the computation for the cells in the anti-diagonal d proceeds to the second stage of the PEs. At the same cycle, the anti-diagonal $d + 1$ cannot enter the first stage, since it depends on the cells in d, which are still being computed.

The authors introduced an approach to pipeline the PEs and exploit parallelism inside them, while satisfying the Viterbi algorithm data dependencies. The pipelined systolic array of PEs simultaneously computes the Viterbi matrices cells of multiple sequences being compared against the same profile HMM. This way, the anti-diagonal d_s enters the first stage of the PE at cycle c, and anti-diagonal $d_{s'}$ at cycle $c + 1$, where s and s' are different sequences being processed. As long as there are as many sequences as is the number of stages in the PE, this approach is successful due to the lack of dependences between different sequences. The input residue stream must be modified in a pre-processing step in order to contain multiple sequences interleaved with each other.

The proposed architecture was implemented in VHDL and synthesized for the Xilinx Virtex-II 6000 FPGA and supports up to 68 four-stage PEs, processing a HMM with at most 544 nodes and a maximum sequence length of 1024 residues. They achieved a clock frequency of 180MHz and the performance of 10.6 GCUPS.

4.4. Systolic Array with Divergences

Like the designs discussed in the previous sections, the FPGA-based approach presented by Eusse et al. (2009) calculates the best alignment score and does not provide the alignment itself of the query sequence against the profile HMM. Nevertheless, the accelerator also produces the alignment limits, which determine the area in the dynamic programming matrices where the best alignment occurs. If the similarity score is significant enough, then the sequence is reprocessed in software to generate the alignment. However, in the software execution of the Viterbi algorithm it is not necessary to compute the whole dynamic programming matrices again. The alignment limits produced by the accelerator are used, in order to calculate only the cells inside the area where the best alignment occurs, saving memory space and execution time.

The alignment limits are obtained by adapting the divergence concept presented in Batista et al. (2008) (Section 2.2.3) to the Viterbi algorithm. Figure 17 illustrates the divergence concept. Given a profile HMM with *k* nodes and a query sequence of length *n*, the figure shows the matrices of the Viterbi algorithm. The best alignment of the sequence to the HMM is a path along the cells of the matrices. The limits of the best alignment are expressed by its initial and final rows and superior and inferior divergences. The initial and final rows indicate the row of the matrices where the alignment starts and ends. The superior and inferior divergences represent how far the alignment departs from the main diagonal, in up and down directions, respectively. These divergences are calculated as the difference i–j between the row (i) and column (j) coordinates of the matrix cell. The initial and final rows and the superior and inferior divergences of the alignment determine the alignment region, shown in shadow in the figure. This region contains the cells of the score matrices M, I, and D that must be computed in order to obtain the best alignment. The other Viterbi algorithm vectors are also limited by the initial and final rows, as well.

