A SYSTOLIC ARRAY ARCHITECTURE FOR THE SMITH-WATERMAN ALGORITHM WITH HIGH PERFORMANCE CELL DESIGN

Laiq Hasan
Computer Engineering Laboratory, Delft University of Technology (TU Delft)
Mekelweg 4, 2628 CD Delft, The Netherlands
L.HASAN@EWI.TUDELFT.NL

Yahya M. Khawaja*
Abdul Bais*
* *Department of Computer Systems Engineering*
N-W.F.P. University of Engineering & Technology Peshawar, Pakistan

ABSTRACT

To infer homology and subsequently gene function, the *Smith-Waterman (S-W)* algorithm is used to find the optimal local alignment between two sequences. When searching sequence databases that may contain hundreds of millions of sequences, this algorithm becomes computationally expensive. In this paper, we present a systolic array architecture for the S-W algorithm with a new high performance cell design. The results demonstrate that the implementation of this architecture achieves a speedup of up to 652*x*, as compared to a software-only implementation, which is almost double the best case reported in the literature. The results also demonstrate that when mapped on the same FPGA platform, our design performs 1.47 to 5.75 times faster in terms of Cell Updates Per Second (CUPS), in comparison with other published systolic array designs, while utilizing 3.69 to 6.36 times less resources.

KEYWORDS

Bioinformatics, Sequence Alignment, Dynamic Programming, Smith-Waterman Algorithm, FPGAs, Systolic Arrays.

1. 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) (Smith T. F. & Waterman M. S. 1981). Determining how well two sequences align is important in discovering homologous genes and studying the evolutionary history of molecules and species (Page RD. 1998). 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 (Pearson W. R. & Lipman D. J. 1985) and BLAST (Altschul S. F. et al 1990) are used, even though they achieve high speed at the cost of reduced accuracy. Therefore, to achieve both increased speed and the optimal alignment, it is necessary to develop an approach to reduce the processing 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 (Yamaguchi Y. et al 2002, Chiang J. et al 2006, Steve Margerm 2006, L.Hasan 2007, Blas A. Di. et al 2005, Schroder A. et al 2006, Borah M. et al 1994, M Gok et al 2006, Liao H. Y. et al 2004, Yang B. H. W. 2002, Oliver T. et al 2005). In (Yamaguchi Y. et al 2002), 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 (M Gok 2006), 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 (Liao H. Y. et al 2004), 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 (Yang B. H. W. 2002), 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. In (Oliver T. et al 2005), 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. An overview of such approaches is given in (Hasan L. et al 2007).

In this paper, we present a systolic array architecture for the S-W algorithm with high performance cell design. The results demonstrate that the implementation of this architecture achieves a speedup of up to 652x, as compared to a software-only implementation, which is almost double the best case reported in the literature. Also our cell design for the systolic array architecture reduces the amount of resources utilized and improves the performance in terms of CUPS, as compared to similar work reported earlier.

The remainder of the paper is organized as follows: Section 2 provides a brief description of the S-W algorithm and data dependencies in the $H_{i,j}$ matrix. Section 3 presents our systolic array architecture, its implementation and comparison measures for performance evaluation. Section 4 discusses the results obtained from the implementation of our design and compares them with the already published similar work. Section 5 gives a brief conclusion.

2. THE S-W ALGORITHM

Based on *dynamic programming (DP)* (Giegerich R. 2000), the S-W algorithm (Smith T. F. & Waterman M. S. 1981) 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.

2.1 S-W Description

When calculating the local alignment, a matrix $H_{i,j}$ is used to keep track of the degree of similarity between the two sequences to be aligned (A_i *and* B_j). Each element of the matrix $H_{i,j}$ is calculated according to the following equation:

$$H_{i,j} = max\{\, 0,\ (H_{(i-1,j-1)} + S_{i,j}),\ (H_{(i-1,j)} - d), (H_{(i,j-1)} - d)\,\} \quad (1)$$

where $S_{i,j}$ is the similarity score of comparing sequence A_i to sequence B_j 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 $H_{0,j} = 0$ and $H_{i,0} = 0$, 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 $H_{i,j}$ matrix needs to be filled according to Equation 1, making its time complexity equal to the number of cells in the matrix or $O(MN)$. The time complexity of the traceback is also $O(MN)$, 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.

