Performance Comparison between Linear RVE and Linear Systolic Array Implementations of the Smith-Waterman Algorithm

Laiq Hasan Zaid Al-Ars
Delft University of Technology
Computer Engineering Laboratory
Mekelweg 4, 2628 CD Delft, The Netherlands
Tel: +31 15 27 86172 Fax: +31 15 27 84898
L.HASAN@TUDELFT.NL

Abstract—**In this paper, we present a performance comparison between linear recursive variable expansion *(RVE)* and linear systolic array implementations of the Smith-Waterman *(S-W)* algorithm. The results demonstrate that the temporal performance of linear RVE implementation is 2.11 to 3 times better than the traditional linear systolic array implementation at the spatial cost of 2.02 to 2.54.**

Index terms: *Bioinformatics, Sequence Alignment, Smith-Waterman Algorithm, FPGAs, Systolic Arrays, Recursive Variable Expansion.*

I. Introduction

The Smith-Waterman (S-W) algorithm is a well-known algorithm in bioinformatics that finds the optimal alignment between two DNA or protein sequences (the target sequence and the search sequence) [1]. Determining how well two sequences align is important in discovering homologous genes and studying the evolutionary history of molecules and species [2]. However, the S-W algorithm is not commonly used to search sequence databases because it becomes too slow when executed against many long sequences. Instead, faster heuristic algorithms like FASTA [3] and BLAST [4] are used, even though they achieve high speed at the cost of reduced accuracy. Therefore, to achieve both increased speed and an optimal alignment, it is necessary to develop an approach to reduce the processing run time of the S-W algorithm.

Various approaches have been adopted to accelerate the S-W algorithm by implementing either the whole algorithm or some part of it in hardware [5-7], [10-12], [14-19]. In [11], the authors show the implementation of a fully custom processing unit to realize the execution of the S-W algorithm. The authors state that for conducting comparisons of multiple sequence pairs, using the same set of processing units, two approaches can be used i.e. synchronous and asynchronous. The authors claim that the asynchronous parallel approach is (k-1)*(m-1) time steps faster than the synchronous parallel approach, where k represents the size of the existing sequences in the database, which grows exponentially. In [12], an efficient cell design for systolic S-W implementation is presented. The synthesis results show that the performance of the presented design is 54.37 MCUPS. This performance gain is 1.7 times higher than a similar reference design. In [15], the authors present a new approach to bio-sequence database scanning using re-configurable FPGA-based hardware platforms to gain high performance at low cost. Their FPGA implementation achieves a speedup of approximately 170, as compared to a Pentium-IV, 1.6 GHz processor. In [18], an approach to realize high speed sequence alignment using run-time reconfiguration is proposed. With this approach, it is demonstrated that high performance can be achieved using off-the shelf FPGA boards. The performance is almost comparable with dedicated hardware systems. The time for comparing a query sequence of 2048 elements with a database sequence of 64 million elements by the S-W algorithm is about 34 sec, which is about 330 times faster than a desktop computer with a Pentium-III, 1.0 GHz processor. In [19], the design of a small fully custom processing element, called *Proclet*, is shown. This Proclet is used for a new VLSI implementation of the S-W algorithm. The results show that the design achieves a performance of 976 Kilo CUPS, but is not compared with any reference design. An overview of such approaches is given in [9]. In [20], an implementation based on systolic array architecture is presented, where systolic array is an arrangement of processors in an array (that may be either linear or rectangular), where data flows synchronously across the array between neighbours. In [21], a hardware implementation of the S-W algorithm using RVE approach is presented and its performance is compared with an equivalent rectangular systolic array implementation. The results demonstrate that applying the recursive variable expansion technique speeds up the performance by a factor of 1.36 to 1.41, as compared to traditional acceleration approaches at the cost of using 1.25 to 1.28 times more hardware resources. But the main problem with this RVE implementation is that the hardware is underutilized most of the times.

In this paper, we present a linear implementation of the S-W algorithm based on recursive variable expansion approach and its comparison with a linear implementation based

traditional systolic array approach. The results demonstrate that the linear implementation based on RVE approach is 2.11 to 3 times faster than the linear implementation based on a systolic array approach at the cost of utilizing 2.02 to 2.54 times more resources.

