Hardware Accelerator for Biological Sequence Alignment using Coreworks® Processing Engine

José Cabrita, Gilberto Rodrigues, Paulo Flores
INESC-ID / IST, Technical University of Lisbon
jpmcabrita@gmail.com, gilberto.rodrigues@ist.utl.pt, paulo.flores@inesc-id.pt

Abstract

Several algorithms exist for biological sequence alignment. The Smith-Waterman (S-W) algorithm is an exact algorithm that uses dynamic programming for local sequence alignment. Some implementations in software for General Purpose Processors (GPP) as well in hardware (using Field Programmable Gate Array (FPGA)) exist. In this paper it is proposed an implementation of the S-W algorithm for DNA, RNA and amino acids sequence alignment that uses the Coreworks® processing engine. The processor FireWorks™ will be used to control a hardware accelerator named SideWorks™ both developed by Coreworks®. In this paper is proposed an architecture based on Process Elements (PE) to be implemented in SideWorks™ accelerator template with the purpose of accelerating the S-W algorithm.
The developed application is able to read sequences from a file, align them with a library of sequences and present the results for the best local alignments using the Coreworks® processing engine.

Keywords— *DNA, Bioinformatics, Sequence Alignment, Smith-Waterman algorithm, Field Programmable Gate Array (FPGA), Cell Updates Per Second (CUPS), Platform Design, SideWorks™, FireWorks™*

1. Introduction

Sequence alignment is one of the most widely used operations in computational biology. The need for speeding up this operation comes from the exponential growth of biological sequences databases.

The sequence alignment operation consists of finding similarities between a certain test sequence and all the sequences of a database. This operation allows biologists to point out sequences sharing common subsequences. From a biological point of view, this operation leads to identifying similar functionality.

The S-W algorithm is a well-known dynamic programming algorithm for performing local sequence alignment to determine similar regions between two DNA, RNA, proteins or amino acids sequences.

There are two stages in the S-W algorithm. These are the similarity matrix (H matrix) fill and the trace back. In the first stage a matrix is filled with a similarity score for each element of the sequences. The second stage finds the maximum score of the matrix and performs the trace back to find the best local alignment. The first stage of the algorithm will consume the largest part of the total computation time.

One approach used to get high quality results in a short processing time is to use parallel processing on a reconfigurable system (FPGA) to accelerate the H matrix fill stage of the S-W algorithm. The maximum score of the matrix is then transferred to a GPP and the trace back is performed to get the optimal alignment.

2. Smith-Waterman algorithm

The Smith-Waterman algorithm is an optimal local sequence alignment algorithm that uses dynamic programming. Several alignment models can be used by the S-W algorithm. A simple model of the algorithm is the *Linear Gap Penalty* (LGP) model. In this model there is a score penalty (α) for a gap in the alignment of the sequences, the value of the score penalty is linear and defined by the user of the algorithm.

The algorithm uses a substitution matrix (Sbt matrix) that represents the similarity between elements. The matrix positions have a value of -1 if the elements are different and 2 if the elements are equal.

Using two sequences of size N and M the H matrix can be computed using the following expression:

$$H(i,j) = max\begin{cases} 0 \\ H(i-1,j) - \alpha \\ H(i,j-1) - \alpha \\ H(i-1,j-1) + Sbt(S1i, S2j) \end{cases}, \quad (1)$$

for $1 \le i \le N$, $1 \le j \le M$.

$$H(i,0) = H(0,j) = 0 \quad \text{for } 0 \le i \le N,\ 0 \le j \le M,$$

where *i* and *j* represent the element position of the sequences under evaluation. More information on S-W algorithm can be found in [1][2].

The regular computation requires an initialization of the first column and the first line filled with zero value, as presented in Fig. 1, where each cell is computed with equation (1).

Sequence 1: ATGCTGAC
Sequence 2: CGATCGAT

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

Fig. 1 – H matrix for sequence 1 and sequence 2.

Since the biological sequences to be aligned may be too long to be processed in fully paralleled hardware the proposed architecture will be adapted to include the possibility to divide the computation of the H matrix. This division uses the initialization values of the matrix as is show on the following example.

When splitting the computation of the matrix using, for example 4 partitions, the regular computation is repeated 4 times as presented in Fig. 2.

