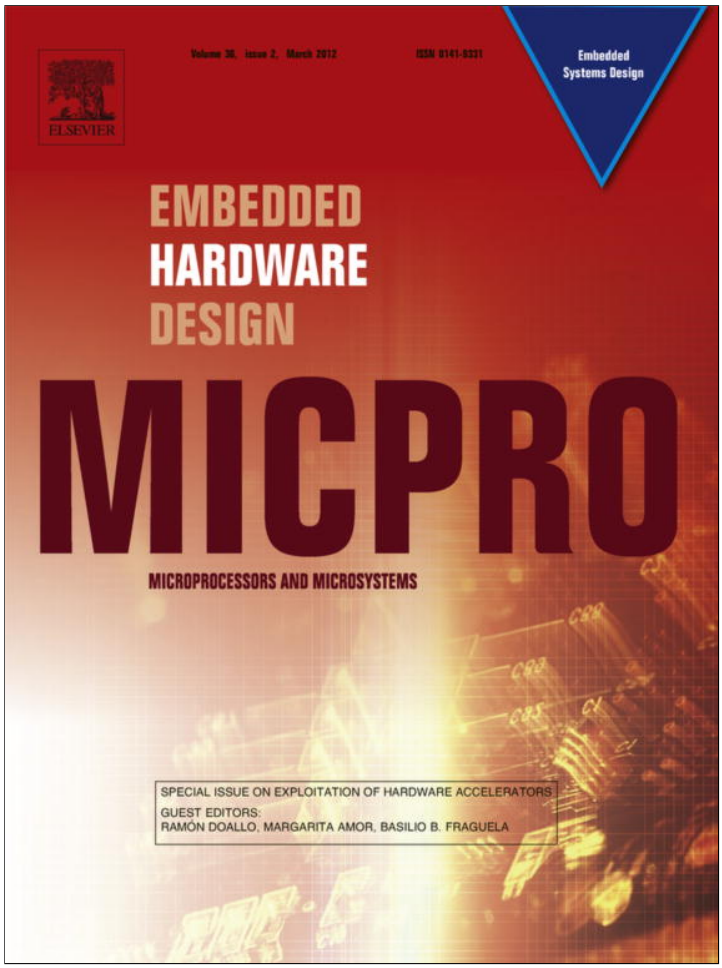

Microprocessors and Microsystems 36 (2012) 96–109

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Hardware accelerator architecture for simultaneous short-read DNA sequences alignment with enhanced traceback phase

Nuno Sebastião *, Nuno Roma, Paulo Flores

INESC-ID/IST, Rua Alves Redol, 9, Lisboa, Portugal

ARTICLE INFO

Article history:
Available online 30 May 2011

Keywords:
Hardware accelerator
DNA
Local sequence alignment
Traceback

ABSTRACT

Dynamic programming algorithms are widely used to find the optimal sequence alignment between any two DNA sequences. This manuscript presents a new, flexible and scalable hardware accelerator architecture to speedup the implementation of the frequently used Smith–Waterman algorithm. When integrated with a general purpose processor, the developed accelerator significantly reduces the computation time and memory space requirements of alignment tasks. Such efficiency mainly comes from two innovative techniques that are proposed. First, the usage of the maximum score cell coordinates, gathered during the computation of the alignment scores in the matrix-fill phase, in order to significantly reduce the time and memory requirements of the traceback phase. Second, the exploitation of an additional level of parallelism in order to simultaneously align several query sequences with the same reference sequence, targeting the processing of short-read DNA sequences. The results obtained from the implementation of a complete alignment system based on the new accelerator architecture in a Virtex-4 FPGA showed that the proposed techniques are feasible and the developed accelerator is able to provide speedups as high as 16 for the considered test sequences. Moreover, it was also shown that the proposed approach allows the processing of larger DNA sequences in memory restricted environments.

1. Introduction

The advent of the latest generations of sequencing technologies [1] has opened many new research opportunities in the fields of biology and medicine, including cell Deoxyribonucleic Acid (DNA) sequencing, gene discovery and evolutionary relationships. These technologies have contributed to the exponential growth of biological data that is available for researchers. For instance, the GenBank [2] has doubled its data size approximately every 18 months and in its December 2010 release it included over 122×10^9 base pairs (bps) from several different species.

To assist the biologists in the extraction of useful information and in the interpretation of the huge sized sequence databases, a set of alignment algorithms (e.g. the widely used Smith–Waterman (S–W) [3]) have been developed to solve many open problems in the field of bioinformatics, such as (i) *DNA re-sequencing*, where genome assembly is done against a reference genome; (ii) *Multiple Sequence Alignment (MSA)*, where multiple genomes are aligned to perform genome annotation; and (iii) *Gene finding*, where Ribonucleic Acid (RNA) sequences (the transcriptome) are aligned against the organism genome to identify new genes.

Currently, a common sequencing approach is based on the application of High Throughput Short Read (HTSR) technologies [4], to reduce the cost of the sequencing process. This technique consists of cutting the DNA fragments under analysis into shorter fragments (*reads*), which are individually sequenced and aligned against a reference sequence.

At present, the three most important HTSR sequencing platforms are: the GS FLX Genome Analyzer (454), the Solexa 1G Sequencer (Illumina) and the SOLiD Sequencer (Applied Biosystems). The biochemistry technology underlying each of these platforms leads to very different characteristics, in terms of *reads* length, throughput and raw errors. However, independently of the adopted platform, the length of the *reads* produced by these platforms is small when compared to previous generation sequencing technologies and much smaller than the original complete DNA sequence. Nevertheless, the sheer volume of data that is generated and the need to align these *reads* to large reference genomes limits a direct and naive application of standard Dynamic Programming (DP) techniques. One simple example of a common challenge comes from the need to align up to 100 million *reads* against a reference genome that can be as large as 3 Gbp. For the SOLiD sequencer, with *reads* as short as 30 bps, this corresponds to the computation of 100 million matrices of dimension $3 \times 10^9 \times 30$, which results in a computational task that is unfeasible even for a standard high performance machine.

* Corresponding author.
E-mail address: Nuno.Sebastiao@inesc-id.pt (N. Sebastião).

0141-9331/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2011.05.003

Hence, the computational demands for the analysis of the biological data produced by the various sequencing technologies has lead to the development of several accelerating strategies that aim at parallelizing the execution of the alignment algorithms. Some of these strategies are software based, while others use dedicated hardware implementations. Among the former, an optimized implementation using Single-Instruction Multiple-Data (SIMD) instructions for current CPUs [5] is commonly adopted in sequence alignment programs, like SSEARCH35. Other software implementations make use of the highly parallel execution capabilities presented by Graphics Processing Unit (GPU) to achieve a high alignment throughput [6]. With regard to the hardware implementations, these include both Application Specific Integrated Circuit (ASIC) [7–10] and Field Programmable Gate Array (FPGA) [11–15] implementations. Regardless of the considered implementation, the most common and efficient hardware architectures map the alignment algorithm to a systolic array of Processing Element (PE). Furthermore, although some bidimensional arrays have been presented [16], the most common implementations adopt unidimensional (linear) arrays [7–13,15]. In fact, the main differences among the several implementations relate to the design of the individual PE. However, some of these designs oversimplify the implemented algorithm, by only calculating the edit distance between a sequence pair [12,14], therefore not being suited to accelerate the more generic S–W algorithm. A commercial solution [17], developed by *CLC bio* and implemented in FPGA, was also made available but little information is given about its architecture.

Nevertheless, all the previously presented hardware solutions only focus on accelerating the first phase of the S–W algorithm (DP matrix fill), completely disregarding the second phase (traceback), which is typically performed using a General Purpose Processor (GPP) in a post processing step. In Ref. [18] it was proposed a hardware architecture that also accelerates the traceback phase. However, only the global alignment problem is addressed. Furthermore, the previously proposed hardware architectures are not easily optimized to deal with short *reads* sequences, obtained from current HTSR sequencing platforms (e.g. Illumina).

In another perspective, there has been a growing interest in the development of processing solutions that merge, in a single package, the reconfiguration capabilities offered by FPGAs with the advantages of a hardwired CPU. Such solutions, like the Intel Atom E645C processor [19], allow to implement highly specialized hardware accelerators tightly coupled with a general purpose CPU, in order to significantly improve the overall system performance. Furthermore, by making use of the offered reconfiguration capabilities, it is possible to implement a wide range of accelerators according to the specific task that is currently being executed. This task-multiplexing capability along the time reduces the total cost of ownership of such system due to its adaptability, low initial cost and high performance.

To overcome the limitations of previous accelerator architectures, to improve the overall sequencing performance and to make use of the advantages provided by current FPGAs, a new hardware accelerator architecture together with a new technique to speedup the sequence alignment, is now proposed. Such accelerator, targeting an embedded platform, is based on the exploitation of the following two important contributions that are extensively described in the remaining sections of the manuscript:

- An innovative and quite efficient technique that makes use of the information gathered during the computation of the alignment scores in the matrix fill phase (in hardware), in order to significantly reduce the time and memory requirements of the traceback phase (later implemented in software) [20]. To support such technique, the developed hardware accelerator architecture was tightly integrated with a GPP, to form a complete and quite efficient local alignment system implemented in an FPGA. The obtained experimental results show that the proposed accelerating structure may provide speedups as high as 16 for the implementation of the whole alignment procedure when compared to an Intel Core2 Duo processor. It is also observed that a significant reduction of the memory resources required by the subsequent traceback phase is achieved.
- An additional level of parallelism is also exploited in the proposed accelerating structure, to further increase its performance. With the presented structure, several query sequences may be simultaneously aligned with the same reference sequence, thus allowing a significant acceleration of the alignment task of the short *reads* against the reference genome, as used by HTSR techniques. This is achieved by configuring the developed accelerator in a *multiple-stream* structure, by including multiple linear arrays that work in parallel. Besides the speedup that is achieved with such improvement, which is proportional to the number of linear arrays that are implemented (defined through platform parameterization), the accelerator also takes advantages of the temporal locality in the manipulation of the larger reference genome, thus reducing the number of required memory and I/O accesses to perform the alignment.

