Sindh Univ. Res. Jour. (Sci. Ser.) Vol.44 (2) 161-166 (2012)

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Performance Evaluation for RVE-based FPGA Acceleration of Genetic Sequence Alignment

L. HASAN , H. ZAFAR

Department of Computer Systems Engineering, University of Engineering and Technology, Peshawar, Pakistan.

Corresponding Author: [laiqhasan, haseeb]@nwfpuet.edu.pk Ph. No. +92 91 9216590

Received 27th November 2011 and Revised 02nd March 2012

Abstract: In this paper, Recursive Variable Expansion (RVE)-based Field-Programmable Gate Array (FPGA) acceleration of genetic sequence alignment and its comparison with traditional systolic array based acceleration is presented. The paper evaluates performance of the RVE-based FPGA acceleration for various blocking factors using performance metrics like throughput and throughput per area. The result is a better understanding of the RVE implications as the previous papers on the subject only considers latency as a performance metric.

Keywords: Bioinformatics, Recursive Variable Expansion, Smith-Waterman, FPGAs, Performance Evaluation.

1. INTRODUCTION

Based on *dynamic programming (DP)* (Giegerich, 2000), the S-W algorithm (Smith and Waterman, 1981) is a method that finds an optimal local sequence alignment (i.e., identifying common regions in sequences that share local similarity characteristics) between two DNA or protein sequences (the target sequence and the search sequence). 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) (Yamaguchi *et al*., 2002). 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 and Lipman, 1985) and BLAST (Altschul *et al*., 1990) 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 time of the S-W algorithm. Various approaches have been adopted to accelerate the SW algorithm in hardware (Borah *et al*., 1994) (Chiang *et al*., 2006) (Di-Blas *et al*., 2005) (Hasan and Al-Ars, 2007) (Schroder *et al*., 2006) (Yamaguchi *et al*., 2002). An overview of such approaches is given in (Hasan *et al*., 2007). In (Nawaz *et al*., 2007) an approach based on *Recursive Variable Expansion (RVE)* is presented, where RVE is a kind of loop transformation that removes all data dependencies from a program, so that the program is parallelized to its maximum. The RVE approach is discussed in detail in (Nawaz *et al*., 2008), where the authors conclude that the RVE approach is 1.6 times faster than the traditional acceleration approach; however the conclusion is based on a theoretical discussion and is not validated by implementation results. In (Hasan *et al*., 2008) 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 neighbors. In (Hasan *et al*., 2008), 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 under utilized most of the time. In (Hasan and Al-Ars, 2009), an efficient and high performance linear implementation of the S-W algorithm based on the RVE approach is presented and compared with a linear implementation based on

the systolic array approach. The linear implementations make sure that the hardware is always utilized at full and the efficiency is thus maximum. Also, the results demonstrate that the linear implementation based on the RVE approach is up to 2.33 times faster than the linear implementation based on the traditional systolic array approach, at the cost of utilizing 2 times more resources.

The RVE design discussed in (Hasan and Al-Ars, 2009) focused mainly on the latency as a performance metric, which is the time it takes a character of the database sequence to travel through the design. In this paper, we use other performance metrics to optimize the designs. RVE designs with various *blocking factors* (b_f) are analyzed using performance metrics like throughput and throughput/area besides latency. This results in a better understanding of the RVE implications.

The remainder of the paper is organized as follows: Section 2 presents an implementation based on the linear systolic array approach followed by an implementation based on the linear RVE approach. Section 3 evaluates the performance using metrics like throughput and performance/area. Section 4 gives a brief conclusion.

2. MATERIALS AND METHODS

Linear Systolic Array Implementation

Linear systolic array is a linear arrangement of processors (hereafter called cells), connected in series, where data flows synchronously across the array between neighbors, as shown in **(Fig 1)**. The cells are used repeatedly during each clock cycle.

