# Integrated Accelerator Architecture for DNA Sequences Alignment with Enhanced Traceback Phase


Nuno Sebastião
*INESC-ID*
*IST-TU Lisbon*
*Portugal*

Tiago Dias
*INESC-ID / IST*
*ISEL-PI Lisbon*
*Portugal*

Nuno Roma
*INESC-ID*
*IST-TU Lisbon*
*Portugal*

Paulo Flores
*INESC-ID*
*IST-TU Lisbon*
*Portugal*

{*Nuno.Sebastiao,Tiago.Dias,Nuno.Roma,Paulo.Flores*}*@inesc-id.pt*



## ABSTRACT

*Dynamic programming algorithms are widely used to find the optimal sequence alignment between any two DNA sequences. This paper presents an innovative technique to significantly reduce the computation time and memory space requirements of the traceback phase of the Smith-Waterman algorithm, together with a flexible and scalable hardware architecture to accelerate the overall procedure. The results obtained from an implementation using a Virtex-4 FPGA showed that the proposed technique is feasible and is able to provide a significant speedup. For the considered test sequences, a speedup of about 6000 was obtained.*



**KEYWORDS:** DNA, Local Sequence Alignment, Hardware Accelerator, Traceback.


## 1. INTRODUCTION

With the most recent advances in sequencing technologies, which allow the determination of the nucleotide sequence of the Deoxyribonucleic Acid (DNA), biologists gained access to enormous amounts of data. In particular, the GenBank [1] database has been doubling its size approximately every 18 months and the version released on December 15th, 2009 had approximately $110 \times 10^9$ base pairs.

Sequence alignment is the method by which useful information is extracted from the large amounts of sequenced DNA. The alignments can be classified as either 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. The local alignment is generally preferred when searching for similarities between distantly related biological sequences, since this type of alignment more closely focuses on the subsequences that were conserved during evolution.

One of the most widely adopted algorithms to find the optimal local alignment between a pair of sequences is the Smith-Waterman (S-W) algorithm [2]. This algorithm is based on a Dynamic Programming (DP) method and is characterized by the smallest runtime among the *optimal* local alignment algorithms, with a time complexity of $O(nm)$, where $n$ and $m$ denote the sizes of the sequences being aligned. The S-W algorithm obtains the alignment in two phases: a DP matrix fill phase and a traceback phase. Alternative *sub-optimal* heuristic algorithms, like BLAST [3], have been proposed to reduce the runtime. However, such speedup comes at the cost of missing the optimal alignments between the sequences. Therefore, the use of the *optimal* alignment algorithms is usually preferred but not always performed due to its significant runtime.

Several approaches have been proposed to accelerate the execution of the S-W algorithm. These solutions range from parallel implementations running in Graphics Processing Units (GPUs) [4] to dedicated hardware architectures. Among the last, the most common are based on systolic arrays, such as the bidimensional structure presented in [5]. Nevertheless, unidimensional (linear) systolic arrays are more commonly adopted [6, 7]. Besides these structures, a commercial solution [8], developed by *CLC bio* and implemented in an Field Programmable Gate Array (FPGA), was also made available.

However, all of these solutions only focus on accelerating the matrix fill phase of the S-W algorithm, disregarding the traceback phase, which is typically performed using a General Purpose Processor (GPP) in a post processing step.

In [9] it was proposed a hardware architecture that also accelerates the traceback phase. Nevertheless, only the global alignment problem is addressed.

To overcome such limitation, this paper presents an innovative and quite efficient technique that makes use of the information gathered during the computation of the local alignment scores (in hardware), in order to significantly reduce the time and memory requirements of the traceback phase, later implemented using a GPP. To support such technique, a new accelerator architecture was developed and integrated with a GPP, to form a complete and quite efficient local alignment system implemented in a FPGA. The presented experimental results show that such new accelerating structure may provide speedups as high as 6000 for the implementation of the whole alignment procedure.

## 2. PAIRWISE LOCAL SEQUENCE ALIGNMENT

Considering any two strings $S_1$ and $S_2$ of an alphabet $\Sigma$ with sizes $n$ and $m$, respectively, a local alignment reveals which pair of substrings of sequences $S_1$ and $S_2$ optimally align, such that no other pairs of substrings have a higher similarity score. A commonly used algorithm to determine the local alignment is the S-W algorithm, with a $O(nm)$ time complexity [2]. This algorithm uses a DP method composed of two main phases: the matrix computation and the traceback.

