A Hardware Accelerator for the Fast Retrieval of DIALIGN Biological Sequence Alignments in Linear Space

Azzedine Boukerche, *Senior Member*, *IEEE*, Jan M. Correa,
Alba Cristina M.A. de Melo, *Senior Member*, *IEEE*, and Ricardo P. Jacobi

Abstract—The recent and astonishing accomplishments in the field of Genomics would not have been possible without the techniques, algorithms, and tools developed in Bioinformatics. Biological sequence comparison is an important operation in Bioinformatics because it is used to determine how similar two sequences are. As a result of this operation, one or more alignments are produced. DIALIGN is an exact algorithm that uses dynamic programming to obtain optimal biological sequence alignments in quadratic space and time. One effective way to accelerate DIALIGN is to design FPGA-based architectures to execute it. Nevertheless, the complete retrieval of an alignment in hardware requires modifications on the original algorithm because it executes in quadratic space. In this paper, we propose and evaluate two FPGA-based accelerators executing DIALIGN in linear space: one to obtain the optimal DIALIGN score (DIALIGN-Score) and one to retrieve the DIALIGN alignment (DIALIGN-Alignment). Because it appears to be no documented variant of the DIALIGN algorithm that produces alignments in linear space, we here propose a linear space variant of the DIALIGN algorithm and have designed the DIALIGN-Alignment accelerator to implement it. The experimental results show that impressive speedups can be obtained with both accelerators when comparing long biological sequences: the DIALIGN-Score accelerator achieved a speedup of 383.4 and the DIALIGN-Alignment accelerator reached a speedup of 141.38.

Index Terms—Biology and genetics, dynamic programming, special-purpose and application-based systems.

1 Introduction

The rapid evolution of sequencing techniques combined with the intense growth in the number of large-scale genome projects is producing a huge amount of biological sequence data. Nevertheless, determining the genome sequence is only the first step toward deciphering the genetic message encoded in those sequences. In genome projects, newly determined sequences are first compared with those placed in genomic databases, in order to discover similarities [1]. This is done because relevant sequence similarity is evidence of common evolutionary origin and homology relationship.

Pairwise sequence comparison is, therefore, a very basic but important step in genome projects. As a result of this step, one or more sequence alignments can be produced. A sequence alignment has a similarity score associated to it that is obtained by placing one sequence above the other, making clear the correspondence between the characters [2].

- *A. Boukerche is with the School of Information Technology and Engineering (SITE), University of Ottawa, 800 King Edward Avenue, Ottawa, Ontario KIN 6N5, Canada. E-mail: boukerch@site.uottawa.ca.*
- *J.M. Correa and R.P. Jacobi are with the Department of Computer Science, University of Brasilia (UnB), Campus UNB—ICC-Norte—sub-solo 70910-900, Brasilia-DF, Brazil. E-mail: {jan, rjacobi}@cic.unb.br.*
- *A.C.M.A. de Melo is with the Department of Computer Science, University of Brasilia (UnB), Campus UNB—ICC-Norte—sub-solo 70910-900, Brasilia-DF, Brazil, and with the PARADISE Research Laboratory, University of Ottawa, Canada. E-mail: albamm@cic.unb.br.*

Manuscript received 29 July 2008; revised 6 Feb. 2009; accepted 16 July 2009; published online 11 Feb. 2010.
Recommended for acceptance by A. George.
For information on obtaining reprints of this article, please send E-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-07-0378.
Digital Object Identifier no. 10.1109/TC.2010.42.

Smith-Waterman (SW) [3] is an exact algorithm based on the longest common subsequence (LCS) concept that uses dynamic programming to find optimal local alignments between two sequences of size n in quadratic space and time. In this algorithm, a similarity matrix of size $n \times n$ is calculated. Nowadays, SW is the most widely used exact method to locally align two sequences, and it is very accurate if the sequences have a single common region of high similarity. However, if the sequences share more than one region of high similarity, SW is not very effective.

DIALIGN [4] is based on the idea that a biological sequence alignment must be built from significant gapless fragments and is thus able to cope with the situation of sequences sharing many high similarity regions. DIALIGN can be used for either local or global alignment as well as pairwise or multiple sequence alignment. In [5], a variant of DIALIGN was successfully used to obtain multiple sequence alignments of noncoding DNAs. One drawback of DIALIGN is that it is slower than SW. To overcome this, alternatives have been proposed to run DIALIGN in parallel [6] and to combine it with a fast local search similarity tool [7].

Several high performance hardware-based architectures have been proposed in the literature [8]. In this paper, we propose two FPGA-based architectures that execute DIALIGN in linear space. The goal of the first architecture, called DIALIGN-Score, is to obtain the DIALIGN similarity score. A partition technique is used in this design, enabling sequences of any size to be compared.

In many cases, the biologists also need to observe the alignment between the sequences. It is for this reason that DIALIGN-Alignment, a second architecture which is able to retrieve the optimal alignment entirely in hardware, is

proposed. This is particularly intricate because the quadratic DIALIGN similarity matrix cannot be completely stored in the FPGA internal memory, unless the sequences being compared are very small. Even if complete storage was possible, the large volume of internal memory accesses would quickly become a severe bottleneck.

To fully retrieve DIALIGN alignments on hardware in linear space, we propose a new variant of the DIALIGN algorithm and designed the DIALIGN-Alignment accelerator to execute it. In both accelerators, a wavefront array processor design is used. To our utmost knowledge, this is the first time DIALIGN has been implemented in hardware.

The architectures proposed in this paper were designed using SystemC [9] and compiled to Verilog with FORTE [10]. The Quartus tool [11] was used to synthesize them to the FPGA Altera STRATIX II EP2S180F1508I4 and for simulation. The results obtained with real and synthetic nucleotide sequences show that impressive speedups were obtained, when compared to a C program that executes the same algorithm, running on a Pentium 4 3 GHz. For instance, a speedup of 383.41 was achieved by DIALIGN-Score when comparing real DNA sequences of size 194,439 bp (*base pairs*) and 169,786 bp. In the previously cited case, the software implementation took 3 h 4 min and our FPGA implementation took 28.8 s. In order to retrieve the alignment produced through the comparison of one real ncRNA (*noncoding* RNA) sequence of 118 bases with a DNA sequence of size 5,057,142 bp, a speedup of 126.61 was achieved by DIALIGN-Alignment. This reduced the execution time from 1 min 42 s (software) to 0.8 s (FPGA). In order to compare synthetic DNA sequences of 128 bp and 10 Mbp (*Mega base pairs*), the execution time was reduced from 3 min 45 s (software) to 1.59 s (FPGA), yielding a speedup of 141.38.

The remainder of this paper is organized as follows: Section 2 introduces the biological sequence comparison problem and presents two widely used exact algorithms that solve it: Smith-Waterman and DIALIGN. In Section 3, related work in the area of FPGA-based architectures for sequence comparison is discussed. Section 4 presents the design of two hardware accelerators made to execute DIALIGN: DIALIGN-Score and DIALIGN-Alignment. Experimental results are discussed in Section 5 whereas Section 6 concludes the paper and presents future work.

2 Biological Sequence Comparison

2.1 Sequence Alignment and Score

To compare two 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 characters [2].

Given a pairwise alignment between sequences s and t, the following score can be assigned. For each two bases in the same column, we associate, for instance, $+1$ if both characters are identical (*match*), -1 if the characters are different (*mismatch*) and -2 if one character is a space (*gap*). The score is the sum of the values computed for each column. Fig. 1 illustrates an alignment.

The maximum score is called the similarity between the sequences. To solve a biological sequence alignment problem, we need to obtain the alignment with the highest score.

T	T	G	T	C	-	A	G	A	
T	T	G	T	C	G	A	G	G	**score**
+1	+1	+1	+1	+1	-2	+1	+1	-1	**4**

Fig. 1. Example of alignment between sequences $\mathrm{s} = \mathrm{TTGTCAGA}$ and $\mathrm{t} = \mathrm{TTGTCGAGG}$, with alignment $\mathrm{score} = 4$.

One of the first exact methods to globally align two sequences was proposed by Needleman-Wunsh (NW) [12]. This method 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 ($|s| = m$ and $|t| = n$).

2.2 The SW Algorithm and Its Variants

The NW algorithm was modified by Smith-Waterman [3] to deal with local alignments. Like NW, SW is also based in dynamic programming with quadratic time and space complexity.

The algorithm consists of two parts: the computation of a similarity matrix where the highest score indicates the similarity between the sequences and the identification of the optimal alignment(s) with the "traces" left by the highest scores along the matrix.

As input, the algorithm receives two sequences s and t, with sizes m and n, respectively ($|s| = m$ and $|t| = n$). There are $m + 1$ possible prefixes for s and $n + 1$ possible prefixes for t, including the empty string. An array is built where the (i, j) entry contains the value of the similarity between two prefixes of s and t, $sim(s[1..i], t[1..j])$.). If we denote the array by A, the value of $A[i, j]$ is the similarity between the prefixes $s[1..i]$ and $t[1..j]$. The recurrence relation proposed by SW is expressed by (1). In this equation, g is the gap penalty and $p(i, j)$ is the punctuation for a match ($s[i] = t[j]$) or a mismatch ($s[i] \neq t[j]$).

$$A[i,j] = \max \begin{Bmatrix} A[i-1, j-1] + p(i,j), \\ A[i, j-1] + g, \\ A[i-1, j] + g, \\ 0 \end{Bmatrix} \quad (1)$$

Fig. 2 shows the similarity array A between $s = AACGTTGAGCAG$ and $t = \mathrm{ACGCATTGAGTCAG}$. The first row and column are initialized with zeros. The other entries are computed using (1). In Fig. 2, $g = -2$ and $p(i, j) = +1$ if $s[i] = t[j]$ and -1 if otherwise.

