Reconfigurable Architecture for Biological Sequence Comparison in Reduced Memory Space*

Azzedine Boukerche[1], Jan M. Correa[2], Alba Cristina M. A. de Melo[2], Ricardo P. Jacobi[2], Adson F. Rocha[3]

[1] *SITE, University of Ottawa, Canada*
[2]*Department of Computer Science, University of Brasilia, Brazil*
[3]*Department of Electrical Engineering, University of Brasilia, Brazil*
boukerch@site.uottawa.ca, {jan, albamm, rjacobi}@cic.unb.br, adson@ene.unb.br

Abstract

DNA sequence alignment is a very important problem in bioinformatics. The algorithm proposed by Smith-Waterman (SW) is an exact method that obtains optimal local alignments in quadratic space and time. For long sequences, quadratic complexity makes the use of this algorithm impractical. In this scenario, the use of a reconfigurable architecture is a very attractive alternative. This article presents the design and evaluation of an FPGA-based architecture that obtains the similarity score between DNA sequences, as well as its coordinates. The results obtained in a Xilinx xc2vp70 FPGA prototype presented a speedup of 246.9 over the software solution to compare sequences of size 100MBP and 100BP, respectively. Different from others hardware solutions that just calculate alignment scores, our design was able to avoid architecture's bottlenecks and accelerate the most computer intensive part of a sequence alignment software algorithm.

1 Introduction

Bioinformatics can be defined as any type of study or tool used to produce or organize biological information [9]. Bioinformatics has a great potential to support research of new medicines and biotechnological-based materials. Pairwise biological sequence comparison is one of the most important and basic operations in bioinformatics since it is often used as a basis to solve more complex problems such as multiple sequence alignment, phylogeny relations inference and RNA second structure prediction [8].

Sequence alignment is in fact a problem of finding an approximate pattern matching between two sequences, possibly introducing spaces (gaps) into them [29]. The most common types of sequence alignment are global and local. The goal of a global alignment algorithm is to obtain the similarity between two sequences as a whole [26]. On the other hand, local comparison is used to obtain the similarity between parts of the sequences

Dr. A. Boukerche work is partially sponsored by NSERC and Canada research Chair Program

To solve biological sequence comparison problems, there have been proposed exact methods based on dynamic programming such as Smith-Waterman (SW), Myers and Miller [25] and Gotoh [11]. These methods have, at least, quadratic time complexity.

In order to obtain results faster, heuristic methods such as BLAST [1] and Fasta [22] have been proposed. However, the performance gain is often achieved by reducing the quality of the results produced. To produce optimal results faster, parallel algorithms such as [7], [4] and [3] have been proposed in the literature. However, for long sequences, execution times are still high. For instance, to compare two 3MBP (*Mega Base Pairs*) DNA sequences with an affine gap model, the parallel algorithm proposed in [3] takes more than 13 hours, with 16 processors.

One approach to further accelerate the dynamic programming methods is to design application-specific hardware. Many dedicated architectures [17] have been proposed in the literature. Some of them, such as FPGA (*Field Programmable Gate Array*) based solutions that can be integrated to a parallel algorithm, leading to a hardware-software approach.

In this article, we propose and evaluate a reconfigurable architecture that executes the most compute-intensive phase of the SW algorithm [30] in linear space, providing, as output, the coordinates and the value of the similarity between two sequences *s* and *t*. This information can be easily used to retrieve the actual alignment [14].

2 Biological Sequence Comparison

2.1 Similarity score

To compare two sequences, we need to find the best alignment between them, which is to place one sequence above the other making clear the correspondence between similar characters [30]. In an alignment, spaces can be inserted in arbitrary locations along the sequences.

In order to measure the similarity between two sequences with a linear gap function, a score is calculated as follows. Given an alignment between sequences *s* and *t*, the following values are assigned, for instance, for each column: a) *+1*, if both characters are identical (*match*); b) *-1*, if the characters are not identical (*mismatch*); and c) *-2*, if one of the characters is a space (*gap*).

The score is the sum of all these values. The similarity between two sequences is the highest score. Figure 1 presents one possible alignment between two DNA sequences and its associated score.

A	C	T	T	G	T	C	C	G	-	A	G	A	
A	-	T	T	G	T	C	A	G	G	A	G	G	score
+	-	+	+	+	+	+	-	+	-	+	+	-	4

Figure 1. Example of alignment and score

2.2 Algorithm SW for local alignment

The algorithm Smith-Waterman (SW) [30] is an exact method based on dynamic programming to obtain the best local alignment between two sequences in quadratic time and space. It is divided in two phases: create the similarity matrix and obtain the best local alignment.