2.2 Data Dependencies in the Hi,j Matrix

In order to reduce the $O(MN)$ complexity of the matrix fill stage, multiple entries of the $H_{i,j}$ matrix are calculated in parallel. This is however complicated by data dependencies, whereby each $H_{i,j}$ entry depends on the values of three neighboring entries $H_{i,j-1}$, $H_{i-1,j}$ and $H_{i-1,j-1}$, with each of those entries in turn depending on the values of three neighboring entries, which effectively means that this dependency extends to every other entry in the region $H_{x,y : x \le i, y \le j}$. 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 $H_{i,j}$ 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.

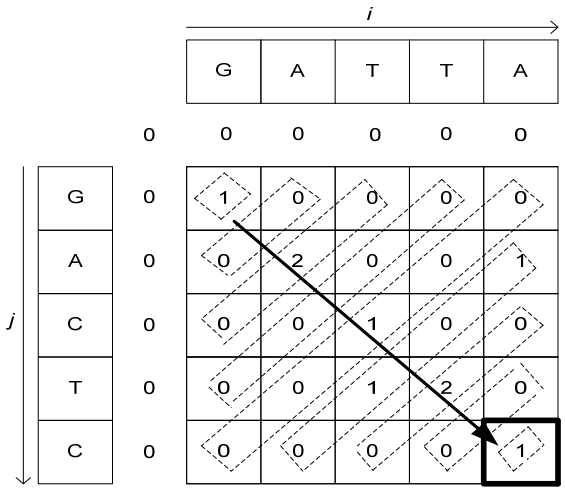

Figure 1: A sample Hi,j matrix, showing the parallelization possibilities within the S-W algorithm

3. SYSTOLIC ARRAY ARCHITECTURE

Systolic array is an arrangement of processors in an array (often rectangular) where data flows synchronously across the array between neighbors. Figure 2 is a simple example of a systolic array architecture. In this configuration there are two vector array inputs, M and N. The processing cells have a value, U_{ij}, that is usually a result due to a defined algorithm within the cells. In the following subsections, we describe comparison measures for systolic array architectures, our cell design, and the subsequent construction of a sample systolic array architecture.

3.1 Comparison Measures

To compare the efficiency of similar systolic architectures various comparison measures are used, as listed below.

- Performance is often measured in terms of CUPS, which stands for cell updates per second and is

quantitatively equivalent to the ratio of the number of cells in the array to the total time consumed, i.e. CUPS = $\frac{Number\ of\ cells}{Total\ time\ consumed}$

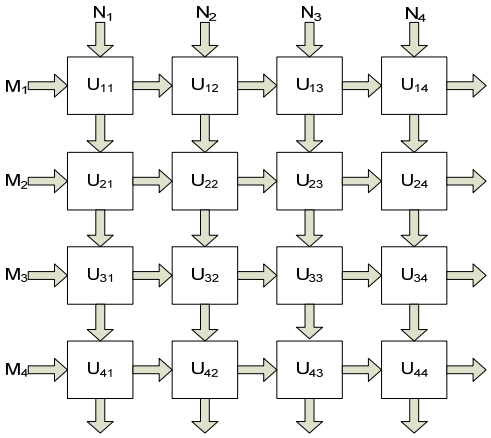

Figure 2: Simple example of a systolic array architecture

Alternatively, it can also be defined as the product of the total number of cells in the array and the clock frequency used, i.e. CUPS = $Number\ of\ cells * Clock\ frequency$.
Practical cell performance is usually measured using KCUPS, MCUPS and GCUPS. CUPS is an important comparison measure, as it tells us about the efficiency of a particular design in comparison with other similar designs.

- The speed of a particular design in terms of frequency or absolute time consumed, is also often used as an alternative measure for comparing the performance of two similar designs.
- Resource utilization is an important comparison measure, which refers to the amount of hardware used in implementing a given design. In case of FPGAs, resource utilization is often measured in terms of slices, where a slice contains look-up tables (LUTs), registers, and dedicated carry logic. The number of slices utilized, represent the amount of hardware used by a particular design, as FPGAs are divided into fixed number of slices, e.g. 13696 slices for Virtex II Pro.

3.2 High Performance Cell Design

The performance of systolic array architecture for the SW algorithm mainly depends on how simplified and efficient the corresponding cell design is. By efficient we mean both in terms of performance and area. Figure 3 shows the block diagram description of our cell design for computing $H_{i,j}$ values for the systolic array architecture, according to Equation 1. In Figure 3, Comp1 is a comparator that compares the two input sequences and outputs the corresponding value of $S_{i,j}$, depending on the values of the match and mismatch scores, such that $S_{i,j}$ = match score, if Sequence1 = Sequence2, otherwise $S_{i,j}$ = mismatch score. Add1 is an adder that adds the diagonal element $H_{i-1,j-1}$ and the value of $S_{i,j}$. Comp2 is a comparator that compares the output of Add1 with a constant value 0 and outputs the greater of the two numbers. Add2 is an adder that adds the left element $H_{i-1,j}$ and -d, where d is the gap penalty. Add3 is an adder that adds the upper element $H_{i,j-1}$ and -d. Comp3 compares the outputs of Add2 and Add3 and outputs the greater of the two numbers. Comp4 compares the outputs of Comp2 and Comp3 and outputs the greater of the two numbers. The output of Comp4 is stored in the register $R_{i,j}$, and is the corresponding $H_{i,j}$ value.

