CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2013; **25**:1319–1339
Published online 12 October 2012 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.2934

SPECIAL ISSUE PAPER

Configurable and scalable class of high performance hardware accelerators for simultaneous DNA sequence alignment

Nuno Sebastião*,†, Nuno Roma and Paulo Flores

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

SUMMARY

A new class of efficient and flexible hardware accelerators for DNA local sequence alignment based on the widely used Smith–Waterman algorithm is proposed in this paper. This new class of accelerating structures exploits an innovative technique that tracks the origin coordinates of the best alignment to allow a significant reduction of the size of the dynamic programming matrix that needs to be recomputed during the subsequent traceback phase, providing a considerable reduction of the resulting time and memory requirements. The significant performance of the enhanced class of accelerators is attained by also providing support for an additional level of parallelism: the capability to concurrently align *several* query sequences with *one or more* reference sequences, according to the specific application requisites. Moreover, the accelerator class also includes specially designed processing elements that improve the resource usage when implemented in a Field Programmable Gate Array (FPGA), and easily provide several different configurations in an Application Specific Integrated Circuit (ASIC) implementation. Obtained results demonstrated that speedups as high as 278 can be obtained in ASIC accelerating structures. A FPGA-based prototyping platform, operating at a 40 times lower clock frequency and incorporating a complete alignment embedded system, still provides significant speedups as high as 27, compared with a pure software implementation. Copyright © 2012 John Wiley & Sons, Ltd.

Received 12 January 2011; Revised 12 June 2012; Accepted 6 September 2012

KEY WORDS: hardware accelerator; DNA; local sequence alignment; traceback; FPGA; ASIC

1. INTRODUCTION

The advent of the High Throughput Short Read (HTSR) sequencing technologies [1] has lead to an exponential increase of the amount of sequenced DNA. For instance, the GenBank [2], one of the main public access databases, has presented a continuous and steady growth, and in the August 2011 release, it included over 130×10^9 base pairs (bps) from several different species.

One important characteristic of these HTSR technologies is concerned with the significant amounts of short DNA segments (reads) that are generated. However, the length of the *reads* produced by most these platforms (35−150 bp) is small when compared with previous generation sequencing technologies and much smaller than the original complete DNA sequence.

To assist the biologists in the interpretation of the huge sized sequence databases and in the extraction of useful information, a set of alignment algorithms is usually applied to solve some open problems in the field of bioinformatics, such as: *DNA re-sequencing*, where genome assembly is carried out against a reference genome; *Multiple Sequence Alignment*, where multiple genomes are aligned to perform genome annotation; *Gene finding*, where RNA sequences (the transcriptome) are aligned against the organism genome to identify new genes, and so forth. Nevertheless, the

*Correspondence to: Nuno Sebastião, INESC-ID, Rua Alves Redol, 9, Lisboa, Portugal.

†E-mail: nuno.sebastiao@inesc-id.pt

alignment algorithms that are adopted by these applications often have to process sequences of quite dissimilar length. Moreover, the number of sequences to be aligned may be significantly different: in a *short-read versus reference genome* scenario, there can be 10^6 short query sequences (each one with as few as 35 nucleotides) to align with one single reference sequence (composed by about 10^8 nucleotides), whereas in a *gene versus gene database* scenario, both the query and reference sequences have similar sizes (in the order of 10^3 nucleotides), but the database may incorporate as much as 10^5 reference sequences. Besides these, other application scenarios can also be considered, in which the sizes and the number of sequences to align may be even different from those of the previous two examples.

One of the most used alignment techniques is based on the Smith–Waterman (S–W) algorithm [3]. It is a Dynamic Programming (DP) algorithm that determines the *optimal* alignment between any pair of sequences and has a time complexity of $\mathcal{O}(nm)$, where n and m represent the size of the sequences being aligned. However, the sheer volume of data that needs to be aligned poses a practical limitation to the execution of this algorithm in typical off-the-shelf machines. 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. With *reads* as short as 35 bps, this corresponds to the computation of 100 million matrices of dimension $3 \times 10^9 \times 35$, which results in a computational task that is unfeasible even for a standard high performance machine.

To overcome the limitations of the S–W algorithm, other alignment tools, such as Basic local alignment search tool (BLAST) [4] and FASTA [5], have been developed. Although these algorithms present a much smaller runtime, they are based on heuristics that not always guarantee the *optimal* alignment. Therefore, whenever the best quality alignment is required, alignment procedures based on the S–W algorithm need to be adopted.

To significantly reduce the alignment time using the S–W algorithm, it has been frequently considered the usage of efficient accelerators, tightly coupled with General Purpose Processors (GPPs). These solutions range from parallel implementations running in Graphics Processing Units (GPUs) [6] to dedicated hardware accelerators. The most common dedicated hardware architectures are based on systolic arrays, such as the bidimensional structure presented in [7]. Nevertheless, unidimensional (linear) systolic arrays are the most frequently adopted structures [8–10]. Meanwhile, a commercial solution [11], developed by *CLC bio*, was also made available. Independently of the adopted structure, most of the previously mentioned dedicated accelerators are targeted and implemented in Field Programmable Gate Array (FPGA) platforms and use the inherent reprogrammability capabilities to broaden their application scenarios. Furthermore, architectures targeted at Application Specific Integrated Circuit (ASIC) implementation have also have been presented [12–14] and are also based on systolic arrays. However, these static accelerating structures tend to be tuned for specific applications in order to satisfy particular computational requirements. For this reason, when used in a wider range of application scenarios, the actual performance of these statically configured structures can be significantly compromised.

On the other hand, it is observed that almost all these solutions were only focused on accelerating the first phase of the S–W algorithm (the DP *matrix fill*), completely disregarding the second phase (the *traceback*), which is typically performed using a GPP in a post-processing step. The exception is observed in [15] and [10], where hardware architectures that also accelerate the traceback phase were recently presented. Nevertheless, only the global alignment problem is addressed in [15] and in [10] the memory space requirements when aligning long DNA sequences are extremely large. Moreover, most of the previously proposed hardware architectures are not perfectly adapted and optimized to deal with sequences of short *reads* ($35-150$ bp). Therefore, a flexible and configurable accelerating structure may be regarded as a better solution to attain a good performance compromise in a much wider range of application domains.

In this scope, this paper proposes a new configurable and fully parameterizable class of hardware accelerators that are capable of improving even further the alignment performance levels that are required by current DNA processing applications. Such performance improvement is the result of two important contributions: (i) 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) [16], and (ii) the exploitation of an additional level of parallelism based on the offered capability to simultaneously align *several* query sequences with *one or more* reference sequences, according to the target application requisites.

Furthermore, this new class of hardware accelerators can be implemented in an FPGA or in an ASIC device and exploits the characteristics of both implementation technologies to make it more suitable for a broader range of application scenarios. When implemented in an FPGA, the reprogrammability capabilities of such devices are exploited to fine tune the accelerator characteristics (such as the data bit-width and number of processor elements) to the target alignment conditions. On the other hand, when implemented in an ASIC, the proposed class of accelerators offers the possibility to use several different configurations of the included long systolic array structure by splitting it into several smaller arrays by means of specially designed switching elements, thus allowing the use of the same hardware to concurrently align more than one pair of sequences and thus improve the overall performance. This additional level of parallelism, achieved by configuring the accelerator in a *multiple-stream* way, provides a significant acceleration of the alignment of short *reads* against the reference genome, as used by HTSR techniques.

This manuscript is organized as follows: Section 2 gives a brief overview on the widely adopted S–W algorithm. The proposed technique to speedup the traceback phase is presented in Section 3. Section 4 describes the basic accelerator architecture, whereas Section 5 introduces the new class of accelerators that implement the concurrent processing scheme of the alignment procedure. The used prototyping platform is presented in Section 6, whereas in Section 7 the obtained results are discussed and the achieved speedups are presented. Finally, the conclusions are drawn in Section 8.

