Scalable Accelerator Architecture for Local Alignment of DNA Sequences

Nuno Sebastião, Nuno Roma, Paulo Flores
INESC-ID / IST-TU Lisbon
Rua Alves Redol, 9, Lisboa
PORTUGAL
{*Nuno.Sebastiao, Nuno.Roma, Paulo.Flores*} *@inesc-id.pt*

Abstract

The Smith-Waterman algorithm is widely used to determine the optimal sequence alignment between two DNA sequences. This paper presents an innovative method to significantly reduce the computation time and memory space requirements of the traceback phase of this alignment algorithm. It also presents a flexible and scalable hardware architecture for accelerating such method, which can be easily expandable by the interconnection of several FPGAs. The results obtained from an implementation using a Virtex-5 FPGA showed that the proposed method is highly feasible in order to provide significant gains in terms of the overall performance of the whole alignment procedure when long sequences are processed. The obtained results also showed that it is preferable to span the array of processing elements through several FPGAs, rather than reusing the hardware resources of the individual array.

Keywords DNA; Local Sequence Alignment; Hardware Accelerator; FPGA

1. Introduction

With the recent advances in sequencing technologies, which allow the determination of the nucleotide sequence of the Deoxyribonucleic Acid (DNA), biologists gained access to an enormous amount of data. However, the DNA sequence size of most living cells can be quite large. For example, the size of the human DNA can be as large as 3×10^9 base pairs. This means that for each complete human individual genome that is sequenced, an additional dataset of 3×10^9 base pairs will be available for researchers. Such datasets are usually stored in databases to which biologists submit the newly sequenced DNA segments. One of these well known public databases is the GenBank [1]. The size of this database has doubled approximately every 18 months and the version released on June 15th, 2009, had approximately 105×10^9 base pairs.

The information contained in the DNA sequences is mainly extracted by homology, therefore requiring a large number of comparisons between sequences. However, exact search of a given sequence in the whole sequences database is often unfeasible due to the frequent mutations to which DNA is affected (nucleotide insertion, deletion and substitution). To overcome this complication, several techniques have been devised to find the optimal position where as many as possible nucleotides are found in the same positions. These methods, denoted by sequence alignment algorithms, are used to determine which sequences match more closely and how they align in order to show the zones that are common.

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 in the alignment. 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.

The computational effort to perform such tasks in such a large dataset poses considerable challenges. The Dynamic Programming (DP) algorithm to find the optimal local sequence alignment between any two sequences has $O(nm)$ time complexity, where n and m denote the sizes of the sequences being aligned. Alternative sub-optimal heuristic algorithms, like BLAST, have been proposed to reduce the runtime. However, they may miss the optimal alignments between the sequences. Therefore, the use of the optimal alignment algorithms is usually preferred but not always performed due to the excessive runtime.

The use of hardware accelerators based on Field Programmable Gate Arrays (FPGAs) for High Performance Computing has been increasing over the past few years. Several algorithms have been accelerated with specialized architectures that were implemented in these devices. One of such algorithms is the Smith-Waterman (SW) algorithm [2], which uses DP to determine the optimal local alignment between any two sequences with $O(nm)$ complexity.

Several accelerator architectures have been proposed to implement the Smith-Waterman algorithm in FPGAs [3]. The most common architecture is based on a systolic array of Processing Elements (PEs). An example of a bidimensional systolic array, described using VHDL, is presented in [4]. Nevertheless, unidimensional (linear) systolic arrays are more commonly adopted [5, 6]. Some of these

accelerators can take advantage of the reconfiguration capabilities provided by FPGAs to optimize the PEs to the particular conditions of a given alignment [5]. Another implementation, which is available as a commercial solution, was developed by *CLC bio* [7]. The offered product also makes use of a FPGA to accelerate the matrix fill stage of the Smith-Waterman algorithm.