Array A is computed row by row and left to right on each row; column by column and top to bottom on each column; or antidiagonal by antidiagonal, from top to bottom. For each cell calculated, an arrow is drawn to indicate the cell that is used in this calculation ($A[i-1, j]$, $A[i, j-1]$ or $A[i-1, j-1]$), according to (1).

Having calculated the whole similarity array, an optimal local alignment between two sequences is obtained as follows: The maximum score value in array A is found and the arrow departing from this entry is followed until an entry with value *0* is reached. Each arrow used determines one column of the alignment. A left arrow in $A_{i,j}$ indicates the alignment of $s[i]$ with a gap in t. An upward 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 $t[j]$. In SW, an optimal local alignment is therefore constructed from its end to its beginning.

		A	A	C	G	T	T	G	A	G	C	A	G
	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	↖1	↖1	0	0	0	0	0	↖1	0	0	↖1	0
C	0	0	0	↖2	0	0	0	0	0	0	↖1	0	0
G	0	0	0	0	↖3	←1	0	↖1	0	↖1	0	0	↖1
C	0	0	0	0	↖1	↑1	↖2	0	0	0	0	↖2	0	0
A	0	↖1	↖1	0	0	0	0	0	↖1	0	0	↖3	←1
T	0	0	0	0	0	↖1	↖1	0	0	0	0	↑1	↖2
T	0	0	0	0	0	↖1	↖2	0	0	0	0	0	0
G	0	0	0	0	↖1	0	0	↖3	←1	↖1	0	0	↖1
A	0	↖1	↖1	0	0	0	0	↑1	↖4	←2	0	↖1	0
G	0	0	0	0	↖1	0	0	↖1	↑2	↖5	←3	←1	↖2
T	0	0	0	0	0	↖2	↖1	0	0	↑3	↖4	↖2	0
C	0	0	0	0	↖1	0	0	0	0	0	↑1	↖4	↖3	↖1
A	0	↖1	↖1	0	0	0	0	0	↖1	0	↑2	↖5	↖4
G	0	0	0	0	↖1	0	0	↖1	0	↖2	0	↑3	↖6

Local alignment: T T G A - G C A G
T T G A G T C A G

Fig. 2. Array to compute the similarity between sequences $\mathrm{s} = \mathrm{AACGTTGAGCAG}$ and $\mathrm{t} = \mathrm{ACGCATTGAGTCAG}$. The optimal score is 6 and the traceback path is shown in gray.

The algorithms NW and SW assign a constant value to gaps (*linear gap function*). However, in a biological perspective, results are more significant when gaps are kept together [13]. For this reason, the opening of a gap must have a greater penalty than its extension. Based on this observation, Gotoh [14] proposed an algorithm where the gap penalty is calculated with (2), 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*).

$$w(k) = uk + v, k \geq 1. \tag{2}$$

In order to calculate gaps according to (2), two matrixes are needed (P and Q), in addition to the similarity matrix A. These additional matrices are used to compute the cost of a set of gaps in sequences s and t, respectively. Even with the computation of additional matrices P and Q time complexity remains $\mathrm{O}(mn)$ [14].

Myers and Miller [15] proposed the use of Hirschberg's algorithm [16] to compute global alignments in linear space. This algorithm uses a divide-and-conquer technique that locates a point where the optimal alignment occurs and recursively splits the similarity matrix calculation to obtain the actual alignment in linear space. This approach doubles the execution time, in the worst case [16], when compared with NW. The same idea can also be applied to the local alignment problem [13].

2.3 The DIALIGN Algorithm

DIALIGN (*DIAgonal ALIGNment*) [4] is a method for sequence alignment that searches for fragments (or diagonals) that have no gaps and aligns them. In DIALIGN, an alignment is defined as a chain of fragments.

One example of a DIALIGN alignment is given in Fig. 3. In Fig. 3a, the subsequences belonging to fragments are shown in gray and the aligned fragments are depicted as

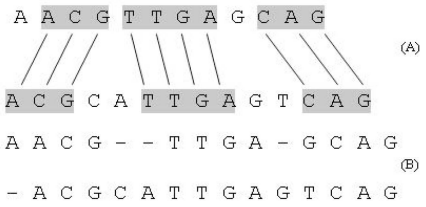

Fig. 3. Example of DIALIGN alignment between sequences $s = \mathrm{AACGTTGAGCAG}$ and $t = \mathrm{ACGCATTGAGTCAG}$, with three fragments $(\mathrm{ACG}, \mathrm{TTGA}, \text{ and } \mathrm{CAG})$.

lines. Fig. 3b shows the final alignment. Although sequences s and t are the same as those considered in Fig. 2, the alignments produced by SW and DIALIGN are different.

For each DIALIGN pairwise alignment, it is necessary to calculate the relevance of each diagonal found before attempting to align it. This is done through the equation $E(l, sm) = -\ln(P(l, sm))$, where $P(l, sm)$ is the probability of a diagonal D of size l have at least sm matches [4].

For each candidate fragment D_i, a weight $w(D_i)$ is assigned as $E(l, sm)$ if $E(l, sm)$ is above the given threshold T and 0, otherwise.

When DIALIGN obtains a new significant fragment, the algorithm tries to align it consistently with other previously calculated significant fragments [4]. In an alignment of k fragments $D_1, D_2, \ldots, D_k$ the total score S is obtained by the addition of all weights $w(D_i), i = 1$ to k.

To discover the score S, a dynamic programming strategy is used. Consider two sequences s and t with 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 fragment $(s_{i-k}t_{j-k}, \ldots, s_it_j)$ 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 of the prefixes $s[1..i]$ and $t[1..j]$.

The last fragment D_k which is aligned in position (i, j) is recovered by the function $prec(i, j) = D_k$. For each fragment D_k aligned in position (i, j), $prec(i, j)$ chooses the chain of fragments with the greatest score to date. The score is calculated as shown in (3), where $\sigma(D_{i,j})$ is defined as the largest score of a chain of fragments that ends in (i, j).

$$\mathrm{score(i,j)} = \max \begin{cases} \mathrm{score(i-1,j)}, \\ \mathrm{score(i,j-1)}, \\ \sigma\mathrm{(D(i,j))}, \end{cases} \tag{3}$$

$$prec(i,j) = \begin{cases} \mathrm{prec}(i, j-1), \mathrm{Iff\ score}(i,j) = \mathrm{score}(i, j-1) \\ \mathrm{prec}(i-1, j), \mathrm{Iff\ score}(i, j-1) \\ \qquad < \mathrm{score}(i,j) = \mathrm{score}(i-1, j) \\ D_{i,j}, \mathrm{Iff\ score}(i, j-1), \mathrm{score}(i-1, j) \\ \qquad < \mathrm{score}(i,j) = \sigma(D_{i,j}) \end{cases} \tag{4}$$

As in SW, DIALIGN executes in two steps. In the first one, two dynamic programming matrices are calculated: one for scores (3) and other for the preceding fragment (*prec* in (4)) [4]. Once these matrices are entirely calculated, the highest score is found and the reverse path on the *prec* matrix is used to retrieve the alignment.

3 Related Work

In the literature of this field, there are many proposals to execute dynamic programming biological sequence comparison algorithms in hardware. Most of them have the following characteristics in common:

1. *A unidimensional systolic array processor is used:* A systolic array processor is defined as an array of processor elements that execute in a lock-step basis. As the blood is pumped to the heart on a regular basis, data passes through the systolic processing elements at clock rate, flowing between neighbors [17]. The seminal work of Lipton and Lopresti [18] proposed a successful implementation of a dynamic programming sequence comparison algorithm in a systolic array. Currently, Lipton and Lopresti's architecture design is frequently used to deal with this problem.
2. *The antidiagonals are calculated in parallel*: The algorithms presented in Sections 2.2 and 2.3 impose a dependency where each value of the matrices is dependent upon the values of the upper, left, and upper-left cells. Such dependency leads to a nonuniform amount of parallelism, where each antidiagonal can be calculated independently. Therefore, the natural choice is to compute the elements of the same antidiagonal in parallel.
3. *The highest score is retrieved*: The retrieval of the alignment requires a traceback procedure over one or more $m \times n$ matrices (Sections 2.2 and 2.3). The storage of these matrices in hardware severely restricts the sizes of the sequences compared and, also, would possibly lead to great overheads due to memory access conflicts and problems with synchronization, mainly at the matrix calculation phase. For this reason, most of the approaches retrieve only the highest score, providing a measure of the similarity between the sequences;
4. *A partition technique is used*: The characters that compose the smallest sequence (also called *query sequence*) are frequently stored directly at the processing elements. In the basic design, each processing element (PE) stores one character and is, thus, able to compute one column of the matrix. This restricts the size of the smallest sequence. In order to compare sequences of any size, a partition technique is used that either stores more than one character in each PE or that computes the matrix in many phases, using one part of the query sequence per phase.

Fig. 4 shows how each antidiagonal of the dynamic programming matrix can be calculated in parallel by a five-element systolic array. The *query sequence* (ACGAT) is previously stored in the processing elements and the *database sequence* (CTTAG) flows through the systolic array. Each element calculates one cell in the current antidiagonal (shown in gray in Fig. 4) at the same time.

In the following paragraphs, we discuss some hardware-based architectures for biological sequence comparison that are present in the literature.

