Integrated Hardware Architecture for Efficient Computation of the n-Best Bio-Sequence Local Alignments in Embedded Platforms

Nuno Sebastião, *Student Member, IEEE*, Nuno Roma, *Member, IEEE*, and Paulo Flores, *Member, IEEE*

Abstract—**A flexible hardware architecture that implements a set of new and efficient techniques to significantly reduce the computational requirements of the commonly used Smith–Waterman sequence alignment algorithm is presented. Such innovative techniques use information gathered by the hardware accelerator during the computation of the alignment scores to constrain the size of the subsequence that has to be post-processed in the traceback phase using a general purpose processor (GPP). Moreover, the proposed structure is also capable of computing the n-best local alignments according to the Waterman–Eggert algorithm, becoming the first hardware architecture that is able to simultaneously evaluate the n-best alignments of a given sequence pair, by incorporating a set of ordering units that work in parallel with the systolic array. A complete alignment system was developed and implemented in a Virtex-4 FPGA, by integrating the proposed accelerator architecture with a Leon3 GPP. The obtained experimental results demonstrate that the proposed system is flexible and allows the alignment of large sequences in memory constrained systems. As an example, a speedup of 17 was obtained with the conceived system when compared with a regular implementation of the LALIGN35 program running on an Intel Core2 Duo processor running at a 40× higher frequency.**

Index Terms—**DNA sequence alignment, hardware accelerator, Smith–Waterman (S-W), Waterman–Eggert (W-E).**

I. Introduction

With the advent of the *Next-Generation Sequencing Technologies* [1], large amounts of sequenced deoxyribonucleic acid (DNA) are currently available for analysis by biologists and researchers. The current release of the GenBank [2] database, one of the largest public databases of DNA sequences, includes over 117×10^9 base pairs.

Biologists typically use the sequence alignment procedure as the main tool to extract information from this huge amount of data. Such a procedure can be divided in two major classes: *global* and *local* alignments. While in global alignments the complete sequences are aligned from one end to the other, in local alignments only the subsequences that present the highest similarity are considered.

One of the most widely adopted algorithms to find the optimal local alignment between a pair of DNA sequences is the Smith–Waterman (S-W) algorithm [3]. This algorithm makes use of a dynamic programming (DP) method and is characterized by the smallest runtime, with a time complexity of $\mathcal{O}(nm)$, where n and m denote the sizes of the sequences being aligned. The S-W algorithm operates in two phases: a *matrix fill* phase, which computes the alignment score, and a *traceback* phase, where the alignment is actually determined. Alternative *sub-optimal* heuristic algorithms, like BLAST [4], have been proposed to reduce this runtime. However, such speedup comes at the cost of a nonnegligible possibility of missing the optimal alignment between the sequences. As a consequence, the *optimal* algorithm is often preferred but not always used due to its significant runtime. Furthermore, besides determining the optimal local alignment, it is also frequently of interest to find other alternative alignment positions with approximate scores: the n-best local alignments [5].

Several different approaches have been proposed to accelerate the execution of the S-W algorithm. These solutions range from parallel implementations running in general purpose processors (GPPs) using SIMD instructions [6]–[8] and graphics processing units (GPUs) [9]–[12] to specialized processors [13] and dedicated hardware architectures. Among the last, the most common are based on systolic arrays, such as the bidimensional structure presented in [14]. Nevertheless, unidimensional (linear) systolic arrays tend to be more commonly adopted [15]–[17]. Some commercial solutions using proprietary architectures implemented in field-programmable gate arrays (FPGAs) were also made available, such as the *Bioinformatics Cube* by *CLC bio* [18] and the SeqCruncher by TimeLogic [19].

However, most of these solutions are only focused on accelerating the *matrix fill* phase of the S-W algorithm, disregarding the *traceback* phase, which is typically performed using a GPP in a postprocessing step. Although [20] proposes an architecture that also accelerates this traceback phase, such a solution is only applicable to the global alignment problem, which has different requirements than the local alignment. Moreover, none of the proposed architectures has dealt with the problem of simultaneously determining, in a single pass, the set of the n-best local alignment scores of the sequence pair under processing.

Meanwhile, there has been a growing interest in the development of efficient and reconfigurable platforms for embedded processing. Examples of such platforms include the new Intel Atom Processor E600C series [21] and the upcoming Xilinx Ex-

Manuscript received October 07, 2010; revised March 12, 2011; accepted April 26, 2011. Date of publication July 14, 2011; date of current version June 01, 2012. This work was supported by the Portuguese Foundation for Science and Technology (FCT) under the project "*HELIX: Heterogeneous Multi-Core Architecture for Biological Sequence Analysis*, with reference PTDC/EEA-ELC/113999/2009 , the PIDDAC Program funds (INESC-ID multiannual funding), and the Ph.D. grant with reference SFRH/BD/43497/2008.

The authors are with the INESC-ID/IST-TU Lisbon, 1000-029 Lisboa, Portugal (e-mail: Nuno.Sebastiao@inesc-id.pt; Nuno.Roma@inesc-id.pt; Paulo.Flores@inesc-id.pt).

Digital Object Identifier 10.1109/TVLSI.2011.2157541

tensible Processing Platform [22], which merge the benefits of a reasonably powerful GPP with the flexibility provided by a FPGA. These platforms allow an embedded system to achieve a reasonable performance level by implementing, in the reconfigurable fabric, an optimized accelerator, while still maintaining a low power consumption and a reduced market cost.

In this paper, we propose a new hardware accelerator architecture that is the first to accelerate the execution of the Waterman–Eggert (W-E) algorithm for the n-best sequence alignment problem and that is capable of efficiently exploiting reconfigurable embedded platforms. To attain the offered performance levels, this architecture exploits several important contributions: 1) an innovative and efficient technique to significantly reduce the time and memory requirements of the software *traceback* phase, by making use of a specially conceived hardware processing array to gather additional information during the computation of the alignment scores in the *matrix fill* phase; 2) a new hardware processing unit to track and accelerate the calculation of the n-best alignments; 3) a partition buffer structure to compact and store intermediate results during the alignment of large sequences; and 4) integration of the proposed hardware accelerator with a GPP to form a complete and efficient local alignment system.

This paper is organized as follows. After a brief overview of the commonly used S-W algorithm for pairwise sequence alignment and of the W-E algorithm used to solve the n-best alignment problem, provided in Section II, the proposed approach to speed up the execution of the two previous algorithms is introduced in Section III. Section IV presents the flexible accelerator architecture that was developed to implement the previous method. The used prototyping platform is detailed in Section V, while Section VI presents the obtained experimental results. Finally, the conclusions are drawn in Section VII.

II. Pairwise Local Sequence Alignment

Considering any two strings S_1 and S_2 with sizes n and m, respectively, the optimal local alignment reveals the pair of substrings of S_1 and S_2 that optimally align, such that no other pair has a higher similarity score. Besides the optimal local alignment, it is also possible to find other locations where the two sequences are similar, by finding the n-best alignments.

A. Optimal Local Alignment

The S-W algorithm, characterized by a $\mathcal{O}(nm)$ time complexity [3], is commonly used for this purpose. This algorithm operates in two distinct phases: it starts by filling a score matrix H, followed by a traceback phase over this matrix. The matrix is often filled using an *affine* gap penalty model, defined as

$$H(i,j) = \max\begin{cases} H(i-1,j-1) + Sbc(S_1(i), S_2(j)), \\ E(i,j), \\ F(i,j) \\ 0 \end{cases}$$

$$F(i,j) = \max\begin{cases} H(i-1,j) - \alpha, \\ F(i-1,j) - \beta \end{cases}$$

$$E(i,j) = \max\begin{cases} H(i,j-1) - \alpha, \\ E(i,j-1) - \beta \end{cases} \tag{1}$$

in which opening and extending a gap have different costs given by α and β, respectively. $Sbc(S_1(i), S_2(j))$ denotes the substitution score value obtained by aligning character $S_1(i)$ against character $S_2(j)$. In the particular case of $\alpha = \beta$, a *linear* gap penalty model is obtained, where opening and extending a gap has the same cost (α). The recurrence equations for this simpler model are

$$H(i,j) = \max\begin{cases} H(i-1,j-1) + Sbc(S_1(i), S_2(j)) \\ H(i-1,j) - \alpha \\ H(i,j-1) - \alpha \\ 0 \end{cases} \tag{2}$$

with initial conditions $H(i,0) = H(0,j) = 0$.

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

Best local alignment

```
S2: G C C A T T G
    | | |   |   |
S1: G C C _ T C G
```

		0	1	2	3	4	5	6	7	8	9	10	11	12
	H	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	G	0	0	0	0	3	2	0	1	1	2	9	9	5
10	T	0	0	0	3	0	2	1	0	4	4	5	8	8

Fig. 1. Example of a local alignment calculation using a linear gap model and the presented substitution score matrix.

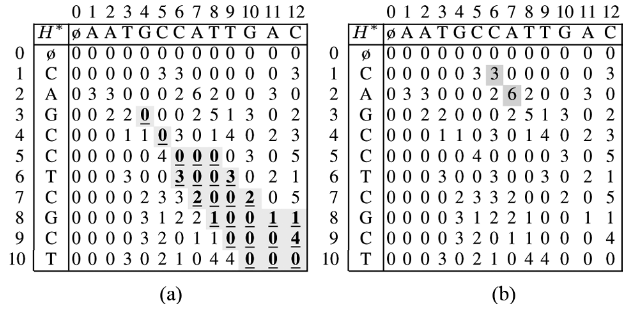
The substitution score value $Sbc(S_1(i), S_2(j))$, typically represented by a score substitution matrix as the example presented in Fig. 1, is usually positive for characters that match and negative for mismatching characters. The gap penalty costs α and β are always positive and $\alpha > \beta$. Different sets of values may be used to reveal different types of alignments.