However, most of these accelerators have only focused on the part of the algorithm that calculates the alignment score. The alignment, itself, is usually obtained in a post-processing stage (usually implemented in a general purpose processor) where the scores are recalculated for the highest scoring sequences, by saving additional information that is required to retrieve the best alignment. In this paper, a new and more efficient method is proposed that makes use of the information obtained during the calculation of the alignment scores (in hardware), in order to reduce the time required to determine the alignment. To implement such technique, a scalable architecture that also enables the interconnection of several accelerators is presented and implemented in FPGA, thus allowing the use of a larger number of processing elements to permit a higher throughput.

This paper is organized as follows: In Section 2 it is presented the SW algorithm, which is used to determine the optimal local alignment. Section 3 presents the architecture used to accelerate the local alignment procedure. Section 4 shows the obtained results in an FPGA. The conclusions are presented in Section 5.

2. Local alignment

Considering two strings S_1 and S_2 of an alphabet Σ 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 SW algorithm, which has a $O(nm)$ time complexity [2]. This algorithm uses a DP method composed of three essential parts: the recurrence relation, the matrix computation and the traceback [8].

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 SW 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 relations to calculate the local alignment score $G(i,j)$ are given by Equation 1

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

The $Sbc(S_1(i),S_2(j))$ function denotes the value obtained by aligning character $S_1(i)$ against character $S_2(j)$. This value represents the substitution score. The α value represents the gap penalty cost (the cost of aligning a character to a space). An example of a substitution function is shown in Table 1.

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. On the contrary, mismatching characters may have either positive and negative scores, according to the type of alignment that is being performed, denoting the biological proximity between the two. In contrast, the gap penalty cost α is always positive. Different substitution score matrices are used to reveal different alignments. The particular score values defined in these matrices are determined by biologists according to evolutionary relations.

The initial conditions for the calculation are the following:

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

After filling the entire matrix G, the substrings of S_1 and S_2 that best align are 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. The traceback procedure ends when a cell with a score of zero is reached. Such traceback identifies the 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 based on the recurrence equations (eq. 1). When the neighbor is the left cell ($G(i,j-1)$) then this corresponds to inserting a space (opening a gap) in S_1 at position i. If it was the top cell ($G(i-1,j)$), then this corresponds to inserting a space (opening a gap) in S_2 at position j. When the neighbor is the top-left cell ($G(i-1,j-1)$) then this corresponds either to a match or to a substitution. The traceback phase has a $O(n+m)$ time complexity.

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, where a match has a score of 3 and a mismatch a score of -1. A gap has a penalty of 4. The shadowed cells in the table represent the traceback path that was taken to determine the best alignment, which is shown in Figure 1.

978-972-789-304-1 REC'2010

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 alignment.

2.2. Tracking of the Origin and End alignment indexes

When only the alignment score is required, it is not necessary to perform the traceback phase of the SW algorithm. However, whenever the alignment between the sequences must also be determined, the traceback phase must be implemented. However, most hardware accelerators that have been proposed for the alignment algorithms only implement the 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 threshold, the whole G matrix is recalculated (usually by using a general purpose processor) maintaining enough intermediate data to perform the traceback and retrieve the corresponding alignment. Hence, the recalculation of the entire G matrix is performed outside the accelerator without keeping any data from the previously calculated matrix score.

However, it can be shown that the time and memory space that is required to find the local alignment can be significantly reduced. In fact, and considering a given pair of sequences S_1 and S_2, if it is possible to know that the local alignment starts in characters at position $S_1(p)$ and $S_2(q)$ represented as (p,q) and ends in characters at position $S_1(u)$ and $S_1(v)$ represented as (u,v), then the local alignment can be obtained by just recalculating the alignment between the subsequences $S_a = S_1[p..u]$ and $S_b = S_2[q..v]$.