SW (Section 2.2) was implemented in a 128-element systolic array board called SAMBA (*Systolic Accelerator for Molecular Biological Applications*) by Lavenier [19]. In this

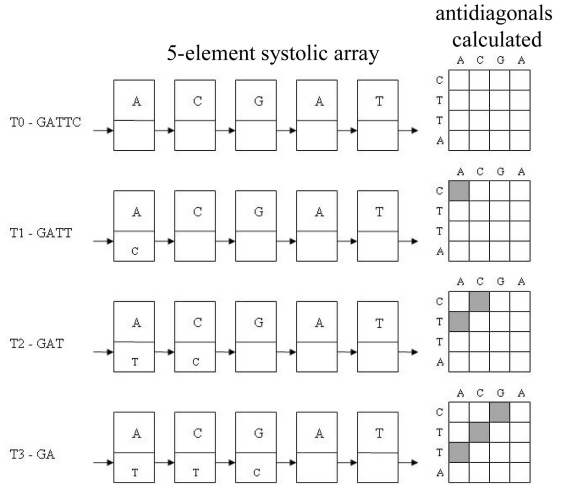

Fig. 4. Generic systolic array to calculate the dynamic programming matrix.

work, the antidiagonals are calculated in parallel, as shown in Fig. 4, and the highest score is obtained. A sequence partitioning technique is used that enables comparison between sequences of any size, by comparing each part of the query sequence with the entire database sequence. A speedup of 83 was obtained over the software when comparing a sequence of 3 kbp (*kilo base pairs*) with a genomic database of 2 Mbp (*Mega base pairs*).

A systolic array of computing elements was proposed by Yamaguchi and Maruyama [20] to retrieve SW alignments. If the size of the smallest sequence is greater than the number of elements available, the sequence is broken in many parts, in a multithreaded way. The overall procedure is divided in two phases. In the first phase, the query sequence is broken so that it can fit into the FPGA internal memory. As long as the scores are calculated, they are stored in the FPGA internal memory and sent to the host computer. In the second phase, the similarity matrix calculated in phase 1 is used to compute the best alignment by the FPGA. In order to align a 2 kbp sequence with a 8 kbp database, a speedup of 102 was obtained, when compared with the software implementation.

Puttegowda et al. [21] proposed a systolic array architecture to run the SW algorithm at the OSIRIS board. This board contains 2 FPGAs: one interfaces with the host machine whereas the other to executes user-programmed hardware designs. Dynamic 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 array.

A tool called PHG (*Parallel Hardware Generator*) was used to automatically generate a VHDL code and synthesize hardware from the SW recurrence relations by Marongiu et al. [22]. The basic comparison function was hand-designed. The goal was to search a small DNA sequence in a huge DNA database in order to obtain the highest score. A speedup of 55.6 was obtained over the software implementation, when comparing a 24 bp DNA sequence with a 520 kbp subset of a genomic database.

An FPGA-based systolic array processor was proposed by Oliver et al. [23] to execute either SW or Gotoh (Section 2.2).

In both instances, the goal of the architecture was to obtain the highest score. A partition technique is used where each processing element could store up to four sequence characters. The FPGA calculated the highest score and sent it to the host computer. A zigzag system floor design was used. Compared to the results obtained by an optimized C program, the FPGA attained a speedup of 170, when using SW to compare proteins of 1,428 amino acids with the SwissProt database (around 54 M amino acids).

Kestrel [24] is a board that includes a 512-element linear array of SIMD processing elements, which was initially designed to execute SW (Section 2.2) and Hidden Markov Model algorithms [2]. Kestrel is now a more generic architecture, targeted not only for bioinformatics applications but also for image processing. In Kestrel, each processing element completes its operation in one clock cycle. The goal of the SW algorithm implemented in Kestrel is to obtain the highest score. It computes each antidiagonal in parallel and each processing element is responsible to calculate one row. The best speedups were obtained when comparing a 500 amino acid sequence with a 10 M amino acid subset of a protein database. In this case, a speedup of 99 was achieved.

Zhang et al. [25] propose the use of a systolic array to execute SW with both linear and affine gap function (Section 2.2). In order to achieve better performance, the architecture executes in many stages, with uneven stage latencies. The proposed architecture calculates the highest score in hardware. A partition technique is also used.

Jiang et al. [26] propose a systolic array architecture to calculate the highest score according to the Gotoh algorithm (Section 2.2). An optimized zigzag floor plan is used to accommodate the PEs. Also, a partitioning technique is used where the query sequence is subdivided in many parts. A speedup of 352 was attained for the comparison of two 80 kbp sequences, when compared with software.

The Cray/DRC platform was used in [27] to implement the FASTA heuristic algorithm [28] in a hardware/software solution. In this platform, an Opteron is connected through HyperTransport to a Virtex FPGA, which acts as a coprocessor. The most compute-intensive phase of FASTA executes the SW algorithm in selected parts of the sequences, and this was the phase implemented in FPGA. The other phases of the algorithm were executed by the Opteron. SW was implemented as a linear array of systolic elements that retrieves the highest score. A partition technique was used, enabling sequences of any size to be compared. Both the internal FPGA block RAM and the QDR II RAM were used. In order to compare sequences of sizes $16\ \mathrm{kbp} \times 512\ \mathrm{kbp}$, a speedup of 100 was obtained.

A hardware-software solution that explores both fine-grained and coarse-grained parallelism was proposed by [29]. The SRC-6 and the Cray XD1 platforms were used to implement a systolic array that retrieves the score of the SW algorithm. As in [27], the FPGA acted as a coprocessor and is connected to the main processor through a high-speed network. Many nodes were interconnected through Gigabit Ethernet and they communicate with MPI (Message Passing Interface). A partition technique was also used. Speedups reaching up to 98 were achieved to compare a query sequence of 2 KB with a 64 KB database, when considering one engine/chip. The serial execution was also compared with a six-node parallel execution, where each node contains eight cores. In this case, a speedup of 2,794 was achieved.

TABLE 1
Comparative Overview of the Hardware Based Accelerators for Sequence Comparison Algorithms

Paper	Architecture	Algorithm	Goal	Partitioning	Speed up
[19]	FPGA systolic	SW	Score	Yes	83.0
[20]	FPGA systolic	SW	Alignment	Yes	102.0
[21]	FPGA systolic	SW	Computation	No	500.0
[22]	FPGA systolic	SW	Score	No	55.6
[23]	FPGA systolic	SW/Gotoh	Score	Yes	170.0
[24]	SIMD array	SW	Score	Yes	99.0
[25]	FPGA systolic	SW/Gotoh	Score	Yes	249.5
[26]	FPGA systolic	Gotoh	Score	Yes	352.0
[27]	FPGA systolic	SW	Score	Yes	100.0
[29]	FPGA systolic	SW	Score	Yes	98.0
[30]	FPGA systolic	Gotoh	Score	No	16.9

A combined hardware/software architecture for BlastP was proposed in [30], where a banded version of the Gotoh algorithm was implemented as an FPGA accelerator. The third phase of BlastP, *gapped extension*, consists of executing the Gotoh algorithm over parts of the sequences. In this design, only a subset of antidiagonals, called a *band*, is calculated. A systolic array design is used where the communication between neighbor elements is done through registers. There is, as a result, no direct link between the processing elements. Since only small parts of the sequences are compared, there is no need for partitioning. Instead, the query sequences are packed into 2,048 residues. Speedups of up to 16.99 were achieved in the Mercury platform, composed by two Opteron CPUs and two FPGA coprocessors. Although the sizes of the query and database sequences were impressive ($1.35\ \mathrm{MB} \times 282\ \mathrm{MB}$, respectively), the query sequence was filtered by BlastP (phases 1 and 2) and only a small part of it was actually used in the FPGA accelerator.

Table 1 presents a comparative analysis of the hardware-based architectures discussed in the previous paragraphs.

In this table, the paper's references are presented in the first column. The target architecture is shown in column 2. As can be seen, with the exception of [24], all proposals discussed use one or more FPGA-based systolic arrays.

The third column contains the algorithm implemented. In this case, SW either stands for Needleman-Wunsh (Section 2.1), Smith-Waterman (Section 2.2), or edit distance computation, all of which are similar algorithms. Most of the approaches analyzed [19], [20], [21], [22], [24], [27], [29] execute SW. Some of them [23], [25] execute either SW or Gotoh (Section 2.2) whereas [26] and [30] implement only Gotoh. At this time, however, there appears to be no hardware implementation for DIALIGN.

The goal of the architecture is presented in column 4. Most of the proposals [19], [22], [23], [24], [25], [26], [27], [29], [30] calculate the highest score because it can be done in linear space and it is very efficient in systolic arrays. Yamaguchi et al. [20] calculate the score and retrieve the alignment in hardware. Since the space complexity of SW is $\mathrm{O}(\mathrm{n}^2)$, only small sequences can be aligned. In Puttegowda et al. [21], only the similarity matrix computation was made,

without storing the matrix elements neither obtaining the highest score.

Most of the proposals [19], [20], [23], [24], [25], [26], [27], [29] use some kind of partitioning technique (column 5), in order to deal with sequences of any size. Only three [21], [22], [30] do not provide partitioning. In [30], partitioning is not required because the Gotoh algorithm is executed in the third phase of BlastP.

Finally, column 6 presents the best speedups obtained with the proposals. They range from 16.9 [30] to 500 [21] and most of the proposals [20], [23], [24], [25], [26], [27], [29] achieved speedups higher or equal than 90, when compared with the software implementation. This demonstrates the great potential for improvement in performance that can be achieved though hardware-based approaches.

In addition to those SW/Gotoh variations, a hardware/software approach is proposed in [31], that implements the BLAST heuristic method on the SGI Altix RC100 system. In this architecture, each node has two Virtex-4 FPGAs connected to the CPU by the NUMALink high bandwidth bus. The first phases of BLAST were ported to the FPGA and a systolic array was used to accelerate stage 2. Speedups of 19.52 were obtained, when compared to the CPU-only solution.

4 Design of the FPGA-Based Architectures to Execute DIALIGN

As discussed in Section 3, the great majority of the biological sequence comparison architectures in the literature use a systolic design. For SW and Gotoh (Section 2.2), this type of architecture is appropriate because the operations executed by each processing element are quite similar and take almost the same time to complete.