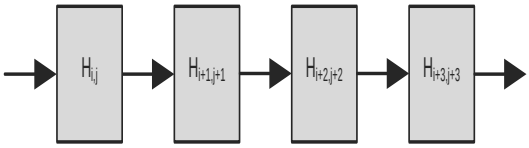

Fig. 1: Description of a 4-element linear systolic array

(Fig. 2) shows the block diagram representation of a basic cell design, for computing the elements of the *H* matrix for linear systolic array (Hasan and Al-Ars, 2009).

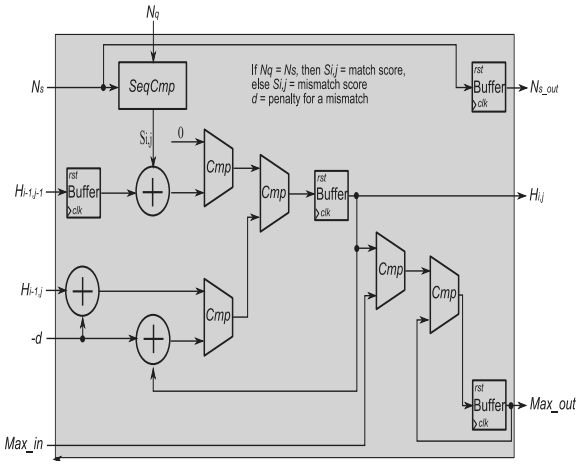

Fig. 2: Cell design for linear systolic array implementation

In the cell design of **(Fig. 2)**, *SeqCmp* compares the corresponding characters of the two input sequences and generates a similarity score. If the corresponding characters are the same, the similarity score is equal to a specific match score; otherwise it is equal to a mismatch score. The diagonal input from element ($H_{i-1,\,j-1}$) is buffered for one clock cycle, as it is used as a diagonal element after two steps. The similarity score is added with the delayed diagonal element using an adder, the output of which is compared with a *0* using a comparator. The comparator returns *0* if the output of the adder is negative, otherwise it returns the output of the adder. The left element ($H_{i-1,\,j}$) and the up element (which is the current value of the cell) are added with the gap penalty using adders, the outputs of which are compared using a comparator that returns the greater of the two values. This value is then compared with the value of the previous comparator (the one that compared the sum of the diagonal element and the similarity score with a *0*) and the greater of the two values is returned. The value of the cell, stored in a buffer, is also compared with the *Max* in value from the previous cell to find the global maximum. The current maximum value of the cell, stored in another buffer, is also compared with the global maximum. The maximum of the current and global maximums are compared and the greater of the two values is returned and hold in a buffer. Another buffer is used to delay the database sequence (N_s) by one clock cycle

for the next element of the array. The external clock and reset lines are connected with the *clk* and *rst* inputs of all the buffers. The cell design shown in **(Fig. 2)** can be used to build a linear systolic array based systems of any number of *processing elements (PEs)*, depending on the availability of hardware resources. **(Fig. 3)** shows an FPGA-based 4-PE linear systolic array implementation for S-W based sequence alignment using *Block RAM (BRAM)* for intermediate data storage before transmitting the resultant data to the *PC*. In addition, there are two BRAMs for the two input sequences, i.e. (BRAM for N_q) and (BRAM for N_s). These two BRAMs are initialized with the values of the two input sequences. The input sequences are applied to the PEs in such a way that the N_q values stay fixed in their corresponding PEs, whereas the N_s values are propagated through the array in synchronism with the clock.