### 2.1. Smith-Waterman Algorithm

Let $G(i, j)$ represent the best alignment score between a suffix of string $S_1[1..i]$ and a suffix of string $S_2[1..j]$. The S-W algorithm allows the computation of $G(n, m)$ (the local alignment between the two strings) by recursively calculating $G(i, j)$ (the local alignment between prefixes of $S_1$ and $S_2$).

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

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

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

**Table 1. Example of a Substitution Score Matrix**

| $Sbc$ | A | C | G | T |
|---|---|---|---|---|
| A | 3 | -1 | -1 | -1 |
| C | -1 | 3 | -1 | -1 |
| G | -1 | -1 | 3 | -1 |
| T | -1 | -1 | -1 | 3 |

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

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

Table 2 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 1 (a match has a score of 3 and a mismatch a score of -1). The gap penalty has a value of 4. The shadowed cells represent the traceback path (starting at cell $(8, 10)$) that was taken in order to determine the best alignment, which is illustrated in Fig. 1.

**Table 2. Example of an Alignment Score Matrix**

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

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

**Figure 1. Obtained Local Alignment for the Considered Example Sequences**

## 3. TRACKING THE ALIGNMENT ORIGIN AND END INDEXES

As it was previously referred, whenever a sequence pair alignment is required, it is necessary to implement the traceback phase of the S-W algorithm. Most sequence alignment accelerators that have been proposed up until now [5–7] only implement the score matrix computation (without performing the traceback phase). Therefore, only the alignment score is calculated by the accelerator. Afterwards, whenever the alignment score is greater than a given user-defined threshold, the whole $G$ matrix must be recalculated (usually by using a GPP), maintaining enough intermediate data that is required to perform the traceback and retrieve the corresponding alignment. This re-computation does not re-use any data from the previous calculation performed by the accelerator. Such situation can be even aggravated by the fact that typical alignments consider sequences with a quite dissimilar size ($m \gg n$). Therefore, the size of the subsequences that participate in the alignment is always in the order of $n$, meaning that a large part of matrix $G$ that must be completely recomputed in the GPP is not even required to obtain the alignment.

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

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

Hence, with the use of the Alignment Origin and End Indexes (AOEI) tracking technique and by knowing the cell where the maximum score occurred, $G(u, v)$, 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 $S_2[q..v]$,

**Table 3. 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 4. Reduced Alignment Score Matrix**

| $G$ | ø | G | C | C | A | T | T | G |
|---|---|---|---|---|---|---|---|---|
| ø | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 3 |
| C | 0 | 0 | 6 | 3 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 3 | 9 | 5 | 1 | 0 | 0 |
| T | 0 | 0 | 0 | 5 | 8 | 8 | 4 | 0 |
| C | 0 | 0 | 3 | 3 | 4 | 7 | 7 | 3 |
| G | 0 | 3 | 0 | 2 | 2 | 3 | 6 | 10 |

which are usually considerably smaller than the entire $S_1$ and $S_2$ sequences.

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

Hence, the major advantage of this technique outcomes from the fact that the time and memory space required to recompute the $G$ matrix for the subsequences that participate in the alignment is significantly reduced when compared to the entire sequences. As a consequence, this technique also provides a great reduction of the computational effort (time and space) of the whole alignment algorithm.

## 4. ALIGNMENT CORE ARCHITECTURE

The local alignment algorithm described in Section 2 is usually applied to biological sequences in which $m \gg n$ (e.g. $m \approx 10^6$ and $n \approx 10^2$). The matrix fill phase of this algorithm is the most computationally intensive part and is therefore a good candidate for parallelization. However, the data dependencies that exist to calculate each matrix cell highly restrict the parallelization model to the simultaneous

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

computation of the values along the matrix anti-diagonal direction (to calculate the value for cell $G(i,j)$ it is necessary to know the values of $G(i-1,j-1)$, $G(i,j-1)$ and $G(i-1,j)$).

Specialized parallel hardware that is capable of performing a great number of simultaneous arithmetic operations is especially suited for this task. Linear systolic arrays with several identical Processing Elements (PEs), as shown in Fig. 2, have proved to be efficient structures to implement this type of computation, by simultaneously computing the values of the $G$ matrix that are located in a given anti-diagonal [6].