2. SMITH–WATERMAN ALGORITHM

Sequence alignment is a fundamental operation 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 procedure is generally preferred when searching for similarities between distantly related biological sequences, because this type of alignment more closely focuses on the subsequences that were conserved during evolution.

The local alignment of any two strings S_1 and S_2, with sizes n and m respectively, reveals which pair of substrings of S_1 and S_2 optimally align, such that no other substrings pairs have a higher alignment score. The S–W algorithm [3] allows the computation of the $n \times m$ DP matrix G, where each element $G(i,j)$ represents an alignment score. This matrix reveals 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 Equation (1), where $Sbc(S_1(i), S_2(j))$ denotes the substitution score value obtained by comparing the 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 I.

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

Table I. 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

Table II. 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

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

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

The alignment scores are usually positive for characters that match, thus denoting a degree of similarity between them. Mismatching characters may have either positive or negative scores, according to the type of alignment that is being performed and denote the biological proximity between them. Hence, different substitution score matrices may be used to reveal different types of alignments. In fact, the particular score values that are adopted are usually determined by biologists, according to considered 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 on the basis of which of the three neighboring cells (left, top-left, and top) was used to calculate the current score value using the recurrence given by Equation (1).

Table II shows an example of the calculated score matrix for aligning two sequences ($S_1 = CAGCCTCGCT$ and $S_2 = AATGCCATTGAC$) using the substitution score matrix presented in Table I (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 followed to determine the best alignment. The resulting alignment is illustrated in Figure 1.

3. TRACKING THE ALIGNMENT ORIGIN AND END INDEXES

Most of the previously proposed hardware accelerators only implement the score matrix computation (without performing the traceback phase), thus only returning the 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. Hence, with this common approach, the 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 actual 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. 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 by only 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 matrix cell where the alignment of strings $S_1[1..i]$ and $S_2[1..j]$ starts. By 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 maintains a track of 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 Equation (2), with initial conditions of $C_b(i, 0) = C_b(0, j) = (0, 0)$.

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

Hence, by applying the proposed technique, denoted as Alignment Origin and End Indexes (AOEI) tracking, and by simply 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 II, is shown in Table III. In this example, by knowing from matrix G 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 only processing the much smaller 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 IV.

The major advantage of this technique is a significant reduction on the *time* and *memory space* that is required to recompute matrix G, which is now confined to the subsequences that actually participate in the alignment rather than the entire sequences. With this approach, the proposed AOEI technique potentially provides an important contribution on the reduction of the computational effort (time and space) of the whole alignment algorithm.

Table III. Example of an Alignment Origin and End Indexes 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 IV. 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

4. ACCELERATOR 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 $n \approx 10^2$). On the other hand, the matrix fill phase of the alignment algorithm is the most computationally intensive part, making it 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 computations corresponding to the values along the matrix anti-diagonal direction can be performed in parallel, because the values of $G(i-1, j-1)$, $G(i, j-1)$, and $G(i-1, j)$ are needed to calculate the value for cell $G(i, j)$.

As a consequence, specialized parallel hardware that is capable of performing a great number of simultaneous arithmetic operations has been regarded as especially suited for this task [7–10]. Linear systolic arrays with several identical Processing Elements (PEs), as shown in Figure 2, have proved to be particularly efficient structures to implement this type of processing, by simultaneously computing the values of matrix G that are located in a given anti-diagonal [8].

4.1. Base processing element

The PE architecture proposed in this paper is based on the PE structure described in [8] and illustrated in Figure 3. This *base* PE only implements the basic score matrix calculation ($G(i, j)$), and it is composed by a two stage pipelined datapath. The throughput of each element is one score value per clock cycle. Then, because the S–W algorithm requires the evaluation of the maximum score value among the set of scores that compose the entire matrix, an additional datapath is also included to select the maximum value that was calculated in the whole PE array (output $Max(i, j)$). With such datapath, PE_i selects and stores 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, all 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 the $Max(i, j)$ output of the last PE.

The computation that is performed at each PE requires, among other operations, the selection of the substitution score corresponding to the two characters under analysis, that is, the value of $Sbc(S_1(i), S_2(j))$. Because 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 $G(i, j)$, the evaluation of the maximum of the three distinct terms presented in Equation (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 $G(i, j)$. Such reset is infered from the sign bit of the score value, that is, if the maximum value among the three partial scores is negative, then the register that holds such score is automatically cleared.

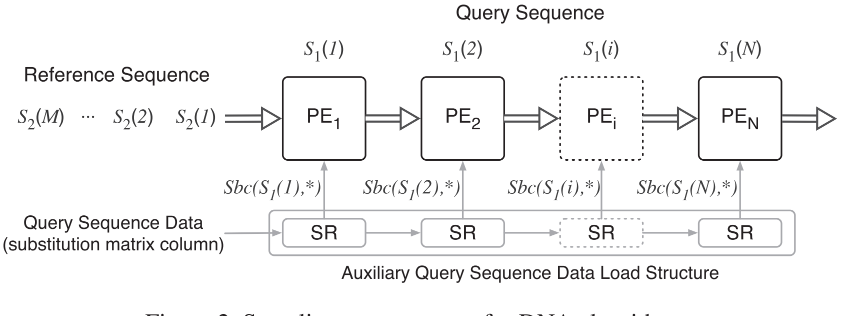

Figure 2. Systolic array structure for DNA algorithms.

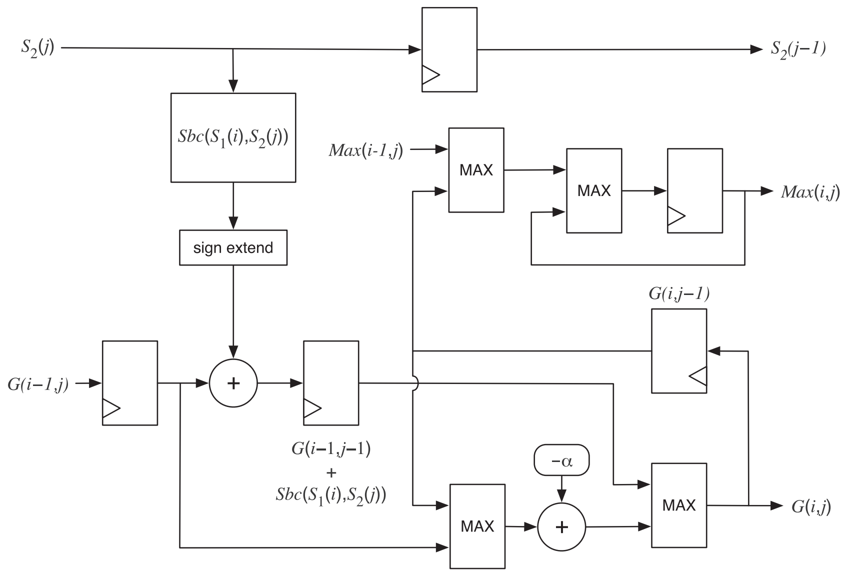

Figure 3. *Base* architecture of processor element PE_i.

4.2. Enhanced processing element

The enhanced PE architecture that is now presented implements the AOEI accelerator technique that was proposed in Section 3. With this technique, the re-computation of the G matrix is reduced to the minimum necessary to perform the traceback phase. 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 has occurred. As it was shown in Section 3, this proposal greatly simplifies the traceback phase, by only focusing on the substrings that are actually involved in the alignment, thus avoiding the re-computation of the whole matrix G.

The architecture of the *enhanced* PE is presented in Figure 4. Each PE features a datapath that implements both Equation (1) and Equation (2). The additional hardware that is required to implement Equation (2) (the AOEI technique) is mainly composed of multiplexers and registers. The set of signals that control these additional multiplexers is generated by the magnitude comparators integrated in the MAX units and that were already present in the *base* PE architecture.