This manuscript is organized as follows: Section 2 gives a brief overview on the widely adopted S–W algorithm to determine the optimal alignment. The proposed technique to speed up the traceback phase is presented in Section 3. Section 4 introduces the newly developed accelerator architecture that implements the proposed enhancements in the alignment procedure, including the optimizations for short *reads* sequences. A performance model of the entire alignment system is presented in Section 5. In Section 6, the prototyping platform that integrates the proposed accelerator and a GPP is presented while in Section 7 the obtained results are discussed and the achieved speedups are presented. The conclusions are drawn in Section 8.

2. Pairwise local sequence alignment

Sequence alignment is the method by which useful information is extracted from the large amounts of sequenced DNA. The alignments can be classified either as local or global. In global alignments, the complete sequences are aligned from one end to the other, whereas in local alignments only the subsequences that present the highest similarity are considered. In practice, the local alignment is generally preferred when searching for similarities between distantly related biological sequences, since this type of alignment more closely focuses on the subsequences that were conserved during evolution.

One of the most widely adopted algorithms to find the optimal local alignment between a pair of sequences is the S–W algorithm [3]. This algorithm is based on a DP method and is characterized by the smallest runtime among the *optimal* local alignment algorithms. With a runtime complexity of $O(nm)$, where n and m denote the sizes of the sequences being aligned, the S–W algorithm computes the alignment in two phases: a DP *matrix fill* phase and a *traceback* phase.

2.1. Smith–Waterman algorithm

Consider any two strings S_1 and S_2 of an alphabet Σ with sizes n and m, respectively. The local alignment of strings S_1 and S_2 reveals which pair of substrings of S_1 and S_2 optimally align, such that no other pairs of substrings have a higher alignment score. Let $G(i,j)$

Table 1
Example of a substitution score matrix.

Sbc	A	C	G	T
A	3	−1	−1	−1
C	−1	3	−1	−1
G	−1	−1	3	−1
T	−1	−1	−1	3

represent the best alignment score between a suffix of string $S_1[1..i]$ and a suffix of string $S_2[1..j]$. The S–W algorithm allows the computation of $G(n, m)$ by recursively calculating $G(i, j)$, which will reveal the highest alignment score between the substrings of strings S_1 and S_2.

The recursive relation to calculate the local alignment score $G(i, j)$ is given by Eq. (1), where $Sbc(S_1(i), S_2(j))$ denotes the substitution score value obtained by aligning character $S_1(i)$ against character $S_2(j)$ and α represents the gap penalty cost (the cost of aligning a character to a space, also known as gap insertion). An example of a substitution function is shown in Table 1.

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + Sbc(S_1(i), S_2(j)) \\ G(i-1,j) - \alpha \\ G(i,j-1) - \alpha \\ 0 \end{cases} \quad (1)$$

$$G(i,0) = G(0,j) = 0$$

The alignment scores are usually positive for characters that match, thus denoting a similarity between them. Mismatching characters may have either positive or negative scores, according to the type of alignment that is being performed, denoting the biological proximity between them. Different substitution score matrices may be used to reveal different types of alignments. In fact, the particular score values are usually defined by biologists, according to evolutionary relations. The gap penalty cost α is always a positive value.

As soon as the entire score matrix G is filled, the substrings of S_1 and S_2 with the best alignment can be found by locating the cell with the highest score in G. Then, all matrix cells that lead to this highest score cell are sequentially determined by performing a *traceback* phase. This last phase concludes when a cell with a zero score is reached, identifying the aligned substrings as well as the corresponding alignment. The path taken at each cell is chosen based on which of the three neighboring cells (left, top-left and top) was used to calculate the current cell value using the recurrence given by Eq. (1).

Table 2 shows an example of the calculated score matrix for aligning two sequences (S_1 = *CAGCCTCGCT* and S_2 = *AATGCCATTGAC*)

Table 2
Example of an alignment score matrix.

		0	1	2	3	4	5	6	7	8	9	10	11	12
	G	ø	A	A	T	G	C	C	A	T	T	G	A	C
0	ø	0	0	0	0	0	0	0	0	0	0	0	0	0
1	C	0	0	0	0	0	3	3	0	0	0	0	0	3
2	A	0	3	3	0	0	0	2	6	2	0	0	3	0
3	G	0	0	2	2	3	0	0	2	5	1	3	0	2
4	C	0	0	0	1	1	6	3	0	1	4	0	2	3
5	C	0	0	0	0	0	4	9	5	1	0	3	0	5
6	T	0	0	0	3	0	0	5	8	8	4	0	2	1
7	C	0	0	0	0	2	3	3	4	7	7	3	0	5
8	G	0	0	0	0	3	1	2	2	3	6	10	6	2
9	C	0	0	0	0	0	6	4	1	1	2	6	9	9
10	T	0	0	0	3	0	2	5	3	4	4	2	5	8

$$\begin{array}{cccccccc} G & C & C & A & T & T & G \\ | & | & | & & | & & | \\ G & C & C & \_ & T & C & G \end{array}$$

Fig. 1. Obtained local alignment for the considered example sequences.

using the substitution score matrix presented in Table 1 (a match has a score of 3 and a mismatch a score of −1). The gap penalty has a value of 4. The shadowed cells represent the traceback path (starting at the highest score cell (8, 10)) that was taken in order to determine the best alignment. The resulting alignment is illustrated in Fig. 1.

3. Tracking the alignment origin and end indexes

As previously referred, whenever a sequence pair alignment is required, it is necessary to implement the *traceback* phase of the S–W algorithm. Most sequence alignment hardware accelerators that have been proposed until now [11,15,16] only implement the score matrix computation (without performing the traceback phase). Therefore, they simply output the calculated alignment score (the highest value of matrix G). Afterwards, whenever the obtained score is greater than a given user-defined threshold, the whole G matrix must be recalculated (usually by software, using a GPP). However, contrasting to what happened in the hardware accelerator, in this recalculation all the intermediate data that is required to perform the traceback and retrieve the corresponding alignment must be maintained in the GPP memory. Moreover, this re-computation does not re-use any data from the previous calculation performed by the hardware accelerator. Such situation can be even aggravated by the fact that typical alignments consider sequences with a quite dissimilar size, with $m \gg n$ (e.g. HTSR sequencing analysis). Therefore, the size of the subsequences that participate in the alignment is always in the order of n, meaning that a large part of matrix G that must be completely recomputed in the GPP is not even required to obtain the alignment.

To overcome this inefficiency, an innovative technique is now proposed to significantly reduce the time and memory space that is required to find the local alignment in the *traceback* phase of this algorithm. In fact, assuming that it is possible to know that the local alignment of a given sequence pair S_1 and S_2 starts at position $S_1(p)$ and $S_2(q)$, denoted as (p, q), and ends at position $S_1(u)$ and $S_1(v)$, denoted as (u, v), then the local alignment can be obtained in the *traceback* phase by just considering the score matrix corresponding to substrings $S_a = S_1[p..u]$ and $S_b = S_2[q..v]$.

To determine the character position where the alignment starts, an auxiliary matrix C_b is proposed. Let $C_b(i, j)$ represent the coordinates of the score matrix cell where the alignment of strings $S_1[1..i]$ and $S_2[1..j]$ starts. Using the same DP method that is used to calculate matrix $G(i, j)$, it is possible to simultaneously build matrix C_b, with the same size as G, that tracks the cell that originated the score that reached cell $G(i, j)$ (i.e. the start of the alignment ending at cell (i, j)). The recursive relations to compute matrix C_b are given by Eq. (2), with initial conditions of $C_b(i, 0) = C_b(0, j) = (0, 0)$.

$$C_b(i,j) = \begin{cases} (i,j), & \text{if } G(i,j) = G(i-1,j-1) + Sbc(S_1(i), S_2(j)) \\ & \text{and } C_b(i-1,j-1) = (0,0) \\ C_b(i-1,j-1), & \text{if } G(i,j) = G(i-1,j-1) + Sbc(S_1(i), S_2(j)) \\ & \text{and } C_b(i-1,j-1) \neq (0,0) \\ C_b(i-1,j), & \text{if } G(i,j) = G(i-1,j) - \alpha \\ C_b(i,j-1), & \text{if } G(i,j) = G(i,j-1) - \alpha \\ (0,0), & \text{if } G(i,j) = 0 \end{cases} \quad (2)$$

Table 3
Example of an AOEI tracking matrix.

		0	1	2	3	4	5	6	7	8	9	10	11	12
	C_b	ø	A	A	T	G	C	C	A	T	T	G	A	C
0	ø	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)
1	C	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(1, 5)	(1, 6)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(1, 12)
2	A	(0, 0)	(2, 1)	(2, 2)	(0, 0)	(0, 0)	(0, 0)	(1, 5)	(1, 6)	(1, 6)	(0, 0)	(0, 0)	(2, 11)	(0, 0)
3	G	(0, 0)	(0, 0)	(2, 1)	(2, 2)	(3, 4)	(0, 0)	(0, 0)	(1, 6)	(1, 6)	(1, 6)	(3, 10)	(0, 0)	(2, 11)
4	C	(0, 0)	(0, 0)	(0, 0)	(2, 1)	(2, 2)	(3, 4)	(4, 6)	(0, 0)	(1, 6)	(1, 6)	(0, 0)	(3, 10)	(4, 12)
5	C	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(2, 2)	(3, 4)	(3, 4)	(3, 4)	(0, 0)	(1, 6)	(0, 0)	(3, 10)
6	T	(0, 0)	(0, 0)	(0, 0)	(6, 3)	(0, 0)	(0, 0)	(3, 4)	(3, 4)	(3, 4)	(3, 4)	(0, 0)	(1, 6)	(3, 10)
7	C	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(6, 3)	(7, 5)	(7, 6)	(3, 4)	(3, 4)	(3, 4)	(3, 4)	(0, 0)	(1, 6)
8	G	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(8, 4)	(6, 3)	(7, 5)	(7, 6)	(3, 4)	(3, 4)	(3, 4)	(3, 4)	(3, 4)
9	C	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(8, 4)	(6, 3)	(7, 5)	(7, 6)	(3, 4)	(3, 4)	(3, 4)	(3, 4)
10	T	(0, 0)	(0, 0)	(0, 0)	(10, 3)	(0, 0)	(8, 4)	(8, 4)	(6, 3)	(7, 5)	(7, 6)	(3, 4)	(3, 4)	(3, 4)

Table 4
Reduced alignment score matrix.