As an example, from the data shown in Table 2, it is possible to determine that the alignment starts in characters at position $(3,4)$ and ends in the characters at position $(8,10)$. With this information, the optimal local alignment between S_1 and S_2 can be found by only calculating the alignment between subsequences $S_a = S_1[3..8] = GCCTCG$ and $S_b = S_2[4..10] = GCCATTG$. The alignment between S_a and S_b can now be determined by computing a much smaller G matrix and performing the traceback, as shown in Table 3.

Table 3: Reduced alignment score matrix.

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

Hence, the advantage of this method resides in the fact that the time and memory space required to recompute the G matrix for the subsequences that participate in the alignment is usually significantly reduced when compared to the entire sequences. Consequently, this method also reduces the computational effort of the alignment algorithm.

To determine the character positions where the alignment starts an auxiliary matrix, C_b, will be used. Let $C_b(i,j)$ represent the coordinates of the matrix cell where the alignment of string $S_1[1..i]$ and string $S_2[1..j]$ starts. Using the DP method that is used to calculate $G(i,j)$, it is possible to simultaneously build a matrix C_b, with the same size as G, that maintains a track of which cell originated the score that reached cell (i,j).

The recursive relations that determine the coordinates of the matrix cell that originated the alignment ending at cell (i,j) are given by Equation 2.

The initial conditions for the calculation are:

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

With this method, it is possible to find, at cell $C_b(i,j)$, the coordinates of the cell where the alignment ending at cell $G(i,j)$ was originated. Afterwards, 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. Then, to obtain the desired alignment, the score matrix has to be rebuilt only for the subsequences $S_1[p..u]$ and $S_2[q..v]$ which are usually considerably smaller than the entire S_1 and S_2 sequences.

An example of table C_b for the alignment of sequences S_1 and S_2, whose G matrix was presented in Table 2, is shown in Table 4. 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 $C_b(8,10) = (3,4)$.

3. Architecture

The local alignment algorithm described in Section 2 is usually applied to biological sequences in which $m \gg n$ (e.g. $n \approx 500$ and $m \approx 10^6$). The matrix fill stage of this algorithm is the most computationally intensive and is therefore a good candidate for parallelization. However, the data dependencies that exist to calculate each matrix cell value

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

Table 4: Example of an Origin and End Alignment 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)

(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 $G(i-1,j)$) highly restrict the parallelization to the simultaneous computation of the values along the matrix anti-diagonal direction.

Specialized parallel hardware that is capable of performing a high number of simultaneous matrix computations is especially suited for this task. A linear systolic array with several identical PEs, as shown in Figure 2, is an efficient architecture to implement this type of computation, by simultaneously computing the values of the *G* matrix that are located in a given anti-diagonal.

3.1. Processing Element

The PEs architecture described in this paper is based on the PEs architecture presented in [5]. The simplest PE only implements the function of the basic local alignment algorithm and is shown in Figure 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 Smith-Waterman algorithm requires the determination 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 $Max(i,j)$). The PE *i* selects and stores the maximum score that was computed by PEs 1 through *i*.

The array evolves along the line by shifting the reference sequence character symbols through the PEs. In this array, the 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 involves, among other operations, determining the substitution score between two characters (the value

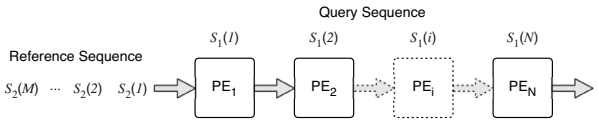

Figure 2: Systolic array structure.

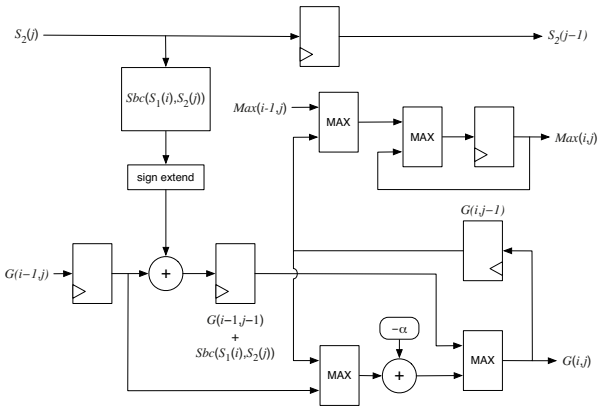

Figure 3: Simple PE architecture.