The coordinates of the currently processed matrix cell 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 $MaxC_b(i,j)$.

5. CONFIGURABLE CLASS OF CONCURRENT PROCESSING ACCELERATORS

One important observation that should be noted is concerned with the impossibility to achieve the maximum performance of a linear systolic array when the number of symbols of the query sequence is lower than the number of PEs. This is due to the fact that several of the instantiated PEs cannot perform any useful calculations (no query sequence symbol is attributed to them), thus lowering the PE occupancy rate. This situation can be even aggravated when the query sequences under processing are acquired by short-read sequencing platforms, whose sample sequences can be extremely short. For instance, the *reads* generated by the Illumina platform can be as short as 35 nucleotides long. This scenario would certainly lead to a substantial decrease of the array throughput.

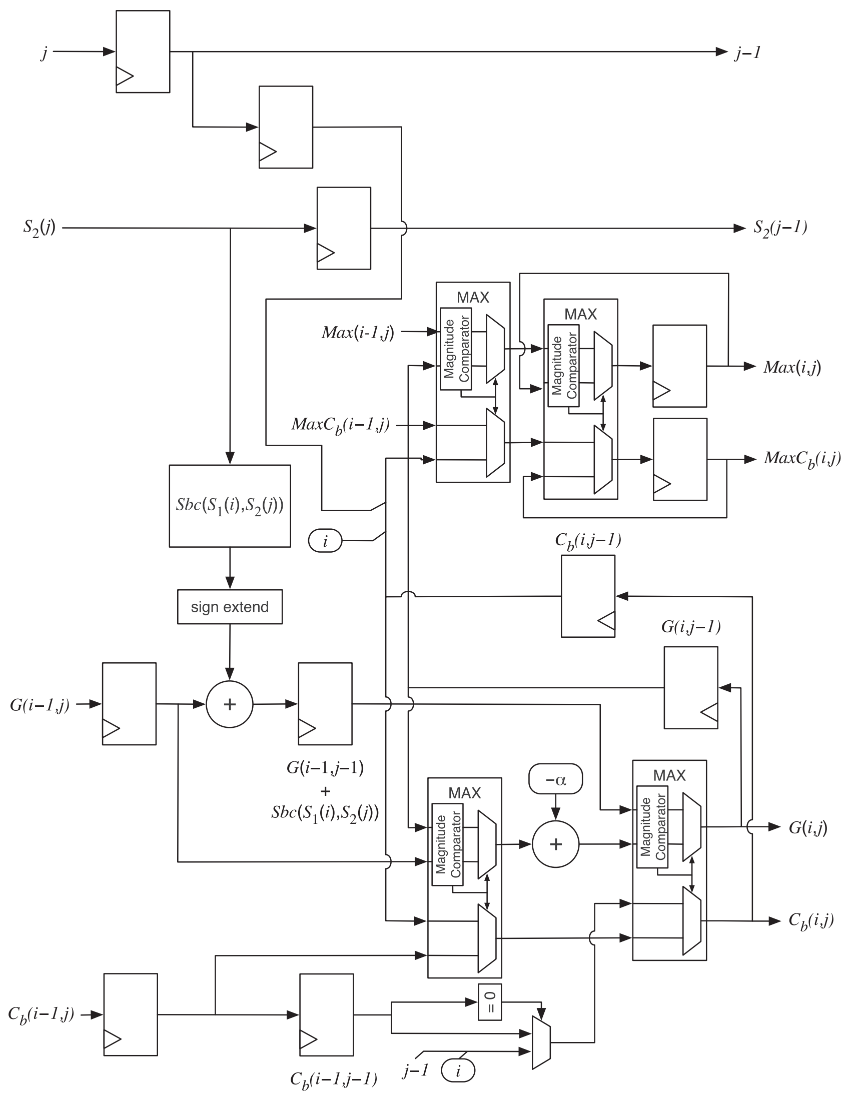

Figure 4. *Enhanced* architecture of processor element PE_i .

Therefore, considering that in most practical setups, there is a very significant number of short-read sequences that must be aligned, or there are several medium-sized query sequences to be aligned to different reference sequences, alternative arrangements of the available PEs are now proposed to maximally use the whole set of implemented PEs and perform several alignments at the same time. Hence, besides the typical single-stream operation mode, in which one query sequence is aligned to one reference sequence, the class of accelerator architectures that is now proposed possesses the capability to be easily reconfigured to operate in several multiple-stream modes: Single-Reference Multiple-Query (SRMQ) or Multiple-Reference Multiple-Query (MRMQ). This new feature significantly improves the actual performance of the array, because it allows a higher PE occupancy rate with the consequent increase on the achieved array throughput and leading to a greater speedup than would be achieved with just a single array.

5.1. Single-Reference Multiple-Query operation mode

When the alignment of various short-read sequences with the same large reference sequence is considered, it is possible to optimize the performance of the proposed accelerator architecture by configuring the hardware accelerator with an SRMQ processing scheme. In such configuration, the accelerator is structured into several coupled linear arrays of PEs that work in parallel and align several query sequences with the same reference sequence, as shown in Figure 5.

Hence, although the reference sequence is shifted into the multiple array structure, the set of independent query sequences to be processed is distributed and assigned to the PEs of the multiple-stream array. The exact number of instantiated parallel PE arrays is configurable according to the size of the short-read sequences to be aligned and to the configuration parameters of the implemented accelerator.

5.2. *Multiple-Reference Multiple-Query operation mode*

Besides aligning several query sequences to the same reference sequence, it is also of interest to align several query sequences to different reference sequences, especially when a large number of PEs is available. In the MRMQ mode of operation, the initial PE array is divided in several smaller and equally sized arrays, which independently process the alignment between distinct pairs of sequences, as shown in Figure 6. This is equivalent to having several accelerators with less PEs working in parallel, but using the same PE resources.

In this mode of operation, each reference sequence is aligned to a single query sequence, thus forming a given sequence pair, which is assigned to a particular smaller PE array. Hence, the several sequence pairs are processed in parallel, and the alignment results of the several pairs are concurrently obtained.

5.3. *Configuration and implementation*

All possible configurations that are offered by the proposed class of hardware accelerators can be obtained by specifying a predefined set of configuration parameters such as (i) the number of PEs that compose the array, (ii) the resolution (bit-width) of the calculated score value ($G(i, j)$), (iii) the resolution of the coordinates ($C_b(i, j)$), (iv) the type of PE that is implemented (*base* or *enhanced*), and (v) the number of multiple-streams to be processed. Each of these parameters can be adjusted to obtain a specific configuration of the accelerating structure, which can subsequently be implemented either on an FPGA or as an ASIC.

On one hand, the inherent reprogrammability of FPGAs can be used to easily implement accelerating structures that are fine-tuned to specific characteristics of the alignments at-hand. For

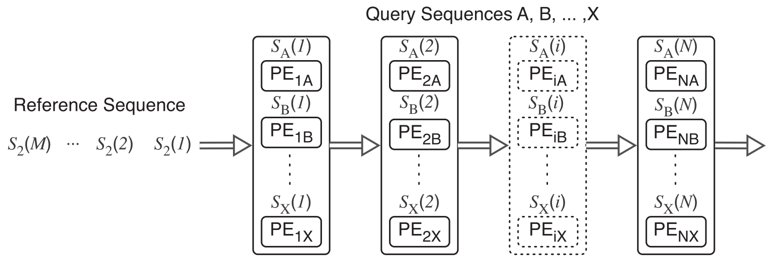
Query Sequences A, B, ... ,X
Reference Sequence
S2(M) ··· S2(2) S2(1)
SA(1) PE1A SB(1) PE1B SX(1) PE1X
SA(2) PE2A SB(2) PE2B SX(2) PE2X
SA(i) PEiA SB(i) PEiB SX(i) PEiX
SA(N) PENA SB(N) PENB SX(N) PENX

Figure 5. Example configuration of a Single-Reference Multiple-Query Processing Element (PE) array.