G'	ø	G	C	C	A	T	T	G
ø	0	0	0	0	0	0	0	0
G	0	3	0	0	0	0	0	3
C	0	0	6	3	0	0	0	0
C	0	0	3	9	5	1	0	0
T	0	0	0	5	8	8	4	0
C	0	0	3	3	4	7	7	3
G	0	3	0	2	2	3	6	10

Hence, by applying the proposed technique, denoted as Alignment Origin and End Indexes (AOEI) tracking, and by knowing the cell where the maximum score ($G(u, v)$) occurred, it is possible to determine from $C_b(u, v) = (p, q)$ the coordinates of the cell where the alignment began. Consequently, to obtain the desired alignment, the *traceback* phase only has to rebuild the score matrix for the subsequences $S_1[p..u]$ and $S_2[q..v]$, which are usually considerably smaller than the entire S_1 and S_2 sequences.

The obtained matrix C_b for the alignment example of sequences S_1 and S_2, whose G matrix was presented in Table 2, is shown in Table 3. In this example, by knowing from G matrix that the maximum score occurs at cell (8, 10), it is possible to retrieve the coordinates of the beginning of the alignment in cell $C_b(8, 10) = (3, 4)$. With this information, the optimal local alignment between S_1 and S_2 can be found by processing only the substrings $S_a = S_1[3..8] = GCCTCG$ and $S_b = S_2[4..10] = GCCATTG$. Such alignment (between S_a and S_b) can now be determined by computing a much smaller G' matrix in the *traceback* phase, as shown in Table 4.

The major advantage of this technique is the significant reduction of the time and memory space required to recompute matrix G for the subsequences that actually participate in the alignment, when compared to the entire sequences. Therefore, it provides a great reduction of the computational effort (time and space) of the whole alignment algorithm.

4. Alignment core architecture

The local alignment algorithm described in Section 2 is usually applied to process biological sequences with pronounced dissimilar sizes m and n, where $m \gg n$ (e.g. $m \approx 10^6$ and $n \approx 10^2$). The matrix fill phase of the alignment algorithm is the most computationally intensive part being, therefore, a good candidate for parallelization. However, the data dependencies that exist in the calculation of each matrix cell highly restrict the parallelization model. In fact, only the computation of the values along the matrix anti-diagonal direction can be performed in parallel (to calculate the value for cell $G(i, j)$ it is necessary to know the values of $G(i-1, j-1)$, $G(i, j-1)$ and $G(i-1, j)$).

Specialized parallel hardware that is capable of performing a great number of simultaneous arithmetic operations is especially suited for this task. In particular, linear systolic arrays with several identical Processing Elements (PEs), as shown in Fig. 2, have proved to be efficient structures to implement this type of computation, by simultaneously computing the values of matrix G that are located in a given anti-diagonal [15].

4.1. Base processor element architecture

The PE's architecture proposed in this paper is based on the PE structure described in Ref. [15] and illustrated in Fig. 3. This *base* PE only implements the basic score matrix calculation and it is composed by a two stage pipelined datapath that calculates each matrix cell value (output in $G(i, j)$). The throughput of each element is one score value per clock cycle. Since the S–W algorithm requires the evaluation of the maximum score value among the set of scores that compose the entire matrix, it is necessary to include an additional datapath that selects the maximum value that was

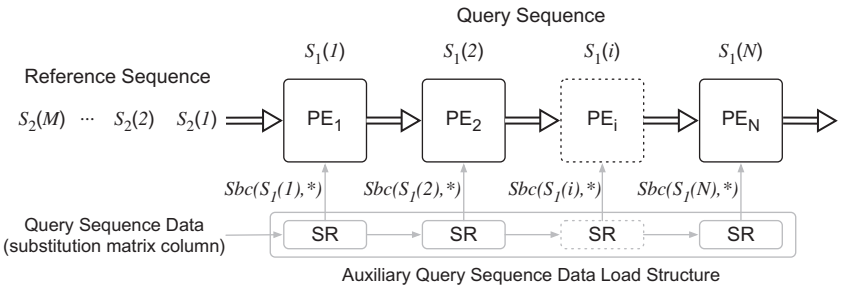

Fig. 2. Systolic array structure for DNA algorithms.

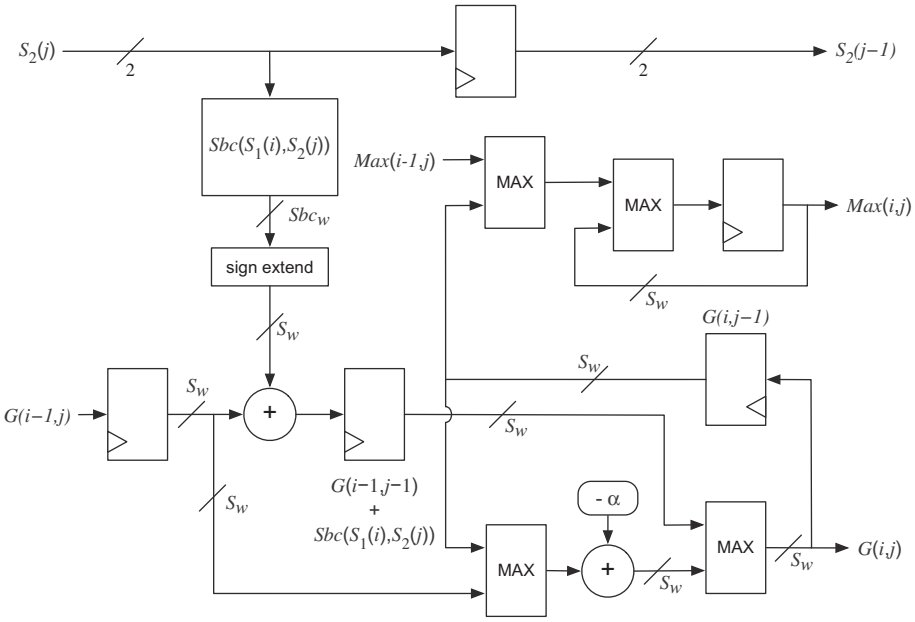

Fig. 3. *Base* architecture of processor element PE_i.

calculated in the whole PE array (output $Max(i,j)$). The width of the buses, denoted as S_w and Sbc_w, are constrained by the considered implementation conditions of the accelerator. In particular, the width of the score bus, S_w, is directly constrained by the maximum size of the query sequence (the shortest among the two sequences) and the score matrix values. Such substitution score values also determines the size of the corresponding bus (Sbc_w). With such datapath, PE_i outputs the maximum score that was computed by PEs 1 through i.

The array evolves along the time, by shifting the reference sequence characters through the PEs. The query sequence character $S_1(i)$ is allocated to the ith PE and this PE performs, at every clock cycle, the computations required to determine the score value of a certain matrix cell. After all the reference sequence characters $S_2(j)$ have passed through all the PEs, the alignment score is available at output $Max(i,j)$ of the last PE.

The computation that is performed in each PE requires, among other operations, the selection of the substitution score corresponding to the two characters, i.e. the value of $Sbc(S_1(i), S_2(j))$. Since each PE always operates with the same character of S_1, it only needs to store the column of the substitution score matrix (Sbc) that represents the costs of aligning character $S_1(i)$ with the entire alphabet.

In the computation of each matrix cell value $G(i,j)$, the evaluation of the maximum value among the results of the three distinct possibilities presented in Eq. (1) is also required. In particular, the zero condition of the S–W algorithm is implemented by controlling the reset input of the registers that store the $G(i,j)$ value. Such reset makes use of the sign bit of the score value, i.e. if the maximum value among the three partial scores is negative, then the registers that hold that score are cleared.

4.2. Enhanced processor element architecture

The PE architecture that is now proposed implements the AOEI accelerator technique that was described in Section 3. With this technique, the re-computation of the entire G matrix when performing the traceback phase is avoided. It is implemented by propagating, through the PEs, not only the partial maximum scores (as in the *base* PE), but also the coordinates of their origin (the beginning of the alignment), together with the coordinates where the maximum score occurred. As it was shown in Section 3, this greatly simplifies the traceback phase by only focusing on the substrings that are actually involved in the alignment and avoiding the re-computation of the whole matrix G.

The architecture of the *enhanced* PEs is presented in Fig. 4. Each PE features a datapath that implements Eqs. (1) and (2). The additional hardware that is required to implement Eq. (2) (the AOEI technique) is mainly composed of multiplexers and registers. The signals that control these additional multiplexers are generated by the magnitude comparators integrated in the MAX units and that were already present in the *base* PE architecture. The widths of the coordinates' buses, Cq_w and Cr_w, are constrained by the maximum query sequence size and the maximum reference sequence size, respectively. The width of the C_w bus is the sum of Cq_w and Cr_w.

The coordinates of the matrix cell under processing are obtained by using the hardwired PE index (i) and the symbol coordinate (j) that comes alongside with the sequence character present at input $S_2(j)$. Regarding the input data signals, the origin coordinates that correspond to the score at input $G(i-1,j)$ are present at input $C_b(i-1,j)$. Likewise, the origin coordinates corresponding to the score at output $G(i,j)$ are present at output $C_b(i,j)$. Finally, the coordinates of the currently highest score (present at $Max(i,j)$) are output at $MaxC_b(i,j)$.