As soon as the entire score matrix H is filled, the substrings of S_1 and S_2 that best align can be found by first locating the cell with the highest score in H. Then, all matrix cells that lead to this highest score cell are sequentially determined during the *traceback* procedure, as described in [3].

Fig. 1 illustrates a simple and rather small example of the calculated score matrix for aligning two sequences ($S_1 = CAGCCTCGGT$ and $S_2 = AATGCCATTGAC$) using a substitution score function where a match has a score of 3 and a mismatch a score of -1. A linear gap model was used with $\alpha = 4$. The shadowed cells represent the traceback path that was taken in order to determine the best alignment, starting at the cell with the maximum score of 10. If an affine gap model had been adopted, it would be necessary to also build matrices F and E at the same time as matrix H.

B. Determining the n-Best Alignments

Besides finding the optimal alignment between two sequences, it is often useful to find other alignments that still have a significant and approximate score. The W-E algorithm, as described in [5], is an extension of the S-W algorithm which solves the n-best local alignments problem. This algorithm determines the kth-best alignment $(1 < k \leq n)$ by resetting (replacing with value zero) the cells of matrix H that belong to the $(k-1)$th-best alignment. After this replacement, the



(a)

		0	1	2	3	4	5	6	7	8	9	10	11	12
	H^*	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	**0**	0	0	2	5	1	3	0	2
4	C	0	0	0	1	1	**0**	3	0	1	4	0	2	3
5	C	0	0	0	0	0	4	**0**	**0**	**0**	0	3	0	5
6	T	0	0	0	3	0	0	**3**	**0**	**0**	**3**	0	2	1
7	C	0	0	0	0	2	3	3	**2**	**0**	**0**	**2**	0	5
8	G	0	0	0	0	3	1	2	2	**1**	**0**	**0**	**1**	**1**
9	C	0	0	0	0	3	2	0	1	1	**0**	**0**	**0**	**4**
10	T	0	0	0	3	0	2	1	0	4	4	**0**	**0**	**0**

(b)

		0	1	2	3	4	5	6	7	8	9	10	11	12
	H^*	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	0	0	0	2	5	1	3	0	2
4	C	0	0	0	1	1	0	3	0	1	4	0	2	3
5	C	0	0	0	0	0	4	0	0	0	0	3	0	5
6	T	0	0	0	3	0	0	3	0	0	3	0	2	1
7	C	0	0	0	0	2	3	3	2	0	0	2	0	5
8	G	0	0	0	0	3	1	2	2	1	0	0	1	1
9	C	0	0	0	0	3	2	0	1	1	0	0	0	4
10	T	0	0	0	3	0	2	1	0	4	4	0	0	0

Fig. 2. Example of the calculation of the second-best alignment. (a) Recalculated cells. (b) Traceback for the second-best alignment.

neighbor cells are recomputed and a new matrix, H^*, is calculated. For this new matrix, the highest score is determined and the kth-best alignment is obtained using the same traceback technique.

An illustration of this algorithm is shown in Fig. 2. The original H matrix is the one presented in Fig. 1. The algorithm started by replacing with zero all the scores of the cells that composed the optimal alignment, recalculating all the neighboring values that are dependent on these original cells [bold underlined values on gray cells in Fig. 2(a)]. Afterwards, the highest score in the H^* matrix is found and a traceback is performed for this new score, as shown in Fig. 2(b).

This algorithm only requires the recomputation of the neighboring cells of the previously reported alignment, thus being very efficient in terms of time [5]. However, it requires the entire matrix H to be stored in memory ($\mathcal{O}(nm)$ space), thus imposing a large memory overhead.

III. Tracking the Alignment Origin Indexes

As was previously referred, most sequence alignment accelerators that have been proposed until now [14]–[16] only implement the score matrix computation, thus only returning the alignment score. When the alignment is required, the H matrix must be recalculated. Such a task is typically done by a GPP, without reusing any information returned from the accelerator.

However, when the considered sequences have a very dissimilar size ($m \gg n$), the size of the subsequences that participate in the actual alignment is always in the order of n, meaning that a large part of matrix H that is recomputed is not actually needed to obtain the alignment. Moreover, even when sequences of similar size are aligned, a significant part of matrix H is not necessary if the subsequences that actually participate in the local alignment are small.

From these observations, it can be shown that the *time* and *memory space* that are required to find the n-best optimal local alignments during the subsequent traceback phase can be significantly reduced. In fact, if only the cells required to obtain the alignment are recomputed, the associated time will be significantly reduced. Furthermore, when calculating the n-best alignments, which typically requires the entire H matrix to be stored in memory, a significant reduction of space can be achieved if it is possible to constrain the recomputation of the H matrix to only those subsequences that include the n-best alignments.

Hence, in the event 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 $S_1(p')$ and $S_2(q')$, denoted as (p', q'), then the local alignment can be obtained by just considering the score submatrix corresponding to substrings $S'_1 = S_1[p \cdots p']$ and $S'_2 = S_2[q \cdots q']$.

To determine the character position where the alignment starts, a new method is now proposed, denoted as the Alignment Origin Index (AOI) tracking method. This method is based on the computation of an auxiliary coordinate matrix C_b. For simplicity of explanation, a linear gap model (2) will be considered without any loss of generality. Let $C_b(i, j)$ represent the coordinates of the matrix cell where the alignment of strings $S_1[1 \cdots i]$ and $S_2[1 \cdots j]$ starts. Using the same DP method and associated branch conditions that were used to calculate matrix $H(i, j)$, it is possible to simultaneously build matrix C_b, with the same size as H, which maintains a track of the cell that originated the alignment ending at cell (i, j) with score $H(i, j)$. The recursive relations to calculate this matrix are

$$
\begin{aligned}
C_b(i,j) &= \begin{cases} (i,j), & \text{if } H_d \wedge C_{b_d} \\ C_{b_d}, & \text{if } H_d \wedge \text{not} C_{b_d} \\ C_{bF}(i-1,j), & \text{if } H_u, \\ C_{bE}(i,j-1), & \text{if } H_l, \\ (0,0), & \text{if } H(i,j) = 0 \end{cases} \\
C_{bF}(i,j) &= \begin{cases} C_b(i-1,j), & \text{if } F_o, \\ C_{bF}(i-1,j), & \text{if } F_x, \end{cases} \\
C_{bE}(i,j) &= \begin{cases} C_b(i-1,j), & \text{if } E_o, \\ C_{bE}(i-1,j), & \text{if } E_x. \end{cases} \\
H_d &: H(i,j) = H(i-1,j-1) + Sbc(S_1(i), S_2(j)) \\
H_u &: H(i,j) = F(i,j) \\
H_l &: H(i,j) = E(i,j) \\
F_o &: F(i,j) = H(i-1,j) - \alpha \\
F_x &: F(i,j) = F(i-1,j) - \beta \\
E_o &: E(i,j) = H(i,j-1) - \alpha \\
E_x &: E(i,j) = E(i,j-1) - \beta \\
C_{b_d} &: C_b(i-1,j-1) = (0,0)
\end{aligned} \tag{3}
$$

with initial conditions $C_b(i, 0) = C_b(0, j) = (0, 0)$.

Hence, by applying the proposed AOI tracking method and by knowing the cell where the maximum score occurred $(H(p', q'))$, it is possible to determine from $C_b(p', q') = (p, q)$ the coordinates (p, q) of the cell where the alignment began. Subsequently, to obtain the desired alignment, the postprocessing traceback phase (performed by the GPP, as described in Section IV-F) will use the information of the alignment origin coordinates $((p, q))$ and the alignment score value $(H(p', q'))$, both returned by the accelerator, to rebuild the score matrix for the rather small subsequences $S_1[p \cdots p']$ and $S_2[q \cdots q']$, leading to a significant reduction of processing *time* and storage *memory*.

Fig. 3 shows the obtained C_b matrix for the same alignment example illustrated in Fig. 1. By knowing from the computed H matrix that the maximum score occurs at cell $(8, 10)$, it is possible to retrieve the coordinates of the beginning of such alignment in cell $C_b(8, 10) = (3, 4)$ (black cell in Fig. 3). With this

		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	G	(0,0)	(0,0)	(0,0)	(0,0)	(9,4)	(8,4)	(0,0)	(7,5)	(7,6)	(3,4)	(3,4)	(3,4)	(3,4)
10	T	(0,0)	(0,0)	(0,0)	(10,3)	(0,0)	(9,4)	(8,4)	(0,0)	(7,5)	(7,6)	(3,4)	(3,4)	(3,4)

Fig. 3. Example of an AOI tracking matrix.

H	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

Fig. 4. Reduced alignment score matrix.

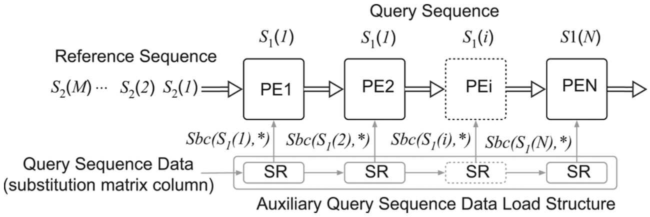


Fig. 5. Systolic array structure for DNA alignment algorithms.

information, the optimal local alignment between S_1 and S_2 can be found by only postprocessing substrings $S_1' = S_1[3 \cdots 8] = GCCTCG$ and $S_2' = S_2[4 \cdots 10] = GCCATTG$ in the subsequent traceback phase. Such alignment (between S_1' and S_2') can now be determined by computing a much smaller H matrix, as shown in Fig. 4.