2.2.1 Creating the similarity matrix. This phase receives input sequences s and t, with $|s| = m$ and $|t| = n$, where $|s|$ represents the size of sequence s. For sequences s and t, there are $m+1$ and $n+1$ possible prefixes, respectively, including the empty sequence. The notation used to represent the *n-th* character of a sequence *seq* is *seq[n]* and, to represent a prefix with n characters, we use *seq[1..n]*. The similarity matrix is denoted $D_{m+1,n+1}$, where $D_{i,j}$ contains the similarity score between prefixes *s[1..i]* and *t[1..j]*.

At the beginning, the first row and column are filled with zeros. The remaining elements of D are obtained from equation (1). In equation (1), $p(i,j) = 1$ if $s(i)=t(j)$ (*match*) and –1 otherwise *(mismatch)*. In this case, -2 is the gap penalty. The similarity between sequences s and t is the highest score.

$$sim(s[1..i],t[1..j]) = \max \begin{cases} sim(s[1..i],t[1..j-1]) - 2 \\ sim(s[1..i-1],t[1..j-1]) + p(i,j) \\ sim(s[1..i-1],t[1..j]) - 2 \\ 0 \end{cases} \quad (1)$$

Figure 2 presents the similarity matrix between sequences s = TATGGAC and t = TAGTGACT. The arrows indicate the cell from where the value was obtained.

		T	A	T	G	G	A	C
	0	0	0	0	0	0	0	0
T	0	↖2	0	2	0	0	0	0
A	0	0	↖4	2	1	0	2	0
G	0	0	↑2	3	4	3	1	1
T	0	2	0	↖4	2	3	2	0
G	0	0	1	2	↖6	←4	2	1
A	0	0	2	0	4	5	↖6	4
C	0	0	0	1	2	3	4	↖8
T	0	2	0	2	0	1	2	6

Figure 2. Similarity matrix to locally align two DNA sequences. The black arrows indicate the traceback to obtain the actual alignment

2.2.2 Obtaining the best local alignment. In order to obtain the best local alignment, the algorithm starts from the cell that contains the highest score and follows the arrows until the value zero is reached. A left arrow in $D_{i,j}$ (figure 2) indicates the alignment of *s[i]* with a gap in t. An up arrow represents the alignment of *t[j]* with a gap in s. Finally, an arrow on the diagonal indicates that *s[i]* is aligned with *t[j]*.

Note that many best local alignments can exist, since many arrows can exist in the same cell $D_{i,j}$, indicating that the score value was produced from more than one cell.

2.3 Obtaining local alignments in linear space

One of the most restrictive characteristics of the SW algorithm is the quadratic space needed to store the similarity matrix. For instance, in order to compare two 100Kbps (*Kilo Base Pair*) DNA sequences, we would need at least 10GB of memory. This fact was observed by [25], that proposed the use of Hirschberg´s algorithm [15] to compute global alignments in linear space. The algorithm uses a divide and conquer technique that finds a point where the optimal alignment occurs and recursively splits the similarity matrix to obtain the actual alignment. This approach can double the execution time, in the average case [15], when compared with the basic dynamic programming method.

In order to use this approach to calculate local alignments, the following steps must be done [14]. First, the similarity array is completely calculated in linear space, in order to obtain the location (coordinates i and j at the similarity matrix) where the highest score occurs. This is in fact the position where the best local alignment ends. Second, the similarity array is re-calculated from the highest score position over the reverses of the sequences, in order to obtain the position where the best local alignment begins. Having the beginning and end coordinates of the best alignment, this problem is transformed into a global alignment problem and Hirschberg´s algorithm [15] can be used.

2.4 Parallel variations of the basic algorithms

In the SW algorithm, most of the time is spent calculating the similarity matrix D (figure 2) and this is the part which is usually parallelized. The access pattern presented by the matrix calculation is non-uniform and the parallelization strategy that is traditionally used in this kind of problem is known as the wavefront method since the calculations that can be done in parallel evolve as waves on diagonals.

Figure 3 illustrates the wavefront method where 4 processors (P1-P4) are used to calculate the matrix. Each processor Pi will calculate a set of columns in matrix D. In this calculation, the precedence relations of equation 1 must be obeyed, thus, each anti-diagonal can be computed in parallel. At the beginning of the computation, only P1 can compute (figure 3.a). When

P1 finishes calculating the values of a border column, it sends them to P2, that can start calculating, while P1 continues processing its next block (figure 3.b). In figure 3.c, the maximum parallelism is attained.