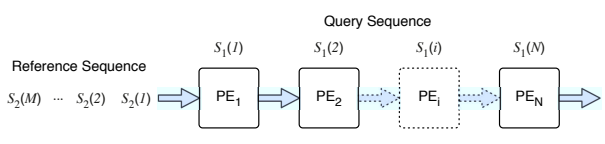


**Figure 2. Systolic Array Structure for DNA Algorithms**

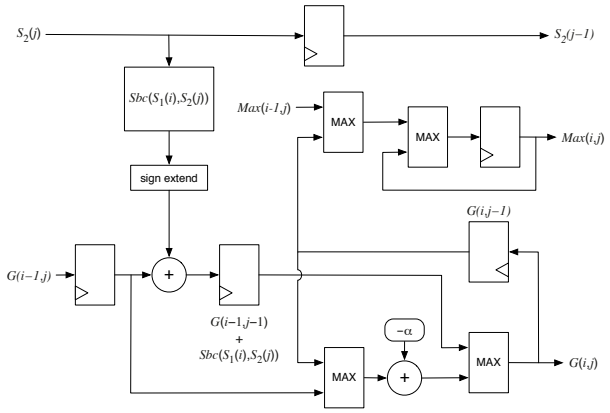


**Figure 3. *Base* PE Architecture**

### 4.1. Processing Element

The PE's architecture described in this paper is based on the PE structure described in [6]. This *base* PE only implements the basic score matrix calculation and is shown in Fig. 3. It has a two stage pipelined datapath to calculate a score matrix cell value (output in $G(i,j)$). The throughput of each element is one score value per clock cycle. Since the S-W algorithm requires the evaluation of the maximum score value throughout the entire matrix, it is necessary to have an additional datapath that selects the maximum score that has been calculated in the array (output $Max(i,j)$). Hence, $PE_i$ 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. In this array, character $S_1(i)$ is allocated to the $i$th PE and this PE performs, at every clock cycle, the computations required to determine the score value of a certain matrix cell. This computation requires, among other operations, the selection of the substitution score between the two characters, i.e. the value of $Sbc(S_1(i), S_2(j))$. Since each PE performs the operations only over one single character of $S_1$, it only needs to store the column of the substitution cost matrix that represents the costs of aligning character $S_1(i)$ to the entire alphabet. The computation of the matrix cell value $G(i,j)$ also requires the evaluation of the maximum value among the results of the three distinct possibilities presented in Eq. 1. The zero condition of the S-W algorithm is implemented by controlling the reset signal of the registers that store the $G(i,j)$ value. Such reset makes use of the sign bit of the score value, i.e., if the maximum value among the three partial scores is negative, then the registers that hold that score are cleared. After all the reference sequence ($S_2$) characters have passed through all the PEs, the alignment score is available at the $Max(i,j)$ output of the last PE.

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

Each of the *enhanced* PEs that form the array, whose architecture is presented in Fig. 4, features a datapath that implements both the calculations of Eq. 1 and 2. The additional hardware required to implement Eq. 2 (the AOEI

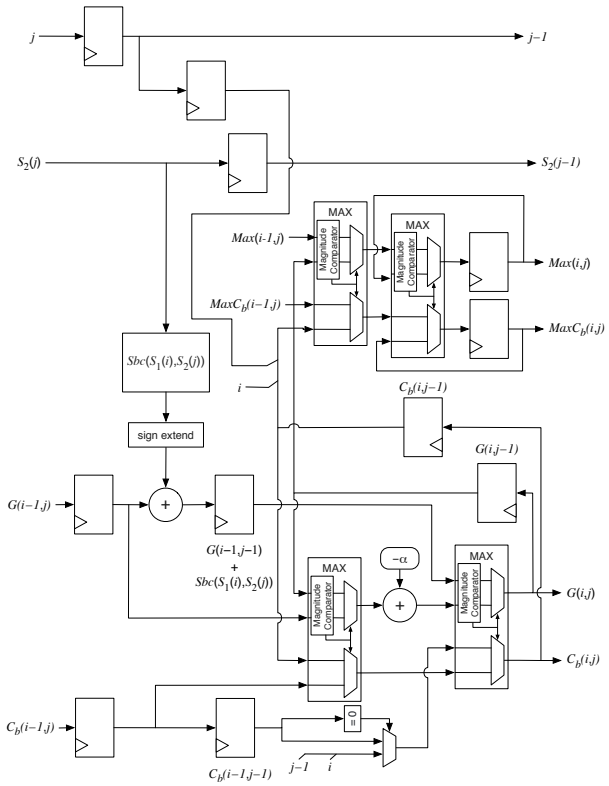


**Figure 4. *Enhanced* PE Architecture**

technique) is mainly composed of multiplexers and registers. The signals that control the new multiplexers required by the AOEI technique are generated by the magnitude comparators that are integrated in the $Max$ units and that were already present in the *base* PE architecture. 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 $C_b(i,j)$. Moreover, the coordinates of the highest score, present at $Max(i,j)$, are output at $MaxC_b(i,j)$. The coordinates of the currently processed 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 $S_2(j)$.