One key observation on matrix C_b is the *coordinate zone*, which is composed of all of the cells that have the same coordinates of the alignment origin and that reveals the zone of influence of such origin. The *coordinate zone* typically extends in a diagonal way, as it is shown in Fig. 3 by the gray shaded cells. This is an important characteristic that will be further exploited in Section IV-F.

IV. Alignment Core Architecture

Specialized parallel structures capable of performing a great number of simultaneous arithmetic operations are especially suited for accelerating the computation of matrix H. As such, linear systolic arrays with several identical Processing Element (PEs) have proved to be particularly efficient to implement this type of computation [15] (see Fig. 5). Their efficiency is attained by simultaneously computing the values of the H matrix that are located in each anti-diagonal, since these elements do not present data dependencies between each other. In these particular structures, each symbol of the query sequence $(S_1(i))$ is assigned to a single PE in the array (PE_i) whereas each symbol of the reference sequence is streamed through all of the PEs.

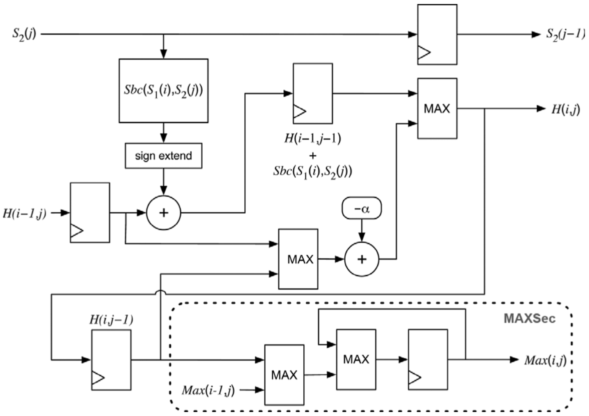


Fig. 6. *Base* architecture of processor element PE_i.

The proposed accelerator architecture is a flexible and configurable structure that can be adapted to a wide range of application scenarios, ranging from aligning short-reads against a reference genome (where $m \gg n$) to gene database searches (where $m \approx n$). It can also be optimized for either affine or linear gap models, as well as for different sequence size requirements. Moreover, to cope with hardware resources restrictions, the computation of matrix H can be partitioned into several parts, allowing a small processing array to compute a large H matrix.

The high flexibility of the proposed accelerator is provided by an important set of distinct configurable options: 1) the number of PEs instantiated in the array, which allows a fine adaptation of the size of the array to better suite the query sequence size; 2) optional inclusion of the n-best calculation unit, to enable the acceleration of the W-E algorithm; and 3) optional inclusion of partition buffers, which allow the alignment of query sequences larger than the number of available PEs.

Both the n-best calculation unit and the partition buffers can be used in conjunction or independently, allowing for a wide range of potential applications for this accelerator.

Consequently, the reconfiguration capabilities offered by an FPGA device make it the most suitable platform to exploit the proposed architecture. However, it is also possible to implement the accelerator as an application-specific integrated circuit (ASIC), specifically tuned for a predefined set of application scenarios.

A. Processing Element

The simplest configuration of the PE's architecture described in this paper is based on the structure described in [15]. This *base* PE (shown in Fig. 6) is composed of a two-stage pipelined datapath capable of calculating a given matrix cell value $H(i,j)$. The throughput of each element is one score value per clock cycle. The computation of each cell $H(i,j)$ requires, among other operations, the evaluation of the substitution score corresponding to the pair of characters under comparison, i.e., the value of $Sbc(S_1(i), S_2(j))$. Since each PE performs the operations over only one single character of S_1, it only needs to store the corresponding column of the substitution score matrix. Such column represents the costs of aligning character $S_1(i)$ to the entire alphabet. The computation of $H(i,j)$ also requires the

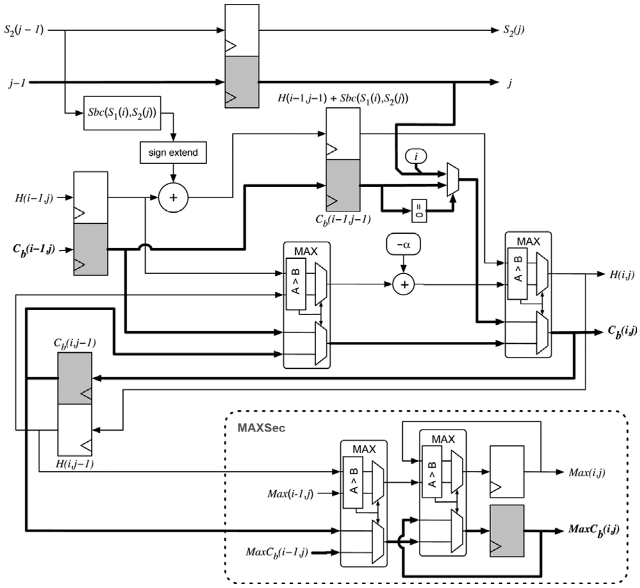


Fig. 7. *Enhanced* architecture of processor element PE_i for linear gap model.

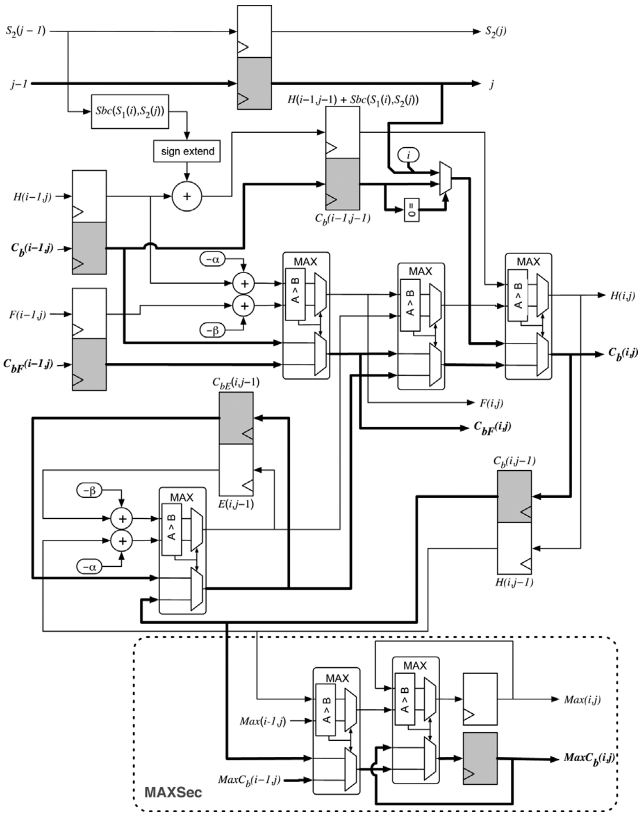


Fig. 8. *Enhanced* architecture of processor element PE_i for affine gap model.

evaluation of the maximum value among the results of the three distinct possibilities presented in (2). The zero condition of the S-W algorithm is implemented by controlling the reset signal of the registers that store $H(i,j)$. Such reset makes use of the sign bit of the evaluated score, i.e., if the maximum value among the three partial scores is negative, then the registers that hold such score are cleared.

Each PE also contains the logic structures required to calculate the absolute maximum score of the entire H matrix. This logic, enclosed in the area denoted as MAXSec in Fig. 6, outputs the maximum of the set of scores that have been calculated by elements PE_1 through PE_i. After all the reference sequence (S_2) characters have passed through all the PEs, the alignment score is available at the output *Max*(i,j) of the last PE.

To support the implementation of the new AOI method that was described in Section III, an *enhanced* architecture of this *base* PE structure was developed. The logic diagram of such *enhanced* PE architecture is presented in Fig. 7 for the particular case of the linear gap model. It features a datapath that implements the whole set of calculations of (2) and (3), in order to propagate, 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). In a similar way, an alternative PE architecture for the affine gap model [(1) and (3)] was also developed and is shown in Fig. 8. This PE architecture features some additional hardware structures to store the values of matrices F and E that are required for the calculations.

The additional hardware that is required to implement (3) (the AOI method) is mainly composed of multiplexers and registers. The control signals of the multiplexers are generated by the magnitude comparators of the MAX units that were already present in the *base* PE architecture. In what concerns the multiplexers input data signals, the origin coordinates corresponding to the score value at *input* $H(i-1,j)$ are present at $C_b(i-1,j)$. Likewise, the origin coordinates corresponding to the score value at *output* $H(i,j)$ are present at $C_b(i,j)$. Finally, the coordinates of the highest score, present at *Max*(i,j), are output at *MaxC*$_b(i,j)$. The coordinates of the current processed cell are obtained by using the hardwired PE index (i) and the symbol coordinate (j) that comes alongside with the reference sequence character, present at input $S_2(j)$.

B. n-Best Calculation

To allow the simultaneous evaluation of the n-best alignment scores using the proposed hardware accelerator (required to speedup the execution of the W-E algorithm), the architecture of an ordering unit responsible for keeping track of the n-best scores is now proposed (henceforward denoted as n-best calculation unit). This additional unit is integrated in the linear systolic array structure and receives, in parallel, all the data values processed by the PEs. The actual value of n is defined by the user and specified at compile time. A typical range would be $4 \le n \le 16$.

For a given sequence pair, the n-best calculation unit determines, alongside with the processing of the H matrix by the accelerator, the n-best local alignment scores which do not share the same *coordinates zone* (do not have the same alignment origin coordinate).

In order to provide for a scalable solution, the ordering unit is subdivided in several subunits that process the data generated by each group of k PEs, as shown in Fig. 9 (typically, $2^2 \le k \le 2^5$). Each of these subunits maintains a list of the n-highest scores and of the corresponding coordinates, calculated up to that instant by the respective PEs, thus working completely independently of the remaining subunits.