The remainder of the paper is organized as follows: Section 2 provides a brief description of the S-W algorithm, data dependencies in the Hi,j matrix and a brief introduction of the RVE approach. Section 3 presents our linear implementation based on systolic array approach. Section 4 presents the linear implementation based on RVE approach. Section 5 discusses and compares the results obtained from the two implementations presented in Section 3 and Section 4. Section 5 gives a brief conclusion.

II. THE S-W ALGORITHM

Based on *dynamic programming (DP)* [8], the S-W algorithm [1] is a method used for local sequence alignment (i.e., identifying common regions in sequences that share local similarity characteristics). In the following subsections, we give a brief description of the algorithm and its inherent data dependencies.

A. S-W Description

When calculating the local alignment, a matrix Hi,j is used to keep track of the degree of similarity between the two sequences to be aligned (Ai and Bj). Each element of the matrix Hi,j is calculated according to the following equation:

$$H(i,j) = max\begin{cases} 0 \\ H(i-1,j-1) + Si,j \\ H(i-1,j) - d \\ H(i,j-1) - d \end{cases} \quad (1)$$

where Si,j is the similarity score of comparing sequence Ai to sequence Bj and d is the penalty for a mismatch.

The whole algorithm is divided into the following three steps:

1. Initialization step
2. Matrix fill step
3. Trace back step

The matrix is first initialized with $H0,j = 0$ and $Hi,0 = 0$ for all i and j. This is referred to as the *initialization step*. After the initialization, a *matrix fill step* is carried out using Equation 1, which fills out all entries in the matrix. The final step is the *trace back step*, where the scores in the matrix are traced back to inspect for optimal local alignment. The trace back starts at the cell with the highest score in the matrix and continues up to the cell, where the score falls down to a predefined minimum threshold. In order to start the trace back, the algorithm requires to find the cell with the maximum value, which is done by traversing the entire matrix.

The time complexity of initialization step is $O(M+N)$. During the matrix fill step, the entire Hi,j matrix needs to be filled according to Equation 1, making its time complexity equal to the number of cells in the matrix or $O(M+N)$. The time complexity of the traceback is also $O(M+N)$, as the entire matrix needs to be traversed during this step. Thus the total time complexity of the S-W algorithm is $O(M+N) + O(MN) + O(MN) = O(MN)$. The total space complexity of the S-W algorithm is also $O(MN)$, as it fills a single matrix of size MN.

B. Data Dependencies in the Hi,j Matrix

In order to reduce the $O(MN)$ complexity of the matrix fill stage, multiple entries of the Hi,j matrix are calculated in parallel. This is however complicated by data dependencies, whereby each Hi,j entry depends on the values of three neighbouring entries $Hi,j-1$, $Hi-1,j$ and $Hi-1,j-1$, with each of those entries in turn depending on the values of three neighbouring entries, which effectively means that this dependency extends to every other entry in the region Hi,j $Hx,y : x \leq i, y \leq j$. This implies that it is possible to simultaneously compute all the elements in each anti diagonal, since they fall outside each other's data dependency regions. Figure 1 shows a sample Hi,j matrix for two sequences, with the bounding boxes indicating the elements that can be computed in parallel. The right bottom cell is highlighted to show that its data dependency region is the entire remaining matrix. The dark diagonal arrow indicates the direction in which the computation progresses. At least 9 cycles are required for this computation, as there are 9 bounding boxes representing 9 anti diagonals and a maximum of 5 cells may be computed in parallel.

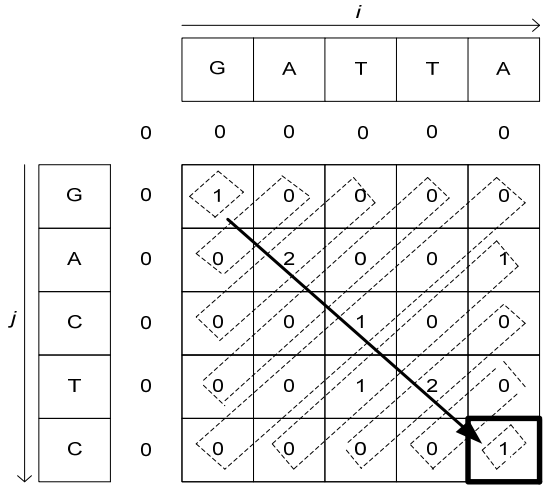

Figure 1: A sample Hi,j matrix, showing the parallelization possibilities within the S-W algorithm