978-972-789-304-1 REC'2010

of $Sbc(S_1(i), S_2(j))$. Since each PE performs the operations only over one 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 determination of the maximum values that are the result of the three distinct possibilities presented in Equation 1. The zero condition of the Smith-Waterman algorithm is implemented by controlling the reset signal of the registers that store the value $G(i, j)$, by using the most significant bit (sign bit) of the score value, i.e., if the maximum value among the three partial scores is negative, then it clears the registers that hold that given score value.

After all the reference sequence (S_2) characters have passed through all the PEs, the alignment score is available at $Max(i, j)$ output of the last PE.

3.2. Array programming

The query sequence (S_1) data which is loaded into the array is the substitution score matrix column that corresponds to the symbol at that position. In fact, since each PE only performs comparisons to a given query sequence character, it will just access the values present in a certain matrix column. Therefore, each PE will only receive the substitution score matrix column that corresponds to the query sequence character allocated to that PE.

Within each PE, such data is stored using dedicated registers since this allows for a fast reprogramming of the PEs for 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 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 operation of a *new* query sequence into this temporary storage shift register, by serially shifting the substitution matrix column data while the array is 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, while the array is processing the *current* data and 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.

To allow the usage of the same array to process query sequences (S_1) larger than the number of available PEs (N), it is possible to store intermediate results in a local memory. These results are the output values of the last PE in the array and correspond to the scores of a complete row of matrix G. The size of this memory limits the size of the reference sequence (S_2), since it must entirely fit, along with the intermediate calculation data, in this memory. This memory is organized as a FIFO memory and the values stored on it will be later reintroduced in the array and used to compute the alignment for larger sequences.

3.3. Tracking of the Origin and End Alignment Indexes

As it was previously referred, typical applications of hardware accelerators for sequence alignment focus on accelerating only the matrix computation, leaving the traceback for a posterior phase. Furthermore, such implementations only return the alignment score between the two sequences and not all the values of matrix G. Therefore, to obtain the actual alignment, these accelerators force the recomputation of the entire matrix (using a general purpose processor) to be able to perform the traceback phase. This recomputation (and subsequent traceback) is performed in the cases when the alignment score, calculated by the accelerator, is above a given threshold which is defined by the user.

The proposed architecture avoids the recomputation of the entire G matrix by propagating through the PEs, not only the partial maximum scores in the matrix, but also the coordinates where such scores had their origin (the beginning of the alignment), together with the coordinates where the maximum score occurred. As it was shown in Section 2.2, this enables the recomputation phase of matrix G to only focus on the substrings that are actually involved in the alignment and avoid the recomputation of the whole matrix G. Thus, the time and memory space requirements to obtain the sequence alignment are substantially reduced.

To achieve this, an enhanced PE, whose architecture is presented in Figure 4, was developed with the hardware necessary to implement the calculation and propagation of matrix C_b. The datapath that implements this computation is similar to the datapath of a simple PE. The decision logic (inside the maximum calculation units - *Max*) is also used, in this case, to control the selection units of the Origin and End Alignment Indexes (OEAI) tracking coordinates. In each PE, the origin index coordinates, which indicate where the alignment began, are propagated based on the conditions shown in Equation 2.