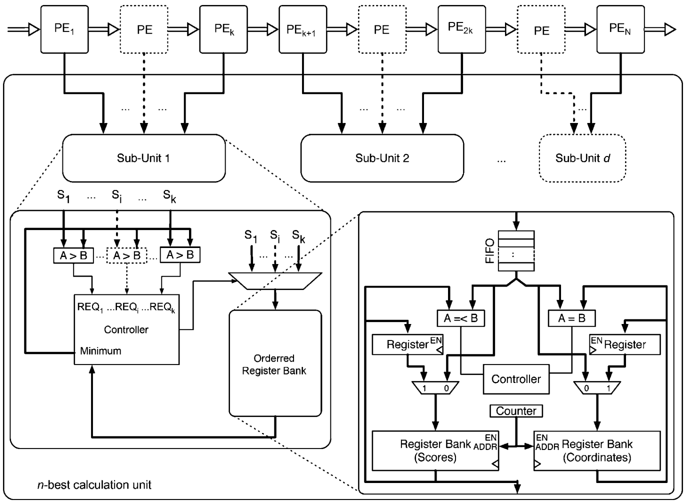


Fig. 9. n-best calculation unit with k grouping.

Within each subunit, the n-highest scores are stored in an *ordered Register Bank*, alongside with the corresponding coordinates (C_b). This bank has n positions and is always ordered by score value. Position 0 holds the highest score value, whereas position $n - 1$ holds the nth highest score. The ordered bank only stores the scores that have distinct origin coordinates. A new data pair (score and origin coordinate) is ordered by sequentially comparing the score with the previous scores already stored at the bank. If it is found to be higher than the score at a given position, the previous data pair is immediately replaced by the new one at that position, forcing a cascading replacement of the remaining data pairs in the bank. A different case occurs if during the comparison process it is found that the new data pair has the same origin coordinates as an already stored data pair. In such a case one of two situations will occur: 1) if the new data pair has a score that is lower or equal than the score of the already stored data pair, it is immediately discarded, since a better alignment starting at the same position has already been found, otherwise 2) the new data pair replaces the stored data pair, since it represents a better alignment starting at the same position. In both cases, the ordering procedure stops at that position, since it is not possible to store two score values with the same associated origin coordinate. Since a single comparison is performed in every clock cycle, this unit guarantees that a new data pair is ordered at most within n clock cycles.

The *ordering requests* (REQ_i) are generated within each subunit by comparing the current score output of each of the k PEs with the smallest of the n-best scores stored at the corresponding subunit. This leads to a reduction in the overall amount of *ordering requests* since only those values greater than the minimum score stored at a given subunit will activate the corresponding REQ_i line. Since each of these subunits receives scores from k PEs $(S_1 \cdots S_k)$, there is a chance that several of the k PEs connected to the same subunit simultaneously generate values that should be included in the n-best list. Therefore, to guarantee that none of the calculated scores are improperly discarded, the processor array is halted if more than one *ordering request* (line REQ_i in Fig. 9) is simultaneously active. The score associated with a given REQ_i line, and the corresponding coordinates, are then stored in a FIFO queue where they are kept until the *ordered bank* is available to process them. This FIFO queue allows the array to resume its operation as soon as the data to be ordered is stored in it. In the event of the FIFO queue is full, the processing of the array is further halted until there are sufficient available positions to store the data corresponding to all of the *ordering requests*. Considering that d subunits are present to process the ordering requests, the array will proceed as soon as all *requests* of all the d subunits are serviced.

It is worth noting that whenever this unit is instantiated to evaluate the n-best alignments, the logic circuit that locally computes the maximum score at each PE (MAXSec) becomes redundant. As a consequence, to maximize the hardware efficiency of this configuration of the accelerator, the MAXSec circuit is not implemented whenever this setup is adopted.

C. Array Programming

As it was previously observed, for a given query sequence each PE only performs comparisons with a single query sequence character. Consequently, the query sequence (S_1) data which has to be loaded into each PE is simply the substitution score matrix column that corresponds to the symbol at that position. Thus, when the accelerator is initialized, the entire substitution score matrix is loaded and stored in a central location. During computation, the relevant query sequence symbols are input and their values are used to index this global substitution score matrix which, in turn, will select the corresponding substitution matrix column to be loaded into the corresponding PE.

The substitution score 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 (shown in Fig. 5), composed by an 8 bit-wide shift register, was included in the array. This structure allows the pre-loading 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. This allows to mask the time that would be required to shift the *new* query sequence data into the array. Therefore, it ends-up by programming the actual query sequence in just one clock cycle, which significantly reduces the amount of time required for programming the array with a *new* query sequence. Furthermore, the use of this shift register also provides a scalable method to program the processor array, as it avoids a common data bus to program the several PEs.

D. Partitioned Processing

In order to make the processing of very long sequences possible (both the query and reference sequences), the proposed accelerator architecture is able to partition the evaluation of ma-

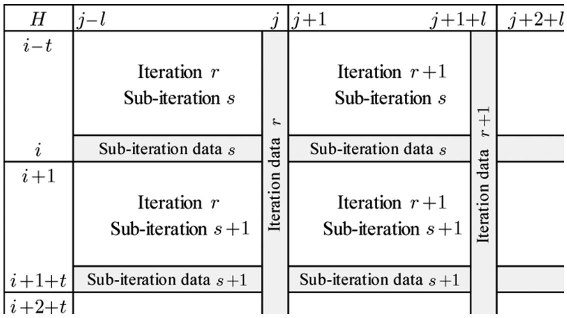


Fig. 10. Partitioned processing with blocks of t rows by l columns.

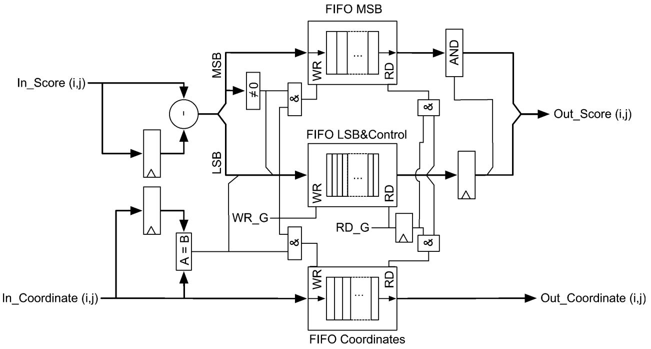


Fig. 11. Architecture of a partition buffer to store a data pair (score and origin coordinates).

trix H. With such procedure, the accelerator divides the H matrix in several submatrices, with the number of rows equal to the number of implemented PEs in the array. The number of columns in such submatrices depends on the amount of available RAM to store the intermediate results.

The processing is performed in several iterations, as shown in Fig. 10. Each iteration is further divided in several subiterations corresponding to the computation of a restricted set of rows, where a section of the query sequence is aligned with a section of the reference sequence. After the processing of a complete iteration, the whole query sequence is aligned to a section of the reference sequence.

To implement this partitioned processing procedure, the architecture must be able to store the intermediate results of the several computations that are undertaken. For each subiteration, it is necessary to store the respective bottom row of the processed submatrix to be used in the following subiteration. Furthermore, the right column values also need to be stored, in order to be used in the next iteration. These values are accomodated in two buffers: the *iteration* and the *subiteration* buffers. Each of these buffers store all the information required to continue the processing. In the particular case of the PEs that implement the affine gap model, the subiteration buffer must store the $H(i,j)$, $C_b(i,j)$, $E(i,j)$ and $C_{bE}(i,j)$ values, while the iteration buffer stores the $H(i,j)$, $C_b(i,j)$, $F(i,j)$ and $C_{bF}(i,j)$ [see (1) and (3)].

The maximum size of the partition buffers is constrained by the amount of memory that is available in the device. Therefore, this constraint will also limit the maximum length of the query sequence, since the contents of an entire column of matrices H, C_b, E, and C_{bE} will have to be stored in the iteration buffer. Furthermore, some memory resources are also required for the subiteration buffer. These do not impose such a hard constraint on the reference sequence length, but if its size is too small, the performance of the array will be affected, since the amount of load operations on the array will be significantly higher.

Hence, since the space that is required to store all of the temporary information is significant, some optimizations were considered in order to achieve some data compression on the stored data. Two important observations are worth noting: 1) the difference between the scores of two adjacent cells is usually small and its absolute value is commonly less than the gap open penalty value (α) and 2) the probability of the coordinates corresponding to two adjacent cells being the same is high. From the first observation, it is possible to design a buffer that stores the difference between consecutively stored score values (ΔS) instead of storing the actual scores. However, considering the fact that this difference is small in terms of its absolute value, it is also possible to distinctively store the least significant bits (LSBs) and the most significant bits (MSBs) of ΔS, since the MSBs have a zero value in most of the cases. The LSBs are always stored in a FIFO queue with a depth equal to the number of storable cell values and a bit-width related to the α value. The MSBs are stored in an independent FIFO queue, with a bit-width corresponding to the difference between the number of bits used in the score calculation and the number of LSBs already stored, when its value is different from zero. The depth of this FIFO is heuristically determined by the user at compile time. A typical value would be one eighth of the depth of the LSBs' FIFO. Similarly, the coordinates values are stored in a FIFO queue, which has a depth that is also configurable by the user at compile time, when they differ from the previously stored value. A typical value would be one quarter of the number of cells to store.

Fig. 11 shows the architecture of the developed partition buffer. This buffer contains the different depth FIFOs (LSBs, MSBs, and Coordinates) as well as a FIFO to store the control signals that ensure that, at the output, the values from the three different depth FIFOs are correctly synchronized. Since the depth of the control signals' FIFO and the LSB's FIFO are equal, they were collapsed into a single FIFO. Hence, according to what was previously described, to perform the partitioned processing it is necessary to store four different data pairs. Therefore, the accelerator structure must include four of these complex partition buffers, as it can be seen in Fig. 12: two for the iteration data ($H(i,j)$, $C_b(i,j)$ and $E(i,j)$, $C_{bE}(i,j)$) and another two for the subiteration data ($H(i,j)$, $C_b(i,j)$ and $F(i,j)$, $C_{bF}(i,j)$).