4.3. Short-read optimizations

When the query sequences under processing are acquired by short-read sequencing platforms, the sample sequences can be extremely short and in some cases they may even have less symbols than the number of available PEs in the array. For instance, the *reads* generated by the Illumina platform can be as short as 35 nucleotides long. In such a case, several of the PEs do not perform any useful calculations, due to the fact that no query sequence symbol is attributed to them. This situation would certainly result in a substantial decrease of the throughput of the array. Therefore, considering that in most practical setups there is a very significant number of short-read sequences that must be aligned with the same reference sequence, alternative arrangements of

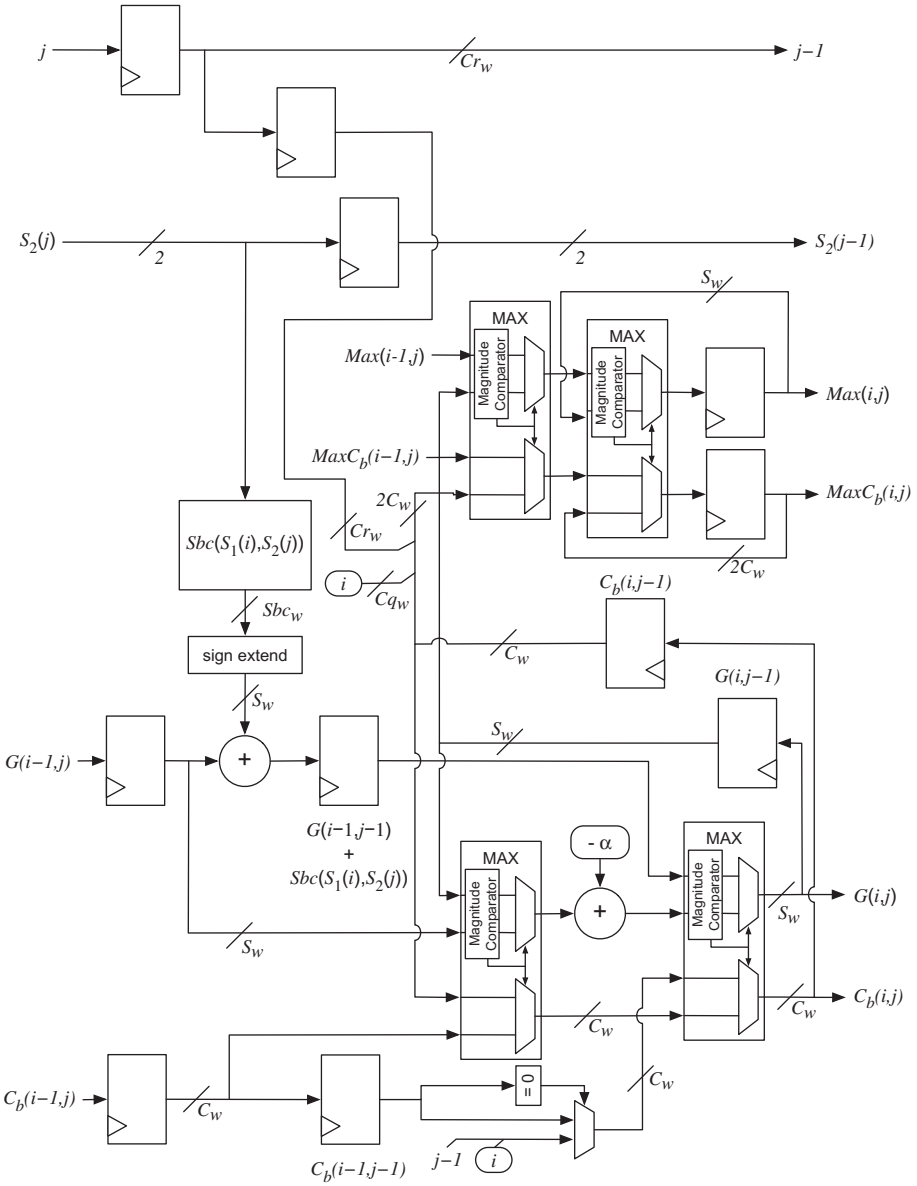

Fig. 4. *Enhanced* architecture of processor element PE_i.

the available PE resources may be considered in order to make it possible to simultaneously perform the alignment of more than one short query sequence to the same reference sequence.

This optimization can be achieved with the proposed architecture by configuring the hardware accelerator in a *multiple-stream* processing scheme. In such configuration, the accelerator includes several coupled linear arrays of PEs that work in parallel and align to the same reference sequence. Hence, while the reference sequence is simultaneously shifted to the multiple arrays, the set of independent query sequences to be processed is distributed and assigned among the PEs of the multiple-stream array, as shown in Fig. 5. The exact number of parallel PE arrays is

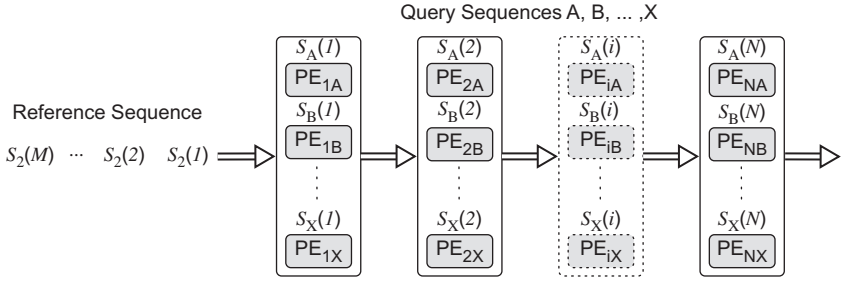

Fig. 5. Example configuration of a multiple-stream PE array (several independent streams).

configurable according to the size of the short-read sequences to be aligned and to the amount of available hardware resources.

It is worth noting that the implemented multiple-stream array also allows an improvement of the resource usage of the accelerator, since it is possible to share a set of resources that are common among the multiple parallel PEs that are processing the same reference sequence. This is accomplished by using a common set of registers that hold the reference symbol ($S_2(j)$) and the respective coordinate (j) for the several elements of the array that work in parallel, as shown in Fig. 6 for a dual-stream configuration.

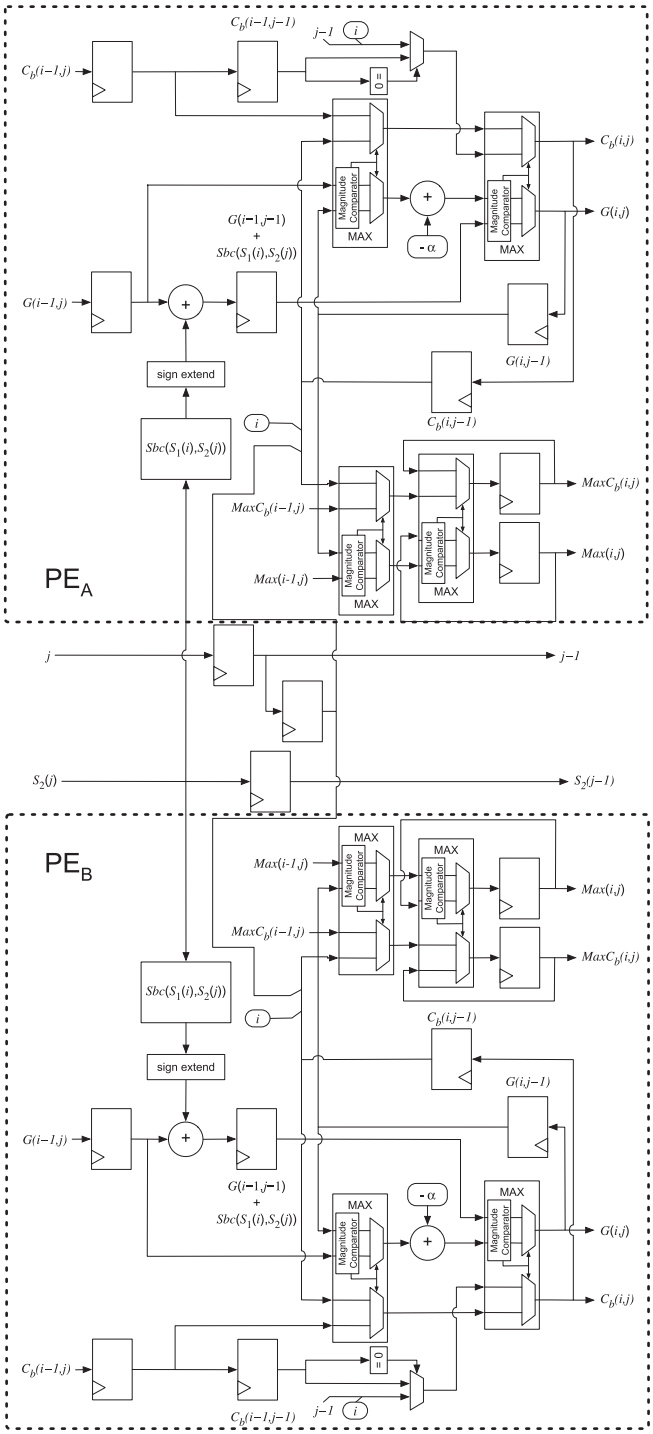

Fig. 6. Example of a multiple-stream PE in dual-stream configuration.

This optimization can significantly increase the actual throughput of the array since an increased number of PEs is performing useful computations and the alignment of more than one query sequence may be simultaneously performed, therefore leading to a greater speedup than would be achieved with just a single array. Furthermore, the use of such configuration also leads to a reduction of the amount of data that is transferred to the accelerator, since the reference sequence is simultaneously aligned with more than one query sequence. This is especially significant when a large number of short-read query sequences are aligned to a large reference genome sequence.

4.4. Array programming

Since each PE compares the reference sequence symbols with a single query sequence character, it will just access the values present at the corresponding column of the substitution matrix. Therefore, each PE will only receive the substitution score matrix column that corresponds to the query sequence character allocated to that PE.

Such data is stored in dedicated registers within each PE, since this allows for a fast reprogramming of a new query sequence. In the event of a PE is not being used (because the query sequence has a smaller size than the number of available PEs (N)), the substitution score data that is stored in such PE corresponds to a matrix column in which every value is zero.

To program the score values corresponding to query sequence S_1, an auxiliary data load structure, composed by a n bit-width shift register, was included in the array. This structure allows the preloading of the *next* query sequence data into this temporary storage shift register, by serially shifting the substitution matrix column, while the array is still processing the data corresponding to the *current* query sequence. As soon as the array has finished the processing of the *current* query sequence, the *next* query sequence data (already stored in the auxiliary shift register) is parallel loaded (in just one clock cycle) into the respective PEs. In case the proposed accelerator architecture is configured as a multiple-stream structure, each individual PE array has the corresponding auxiliary data load structure for the query sequence, which allows the simultaneous load of the query information to the several PEs. This allows to mask the time that would be required to shift the *next* query sequence data into the array and therefore significantly reduces its programming time. Furthermore, the use of this shift register also provides a scalable method to program the processor array, as it avoids the use of a common data bus to program the several PEs.