C. Recursive Variable Expansion

RVE [13] is a kind of loop transformation which removes all data dependencies from a program, so that the program is parallelized to its maximum. The basic idea is that if any statement Gi is dependent on statement Hj for some iteration i and j, then instead we wait for Hj to complete and then execute Gi, we will replace all the occurrences of the variable in Gi that create dependency with Hj with the computation of that variable in Hj. In this way there is no need to wait for the statement Hj to complete and statement Gi can be executed independently of Hj. This step is recursively repeated until

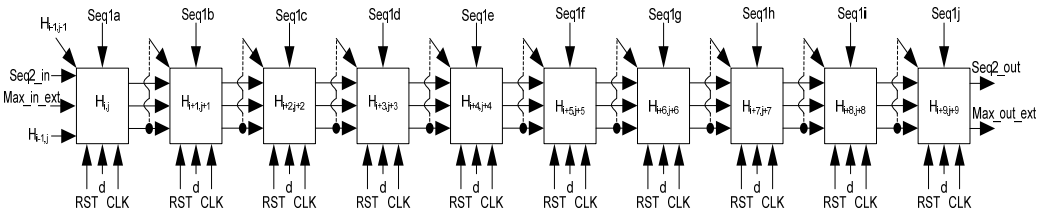

Figure 2: Linear systolic array implementation of the S-W algorithm using 10 elements

the statement Gi is not dependent on any other statement, other than inputs or known values, which essentially means that Gi can be computed without any delays. This transformation is explained clearly in Example 1, which adds the loop counter.

Example 1: A simple example which adds the loop counter

$A[1] = 1$
for $i = 2\ to\ 5$
$A[i] = A[i-1] + i ---------- (Gi)$
end for

Therefore after applying the RVE, we get an expression with five terms to be added, as shown in Example 2.

Example 2: After applying RVE on Example 1

$$A[5] = A[4] + 5$$
$$= A[3] + 4 + 5$$
$$= A[2] + 3 + 4 + 5$$
$$= A[1] + 2 + 3 + 4 + 5$$
$$= 1 + 2 + 3 + 4 + 5$$

In this way, the whole expanded statement in Example 2 can be computed in any order by computing the large number of operations in parallel and efficiently. The major drawback of this technique is that the speed up is achieved at the cost of redundancy, which consumes a lot of resources. The RVE approach is discussed in detail in [14].

III. Linear Systolic Array Implementation

Systolic array is an arrangement of processors in an array where data flows synchronously across the array between neighbours. It can be either linear or rectangular. Linear systolic array is used to compute the elements of Hi, j matrix in the S-W algorithm [12, 15]. Figure 2 shows a 10 elements linear systolic array, where each element is composed of a cell design as shown in Figure 3.

In the cell design of Figure 3, Comp1 compares the corresponding characters of the two input sequences and generates a similarity score. The similarity score is equal to the match score if the corresponding characters are similar, otherwise it is equal to the mismatch score. The diagonal element $Hi-1, j-1$ is delayed by Buf1 for one clock cycle, as it comes from the previous element in the array. Add1 adds the similarity score with the delayed diagonal element. Comp2 compares the output of Add1 with a 0.

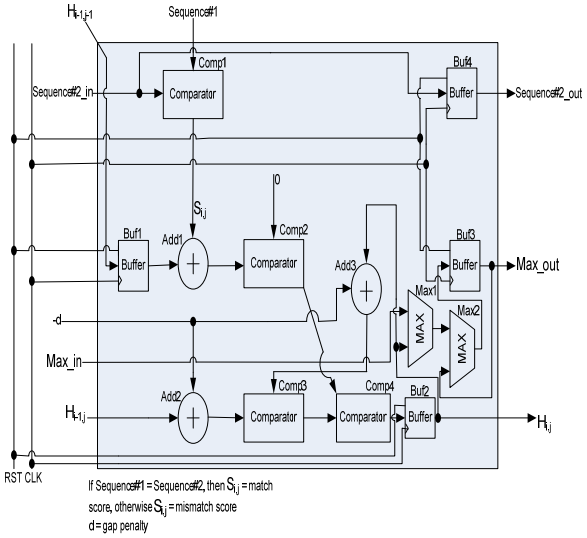

Figure 3: Cell design for the linear systolic array implementation of the S-W algorithm