The number of cells to store in the *iteration* buffer is equal to the maximum query sequence size handled by the accelerator. On the other hand, the number of cells to store in the *subiteration* buffer is dependent on the remaining RAM resources present at the implementation device (e.g., FPGA). However, the higher the number of stored cells in the *subiteration* buffer the better, since this leads to a smaller number of partitions to be processed, which decreases the overall penalty imposed by the switching procedure between the processed submatrices.

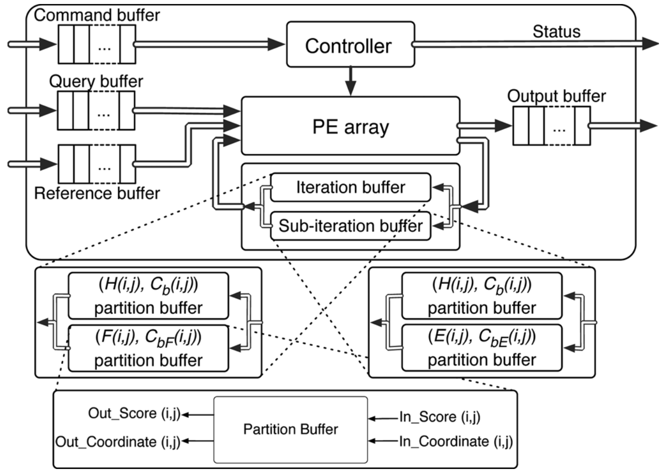


Fig. 12. Architecture and interface of the developed accelerator.

This penalty is related to the fact that the "initial" values for the next submatrix to be processed (stored in the partition buffers) need to be loaded into the array. This is done by shifting in the new values at the same time as the remaining values of the previous submatrix are still being computed and shifted-out of the array. This is accomplished with just one additional multiplexer on specific registers of the PE, to select between the regular value or the "initial" value being shifted in. Although this load operation is being done while the array is processing the remaining data (therefore having a minimal impact on performance), there is still a slight penalty imposed by the array itself for each submatrix to be processed (load operation of the array until all PEs are processing valid data). Therefore, to reduce the number of processed submatrices, the subiteration buffer should be as large as possible.

Since the selected values for the partition buffers' depths are heuristically determined by the user, it is possible that they may not be well dimensioned for all situations and an overfill may occur during processing. In such situation (either due to the score differences being higher than anticipated or to the coordinates changing more than anticipated), the accelerator reports this fact to the controlling host which will proceed with one of two possible options: either align the sequences in the GPP (if there are enough resources to perform such an alignment), or the sequences are placed on hold until a new configuration is loaded into the FPGA, so that the buffers have more appropriate sizes and different tradeoffs between the several parameters of the *iteration* and *subiteration* buffers are selected.

E. Interface

In order to integrate the proposed accelerator with the GPP that will implement the remaining tasks of the alignment procedure (i.e., the traceback), the processing array includes an embedded controller that is responsible for decoding nine distinct instructions. These instructions are used to properly control the array, as well as to receive the data to be processed. The developed interface, illustrated in Fig. 12, is composed of three input FIFO queues (one for the reference sequence, another for the query sequence, and the other for commands), one output FIFO (to return the processed values) and one status register. The three input FIFOs allow an efficient processing of the data, with independent access to the query and reference sequences, as needed by the processing flow in the array.

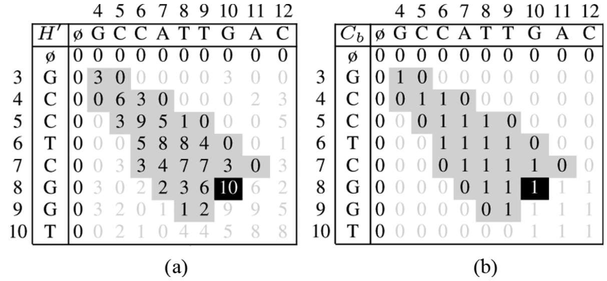
Each of these FIFOs has a depth of 64 words and is 32 b wide, to match the typical bus-width of current GPPs. The status register contains information about the available positions in each of the input FIFOs, to allow the implementation of a flow control mechanism. Complementary information regarding the availability of data in the output FIFO is also provided, indicating when the accelerator has concluded the alignment between the two processed sequences.

F. Postprocessing Operations

The type of information that is provided by the accelerator varies according to the adopted configuration. At its simplest setup, the accelerator returns the best local alignment score and the alignment origin coordinates, whereas at its most complex form it returns several possible (non)optimal local alignment scores and the corresponding origin coordinates. In both cases, a postprocessing step executed in the GPP is needed to obtain the final information: whether it is the optimal local alignment or the n-best local alignments.

With the information provided by an outputted data pair (the alignment score and the corresponding origin coordinate), it is possible to significantly constraint the computation of matrix H in the traceback phase to a much smaller submatrix H' that must be built for the alignment of the subsequences $S_1[p\cdots n]$ and $S_2[q\cdots m]$. Moreover, it can also be shown that, by processing this H' matrix in a wavefront manner (progressively processing the anti-diagonals), it is possible to further constraint the number of cells that actually need to be computed. In fact, by sequentially calculating the anti-diagonals of H', it is possible to halt its processing as soon as the score, calculated at a given cell, equals the alignment score that was previously returned by the accelerator. This will reveal the location of the cell where the alignment ended. With this optimization, the cells that are on the right of the anti-diagonal containing the cell that holds the alignment score value are not computed.

To further reduce the number of computed cells, it is also possible to apply another optimization based on the proposed AOI method. Such optimization takes into account that, when the computation of the H' matrix begins, it is already known that the alignment origin is the cell (1,1) of H' and whichever cells (i,j) participate in the alignment, they will necessarily have the corresponding value $C'_b(i,j) = (1,1)$. The same is true for the remaining cells that belong to the *coordinate zone* of the alignment starting at (1,1). Therefore, for the purpose of computing matrix H', it is sufficient to use a coordinate representation that simply holds a boolean value of 1 or 0 ($C'_b(i,j) = 1$ if the origin is cell (1,1), $C'_b(i,j) = 0$ otherwise). With this method, it is possible to constrain the computation to only those cells that have $C'_b(i,j) = 1$, as well as to those that are immediately adjacent to the right or below. This reduces the number of cells that are computed within each of the anti-diagonals. The proposed optimization is a direct consequence of one of the properties of the adopted coordinates system: if a given cell (u,v) has all of the corresponding coordinates with a null value ($C'_b(u,v) =$



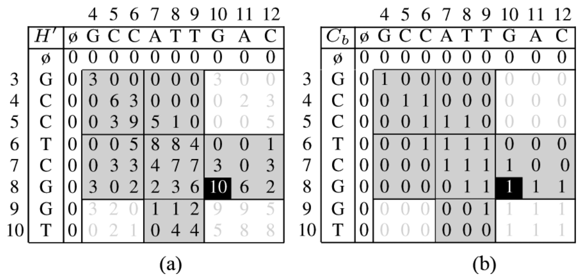
			4	5	6	7	8	9	10	11	12
	H'	ø	G	C	C	A	T	T	G	A	C
	ø	0	0	0	0	0	0	0	0	0	0
3	G	0	3	0	0	0	0	0	3	0	0
4	C	0	0	6	3	0	0	0	0	2	3
5	C	0	0	3	9	5	1	0	0	0	5
6	T	0	0	0	5	8	8	4	0	0	1
7	C	0	0	3	3	4	7	7	3	0	3
8	G	0	3	0	2	2	3	6	10	6	2
9	G	0	3	2	0	1	1	2	9	9	5
10	T	0	0	2	1	0	4	4	5	8	8

(a)

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

(b)

Fig. 13. Example of the postprocessing calculations. (a) Scores matrix H'. (b) Coordinates matrix C'_b.



			4	5	6	7	8	9	10	11	12
	H'	ø	G	C	C	A	T	T	G	A	C
	ø	0	0	0	0	0	0	0	0	0	0
3	G	0	3	0	0	0	0	0	3	0	0
4	C	0	0	6	3	0	0	0	0	2	3
5	C	0	0	3	9	5	1	0	0	0	5
6	T	0	0	0	5	8	8	4	0	0	1
7	C	0	0	3	3	4	7	7	3	0	3
8	G	0	3	0	2	2	3	6	10	6	2
9	G	0	3	2	0	1	1	2	9	9	5
10	T	0	0	2	1	0	4	4	5	8	8

(a)

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

(b)

Fig. 14. Example of the postprocessing calculations when performed in blocks. (a) Scores matrix H'. (b) Coordinates matrix C'_b.

$C'_{bE}(u,v) = C'_{bF}(u,v) = 0$) and all cells in the left column and below have the same null value ($C_b(u+\delta, v-1) = C_{bE}(u+\delta, v-1) = C_{bF}(u+\delta, v-1) = 0$ $(0 \leq \delta \leq n-u)$), then all of the cells below (u,v) (cells $(u+\delta, v)$, with $0 < \delta \leq n-u$) will also have a zero value in the corresponding coordinates. Therefore, all of the cells below (u,v) will not participate in the alignment. The same holds true for each row: if all of the cells in the above row and to the right have the same null value, then all cells to the right of such cell will also have a zero value.