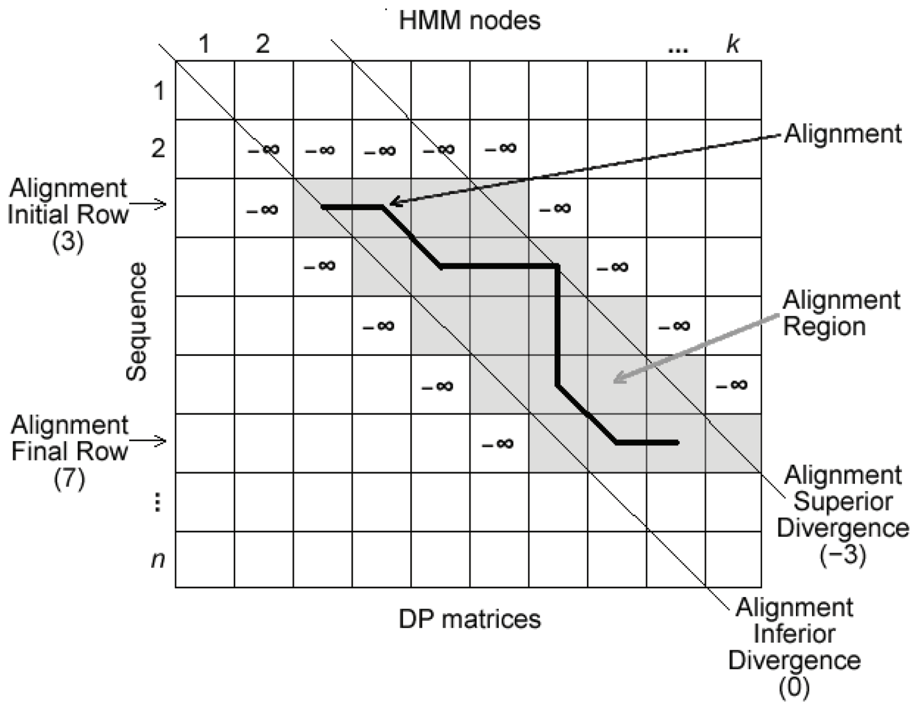
The proposed architecture consists of the systolic array of PEs, implementing a simplified version of the Viterbi algorithm without the state J. Besides computing the scores, each PE also calculates the alignment limits. These limits are computed as new dynamic programming matrices and vectors.

The accelerator was implemented with a parameterized VHDL code, in which the word bit-width, the number of PEs and the size of the memories can be modified at synthesis time. The system was designed in VHDL and prototyped on a Altera Stratix II EP2S180F1508C3 FPGA, obtaining a clock frequency of 67MHz for 85 PEs. The system can process HMMs with up to 2295 nodes and sequences with maximum length of 8192 amino acids, achieving a performance of 5.8 GCUPS.

4.5. Two-Pass Systolic Array

Maddimsetty et al. (2006) proposed a two-pass accelerator to reduce the error induced by the elimination of the J state. Their technique allows the detection of the sequences that would produce a multiple hit alignment when compared to a profile HMM, but, if limited to only one hit,

Figure 17. Alignment limits and alignment region for a HMM with k nodes and a sequence of length n (Eusse et al., 2009)


HMM nodes
1
2
...
k
1
2
Sequence
n
-∞
Alignment
Alignment Initial Row (3)
Alignment Final Row (7)
Alignment Region
Alignment Superior Divergence (−3)
Alignment Inferior Divergence (0)
DP matrices

would obtain a low score and be discarded for not satisfying the threshold.

The profile HMM model is modified in order to compute the total score of the best two matches of subsequences of the query sequence to the original profile HMM. If this score is higher than the score obtained with only one hit, then the sequence is accepted, based on the assumption that it will produce a significant multiple hit alignment.

The two-pass accelerator can be viewed as running the Viterbi algorithm on a doubled version of the original profile HMM, without the state J. The two-pass profile HMM in showed in Figure 18 and consists of two copies of the original HMM (with the J state removed). The two copies are connected by a linker state L that permits skipping residues from the query sequence between the two matches. The accelerator can be implemented using a design similar to the systolic array structure.