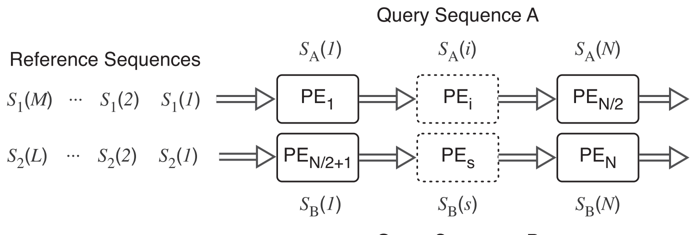

Figure 6. Example configuration of a Multiple-Reference Multiple-Query Processing Element (PE) array.

instance, it is possible to implement an accelerator on the basis of the SRMQ mode of operation by considering specific score and coordinates resolutions. FPGAs also allow the implementation of PE versions specifically optimized for a given operation mode to provide certain improvements in what concerns the resource usage of the accelerator. As an example, in the SRMQ mode, it is possible to share a set of resources that are common to 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), used by the several elements of the arrays that work in parallel, as shown in Figure 7 for a dual-stream SRMQ configuration.

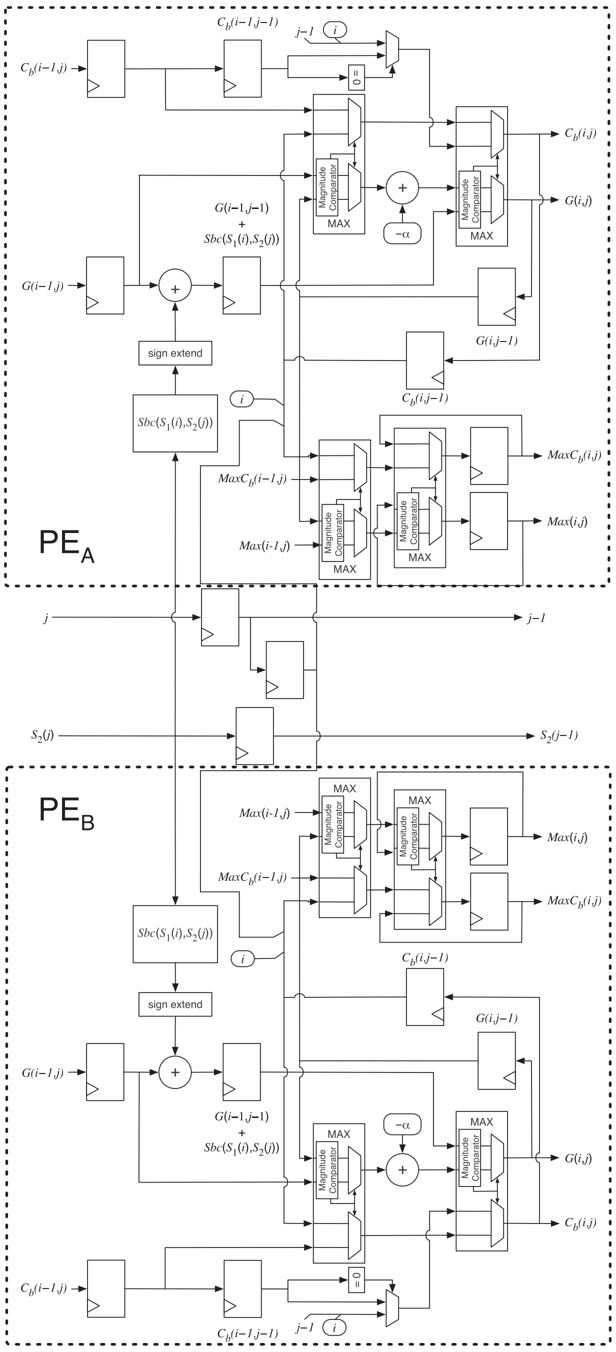

Figure 7. Example of a Single-Reference Multiple-Query Processing Element (PE) in dual-stream Single-Reference Multiple-Query configuration.

On the other hand, ASICs typically offer better performance levels, because they are usually capable of operating at higher clock frequencies and are not constrained by the predefined amount of available resources as in FPGAs, thus allowing the implementation of a larger number of PEs. Moreover, ASIC platforms allow the implementation of accelerator configurations targeting a broader range of possible application scenarios, by including a larger number of PEs and wider resolution of the score and coordinates values.

At this respect, it is important to recall that contrary to what usually happens with statically optimized ASIC devices, the inclusion of the reconfiguration capabilities in the proposed class of accelerators allows the implementation of more than one single configuration of the proposed class in when implemented in an ASIC. Such multiple configuration ASIC device is able to be used in a wider range of application scenarios and still allows it to provide high performance levels because of the higher PE occupancy rate. This dynamic reconfiguration capability is achieved with the inclusion of a special array element in the PE array. Such switching element, depicted in Figure 8, is instantiated at the locations where the array can be split, to form smaller and equally sized PE arrays. For instance, if a 256 PE array is to be split in four smaller arrays with 64 PEs, than three of these switching elements will have to be included in the array. With this approach, the performance of the original array is kept and assured, by including additional registers in the datapath to avoid any increment on the propagation delays of the critical path, thus maintaining the maximum achievable clock frequency. One resulting consequence is that the array latency is increased by one clock cycle for each of the included switching elements. Nevertheless, the impact of this additional latency is not significant in most realistic scenarios, where the smaller arrays do not have less than 35 PEs. With the inclusion of such switching elements, a given hardware accelerator implemented in an ASIC device can be dynamically configured to provide any of the multiple-stream modes of operation (SRMQ and MRMQ) that are proposed in this manuscript. Such feature approximates the flexibility levels provided by the ASIC accelerator with those that are traditionally only offered by FPGA devices.

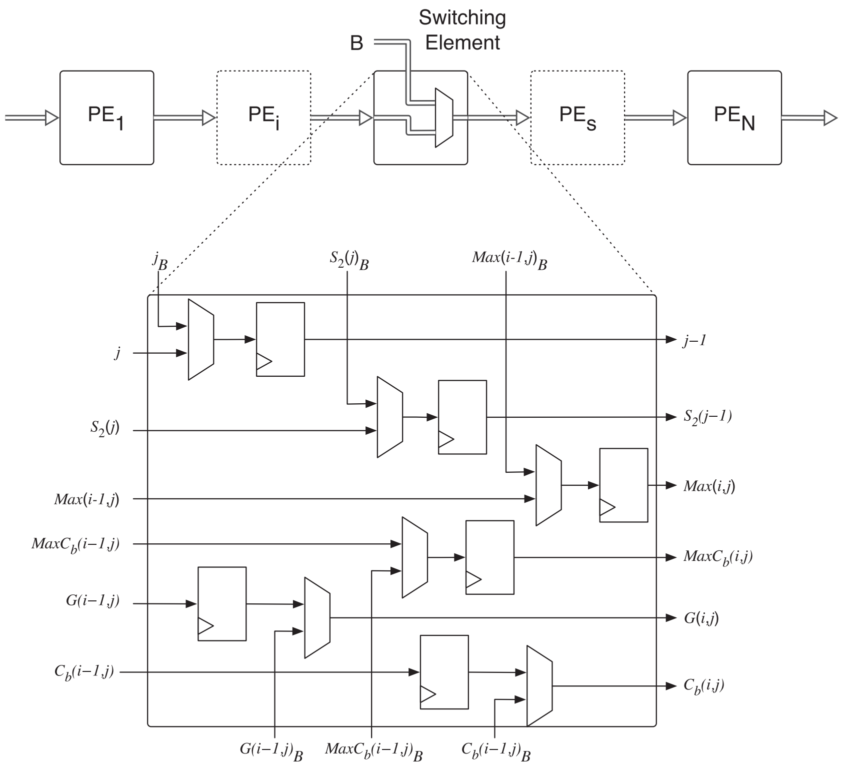

Figure 8. Processing Element (PE) array with the switching element.