The block diagram shown in Figure 3 is implemented in VHDL and the post place and route simulations show that the asynchronous time consumed by such a cell is 9.8 *n*sec.

3.3 A Sample Systolic Array Architecture

The cell design shown in Figure 3 can be used to build a systolic array architecture of any size, depending on the availability of hardware resources. As a case study, we implemented a 4*x*4 systolic array architecture, as shown in Figure 4.
Table 1 shows an example of comparing two sequences, using the systolic architecture given in Figure 4.

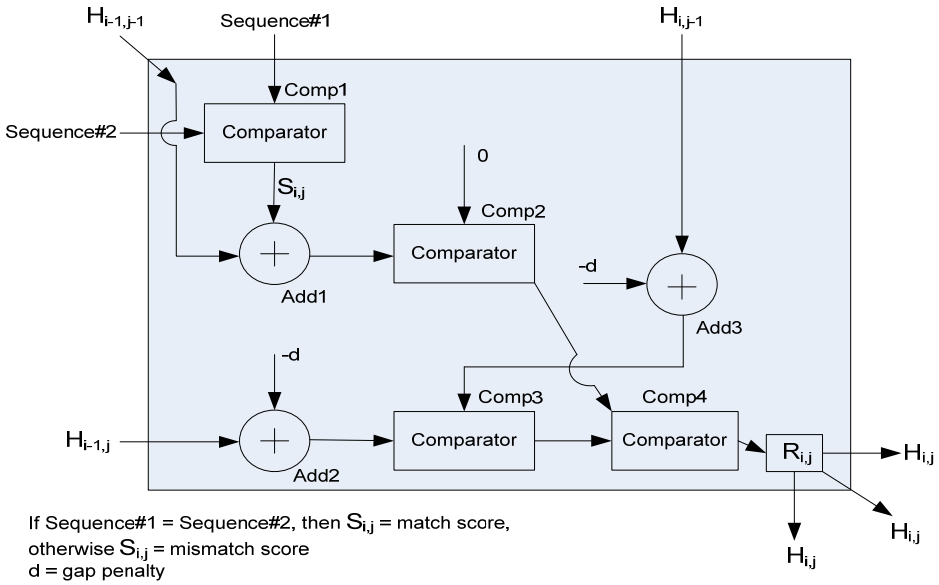

Figure 3: Block diagram of our cell design for computing Hi, j values of Equation 1

The sequences taken for comparison are named as Sequence 1 and Sequence 2, where,
Sequence 1 = A G T A and Sequence 2 = G G T C

Table 1: Hi, j matrix and the trace back path

		A	**G**	**T**	**A**
	0	*0*	*0*	*0*	*0*
G	*0*	0	2	2	2
G	*0*	0	2	2	2
T	*0*	0	2	4	4
C	*0*	0	2	4	4

The matrix is initialized with the value zero, such that $H_{0,0} = H_{0,1} = H_{0,2} = H_{0,3} = H_{0,4} = H_{1,0} = H_{2,0} = H_{3,0} = H_{4,0} = 0$. The gap penalty is assumed to have a value zero.

The remaining values of the $H_{i,j}$ matrix are computed using the systolic array structure, shown in Figure 4. The bold digits in Table 1 show the trace back path. Since the elements within each anti diagonal are independent of each other, they can be computed in parallel in the array. Therefore the time consumed by an anti diagonal is the same as the time consumed by one cell, which is 9.8 *n*sec. Furthermore, since there are 7 anti diagonals in a 4*x*4 systolic array, the speedup factor (calculating the elements in anti diagonals in parallel) = 16/7 = 2.29.

A function was written in C to compute the values of $H_{i,j}$ matrix according to Equation 1. This function is a software equivalent of the cell design shown in Figure 3. When run on a 100 MHz power PC, this function consumed 279 cycles. Thus the equivalent time consumed = 2790 *n*sec.
Time consumed by our cell design in hardware = 9.8 *n*sec. Relative speedup = 2790/9.8 = 285.
Speedup for a 4*x*4 systolic array = 285x2.29 = 652.