In the case of DIALIGN (Section 2.3), the recurrence relations are more complex and involve a set of conditional statements. For this reason, the time needed for each PE to complete its operations can greatly vary. If a systolic design is used in this case, the clock frequency is usually set accordingly to the slowest operation path. Zhang et al. [25] faced this problem when implementing Gotoh with a systolic array and overcame this concern by using several clocks that operated on the same frequency but ran on different phases. This is an efficient solution even though it requires careful and time-consuming tuning in each target FPGA.

Unlike the architectures presented in Section 3, we propose the use of wavefront array processors [32] instead of systolic arrays, for our FPGA-based architectures that execute DIALIGN. We claim that wavefront array processors are better suited to deal with our problem since communication between processing elements is asynchronous, occurring exactly when output data are available. This is useful when there are timing uncertainties due to multiple possible completion instants. Nevertheless, wavefront designs are more elaborate than the systolic ones since they need a handshaking protocol.

Wavefront array processors have been successfully used to implement applications in many research domains, such as motion estimation [33], [34] and computation of eigenvalues [35], among others.

The FPGA-based architectures proposed in this section follow the idea illustrated in Fig. 4, where the query sequence bases are stored in the processing elements and the database sequence passes through them. In this way, if there are enough processing elements, each antidiagonal is calculated in parallel. Therefore, time complexity of the sequential DIALIGN, which is $O(nm)$, is reduced to $O(n+m)$, where n and m are the sizes of the sequences.

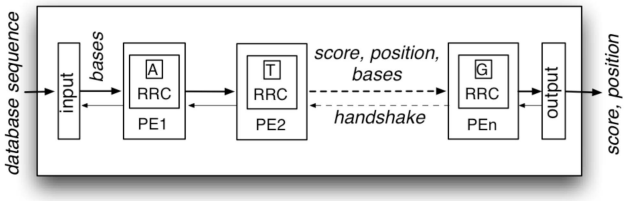

Fig. 5. Basic wavefront array design.

The output produced depends on the architecture being used: DIALIGN-Score (Section 4.2) produces as output the similarity score, and its positions in the similarity matrix. DIALIGN-Alignment (Section 4.4) outputs the DIALIGN alignment itself.

4.1 Basic DIALIGN Architecture

Like most of the previous work (Section 3), we will parallelize the antidiagonal calculation of the dynamic programming matrices using a processor array (Fig. 4). Instead of using a systolic array, a wavefront array processor design is proposed. In addition, because the recurrence relations of DIALIGN (3,4) are different from the ones in SW or Gotoh (Section 2.2), an entirely distinct design must be made for each Processing Element (PE).

Fig. 5 illustrates the basic wavefront array processor design that is used in DIALIGN-Score (Section 4.2) and DIALIGN-Alignment (Section 4.4). In this architecture, each PE is responsible to calculate one column of the DIALIGN matrices (*score* and *prec* in (3) and (4)). In a previous phase, the query sequence is read and each PE stores one base (represented by A, T, and G in Fig. 5). The *input* and *output* elements implement a bidirectional handshaking protocol, containing input and output data transfer buffers. The database sequence flows from left to right (one base at a time) and is represented by coarse arrays. In each processing element, all computations are made by the RRC (*Recurrence Relations Component*), which can be modified to implement different recurrence relations. At the end of the computation, the output is produced. Since we opted to design a wavefront array processor, instead of a systolic array, we included a handshaking protocol in each processing element. This protocol guarantees that data are sent to the next element as soon as it can process it.

The finite state machine diagram of the RRC module is represented in Fig. 6.

The initial state for every processing element is *Configuration*. In this state, the bases composing the query sequence are input through the wavefront array and stored in each processing element. After this phase, the elements enter into the *Computation* stage and execute the RRC module (Fig. 5). However, if there are fewer bases than processing elements, the remaining processing elements enter into the *Idle* state. Elements in this state simply receive input data and send the same data to the next element. After calculating the recurrence relations, the

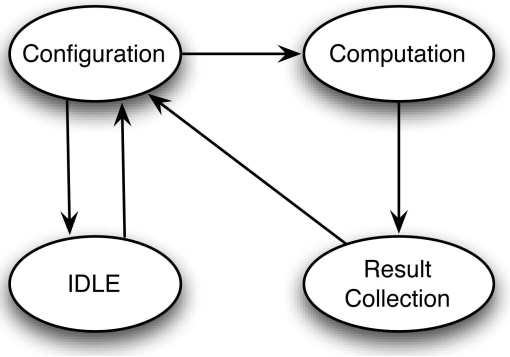

Fig. 6. Transition state diagram for the PE elements.

elements enter into the *Result Collection* state. In this state, the results are output from the circuit.

4.2 The DIALIGN-Score Architecture

The goal of the DIALIGN-Score architecture is to find the best DIALIGN score and its position. In order to do this in hardware, the following modifications were applied to the original DIALIGN algorithm (Section 2.3). First, we set $sm = l$ in the probability calculation. Second, the ln logarithm was replaced by a base 2 logarithm.

The linear wavefront array calculates the antidiagonals as shown in Fig. 7, using as a base the generic wavefront design (Fig. 5). In Fig. 7, the scores already calculated are shown in gray. The border between the gray and white section shows the antidiagonal being calculated. Diagonals greater than the threshold T are shown in black. For a diagonal that ends in position (i, j), the processing element decides if it will be extended or ended and, in this case, whether it can be consistently aligned to other diagonals (Section 2.3).

An architecture that performs DIALIGN must contain formulae (3) and (4). To improve the performance, finding and alignment of diagonals (fragments) are done simultaneously in the wavefront vector. The algorithm for each processing element is shown in Fig. 8.

In this algorithm, the best diagonal alignment ending before position (i,j) in the dynamic programming matrix is retrieved in line 1. In this position, the database and query bases are compared. If both are equal, the current diagonal is extended, and it does not need to be aligned (line 3). If the bases do not match, the diagonal has ended at position (i,j) and may be added to the alignment (line 4). A diagonal having a score below the threshold is discarded (lines 5 and 6). If the diagonal is consistent with current alignment, it is added to the alignment, extending it (lines 10 and 11). However, it is possible for a current diagonal to be

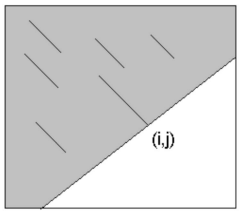

Fig. 7. DIALIGN dynamic programming matrix calculation.

```
Compute_DIALIGN_PE(database_base, prec[i,j-1], score[i,j-1], D[i-1][j-1])
1: prec[i,j] = find_prec(prec,score) ;
2: if match(i,j)
3:    D[I,j] = extend_current_diagonal();
4: else
5:    if (w(D[I,j]) < T)
6:       discard(D[I,j]);
7:    else
8:       if (consistent(D[i,j], prec[i-1,j], prec[i,j-1])
9:          begin
10:            prec[i,j] = D[i,j];
11:            score[i,j] = σ (D[i,j]);
12:         end
13:      else
14:         if ((w(D[i,j]) > prec[i-1,j]) and (w(D[i,j]) > prec[i,j-1])
15:            begin
16:               prec[i,j] = D[i,j];
17:               score[i,j] = w(D[i,j]);
18:            end
19:         endif
20:      endif
21:   endif
22:endif
23:best_diagonal = prec[i,j];
24:best_score = score[i,j];
25:send_to_next_PE(database_base,prec[i,j], score[i,j], D[i,j]);
26:end
```

Fig. 8. Algorithm executed in each processing element.

inconsistent with the alignment and have a better score than this entire alignment. In this case, the inconsistent diagonal will become the new alignment (line 16 and 17). The best score obtained and the last diagonal are stored in the Processing Element in lines 23 and 24. Finally, the values required for the recurrence relation are forwarded to the next Processing Element (line 25).

Fig. 9 shows the structure of the RRC element that was automatically generated from a SystemC description of DIALIGN. It is based on a general purpose processor architecture, which consists of a control unit and a datapath. The datapath is composed by two switching networks, a register bank and the recurrence module. The register bank stores the values used in recurrence relations. The recurrence module implements all operations needed to execute the algorithm. Its inputs are selected from the register bank through a switching network based on multiplexers. The outputs of the recurrence module are stored back in the register set through another switching network. Switching networks are needed because each register may store

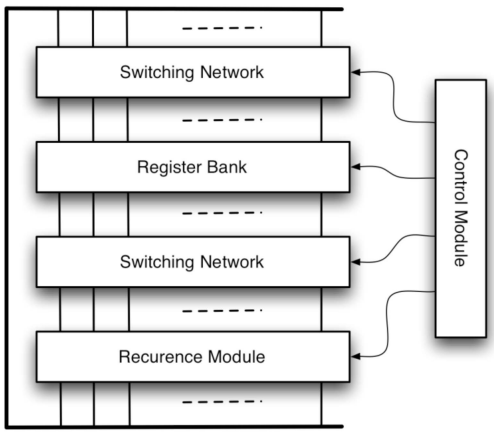

Fig. 9. Recurrence relation component (RRC).