Since only the coordinates of the origin cell need to be selected alongside with the scores, the PE only incorporates hardware resources to implement such selection in the score calculation datapath. Furthermore, since the simple PEs array is not capable of determining and keep track of the location of the maximum score cell, additional hardware was also included, in the maximum selection datapath, to support the propagation of the coordinates of the cell where the maximum value occurred. Within each PE, the coordinates of the current cell are obtained by using the PE index (i) and a symbol coordinate (j) that comes alongside with the symbol that is at the input of PE_i.

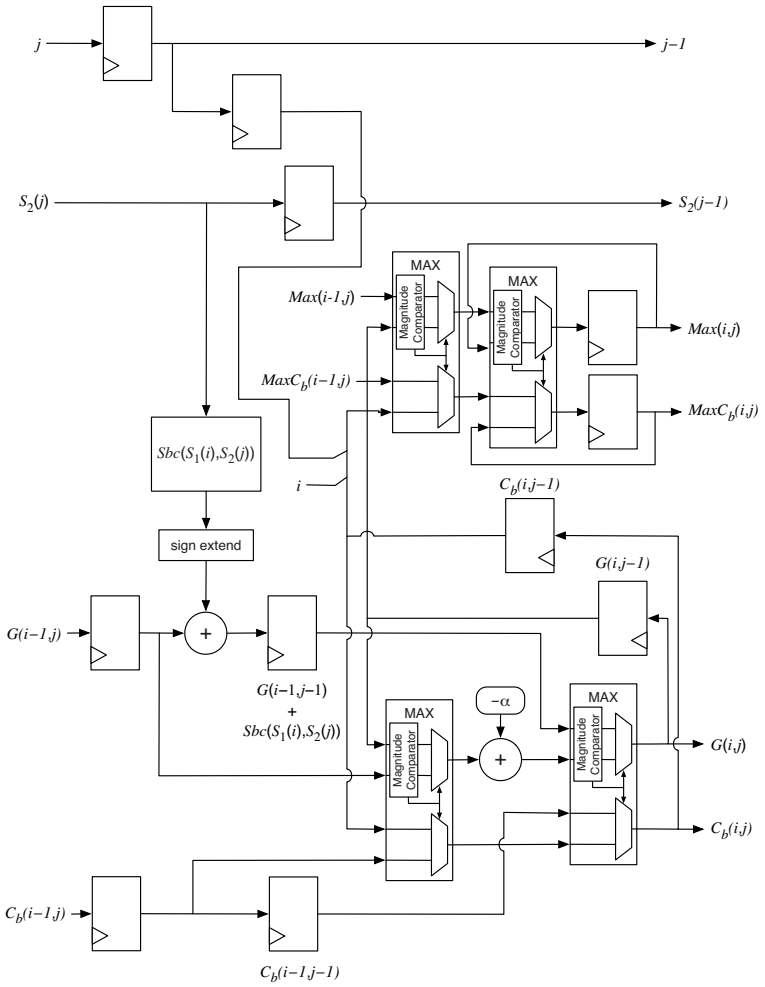

Figure 4: Architecture of PE with OEAI tracking.

3.4. Scalability and Reconfigurability

Whenever the query sequences (S_1) to be aligned are larger than the number of PEs that fit in a FPGA, it is necessary to either reuse or expand the array. Both of these capabilities are supported by the proposed architecture.

When the array is reused in order to perform the alignment with query sequences longer than the number of available PEs, an additional set of control hardware and memory are included in the architecture. The added memory is used to store all the information of a single row of the G matrix (and of the C_b matrix, in case the OEAI tracking function is used). This enables to compute an entire horizontal section of the G matrix, which corresponds to aligning a segment of the query sequence with the entire reference sequence. Afterwards, a new segment of the query sequence is loaded into the PEs and the next horizontal section of the G matrix is computed. This process is repeated until the query sequence has ended. With this implementation, the array limits the size of the reference sequence (m), since the complete data of a single row of matrix G ($m+1$ elements) must fit in a memory block that is available in the device (FPGA). Since the available memory blocks inside this type of devices are usually not large, this capability is only advised for alignments in which the reference sequence is not too long.