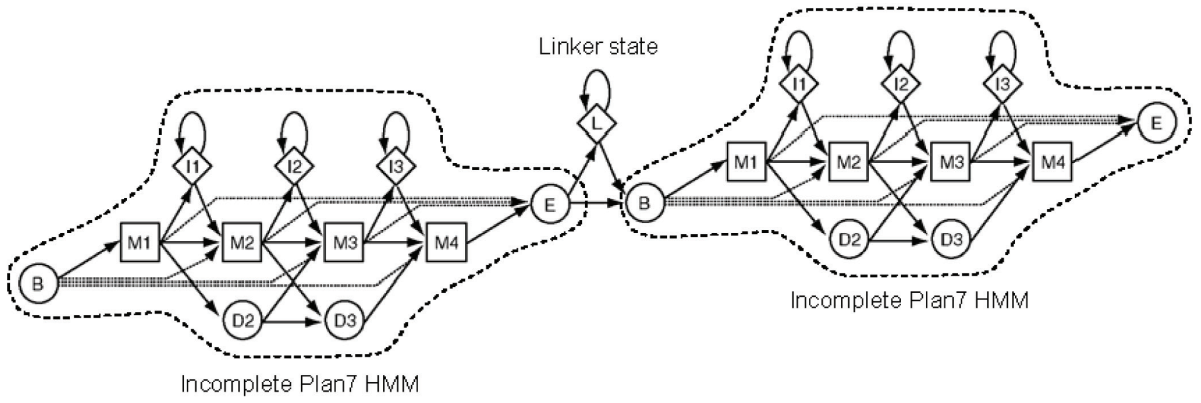
Based on technology assumptions about current FPGAs, the authors evaluated the design of a systolic array of 50 PEs running at a clock frequency of 200MHz, and obtained an estimated performance of 5 to 20 GCUPS.

4.6. Systolic Array with Recalculation

Sun et al. (2009) proposed a FPGA accelerator to the Viterbi algorithm, based on a systolic array structure with special units to handle data access and recalculation, in order to correct the error caused by the elimination of the state J. The authors argue that the feedback loop induced by the J state rarely occurs in the sequence-profile comparison.

Their architecture speculatively computes all the cells in the same anti-diagonal in parallel (similar to the previous works), assuming that the feedback loop induced by the state J does not take place. After computing the scores of a

Figure 18. Two-pass profile HMM to reduce the error caused by J state elimination



set of consecutive rows of the Viterbi matrices, the system checks if the feedback loop actually occurred. That is, the system detects if the best alignment so far should go through the state J and, consequently, the B state score should be calculated using the J state score.

If the feedback loop indeed occurred, a recalculation unit initiates a rollback mechanism to discard the scores of the cells computed with error, and updates the score of state B, this time considering the J state. It also restarts the computation in the systolic array, so that the PEs will recalculate the cells in the matrices rows with inaccurate scores. With this approach, the system ensures the correctness of the resulting similarity score.

The accelerator was coded in VHDL and synthesized to a Xilinx Virtex 5 110-T FPGA, with 25 PEs and clock frequency of 130MHz. The performance achieved was 3.2 GCUPS.

4.7. Complete Plan7 HMM

Oliver et al. (2008) proposed a FPGA-based accelerator that implements the Viterbi algorithm for the complete Plan7 HMM. Their accelerator acts as a filter that outputs the similarity score between a profile HMM and a sequence, as the previous solutions.

Because of the data dependences of the complete Plan7 Viterbi algorithm, low parallelism can be exploited inside the comparison of a sequence to a profile HMM. Therefore the authors propose to execute several comparisons in parallel, that is, to process different sequences against the same HMM at the same time. Since there are no data dependences between these comparisons, this approach exploits the coarse-grained parallelism inherent in the processing of different sequences.

Each PE computes all the cells of the matrices in the Viterbi algorithm, for a different sequence. All PEs are synchronized to process the same HMM state at the same clock cycle. This PE is more complex and resource demanding than those in the systolic array structure of previous works, since it computes the scores for all states in the profile HMM, including the state J.

The proposed accelerator was designed in Verilog and targeted to two FPGAs, Xilinx Spartan-3 XC3S1500 and XC3S4000, achieving a clock frequency of 70 MHz. In the first device, a system with 10 PEs was generated to process HMMs with up to 256 nodes, obtaining a performance of 700 MCUPS (MegaCUPS). In the second FPGA, 13 PEs were generated to process HMMs with up to 1024 nodes, achieving a performance of 910 MCUPS. In this last device, designs with 30 PEs were also generated to handle HMMs with