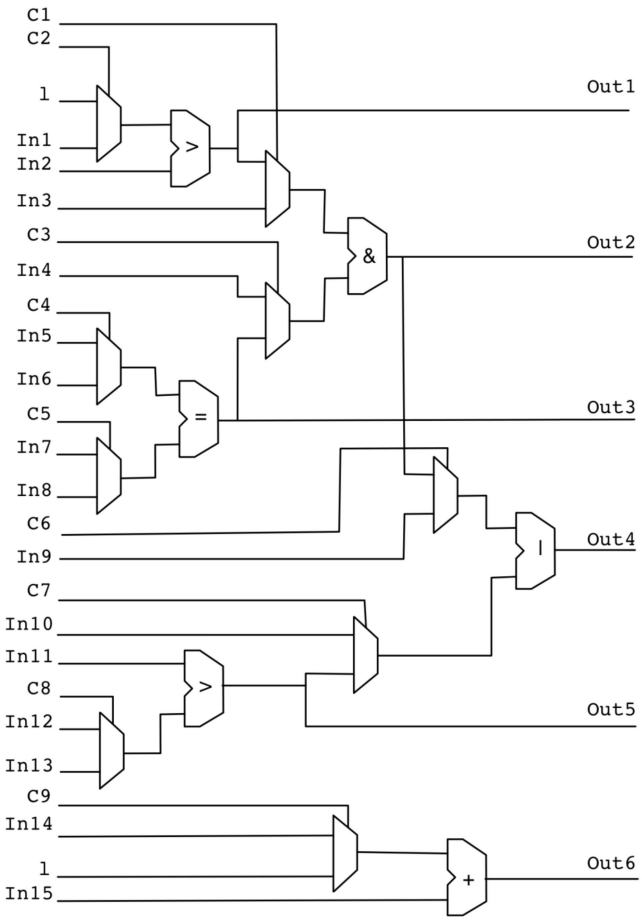

Fig. 10. Recurrence module implemented in DIALIGN-Score.

different variables and the inputs needed by the operators may come from different registers during the execution of the algorithm.

Fig. 10 presents the recurrence module circuit. Input values (from In1 to In15) and control lines (C1 to C9) for the multiplexers are on the left side and outputs (Out1 to Out6) are on right side. This circuit is utilized many times to perform all relations.

The adder (+) in In15 and C9 are utilized to extend weights $w(D)$ of the current diagonal D when a match happens. In15, In14 and "+" are used to calculate the sum of scores $\sigma(D)$. The comparator "=" verifies if the bases are equal and whether some flag values are equal to zero or one (In5 to In8 depend on C4 and C5 giving Out3). The comparator ">" decides if $w(D)$ is above T and is also used to find score(i,j) (3). The recurrence relation in (4) is implemented by ">", "=", and "&". The first line in (4) is computed by the "=" comparator. The second and third lines are translated to ">", "=", and "&" according to the expression $(\mathrm{score}(i,j) > score(i,j-1)\&(score(i,j) = score(i-1,j)))$.

To eliminate the current diagonal $D_{i,j}$ if it is inconsistent, we must test wether the ending position of the previously aligned diagonal is greater than the starting point of the current diagonal. To calculate this, an OR ("|") and two ">" are used. If $D_{i,j}$ is inconsistent with $\mathrm{prec(i-1,j)}$ or it is inconsistent with $\mathrm{prec}(i,j-1)$ then $D_{i,j}$ is inconsistent.

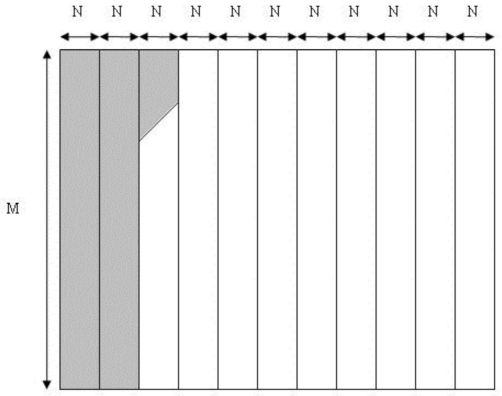

Fig. 11. Partition technique implemented in DIALIGN-Score.

Like most of the approaches presented in Section 3, we also designed a partition method to compare query sequences that have more bases than the FPGA systolic elements.

Fig. 11 illustrates the partition technique used in DIALIGN-Score, where the gray section of the matrix is already calculated. The query sequence is split into sets composed of N bases, where N is the total number of processing elements in the wavefront array. The database sequence flows entirely through the PEs, and its elements are compared, each step, with N characters of the query sequence. This process continues until all the query sequence characters are compared.

In our design, data is mainly stored at the register banks inside each Processing Element (PE), at the RRC component (Fig. 9). The only exception occurs when the partition technique is used. In this case, the last calculated column is stored into the RAM memory present in the FPGA board.

4.3 Executing DIALIGN in Linear Space

As discussed in Section 2.3, the DIALIGN alignment is retrieved by executing a traceback procedure over the *prec* matrix. The initial point of departure for this traceback is the highest score contained in the *score* matrix. If the sizes of the sequences are equal to n, the space needed is quadratic $O(n^2)$ [4].

We propose a variant of DIALIGN that executes in linear space. The proposed variant calculates the whole DIALIGN matrices but only stores, for each column j, the maximum score $(score[j].value)$, the row where it occurs $(score[j].row)$ and its size $(score[j].size)$. Moreover, the ending position of the preceding fragment $(prec\_frag[j].row, prec\_frag[j].col)$ of the fragment that has the highest score in column j is stored.

The main idea of our algorithm is illustrated in Fig. 12. At the end of the first DIALIGN execution, the vectors $prec\_frag$ and *score* are filled using formulae (3) and (4).

The traceback procedure starts at the highest score. In Fig. 12, the highest score occurs in column 80 and row 80 $(score[80].row)$. The last fragment ends at this position. This fragment is guaranteed to belong to the optimal alignment (because it has the highest score), so it is stored in the vector *Alignment*. The previous fragment ends at row 41 and column 44 ($prec\_frag[80].row$ and $prec\_frag[80].col$). We compare $score[44].row$ with $prec\_frag[80].row$ to verify if the fragment stored in column 44 belongs to the optimal alignment. In this case, it does and so the fragment is also

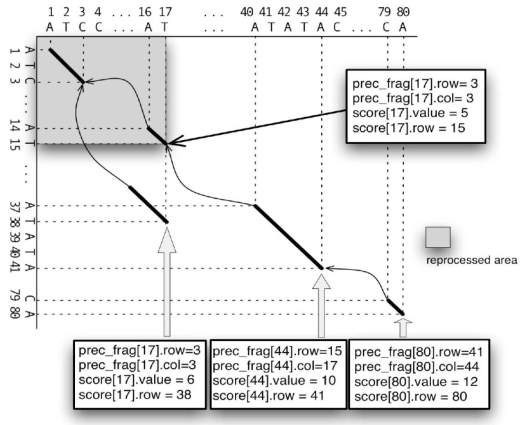

Fig. 12. Example of the execution of our variant of DIALIGN with sequences s and t, both with 80 base pairs.

stored in the vector *Alignment*. The previous fragment ends at row 15 and column 17 ($prec\_frag[44].row$ and $prec\_frag[44].col$, respectively). However, in this case, the fragment that ends in row 38 was stored ($score[17].row$). Thus, this is not the fragment we are searching for, and the area that comprises rows 1 to 15 and columns 1 to 17 needs to be reprocessed. The traceback procedure will now start from the fragment that ends in row 15, column 17, since it is known that it belongs to the optimal alignment. This procedure is repeated until all the fragments are retrieved.

The algorithm that implements the idea illustrated in Fig. 12 is shown in Fig. 13. In this algorithm, the DIALIGN alignment for sequences s and t for a threshold T (Section 2.3) is obtained.

In line 1 (Fig. 13), the vector *Alignment* is initialized with zeroes. Each position of this vector will contain the row and column where the fragment ends as well as its size. In line 2, the DIALIGN algorithm is executed for the entire sequences s and t, with sizes |s| and |t|, respectively, and a threshold T. This execution returns, for each column j, the highest score as well as information about the preceding fragment.

In line 4, the global maximum score is computed and stored in the structure $\mathrm{Max\_score}\{\mathrm{value, col, row}\}$. The position i, j where the maximum score occurs represents the ending of the last fragment, and $prec\_frag[j].row$, $prec\_frag[j].col$ contain the ending of the previous fragment. In line 7, a traceback is performed while the row where the preceding fragment ends is equal to the row where the highest score occurred (Fig. 12). Each fragment retrieved is stored in the vector *Alignment* (line 10). When this condition does not hold, it means that the information that is stored in $prec\_frag[j]$ does not concern a fragment that belongs to the optimal alignment. For this reason, we need to reexecute the DIALIGN (line 14) over the same sequences but only a part of the *score* and *prec* matrices will be calculated, defined by the position i, j where the last fragment of the optimal alignment has occurred ($Max\_score.row, Max\_score\_col$). At the end of the execution, this algorithm outputs the set of fragments, from the end to the beginning, which represents the optimal DIALIGN alignment. Considering the example illustrated in Fig. 3, the algorithm in Fig. 13 will output the following tuples *(row, column, size)*: (14,12,3), (9,8,4), and (3,4,3). With this information at hand, it is trivial to reconstruct the alignment.

```
Retrieve_DIALIGN_Alignment(s, t,T)
1: initializes Alignment.score, Alignment.prec_row, Alignment.prec_col;
2: DIALIGN (s,t,|s|,|t|,T) returns score, prec_frag;
3: Partial_score = 0;
4: Max_score = Compute_maximum (score);
5: while Partial_score < Max_score.value
6: begin
7:    while score[Max_score.col].row = Max.score.row
8:    begin
9:        Partial_score = Partial_score + score[prec_frag[Max_score.col]].value;
10:       Store_fragment(Alignment);
11:       Max_score.row = Prec_frag[Max_score.col]. row
12:       Max_score.col = Prec_frag[Max_score.col].col
13:   end while
14:   DIALIGN(s, t, Max_score.row, Max_score.col, T) returns score, prec_frag;
15: end while
16: returns Alignment
```

Fig. 13. Algorithm to retrieve DIALIGN alignments for sequences s and t. with sizes $|s|$ and $|t|$, respectively, and threshold T.