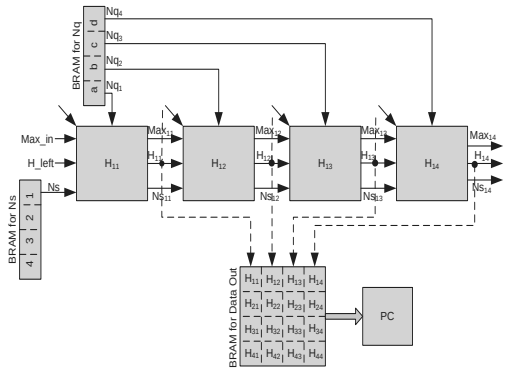

Fig. 3: Cell design for linear systolic array implementation

Linear RVE Implementation

This section presents the design of a basic building block for the linear RVE implementation. Further, it presents system designs based on this building block and presents a discussion of the results achieved. **(Fig. 4)** shows the block diagram representation of the linear RVE design that implements a 2×2 array. This RVE block depends on the search and target sequences (i.e. the query and database sequences), the *gap penalty (d)*, *Max* input, *CLK* and *RST*, in addition to the three external elements i.e. $H_{i-2,\ j-2}$, $H_{i,\ j-2}$ and $H_{i-1,\ j-2}$, and two feedback elements $H_{i-2,\ j}$ and $H_{i-2,\ j-1}$. Similarly, in addition to the four elements of the H matrix i.e. $H_{i,\ j}$, $H_{i,\ j-1}$, $H_{i-1,\ j}$ and $H_{i-1,\ j-1}$, the RVE block also outputs *Max* output, N_{s1} and N_{s2}, which become inputs for the next block, when the array is extended.

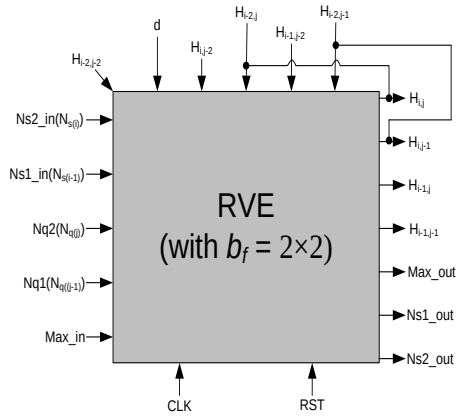

Fig. 4: Block diagram representation of the linear RVE design with b_f = 2x2

The basic building block for the linear RVE design shown in **(Fig. 4)** is used to develop RVE based systems of various sizes for sequence alignment applications. **(Fig. 5)** shows a 2-block linear RVE implementation as an example, where the blocks are connected in a linear systolic array fashion.

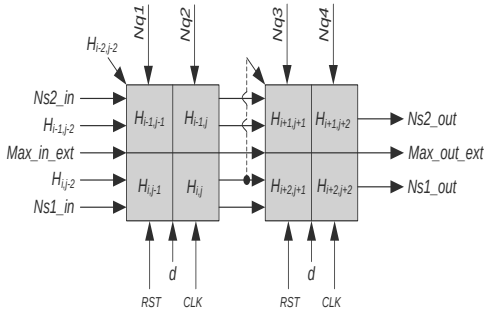

Fig. 5: 2-Block linear RVE design

The 2-block linear RVE design shown in **(Fig. 5)**, which is equivalent to the 4-element linear systolic array design, is implemented in VHDL and the post place and route simulation results show that the latency of the array is 300 *ns*, whereas the slices consumed are 254 out of 13696. The platform used for implementation is Xilinx Virtex-II Pro FPGA.

Table 1 presents the implementation results for various linear systolic array and linear RVE designs. It demonstrates that a 4-element linear systolic array implementation consumes 700 *ns* and utilizes 127 out of 13696 slices, when implemented on a Xilinx Virtex-II Pro FPGA. Thus a maximum of 107 PEs can be implemented on the same device, thereby consuming most of the available slices on the

FPGA. This implementation is used as a reference for comparison, which is traditionally used for accelerating the S-W based sequence alignment applications. The 2-block linear RVE implementation consumes 300 *ns* and utilizes 254 out of 13696 slices, when implemented on a Xilix Virtex-II Pro FPGA. Thus a maximum of 53 PEs can be implemented, using the same device. Thus in comparison with a traditional 4-element linear systolic array implementation, the 2-block linear RVE implementation improves the performance by a factor of 700/300 = 2.33, at the cost of utilizing 254/127 = 2 times additional hardware resources.