For computing the delay of the entire array, we run the asynchronous time simulation (post place and route simulation), which shows that the time consumed to fill a 4*x*4 array asynchronously = 26.4 *n*sec. On the other hand, the time consumed to fill a 4*x*4 systolic array synchronously = 9.8x7 = 68.6 *n*sec, showing thereby that the asynchronous approach is 2.6 times faster than the synchronous approach. This speedup is only significant in terms of computing the delay of the entire circuit, as the asynchronous approach only

outputs the final $H_{i,j}$ value. For the intermediate $H_{i,j}$ values, we have to use the synchronous approach. To evaluate the performance gain in terms of CUPS, we implemented our design on a Xilinx Virtex-II XC2V6000 FPGA, such that the available hardware resources were utilized to the maximum. In this way, we were able to fit a maximum of 1778 cells on the FPGA, where each cell utilized 19 slices. The clock frequency used for our implementation was 45 MHz and the performance thus achieved was,

Performance = 1778 * 45 = 80 GCUPS

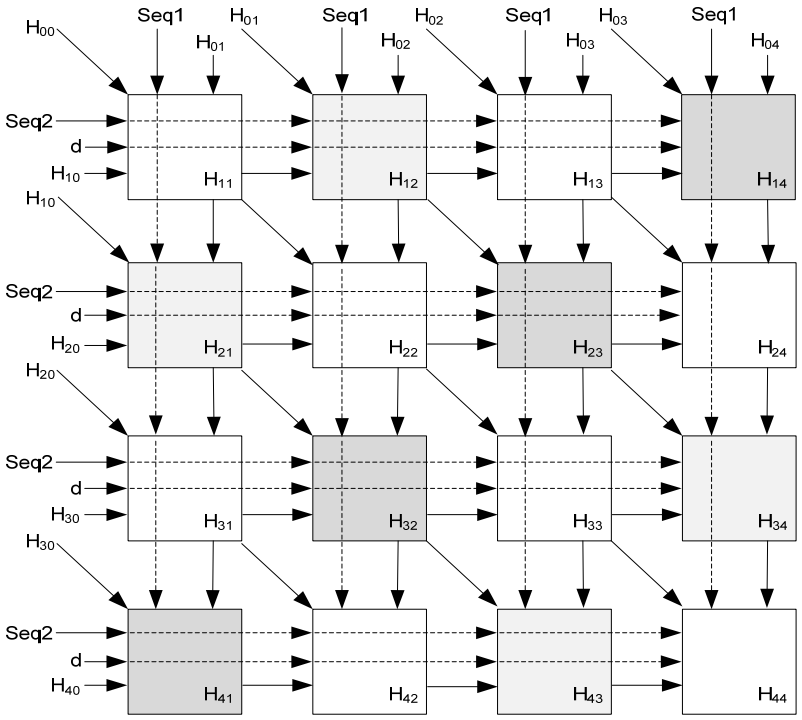

Figure 4: Block diagram description of a 4*x*4 systolic array

3.4 Max Finder Circuit for Systolic Array Architecture

Figure 5 shows the max finder circuit, which is used for keeping track of the maximum value in the systolic array. In this circuit, a comparator is used to compare two input values and sets the value of the output flag as a result. This flag is used as a select line of a multiplexer, which selects the maximum of the two inputs.

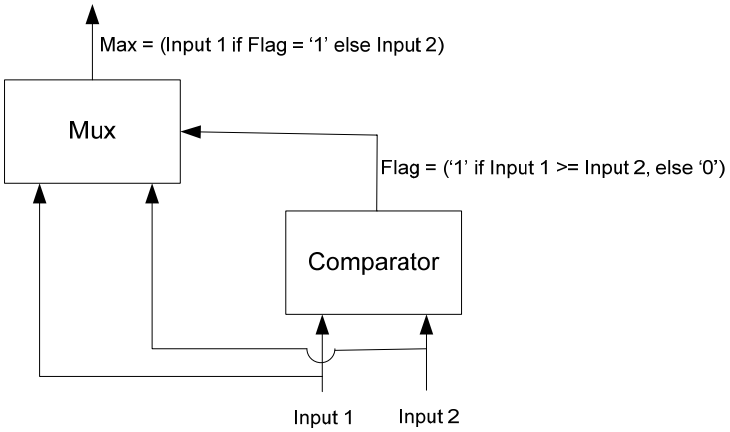

Figure 5: Max finder circuit, used to keep track of the maximum Hi,j value in the systolic array

Figure 6 shows the block diagram description of a 2*x*2 systolic array, where the max finder circuit of Figure 5 is used to keep track of the maximum $H_{i,j}$ value. The same structure is extendable to a 4*x*4 or larger systolic array.