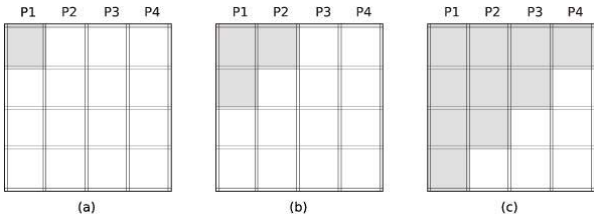

Figure 3. The wavefront method

A parallel exact algorithm that solves the global sequence alignment problem with affine gap functions in *O((mn)/p)* time and requires *O((m+n)/p)* space is proposed in [28]. The key idea of this algorithm is to add an additional phase to find a partial balanced partition between sequences of *s* and *t*. To compute the partial balanced partition in parallel, three dynamic matrices are used to find the cells where a recursive decomposition of the problem can be performed. An optimal alignment *c* between sequences *A* and *B* is proved to be the concatenation of optimal alignments to the subproblems corresponding to each region [28].

In [6], a parallel exact variation of a linear space algorithm based on SW is proposed that finds a set of local alignments that are close to the best (near-best). The algorithm has three phases. In the first phase, the end coordinates of the best (and near best) alignments are found. In the second phase, the coordinates of the beginning of the alignments are found over the reversed sequences. In the third phase, having the beginning and the end coordinates of each non-overlapping alignment (best and near-best), the Hirschberg algorithm [15] is used to retrieve the actual alignments.

In [3], an exact parallel SW variation is proposed that calculates local alignments in user-restricted memory space. The algorithm is divided in four phases. At the first phase, the input sequences *s a*nd *t* are distributed to the nodes. The second phase is the most compute-intensive since it calculates the entire similarity array in linear space over the reverses of the sequences, obtaining the begin coordinate(s) of the best alignment(s). In this phase, the number of diagonals needed to obtain the alignments (superior and inferior divergences) is also calculated. At the third phase, all nodes send the best score found and its coordinates to the master node. The master node decides which one is the global best. At the last phase, the beginning coordinates of the optimal alignment(s) are obtained and the alignment is retrieved using the superior and inferior divergences. This phase executes in user-restricted memory space.

3 Dedicated and FPGA Architectures

In contraposition to general-purpose processor architectures, a dedicated architecture is designed and optimized to solve a specific problem. Usually, dedicated architectures can be built by using ASIC (*Application Specific Integrated Circuit*) processors or FPGAs. Although very fast circuits can be obtained with dedicated ASIC processors, the resulting architecture is not flexible and often expensive. A very promising alternative to build dedicated architectures is by using FPGAs. FPGAs are slower than ASIC circuits but, as advantages, they can be reconfigured and the circuit can be obtained in much less time.

The reconfigurability of FPGAs may be exploited to tailor hardware resources to the problem being solved. It can be seen as a soft component, which may be changed to be adapted to a specific application. Indeed, currently supercomputers start to introduce FPGAs in order to speed up some critical parts of an application.

A common parallel organization used in FPGAs is a systolic array, a hardwired mesh-connected pipe network of datapath units, using only nearest-neighbor (NN) interconnects. Each element of the array performs a simple task and passes its results to the next connected element [17]. All systolic cells work in parallel, processing streams of data at the array's clock rate. Systolic architectures are normally arranged in one or two dimension matrices. Nevertheless, there are some problems that arise when trying to apply FPGAs to solve specific problems. First, the FPGA communicates with the host computer to receive data and instructions. The communication speed is limited by the channel data rate (in many cases, the PCI). According to the communication technology adopted, this channel may be a bottleneck between the host computer and the FPGA board. For this reason, this type of communication must be reduced. Second, to be efficient, an FPGA must be able to do as much work as possible in parallel and that demands a lot of computing elements, ideally one for each parallel computation. Unfortunately, for many problems, this ideal number of elements goes far from the capacity of today´s FPGAs.

4 Related Work

There are many bioinformatics algorithms on FPGAs. Some attempted to use heuristic methods such as BLAST [1] to solve the sequence alignment problem such as [5] [18] and [19]. The best results were obtained by [18] with a speedup of a 100 over a 1.4 GHz AMD Operon but this proposal only finds the best score and it only can deal with small sequences. The Blast section implemented in [19] had worse time than a software implementation. The time required to transfer the data to the FPGA was already more than the time to execute the entire algorithm in software showing that this algorithm is difficult to implement taking full advantage of the parallel nature of FPGA

and due to the board communication bottleneck problem.

There are many proposals in the literature of FPGA-based architectures to accelerate the SW algorithm ([17] [13] [16] [21] [24] [27]]) that calculate the similarity matrix antidiagonals in parallel, taking full advantage of the wavefront method (figure 3). This approach allows using the parallel potential of FPGA circuit for calculating many matrix cells at the same time. In this case, the architecture is formed of *N* computing elements. Each element is capable of calculate one matrix score per turn. Thus, an *N* elements array can generate *N* scores at a time.