4.5. Interface

To integrate the proposed hardware accelerator with the GPP that will implement the remaining alignment procedure (i.e. the

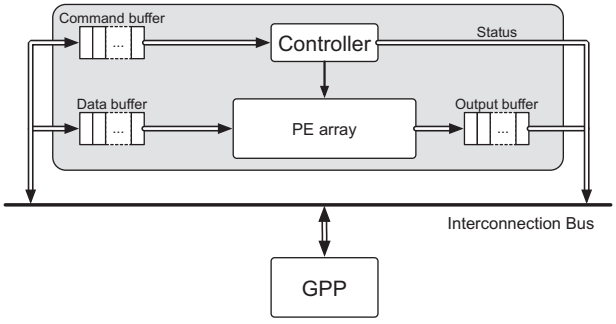

Fig. 7. Accelerator interface and interconnection with the GPP in the prototyping platform.

traceback), the systolic array includes an embedded controller that is responsible for decoding seven instructions (required to properly control the array), as well as to receive the data to be processed. The developed interface, illustrated in Fig. 7, is composed of two input First-In First-Out (FIFO) queue (one for the reference sequence and the other for commands and the query sequence), one output FIFO queue (to return the processed values) and one status register. The two input FIFOs allow the next query sequence to be loaded into the array while the current alignment is being processed, without increasing the complexity of the control that would arise from having all of the data (query and reference sequences data) input through the same FIFO. In the case of a multiple-stream configuration, the several query sequences are input using the previously mentioned input FIFO and are then sent to a specific PE array, based on the information defined by the main program running on the GPP. Afterwards, as soon as the alignment scores and corresponding AOEI coordinates are calculated, they are serially stored in the output FIFO for later processing in the GPP.

Each FIFO has a depth of 64 words and is 32-bits wide, to match the bus-width adopted by most current GPPs. The status register contains some information about the available positions in each of the input FIFOs, allowing the implementation of a flow control mechanism. Furthermore, this status register also contains some information regarding to the availability of output data in the output FIFO, indicating when the accelerator has concluded the alignment.

The developed interface allows this accelerator to be interconnected to several types of interconnection buses, requiring only the design of the appropriate logic to decode the specific bus control signals. The input and output FIFOs can be mapped to the GPP memory address space and therefore be easily accessible using common load/store instructions. This type of interface can be used either in PCI, PCIe, AMBA–APB or other types of interconnections, therefore allowing this accelerator to be used in a wide range of platforms.

5. Performance model

The performance of a complete alignment system composed of several different modules depends on the performance of each individual module and how they interact. Among these are the CPU performance, the interconnection (bus) throughput and the accelerator performance (if present). To better understand and evaluate the advantages provided by the proposed alignment structure, this section presents a thorough modelization of the resulting global performance.

Typically, the set of operations that are required to perform an alignment in a system without an accelerator are: (i) database read, (ii) data transfer to the processing device and (iii) computation, which includes the *matrix fill* and the *traceback* phases. Assuming that these operations are completely sequential, the total alignment time (T^s) can be modeled as the sum of the database read time (T_{db}), the data transfer time $\left(T_i^{ds}\right)$ and the CPU processing time for the matrix fill phase $\left(T_c^{M_s}\right)$ and the traceback phase $\left(T_c^{T}\right)$:

$$T^s = T_{db} + T_i^{ds} + T_c^{M_s} + T_c^T \tag{3}$$

The time corresponding to each individual component is given by:

$$\begin{aligned} T_{db} &= \tfrac{n+m}{f_d B_d^w C \eta_d}, && (0 < \eta_d \leqslant 1) \\ T_i^{ds} &= \tfrac{n+m}{f_i B_i^w C \eta_i}, && (0 < \eta_i \leqslant 1) \\ T_c^s &= T_c^{M_s} + T_c^T = \tfrac{\gamma(n\times m)}{f_c \eta_c} + \tfrac{\gamma k}{f_c \eta_c}, && (0 < \eta_c \leqslant 1, \quad \gamma \geqslant 1) \end{aligned} \tag{4}$$

where n and m denote the query and reference sequence sizes, k represents the number of cells traversed during the traceback phase, f_d, f_i and f_c denote the database read, interconnection bus and CPU processing frequencies, respectively. C represents the compression factor (how many nucleotides are encoded in an 8-bit word), while B_d^w and B_i^w denote the width (in bytes) of the database read device and of the interconnection (bus), respectively. The η_d, η_i and η_c parameters denote efficiency factors, which take into account eventual contention on accessing the database, the interconnection and the CPU, as well as inherent wait states and protocol dependent control operations. Finally, γ represents the average number of CPU clock cycles required to process a single cell of the DP matrix.

In sequential single-core CPUs (without any accelerator), $T_c^s \gg T_{db} + T_i^{ds}$, due to the $\mathcal{O}(nm)$ runtime of the matrix fill phase, which leads to the commonly observed total alignment time of $T^s \approx T_c^s$. In contrast, when the proposed accelerator is present, the processing is split among the accelerator and the CPU. By considering (as an example) the architecture of the proposed accelerator in a single-stream configuration, the time it takes to compute the whole DP score matrix, in the accelerator (T_a) is given by:

$$T_a = \frac{N+m-1}{f_a}, \qquad (n \leqslant N) \tag{5}$$

where N represents the number of PEs in the array and f_a denotes their operating frequency. In this parallel processing scheme, the accelerator computes the whole score matrix (G), of size $n \times m$, while the CPU performs a much simpler matrix fill (time $T_c^{M_r}$) and traceback over the smaller matrix G', totaling a computation time of $T_c^r = T_c^{M_r} + T_c^T$. Typically, an alignment only includes part of the considered sequences. Hence, the number of traversed cells during the traceback (k) can be used to major the size of the subsequences that are used to compute the smaller matrix G', which will thus have a maximum size of $k \times k$.

By using the proposed accelerator, it is possible to parallelize some operations. In this case, the accelerator performs the DP *matrix fill* phase of the *current* sequence pair alignment, while the CPU implements the *traceback* of the *previous* sequence pair. Therefore, both the accelerator and the CPU work in a pipelined way. Furthermore, it is also possible to read the *next* query sequence (as well as the reference sequence, if necessary) from the database in parallel with the processing of both the accelerator and the CPU. This type of processing involves three distinct data transfers, with the respective duration: (i) from the database to the system's main memory $\left(T_i^{ds}\right)$, (ii) from the system's main memory to the accelerator (T_i^{sa}), and (iii) from the accelerator to the system's main memory (T_i^{as}). The time to transfer the score and coordinates output from the accelerator to the CPU (T_i^{as}), which consists of no more than five 32-bit values, is quite small and thus can be neglected when compared to the other parcels $(T_i^{as} \ll T_i^{sa})$. The data transfers between the several components can occur in parallel with the remaining processing (time $T_i^r \approx T_i^{ds} + T_i^{sa}$). Therefore, the total alignment time can be modeled as:

$$T^p \approx \max\left\{T_a;\ T_i^{ds} + T_i^{sa};\ T_c^{M_r} + T_c^T;\ T_{db}\right\} \tag{6}$$

Assuming that a data parallel 32-bit wide bus is used to interconnect the accelerator, then $B_i^w = 4$. The same width is also typically used in the database device interface, making $B_d^w = 4$. Furthermore, the used 2-bit encoding per nucleotide leads to $C = 4$. Therefore, the alignment time for these particular conditions becomes:

$$T^p \approx \max\left\{\frac{N+m-1}{f_a}; \frac{n+m}{4\cdot 4 f_i \eta_i} + \frac{n+m}{4\cdot 4 f_i \eta_i}; \frac{\gamma(k\times k)}{f_c \eta_c} + \frac{\gamma k}{f_c \eta_c}; \frac{n+m}{4\cdot 4 f_d \eta_d}\right\}$$

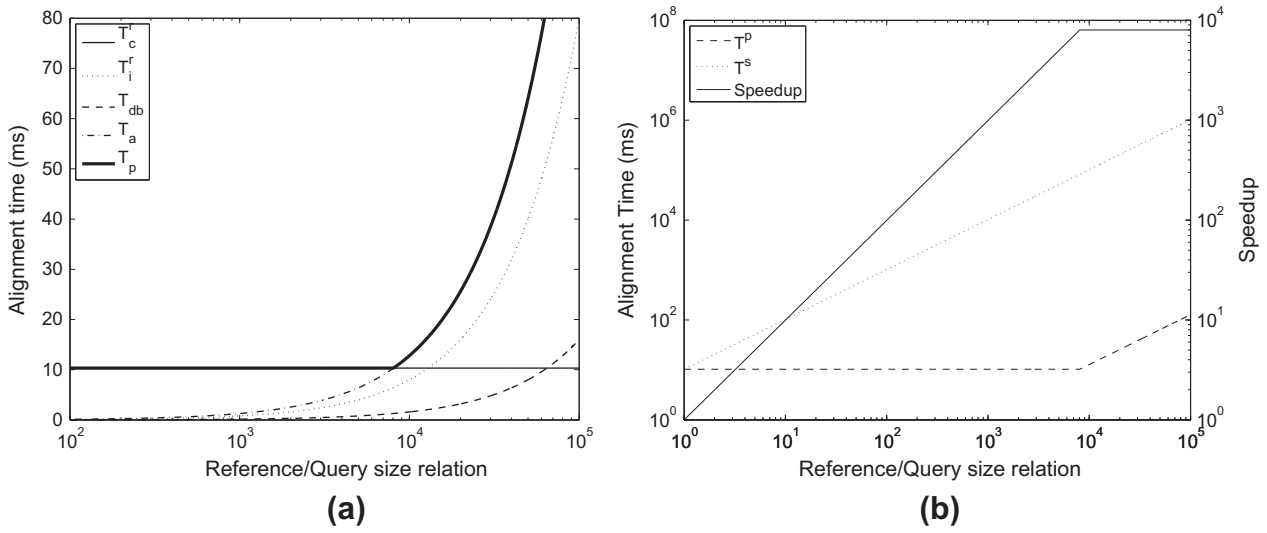

Fig. 8. Variation of the alignment time (T^p) and speedup (T^s/T^p) according to the model described in Eq. (7).