Add2 adds the left element $Hi-1, j-1$ with the gap penalty. Add3 adds the up element (which is the current value of the cell) with the gap penalty. Comp3 compares the outputs of Add2 and Add3. Comp4 compares the outputs of Comp2 and Comp3. Buf2 keeps the output of the cell and also feeds it back to Add3 and Max1, where Max1 compares the current value of the cell with the external Max_in input. Max2 compares the output of Max1 with the previous max value. The output of Max2 is stored back in Buf3. Buf4 delays the Sequence#2_in input by one clock cycle for the next element of the array. The cell design of Figure 3 is used as a building block for implementation of the linear systolic array shown in Figure 2. The array shown in Figure 2 is implemented in VHDL and the post place and route simulation results show that for a clock period of 100 ns, the latency of the linear systolic array is 1900 ns, whereas the slices utilized are 297 out of 13696. The platform used for implementation is Xilinx Virtex II Pro. Apart from the design of Figure 2, a 2 elements linear systolic array is also implemented for comparison with the linear RVE block, to be discussed in the next section. For a clock cycle of 100 ns, the latency of the 2 elements linear systolic array is 300 ns, whereas the slices utilized are 50 out of 13696.

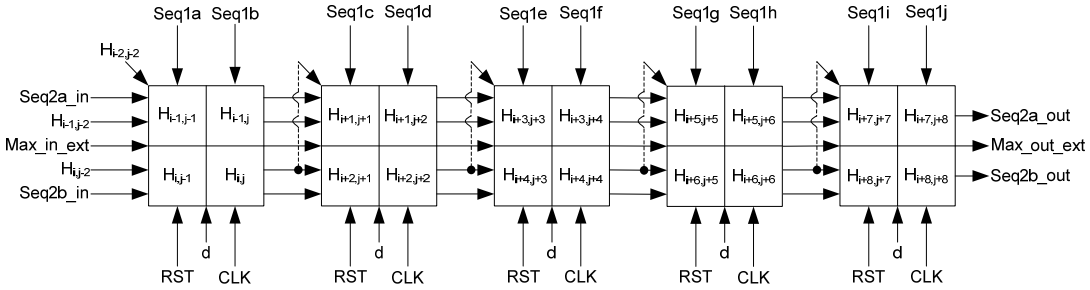

Figure 5: 5 blocks linear RVE design

IV. LINEAR RVE IMPLEMENTATION

This section presents our implementation of the S-W algorithm based on linear RVE design. We define the size of RVE block as the *blocking factor (b)*, such that for a 2x2 array, implemented using linear RVE design, the blocking factor b = 2. Figure 4 shows the block diagram representation of our linear RVE design with b = 2.

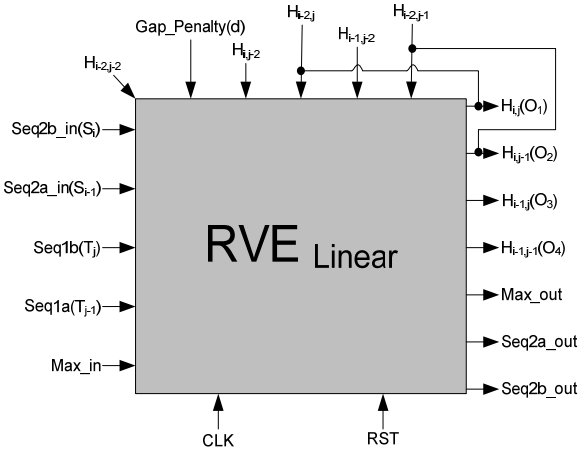

Figure 4: Block diagram representation of the linear RVE design with b=2

This design is implemented in VHDL and the post route simulation shows that for a clock period of 100 ns, the latency is 1 clock cycle. The slices utilized are 127 out of 13696. The device utilized is Xilinx Virtex II Pro. Using this linear RVE design as a building block, we implemented a 5 blocks linear RVE array as shown in Figure 5. This is equivalent to the 10 elements linear systolic array implementation.