The example presented in Fig. 13 illustrates the matrix H' that is calculated in the postprocessing phase, considering the previously used sample sequences S_1 and S_2. In this example, the subset of cells of the smaller H' matrix whose values are actually calculated in this phase are highlighted with a gray background. For simplicity, the presented example considers that only the best alignment is requested, where the cell computations are performed until the highest score is found. As mentioned before, the cells are processed in an anti-diagonal way starting from the top left cell and, therefore, the processing stops in the anti-diagonal that contains the cell where the alignment score occurs [cell (8,10)]. The simplified matrix C'_b is presented in Fig. 13(b). This auxiliary matrix is used to constrain the computations of H' to those cells that belong to the same *coordinates zone* as the origin cell and is progressively and simultaneously built with matrix H'.

To improve the performance of such postprocessing computation, the GPP memory is allocated in rectangular blocks, each one comprising several columns and rows, instead of cell by cell. To allocate a new block in the vertical (horizontal) direction, it is necessary that one of the columns (rows) encompassed by the current block still requires the computation of additional cells. The example in Fig. 14 shows the previous example matrices but when calculated using 3×3 cell blocks. Note the values of the C'_b cells in the borders of the blocks, which dictate the conditions to not process the adjacent block on the right or below. One exception is observed in the bottom-right block, since the cell with the maximum score [located at (8,10)], and whose score value was determined during the matrix-fill phase (implemented by the accelerator), is used as the stopping condition for this processing step.

In case the n-best calculation unit is used, the information returned by the accelerator comprises $d \times n$ data pairs composed of the alignment score value and the corresponding origin coordinate (where d is the number of ordering subunits present in the hardware accelerator). Each of these data pairs represents a candidate for the set of n-best alignments. Thus, the postprocessing operations on this data start by first ordering the $d \times n$ data pairs by score value, while guaranteeing that only data pairs with distinct origin coordinates are kept. Afterwards, only the data pairs with the n-highest scores are stored in an ordered list that will be considered for the next postprocessing step. However, in this particular case, the processing of matrix H' does not halt when the cell with the alignment score is found, but rather when all the cells in the *coordinate zone* have been recalculated. This means that the calculation of matrix H' continues until all of the newly calculated cells in an anti-diagonal do not belong to the *coordinate zone* of the upper left cell (cells with $C'_b(u,v) = C'_{bE}(u,v) = C'_{bF}(u,v) = 0$). This is done in order to ensure that the results are consistent with the W-E algorithm, which requires the recalculation of the scores in the cells belonging to the same *coordinate zone*.

For each of the data pairs stored in the ordered list, and once the corresponding smaller H' matrix is calculated, the W-E algorithm is iteratively applied to this matrix to obtain the next best alignment occurring within the *coordinate zone* of the initial alignment. If this alignment score is higher than the smallest of the n-best alignment scores determined until that moment, the n-best alignments list is updated. Otherwise, the iterations of the W-E algorithm over the current H' matrix are halted and the processing of the next data pair stored in the ordered list is started.

It can be easily seen that after the execution of these postprocessing operations, the alignment results are consistent with the results produced by the original W-E algorithm, using the whole sequences. In fact, the propagation of the origin coordinates in the proposed accelerator guarantees that the reported alignment scores and corresponding origins are nondependable (a change in a score value of one of the considered alignments will not change any value in the score calculation of the other alignments). However, the W-E algorithm applies to all nonoverlapping alignments and these also include those alignments which are *close* to another alignment and which, in this case, may appear in the same *coordinate zone* of the first alignment. Therefore, to determine possible alignments that could be present in the same *coordinate zone* of a given alignment, the W-E algorithm has to be implemented over the restricted submatrix H' during the postprocessing phase to completely determine the n-best alignments.

TABLE I
CONFIGURATION PARAMETERS OF THE ACCELERATOR PLATFORM

		User-defined values					
		Max RS[1]	Max QS[2]	Gap Penalty Model	Scoring Matrix	Number of scores	FPGA Resources
Inferred Accelerator Parameters	# PEs		×				×
	Score bit-width	×	×		×		
	Coordinates bit-width	×	×				
	PE type			×			
	# n-best scores					×	
	Iteration buffer depth		×				
	Sub-iteration buffer depth						×
Sub-unit grouping size (k)		preset value $k = 8$ (see Section VI-B)					

([1]RS - Reference Size; [2]QS - Query Size)

G. System Configuration

The proposed accelerator has several different configuration parameters, such as the score bit-width and the coordinates bit-width. Although a trained user could directly specify each of the configuration parameters, typically these will be automatically inferred from a restricted set of user-defined values, as shown in Table I. The inferred accelerator parameters are determined by mathematical relations considering the set of user-defined parameters as input variables. These user-defined values are the same that are usually found in popular software frameworks and tools that are commonly used by biologists (e.g., SSEARCH35), therefore being easy for a typical user of this type of software to also use the proposed system.

V. Prototyping Platform

To validate the functionality and to assess the performance of the proposed hardware accelerator in a practical realization, a complete local alignment embedded system based on the described algorithms and architecture was developed and implemented. The base configuration of this system consists of a Leon3 [23] GPP that executes all of the operations of the S-W algorithm, except for those concerning to the demanding 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 consists of a highly configurable and fully synthesizable VHDL core of a RISC architecture conforming to the SPARC v8 definition. It integrates a seven-stage instruction pipeline Harvard micro-architecture, with 32-b internal registers. The adopted Leon3 processor is based on version gpl-1.0.20-b3403 of GRLIB.

The proposed accelerator architecture was interconnected to the GPP as a specialized alignment peripheral, by making use of the AMBA-2.0 APB bus. Besides the proposed DNA alignment accelerator, the GPP core also encompasses two 32-b timers for benchmarking purposes, which were all connected to the AMBA-APB bus.

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 SDRAM memory bank, and several peripherals for control, communication, and storage purposes.

VI. Experimental Results

The presented architecture was fully described using parameterizable VHDL code, synthesized using the Xilinx ISE 10.1 (SP3) software tools, and implemented in the previously described FPGA. This embedded system, used fundamentally as a proof-of-concept prototyping platform, is composed by the Leon3 GPP and the alignment accelerator core including the n-best score calculation unit. The operating frequency of the entire system is limited to 60 MHz, restricted by the adopted Leon3 IP core.

To properly evaluate the proposed system, two distinct application scenarios were considered: 1) a *gene versus gene* database search, where the two sequences to be aligned have similar sizes and 2) a *short-read versus a reference genome*, where the reference sequence is much larger than the query sequences.

The dataset used in the first scenario comprises 43 887 sequences with a length ranging from 21 to 104 026 nucleotides (an average of 1273) used as the reference and 100 sequences with an average length of 988 nucleotides used as the query sequences. For the second scenario, a reference sequence with about 250×10^6 nucleotides was used as the reference and 100 sequences with 35 nucleotides each were used as the query sequences.

All of the implemented configurations that adopted the proposed partitioned processing scheme had the partition buffers size adjusted in such a way that: 1) the number of positions of the LSB&Control FIFOs of the iteration and subiteration buffers is equal to the maximum query size and to the maximum reference iteration size, respectively; 2) the number of positions of the score's MSB FIFO is one eighth of the positions of the LSB&Control FIFO; and 3) the number of positions of the coordinates FIFO is one quarter of the positions of the LSB&Control FIFO. The bit-width of the score's LSBs part is 5 b, considering the adopted score substitution function ($match = 3$, $mismatch = -1$, $\alpha = 4$, $\beta = 3$). The remaining score bits are stored in the MSB FIFO. Furthermore, the maximum reference iteration size was set to 32 768 (2^{15}) positions, since this value is feasible for all of the considered configurations and does not impose a noticeable reduction in the performance of the array.

With the considered configurations, it is possible to align DNA sequences with a wide range of sizes starting from a few nucleotides to several million nucleotides.

A. Processing Efficiency of the PE Array

The maximum attainable throughput of the proposed PE array can be defined as the product of the adopted operating frequency and the number of PEs present in the array. Nevertheless, there are some situations where the maximum performance value may not be achieved: 1) when the query sequence does not make use of the full number of PEs; 2) when the partitioned processing is used, since loading operations into the array need to be performed for each of processed blocks; and 3) when the n-best calculation unit is used, since some wait states may be imposed on the array to guarantee the ordering of the n-best scores. Independently of these situations, the greatest performance level

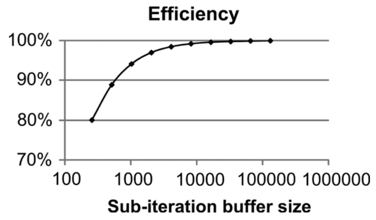


Fig. 15. Array efficiency for several subiteration buffer sizes.

TABLE II
NUMBER OF CLOCK CYCLES NECESSARY TO OBTAIN THE n-BEST ALIGNMENTS (CONSIDERING $k = 8$)

n-best alignments	Average number of clock cycles $(\times 10^3)$	Increase
1	35.6	1.0
2	53.1	1.5
4	109.3	3.1
8	264.8	7.4

is achieved by defining the largest sequence to be aligned as the reference sequence, whereas the shortest sequence should be used as the query sequence. This improves the array efficiency, since less load operations of the array need to be performed.