Figure 4 shows how each anti-diagonal of the dynamic programming matrix is calculated in parallel by the systolic array, as shown in figure 2. A query sequence (ACGAT) is previously put in the elements of the array and the database sequence (CTTAG) flows through the systolic array. Each element will calculate one cell in the current anti-diagonal (shown in gray in figure 4) at the time.

This approach is very powerful because an FPGA can calculate billions cells per second for a plain Smith Waterman [13], [20], [27]]. However, the quadratic space complexity of the SW algorithm is a great restriction. For this reason, most of the solutions proposed in the literature do not store the entire similarity matrix, obtaining only the similarity score.

Besides that, there is a limited number of computing elements that can be put in the systolic array. This number is restricted by the amount of hardware resources that are available. Usually, the number of computing elements is very small rarely more than a few hundreds. To deal with it, the smallest sequence being aligned is often put on the computing elements as a query sequence. The other sequence can be of any size, since it "passes" through the FPGA (figure 4).

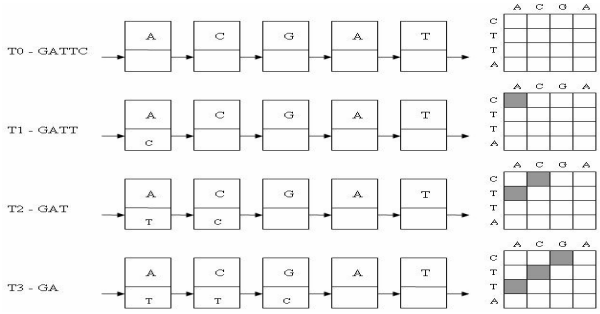

Figure 4. Generic systolic array to calculate the similarity matrix

Frequently, it happens that the query sequence is greater than the number of computing elements. In this case, a partitioning technique is needed. To break query sequences, it is necessary to keep some scores on the board to allow new scores to be calculated (according to the recurrence relation). Some designs like [12] avoid this problem by putting many query bases on the same computing element. The drawback of this approach is that to put more bases at each cell requires more registers per element and thus decreases the maximum number of computing elements in the systolic array.

To reduce the amount of resources consumed and increase the speed of calculations, [13] utilized a Xilinx JBits toolkit that allows dynamic reconfiguration. In this case, the query sequence is put directly in the processing elements using the dynamic reconfiguration capability instead of being stored by registers inside. This resulted in a 25% reduction in the overall circuit sparing 2 flip-flops for each base storage. A drawback of this approach is that configuration time that normally takes milliseconds. That makes it difficult to use for large query sequences that would require many reconfigurations of the FPGA to put new splits of the sequence in processing elements.

One metric used to measure the performance of FPGA-based approaches is the number of CUPS (Cell Updates Per Second) where each cell update is a matrix cell calculation. This measure indicates the computation power of an architecture but it is highly dependent of the complexity of the calculation. For instance, an architecture that generates alignments cannot be compared fairly with an architecture that only generates scores in terms of CUPS because the former is doing more work per cell. To be fair, each cell must to be doing similar work.

In the following paragraphs, we discuss some FPGA-based architectures. In [17] is described an implementation of the SW algorithm using a systolic array that calculates each anti-diagonal in the matrix in parallel. It was noted that the Smith-Waterman matrix has many zeros and many optimizations were made to shorten memory usage. In addition, with a JDS (Jagged Diagonal Storage) matrix representation cut by 75 to 80 percent the amount of memory needed.

In [21], it was shown an implementation of SW in a SAMBA board (*Systolic Accelerator for Molecular Biological Applications*). This proprietary board had a systolic array with 128 processors of 12 bits with a configurable interface that can be programmed through C library functions. It was built a systolic linear array where each element of the anti-diagonal was calculated in each cycle. The sequence can be split to fit in the FPGA and each part can be compared against a database to generate the entire matrix. Comparing SAMBA with a DEC Alpha 150 MHz for searching a 3000 amino acids sequence in a database sequence of 21210389 it took less than 4 minutes for SAMBA while the DEC Alpha took 280 minutes, yielding a 83 fold improvement. In [23], a tool called PHG (Parallel Hardware Generator) was used to generate a VHDL code and synthesize hardware from the recurrence relations (equation 1). The basic comparison function was hand-made. The problem was to search a small sequence of amino acid (a peptide) in a huge amino acid database. The implementation was done in a FPGA Xilinx XV1000-4 and it did a result 5.6 times

better than a Pentium III 1 GHz to search a 24 BP sequence in a 2MBP database.