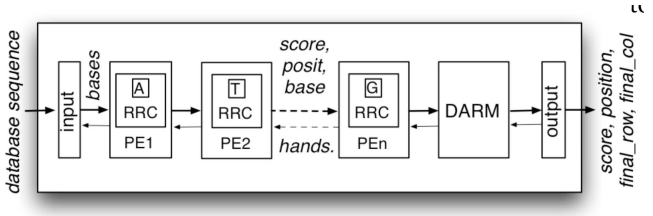

Fig. 14. Wavefront array processor designed to execute DIALIGN-Alignment.

4.4 The DIALIGN-Alignment Architecture

The goal of the DIALIGN-Alignment architecture is to obtain the DIALIGN alignments in linear space. In order to do that, we implemented the algorithm proposed in Section 4.3 in a wavefront array processor that is illustrated in Fig. 14.

In DIALIGN-Alignment, each PE in Fig. 14 executes exactly the same algorithm as DIALIGN-Score (Fig. 8). The only difference is that in DIALIGN-Aligment each element stores not only the highest score and its position but also information about the preceding fragment (*prec-frag* in Fig. 12). The *Dialign Alignment Retrieval Module* (DARM) component (Fig. 14) is fundamental to DIALIGN-Alignment since it executes most part of the algorithm shown in Fig. 13. Its goal is to determine which part of the dynamic programming DIALIGN matrices must be recalculated (e.g., the striped area in Fig. 12).

After detecting that the DIALIGN computation (line 2 in Fig. 13) is finished, the DARM begins to collect the vectors *score* and $prec\_frag$ that were calculated by each RRC. The position of the last fragment that belongs to the optimal alignment is obtained (e.g., row 15 and column 17 in Fig. 12) by executing lines 5 to 12 (Fig. 13).

The algorithm executed in the DARM module is shown in Fig. 15. Recall that, in DIALIGN, an alignment is defined as a sequence of fragments. Also, the last fragment contains the alignment final score and it is linked to the previous fragment (Section 2.3). Given the wavefront array processor composed by *PE 1* to *PE N* (Fig. 14), the last fragment of the alignment is the first to enter the DARM module. The DARM module retrieves as many as possible fragments from a given final

```
Dialign_Alignment_Retrieval_Module (actual_fragment)
1:  Receive (actual_fragment);
2:  Increment counter;
3:  if counter<=size_vector
4:     if  dialign_score > better_score
5:       Final_column = size_vector – counter;
6:       Final_row = actual_fragment.row;
7:       Position_previous_fragment =  actual_fragment.precedent;
8:       better_score = dialign_score;
9:     else
10:      if actual_fragment.position==Position_previous_fragment
11:        Final_column = size_vector – counter;
12:        Final_row = actual_fragment.row;
13:        Position_previous_fragment =   actual_fragment.precedent;
14:   Forward (actual_fragment);
15: returns Final_column, Final_row;
```

Fig. 15. Algorithm to executed in the DARM component

fragment and only stores the last fragment processed. If the last fragment processed is in position (0,0), the best DIALIGN alignment is completely found and no further reprocessing is needed. Otherwise, information about the last fragment processed (Final_column, Final_row in Fig. 15) is the output of the DARM module, which will be utilized to restrict the next processing step (line 13 in Fig. 13).

Fig. 16 shows the circuit that implements this part of the algorithm. Having the last row and column that mark the limits of the area to be reprocessed, the wavefront array processor is instructed to execute the next round. If zero values are obtained for a row and column, no further reprocessing is done.

5 Experimental Results

Our proposed architectures were designed in SystemC [9] and translated to Verilog with the FORTE tool [10]. Both the DIALIGN-Score and DIALIGN-Alignment were synthesized for the FPGA Altera STRATIX 2 EP2S180F1508I4 using QUARTUS II. In order to obtain the speedups, we implemented DIALIGN-Score and DIALIGN-Alignment in C and ran these programs on a Pentium 4 3 GHz.

Before the synthesis and simulation with Quartus II, our FPGA design was tested as follows: First, we ran a behavioral simulation with the tools provided by FORTE.

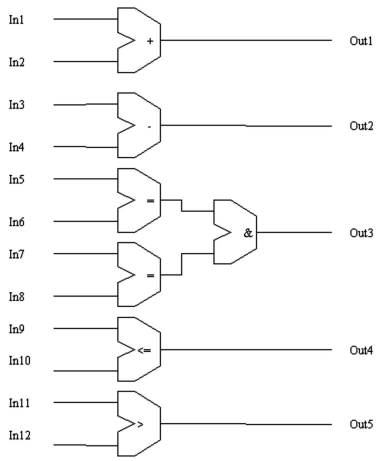

Fig. 16. Circuit implemented in the DARM module.

TABLE 2
Synthesis at the Stratix II EP2S180F1508I4

Architecture	PEs	ALUTs	Regs	I/O	Clock (MHz)
DIALIGN-Score	200	77%	62%	14%	74.48
DIALIGN-Alignment	128	81%	91%	14%	100.10

Several tests were made and the results were compared to the ones produced by the software implementation. Second, we used the FORTE's Cynthesizer tool to generate a Verilog RTL description of the system. In this phase, the same tests were run with a Synopsys VCS simulator, and the results were compared. Finally, the Verilog files generated by FORTE were used for synthesis and simulation for the Altera EP2S180F1508I4 device using the Quartus II tool.

Information related to this synthesis is shown in Table 2. We were able to accommodate 200 Processing Elements (PEs) for DIALIGN-Score and 128 PEs for DIALIGN-Alignment. This reduction in the number of PEs for DIALIGN-Alignment can be explained by the addition of the DARM element (Fig. 14) and also by the need to store more data in each PE.

5.1 Results for DIALIGN-Score

In order to verify the speedup of DIALIGN-Score, we implemented DIALIGN in C, generating an optimized C program. Instead of using the DIALIGN implementation available at *bibserv.techfak.uni-bielefeld.de/dialign*, we opted to implement optimized C programs for DIALIGN-Score and DIALIGN-Alignment, using the same algorithms that were implemented in hardware, for the following reasons. First, in DIALIGN-Score, some simplifications were made (Section 4.2) over the original DIALIGN proposal. Moreover, the DIALIGN-Score and DIALIGN-Alignment versions do not store the entire dynamic programming matrices, using much less memory than the original DIALIGN.

The C programs were compiled with gcc 4.2.4 with option −O2. We used the C program, and our synthesized circuits to compare real DNA sequences retrieved from the NCBI site (www.ncbi.nlm.nih.gov). The sequences compared, and their sizes are illustrated in Table 3.

Table 4 presents the wallclock times and the speedups achieved by DIALIGN-Score in some DNA comparisons. Note that, in this table, the wallclock times do not include data transfer times (FPGA prototype) nor disk read operations (software implementation). The speedups obtained are between 383.41 and 343.03. The best speedups were achieved when comparing sequences AM270375 and AL590443, of sizes 169,786 and 194,439, respectively. In this case, the software comparison took 3 h 4 min in software and 28.83 s in our FPGA prototype, leading to a speedup of 383.41 (Table 4).

The pairwise stage of multiple alignment with DIALIGN was performed with four variants of Human Adenovirus (Table 3). Each cell in Table 5 shows the time in seconds for a given pairwise alignment for both the FPGA architecture and the software implementation.

TABLE 3
DNA Sequences Used in the DIALIGN-Score and DIALIGN-Alignment Comparisons

Name	Identification	Size (bp)
Aspergillus niger contig An18c0160	AM270408	121,589
Aspergillus niger contig An16c0230	AM270375	169,786
Encephalitozoon cuniculi	AL590443	194,439
Torque teno midi virus	NC_009225	3,245
Torque teno midi virus DNA	AB290918	3,242
Human adenovirus B	NC_004001	34,794
Human adenovirus C	NC_001405	35,937
Human adenovirus D	NC_002067	35,100
Human adenovirus E	NC_003266	35,994
Rhizobium leguminosarum	AM236080	5,057,142
Synthetic 128bp sequence	SS128	128
Synthetic 10Mbp sequence	SS10000000	10,000,000

TABLE 4
Speedups Achieved by DIALIGN-Score

Query Seq size	Database Seq size	Time FPGA (s)	Time software (s)	Speedup
169,786	194,439	28.83	11,053.70	383.41
121,589	169,786	17.9	6,812.00	380.55
3,245	3,242	0.01	3.48	348.00
200	10,000,000	1.74	661.39	343.03

TABLE 5
Pairwise Aligment Times for DIALIGN-Score and the Software Implementation

	Time (s) FPGA / Software		
Seq	34794bp	35937bp	35100bp
35937bp	1.09 / 415.31	---	---
35100bp	1.07 / 406.81	1.11 / 419.35	---
35994bp	1.10 / 416.33	1.14 / 431.42	1.11 / 420.12

In Table 5, six comparisons were made. The total time for software comparison and the FPGA synthesized circuit were 2,509.34 s and 6.62 s, respectively. The speedup achieved, when considering all six comparisons, was 379.05.

5.2 Results for DIALIGN-Alignment

In order to verify the speedup of DIALIGN-Alignment, we implemented the variant of DIALIGN proposed in Section 4.3 in C, generating an optimized C program. We used the C program and the FPGA prototype to compare noncoding RNA (nc-RNA) sequences retrieved from the *miRBase Sequence database* (at www.ncrna.org), the *Functional RNA Project* (at www.ncrna.org) and the *Arabidopsis Small RNA Project* (at http://asrp.cgrb.oregonstate.edu).

Table 6 shows information about the nc-RNA sequences compared and Table 7 presents the results obtained.

In Table 7, the first two columns show the sequences compared. The threshold T used in all comparisons was 0, which means that a new fragment begins every time a match occurs. This parameter was set to 0 in order to generate a great number of fragments, thus, possibly increasing the number of rounds. The fourth column shows how many times the circuit was reprocessed (Fig. 12, gray region). The wallclock times (s) for the FPGA and for the C program are shown in the next columns. Finally, the last column presents the speedup achieved by the FPGA.