Table 1: Comparison between linear systolic and RVE implementations

Implementation	Time (*ns*)	Speed up	Slices	Hard ware cost
4-element Systolic array	700	1	127	1
2-block RVE	300	2.33	254	2
10-element Systolic array	1900	1	297	1
5-block RVE	900	2.11	601	2.02
200-element Systolic array	39900	1	6350	1
100-block RVE	19900	2.01	12700	2

The table also shows a comparison between 10-element linear systolic array and 5-block linear RVE implementations, where the 5-block linear RVE design performs 2.11 times better than the 10-element linear systolic array implementation at the cost of utilizing 2.02 times additional hardware resources. The table further demonstrates a comparison between 200-element linear systolic array and 100-block linear RVE implementations, where the 100-block linear RVE implementation achieves 39900/19900 = 2.01 times higher performance than the linear systolic array implementation at the cost of utilizing 12700/6350 =2 times additional hardware resources. A full scale linear systolic array implementation fits a maximum of 428 elements, whereas a full scale linear RVE implementation fits a maximum of 106 RVE blocks, where the device utilized for implementation is Xilinx Virtex-II Pro FPGA. Thus due to higher resource utilization by the linear RVE design, the full scale implementations are not comparable. From **Table 1**, it can be concluded that the linear RVE implementation is preferred in cases where high performance is desired and hardware cost is not a big concern.

The chart in **(Fig. 6)** shows a graphical comparison between various linear systolic array and linear RVE implementations, where the factors considered for comparison are the time consumed and the number of slices utilized. Clearly, the time consumption decreases by applying RVE, as shown in **(Fig. 6a)**. On the other hand, the resource utilization increases by applying RVE, as shown in **(Fig. 6b)**.

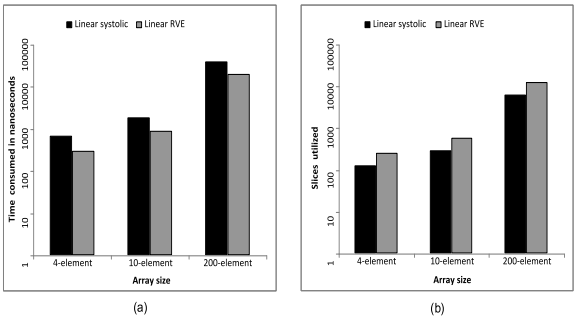

Fig. 6: Comparison between systolic array and RVE designs

3. RESULTS AND DISCUSSION

RVE Performance Evaluation

As discussed in (Vermij, 2011), the RVE designs discussed in the previous section focused mainly on the latency as a performance metric, which is the time it takes a character of the database sequence to travel through the design. The latency is given by Equation 1, where f_{opr} is the maximum operating frequency and each RVE block consumes one cycle to compute the results.

$$\text{Latency} = \text{Number of RVE blocks} \times \frac{1}{f_{opr}} \quad (1)$$

In this section, we use other performance metrics to analyze the designs. RVE designs with various blocking factors, as shown in **(Fig. 7)**, are analyzed using performance metrics like throughput and throughput/area besides latency. This results in a better understanding of the RVE implications (Vermij, 2011). The diagonal lines in **(Fig.7)** indicate the RVE blocks that can be computed in parallel.