$$T^p \approx \max\left\{\frac{N+m-1}{f_a};\ \frac{n+m}{16\,f_i\,\eta_i}+\frac{n+m}{16\,f_i\,\eta_i};\ \frac{\gamma\,(k^2+k)}{f_c\,\eta_c};\ \frac{n+m}{16\,f_d\,\eta_d}\right\} \tag{7}$$

Hence, in an alignment scenario where the same reference sequence is aligned to a large number of query sequences (Q) and assuming that the reference sequence can be permanently stored in the system's main memory while aligning all the respective query sequences, the database reading time and the corresponding data transfer time to the system's memory are reduced, leading to an average alignment time per query sequence ($\overline{T^1}$):

$$\overline{T^1} \approx \max\left\{\frac{N+m-1}{f_a};\ \frac{n+m/Q}{16\,f_i\,\eta_i}+\frac{n+m}{16\,f_i\,\eta_i};\ \frac{\gamma\,(k^2+k)}{f_c\,\eta_c};\ \frac{n+m/Q}{16\,f_d\,\eta_d}\right\} \tag{8}$$

Moreover, considering that the accelerator may be able to perform b simultaneous alignments by using the multiple-stream feature, the average alignment time for each sequence pair ($\overline{T^b}$) is given by

$$\overline{T^b} \approx \max\left\{\frac{N+m-1}{f_a\,b};\ \frac{n+m/Q}{16\,f_i\,\eta_i}+\frac{n+m/b}{16\,f_i\,\eta_i};\ \frac{\gamma\,(k^2+k)}{f_c\,\eta_c};\ \frac{n+m/Q}{16\,f_d\,\eta_d}\right\} \tag{9}$$

As an example, Fig. 8 depicts the total alignment time according to the model described by Eq. (7), in which the reference sequence is read from the database for each query sequence (worst case). The considered model parameters are: $n = k = 128$, $\gamma = 50$, $N = 128$, $f_a = f_i = f_c = f_d = 100$ MHz, $\eta_i = 0.2$, $\eta_c = 0.8$ and $\eta_d = 0.5$.

One interesting observation that can be extracted from the presented model is concerned to the accelerator role in the resulting performance of the whole alignment system. In fact, as the relation between the reference and the query sequence size increases, the accelerator becomes the most limiting performance factor, as it has the highest workload. Therefore, the increase of the CPU performance, above a given minimum value, does not significantly influence the performance of the alignment system leading to a quasi-stationary speedup value. In the presented example, the threshold value is about 8000, which corresponds to the relation between the reference and query sequences sizes frequently adopted in bioinformatic applications.

6. Prototyping platform

To validate the functionality and to assess the performance of the proposed hardware accelerator in a practical realization, a complete local alignment system based on the S–W algorithm was developed and implemented. The basic configuration of this system, used as a proof-of-concept, consists of a Leon3 GPP processor [21] that executes all operations of the S–W algorithm, except those concerning the score matrix computation phase. Such phase is executed by the proposed hardware accelerator, acting as a specialized functional unit of the GPP.

The software implementation of the S–W algorithm includes some optimizations in order to achieve more efficient applications in embedded systems. In particular, all memory accesses were optimized by using a static memory allocation mechanism. Special attention was also devoted to the data transfers of both the reference and query sequences from the GPP to the proposed hardware accelerator, so that a high level of efficiency is achieved.

6.1. Leon3 processor

The Leon3 processor [21] is one of the most used processor cores that are freely available. It was specifically designed for embedded applications by the European Space Agency, although nowadays it is maintained by Gaisler Research. It consists of a highly configurable and fully synthesizable core, described in VHDL, implementing a RISC architecture conforming to the SPARC v8 definition. Such freely available VHDL description allows this GPP to be implemented in several different platforms (e.g. ASIC), unlike other proprietary GPPs (e.g. Xilinx's MicroBlaze). Furthermore, the availability of reliable software development tools (e.g. compiler and debugger) for the Leon3 processor make it an adequate choice for the proof-of-concept system.

The Leon3 32-bit core is based on a Harvard micro-architecture with a 7-stage instruction pipeline and 32-bit internal registers. The core functionality can be easily extended by means of the AMBA–2.0 AHB/APB on-chip buses. The AMBA–2.0 AHB is used to connect the Leon3 processor with high-speed controllers, such as the cache and memory controllers. On the other hand, the AMBA–2.0 APB is used to access most on-chip peripherals and is connected to the Leon3 processor via the AHB/APB Bridge. External memory access and memory mapped I/O operation are

provided by a programmable memory controller with interfaces to PROM, SRAM and SDRAM chips.

6.2. DNA alignment peripheral

A new peripheral, consisting of the proposed hardware accelerator for DNA alignment, was developed and embedded in the Leon3 processor (see Fig. 7). This alignment peripheral was connected to the AMBA–2.0 APB as a slave device. This bus was selected not only because it has enough bandwidth for all of the sequence data transfers, but also because it offers a simple interface and low-power consumption.

Some additional wrapper logic, responsible for the adaptation of the accelerator to the AMBA–2.0 APB bus, was also included, consisting mostly of multiplexers, decoders and a simple control unit that implements the bus protocol. The I/O FIFOs and the status register of the alignment core are mapped in the Leon3 memory address space. Hence, by using such interface, the write and read operations over this peripheral can be easily implemented using simple load and store operations.

6.3. FPGA implementation

The implementation of the proposed local alignment system was realized in an FPGA device by using a GR-CPCI-XC4V development board from Pender Electronic Design. Such development system includes a Virtex4 XC4VLX100 FPGA device from Xilinx, a 133 MHz 256 MB SRAM memory bank, and several peripherals for control, communication and storage purposes.

The adopted Leon3 processor is based on version gpl-1.0.20-b3403 of GRLIB. This soft-processor core was configured to incorporate a hardware divide and multiply unit, an interrupt controller, separate data and instruction cache controllers and an SRAM memory controller, all interconnected with the AMBA–2.0 AHB interface. Moreover, such core also encompasses two 32-bit timers and the proposed DNA Alignment peripheral, which were all connected to the system AMBA–2.0 APB.

7. Experimental results

The previously presented accelerator architecture, described using parameterizable VHDL code, was synthesized using Xilinx ISE 10.1 (SP3) software tools and implemented in the previously described FPGA. This reconfigurable embedded system, used fundamentally as a proof-of-concept prototyping platform, is composed by the Leon3 GPP and the alignment accelerator core with an array composed by a maximum of 128 PEs. Although the maximum operating frequency of the accelerator core is 120 MHz, the actual operating frequency of the entire system is 60 MHz, as a consequence of a limitation imposed by the considered Leon3 processor implementation. However, as it was explained in Section 5, for the usual ranges of the relation between the reference and query sequences sizes this GPP limitation does not significantly constraint the overall performance of the system.

7.1. Single-stream configuration

The obtained resource allocation results of the entire alignment system, when considering the single-stream array configurations, are presented in Table 5. The resources solely occupied by the Leon3 processor are also presented as a reference. These results show that the Leon3 processor alone occupies 18% of the available logic resources of the used FPGA device. In what concerns the resource allocation for the systolic array using the *enhanced* PEs, it is possible to observe that it is 77% larger in relation to the corresponding *base* configuration, without the AOEI tracking functionality. However, the exact increase of the amount of used hardware depends on the considered operating environment, namely, the size of the sequences to be aligned (which determines the bit-width of the coordinate representation) and the adopted scoring scheme (which influences the bit-width of the score calculations).

To validate and assess the performance of the proposed system, a set of real DNA sequences was used for the reference sequence. These sequences were obtained from the GenBank database [2] and their size ranges from about 17×10^3 to 2.6×10^6 nucleotides.

Table 5
FPGA resource allocation of a single-stream array.

PE		Score width	Maximum size		Resource usage	
Type	#		Reference	Query	Registers	LUTs
Leon3	0	–	–	–	6246 (6%)	17,788 (18%)
Base	16	7	–	16	7441 (8%)	19,818 (20%)
Base	64	9	–	64	11,736 (12%)	28,148 (29%)
Base	128	10	–	128	16,031 (16%)	34,130 (35%)
Enh.	16	7	2^{16}	16	9499 (10%)	22,168 (23%)
Enh.	64	9	2^{22}	64	22,625 (23%)	36,114 (37%)
Enh.	128	10	2^{22}	128	40,024 (41%)	56,541 (58%)

Table 6
Processing time results for the alignment system when using a single-stream array with 128 PEs and a query sequence of 128 nucleotides.

Reference size	Processing time using only the Leon3 processor (ms)			Processing time using the Leon3 processor and the proposed accelerator (ms)				Speedup
	Matrix fill $T_c^{M_s}$	Traceback T_c^T	Total T^s	Score and coordinates (HW) $\max\{T_a;\ T_i^{sa}\}$	Reduced matrix fill (Leon3) $T_c^{M_r}$	Reduced traceback (Leon3) T_c^T	Cycle period T^p	
17,878	7493.6	0.9	7494.5	0.7	45.7	0.9	46.6	161
83,648	35049.2	0.9	35050.1	3.2	54.4	1.0	55.4	633
136,980	57390.4	1.0	57391.4	5.2	54.4	1.0	55.4	1036
295,301	123757.5	0.9	123758.4	11.1	49.5	1.0	50.5	2451
566,490	237377.8	0.9	237378.8	21.3	49.5	0.9	50.5	4701
745,211	312359.1	1.0	312360.0	28.0	50.7	1.0	51.7	6042
1,311,701	–	–	–	49.3	50.9	1.0	51.9	–
2,623,402	–	–	–	98.5	50.8	1.0	98.5	–

In what concerns the query sequences, their maximum size is limited by the number of available PEs in the array. Consequently, for the implemented configuration, it must not be greater than 128 nucleotides long (a size entirely compatible with the latest Next-Generation Sequencing technologies [1]). In this particular instantiation, the size of the chosen query sequences is 128 nucleotides. For larger query sequences, the number of PEs in the array has to be increased and, if necessary, the array can be expanded by connecting another FPGA device.