TABLE 6
Nc-RNA Sequences Used to Obtain the Alignments with DIALIGN-Alignment

Name	Identification	Size
Arabdopsis thaliana	MIR156d|stem	118
Caernorhabditis elegans	MI0000001	99
Homo sapiens let-7a-2	MI0000061	72
Mus musculus 18-day embryo	FR220660	128
Homo sapiens truncated RAD51	FR379163	128
Archaeoglobus fulgidus small non-messenger RNA	FR063329	128
Methanocaldococcus jannaschii HgcB RNA	FR000859	127
Pseudomonas putida KT2440	FR003139	127

TABLE 7
Results Obtained to Align Small ncRNA Sequences

Id. S1	Id. S2	T	Rounds	Time FPGA	Time CPU	Speedup
MIR156d|stem	MI0000001	0	6	0.000428s	0.007s	16.35
MIR156d|stem	MI0000061	0	10	0.000702s	0.009s	12.82
MI0000061	MI0000001	0	5	0.000374s	0.004s	10.69
FR000859	FR003139	0	14	0.001025s	0.016s	15.60
FR000859	FR220660	0	13	0.000945s	0.014s	14.80
FR000859	FR063329	0	9	0.000660s	0.011s	16.60
FR000859	FR379163	0	11	0.000803s	0.012s	14.90
FR003139	FR220660	0	10	0.000738s	0.013s	17.60
FR003139	FR063329	0	11	0.000806s	0.013s	16.10
FR003139	FR379163	0	13	0.000933s	0.013s	13.90
FR220660	FR063329	0	13	0.001046s	0.016s	15.30
FR220660	FR379163	0	11	0.000890s	0.015s	16.80
FR063329	FR379163	0	11	0.000897s	0.013s	14.50

The wallclock time to compare the first two ncRNA sequences for the optimized C program was 7 ms and the FPGA took 0.42 ms, achieving a speedup of 16.35. Note that the execution times depend not only on the size of the sequences but also on the number of rounds executed. The best speedup achieved to align these small nc-RNAs with DIALIGN-Alignment was 17.6.

To obtain similarities between a ncRNA and a DNA sequence is very important, since it can help us to understand genetic regulation mechanisms. Therefore, we used DIALIGN-Alignment to compare the ncRNAs MI0000061 and MIR156d | stem, shown in Table 6, with four DNA sequences illustrated in Table 3.

Table 8 presents a very interesting result. The speedups seem to be dependent mostly on the size of the smaller sequence. This fact was also observed by [27]. Note the huge variation on the size of the DNA sequence in the MIR156d | stem comparison (194,431 bp and 5,047,142 bp, respectively). This variation is reflected on the execution times, which are, as we expected, proportional to the sizes of the sequences.

Nevertheless, the speedups obtained are very similar (130.40 and 126.61, respectively). The same observation can be made for the MI0000061 case. This very small variation on

TABLE 8
Results Obtained to Align Small ncRNAs with Long DNA Sequences

Id. RNA	Id. DNA	T	Rounds	Time FPGA	Time CPU	Speedup
MI0000061	AM270408	0	12	0.020441s	1.57s	76.80
MI0000061	AM270375	0	12	0.028396s	2.23s	78.53
MI0000061	AL590443	0	12	0.032188s	2.59s	80.46
MIR156d|stem	AM236080	0	11	0.809720s	102.52s	126.61
MIR156d|stem	AL590443	0	15	0.001025s	4.05s	130.40
SS128	SS10000000	0	10	1.599409s	226.14s	141.38

the speedups when varying the size of the longest sequence can be explained by the fact that the size of the smallest sequence determines the number of elements used at the FPGA and, consequently, the amount of parallelism that can be achieved. This is clear in Table 8, where the best speedups for real sequences are achieved with the 118 bp sequence ($\mathrm{MIR156d} \mid \mathrm{stem}$). Considering the best speedups for $\mathrm{MIR156d} \mid \mathrm{stem}$ and MI0000061 (130.40 and 80.46, respectively) an augmentation of 61.01 percent in the size of the smallest sequence resulted in an increase of 61.70 percent on the speedup. This shows clearly that a very high level of parallelism is achieved by DIALIGN-Alignment, with very low overheads.

In the last row in Table 8, a comparison between two synthetic DNA sequences with sizes 128 and 10,000,000, respectively, is shown. In this case, the speedup obtained was 141.38 and the execution time was reduced from 3 min 45 s to 1.59 s. This result is a very good one, considering that not only the DIALIGN matrix is calculated but the optimal alignment is also retrieved.

5.3 Bandwidth Requirements

In this section, we assume that the FPGA is connected to a host computer through a PCI Express 1.x interface (www.pcisig.com), which has a bandwidth of 64 Gbps.

In the DIALIGN-Score architecture, interaction with the CPU is needed at the following phases: Configuration, Computation, and Result Collection (Fig. 6). During the configuration phase only a few bits are needed. Essentially, a database sequence base (2 bits), a flag (1 bit) and a protocol indication (1 bit) are transferred. The maximum bandwidth required is then $(4 \times 74.48\,\text{Mhz}\ (\text{Table 2})) = 0.297$ Gbps. This is far below the maximum PCI-E speed of 64 Gbps (8.0 GB/s). During the computation phase, the scores are calculated and kept inside the wavefront vector. Thus, the only relevant data to enter the vector each time is a query sequence base (2 bits), a flag (1 bit) and protocol indication (1 bit), which also require 0.297 Gbps. In DIALIGN-Score, the wavefront vector outputs the following information: score.value (8 bits), score.row (24 bits), flag (1 bit), and protocol (1 bit). The number of bits required is then $34(8 + 24 + 2)$. At a clock rate of 74.48 MHz, the bandwidth needed in the output phase is therefore 2.53 Gbps, which is also fairly below the PCI-E 1.x and PCI-E 2 capabilities.

In the DIALIGN-Alignment architecture, the same considerations made for the Configuration and Execution phases in the previous paragraph are valid. Nevertheless, in the Collect Result phase, fragments that compose the optimal DIALIGN alignment are transferred from the FPGA to the host, and that phase places the most demands on bandwidth. In this phase, the values of $prec\_frag$ (Fig. 13) need to be retrieved from the wavefront vector. In our implementation, we set $prec\_frag.col$ and $score.value$ to 24 bits and $prec\_frag.row$ and $prec\_frag.value$ to 7 bits, so that a large database comparison could be done $(10{,}000{,}000 \times 128)$. The total number of bits required for each fragment transfer is then $64(2 \times 7 + 2 \times 24 + 2)$, where two bits are used as flags. At a clock rate of 100.1 MHz, DIALIGN-Alignment would need 6.4 Gbps. This is far below the bandwidth of PCI-E 1.x.

In view of the considerations above, it is clear that the FPGA designs proposed in this paper will work properly with the interconnections that usually connect the FPGA to the CPU: PCI-E 1 (64 Gbps), PCI-E 2 (128 Gbps), Hypertransport 3.0 (332.8 Gbps), and NUMALink (12.8 Gbps).

6 Conclusions and Future Work

In this paper, we proposed and evaluated two FPGA wavefront array processors to execute DIALIGN in linear space. The first architecture, called DIALIGN-Score, is able to compare sequences of any size, retrieving the DIALIGN score. The second architecture (DIALIGN-Alignment) implements a variant of the DIALIGN algorithm that runs in linear space and is able to retrieve DIALIGN alignments.

The results obtained with the FPGA Altera STRATIX 2 EP2S180F1508I4 show that speedups up to 141.38 can be achieved by DIALIGN-Alignment, when compared to an optimized C program running on a Pentium 4 3 GHz. In this case, the execution time was reduced from 3 min 45 s to 1.59 s. We observed that the speedup is proportional to the size of the smallest sequence, because it is this sequence that is put directly onto the processing elements. Additionally, a speedup of 383.41 was achieved for DIALIGN-Score, when comparing real DNA sequences.

Moreover, in future work, we intend to design a partitioning technique for DIALIGN-Alignment that will allow the smallest sequence to have sizes bigger than the number of processing elements. Although we have already implemented a partition technique for the DIALIGN-Score architecture, partitioning is a much more complex problem when the alignment is retrieved. In this particular case, much more information must be stored. Nevertheless, the partition technique is very important in this case, because it will enable sequences of any size to be compared in DIALIGN-Alignment.

We also intend to integrate our DIALIGN FPGA-based architectures to a NNUS (Nonuniform Node Uniform System) high-performance reconfigurable computing platform [8]. Finally, we plan to design hardware-based solutions for other important problems in bioinformatics such as RNA secondary structure prediction and phylogenetic tree generation, among others.

Acknowledgments

This work was supported in part by grants from Capes/Brazil (4740/07-6), CNPq/Brazil, FINEP (010801660/2007), and FAPDF (8-004/008-2007). Part of this work was done while Dr. A.C.M.A. de Melo was a Post-Doctoral Fellow at the PARADISE Research Laboratory.

REFERENCES

[1] C. Wang, B.B. Zhou, and A. Zomaya, "Scaling up Genome Similarity Search Services through Content Distribution," *Proc. Int'l Conf. Parallel Processing (ICPP),* 2007.

[2] R. Durbin, S. Eddy, A. Krogh, and G. Mitchson, *Biological Sequence Analysis.* Cambridge Univ. Press, 1998.

[3] T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences," *J. Molecular Biology,* vol. 147, no. 1, pp. 195-197, Mar. 1981.

[4] B. Morgenstern, K. Frech, A. Dress, and T. Werner, "DIALIGN: Finding Local Similarities by Multiple Sequence Alignment," *Bioinformatics,* vol. 14, no. 3, pp. 290-294, Mar. 1998.