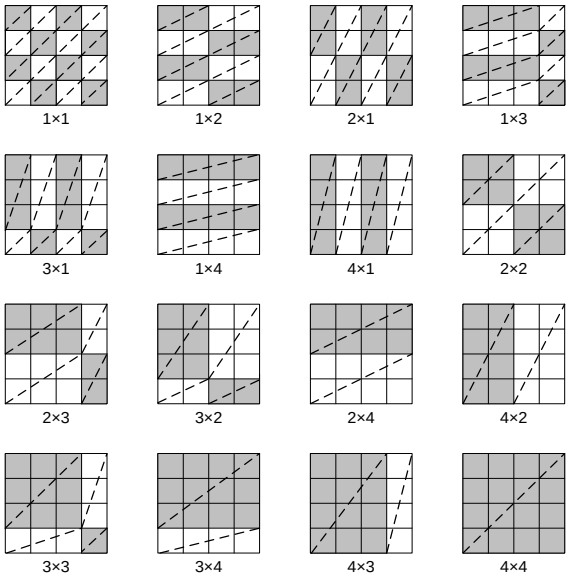

Fig. 7: RVE designs with various blocking factors

Table 2: Performance evaluation for various RVE implementations

Blocking factor (b_f)	Number of PEs	f_{opr} (*MHz*)	Latency (*ns*)	Throughput (MCUPS)	Throughput/area (MCUPS/slice)
1 × 1	36	159.0	226.4	5724	6.30
2 × 1	36	158.0	227.8	11376	6.81
3 × 1	36	121.0	297.5	13068	4.81
4 × 1	36	115.0	313.0	16560	4.19
2 × 2	18	118.0	152.5	8496	4.69
3 × 2	18	86.0	209.3	9288	2.61
4 × 2	18	67.0	268.6	9648	1.57
3 × 3	12	74.0	162.2	7992	1.76
4 × 3	12	66.8	179.6	9619	1.22
4 × 4	9	59.0	152.5	8496	0.81

Table 2 presents the performance evaluation results of linear RVE implementations with various blocking factors. The number of PEs is chosen such that a query sequence that is 36 characters long can be aligned in one run. It is clear from the 3rd column of the table that for square blocking factors (i.e. 1 × 1, 2 × 2, 3 × 3 and 4 × 4) increasing blocking factors result in lower frequencies. All non-square blocking factors (e.g., 2 × 1, 3 × 1, 4 × 1, 3 × 2, 4 × 2 and 4 × 3) run on lower frequencies as well. For example, the 1 × 1 and 2 × 1 blocks run at similar frequencies, but if the blocking factor is increased from 2 × 1 to 3 × 1, the frequency drops by 37 MHz. This can partly be explained by the way the formulas expand, i.e. the number of sequential additions and the logic depth of the comparators. For the 1 × 1, 2 × 1 and 3 × 1 blocking factors, the number of adder stages is respectively 2, 3 and 3, whereas, the number of comparator stages is respectively 2, 3 and 3, as reported by the synthesis tool. This shows that not only the theoretical logic depth, but also the actual implementation of the circuit plays a key role in determining the maximum frequency (Vermij, 2011).

The 4th column of **Table 1** shows that for the 2 × 2 square blocking factor, the latency of the design decreases significantly. But increasing the blocking factor further, the latency does not improve anymore, suggesting that going beyond 2 × 2 has no advantage. For the non-square blocking factors like 2 × 1 and 3 × 1, it is more complicated, as these blocks can be organized in two different ways giving rise to different latencies. This indicates that latency alone is not the best metric to evaluate the performance of an RVE design, as it only tells about how fast the end of the design is reached. To investigate the amount of work done by the design during every time unit, we compute throughput, which is defined as the number of cell updates per second. The throughput for RVE designs with various blocking factors is shown in the 5th column of **Table 2** and is calculated as per Equation 2, where each RVE block consumes one cycle to compute the results (Vermij, 2011).

$$\text{Throughput} = \text{Number of RVE blocks} \times b_f \times f_{opr} \quad (2)$$

Where b_f is the blocking factor. Like latency, the throughput also becomes better from 1 × 1 to 2 × 2, but the improvement becomes small for larger blocking factors. This is due to the fact that the frequency decreases rapidly for larger blocking factors, as shown in **Table 2**. Only the n × 1 blocks perform increasingly better. This is due to the fact that the frequency does not drop that fast here. The last column of **Table 2** gives performance/area in terms of MCUPS/slice. It is a useful performance metric that takes the hardware resource utilization or area cost into account and is calculated as per Equation 3 (Vermij, 2011).

$$\text{Performance per slice} = \frac{\text{Throughput}}{\text{Number of slices}} \quad (3)$$

It becomes clear that when taking the area into account, RVE does not perform better than the

default 1 × 1 design for most cases. Only the RVE design with the 2 × 1 blocking factor gives a better performance per slice than the default case, giving rise to the conclusion that RVE designs with non-square blocking factors should also be explored for higher performance (Vermij, 2011).