An implementation of the SW algorithm with an affine gap function was proposed in [2]. The scores are calculated in anti-diagonal on a systolic array. Due the FPGA memory restrictions, the sequences are partitioned. The database sequence is divided in parts that fit on the systolic array. If the query sequence is bigger than the systolic array, each element in the array may hold up to 4 bases in its internal registers. If the query sequence is not a multiple of the array size the remaining, registers are filled with zeros. The longest query sequence compared had 1512 BP. In a Virtex II XC2V6000 FPGA they achieved 1,390 GCUPS (*Giga Cell Updates per Second)*.

In this proposal, the FPGA calculates the score matrix and sends it to the host computer CPU that finds the best alignment. Comparing the results obtained by an optimized C program in a 1.6 GHz Pentium 4, the FPGA had a speedup of 170. In [37], an array of computing elements was also used. If the sequence are greater than the number of elements available the sequence is broken in many parts, in a multithreaded way. In the first stage, the upper half of the anti-diagonal is calculated and stored in registers (in case query sequence is divided in 2). In a second stage the lower half of the anti-diagonal e calculated using the same elements. It is useful in this particular implementation where there is an idle cycle between comparisons.

The procedure is divided in 2 phases. In the first phase, the database sequence in broken in a way it can fit in the FPGA internal memory. In the second phase, the best alignment is computed by the FPGA. In order to align a 2048 BP sequence against a 64MBP database in a Xilinx XCV2000E FPGA, a result of 5,76 GCUPS was obtained. In time, it took 34 seconds on the FPGA and a speedup of 330 over a 1GHz Pentium III was achieved. In the second phase, a 1 KBP query sequence was compared with a 4KBP database sequence. It took 0.007 seconds on the FPGA whereas the Pentium III did the same work in 0.35 seconds.

Table 2 presents a comparative analysis among some of these FPGA-based architectures to locally align biological sequences.

Table 1. Comparative Analysis

Article	FPGA	size. Query Seq / size.Base Seq	Sequence Splicing	SpeedUp	Type	Alingnment
[21]	SAMBA	3Kbp / 2.1Mbp	Yes	83	DEC 150 MHz	No
[23]	Xilinx XV1000	24 / 2Mbp	No	5.6	Pentium III 1 GHz	No
[32]	XC2V6000	1.5kbp/ N.A.	Yes	170	Pentium 4 1.6 GHz	No
[37]	XCV2000E	2K / 64Mbp	Yes	330	PentiumIII 1GHz	Yes

In Table1, we can see that sequence splitting was used in most of the proposals [21][32][37]. In [23], tests were done with small queries that fit in the systolic array. It was done probably for performance results because doing in this way the systolic array is faster and requires less communication with host computer.

Although many Smith-Waterman implementations had excellent results when compared against software solutions, it is not easy to implement this in FPGA for real world problems due its constraints. There is a perspective that, in the future, FPGA boards will have larger storage spaces and may be connected to higher speed slots. When that happens most of problems related to sequence splitting could be improved.

5 Proposal of an FPGA-based SW Architecture

The goal of our FPGA-based architecture is to execute the first phase of the SW algorithm in linear space (section 2.3). As input, we receive the sequences to be compared (*s* and *t)* and the output produced is the value of the highest score as well as its coordinates in the similarity matrix. This solution can be easily integrated to parallel algorithms such as [3] and [7] that will produce the alignments in software. As in [21][23][32] and [37], we will also use a systolic array to calculate the scores of each antidiagonal of the similarity matrix in parallel. Figure 5 presents our systolic array design, which is based on the solution illustrated in figure 4. In figure 5, the query sequence ACGC is stored at each systolic element and the database sequence (ACTA) flows through the elements. Each anti-diagonal of the similarity matrix is calculated by our systolic array in a way that is similar to the one illustrated in figure 4. Besides that, in our design, each element has two additional fields that must be calculated by the element's circuit. The lower number in figure 5 is the best score calculated in that column so far. The upper number is the cycle when that score was calculated. After the entire matrix is calculated, these fields are used to find out the location of the best score. With the cycle field, the row containing the best score is found.