To cope with simultaneous long query and reference sequences, this architecture also allows to span the array of PEs over more than one FPGA. This allows to increase the number of PEs in the array, therefore providing the computation of alignments with longer query sequences and without constraining the size of the reference sequence to the amount of available memory inside the FPGA. To implement this capability, relatively small FIFO memories, which store the outputs of the last PE, are used as buffers for the communication between the FPGAs and high-speed communication links are also used to enable a high-speed connection between the devices (see Figure 5).

Moreover, by taking advantage of the reconfiguration capabilities of the FPGA, it is possible to generate systolic array structures that have the number of PEs adapted to the size of the query and reference sequences that will be aligned. Therefore, the reconfiguration capability of the FPGA allows to maximize the obtained performance for a given set of query sequences.

Table 5: Obtained results when using a single FPGA with simple PEs.

# PEs	Symbol bit-width	Score bit-width	Occupied Slice Registers	Occupied Slice LUTs	Maximum Frequency [MHz]	Maximum Throughput [GCUPS]
16	2	8	928 (0.4%)	1656 (0.8%)	205	3.2
16	2	16	1456 (0.7%)	2997 (1.4%)	173	2.7
128	2	11	9088 (4.4%)	19653 (9.5%)	201	25.7
256	2	12	19200 (9.3%)	41919 (20.2%)	171	43.8
512	2	13	40448 (19.5%)	88246 (42.6%)	155	79.0

Table 6: Obtained results when using Origin and End Alignment Indexes tracking.

# PEs	Score bit-width	Maximum Reference size	Maximum Query size	Occupied Slice Registers	Occupied Slice LUTs	Maximum Frequency [MHz]	Maximum Throughput [GCUPS]
16	8	1024 (2^{10})	16	2309 (1.1%)	2816 (1.4%)	216	3.45
16	16	8192 (2^{13})	16	3156 (1.5%)	5187 (2.5%)	149	2.38
128	11	8192 (2^{13})	16	23732 (11.4%)	39822 (19.2%)	161	20.62
128	11	131×10^3 (2^{17})	128	28681 (13.8%)	46635 (22.5%)	162	20.72
256	12	134×10^6 (2^{27})	256	76546 (36.9%)	100489 (48.5%)	147	37.55

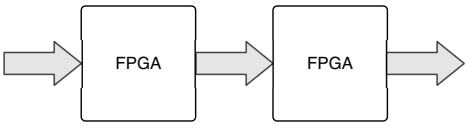

Figure 5: Array extension.

4. FPGA Results

The previously presented architecture was described using parameterizable VHDL code and synthesized for a Xilinx Virtex-5 FPGA (xc5vlx330t) using Xilinx ISE 9.2.04i. Initially, only the simple PE architecture was used, in order to evaluate the resource usage and performance of such array. The obtained results are shown in Table 5

The symbol bit-width represents the number of bits of the registers that hold the characters to be aligned. Since the results were obtained with DNA sequences, which are composed of only four different nucleotides, the characters can be encoded using only 2 bits. The score bit-width represents the number of bits of the registers that hold the score values of G. This resolution is determined according to the specific needs of the system and should have a value that guarantees that no overflows will occur during the processing of matrix G.

As it would be expected, the results show that the throughput increases with the number of PEs, despite the fact that the maximum operating frequency decreases when the device occupancy increases. The maximum obtained throughput is 79×10^9 CUPS (Cell Updates per Second) for a configuration with 512 PEs.