Figure 6 gives the logical description of our linear RVE design with b = 2. The comparators in the 1st column of the figure compares the corresponding characters of the input sequences and generates the similarity score accordingly, whereas the buffer in the 1st column delays the element $Hi-2, j-2$ by one clock cycle. The adders in the 2nd column adds the gap penalty with the elements $Hi, j-2$, $Hi-1, j-2$, Hi, j and $Hi, j-1$, where the 1st two are external elements and the second two are feedback elements. The AND gates in the 3rd column performs logic anding between the outputs of the upper 3 comparators in the 1st column. The adders and comparators in the following columns performs addition and max operation on the inputs from the preceding columns. The values of the five outputs Hi, j, $Hi, j-1$, $Hi-1, j$, $Hi-1, j-1$ and Max_out are stored in registers named BUF. Seq2a_in and Seq2b_in are delayed by one clock cycle using buffers in the last column to get Seq2a_out and Seq2b_out for the next block in the array. The 5 blocks linear RVE design shown in Figure 5 is implemented in VHDL and the post place and route simulation results show that for a clock period of 100 ns the latency of the array is 900 ns, where as the slices consumed are 601 out of 13696. The platform used for implementation is Xilinx Virtex II Pro.

Table 1: Filled matrix obtained using linear systolic array and linear RVE implementations

		A	**G**	**T**	**A**	**A**	**G**	**T**	**A**	**C**	**A**
	0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
G	*0*	**0**	2	2	2	2	2	2	2	2	2
G	*0*	0	**2**	2	2	2	4	4	4	4	4
T	*0*	0	2	**4**	4	4	4	6	6	6	6
C	*0*	0	2	**4**	**4**	4	4	6	6	8	8
G	*0*	0	2	4	**4**	**4**	6	6	6	8	8
G	*0*	0	2	4	4	4	**6**	6	6	8	8
T	*0*	0	2	4	4	4	6	**8**	8	8	8
C	*0*	0	2	4	4	4	6	**8**	8	10	10
A	*0*	2	2	4	6	6	6	8	**10**	10	12
C	*0*	2	2	4	6	6	6	8	10	**12**	12