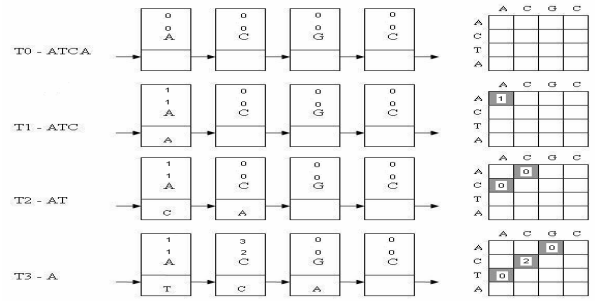

Figure 5. Proposed systolic array design

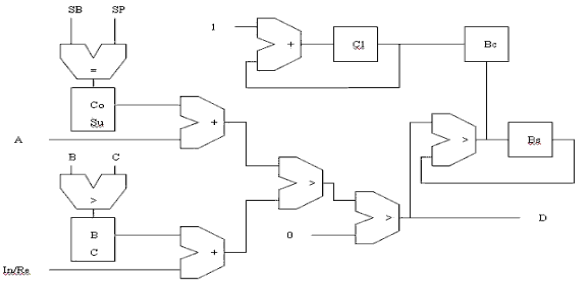

Figure 6. Sequence similarity calculation

Figure 6 illustrates the computation of the similarity matrix, best score and its position. To calculate each cell D in the matrix, three cells are needed. Upper cell B, left cell C and diagonal cell A. In each cycle, values A, B stored in registers and value C transmitted from left element are used in D calculation. Two bases are compared: SP that is fixed in the element and SB that comes from the left element. If they are equal, it means that a match occurred and the coincidence value Co is used.

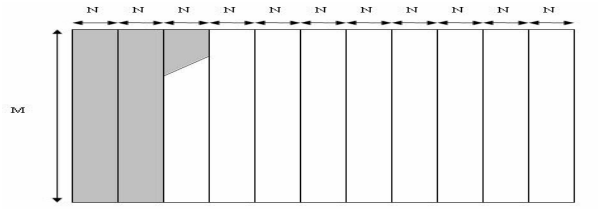

Figure 7. Sequence partitioning in our design

Otherwise, it is a mismatch and the substitution value Su is used. In parallel, values B and C are compared and the greater is added to Insertion and Removal In/Re score (gap calculation). These two generated values are compared and the greater one is compared to zero. If it is greater than zero it doesn't change otherwise it is replaced by zero.

To calculate the best score, a comparator is connected to D. If the new D score is greater than the value stored in register Bs (best score in that column so far) it replaces the old value. To recover the antidiagonal and therefore the row where the best score was calculated, register Cl is utilized. It is incremented by one each time a new score is calculated. If the current score is greater than the one stored in Bs, the writing on register Bc is enabled and its value is replaced by the current value of Cl. Therefore, Cl stores the antidiagonal where Bs is calculated.

In our design, a large database sequence can be put in the FPGA board SRAM memory that can handle several megabytes in most modern models. On the other hand, the query sequence must be put in the systolic array that has limited space. Large query sequences can be partitioned in subsequences of size *N* where *N* is the number of elements in systolic array. Sequence partitioning is shown in figure 7.

In figure 7, the *N* systolic elements can calculate a part of the similarity matrix in antidiagonals as explained before. When the computation of an *M*x*N* part is finished (shown in gray), the next *N* bases can be put in the systolic and calculation proceeds.

6 Experimental Results

The designed systolic array was simulated in SystemC [31], which is a very powerful tool for circuit design and simulation. SystemC allows a very high level and flexible simulation.

After the design was tested, it was translated to a language that could be synthesized in FPGA with a tool called Forte [10]. Forte takes a customized SystemC program as input and generates an optimized Verilog design as output. Verilog can be used for synthesis in many commercially available FPGAs.

To test the Verilog design on a real platform, we synthesized the circuit to a Xilinx xc2vp70 using Xilinx's ISE 8.1 on a Pentium 4 3 GHz.

The prototype circuit had 100 elements. The ISE generated three separated diagrams (left, middle and right). The left and right parts of our circuit are shown in figures 8 and 9.