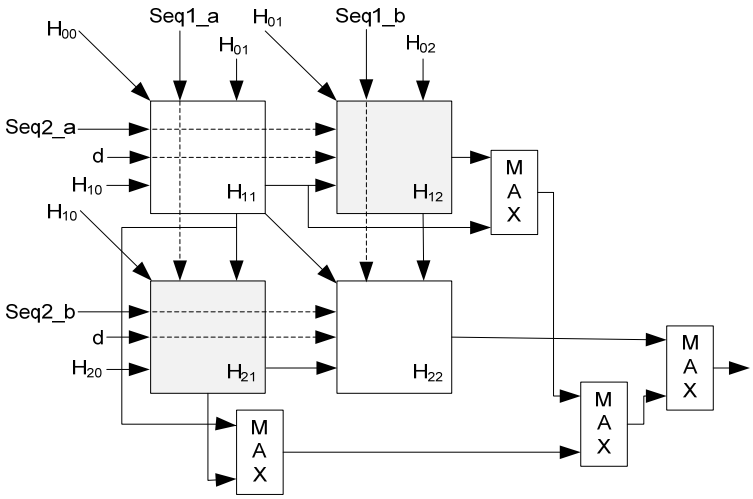

Figure 6: Block diagram description of a 2x2 systolic array with max finder circuit

4. DISCUSSION AND RESULTS

The results presented in Section 3 demonstrate that the implementation of the systolic array architecture with our cell design achieves a speedup of 652 *x*, as compared to a software-only implementation. This speedup is almost double the best case reported in the literature. Table 2 gives a comparison between our cell design and other systolic array designs, already reported in the literature.

Table 2: Comparison between our cell design and already reported similar designs

Cell Design	**Resource utilization**		**Maximum number of cells fitted**	**Frequency in MHz**	**Performance in terms of CUPS**
	Logic units per cell	Slices per cell			
Ours	7	19	1778	45	80 GCUPS
M Gok et al 2006	13	70	482	112.8	54.3 GCUPS
Oliver T. et al 2005	13	121	252	55	13.9 GCUPS

Our cell design is simpler and more efficient both in terms of resource utilization and performance gain than the design presented in (M Gok et al 2006) and (Oliver T. et al 2005). Figure 3 shows that our cell design utilizes only 7 logic units per cell i.e. 4 comparator and 3 adder units. In contrast, the design in (M Gok et al 2006) utilizes 13 logic units per cell i.e. 4 registers, 2 adders, 4 multiplexer units, 2 subtractors and a lookup table. The design in (Oliver T. et al 2005) also uses 13 logic units per cell i.e. 5 registers, 2 adders, 4 multiplexer units, a lookup table and a logic unit for sign extension. Also the synthesis results show that our design utilizes only 19 slices per cell, when implemented on a Xilinx Virtex-II XC2V6000 FPGA. In contrast, the design in (M Gok et al 2006) utilizes 70 slices per cell, whereas the design in
(Oliver T. et al 2005) utilizes 121 slices per cell, when implemented on the same Xilinx Virtex-II XC2V6000 FPGA. Thus our design utilizes 3.69 to 6.36 times less resources in comparison. Table 2 also shows that the performance achieved in terms of CUPS by our design is 1.47 to 5.75 times higher than the other two reported designs.

5. CONCLUSION

In this paper, we presented a systolic array architecture for the S-W algorithm with a high performance cell design. The results demonstrate that the implementation of our systolic array architecture achieves a speedup of upto 652*x*, in comparison with a software-only implementation, which is almost double the best case reported in the literature so far. The results also demonstrate that our design achieves 1.47 to 5.75 times higher performance in terms of CUPS as compared to similar cell designs already reported in the literature. Our cell design is also efficient in terms of resource utilization, as it uses 3.69 to 6.36 times less resources in comparison with the already reported similar designs.

ACKNOWLEDGEMENT

This work is financially supported by the Higher Education Commission of Pakistan.

REFERENCES

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

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

Borah M. et al, 1994. 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.

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

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

Hasan L. et al, September 2–5, 2007. 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.

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

Liao H. Y. et al, September 1–5, 2004. 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.

Mustafa Gok and Caglar Yilmaz, 2006. 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.

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

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

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

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

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

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

Yamaguchi Y. et al, 2002. 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.

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