4. CONCLUSION

This paper presented linear systolic array implementation, its corresponding building block descriptions and subsequent system design for genetic sequence alignment applications. The paper further evaluated the performance for various RVE implementations considering different rectangular and square blocking factors. The paper further presented linear RVE implementations, its comparison with the equivalent linear systolic array implementations and a discussion of the results. The paper continued with performance evaluation of RVE designs with various blocking factors. The results demonstrate that that when taking the area into account, RVE does not perform better than the default 1 × 1 systolic array design for most cases. Only the RVE design with n × 1 blocking factor gives a better throughput per slice than the default case, giving rise to the conclusion that RVE designs with non-square blocking factors should be further explored for higher throughput.

REFERENCES:

Altschul S. (1990) A Basic Local Alignment Search Tool. Journal of Molecular Biology. (215): 403–410.

Borah M., R. Bajwa S. Hannenhalli and M. Irwin (1994) A SIMD Solution to the Sequence Comparison Problem on the MGAP. Proc. International Conference on Application Specific Array Processors, IEEE Computer Society Press: 336-345.

Chiang J., M. Studniberg J. Shaw S. Seto and K. Truong (2006) Hardware Accelerator for Genomic Sequence Alignment. Proc. 28th IEEE EMBS Annual International Conference. New York City, USA.

DiBlas A. (2005) The UCSC Kestrel Parallel Processor. IEEE Transactions on Parallel and Distributed Systems. (16): 1, 80–92.

Giegerich R. (2000) A Systematic Approach to Dynamic Programming in Bioinformatics. Bioinformatics. (16): 665–677.

Hasan L. and Z. Al-Ars (2007) Performance Improvement of the Smith-Waterman Algorithm. Proc. Annual Workshop on Circuits, Systems and Signal Processing. Veldhoven, The Netherlands.

Hasan L. and Z. Al-Ars (2009) An Efficient and High Performance Linear Recursive Variable Expansion Implementation of the Smith-Waterman Algorithm. Proc. 31st Annual International Conference of the IEEE EMBS. Minneapolis, Minnesota, USA. 3845-3848.

Hasan L., Z. Al-Ars and S. Vassiliadis (2007) Hardware Acceleration of Sequence Alignment Algorithms - An Overview. Proc. International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS'07). Rabat, Morocco.

Hasan L., Z. Al-Ars Z. Nawaz and K. Bertels (2008) Hardware Implementation of the Smith-Waterman Algorithm Using Recursive Variable Expansion. Proc. 3rd Inernational Design and Test Workshop IDT08. Monastir, Tunisia.

Hasan L., Y. Khawaja and A. Bais (2008) A Systolic Array Architecture for The Smith-Waterman Algorithm With High Performance Cell Design. Proc. IADIS European Conference on Data Mining. Amsterdam, The Netherlands.

Nawaz Z., O. Dragomir T. Marconi E. Panainte K. Bertels and S. Vassiliadis (2007) Recursive Variable Expansion: A Loop Transformation for Reconfigurable Systems. Proc. International Conference on Field-Programmable Technology. Kokurakita, Kitakyushu, Japan.

Nawaz Z., M. Shabbir Z. Al-Ars and K. Bertels (2008) Acceleration of Smith-Waterman Using Recursive Variable Expansion. Proc. 11th Euromicro Conference on Digital System Design 2008. Parma, Italy.

Pearson W. and D. Lipman (1985) Rapid and Sensitive Protein Similarity Searches. Science. (227): 1435–1441.

Schroder A. (2006) Bio-Sequence Database Scanning on a GPU. Proc. HICOMB, Rhodes Island, Greece.

Smith T. and M.Waterman (1981) Identification of Common Molecular Subsequences. Journal of Molecular Biology. (147): 195-197.

Vermij E. (2011) Genetic Sequence Alignment on a Supercomputing Platform. M.Sc. Thesis. CE-MS-2011-02.

Yamaguchi Y., Y. Miyajima T. Maruyama and A. Konagaya (2002) High Speed Homology Search Using Run-Time Reconfiguration. Proc. 12th International Conference on Field-Programmable Logic and Applications, Montpellier, France.