Table 3. Comparative overview of the sequence-profile comparison accelerators

Paper	Approach	Number of PEs	Max HMM nodes	Max sequence length	Clock rate (MHz)	Performance (GCUPS)	FPGA
Benkrid et al.(2008)	Systolic array without J state	90	1440	1024	100	5.2	Xilinx Virtex-II Pro 2VP100
Oliver et al. (2009)	Systolic array without J state	72	236	8192	74	3.95	Xilinx Virtex-II XC2V6000
Jacob et al. (2007)	Systolic array with interleaved sequences	68	544	1024	180	10.6	Xilinx Virtex-II 6000
Eusse et al. (2009)	Systolic array with alignment coordinates	85	2295	8192	67	5.8	Altera Stratix-II EP2S180F1508C3
Maddimsetty et al. (2006)	Two-pass systolic array	50	Not reported	Not reported	200 (estimated)	5 to 20 (estimated)	Not synthesized
Sun et al. (2009)	Systolic array with recalculation	25	Not reported	Not reported	130	3.2	Xilinx Virtex-5 110-T
Oliver et al. (2008)	Complete Plan7 HMM	10 to 30	256 to 1024	Not reported	70	0.7 to 2.1	Xilinx Spartan-3 XC3S1500/4000

at most 512 nodes, producing a performance of 2.1 GCUPS.

4.8. Comparative Overview

Table 3 summarizes the FPGA-based sequence-profile comparison proposals discussed in Sections 4.2 to 4.7 and presents a comparative overview of them.

The second column of Table 3 describes the main approach used in the corresponding accelerator. All accelerators act as filters and compute the alignment score, but do not produce the alignment itself. Only the solution proposed by Eusse et al. (2010) produces the alignment limits besides the score, reducing the computations that must be performed in software. All accelerators, except the last one, use the systolic array of PEs structure. Among the systolic array implementations, the solution of Jacobi et al. (2007) is the only one that implements a pipelined PE to process interleaved sequences. All systolic array accelerators eliminate the state J of the Plan7 HMM, however the systems proposed by Maddimsetty et al. (2006) and Sun at al. (2009) reduce or correct the error produced by this simplification. Only Oliver et al. (2008) implement the complete Plan7 HMM Viterbi algorithm (including the state J).

The number of PEs implemented in each accelerator is indicated in the third column of the table. This number is limited by the FPGA capacity and by the memory banks available in the device. Considering the solutions based on the systolic array structure, the number of PEs ranges from 25 to 90. As expected, the accelerator with the greatest number of PEs (Benkrid et al., 2008) implements the basic approach of computing the anti-diagonals in parallel, without additional mechanisms. This way, the design of the PE is simple and consumes a reduced area in the FPGA, permitting the instantiation of many PEs. Eusse et al. (2010) were able to implement a good number of PE (85), considering that their PEs also include the alignment limits computation. Among the solutions that use the systolic array, the accelerator of Sun et al. (2009) has the smallest number of PEs. This was probably a consequence of the implementation of the special units for data access and recalculation, consuming a fraction of the FPGA capacity. Oliver at al. (2008) implemented a few PEs in their accelerator, which executes the complete Plan7 Viterbi algorithm. Their PE is more resource demanding than those in the systolic array solutions, since it computes the scores for all states in the profile HMM.

The size of the sequence-profile comparison problem is expressed by the number of nodes of the profile HMM and the sequence length. The maximum size supported by the proposed FPGA systems is informed in the fourth and fifth columns of Table 3. These values impose restrictions on the profile HMM models and sequences that can be processed in each accelerator. The system proposed by Eusse et al. (2010) supports profile HMM with many more nodes than the other solutions, and also handles long sequences.

The maximum clock frequency obtained by the system and the peak performance achieved (expressed in GCUPS) are shown in columns 6 and 7. Considering the designs that were synthesized to FPGAs, the clock frequency achieved ranges from 67 to 180 MHz.