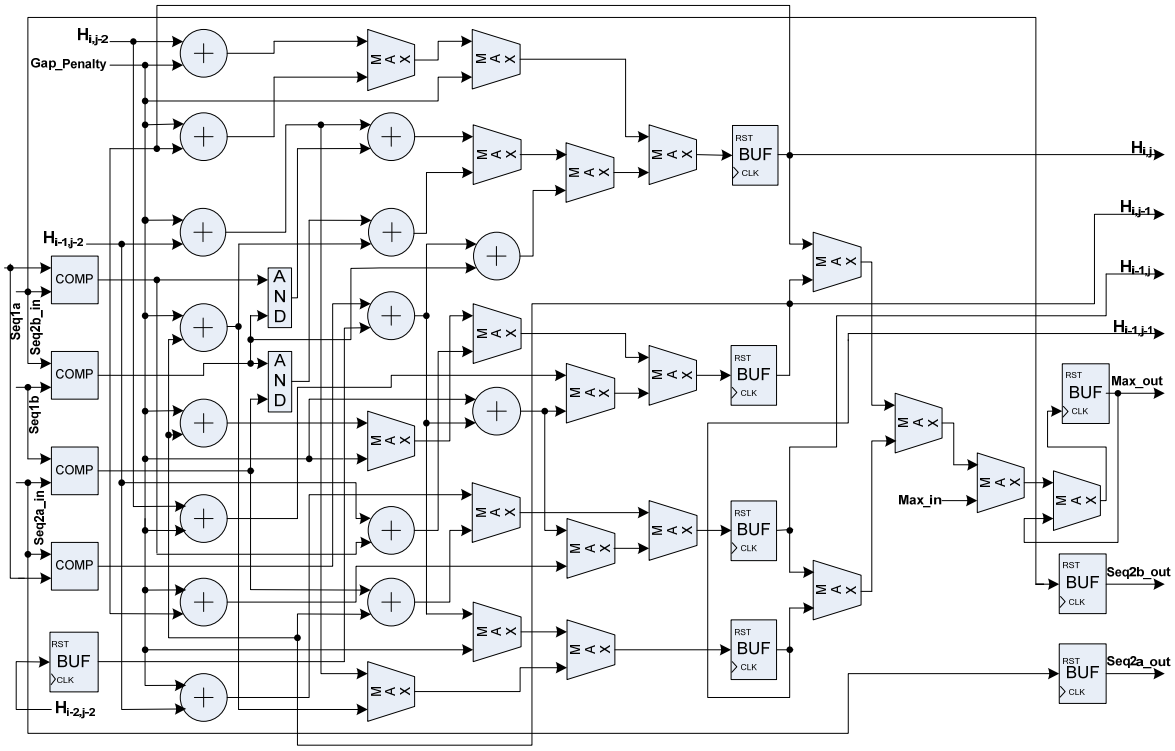

Figure 6: Logical description of the linear RVE design with b=2

Table 2: Comparison between linear systolic array and linear RVE implementations

Implementation	Time consumed	Clock frequency	Speedup w.r.t. linear systolic array implementation	Number of slices	Cost
2 elements linear systolic array	300 ns	10 MHz	1	50 out of 13696	1
Single block linear RVE	100 ns	10 MHz	3	127 out of 13696	2.54
10 elements linear systolic array	1900 ns	10 MHz	1	297 out of 13696	1
5 blocks linear RVE	900 ns	10 MHz	2.11	601 out of 13696	2.02

V. DISCUSSION OF RESULTS

Table 1 presents the filled matrix obtained using the 10 elements linear systolic array implementation and 5 blocks linear RVE implementation. The same input sequences are used in both the cases and the same correct results verify the correctness of both the designs. The bold digits in Table 1 indicate the trace back path.

Table 2 summarizes the results presented in Section 3 and Section 4. It demonstrates that the 2 elements linear systolic array implementation consumes 300 ns with a 10 MHz clock frequency and utilizes 50 out of 13696, when implemented on a Xilinx Virtex II Pro FPGA. The speedup and cost is 1, as the reference for comparison is the same linear systolic array design. The single block linear RVE implementation consumes 100 ns for a 10 MHz clock and utilizes 127 out of 13696 slices, while using the same Virtex II Pro device. Thus the single block linear RVE design is 3 times faster than the two elements linear systolic array design at the cost of 2.54 times more hardware resource utilization.

The table further demonstrates that the 10 elements linear systolic array implementation consumes 1900 ns for a 10 MHz clock frequency and utilizes 297 out of 13696 slices, when implemented on a Xilinx Virtex II Pro FPGA. The speedup and cost is 1, because the reference for comparison is the same linear systolic array design, which is traditionally used for accelerating the S-W algorithm. The 5 blocks linear RVE implementation consumes 900 ns for a 10 MHz clock frequency and utilizes 601 out of 13696 slices, when implemented on a Xilix Virtex II Pro FPGA. Thus in comparison with a 10 elements traditional linear systolic array implementation, the 5 blocks linear RVE implementation improves the performance by a factor of 1900/900 = 2.11 at the cost of utilizing 601/297 = 2.02 times more resources.

VI. Conclusion

In this paper, we presented a new implementation of the S-W algorithm based on the linear RVE approach and compared its performance with a traditional linear systolic array implementation. The results demonstrate that the linear RVE implementation is 2.11 to 3 times faster than the traditional linear systolic array implementation at the cost of utilizing 2.02 to 2.54 times more hardware resources. The results lead to the conclusion that the implementation based on linear RVE approach is preferred to any other approach, in cases where hardware resources utilization cost is not a big concern.

Acknowledgment

This work is financially supported by the Computer Engineering Laboratory TU Delft and the Higher Education Commission of Pakistan.

References

[1] Smith T. F. and Waterman M. S., "Identification of Common Molecular Subsequences", *In Journal of Molecular Biology*, vol. 147, pp 195-197, 1981.

[2] Page RD, "GeneTree: Comparing Gene and Species Phylogenies using Reconciled Trees", *Bioinformatics*, vol. 14(9), pp 819–820, 1998.