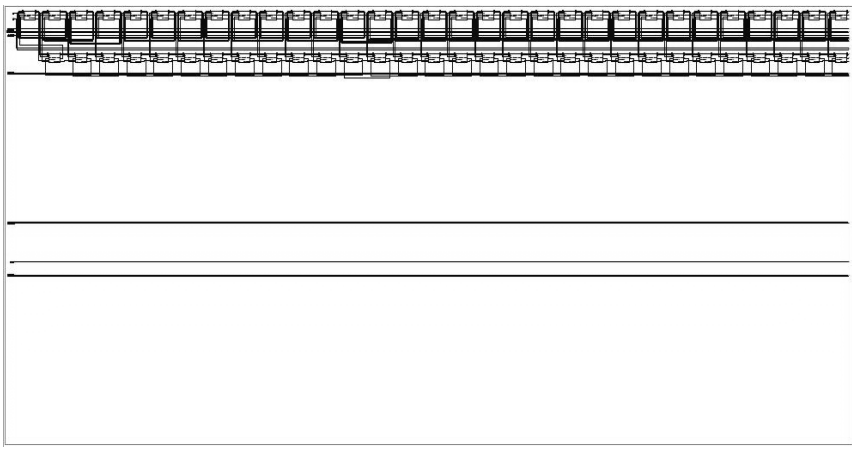

Figure 8. Left part of the generated circuit

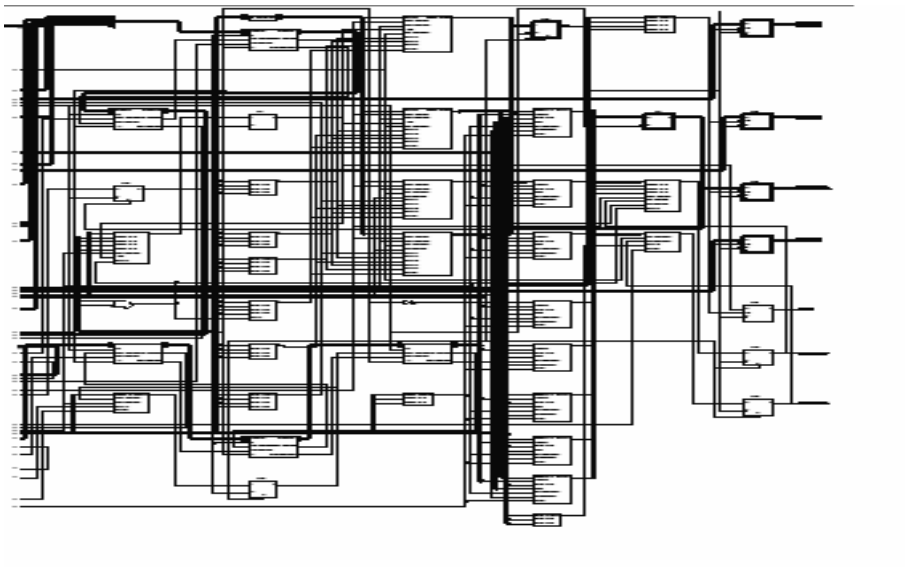

Figure 9. Right part of the circuit

Figure 8 shows the circuit elements that are used to calculate the similarity matrix and the highest score and its row, for each column. As can be seen, there is space to add much more elements. The right part of the circuit (figure 9) contains the logic used to control the matrix calculation as well as the computation of the highest global score and its coordinates. The circuit's operation frequency reported by the ISE tool is 174.749 MHz for 100 elements. The results discussed here can be considered as a first approximation to the real performance. In a near future, real tests will be performed on a prototyping board. Table 2 shows the amount of resources used in the Xilinx xc2vp70 FPGA.

Table 2. Characteristics of the Generated Circuit

Elements	Slices	Flip-flops	Luts	IOBs	GCLKs	Freq.
100	69%	25%	65%	7%	1	174.7

As can be seen in table 2, for 100 elements, only less than 70% of the FPGA was used. In our tests, we used a query sequence of size 100 BP, which was compared with a database of size 10MBP. In this case, it took 0.744 seconds to calculate the similarity array of size 10MBPx100BP as well as the highest score and its coordinates in our test FPGA (Xilinx xc2vp70).

An optimized C program that implemented the same algorithm (i.e. computation of the same matrix and highest score) on a Pentium 4 3 GHz 512 MB, took 183.72 seconds. This means that a FPGA has a speedup of 246.9.To compare a FPGA fairly with a software implementation some caution in needed. Only the CPU time must be taken in account disregarding I/O operations like file readings. The software must do the same work as the FPGA. As the systolic array computes the best score and location the software must do just it. In this case, the most computer intensive part of sequence alignment algorithm was done in FPGA with a speedup more than two hundred of the equivalent part of software. After the calculation is

done on FPGA, only a few bytes need to be transferred to the host, and that can be done in few milliseconds through the PCI bus.

7 Conclusions and Future Work

In this paper, we proposed and evaluated an FPGA-based solution to accelerate the most compute intensive part of a linear space variant of the SW algorithm to locally align biological sequences. As input, our circuit receives the sequences to be compared. The smallest one is placed at the FPGA and the longest one is passed through the circuit. As output, we produce the similarity score as well as its coordinates at the similarity matrix. These information are very useful to retrieve the actual local alignments. The results obtained in a prototype synthesized for a Xilinx xc2vp70 board that contained 100 processing elements were very good. When compared to a software implementation of the same part of the algorithm, our prototype achieved a speedup of 246.9, reducing execution time from more than 3 minutes to less than 1 second.

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. & Lipman, D.J. (1990) "Basic local alignment search tool." J. Mol. Biol. 215:403-410.

[2] Anish, Alex, Hardware Accelerated Protein Identification, MSc thesis, University of Toronto, 2003.

[3] Batista, R. B. and Melo, A. C. M. A., Z-align: An Exact Parallel Strategy for Biological Sequence Alignment in User-Restricted Memory Space, Proc. of the IEEE Int. Conf. on Cluster Computing, September, 2006.