The HMMER software (Section 2.3.3) is reported to achieve 24 MCUPs, running on a Pentium 4 processor with 3GHz and 1GB of RAM memory (Oliver et al., 2009). It can be considered a baseline solution for the performance evaluation. Comparing the performance of the FPGA implementations against the baseline solution, we see that most accelerators achieve a performance improvement of two orders of magnitude.

The accelerator described in Jacob et al. (2007) has the highest clock frequency (180 MHz), since the PE computation is divided into four stages of a pipeline. Consequently, it achieves the best performance among all solutions. Comparing the results produced by Benkrid et al. (2008) and Jacob et al. (2007), we see that the second yields a 2-fold increase in the performance, even though the first one has more PEs. These results demonstrate that exploiting pipelining is crucial to improve the performance of the system.

The system proposed by Sun et al. (2009) has the worst performance, among the solutions based on the systolic array, due to the small number of PEs implemented. A more accurate performance measurement may produce even worse results, since the reported performance ignored the recalculation overhead.

Oliver at al. (2008) report the worst performance among all accelerators, since their system has few PEs working in parallel. From the performance results, it becomes clear that without the data dependence induced by state J of the Plan7 HMM, better performances can be achieved, since much more parallelism can be exploited in the execution of the Viterbi algorithm.

Finally, the last column shows the target FPGA devices used for the synthesis of the accelerators. All proposed accelerators were synthesized to FPGAs, with the exception of the work of Maddimsetty et al. (2006), which estimated the clock frequency and the performance of the system.

5. CONCLUSION AND PERSPECTIVES

Many biological data analysis algorithms exhibit a great amount of fine-grained parallelism. The arithmetic is mostly composed of integer operations on small data width operands, and floating point operations are generally not required. Since FPGAs are suitable for implementing these regular structures, FPGA-based accelerators have exhibited extremely good performance for time consuming Bioinformatics algorithms related to DNA or protein sequence analysis over a huge volume of data. These accelerators have demonstrated that FPGAs can be a very attractive alternative compared or in complement to supercomputers or clusters in both academic and industrial projects dedicated to this domain. The BioXL/H system from Biocceleration (2009) and the DeCypher system from TimeLogic (2009) are two examples of commercial FPGA-based implementations available to Bioinformatics applications.

In this chapter, we discussed in detail several state-of-the-art accelerators for two fundamental problems in Bioinformatics: pairwise sequence comparison and sequence-profile comparison. As discussed in this chapter, most of the accelerators in the literature are designed as systolic arrays and they are able to accommodate up to 248 processing elements. Impressive speedups of up to 500 were obtained, when compared to the software-only based approach. This shows that FPGA-based designs are an extremely good alternative to increase drastically the performance of Bioinformatics algorithms.

5.1 Future Trends and Open Problems

Even though the first FPGA-based accelerators for Bioinformatics applications appeared in 1992 and impressive speedups have been obtained, this is still a very active research area, with many challenging problems yet to be solved.

A very important open problem that must be addressed is the reduction of the time needed to produce correct and functional code for FPGAs. This time is still very high, if compared to software-only approaches. Regarding this issue, a clear tendency for Bioinformatics applications is the focus on customizable designs or even skeletons where the structure of the basic systolic array processor is provided and the programmer can concentrate efforts on the distinguishing characteristics of his/her design. Combined with high-level languages such as System-C and Handel-C, the time to produce the accelerators can be substantially reduced.

Another very important issue is the exploitation of the FPGA's reconfiguration capabilities. Although current FPGA boards do support dynamic reconfiguration, most accelerators for pairwise or sequence-profile comparison do not exploit this characteristic. Usually, the parameters in the customizable designs are set statically, at synthesis time. Solutions that fully exploit this feature in order to enhance the system flexibility or capacity are yet to be proposed.