1

		A	T	G	C
	0	0	0	0	0
C	0	0	0	0	2
G	0	0	0	2	1
A	0	2	1	1	1
T	0	1	4	3	2

3

		T	G	A	C
	0	0	0	0	0
C	2	1	0	0	2
G	1	1	3	2	1
A	1	0	2	5	4
T	2	3	2	4	4

2

		A	T	G	C
	0	1	4	3	2
C	0	0	3	3	5
G	0	0	2	5	4
A	0	2	1	4	4
T	0	1	4	3	3

4

		T	G	A	C
	2	3	2	4	4
C	5	4	3	3	6
G	4	4	6	5	5
A	4	3	5	8	7
T	3	6	5	7	7

Fig. 2 – Divided computation of H matrix for sequence 1 and sequence 2.

Each computation inherits the line and column of previous computations as its own initialization line and column. Using this implementation is possible to obtain the exact same score result of H matrix. More information on H matrix partition computation can be found in [3].

For this application it will be used a simple trace back function [4]. This function finds the maximum score position in the H matrix and recalculates expression (1) for that position, this time evaluating from which cell the result derivate from. As show in expression (1), each cell result can only come from 3 cells, the up neighbor cell, the left neighbor cell or the up-left neighbor cell. With this information the traceback function will then move to the cell that generated the result and perform again the same operation. This will continue until the score from the cell that generated the result is zero.

The example in Fig. 3 illustrates the trace back function working for sequence 1 and sequence 2.

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

Fig. 3 – Trace back for sequence 1 and sequence 2.

From the trace back in Fig. 3 results that the best local alignment with a score of 8 is:

ATGCTGA
AT- C -GA

To parallelize the H matrix fill in the S-W algorithm it is necessary to respect the data dependency. Through expression (1) is possible to realize that iteration (i,j) cannot be executed until iterations (i-1,j), (i,j-1) and (i-1,j-1) are executed first due to data dependencies. However if the elements are calculated on different time cycles it is possible to execute several calculus in the same time cycle as show in Fig. 4.

		A	T	G	C	T	G	A	C
	0	0	0	0	0	0	0	0	0
C	0	PE1, C1	PE2, C2	PE3, C3	PE4, C4	PE5, C5	PE6, C6	PE7, C7	PE8, C8
G	0	PE1, C2	PE2, C3	PE3, C4	PE4, C5	PE5, C6	PE6, C7	PE7, C8	PE8, C9
A	0	PE1, C3	PE2, C4	PE3, C5	PE4, C6	PE5, C7	PE6, C8	PE7, C9	PE8, C10
T	0	PE1, C4	PE2, C5	PE3, C6	PE4, C7	PE5, C8	PE6, C9	PE7, C10	PE8, C11
C	0	PE1, C5	PE2, C6	PE3, C7	PE4, C8	PE5, C9	PE6, C10	PE7, C11	PE8, C12
G	0	PE1, C6	PE2, C7	PE3, C8	PE4, C9	PE5, C10	PE6, C11	PE7, C12	PE8, C13
A	0	PE1, C7	PE2, C8	PE3, C9	PE4, C10	PE5, C11	PE6, C12	PE7, C13	PE8, C14
T	0	PE1, C8	PE2, C9	PE3, C10	PE4, C11	PE5, C12	PE6, C13	PE7, C14	PE8, C15

Fig. 4 – H matrix example with indication on which PE and cycle the cell score is computed.

As is shown in Fig. 4 it is possible that all elements in the anti-diagonal can be computed in the same cycle (e.g. cycle 8). This parallel execution is called dataflow implementation [2], as all the computations are executed when their data dependencies are available.

This dataflow allows that the computation of the H matrix can be achieved using a chain of PEs. In each PE it will be computed a column of the H matrix and each cell computation will be streamed to the next PE.

3. Coreworks® Processing Engine

The objective of using Coreworks® processing engine is to accelerate, using hardware, the most compute intensive parts of the algorithm. In this case it will be the computation of the H matrix to determine the maximum score value of the alignment.

The Coreworks® processing engine has two major processing elements: the FireWorks™, a Harvard RISC (Reduced Instruction Set Computer) architecture 32-bit processor, and the SideWorks™, a reconfigurable hardware accelerator architecture.