## 4.2. Array Programming

Since each PE only performs comparisons to a given query sequence character, it will just access the values present at the corresponding substitution matrix column. 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 within each PE using dedicated registers since this allows for a fast reprogramming of the PEs for a new query sequence. In the event of a PE not being used (because the query sequence has a smaller size than the number of PEs ($N$)), the substitution score data that is stored in such PE corresponds to a substitution matrix column in which every value is zero.

To program the query sequence ($S_1$) score values, an auxiliary structure was included in the array. This structure is composed by a $n$ bit-width shift register that allows to shift the values of a substitution matrix column through the several PEs. This approach provides the load of a *new* query sequence into this temporary storage shift register, by serially shifting the substitution matrix column data while the array is still processing the data regarding the *current* query sequence. As soon as the array has finished processing the data regarding the *current* query sequence, the *new* query sequence data, which is 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 and therefore significantly reduces the amount of time required for programming the array with the *new* query sequence. Furthermore, the use of this shift register provides a scalable method to program the processor array, as it avoids the use of a common bus to program the several PEs.

## 4.3. Interface

In order to integrate the proposed accelerator with the GPP that will implement the remaining alignment procedure (i.e. the traceback), the systolic array includes an embedded controller that is responsible for decoding 7 instructions (required to properly control the array), as well as to receive the data to be processed. The developed interface, shown in Fig. 5, is composed of two input FIFOs (one for the reference sequence and the other for commands and query sequence), one output FIFO (to return the processed values) and one status register. The two input FIFOs allow the next query sequence to be loaded into the array in parallel with the processing of the current alignment, without increasing the complexity of the control that would arise from having all of the data (query and reference sequences data) input through the same FIFO.

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

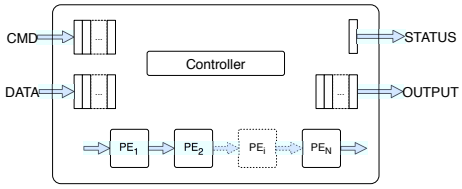


**Figure 5. Accelerator Interface**

## 5. 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 base configuration of this system consists of a Leon3 processor [10] that executes all operations of the S-W algorithm, except for the ones 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 implemented local alignment system consists of a software implementation of the S-W algorithm, specially developed in the scope of this research work. Such algorithm implementation includes some optimizations, in order to achieve more efficient applications in embedded systems. In particular, memory accesses were optimized by using a static memory allocation mechanism. Special attention was also devoted to the data transfers of both the reference and query sequences from the Leon3 processor to the proposed hardware accelerator, so that a high level of efficiency is achieved.

### 5.1. Leon3 Processor

The Leon3 processor [10] is one of the most used free processor cores available today. It has been specifically designed for embedded applications by the European Space Agency, although nowadays it is maintained by Gaisler Research. It consists of a highly configurable and fully synthesizable core, written in VHDL, implementing a RISC architecture conforming to the SPARC v8 definition.

The Leon3 32-bit core is based on a 7-stage instruction pipeline Harvard micro-architecture with 32-bit internal registers. The core functionality can be easily extended by means of the AMBA-2.0 AHB/APB on-chip buses. The AMBA-2.0 AHB bus is used to connect the Leon3 processor with high-speed controllers, e.g. the cache and the memory controller. On the other hand, the AMBA-2.0 APB is used to access most on-chip peripherals and is connected to the Leon3 processor via the AHB/APB Bridge. Finally, external memory access and memory mapped I/O operation are provided by a programmable memory controller with interfaces to PROM, SRAM and SDRAM chips.

### 5.2. DNA Alignment Peripheral

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

The developed alignment peripheral consists of the proposed hardware accelerator, as shown in Fig. 5, and of some additional circuitry responsible for its adaptation to the AMBA-2.0 APB bus. The extra circuitry required to implement the AMBA-2.0 APB wrapper consists mostly of some multiplexers, decoders and a simple control unit that implements the bus protocol. The I/O FIFOs and the status register of the alignment core are mapped in the Leon3 memory address space. Using such interface, the write and read operations over this peripheral can be easily implemented using plain load and store operations.