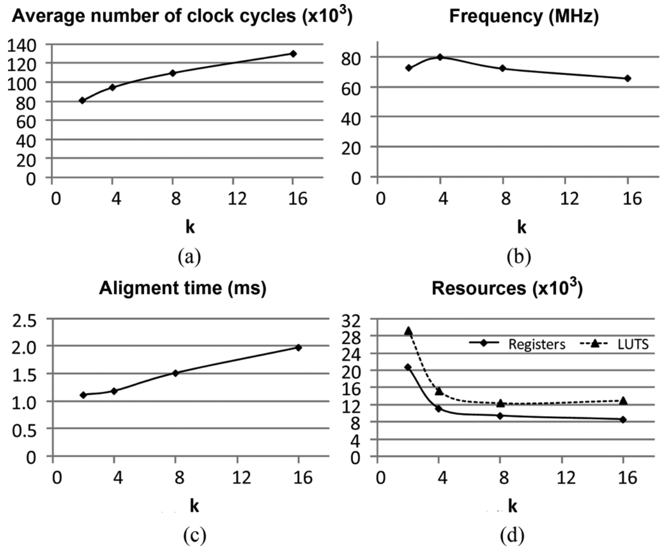
In the partitioned processing scenario, the extra penalty due to the additional load operations is directly related to the size of the *subiteration* buffer, which holds a row of data and limits the maximum length of the reference subsequence that is processed in each iteration. In contrast, and as it was referred to before, the *iteration* buffer size does not influence the array efficiency, since its size only limits the maximum size of the processed query sequence. However, when the hardware resources are limited, a strict balance between the size of the *subiteration* buffer and the size of the *iteration* buffer must be achieved.

The chart in Fig. 15 depicts the efficiency results, measured as the ratio between the actual array throughput and the optimal maximum throughput of the array. These particular results were obtained when aligning a 256-long query sequence with a 10^5–long reference sequence. The exact values may vary according to the sequences that are being aligned, but the trend is the same. As it can be seen, the larger the *subiteration* buffer, the higher the efficiency of the array.

B. Assessment of the n-Best Operation Mode

As it was previously referred, the advantages provided by the newly proposed unit for the computation of the n-best alignment scores arise with an inherent cost of the attainable processing throughput. This is mainly due to the amount of halt cycles that are required in order to execute the simultaneous ordering requests generated by the several PEs connected to the same subunit, as described in Section IV-B. In fact, when the number of best alignments (n) to be evaluated increases, the number of wait states of the array also increases. This is mainly due to a twofold effect: i) the ordering unit will take more time to order each single request and ii) the number of ordering requests will be greater, since the minimum value present at each unit will also be smaller for the same set of sequences.

As an example, Table II presents the variation of the average number of required clock cycles to obtain the corresponding n-best alignments, considering the practical situation of a *gene versus gene database* search scenario. In this situation, the average number of clock cycles to obtain the four best alignments of the considered test sequences increases by a factor of 3. The exact increase rate is dependent on the dataset that is under processing. However, even with such decrease of the array throughput, it was observed that a significant speedup is still attainable by using this accelerator when compared to the LALIGN35 program running on the Intel Core2 Duo processor, as will be seen in Section VI-D.



(a) (b) (c) (d)

Fig. 16. Array performance for several grouping sizes k (considering $n = 4$).

To analyze the impact of the additional configuration parameter k (the number of PEs grouped in each subunit, as referred to in Section IV-B) in the array performance, several configurations were implemented and the average number of clock cycles required to obtain the four best alignments was determined for the considered test sequences. These results, as well as the corresponding values for the maximum operating frequency of the array (without considering the limitation imposed by the Leon3 GPP) and the required hardware resources are presented in Fig. 16.

According to these charts, the grow of k leads to a moderate increment of the amount of clock cycles required to obtain the alignments [Fig. 16(a)], due to the resulting increase of the required wait states. As a collateral effect, the maximum attainable frequency also decreases with the increment of k [Fig. 16(b)]. The combination of these two effects results in the increase of the total processing time required to obtain the alignments, as presented in Fig. 16(c).

On the other hand, it was also observed that the increase of k results in a decrease of the amount of resources used by the accelerator [Fig. 16(d)], since the total number of subunits decreases. Hence, by optimizing the balance between the used resources of the circuit and the corresponding processing time, it was concluded that the configuration corresponding to $k = 8$ yields the best compromise. Note that, for a value of $k = 2$, the significant increase of the amount of used resources and larger interconnections directly impacts the timing performance of the circuit, resulting in a lower maximum frequency than the one obtained with a value of $k = 4$.

TABLE III
FPGA Resource Usage

Config	PE type	# PEs	Score width	Partitioned Processing	Max query size	Max ref size	n-best unit	LUTs	Registers	BRAMs
#1	Leon3 only	0	-	-	-	-	-	17788 (18%)	6246 (6%)	43
#2	Linear	64	11	No	64	65536	No	34197 (35%)	18407 (19%)	46
#3	Affine	64	11	No	64	65536	No	42503 (43%)	22566 (23%)	46
#4	Affine	64	17	Yes	4096	65536	No	72602 (74%)	29315 (30%)	155
#5	Affine	64	17	Yes	4096	65536	n=4	62945 (64%)	25806 (26%)	155
#6	Affine	64	17	Yes	4096	65536	n=8	63777 (65%)	27385 (28%)	155
#7	Linear	128	14	Yes	512	1×10^6	No	57747 (59%)	36085 (37%)	92
#8	Linear	128	13	Yes	512	268×10^6	No	58914 (60%)	35712 (36%)	88

C. Hardware Resource Usage

To demonstrate the flexibility of the proposed alignment platform, several different hardware configurations have been implemented. Table III presents the obtained resource usage results of the considered configurations. A maximum of 128 PEs can be implemented in the adopted FPGA when the affine gap model is considered.

Configuration #1 corresponds to the base setup of the embedded system, where the Leon3 GPP was implemented alone, without the proposed alignment accelerator. The presented results show that this processor alone occupies 18.1% of the available logic resources.

By comparing the results corresponding to configurations 2 and 3, it is observed that the implementation of a processing array using an *affine* gap penalty PE with coordinate tracking support yields a 24% increase of the amount of resources used by the whole system, when compared with a similar *linear* gap penalty PE. This increase of used resources is also dependent on the particular adopted conditions in what concerns the required operating environment, namely, the size of the sequences to be aligned (which determines the bit-width of the coordinates representation) and the substitution score function values (which influence the bit-width of the score calculations).

In what concerns the proposed partition processing scheme, it can be observed, by comparing the BRAMs allocation results of configurations 3 and 4, that this offered feature imposes a significant increase of the number of used memory blocks of this FPGA device, where the required FIFO memories need to be accommodated.

The setups identified as configurations 5 and 6 in Table III correspond to the cases where the n-best alignment facility was assessed. The corresponding ordering unit simultaneously provides the four-best and eight-best alignment scores and has a grouping (k) of 8. By comparing the obtained LUT allocation results with those obtained with configuration 4, it is observed that the amount of required hardware resources slightly reduces. This is due to the fact that, when the n-best unit is used, the MAXSec section present in each PE is discarded (as mentioned in Section IV-B), therefore reducing the overall amount of used resources. However, the absolute amount of used resources imposes a limit on the maximum number of instantiated PEs. For the particular considered cases (with $n = 4$ or $n = 8$) it was decided to fix the number of PEs to the closest lower power of 2, leading to a maximum of 64 PEs.

Finally, when the proposed hardware platform was assessed in what respects the two considered alignment scenarios, it was observed that the most appropriate configuration for the *gene versus gene database* scenario is configuration 5 in Table III. This configuration computes the scores using an affine gap penalty model, includes the n-best calculation unit with $n = 4$ (which is particularly used in this application to find other equal or similar alignments, as well as repetitive DNA regions), and is capable of handling sequence sizes that are common in this particular scenario. When the *short-read versus reference genome* scenario is considered, configuration 8 is regarded as the most appropriate, since it is capable of handling very large reference sequences and does not require the use of the n-best alignment unit, since typically only the position of the best alignment is of interest. This comes with an inherent cost of only handling shorter query sequences. Nevertheless, in this application scenario, the samples are typically smaller than 512 nucleotides, which validate this configuration for this specific scenario.

D. Performance Evaluation

The overall performance of the developed accelerator, together with the proposed AOI method, was assessed using the previously referred test sequences which were aligned using two different approaches: 1) using the developed architecture to accelerate the execution of the S-W and of the W-E algorithms running on the 60-MHz Leon3 GPP and 2) using the SSEARCH35 (for the S-W algorithm) and the LALIGN35 (for the W-E algorithm) software tools from the FASTA framework, by using a 2.4-GHz Intel Core2 Duo processor. Both the SSEARCH35 and the LALIGN35 programs were compiled by considering the SIMD optimizations proposed by Farrar *et al.* [6], as well as with multithread capabilities that are capable of exploiting multicore processors. The SSEARCH35 program was used to align the short-read query sequences with the reference genome, whereas the LALIGN35 program was used to obtain the four best alignments when searching the whole gene database.

The obtained execution times for the two considered scenarios (*gene versus gene database* and *short-read versus reference genome*) are presented in Table IV. The presented results correspond to the computation of complete alignments. In the case of the proposed accelerator, the postprocessing operations in the GPP are performed in parallel, while the accelerator is calculating the scores for the next sequences.

In a preliminary assessment, configuration 7 of the accelerator was applied to align query sequences with 128 nucleotides to a 745 211 nucleotide long reference sequence. The partitioned

TABLE IV
PERFORMANCE RESULTS

	Embedded Comparison	Usage Scenario	
		Gene vs gene database	Short-read vs reference genome
Used Configuration	#7	#5	#8
Pure software alignment time using Leon3 processor	312360ms	-	-
Pure software alignment time using Intel Core 2 Duo processor @ 2.4GHz	-	13968s	4115s
Alignment time using the proposed architecture @ 60MHz	51.7ms	827s	383s
Speedup	6042	17	11

TABLE V
PERFORMANCE COMPARISON

	[14][1]	[15][2]	[16][3]	Proposed[4]
Equivalent Alignment Performance [MCUPS]	812	7600	5400	7605

([1] 140 Linear PEs (Estimated); [2] 168 Affine PEs; [3] 135 Affine PEs; [4] 128 Linear PEs)

processing feature was not used, since all considered query sequences fit in the available PEs. The same alignment task was also executed by using a pure software implementation of the S-W algorithm running in the Leon3 GPP, which performs the whole alignment procedure. The obtained results show that the Leon3 GPP alone takes 312 360 ms to perform the complete alignment (traceback included) which corresponds to an average processing rate of 0.3 million cell updates per second (MCUPS). In contrast, when the proposed accelerator is used together with the Leon3 processor, the same alignment task is performed in just 51.7 ms, which corresponds to a remarkable speedup of 6042. In this assessment, the Leon3 processor was the limiting factor on the attainable performance, while the accelerator alone presented an equivalent performance of 7605 MCUPS. These results are in line with the performance presented by similar architectures proposed in the past [14]–[16] (see Table V). However, it is important to emphasize that 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 AOI method and the conceived n-best alignment option, therefore returning additional information that can be usefully applied to further reduce the computational requirements. Such features are not included in other proposals, being therefore differentiating characteristics of this work and hindering a direct and fair comparison.