The FireWorks™ is used to control the accelerator configurations and data transfer from/into the GPP and from/into the hardware accelerator.

The SideWorks™ is a reconfigurable architecture for hardware acceleration, which will also be implemented in the FPGA. This architecture uses Functional Units (FUs) to build datapaths as shown on the generic SideWorks™ template presented on Fig. 5. These FUs can be as simple as adders or registers to some more complex FUs. In this project one PE will be used as a FU. The reconfigurable possibilities of this accelerator allow more than one datapath to be defined in the FPGA. Therefore the user can select which datapath to use for each set of computations by control of FireWorks™. On this project only one datapath was be developed for the sequence alignment purpose.

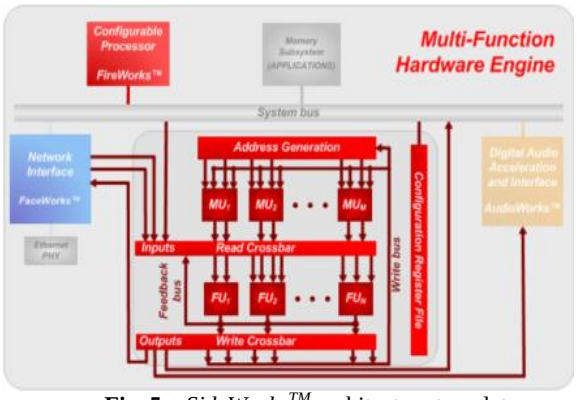

Fig. 5 – *SideWorks™* architecture template.

4. Application Overview

Our main application will run mostly on the GPP. The engine control and the hardware acceleration initialization will run on FireWorks™, but the H matrix computation will run on SideWorks™ hardware accelerator. The application runs according to the flowchart presented on Fig. 6.

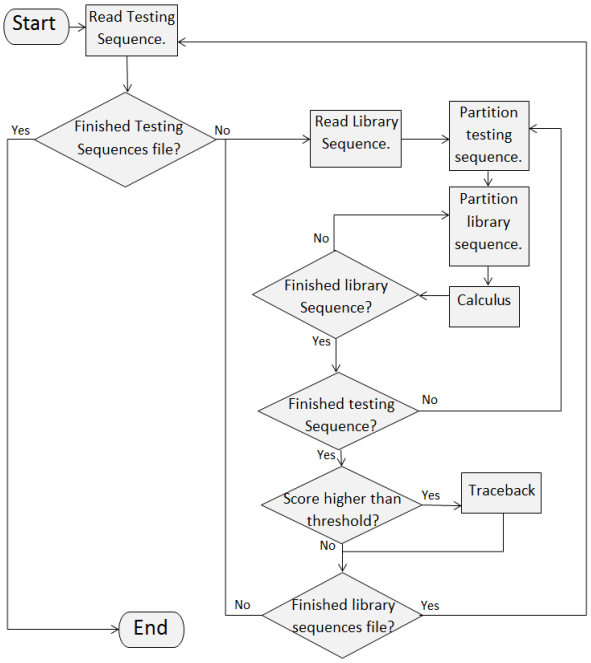

Fig 6 - Application flow chart.

The application begins by reading one sequence from the testing sequences file and one from the library sequences file. These sequences are then partitioned according to the limitations imposed that will be described in section 7. Each partition of the testing sequence will be compared to all the partitions of the library sequence before going on the next partition of the testing sequence. The computation part will end when all the partitions have ended. The result from the hardware accelerator will be the maximum score of the H matrix. As is show on Fig. 6, this value will be compared to a user defined threshold, and the trace back will only be executed if the score value is higher than the given threshold. Note that this threshold is defined in order to trace back only the sequences with high similarity values, because this increases the application efficiency.

Considering that the data transfer of the complete H matrix would take too long versus the processing time of the calculus, the traceback function rebuilds the H matrix until the computed score is equal to the score returned by the hardware and then starts the trace back itself. This option allows that, most of the times, the H matrix is not completely recalculated in software. Once the location of the maximum score is found a trace back is performed and the sequences local alignment is saved in a results file. Therefore the results file will have, for each comparison, the sequences being tested, the maximum score and the best local alignment.

The application ends after each of the sequences in the testing sequences file is compared to all the sequences in the library file.