### 5.3. FPGA Implementation

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

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

## 6. RESULTS

The previously presented architecture, described using parameterizable VHDL code, was synthesized using the Xilinx ISE 10.1 with SP3 software and implemented in the previously described FPGA. This system is composed by the Leon3 GPP and the alignment accelerator core with an array

composed of 128 PEs. The maximum operating frequency of the entire system is 60MHz. This frequency is a limitation imposed by the Leon3 core. The maximum operating frequency of the accelerator core itself is 120MHz.

The obtained resource usage results of the various considered configurations are presented in Table 5. The resources occupied by the Leon3 processor are presented (Leon3 only) for reference. These results show that the Leon3 processor alone occupies 18.1% of the available logic resources. In what concerns the resource usage of the systolic array using the *enhanced* PEs, it is possible to determine that it is 77% larger in relation to the *base* configuration without the index tracking functionality. The exact increase of the amount of used resources depends on the conditions that will be selected according to the required operating environment, namely, the size of the sequences to be aligned (which determines the bit-width of the coordinate representation) and the adopted scoring scheme (which influences the bit-width of the score calculations).

To test the proposed system, a set of real DNA sequences were used as the reference sequence. The size of these sequences range from about $17 \times 10^3$ to $2.6 \times 10^6$ nucleotides. The maximum query sequence size is limited by the number of available PEs in the array. Consequently, in the case of the implemented system, it must be less than or equal to 128 nucleotides long (a size compatible with the latest Next-Generation Sequencing technologies [11]). The chosen query sequence is 128 nucleotides long. If larger query sequences are required, the number of PEs in the array can be increased and, if necessary, the array can be expanded by connecting another FPGA.

The overall performance of the proposed AOEI technique, together with the developed accelerator, was assessed using the previously selected sequences which were aligned using two different methods: *i)* the alignment between each sequence pair is obtained using a pure and straight-forward implementation of the S-W algorithm running exclusively on the Leon3 processor, which keeps the entire score matrix in memory, and *ii)* the alignment is obtained using the developed accelerator and the Leon3 processor. The obtained execution time results for both of these methods are presented in Table 6. The results presented for the proposed accelerator include the communication overhead of the data and code between the GPP and the accelerator. The obtained speedup was determined by comparing the time required to obtain a whole alignment using the *software only* implementation of the S-W algorithm and the time required to obtain the whole alignment with the aid of the proposed technique together with the corresponding accelerator architecture.

**Table 5. FPGA Resource Usage**

| PE | | Score | Maximum Size | | Resource Usage | |
|---|---|---|---|---|---|---|
| Type | # | width | Reference | Query | Registers | LUTs |
| Leon3 | 0 | - | - | - | 6246 | 17788 |
| Base | 16 | 7 | - | 16 | 7441 | 19818 |
| Base | 128 | 10 | - | 128 | 16031 | 34130 |
| Enh. | 16 | 7 | $2^{16}$ | 16 | 9499 | 22168 |
| Enh. | 128 | 10 | $2^{22}$ | 128 | 40024 | 56541 |

When the proposed acceleration is applied, the matrix fill phase of the S-W algorithm is performed using the implemented systolic array. This array returns not only the maximum score but also the coordinates of the cell where it occurred as well as the coordinates of the cell where the alignment begins. Afterwards, a pure and straight-forward implementation of the S-W algorithm is executed in the Leon3 to obtain the alignment between the subsequences that participate in the local alignment (*Reduced matrix fill* and *Reduced traceback*), which were determined by the coordinate information returned from the accelerator. With this new strategy, the matrix fill phase that is executed in the Leon3 processor (required to extract the alignment) is significantly reduced, due to the quite smaller sequences involved. Therefore, the computational requirements, both in terms of memory and time, are substantially reduced.

The obtained results show that the attained speedup may be as high as 6042. These achieved speedups are the consequence of a two-fold contribution: on the one hand, the parallelization of the whole matrix fill phase by the systolic array; on the other hand, the reduction of the processing time required to perform the traceback in the GPP, due to the reduction of the size of the $G$ matrix that must be recomputed in this phase. Regarding to the $G$ matrix fill phase, it is worth noting that its time complexity when executed in the Leon3 processor is $O(nm)$, while the time complexity when executed in the accelerator is reduced to $O(m)$, due to the parallel processing in the $n$ PEs. Therefore, a significant speedup is attained in determining the local alignment score.