In a more challenging comparison, it was determined that the speedup that is achieved by comparing the time required to obtain a whole alignment using the SSEARCH35 and LALIGN35 software implementations running in a Core2 Duo processor and the time required to obtain the same whole alignment with the proposed integrated accelerator architecture. Configuration 5 was adopted for the *gene versus gene database* scenario, while configuration 8 was chosen to implement the *short-read versus reference genome* application scenario. The obtained results reveal that the attained speedup, over a software implementation running on the Core2 Duo, may be as high as 17. These speedup values are the direct consequence of a twofold contribution: on the one hand, the parallelization of the whole matrix fill phase by the processing array, due to the parallel processing in the N PEs; on the other hand, the substantial decrease of the processing time required to perform the *traceback* and the n-best calculation in the GPP, mainly due to the significant reduction of the size of the H' matrix that must be recomputed in this phase. For the entire set of considered test sequences, the usage rate of the partition buffers was always below 75%, therefore demonstrating its adequacy to a large range of application scenarios. For the computation of the optimal local alignments in the *gene versus gene database* scenario, the proposed accelerator presented an equivalent average performance of 6400 MCUPS and a maximum of 7612 MCUPS. These performance metrics contrast with the 750 MCUPS presented, on average, by the Core2 Duo processor to implement the same alignment task.

E. Discussion

As it was shown, the application of the proposed AOI method substantially reduces the computational requirements (both in terms of *memory* and *time*), making it possible to align larger sequences in memory constrained environments as those typically imposed by most current embedded platforms. The proposed accelerator also allows the execution of the W-E algorithm in such embedded platforms, without the need to keep the entire original H matrix in memory, since only the smaller H' matrices are required. Moreover, even thought the conceived embedded platform operates at a clock frequency of 60 MHz (imposed by the adopted GPP), with the consequent advantage of reducing the consumed power, it is still capable to achieve better performance levels than a standard GPP processor (Intel Core 2 Duo processor) operating at a 40 times higher frequency (2.4 GHz).

Although the conducted evaluation was performed in a standard FPGA prototyping board, it is important to note that the proposed architecture could equally be implemented in other more sophisticated platforms. For instance, on the Intel Atom E600C series processor, which includes a GPP and a FPGA interconnected using a PCIe link. Furthermore, the developed accelerator can also be implemented in standard commodity hardware systems. In such a case, it is possible to easily scale the system by adding additional accelerators that work in parallel.

VII. CONCLUSION

An integrated hardware platform for efficient computation of local alignments of DNA sequences was presented. The proposed architecture is based on an innovative strategy that provides a significant reduction of the computational requirements needed by the traceback phase of the S-W algorithm. Such strategy makes use of some information gathered during the computation of the alignment scores by the hardware accelerator, to significantly constrain the size of the subsequences for which it is necessary to recompute the dynamic programming matrix required to perform the traceback phase. Contrasting with previously proposed hardware structures, the presented architecture also allows the simultaneous evaluation of the n-best alignments of a given sequence pair, by incorporating a set of ordering units that are capable of registering the scores and origin coordinates of such alignments. To accommodate the alignment of large sequences, a new buffer architecture for partitioned processing was also implemented, which also provided a significant reduction on the storage space for iterative processing of these larger sequences. Moreover, several different configuration options are provided to allow the usage of the proposed accelerator structure in various different application scenarios.

The developed accelerator was integrated with a Leon3 GPP to form a complete and embedded alignment system that was implemented in a Virtex-4 FPGA. The presented results showed that the developed system is capable of providing speedups up to 17, when compared with the LALIGN35 software tool from the FASTA framework running on an Intel Core2 Duo processor with a 40× higher clock frequency.

REFERENCES

[1] J. Shendure and H. Ji, "Next-generation DNA sequencing," *Nature Biotechnol.*, vol. 26, no. 10, pp. 1135–1145, Oct. 2008.

[2] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers, "Genbank," *Nucleic Acids Res.*, vol. 38, pp. D46–51, Jan. 2010.

[3] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, no. 1, pp. 195–197, 1981.

[4] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," *J. Mol. Biol.*, vol. 215, no. 3, pp. 403–410, 1990.

[5] M. S. Waterman and M. Eggert, "A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons," *J. Mol. Biol.*, vol. 197, no. 4, pp. 723–728, 1987.

[6] M. Farrar, "Striped Smith-Waterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2007.

[7] A. Wirawan, C. Kwoh, N. Hieu, and B. Schmidt, "CBESW: Sequence alignment on the playstation 3," *BMC Bioinformatics*, vol. 9, no. 1, p. 377, 2008.

[8] T. Almeida and N. Roma, "A parallel programming framework for multicore DNA sequence alignment," in *Proc. Int. Conf. Complex, Intell. Software Intensive Syst.*, Feb. 2010, pp. 907–912.

[9] L. Ligowski and W. Rudnicki, "An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, May 2009, pp. 1–8.

[10] A. Khajeh-Saeed, S. Poole, and J. B. Perot, "Acceleration of the Smith Waterman algorithm using single and multiple graphics processors," *J. Comput. Phys.*, vol. 229, no. 11, pp. 4247–4258, 2010.

[11] S. Manavski and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC Bioinformatics*, vol. 9, no. 2, p. S10, 2008.

[12] Y. Liu, D. Maskell, and B. Schmidt, "CUDASW++: Optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units," *BMC Res. Notes*, vol. 2, no. 1, pp. 73–73, 2009.

[13] L. Grate, M. Diekhans, D. Dahle, and R. Hughey, "Sequence analysis with the kestrel SIMD parallel processor," in *Pacific Symp. on Biocomputing*, 2001, pp. 263–274.

[14] L. Hasan, Z. Al-Ars, Z. Nawaz, and K. Bertels, "Hardware implementation of the Smith-Waterman algorithm using recursive variable expansion," in *3rd Int. Design and Test Workshop, IDT 2008*, Dec. 2008, pp. 135–140.

[15] T. Oliver, B. Schmidt, and D. Maskell, "Hyper customized processors for bio-sequence database scanning on FPGAs," in *Proc. 13th Int. Symp. Field-Programmable Gate Arrays*, 2005, pp. 229–237.

[16] K. Benkrid, Y. Liu, and A. Benkrid, "A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 4, pp. 561–570, Apr. 2009.

[17] Z. Nawaz, K. Bertels, and H. E. Sumbul, "Fast Smith-Waterman hardware implementation," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops PhD Forum*, Apr. 2010, pp. 1–4.

[18] CLC Bio, White Paper on CLC Bioinformatics Cube 1.03, CLC Bio, Finlandsgade 10-12—8200 Aarhus N—Denmark 2007.

[19] TimeLogic®, Seqcruncher™ Accelerator Card [Online]. Available: http://www.timelogic.com/seqcruncher.html

[20] S. Lloyd and Q. Snell, "Sequence alignment with traceback on reconfigurable hardware," in *Proc. Int. Conf. Reconfigurable Computing FPGAs*, Dec. 2008, pp. 259–264.

[21] *"Intel®Atom™ Processor E6x5C Series—Product Preview Datasheet,"* Intel Corp., 2010.

[22] K. DeHaven, "White paper: Extensible processing platform," Xilinx Inc., 2010.

[23] A. Gaisler, "SPARC V8 32-bit Processor LEON3/LEON3-FT CompanionCore Data Sheet, Version 1.0.3," 2008.

Nuno Sebastião was born in Lisbon, Portugal, in 1978. He received the M.Sc. degree in electrical and computer engineering from Instituto Superior Técnico (IST), Technical University of Lisbon, Lisbon, Portugal. He is currently working toward the Ph.D. degree in electrical and computer engineering at Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID).

In 2007, he joined the Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID) as a Researcher with the Signal Processing Systems Group (SiPS). His main research interests are focused on dedicated multicore computer architectures and high-performance systems for biological sequence alignment (DNA, RNA, and proteins).

Mr. Sebastião 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 the Instituto Superior Técnico (IST), Technical University of Lisbon, Lisbon, Portugal, in 2008.

He is currently an Assistant Professor with the Department of Computer Science and Engineering, IST and a Senior Researcher with the Signal Processing Systems Group (SiPS), Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID). His research interests include specialized computer architectures for digital signal processing (including biological sequence 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.

Dr. Roma is a member of the IEEE Circuits and Systems Society and the Association for Computing Machinery.

Paulo Flores received the five-year engineering degree, M.Sc., and Ph.D. degrees in electrical and computer engineering from the Instituto Superior Técnico (IST), Technical University of Lisbon, Lisbon, Portugal, in 1989, 1993, and 2001, respectively.

Since 1990, he has been teaching at IST, where he is currently an Assistant Professor with 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 with 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.

Dr. Flores is a member of the IEEE Circuits and Systems Society.