[4] Boukerche, A., Melo A. C. M. A., Ayala-Rincon, M. and Santana, T. M., Parallel Smith-Waterman Algorithm for Local DNA Comparison in a Cluster of Workstations, In. Proc. of the 4th Int. Workshop on Experimental and Efficient Algorithms, LNCS 3503, 2005, p. 464-475.

[5] Chang, Chen, BEE2: A High-End Reconfigurable Computing System, Technical Report, California University at Berkeley, 2004

[6] Chen, C. and Schmidt, B. Computing large-scale alignments on a multi-cluster. In IEEE International Conference on Cluster Computing, 2003.

[7] Chen, C. and Schmidt, B. An adaptive grid implementation of DNA sequence alignment", Future Generation Computer Systems, 21:988-1003, 2005.

[8] Durbin, R., et. al. , Biological Sequence Analysis, Cambridge Univ Press, 1998.

[9] European Commission, Prospective Analysis of the relationship and synergy between Medical Informatics (MI) and Bioinformatics (BI) White Paper EC-IST 2001-35024, 2002

[10] Forte Design Systems, "Cynthesizer User's Guide For Cynthesizer 2.4.0" 2005

[11] Gotoh, O., An improved algorithm for matching biological sequences. Journal of Molecular Biology, 162:705-708, 1982.

[12] Grate, L. et. al.Sequence Analysis With the Kestrel SIMD Parallel Processor, Pacific Symposium on Biocomputing, Hawaii, Estados Unidos, 2001.

[13] Guccione, Steven A. et. al. Matching Using JBits, Field-Programmable Logic and Applications, Reconfigurable Computing Is Going Mainstream, LNCS 2002.

[14] Gusfield, Dan, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology Cambridge University Press, UK, 1997

[15] Hirschberg., D. S. A linear space algorithm for computing maximal common subsequences. Communications of the ACM, 18:341–343, 1975.

[16] Hoang, Dzung T. FPGA Implementation of Systolic Sequence Alignment, Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping, H. Grunbacheret. al. Springer-Verlag, 1992, pp. 183-191.

[17] Jacobi, R. P., et. al. Reconfigurable systems for sequence alignment and for general dynamic programming. Genetics and molecular research, São Paulo, Brasil, v. 4, n. 3, p. 543-552, 2005.

[18] Knowles, Greg, Gardner-Stephen, Paul, A New Hardware Architecture for Genomic Sequence Alignment, 3rd International IEEE Computer Society Computational Systems Bioinformatics Conference 2004

[19] Krishna Muriki, Keith Underwood, Ron Sass, RC-BLAST: Towards an Open Source Hardware Implementation, HiCOMB 2005 Fourth IEEE International Workshop on High Performance Computational Biology

[20] Lavenier, D. Dedicated Hardware for Biological Sequence Comparison, Journal of Universal Computer Science, 2 (2) 1996.

[21] Lavenier, D. Speeding up genome computations with a systolic accelerator, *SIAM news,* 31 (8) 1998.

[22] Lipman, D. and Pearson, W. Improved tools for biological sequence comparison , Proc. of the National Acamemy of Science, USA, 85:2444-2448, 1988.

[23] Marongiu, A. Palazzari, P. Rosato, V., PROSIDIS: a Special Purpose Processor for PROtein SImilarity DIScovery,Second IEEE International Workshop on High Performance Computational Biology

[24] Mosanya, Emeka, "A Reconfigurable Processor for Biomolecular Sequence Processing", Phd Thesis , Ecole Polytechnique Fédérale de Lausanne, 1998.

[25] Myers, E. W. and Miller, W., Optimal alignments in linear space. Computer Applications in the Biosciences, 4:11-17, 1988.

[26] Needleman S. B. e Wunsch C. D., A General Method Applicable to the Search for Similarities in the Amino-Acid Sequence of Two Proteins, Journal of Molecular Biology, 48, pp. 443–453, 1970.

[27] Puttegowda, K.; Worek, W.; Pappas, N.; Dandapani, A.; Athanas, P.; Dickerman, A., A run-time reconfigurable system for gene-sequence searching, Proceedings. 16th International Conference on VLSI Design, 2003

[28] Rajko, S. and Aluru, S. Space and Time Optimal Parallel Sequence Alignments, IEEE Transactions on Parallel and Distributed Systems, 15(2):1070-1081, 2004.

[29] Setubal, J. e Meidanis, J., Introduction to Computational Molecular Biology. PWS Pub.. 1997

[30] Smith. T.F. and Waterman. M.S. (1981) Identification of common molecular sub-sequences. Journal of Molecular Biology, 147 (1) 195-197. PubMed.