The advantages provided by the proposed AOEI technique, as well as the performance of the developed hardware accelerator, were assessed using the previously selected sequences, which were aligned using two different methods: (i) pure software implementation, where the alignment between each sequence pair is obtained using a pure and straight-forward implementation of the S–W algorithm running exclusively on the GPP (keeping the entire score matrix in memory) and (ii) hardware accelerated implementation, where the alignment is obtained by using the developed accelerator (with the *enhanced* PEs) and the GPP. The obtained execution time results for both of these methods are presented in Table 6. While the total processing time for the pure software implementation (T^s) is the sum of the partial times, the total time of the hardware accelerated implementation (T^p) considers the fact that the accelerator and the GPP work concurrently in a pipelined scheme: the accelerator determines the score and the alignment coordinates of a given sequence pair while the GPP is performing the matrix recomputation and traceback for the previous pair of processed sequences. Therefore, in the concurrent configuration, the presented total time (T^p) is the maximum value between the hardware accelerator ($\max\{T_a;\ T_i^{sa}\}$) and GPP execution times ($T_c^{M_r} + T_c^T$) (see Eqs. (6) and (7)). It should be noted that the presented results for the accelerator processing time already consider the communication between the GPP and the accelerator ($\max\{T_a;\ T_i^{sa}\}$) and that the database reading and corresponding data transfer times are not considered, since the queries and reference sequence were pre-loaded to the system's main memory. The obtained speedup was determined by comparing the time required to obtain each whole alignment using the *pure software* sequential implementation of the S–W algorithm and the time required to obtain the same alignment with the aid of the proposed AOEI technique and the corresponding hardware accelerator.

According to the obtained results, the attained speedups may be as high as 6042. These speedups are in accordance to the trends predicted in Section 5 (see Fig. 8) and are the consequence of a twofold contribution: on the one hand, the parallelization of the whole *matrix fill* phase by the systolic array; on the other hand, the reduction of the processing time required to perform the *traceback* in the GPP, due to the significant reduction of the size of the score matrix that must be recomputed in this phase. At this respect it is worth noting that the time complexity of the G matrix computation during the matrix fill phase implemented in the GPP is $O(nm)$, whereas in the accelerator this complexity is reduced to $O(m)$, due to the parallel processing in the n PEs. These two factors justify the significant speedup value that is attained in determining the local alignment score as it was described in Section 5.

In what concerns the traceback phase, the time complexity is the same in both cases ($O(n+m)$). In fact, in order to perform the traceback in the GPP it is necessary to recompute the whole G matrix. Nevertheless, this recomputation time is significantly reduced when the proposed AOEI technique is adopted. As an example, and considering the alignment of the 128 nucleotide query sequence with the 1,311,701 reference sequence, the obtained local alignment spans over only a 124 nucleotide long subsequence of the reference sequence and over a 123 nucleotide subsequence of the query sequence. If the *entire* G matrix had to be recomputed to obtain the alignment, it would have approximately 168×10^6 cells, which significantly contrasts with the situation provided by the proposed AOEI technique, where the size of the G' matrix that needs to be recomputed in the GPP is reduced to only $124 \times 123 \approx 16 \times 10^3$ cells. This significant reduction (of about four orders of magnitude) is particularly important when the implementation of the alignment procedure is considered in embedded systems, with strict memory and power consumption restrictions. Hence, not only does the proposed technique allow to significantly reduce the time required to obtain the alignment, but it also makes it possible to process larger sequences as it significantly reduces the amount of memory used by the GPP (e.g. the 2,623,402 nucleotide long reference sequence, whose memory requirements prevent it from being aligned using the pure software approach on the GPP).

Finally, it is also important to note that the obtained throughput results of the proposed systolic PE array are in line with the results corresponding to similar architectures presented in the past [11,15,16]. However, such past architectures were only focused on accelerating the matrix-fill phase of the S–W algorithm. In contrast, besides accelerating the matrix-fill phase, the presented accelerator architecture also implements the new AOEI method and therefore returns additional information that is subsequently used to further reduce the computational requirements. Such feature is not included in any other proposals, therefore being a differentiating characteristic of this work and hindering a direct and fair comparison.

7.2. Multiple-stream configuration

To evaluate the developed multiple-stream capability, several different configurations of the alignment system were implemented. The corresponding resource usage results are presented in Table 7. The maximum number of implemented multiple-streams was 3, since the resources of the considered FPGA device do not allow for additional streams. However, any number of processing streams is supported if there are enough resources available to implement them. All of the considered configurations have a maximum reference sequence size of 2^{22}.

As expected, the results in Table 7 show a slight reduction in the amount of used resources when compared to an identically sized single-stream array (i.e. when the number of PEs of the single-stream array is equal to the number of PEs of the multiple-stream array multiplied by the number of streams). This reduction is due to the shared resources among the multiple-stream PEs, as well as the reduction in the bit-width required to represent the AOEI coordinates, since the query sequence being aligned is smaller. Therefore, in terms of used resources, a triple-stream configuration is more advantageous to align several short-read sequences when compared to three completely independent arrays.

To evaluate the performance of the alignment task, three streams of short-read query sequences, each with 35 nucleotides and obtained with the Illumina sequencing platform, were aligned

Table 7
FPGA resource usage of the multiple-stream arrays.

PE			Score width	Resource usage	
Type	#	n-Stream		Registers	LUTs
Leon3	0	–	–	6246 (6%)	17,788 (18%)
Enh.	128	1	10	40,024 (41%)	56,541 (58%)
Enh.	64	1	9	22,625 (23%)	36,114 (37%)
Enh.	64	2	9	38,349 (39%)	54,183 (55%)
Enh.	35	1	9	15,427 (16%)	28,071 (29%)
Enh.	35	2	9	24,149 (25%)	38,169 (39%)
Enh.	35	3	9	32,687 (33%)	48,205 (49%)
Enh.	37	3	13	34,299 (35%)	50,143 (51%)

Table 8
Processing time results using multiple-stream array configurations, to align three query sequence streams, each with 35 nucleotides.

# PE	*n*-Stream	Reference size	Processing time using the Leon3 processor and the proposed accelerator (ms)			PE occupancy rate (%)
			Score and coordinates (HW)	Reduced matrix fill (Leon3)	Reduced traceback (Leon3)	
128	1	2,623,402	304.4	11.4	0.8	27
64	1	2,623,402	302.9	11.4	0.8	55
64	2		204.9			
35	1	2,623,402	301.5	11.4	0.8	100
35	2		204.9			
35	3		103.4			

Table 9
Performance comparison with an Intel Core2 Duo CPU.

	Query size				
	128		35		
Device	Intel CPU	Accelerator (128 × 1)	Intel CPU	Accelerator (35 × 1)	Accelerator (35 × 3)
Time (ms)	882.5	98.5	1712.3	301.5	103.4
Speedup	1	9.0	1	5.7	16.6
Equivalent MCUPS	381	3409	161	914	2664

to the same 2,623,402 nucleotides long reference sequence. Six different accelerator configurations, all with the proposed AOEI functionality, were used to obtain the alignments: (i) the single-stream configuration with 128 PEs that was used in the previous section, (ii) a single-stream, and a (iii) dual-stream configurations with 64 PEs each, (iv) a single-stream, (v) a dual-stream, and (vi) a triple-stream configurations with 35 PEs each. The 35 PE arrays are adequately fitted to the size of the short-reads being aligned using this sequencing technology.

The achieved processing time results for aligning the three query streams using the previously described accelerator configurations are presented in Table 8. As it is possible to observe, the alignment task is considerably faster when the accelerator is configured as a multiple-stream array with the number of PEs in each array identical to the query sequence size, since this leads to a configuration where all the PEs are performing useful calculations, leading to a PE occupancy ratio of 100%. If the single-stream array with 128 PEs is used to align the three streams of 35 nucleotides long query sequences, the PE occupancy ratio of the array is significantly decreased (down to 27%). This means that a significant part of the PEs would be performing null operations, since only 35 of them would have a query sequence symbol assigned, therefore decreasing the actual throughput of the array. Consequently, the required time to obtain the score and the index coordinates for the three streams of query sequences is roughly three times the time required to obtain the same information for a single stream (see Table 6). However, using a triple-stream array where the number of PEs is adequately fitted to the query sequence size (35 nucleotides), it is possible to simultaneously align three different query sequences using the same hardware resources, as presented in Table 8. Therefore, not only is the overall efficiency of the system significantly increased, as an additional speedup is also achieved, proportional to the number of implemented linear arrays.

7.3. Comparison and discussion

To complete the presented architecture evaluation, the performance of the proposed accelerator was also compared to the performance of a pure-software implementation running on a common CPU. The SSEARCH35 software program from the FASTA framework was used for this purpose, since it is one of the most used programs to determine the local alignment. This program implements the state-of-the-art SIMD optimizations proposed in Ref. [22] and was executed on a 2.4 GHz Intel Core2 Duo processor. The execution times were obtained by aligning the same query and reference sequences adopted in the previous evaluations.

The obtained execution times, presented in Table 9, show that the speedup attained with the conceived accelerator when compared with a pure software implementation running on the Core2 Duo may be as high as 16. In particular, the lower processing time obtained for the short sequences is due to the better usage of the available hardware resources provided by the accelerator, which enabled a triple-stream configuration using the same FPGA device. Table 9 also includes the equivalent million cell updates per second (MCUPS) metric, which is commonly used to compare the performance of alignment algorithms across different platforms. However, this metric only takes into account the throughput of the matrix fill phase of the S–W algorithm, without considering the traceback phase requirements. Nevertheless, the performance obtained using the developed system still achieves a significant speedup compared to the SSEARCH35 program. The decrease in performance of the software based solution for the smaller query sequences reveals its inability to maintain the performance levels with such short sequences. Furthermore, it is important to recall that the overall performance of the accelerator is proportional to the total number of PEs, thus the apparent smaller equivalent performance of the 35 triple-stream PEs ($35 \times 3 = 105$) array when compared to the 128 single-stream PEs array.