5.4. *Array programming*

Because all the comparisons that are performed at each PE always consider the same query sequence character, the PE will just need to 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, as 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 all of the values are zero.

At this respect, an auxiliary query sequence data load structure (depicted in Figure 2) composed by a n bit-wide shift register, was included in the array to program the score values corresponding to query sequence S_1. This temporary storage shift register allows the pre-loading of the *next* query sequence data by serially shifting the substitution matrix column, while the array is still processing the data corresponding to the *current* query sequence. Hence, 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. 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 array, as it avoids the use of a common data bus to program the several PEs.

In case the proposed accelerator architecture is configured in a multiple-stream mode, the several PE arrays simultaneously process different sequences. Hence, each individual PE array will have the corresponding auxiliary query sequence data load structure, which allows the simultaneous load of the query information to the several PEs. In the particular case of the SRMQ mode, the reference sequence is the same for all the PE arrays. Because the query sequences are independently loaded in the arrays, it is only necessary to stream the reference through the several arrays to obtain the alignment results. When the accelerator is implemented in an ASIC, the SRMQ operation mode is achieved by simultaneously loading the same reference to all of the multiple-stream arrays. In case of an FPGA implementation, an optimized PE version can be used, where the reference sequence is loaded only once.

Finally, if the MRMQ mode is used, the several arrays work with different sequence data. However, because the operation in the several PE arrays is synchronous, they simultaneously start and stop their processing. Therefore, the time to process the distinct sequence pairs depends on the longest reference sequence that is under processing. Hence, this synchronous operation reduces the overall complexity of the control and keeps the same high performance levels as those obtained for the single-stream mode.

5.5. *Interface*

To interconnect the proposed hardware accelerator with the GPP that will implement the remaining alignment procedure (i.e. the traceback), the accelerator includes an embedded controller that is responsible for decoding eight instructions that are required to properly control and configure the array, as well as to receive the data to be processed. The developed interface, illustrated in Figure 9, is composed of an output First-In First-Out queue (FIFO) (to return the processed values), a status output, an input FIFO for query sequences and commands and an input FIFO for the reference sequence of each PE array (whenever the MRMQ mode is available). Each of the used FIFOs has a depth of 64 words and is 32-bits wide, to match the bus-width adopted by most current GPPs.

The query and command input FIFO allows the next query sequence to be loaded into the array while the current alignment is being processed. One single FIFO is sufficient to program all the query sequences required for multiple-stream operation. This is possible because the several queries are stored in the auxiliary query sequence data load structures, and then all the PEs are simultaneously loaded with the corresponding query sequence information. The commands provided through the query and command input FIFO specify which array will be loaded with each specific query sequence. The use of one single input FIFO for each reference sequence allows the synchronous

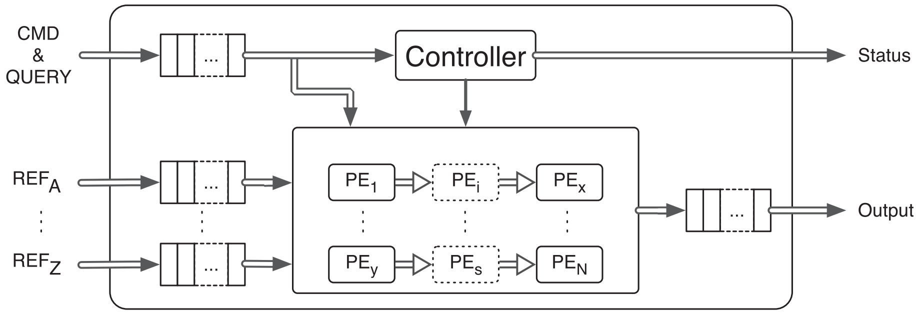

Figure 9. Accelerator interface. PE, Processing Element.

	bits				
Mnemonic	31 - 28	27 - 24	23	22 - 16	15 - 0
`config`	0000	*# arrays*	MS	-	
`rstproc`	0001	-			
`rstquery`	0010	-			
`shiftnxtcost`	0011	*proc array*	-		*query sequence size*
`ldcost`	0100	-			
`ldref`	0101	*proc array*	*reference sequence size*		
`endref`	0110	-			
`getid`	0111	-			

Figure 10. Instruction encoding.

operation of the several arrays without incurring in a higher control complexity. Finally, as soon as the alignment scores and corresponding AOEI coordinates are calculated, they are serially stored in the output FIFO for later processing by the GPP.

The instruction set used by the controller of the proposed hardware accelerator is composed of eight 32-bit instructions, whose encoding is presented in Figure 10. The `config` instruction is only available in the accelerators that implement the multistream MRMQ mode; it is used to configure the desired number of processing arrays (field *# arrays*) and set the arrays to work in MRMQ or SRMQ mode (field *MS*). The `rstproc` instruction is used to reset the PEs without resetting the accelerator's controller, whereas the `rstquery` instruction is used to reset the auxiliary query sequence data load structure. The `shiftnxtcost` is used to load the substitution score data into the appropriate array, selected using the *proc array* field. The *query sequence size* field is used to specify the size of the query sequence that will be loaded. Because of the fact that the instructions and query sequence data are loaded using the same input FIFO, upon receiving the *shiftnxtcost* instruction, the controller will suspend the decoding of further instructions and will immediately start loading the following words received in the *query sequence and command* input FIFO into the corresponding auxiliary query sequence data load structure. This operation will continue until the amount of data corresponding to the *query sequence size* field has been loaded, with the consequent resuming of the instruction decoding operation. The `ldcost` operation performs the loading of the PEs with the new data already stored in the corresponding auxiliary query sequence data load structures. The `ldref` instruction tells the processor to start reading from the reference input FIFO specified by the field *proc array*, loading into the corresponding array the amount of reference elements specified in the field *reference sequence size*. This allows several load commands to be sequentially issued to the same processing array, thus not limiting the size of the reference sequence to a predefined value. The `endref` instruction signals the processor that no additional `ldref` instructions will be given for the current sequences and that the final results can be obtained and written into the output FIFO. The *getid* instruction writes into the output FIFO an identification field that allows the host to determine the capabilities of the current accelerator (e.g. the number of arrays available for the MRMQ mode).

The status output, whose encoding is shown in Figure 11, contains information about the available positions (full / almost full) in each of the input FIFOs, allowing the implementation of a

bits	32	31	30	29 - 0
fields	OA	II	IC	full / almost full bits

Figure 11. Status output encoding.

flow control mechanism. Furthermore, this status output also indicates when output data is available in the output FIFO (field *OA*), reporting that the accelerator has concluded its computation of the alignment. The status output also indicates when an invalid instruction is received (field *II*) and when an invalid configuration is attempted (field *IC*).

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 consists of a Leon3 GPP processor [17] 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 Leon3 processor [17] is one of the most used freely available processor Intellectual Property (IP) cores and consists of a highly configurable and fully synthesizable core, described in VHDL, implementing a Reduced Instruction Set Computer (RISC) architecture conforming to the SPARC v8 definition. The Leon3 32-bit core is based on a 7-stage instruction pipeline adopting a Harvard micro-architecture, with 32-bit internal registers. The core functionality can be easily extended by means of the Advanced Microcontroller Bus Architecture (AMBA) 2.0 on-chip buses (Advanced High-performance Bus (AHB) and Advanced Peripheral Bus (APB)). The AHB bus is used to connect the Leon3 processor with high-speed controllers, such as the cache and memory controllers. On the other hand, the APB bus is used to access most on-chip peripherals and is accessed through the AHB/APB Bridge. External memory access and memory mapped Input/Output (I/O) operation are provided by a programmable memory controller with interfaces to Programmable Read-Only Memory (PROM), Static Random-Access Memory (SRAM) and Synchronous Dynamic Random-Access Memory (SDRAM) chips.