[5] R. Siddhartan, "Sigma: Multiple Alignment of Weakly-Conserved Non-Coding DNA Sequence," *BMC Bioinformatics,* vol. 7, no. 143, Mar. 2006.

[6] M. Schmollinger, K. Nieselt, M. Kaufman, and B. Morgenstern, "DIALIGN P: Fast Pair-Wise and Multiple Sequence Alignment using Parallel Processors," *BMC Bioinformatics,* vol. 5, no. 128, Sept. 2004.

[7] B. Morgenstern, "DIALIGN: Multiple DNA and Protein Sequence Alignment at BiBiServ," *Nucleic Acids Research,* vol. 32, pp. W33-W36, Mar. 2004.

[8] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, "The Promise of High-Performance Reconfigurable Computing," *Computer,* vol. 41, no. 2, pp. 69-76, Feb. 2008.

[9] IEEE Std 1666-2005, *IEEE Standard SystemC Language.* IEEE Press, 2006.

[10] Forte Design Systems, "Cynthesizer User's Guide For Cynthesizer 2.4.0," 2005.

[11] Altera Corporation, "Introduction to the Quartus II Software Version 8.0," Technical Manual,www.altera.com/literature/manual/intro_to_quartus2.pdf.

[12] S. Needleman and C. Wunsh, "A General Method Applicable to the Search of Similarities in the Amino Acid Sequence of Two Proteins," *J. Molecular Biology,* vol. 48, pp. 443-453, 1970.

[13] D. Mount, *Bioinformatics: Sequence and Genome Analysis.* C.S. Harbor Lab Press, 2004.

[14] O. Gotoh, "Am Improved Algorithm for Matching Biological Sequences," *J. Molecular Biology,* vol. 162, pp. 705-708, 1982.

[15] E.W. Myers and W. Miller, "Optimal Alignments in Linear Space," *Computer Applications in the Biosciences (CABIOS),* vol. 4, no. 1, pp. 11-17, 1988.

[16] D.S. Hirshberg, "A Linear Space Algorithm for Computing Maximal Common Subsequences," *Comm. ACM,* vol. 18, pp. 341-343, 1975.

[17] H.T. Kung, "Why Systolic Architectures?," *Computer,* vol. 15, no. 1, pp. 37-46, Jan. 1982.

[18] R.J. Lipton and D. Lopresti, "A Systolic Array for Rapid String Comparison," *Proc. Chapel Hill Conf. VLSI,* pp. 363-376, 1985.

[19] D. Lavenier, "Speeding up Genome Computations with a Systolic Accelerator," *SIAM News,* vol. 31, no. 8, pp. 6-7, 1998.

[20] Y. Yamaguchi, T. Maruyama, and A. Konagaya, "High Speed Homology Search with FPGAs," *Proc. Pacific Symp. Biocomputing (PSB),* 2002.

[21] K. Puttegowda, W. Worek, N. Pappas, A. Dandapani, and P. Athanas, "A Run-Time Reconfigurable System for Gene-Sequence Searching," *Proc. Int'l Conf. VLSI Design,* pp. 561-566, 2003.

[22] A. Marongiu, P. Palazzari, and V. Rosato, "A Specialized Hardware Device for the Protein Similarity Search," *Concurrency and Computation: Practice and Experience,* vol. 16, pp. 917-931, 2004.

[23] T.F. Oliver, B. Schmidt, and D.L. Maskell, "Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs," *Proc. ACM/SIDA Int'l Conf. Field Programmable Gate Arrays,* pp. 229-237, 2005.

[24] A. Di Bias et al., "The UCSC Kestrel Parallel Processor," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 1, pp. 80-92, Jan. 2005.

[25] P. Zhang, G. Tan, and G.R. Gao, "Implementation of the Smith-Waterman Algorithm on a Reconfigurable Supercomputing Platform," *Proc. Int'l Conf. High Performance Networking and Computing,* pp. 39-48, 2007.

[26] X. Jiang, X. Liu, L. Xu, P. Zhang, and N. Sun, "A Reconfigurable Accelerator for Smith-Waterman Algorithm," *IEEE Trans. Circuits and Systems II,* vol. 54, no. 12, pp. 1077-1081, Dec. 2007.

[27] O. Storaasli, W. Yu, D. Strensky, and J. Maltby, "Performance of FPGA-Based Biological Applications," *Proc. Cray User Group Meeting,* May 2007.

[28] D.J. Lipman and W.R. Pearson, "Rapid and Sensitive Protein Similarity Searches," *Science,* vol. 227, pp. 1435-1441, 1985.

[29] M. Abouellail, E. El-Araby, M. Taher, T. El-Ghazawi, and G.B. Newby, "DNA and Protein Sequence Alignment with High Performance Reconfigurable Systems," *Proc. NASA/ESA Conf. Adaptive Hardware and Systems,* 2007.

[30] B. Harris, A.C. Jacob, J.M. Lancaster, J. Buhler, and T.D. Chamberlain, "A Banded Smith-Waterman FPGA Accelerator for Mercury BlastP," *Proc. Int'l Conf. Field Programmable Logic and Applications,* pp. 765-769, 2007.

[31] Silicon Graphics Inc., "SGI Reconfigurable Application Specific Computing: Accelerating Production Workflows," White paper, www.sgi.com/pdfs/3721.pdf., 2006.

[32] S.Y. Kung et al., "Wavefront Array Processors—Concept to Implementation," *Computer,* vol. 20, no. 7, pp. 18-33, July 1987.

[33] J. Lee, V. Narayanan, M.J. Irwin, and W. Wolf, "An Efficient Architecture for Motion Estimation and Compensation in the Transform Domain," *IEEE Trans. Circuits and Systems for Video Technology,* vol. 16, no. 2, pp. 191-201, Feb. 2006.

[34] D.L. Hung, H. Cheng, and S. Sengkhamyong, "Design of a Hardware Accelerator for Real-Time Moment Compensation: A Wavefront Array Approach," *IEEE Trans. Industrial Electronics,* vol. 46, no. 1, pp. 207-218, Feb. 1999.

[35] O.B. Efremides, M.P. Bekakos, and D.J. Evans, "Implementation of the Generalized WZ Factorization on a Wavefront Array Processor," *Int'l J. Computer Math.,* vol. 79, no. 7, pp. 807-815, 2002.

Azzedine Boukerche is a full professor of computer science and holds a Canada research chair position at the University of Ottawa. Prior to this, he was a faculty member at the Department of Computer Sciences and Engineering, University of North Texas. He also worked as a senior research scientist at Metron Corp. located in San Diego, California, where he was leading several Department of Defense (DOD) projects on data distribution management for large-scale distributed and interactive systems. He also worked as a visiting scientist at Caltech/JPL-NASA, where he contributed to a project centered on the specification and verification of the software used to control interplanetary spacecraft operated by JPL/NASA Laboratory. He was a visiting professor at the The University Paris-Dauphine (June 2008). He is the founding director of PARADISE Research Laboratory at uOttawa. He is the recipient of several awards, including The Ontario Distinguished Researcher Award, Canada Foundation for Innovation Researcher Award, the prestigious Premier's Ontario Research Excellence (PREA) Award, the George S. Glinski Award for Excellence in Research, the corecipient of the third National Award for Telecommunication Software 1999 for his work on a distributed security systems on mobile phone operations, and several Best Research Paper Awards including ICC 2009, IWCMC 2009, ICC 2008, PADS 1997-1999, MSWiM 2001, MobiWac 2006, and DS-RT 2008. He serves as an associate editor for the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Vehicular Technology*, *IEEE Wireless Communication Magazine*, *ACM/Springer Wireless Networks*, *Elsevier Ad Hoc Networks*, *Elsevier Int'l Journal on Pervasive and Mobile Computing*, *Wiley Wireless Communication and Mobile Computing*, *Int'l Journal of Parallel and Distributed Computing (JPDC)*, and *SCS Transactions on Simulation*. His current research interests include wireless networks and mobile computing, wireless ad hoc and sensor networks, wireless multimedia, distributed management and security system for wireless and mobile networks, and large-scale distributed interactive simulations and collaborative virtual environment. He serves as a steering committee chair, general chair, and program cochair for several IEEE/ACM Int'l Conferences, including CNRS '09, MASCOTS '02, DCOSS '06, Globecom '08 and '09, and ISCC '09. He was keynote speakers at several IEEE Int'l conferences. He also serves as the secretary of the IEEE ComSoc Technical Committee on Ad Hoc and Sensor Networks. He is also the editor of three books on mobile computing, wireless ad hoc, and sensor networks. He is a senior member of the IEEE and the IEEE Computer Society.

Jan M. Correa received the BS and MSc degrees in computer science and the PhD degree in electrical engineering from University of Brasilia (UnB), Brazil, in 1999, 2001, and 2008, respectively. He is currently an assistant professor at the (UnB). His current research interests include genetic algorithms, bioinformatics, and application-specific accelerators.

Alba Cristina M.A. de Melo received the BS degree in computer science from UnB, Brazil, in 1986, the MsC degree in computer science from UFRGS, Brazil, in 1991, and the PhD degree in computer science from the Institut National Polytechnique de Grenoble, France, in 1996. She is currently an associate professor at the University of Brasilia (UnB). In 2008, Dr. de Melo was a Post-Doctoral Fellow at the PARADISE Research Laboratory, University of Ottawa, Canada. Her current research interests include high performance computing, bioinformatics, and application-specific accelerators. She is a senior member of the IEEE.

Ricardo P. Jacobi received the MSc degree in electrical engineering from UFRGS, Brazil, in 1986 and the PhD degree in applied sciences from the Université Catholique de Louvain, Belgium, in 1993. He is currently an associate professor at the University of Brasilia (UnB). His current research interests include embedded systems design, CAD, and reconfigurable architectures.