5. Datapath Implementation on the SideWorks™

As mentioned before, the datapath for SideWorks™ is built using FUs. Two new FUs where developed for this application, the PE FU and the Trigger FU. All other FUs used were already developed and available on the Coreworks® development platform.

The PE FU is based on existing Process Elements [1][2][3], but modified to support partition computation and to be implemented on the SideWorks™ hardware accelerator. The PE FU will be described in more detail in section 6. The trigger FU is used to generate specific control signals used on the PEs. The Fig. 7 represents the datapath of our application for the SideWorks™.

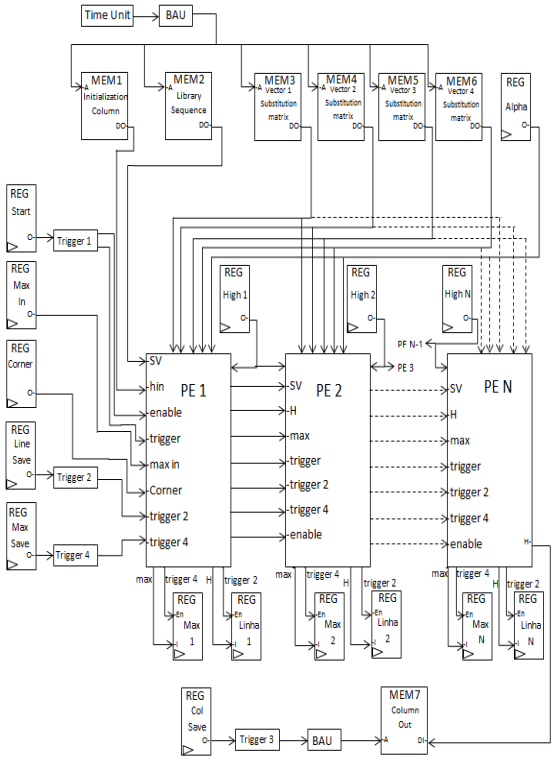

Fig 7 - SideWorks™ datapath.

The datapath is composed by a set of FUs being the most important the chain of PEs with the required FUs for input and output data. There are 6 input memory blocks. Four memory blocks are related to the data from the substitution matrix, this data is already coded according to the element from the testing sequence that is assigned to each PE. Another memory block contains the initialization column to be used only by PE1. Finally the last memory block contains the sequence from the library which will be streamed through the PEs. For each PE there is a register that contains the element from the initialization line and another register with the value of α, which can be configured by the user. For the first PE there are two additional input registers. One contains the initialization of the maximum value, in the first computation contains zero, on all other situations (e.g. computing a partition) contains the maximum score inherited by previous partitions. The other register contains the score of the top-left neighbour initialization value of a partition.

On the outputs there are two registers per PE and a memory block for the last PE. The registers will store the values of maximum score for that PE and the values from the last line that will be inherited by the next computation. The memory block will store the information for the values on the last column that will be inherited by the next computation.

The Trigger FUs are used to generate a trigger signal from an enable signal. The enable signal starts at zero and changes to one at a certain cycle, this signal is used to start the computation. At the clock cycle that enable changes to one, the trigger signal will also change to one and stays with this value only for one clock cycle. This signal is used to store some values internally in the PE. Both signals are propagated through the PEs chain.

6. Process Element Functional Unit

As previously mentioned, this FU is based on the PE described in [1], however some changes have been made to adapt the PE. Fig. 8 illustrates the resulting PE after the changes.

The main changes are related with the substitution matrix storing method and additional components to accommodate the computation of the H matrix using partitions.

There are five registers used to propagate signals to the next PE, of these, four are used for the enable and trigger signals and the remaining one is used to store and propagate the elements of the library sequence. As shown in Fig. 8 the output that results from the maximum computation and the output of H(i,j) will also be propagated to the next PE.

Each PE stores only the column of the substitution matrix related to the element of the testing sequence that has been assigned to. Therefore, each PE only needs to store 4 elements of the substitution matrix (for DNA) instead of the 16 elements that compose the full substitution matrix. The elements of the library sequence are coded using 2 bits (for DNA) and will be used to address the corresponding substitution value.