[3] Pearson W. R. and Lipman D. J., "Rapid and Sensitive Protein Similarity Searches", *In Science*, vol. 227, pp 1435-1441, 1985.

[4] Altschul S. F. et al, "Basic Local Alignment Search Tool", *In Journal of Molecular Biology*, vol. 215, pp 403-410, 1990.

[5] Blas A. Di. et al, "The UCSC Kestrel Parallel Processor", *In IEEE Transactions on Parallel and Distributed Systems*, vol. 16(1), pp 80–92, 2005.

[6] Borah M. et al, "A SIMD Solution to the Sequence Comparison Problem on the MGAP", *Proceedings of the International Conference on Application Specific Array Processors*, San Francisco, California, USA, 1994.

[7] Chiang J. et al, "Hardware Accelerator for Genomic Sequence Alignment", *Proceedings of the 28th IEEE EMBS Annual International Conference*, New York City, USA, Aug 30-Sept 3, 2006.

[8] Giegerich R., "A Systematic Approach to Dynamic Programming in Bioinformatics", *Bioinformatics*, vol. 16, pp 665-677, 2000.

[9] Hasan L. et al, "Hardware Acceleration of Sequence Alignment Algorithms - An Overview", *Proceedings of International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS'07)*, pp 96-101, Rabat, Morocco, September 2–5, 2007.

[10] Laiq Hasan and Zaid Al-Ars, "Performance Improvement of the Smith-Waterman Algorithm", *Annual Workshop on Circuits, Systems and Signal Processing (ProRISC)*, Veldhoven, The Netherlands, November 29–30, 2007.

[11] Liao H. Y. et al, "A Parallel Implementation of the Smith-Waterman Algorithm for Massive Sequences Searching", *Proceedings of the 26th Annual International Conference of the IEEE EMBS*, San Francisco, CA, USA, September 1–5, 2004.

[12] Mustafa Gok and Caglar Yilmaz, "Efficient Cell Designs for Systolic Smith-Waterman Implementation" *Proceedings of the 16th International Conference on Field Programmable Logic and Applications (FPL)*, Meliá Madrid Princesa, Madrid, SPAIN, 2006.

[13] Nawaz Z. et al, "Recursive Variable Expansion: A Loop Transformation for Reconfigurable Systems", *Proceedings of InternationalConference on Field-Programmable Technology (FPT)*, Kokurakita, Kitakyushu, JAPAN, December 2007.

[14] Nawaz Z. et al, "Acceleration of Smith-Waterman Using Recursive Variable Expansion", *Proceedings of 11th Euromicro Conference on Digital System Design (DSD)*, Parma, Italy, September 2008.

[15] Oliver T. et al, "Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs", *FPGA'05*, Monterey, California, USA, February 20-22, 2005.

[16] Schroder A. et al, "Bio-Sequence Database Scanning on a GPU", *Proceedings of the Fifth IEEE International Workshop on High Performance Computational Biology (HICOMB)*, Rhodes Island, Greece, 2006.

[17] Steve Margerm, Cray Inc, "Reconfigurable Computing in Real-World Applications", *FPGA and Structured ASIC Journal (www.fpgajournal.com)*, February 7, 2006.

[18] Yamaguchi Y. et al, "High Speed Homology Search Using Run-Time Reconfiguration", *Proceedings of the 12th International Conference on Field Programmable Logic and Application (FPL)*, Montpellier (La Grande-Motte) – France, 2002.

[19] Yang B. H. W., "A Parallel Implementation of Smith-Waterman Sequence Comparison Algorithm", *Technical Report, Biochemistry department, Stanford University*, USA, December 6, 2002.

[20] L. Hasan, Y.M. Khawaja, A. Bais, "A Systolic Array Architecture for the Smith-Waterman Algorithm with High Performance Cell Design", *Proceedings of IADIS European Conference on Data Mining*, Amsterdam, The Netherlands, July 2008.

[21] L. Hasan, Z. Al-Ars, Z. Nawaz, K.L.M. Bertels, "Hardware Implementation of the Smith-Waterman Algorithm Using Recursive Variable Expansion", *Proceedings of 3rd International Design and Test Workshop IDT08*, Monastir, Tunisia, December 2008.