The proposed hardware accelerator was interconnected with the Leon3 processor by means of the AMBA-2.0 APB bus as a slave device. The selection of this bus arises not only because it offers enough bandwidth for all of the sequence data transfers, but also because it provides a simple interface and low-power consumption. Furthermore, some additional wrapper logic, responsible for the adaptation of the accelerator interface to the AMBA-2.0 APB bus, was needed. The I/O FIFOs and the status register of the alignment core were mapped in the Leon3 memory address space. Hence, with such interface, the write and read operations over this peripheral can be easily implemented using simple load and store operations.

7. RESULTS

Two distinct implementation platforms were considered to evaluate the offered performance of the proposed accelerator: an FPGA and an ASIC. In the FPGA device, it was implemented a complete alignment system composed of the proposed accelerator interconnected with the Leon3 GPP. This embedded alignment system, used fundamentally as a proof-of-concept prototyping platform, was implemented by using a GR-CPCI-XC4V development board from Pender Electronic Design [18]. 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 entire system was described using parameterizable VHDL code and synthesized using Xilinx ISE 10.1 (SP3) software tools.

The ASIC implementation of the accelerator was primarily used to assess its performance and resources requisites in a less restrictive prototyping technology, that is, without the restrictions imposed by the GPP and the adjacent peripherals (e.g. DRAM memory). The accelerator was synthesized with a StdCell library from Faraday Technology, based on a 90 nm CMOS process from UMC [19], using the *Synopsys Design Compiler* tools. The selection of the parameterization that was adopted by the synthesized accelerator took into account not only the possibility to handle very large sequences, but also the ability to support multiple configurations suited for a broad range of application scenarios.

7.1. Resource usage

The obtained resource usage results of the entire embedded prototyping system that was implemented in the FPGA platform are presented in Table V, considering either single-stream ($n-$stream $= 1$) and multiple-stream ($n-$stream > 1) array configurations. This table also includes the resources occupied solely by the Leon3 processor, revealing that this GPP alone occupies 18% of the available logic resources of the used FPGA device. In what concerns the resource usage of the processing array, using the *enhanced* PEs, it is possible to observe that it is 77% larger than the corresponding *base* configuration (i.e., 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 resolution of the coordinate representation) and the adopted scoring scheme (which influences the resolution of the score calculations).

Because of the strict hardware restrictions imposed by this prototyping device, the presented configurations only considered the SRMQ operation mode, because this FPGA does not support the implementation of a larger number of PEs. In fact, as it will be seen in the following subsection, the MRMQ mode is more suited for ASIC implementations, where the newly proposed reconfiguration capabilities (that are not natively present in ASICs) can be efficiently exploited. Because of the already mentioned FPGA restrictions, the maximum number of supported multiple-streams is 3. However, the proposed class of accelerators supports any number of concurrently processed streams, provided that there are enough available resources to implement them. The corresponding hardware requisites for $n-$stream$=2$ and $n-$stream$=3$ were also presented in Table V.

Finally, the maximum operating frequency that was obtained upon the synthesis of the proposed accelerating core in this FPGA is 120 MHz. Nevertheless, the actual clock frequency used by the

Table V. Resource usage of the Field Programmable Gate Array implementation (Single-Reference Multiple-Query multiple-stream mode).

PE			Score width	Maximum size		Resource usage	
Type	#	n-stream		Reference	Query	Registers	LUTs
Base	16	1	7	–	16	7441 (8%)	19818 (20%)
Base	64	1	9	–	64	11736 (12%)	28148 (29%)
Base	128	1	10	–	128	16031 (16%)	34130 (35%)
Enh.	16	1	7	2^{16}	16	9499 (10%)	22168 (23%)
Enh.	35	1	9	2^{22}	35	15427 (16%)	28071 (29%)
Enh.	37	1	9	2^{28}	37	16618 (17%)	28941 (29%)
Enh.	64	1	9	2^{22}	64	22625 (23%)	36114 (37%)
Enh.	128	1	10	2^{22}	128	40024 (41%)	56541 (58%)
Enh.	35	2	9	2^{22}	35	24149 (25%)	38169 (39%)
Enh.	35	3	9	2^{22}	35	32687 (33%)	48205 (49%)
Enh.	37	2	9	2^{28}	37	26471 (27%)	39140 (40%)
Enh.	37	3	9	2^{28}	37	36117 (37%)	49493 (50%)
Enh.	64	2	9	2^{22}	64	38349 (39%)	54183 (55%)
	Leon3		–	–	–	6246 (6%)	17788 (18%)

PE, Processing Element.

Table VI. Application Specific Integrated Circuit implementation hardware resources (Multiple-Reference Multiple-Query multiple-stream mode).

PE			Score width	Maximum size		Implementation area (mm^2)
Type	#	n-Stream		Reference	Query	
Enh.	64	1	9	2^{22}	64	0.826
Enh.	512	8	12	2^{28}	512	9.261

PE, Processing Element.

entire prototyping embedded system is 60 MHz, because of limitations that are strictly imposed by the considered Leon3 GPP IP core.

In what concerns the ASIC implementation of the proposed accelerator, the occupied area for the two analyzed configurations is presented in Table VI. Just as in the FPGA prototype, it is still possible to implement larger arrays, by simply considering parameterizations that include more PEs. In fact, an ASIC usually provides the possibility to implement longer arrays, thus allowing a much higher number of concurrent score calculations and significantly broadening its application scenarios. As an example, an accelerator with 512 PEs provides the capability to align sequences directly obtained from the GS FLX Genome Analyzer (454) sequencing platform [1], which typically has a read length of 400 bp. The same is also true in what concerns the offered performance levels (as it will be seen in the next subsection), because it allows significantly higher operating frequencies (the 250 MHz operating frequency was determined using results from the synthesis tool and was subsequently confirmed and validated). Hence, both of these factors contribute to a much higher processing throughput of the ASIC platform. Furthermore, with the added MRMQ multiple-stream feature, the performance of this accelerator can still be considerably higher.

7.2. *Performance analysis*

To evaluate the performance speedup that is provided by the proposed system, a set of real DNA sequences obtained from the GenBank database [2] were used in the conducted experiments. In what concerns the processed *query* sequences, their maximum size is limited by the number of available PEs in the array. Consequently, for the considered prototyping configuration implemented in the FPGA, it should not be greater than 128 nucleotides long (a size entirely compatible with the reads generated by the latest Next-Generation Sequencing Technologies, such as the Solexa 1G and the SOLiD sequencers [1]). For the considered ASIC implementation, the maximum query sequence size is 512 nucleotides.

In the conducted evaluation of the proposed accelerator, it was mainly considered a *short-read versus a reference genome* alignment task. In fact, because the typical size of the sequences that are processed in this procedure is usually small, it often represents a quite challenging application domain, which can be used to better assess the actual performance provided by the array accelerator in a scenario quite close to the worst case situation. For this alignment task, the unmasked genomic DNA sequence of the Mus Musculus Chromosome 1 from release 58 of the NCBIM37 assembly, composed by 200×10^6 nucleotides, was used as the *reference* genome sequence. The short-reads, used as the *query* sequences, were obtained from run SRR058608 of study SRP002695, available at the Sequence Read Archive (SRA) database of NCBI. A total of 8223733 reads were available, each one with 37 bps. To perform the evaluation tests, the first 100 sequences were aligned with the reference genome.

Besides comparing the results that were obtained by using the two considered implementation technologies, the conducted assessment also considered, as a reference, the alignment performance that can be obtained with a generic computational alignment system, consisting of a 2.4 GHz Intel Core2 Duo processor executing the SSEARCH35 software tool from the FASTA framework [5], with Single Instruction Multiple Data (SIMD) optimizations.

Table VII presents the obtained execution times for both the FPGA and ASIC implementations of the accelerator, and for the GPP running the state-of-the-art software tool from FASTA

Table VII. Performance results for the short-read versus reference genome application.