Another challenge in the design of FPGA-based accelerators for biological sequence analysis is that, in order to achieve large speedups, a high I/O bandwidth between the host system and the FPGA is needed. If the bandwidth to feed the FPGA with sequences and models is insufficient, the accelerator will likely remain bandwidth-constrained and will not be able to process data at its maximum potential capacity. The tendency is to overcome this issue using solutions with a strong coupling between the FPGA and the host system and/or a highly efficient communication interface between them.

Finally, most of the FPGA-based accelerators for either pairwise or sequence-profile comparison output only the score. As discussed throughout this chapter, the retrieval of the alignment is usually left to the software. Most authors argue that the retrieval of alignments in hardware would require the storage of the dynamic programming matrices in the FPGA internal memory, limiting severely the size of the sequences compared. This line of arguing contains a clear fallacy since, in the literature, there are algorithms, such as the one proposed by Myers and Miller (1988), that retrieve alignments in linear space. The basic idea of linear-space or memory-limited algorithms is the combination of limited matrix storage and reprocessing in order to produce the alignments. These algorithms were implemented successfully in software. Therefore, in our opinion, a crucial open question that is worth deeper investigation is the possibility of retrieving the alignment for long biological sequences efficiently in FPGAs. This is surely not an easy task since it will probably require the adaptation of existent algorithms or even the proposal of entirely new ones.

REFERENCES

Batista, R. B., Boukerche, A., & Melo, A. C. M. A. (2008). A parallel strategy for biological sequence alignment in restricted memory space. *Journal of Parallel and Distributed Processing*, *68*(4), 548–561. doi:10.1016/j.jpdc.2007.08.007

Benkrid, K., Liu, Y. & Benkrid, A. (2009). A highly parameterized and efficient FPAG-based skeleton for pairwise biological sequence alignment. *IEEE Transactions on Very large Integration Systems, 17*(4), 561-570.

Benkrid, K., Velentzas, P., & Kasap, S. (2008). A high performance reconfigurable core for motif searching using profile HMM. In *NASA/ESA Conference on Adaptive Hardware and Systems*, (pp. 285-292).

Biocceleration. (2009). *BioXL/H*. Retrieved October 10, 2009, from http://www.biocceleration.com

Boukerche, A., Correa, J. M., Melo, A. C. M. A., Jacobi, R. P., & Rocha, A. F. (2007a). Reconfigurable architecture for biological sequence comparison in reduced memory space. In *IEEE International Parallel & Distributed Processing Symposium, Workshop NIDISC*.

Boukerche, A., Correa, J. M., Melo, A. C. M. A., Jacobi, R. P., & Rocha, A. F. (2007b). An FPGA-based accelerator for multiple biological sequence alignment with DIALIGN. In *International Conference on High Performance Computing*, (LNCS 4873, pp. 71-82).

Derrien, S., & Quinton, P. (2007). Parallelizing HMMER for hardware acceleration on FPGAs. In *International Conference on Application-specific Systems, Architectures and Processors*, (pp. 10-17).

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (1998). *Biological Sequence Analysis*. Cambridge, UK: Cambridge University Press. doi:10.1017/CBO9780511790492

Eddy, S. (2003). *HMMER User's Guide*. Retrieved October 15, 2009, from http://hmmer.janelia.org

Eusse, J. F., Moreano, N., Melo, A. C. M. A., & Jacobi, R. P. (2010). A HMMER hardware accelerator using divergences. In *Design, Automation & Test in Europe Conference*.

Fickett, J. W. (1984). Fast optimal alignments. *Nucleic Acids Research*, *12*(1), 175–179. doi:10.1093/nar/12.1Part1.175

Finn, R. D. (2008). The Pfam protein families database. *Nucleic Acids Research*, *36*, 281–288. doi:10.1093/nar/gkm960

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, *162*, 705–708. doi:10.1016/0022-2836(82)90398-9

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge, UK: Cambridge University Press. doi:10.1017/CBO9780511574931

Hoang, D. T., & Lopresti, D. P. (1992). FPGA implementation of systolic sequence alignment. In *Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping*, (LNCS 705, pp. 183-191). Berlin: Springer-Verlag.