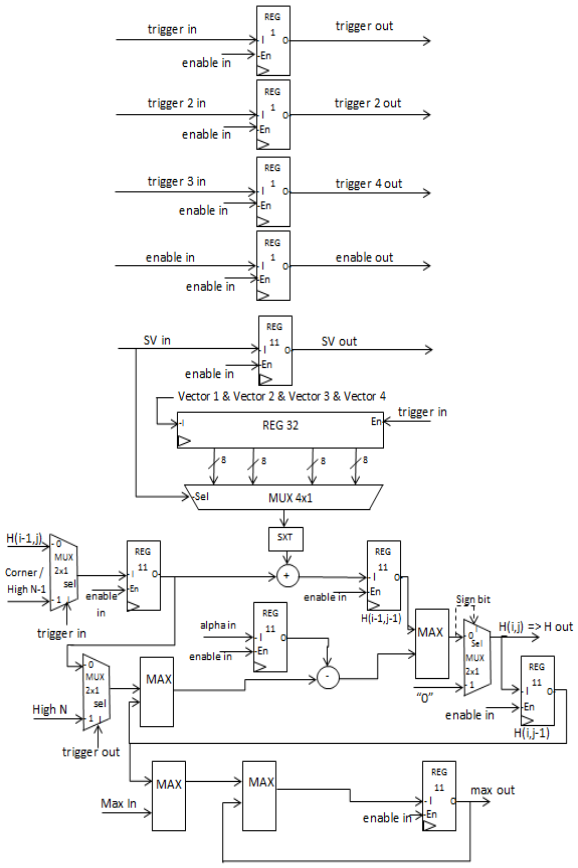

Fig 8 - Process Element Functional Unit.

The two multiplexers on the left of Fig. 8 are used for the partition of the computation of H matrix. The inputs for the multiplexer on the top left change according to the PE position on the PEs chain. In PE1 this multiplexer is used to select between the top-left neighbour initialization element and the input of the initialization column, for all other PEs this multiplexer selects between the initialization line input of a partition or the result from previous PE. The bottom left multiplexer introduces the initialization line top element.

The last multiplexer on the right is used to deal with negative values on the outcome computation of H(i,j). In these cases the output of the PE needs to be zero as shown in equation (1). Finally one register has been added to store a configurable α value.

7. Limitations

During the test of the application some limitations have been detected. The most important limitation is related to the maximum number of FUs that we are able to use in the datapath. The maximum number of PEs that has been synthesized successfully was 30, even when there is space available on the target FPGA. This number limits the partition size for testing sequences to 30 elements per partition. Another limitation is related to the vector size that can be transferred to *FireWorks*^{TM} and *SideWorks*^{TM}. It is possible to transfer vectors up to 128 elements, which limits the partition for the library sequence. For testing purposes the size of the partition used was 30 for the testing sequence and 120 for the library sequence.

The size of the registers used introduces another limitation on the maximum computable score without having overflow. All registers in the datapath have 11 bits, since the datapath uses signed calculation this limits the maximum computed score to 1023.

The results are presented with these limitations, although, there are solutions under study to improve the application.

8. Area Results

The project was implemented in a Spartan 3 XC3S5000 FPGA. Table 1 presents area results of implementations with different numbers of PEs in the Coreworks® processing engine platform.

PEs	Number of occupied slices	Total Number of 4 input LUTs
1	10 788	17 193
10	12 025	19 037
20	13 546	21 198
30	14 805	23 300

Table 1 – Areas of different implementations.

The *SideWorks*^{TM} and *FireWorks*^{TM} templates occupy a considerable amount of area (slices). Adding FUs to the *SideWorks*^{TM} template does not increase the total number of occupied slices too much. One PE alone (without *SideWorks*^{TM} overhead) occupies about 99 slices on this FPGA. However, from Table 1 is possible to average the number of occupied slices for each PE to be 134 slices. These overhead results from the extra FUs required on the data path by the *SideWorks*^{TM} platform. Therefore, for the referred FPGA, we should be able to accommodate 168 PEs on the *SideWorks*^{TM} hardware accelerator if no practical limitations exist.

9. Analysis of Application Performance

After the circuit has been synthesized and implemented using the proposed hardware on the FPGA, the minimum clock cycle attained was 27.7ns, resulting in a maximum frequency of 36MHz. Table 2 presents average load time of a configuration and different types of data transfers for 128 element vectors.