In what concerns the traceback phase, the time complexity is the same on both cases ($O(n + m)$). However, in order for it to be performed, it is necessary to recompute the $G$ matrix in the GPP. The computation time involved in this re-computation is significantly reduced when the proposed AOEI technique is adopted. As an example, and considering the alignment of the 128 nucleotide query sequence with the 1311701 reference sequence, the obtained local alignment spans over a 124 nucleotide long subsequence of the reference sequence and over a 123 nucleotide subsequence of the query sequence. The size of the *entire* $G$ matrix that would have to be recomputed to obtain the alignment has approximately $168 \times 10^6$ cells. However, with the proposed

**Table 6. Alignment Results for the System when using an Array with 128 PEs and a Query of 128 nucleotides**

| Reference size (nucleotides) | Processing time using only the Leon3 processor (ms) | | | Processing time using the Leon3 processor and the proposed accelerator (ms) | | | | Speedup |
|---|---|---|---|---|---|---|---|---|
| | Matrix fill | Traceback | Total | Score and coordinate (accelerator) | Reduced matrix fill (Leon3) | Reduced traceback (Leon3) | Total | |
| 17878 | 7493.6 | 0.9 | 7494.5 | 0.7 | 45.7 | 0.9 | 46.6 | 161 |
| 83648 | 35049.2 | 0.9 | 35050.1 | 3.2 | 54.4 | 1.0 | 55.4 | 633 |
| 136980 | 57390.4 | 1.0 | 57391.4 | 5.2 | 54.4 | 1.0 | 55.4 | 1036 |
| 295301 | 123757.5 | 0.9 | 123758.4 | 11.1 | 49.5 | 1.0 | 50.5 | 2451 |
| 566490 | 237377.8 | 0.9 | 237378.8 | 21.3 | 49.5 | 0.9 | 50.5 | 4701 |
| 745211 | 312359.1 | 1.0 | 312360.0 | 28.0 | 50.7 | 1.0 | 51.7 | 6042 |
| 1311701 | - | - | - | 49.3 | 50.9 | 1.0 | 51.9 | - |
| 2623402 | - | - | - | 98.5 | 50.8 | 1.0 | 51.8 | - |

AOEI technique, the size of the $G$ matrix that needs to be recomputed in the GPP is reduced to $124 \times 123 \approx 16 \times 10^3$.

For larger sequences, the speedup would be even higher, since the reduction in size of the recomputed $G$ matrix would be even more significant. Furthermore, the proposed technique not only allows to significantly reduce the time required to obtain the alignment but it is also capable of processing larger sequences (e.g. the 2623402 nucleotide long reference sequence, whose memory requirements prevent it from being aligned on the GPP) as it significantly reduces the amount of memory used by the GPP.

## 7. CONCLUSIONS

An innovative method to significantly reduce the computational requirements of the traceback phase that is executed by the widely used Smith-Waterman algorithm for local alignment of DNA sequences is presented. Based on this technique, a new hardware accelerator architecture was developed and integrated with a Leon3 general purpose processor. Such developed embedded system for DNA alignment was implemented in a Virtex-4 FPGA. The obtained results demonstrate that this system, based on the proposed technique, offers global speedups as high as 6042, when compared to the pure software version of the Smith-Waterman algorithm running on the Leon3 processor. Furthermore, it was shown that the use of the proposed accelerator enables the alignment of larger sequences, even in a memory restricted environment.

## ACKNOWLEDGMENTS

This work has been partially supported by the PhD grant with reference SFRH/BD/43497/2008 provided by the Portuguese Foundation for Science and Technology (FCT).

## REFERENCES

[1] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers, "GenBank," *Nucleic Acids Res.*, vol. 38, no. suppl_1, pp. D46–51, Jan. 2010.

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

[3] 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.

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

[5] 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.

[6] 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, FPGA'05*. ACM, 2005, pp. 229–237.

[7] 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.

[8] CLC Bio, "White paper on CLC Bioinformatics Cube 1.03," CLC Bio, Finlandsgade 10-12 - 8200 Aarhus N - Denmark, Tech. Rep., May 2007.

[9] S. Lloyd and Q. Snell, "Sequence alignment with traceback on reconfigurable hardware," in *Int. Conf. Reconfigurable Computing and FPGAs - ReConFig '08.*, Dec. 2008, pp. 259–264.

[10] Aeroflex Gaisler, *SPARC V8 32-bit Processor LEON3 / LEON3-FT CompanionCore Data Sheet, Version 1.0.3*, Dec. 2008.

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