Regarding the database read rate, the implemented accelerator requires one reference sequence nucleotide in each clock cycle. As previously mentioned, four nucleotides are encoded in each byte, thus the accelerator requires an input transfer rate of, at least, 15 MB/s. In the worst case scenario, in which the reference sequence is not stored in the main memory and needs to be read from the database at each alignment, the database reading rate (which also needs to account for the much smaller query sequence reads) has to be higher than 15 MB/s in order to sustain the operation of the accelerator at its maximum performance. Current mainstream storage devices (hard disk drives) have a sustained throughput above 100 MB/s. Even when considering the accelerator running at 120 MHz, the database reading rate would double to 30 MB/s, still well below the throughput of the storage devices.

To further demonstrate the implementation alternatives offered by the proposed accelerator, the processing core was also synthesized for the FPGA device available in the Intel Atom E645C processor, an Altera Arria II GX device [19]. The synthesis was performed using the Quartus II v10.1 software from Altera. The obtained results demonstrated that the processor is capable of operating at a clock frequency of 120 MHz. Synthesis results also revealed that the available hardware resources of this device allow to implement accelerators with 128 PEs in dual-stream configuration and with 35 PEs in a 6-stream configuration. According to the model derived in Section 5, these configurations significantly improve the overall performance of the accelerator allowing for the concurrent alignment of two 128 nucleotides long query sequences ($N = 128$, $b = 2$) or six 35 nucleotides long query sequences ($N = 35$, $b = 6$), respectively. In this case, only the accelerator is implemented in the FPGA, while the Intel Atom processor performs the role of the GPP.

Finally, a last observation concerning the system cost is deserved. In fact, the acquisition cost of a system based on a hybrid platform, like the Intel Atom E645C, is similar to the cost of current off-the-shelf computing systems, like those based on the Intel Core2 Duo processors. However, if the higher throughput provided by the accelerator implemented in the FPGA is taken into account, the alignment system based on this new platform will achieve a much smaller cost per alignment than current implementations. Moreover, the memory size reduction provided by the proposed accelerator also allows a further reduction of the total system cost.

8. Conclusions

A highly efficient hardware accelerator architecture that significantly speedups the implementation of DNA local alignment algorithms is presented. Such accelerator is based on the exploitation of an innovative and quite efficient technique to significantly reduce the computational time and memory requirements of the traceback phase that is executed as part of the widely used Smith–Waterman algorithm. Furthermore, the developed structure also exploits an additional level of parallelism, in order to simultaneously align several query sequences with the same reference sequence, by adopting a multi-stream processing flow. Such feature is particularly useful in the processing of short-read DNA sequences obtained from current HTSR sequencing technologies.

The developed accelerator was integrated with a Leon3 general purpose processor, in order to prototype a complete embedded alignment system for DNA processing. The conceived platform was implemented in a Virtex-4 FPGA. The obtained results demonstrate that the developed accelerator provides speedups as high as 6042, when compared with a pure software version of the Smith–Waterman algorithm, running on the Leon3 processor. Speedups up to 16 were also achieved when compared to an highly optimized SIMD software implementation running on an Intel Core 2 Duo Processor.

The obtained results also reveal that the proposed multiple-stream configurations favor the exploitation of the available FPGA resources and aid in maintaining the array running at maximum performance in different alignment scenarios. Moreover, it was shown that the use of the proposed accelerator enables the alignment of larger DNA sequences, even in a memory restricted environments.

Acknowledgments

The presented research was performed in the scope of project "*HELIX: Heterogeneous Multi-Core Architecture for Biological Sequence Analysis*", funded by the Portuguese Foundation for Science and Technology (FCT) with reference PTDC/EEA-ELC/113999/2009, and partially supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and through the Ph.D. grant with reference SFRH/BD/43497/2008.

References

[1] J. Shendure, H. Ji, Next-generation DNA sequencing, Nat. Biotechnol. 26 (2008) 1135–1145.
[2] D.A. Benson, I. Karsch-Mizrachi, D.J. Lipman, J. Ostell, E.W. Sayers, GenBank, Nucleic Acids Res. 38 (2010) D46–D51.
[3] T.F. Smith, M.S. Waterman, Identification of common molecular subsequences, J. Mol. Biol. 147 (1981) 195–197.
[4] M.J. Chaisson, P.A. Pevzner, Short read fragment assembly of bacterial genomes, Genome Res. 18 (2008) 324–330.
[5] M.S. Farrar, Striped Smith-Waterman speeds database searches six times over other SIMD implementations, Bioinformatics 23 (2007) 156–161.
[6] L. Ligowski, W. Rudnicki, An efficient implementation of Smith–Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases, in: IEEE Int. Symp. Parallel & Distributed Processing, IPDPS 2009, IEEE, 2009, pp. 1–8.
[7] E.T. Chow, J.C. Peterson, M.S. Waterman, T. Hunkapiller, B.A. Zimmermann, A systolic array processor for biological information signal processing, in: Proc. of the 5th International Conf. on Supercomputing, ICS '91, ACM, New York, NY, USA, 1991, pp. 216–223.
[8] P. Guerdoux-Jamet, D. Lavenier, SAMBA: hardware accelerator for biological sequence comparison, Bioinformatics 13 (1997) 609–615.
[9] T. Han, S. Parameswaran, Swasad: an asic design for high speed DNA sequence matching, in: Proc. of the 2002 Asia and South Pacific Design Automation Conf., ASP-DAC '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 541–546.
[10] C. White, R. Singh, P. Reintjes, J. Lampe, B. Erickson, W. Dettloff, V. Chi, S. Altschul, BioSCAN: a VLSI-based system for biosequence analysis, in: Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors, 1991. ICCD '91, pp. 504 –509.
[11] K. Benkrid, Y. Liu, A. Benkrid, A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 17 (2009) 561–570.
[12] G. Caffarena, C. Pedreira, C. Carreras, S. Bojanic, O. Nieto-Taladriz, FPGA acceleration for DNA sequence alignment, J. Circuits Syst. Comput. 16 (2007) 245–266.
[13] M. Gokhale, B. Holmes, A. Kopser, D. Kunze, D. Lopresti, S. Lucas, R. Minnich, P. Olsen, Splash: a reconfigurable linear logic array, in: Int. Conf. on Parallel Processing, 1990, pp. 526–532.
[14] S.A. Guccione, E. Keller, Gene matching using JBits, in: Proc. 12th Int. Conf. Field-Programmable Logic and Applications. FPL '02, Springer-Verlag, London, UK, 2002, pp. 1168–1171.
[15] T. Oliver, B. Schmidt, D. Maskell, Hyper customized processors for bio-sequence database scanning on FPGAs, in: Proc. 13th Int. Symp. Field-Programmable Gate Arrays, FPGA'05, ACM, 2005, pp. 229–237.
[16] L. Hasan, Z. Al-Ars, Z. Nawaz, K. Bertels, Hardware implementation of the Smith–Waterman algorithm using recursive variable expansion, in: 3rd Int. Design and Test Workshop, IDT 2008, IEEE, 2008, pp. 135–140.
[17] CLC Bio, White paper on CLC Bioinformatics Cube 1.03, Technical Report, CLC Bio, Finlandsgade 10-12-8200 Aarhus N – Denmark, 2007.
[18] S. Lloyd, Q. Snell, Sequence alignment with traceback on reconfigurable hardware, in: Int. Conf. Reconfigurable Computing and FPGAs – ReConFig '08, IEEE, 2008, pp. 259–264.
[19] Intel® Atom™Processor E6x5C Series – Product Preview Datasheet, Intel Corporation, 2010.
[20] N. Sebastião, T. Dias, N. Roma, P. Flores, Integrated accelerator architecture for DNA sequences alignment with enhanced traceback phase, in: International Conference on High Performance Computing and Simulation. HPCS, 2010, pp. 16–23.
[21] Aeroflex Gaisler, SPARC V8 32-bit Processor LEON3/ LEON3-FT CompanionCore Data Sheet, Version 1.0.3, 2008.
[22] M. Farrar, Striped Smith–Waterman speeds database searches six times over other SIMD implementations, Bioinformatics 23 (2007) 156–161.

Nuno Sebastião was born in Lisbon, Portugal in 1978. Since 2007, he holds a M.Sc. degree in Electrical and Computer Engineering from Instituto Superior Técnico (IST), Technical University of Lisbon, Lisbon, Portugal.
In 2007 he joined the Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID) as a researcher of the Signal Processing Group (SiPS) where he is currently working towards his PhD degree, also in Electrical and Computer Engineering.
His main research interests are focused on Dedicated Multi-Core Computer Architectures and High-Performance Systems for Biological Sequence Alignment (DNA, RNA and proteins). He is a member of the IEEE Circuits and Systems Society.

Nuno Roma was born in Entroncamento – Portugal in 1975. He received the Ph.D. degree in electrical and computer engineering from Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, Lisbon, Portugal, in 2008.

He is currently an Assistant Professor with the Department of Computer Science and Engineering at IST, and a Senior Researcher of the Signal Processing Systems Group (SiPS) of Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID).

His research interests include specialised computer architectures for digital signal processing (including biological sequences processing and image and video coding/transcoding), embedded systems design and compressed-domain video processing algorithms. He has contributed to more than 40 papers to journals and international conferences.

He is a member of the IEEE Circuits and Systems Society and a member of ACM.

Paulo Flores received the five-year engineering degree, M.Sc. and Ph.D. degrees in electrical and computer engineering from the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, in 1989, 1993, and 2001, respectively. Since 1990, he has been teaching at Instituto Superior Técnico, Technical University of Lisbon, where he is currently an Assistant Professor in the Department of Electrical and Computer Engineering.

He has also been with the Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID), Lisbon, since 1988, where he is currently a Senior Researcher in the Algorithms for Optimization and Simulation Group (ALGOS).

His research interests are computer architecture and CAD for VLSI circuits in the area of embedded systems, test and verification of digital systems, and computer algorithms, with particular emphasis on optimization of hardware/software problems using satisfiability (SAT) models.

He is a member of the IEEE Circuit and Systems Society.