Hunter, L. (1993). *Artificial Intelligence and Molecular Biology*. Cambridge, MA: MIT Press.

Jacob, A. C., Lancaster, J. M., Buhler, J. D., & Chamberlain, R. D. (2007). Preliminary results in accelerating profile HMM search on FPGAs. In *IEEE International Symposium on Parallel and Distributed Processing*, (pp. 1-8).

Jiang, X., Liu, X., Xu, L., Zhang, P., & Sun, N. (2007). A reconfigurable accelerator for Smith-Waterman algorithm. *IEEE Transactions on Circuits and Systems II*, *54*(12), 1077–1081. doi:10.1109/TCSII.2007.909857

Krogh, A. (1994). Hidden Markov models in computational biology: applications to protein modeling. *Journal of Molecular Biology*, *235*(5), 1501–1531. doi:10.1006/jmbi.1994.1104

Kung, H. T. (1982). Why systolic architectures? *IEEE Computer*, *15*(1), 37–46.

Lipton, R. J., & Lopresti, D. (1985). A systolic array for rapid string comparison. In *Chapel Hill Conference on VLSI*, (pp. 363-376).

Maddimsetty, R. P., Buhler, J., Chamberlain, R. D., Franklin, M. A., & Harris, B. (2006). Accelerator design for protein sequence HMM search. In *International Conference on Supercomputing*, (pp. 288-296).

Morgenstern, B., Frech, K., Dress, A., & Werner, T. (1998). DIALIGN: Finding local similarities by multiple sequence alignment. *Bioinformatics (Oxford, England)*, *14*, 290–294. doi:10.1093/bioinformatics/14.3.290

Mount, D. (2004). *Bioinformatics: Sequence and Genome Analysis*. New York: C. S. Harbor Lab Press.

Myers, E. W., & Miller, W. (1988). Optimal alignments in linear space. *Computer Applications in the Biosciences*, *4*(1), 11–17.

Needleman, S., & Wunsh, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*, 443–453. doi:10.1016/0022-2836(70)90057-4

Oliver, T., Schmidt, B., Jakop, Y., & Maskell, D. (2009). High speed biological sequence analysis with hidden Markov models on reconfigurable platforms. *IEEE Transactions on Information Technology in Biomedicine*, *13*(5), 740–746. doi:10.1109/TITB.2007.904632

Oliver, T., Schmidt, B., & Maskell, D. (2005). Hyper customized processor for bio-sequence database scanning on FPGAs. In *ACM/SIGDA International Symposium on Field Programming Gate Arrays*, (pp. 229-237).

Oliver, T., Yeow, L. Y., & Schmidt, B. (2008). Integrating FPGA acceleration into HMMer. *Parallel Computing*, *34*, 681–691. doi:10.1016/j.parco.2008.08.003

Puttegowda, K., Worek, W., Pappas, N., Dandapani, A., & Athanas, P. (2003). A run-time reconfigurable system for gene-sequence searching. In *International Conference on VLSI Design*, (pp. 561-566).

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286. doi:10.1109/5.18626

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, *147*(1), 195–197. doi:10.1016/0022-2836(81)90087-5

Sun, Y., Li, P., Gu, G., Wen, Y., Liu, Y., & Liu, D. (2009). Accelerating HMMer on FPGAs using systolic array based architecture. In *IEEE International Symposium on Parallel and Distributed Processing*, (pp. 1-8).

TimeLogic. (2009). *DeCypher FPGA Biocomputing Systems*. Retrieved October 10, 2009, from http://www.timelogic.com

Yamaguchi, Y., Maruyama, T., & Konagaya, A. (2002). High speed homology search with FPGAs. In *Pacific Symposium on Biocomputing*, (pp. 271-282).

Zhang, P., Tan, G., & Gao, G. R. (2007). Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform. In *Conference on High Performance Networking and Computing*, (pp. 39-48).