	Average (in cycles)
Configuration loading time	172
Loading a vector to memory bank in the FPGA(*Fireworks*^{TM} -> *Sideworks*^{TM})	367
Loading a vector from memory banks in FPGA(*Sideworks*^{TM} -> *Fireworks*^{TM})	244
Loading a register in the FPGA (*Fireworks*^{TM} -> *Sideworks*^{TM})	20,26
Loading a register from FPGA (*Sideworks*^{TM} -> *Fireworks*^{TM})	26
Loading a vector to the board memory banks (PC-> *Fireworks*^{TM})	13279
Loading a vector from the board memory banks (*Fireworks*^{TM} ->PC)	1004

Table 2 – Configuration and data transfer times in clock cycles.

From these results is possible to understand the impact of data transfer times on the application performance, especially with small partitions, because smaller partitions require more computations and more data transfers.

In Table 3 is presented performance of the application for different partition sizes measured in *Cell Updates Per Second* (CUPS), the number of cells from the H matrix processed per second.

Testing sequence X Library sequence	N. of partitions	Cycles per partition	Maximum computation (CUPS)	Total alignment time (µs)	Total alignment computation (CUPS)
1X1	1	72	0,5 M	15	67 K
30X30	1	104	320 M	15	60 M
60X60	2	134	498 M	30	120 M
60X120	4	192	347 M	38	189 M
120X30	1	192	694 M	15	240 M
360X120	12	196	680 M	93	464 M
510X510	85	196	655 M	470	553 M

Table 3 – Processing times in CUPS for different sequences.

From Table 3 we can calculate that the application performance increases with the size of the sequences being processed. This occurs until the sequences being tested have to be partitioned and each partition is computed by the *SideWorks*^{TM} accelerator separately. However, the performance of the total alignment computation increases with the sequences sizes.

According to [5] an optimized application (software only) has typically around 52 MCUPS average performance. Comparing this result with our results presented in Table 3 is possible to verify acceleration up to 13 times.

Other FPGA implementations of S-W algorithm achieve performances for LGP in the order of 9.2 GCUPS [1], but these results are achieved with a chain of 168 PEs and without partitions on the H matrix computation. If the limitations described for our application are solved, it is possible to achieve performances in the order of GCUPS as well.

10. Conclusions

In this work we have implemented the S-W sequence alignment algorithm using a hardware accelerator platform. The selected platform was the Coreworks processing engine, which has a RISC processor (*FireWorks*^{TM}), and a specific hardware accelerator (*SideWorks*^{TM}).

Although some practical limitations were encountered on the selected platform, we were able to implement a complete alignment application using the S-W algorithm with partitions.

The results showed that a considerable speed up was achieved even when partitions have to be used and some additional overhead is introduced by data transfer.

As future work we plan to overcome the platform limitations and implement the trace back and other parts of the algorithms in the *FireWorks*^{TM} processor.

Acknowledgements

This work was partially supported by national funds through Fundação para a Ciência e Tecnologia (FCT), under project HELIX: Heterogeneous Multi-Core Architecture for Biological Sequence Analysis (reference number PTDC/EEA-ELC/113999/2009) and by the QREN Project 3487 – Sideworks.

References

[1] T. Oliver, B. Schmidt, D. Maskell, "Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs", Proceedings ACM/SIGDA 13th International Symposium on Field-programmable Gate Arrays, 2005.

[2] P. Zhang, G. Tan, G.R. Gao, "Implementation of the Smith-Waterman Algorithm on a Reconfigurable Supercomputing Platform", Proceedings of the 1st International Workshop on High-performance Reconfigurable Computing Technology and Applications, September 2007.

[3] N. Sebastião, N. Roma, P. Flores, "Integrated Hardware Architecture for Efficient Computation of the n-Best Bio-Sequence Local Alignments in Embedded Platforms", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, July 2012.

[4] Z. Nawaz, M. Nadeem, H. van Someren, K. Bertels, "A parallel FPGA design of the Smith-Waterman traceback", International Conference on Field-Programmable Technology (FPT), December 2010.

[5] L. Hasan, Z. Al-Ars, S. Vassiliadis, "Hardware Acceleration of Sequence Alignment Algorithms – An Overview", International Conference on Design & Technology of Integrated Systems in Nanoscale Era, September 2007.