4.1. Tracking of the Origin and End Alignment Indexes

When using the OEAI tracking functionality, the hardware resources spent on each PE are increased, as it can be seen in Table 6. Therefore, for the same FPGA device, the maximum number of PEs that may be implemented in the device is reduced by a factor of 2, which may affect the throughput when large number of PEs are needed. On the other hand, when comparing arrays with the same number of PEs, the array that uses the PEs with the OEAI tracking functionality has a decrease in performance due to a slight reduction of the maximum operating frequency (e.g 14% decrease in maximum throughput for the array with 256 elements). Even so, in application environments where the FPGA resources are not a constraint, this decrease in peak performance may be largely compensated by the fact that the traceback phase of the Smith-Waterman algorithm will take significantly less time and the memory space requirements will be significantly reduced.

As an example, when aligning a query sequence with 200 characters against a reference sequence with 100×10^3 characters, matrix G will have a dimension of about 20×10^6 cells. By considering the scores given in Section 2.1 (a match has a score of 3 and a gap a score of -4), the maximum alignment size (including gaps) will be approximately 350 characters long (a maximum of 3 gaps can be inserted between a 4 character set of S_1 or S_2, thus expanding the size of the alignment by a maximum of $3/4$). Therefore, with the OEAI tracking functionality the maximum size of the alignment matrix G that needs to be recomputed during the traceback stage to find the best alignment has a size of $[(200 + (3/4) * 200/2)]^2 = 76 \times 10^3$ cells, when the gaps are evenly distributed among the two sequences. This leads to a $1/264$ reduction in the size of the recalculated

Table 7: FIFO sizes required for array reuse using OEAI tracking.

Score bit-width	Maximum Reference size	Maximum Query size	Total FIFO size (Bytes)
8	1024	16	$9,5 \times 10^3$
16	8192	16	104×10^3
11	8192	16	94×10^3
11	131×10^3	128	2.0×10^6
12	134×10^6	256	2.8×10^9

matrix leading to a quite significant decrease of the processing time and the involved memory requirements, which largely compensates the individual PE performance degradation described above.

When using the OEAI functionality, the Maximum Reference Sequence Size is imposed by the bit-width of the registers that hold the coordinates for the reference sequence. In contrast, the Maximum Query Size is imposed by the maximum number of PEs that can be accommodated.

4.2. Array reuse

When the reuse of the PEs array is considered to compute alignments with query sequences larger than the number of PEs, the main concern relates to the amount of memory required to store all the partial values of an entire row of the *G* matrix. Table 7 depicts the amount of memory that is required for several configurations.

As it can be seen, the memory requirements for aligning a query sequence against a reference sequence that has 131×10^3 characters requires about 2MB of memory to hold the values of a single row of *G*. For even larger reference sequences, the required amount of memory is so large that it will not be possible to reuse the array to perform alignments with query sequences larger than the number of PEs.

In such situations, it is preferable to use the expansion method (connecting another FPGA as shown in Figure 5) to enable fast and unconstrained alignments with large query sequences.

5. Conclusions

This paper presented a flexible architecture for accelerating the SW local sequence alignment algorithm using FPGAs. It also proposed an innovative method that provides a significant reduction of the computation time and memory space requirements of the traceback phase of the alignment procedure. The results obtained from an implementation of the proposed architecture using a Virtex-5 FPGA showed that such method is highly feasible in order to provide significant gains in terms of the overall performance of the whole alignment procedure. Furthermore, with long reference sequences and when the query sequences are longer than the number of PEs that can be accommodated in a device, it was shown that it is preferable to span the array of PEs across multiple FPGA devices instead of reusing the array. This is mainly a constraint imposed due to the limited amount of memory space available in current FPGAs.

Acknowledgment

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.

References

[1] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, and E. Sayers, "GenBank," *Nucleic Acids Res.*, vol. 37, no. Database issue, pp. D26–D31, Jan. 2009.

[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] T. Ramdas and G. Egan, "A Survey of FPGAs for Acceleration of High Performance Computing and their Application to Computational Molecular Biology," in *TENCON 2005*, Nov. 2005, pp. 1–6.

[4] 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.

[5] 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.

[6] 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.

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

[8] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: computer science and computational biology*. Cambridge University Press, 1997.