	Intel Core2 Duo	FPGA accelerator	ASIC accelerator
Frequency	2400MHz	60MHz	250MHz
# PE (total)	–	111	512
n-stream	1	3	8
Alignment time (s)	3064	115	11
Speedup	1	27	278

FPGA, Field Programmable Gate Array; ASIC, Application Specific Integrated Circuit; PE, Processing Element.

(SSEARCH35). All those execution times consider all the data transfers and control overheads to compute the complete alignments. In what concerns the SSEARCH35 tool, some limitations already reported in the community were observed when dealing with large sequences, making it necessary to perform several partial runs to obtain the entire set of alignments. Therefore, the presented alignment time is the sum of the partial times, which are, in turn, reported by the tool itself. Regarding to the accelerating platform implemented in the FPGA, whose configuration is reported in Table V (#PE $= 37$ and $n-$stream $= 3$), the reported time includes the post-processing tasks performed by the embedded Leon3 processor to obtain the alignments.

A more detailed analysis of the processing time of the several components is presented in Table VIII. At this respect, it is worth noting that the accelerator works in a data streaming mode of operation, therefore starting its processing as soon as there is some data available, which not only reduces the latency to begin the processing, but also reduces the internal storage memory requirements for the sequence data. Furthermore, the post-processing operations that are executed in the Leon3 GPP are also performed, whereas the accelerator is processing a new set of sequences. Thus, it is possible for the Leon3 processor to perform the post-processing tasks in parallel with the accelerator (using a pipelined processing scheme). The results presented in Table VIII consider two distinct situations: (i) a simple serial processing scheme, where just three queries are aligned (only one 3-stream data set, thus only one iteration of the array) and in which the post-processing operations (② and③) are only performed after the accelerator ends, and (ii) a pipelined processing scheme, where 100 queries are aligned (34 iterations of the array) and in which almost all of the post-processing operations are performed while the accelerator is processing a new data set. The pipelined processing scheme naturally occurs when the accelerator has to align a large set of queries, which is the most common scenario in bioinformatics applications. The presented results depict the time of each of the processing phases: (i) the score and coordinate calculation time, which is performed in the hardware accelerator and includes the time to transfer the code and initial data to the accelerator, (ii) the processing time in the Leon3 GPP to perform the reduced matrix fill, and (iii) the traceback processing time in the Leon3 GPP. As it is possible to observe, the accelerator is, in this case, the limiting factor of the performance on the entire alignment system.

The obtained values reveal that the conceived accelerating platform implemented in the FPGA device provides a speedup as high as 27 when compared with a pure and highly optimized software

Table VIII. Decomposition of the alignment time (s) for the Field Programmable Gate Array accelerator implementation.

# Queries	Score and coordinates (Hardware) ①	Reduced matrix fill (Leon3) ②	Reduced traceback (Leon3) ③	Total Time ④
3	3.4	0.050	0.001	3.5
100	115.3	1.741	0.033	115.4

Legend: ④ $\approx \max\left\{ ① ;\ ② + ③ \right\}$

implementation running in the Intel Core2 Duo processor. Such significant speedup is mainly due to a very efficient usage of the available hardware resources that are provided by the configurable accelerator platform, which enabled a triple-stream configuration using the same FPGA device. It should be noted that such performance advantage is still significant even when compared with the performance offered by typical multicore processors currently available in the end-user market. In fact, in the case in which a multithreaded version of the SSEARCH35 tool is used, whose execution time reduces almost linearly with the number of available cores, the attained speedup when using the accelerator (27) still exceeds the maximum theoretical speedup obtained with a typical multicore system (e.g. with four or six processing cores).

On the other hand, contrasting with the evaluation that was conducted for the FPGA prototype, where the proposed accelerator was integrated in a complete embedded alignment system, the performance of the ASIC prototype was evaluated by considering an IP core perspective of the accelerator. Under this view, it was assumed a common scenario where the required time to compute the alignment score (as well as its origin coordinates) in the hardware accelerator is still much larger than the time that is needed by the GPP to perform the reduced matrix calculation and the subsequent traceback operation. Such an assumption was carefully validated in the considered Intel Core2 Duo processor, which requires about 1 ms to process all the reduced matrices and to perform the corresponding tracebacks. The execution time of the ASIC accelerator (11 s) was obtained using a behavioral simulation model, which allowed to determine the exact number of clock cycles to perform the calculations in the accelerator.

Hence, for the considered implementation scenario, where the accelerator was parameterized to allow the simultaneous alignment of up to eight different query sequences (configuration presented in Table VI), it was obtained a speedup as high as 278, when compared with a sequential execution in the Intel Core2 Duo processor. It is still worth noting that such improved performance was obtained by using an operating frequency that is almost 10 times lower than the GPP clock frequency, which provides extra benefits from the power consumption point of view.

Finally, in Table IX, it is depicted a short presentation of the performance levels that can be obtained by using some configurations that are offered by the adopted parameterization of the ASIC implementation of the proposed class of accelerators. It can be seen from these results that the innovative multiple-stream feature provides a useful and straightforward adaptation of the implemented accelerating structure to the target processing application, allowing a reduction of the alignment time from 91 s to only 11 s when the accelerator is aligning the 37 long query sequences and is configured for 8-stream operation.

7.3. Comparison and discussion

The direct comparison of the obtained performance results of the presented accelerator with previously proposed accelerators also implemented in a FPGA is not entirely fair, because previous accelerators only accelerate the matrix fill phase of the S–W algorithm. Nevertheless, in what concerns the raw throughput (not considering the advantages of the new AOEI method), the performance of the presented accelerator (according to the widely used Giga Cell Updates

Table IX. Possible dynamic configurations of the considered Application Specific Integrated Circuit implementation.

PE			Maximum size		Alignment time (s)
# (Total)	# (Per array)	n-Stream	Reference	Query	
512	64	8	2^{28}	64	11
512	128	4	2^{28}	128	23
512	256	2	2^{28}	256	45
512	512	1	2^{28}	512	91

PE, Processing Element.

per Second (GCUPS) metric) is 6.5 GCUPS for the FPGA implementation and 67 GCUPS for the ASIC implementation. The raw throughput of other already presented solutions ranges from 0.8 GCUPS (140 PEs) [7] to 5.4 GCUPS (135 PEs) [9] and 7.6 GCUPS (168 PEs) [8]. Therefore, not only does the presented hardware accelerator maintain a similar level of raw performance when compared with other solutions implemented in FPGAs, but it also provides the additional benefits of the AOEI method and the performance improvements enabled by the proposed modes of operation.

Nevertheless, there are also other high performance implementations based on programmable devices (e.g. multicore CPUs and GPUs) that may also attain a high processing performance. In the case of the used CPUs, the attained performance of the SSEARCH35 tool is about 0.250 GCUPS. Despite the fact that such performance scales almost linearly with the number of processing nodes, the presented results have shown that a large number of processing nodes is required to attain a performance level equivalent to that of the presented accelerator. Such large number of processing cores will inevitably present a much higher power consumption than the proposed accelerator implementations. On the other hand, there are also efficient implementations of the S–W algorithm specifically designed to run on GPUs, such as the CUDASW++ [20]. However, although the attained performance in the GPU can be significantly high (17 GCUPS), such implementation imposes strict limitations on the maximum size of both sequences. Thus, unlike the proposed accelerator, which is able to align arbitrarily long reference sequences, such tool can not be used to align large reference sequences. Moreover, the power consumption of the GPU device is significantly high when compared with either an FPGA device or a custom ASIC device. Furthermore, considering an ASIC implementation with the presented configuration, the performance of the accelerator is almost four times higher than that of the CUDASW++ implementation executed on a NVidia GeForce GTX 280 GPU.

In what concerns the performance of the complete alignment system, it must be noted that the developed proof-of-concept prototype implements the whole system with limited hardware resources (due to the model of the used FPGA device). Despite the considerable performance results obtained in such a system, it would be possible to improve the overall performance by implementing the accelerator with a more recent and larger FPGA device (e.g. a Xilinx Virtex7 device) interconnected by using the Peripheral Component Interconnect Express (PCIe) bus to a computational system with a typical off-the-shelf GPP (e.g. an Intel multicore processor). This implementation would easily scale not only the dimension of the accelerator array, by allowing either more PEs or a higher number of simultaneous streams (due to the higher amount of available resources in the FPGA device), but also the processing capacity of the GPP and the data-transfer bandwidth, by using one or more PCIe interconnection lanes [21]. This type of computational systems greatly benefits from the reconfiguration capabilities of the FPGA device, which allows the accelerator to be dynamically adapted to the current alignment task required by the considered application. Furthermore, it also significantly benefits from the FPGA's lower power consumption when compared with GPP implementations. However, whenever the highest performance levels are required, an ASIC implementation will possibly provide the best option, but with some reduced flexibility. In terms of power consumption, such highly customized implementation may also achieve the smallest power consumption per cell update.

Finally, from a system integration point of view, the use of a platform that integrates the proposed accelerator (either in an FPGA or an ASIC device) in a typical computational environment (like the one previously described) significantly improves and facilitates the processing of biological sequences, such as DNA. In fact, such environment allows an easier use of the proposed accelerator architecture on already existing alignment tools that either execute a pure S–W alignment algorithm (e.g. SSEARCH) or use such algorithm in one of the processing phases (e.g. BLAST [4] heuristic tool). In fact, by using the accelerator in conjunction with heuristic tools, it will be possible to greatly improve the quality of the whole alignment, because of the fact that heuristic tools constantly balance the sensitivity of the alignment with the execution time. By using this hardware accelerator, it is possible to allow a greater number of sequences to be passed on to the higher sensitivity stage in which the S–W algorithm is executed, while still maintaining the overall speed.

8. CONCLUSIONS

s This paper presents a new class of flexible and configurable hardware accelerators capable of providing a high-performance alignment solution based on the widely adopted S–W algorithm. The resulting accelerating structures are easily adapted to various application scenarios and exploit an innovative technique that significantly reduces the time and memory requirements to perform the subsequent traceback phase. Moreover, a distinctive characteristic of the proposed class is the possibility to concurrently process several small query sequences, which not only improves the performance of the accelerator but also provides a much more efficient usage of the instantiated PEs, when dealing with small query sequences.

The new class of accelerators can be implemented either in FPGA or ASIC technologies. When using FPGAs, their inherent reprogramming capabilities can be used to fine-tune the accelerator instantiation to the current alignment requisites, by choosing the most appropriate number of PEs and the resolution of representation. In contrast, ASIC implementations can take advantage of custom configurations with a higher operating clock frequency and a larger number of PEs. The obtained accelerating structures incorporate dedicated PEs that allow to optimize the resource usage in FPGAs and easily provide several configurations in ASIC implementations.

The obtained results have shown that speedups as high as 278 can be obtained in ASIC implementations when compared with off-the-shelf personal computers. In fact, it was also observed that even the FPGA-based prototyping platform, where a complete embedded alignment system was implemented, provided speedups as high as 27, while running at a 40 times lower clock frequency.

ACKNOWLEDGEMENTS

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project *“HELIX: Heterogeneous Multi-Core Architecture for Biological Sequence Analysis”* with reference to PTDC/EEA-ELC/113999/2009, project PEst-OE/EEI/LA0021/2011, and by the PhD grant with reference to SFRH/BD/43497/2008.

REFERENCES

1. Shendure J, Ji H. Next-generation DNA sequencing. *Nature Biotechnology* 2008; **26**(10):1135–1145.
2. Benson DA, Karsch-Mizrachi I, Lipman DJ, Ostell J, Sayers EW. GenBank. *Nucleic Acids Research* 2010; **38**(Database):D46–51. DOI: 10.1093/nar/gkp1024.
3. Smith TF, Waterman MS. Identification of common molecular subsequences. *Journal of Molecular Biology* 1981; **147**(1):195–197.
4. Altschul S, Gish W, Miller W, Myers E, Lipman D. Basic local alignment search tool. *Journal of Molecular Biology* 1990; **215**(3):403–410.
5. Pearson WR, Lipman DJ. Improved tools for biological sequence comparison. *Proceedings National Academy of Sciences of the United States of America* 1988; **85**(8):2444–2448.
6. Ligowski L, Rudnicki W. An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases. In *IEEE International Symposium Parallel & Distributed Processing. IPDPS 2009*. IEEE: Rome, Italy, 2009; 1–8.
7. Hasan L, Al-Ars Z, Nawaz Z, Bertels K. Hardware implementation of the Smith-Waterman Algorithm using recursive variable expansion. In *3rd International Design and Test Workshop, IDT 2008*. IEEE: Monastir, Tunisia, 2008; 135–140.
8. Oliver T, Schmidt B, Maskell D. Hyper customized processors for bio-sequence database scanning on FPGAs. In *Proceedings 13th International Symposium Field-Programmable Gate Arrays, FPGA’05*. ACM: Monterey, California, USA, 2005; 229–237.
9. Benkrid K, Liu Y, Benkrid A. A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2009; **17**(4):561–570.
10. Nawaz Z, Nadeem M, van Someren J, Bertels KLM. A parallel FPGA design of the Smith-Waterman traceback. *International Conference on Field-Programmable Technology, FPT 10*, Beijing, China, 2010; 454–459.
11. CLC Bio. White paper on CLC Bioinformatics Cube 1.03. *Technical Report*, CLC Bio, Finlandsgade 10-12 - 8200 Aarhus N - Denmark, May 2007.
12. Singh R, Hoffman D, Tell S, White C. BioSCAN: a network sharable computational resource for searching biosequence databases. *Bioinformatics* 1996; **12**(3):191–196. DOI: 10.1093/bioinformatics/12.3.191.
13. Guerdoux-Jamet P, Lavenier D. SAMBA: hardware accelerator for biological sequence comparison. *Bioinformatics* 1997; **13**(6):609–615. DOI: 10.1093/bioinformatics/13.6.609.

14. Grate L, Diekhans M, Dahle D, Hughey R. Sequence analysis with the Kestrel SIMD parallel processor. In *Pacific Symposium on Biocomputing*, 2001; 263–274.
15. Lloyd S, Snell Q. Sequence alignment with traceback on reconfigurable hardware. In *International Conference Reconfigurable Computing and FPGAs - ReConFig '08.* IEEE: Cancun, Quintana Roo, Mexico, 2008; 259–264, DOI: 10.1109/ReConFig.2008.30.
16. Sebastião N, Dias T, Roma N, Flores P. Integrated accelerator architecture for DNA sequences alignment with enhanced traceback phase. *International Conference on High Performance Computing and Simulation. HPCS 2010*, Caen, France, 2010; 16–23, DOI: 10.1109/HPCS.2010.5547154.
17. Aeroflex Gaisler. *SPARC V8 32-bit Processor LEON3 / LEON3-FT Companion Core Data Sheet, Version 1.0.3*, December 2008.
18. Gaisler Research / Pender Electronic Design. *GR-CPCI-XC4V Development Board – User Manual*. 0.1 edn, April 2006.
19. Faraday Techn. Corp.. *FSD0C_A 90nm Generic Core Cell Library (v0.3)*, February 2009.
20. Liu Y, Schmidt B, Maskell D. CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. *BMC Research Notes* 2010; **3**(1):93. DOI: 10.1186/1756-0500-3-93.
21. Lund K. White paper: PCI Express for the 7 Series FPGAs. *Technical Report WP384*, Xilinx Inc., March 2011. Available from: http://www.xilinx.com/support/documentation/white_papers/wp384_PCIe_7Series.pdf [Accessed on 2012/06/11].