31st Annual International Conference of the IEEE EMBS
Minneapolis, Minnesota, USA, September 2-6, 2009

An Efficient and High Performance Linear Recursive Variable Expansion Implementation of the Smith-Waterman Algorithm

Laiq Hasan Zaid Al-Ars
Computer Engineering Lab, Delft University of Technology,
Mekelweg 4, 2628 CD, Delft, The Netherlands
Phone:+31-(0)15-27-86172 E-mail: L.HASAN@EWI.TUDELFT.NL

***Abstract*— In this paper, we present an efficient and high performance linear *recursive variable expansion (RVE)* implementation of the *Smith-Waterman (S-W)* algorithm and compare it with a traditional linear systolic array implementation. The results demonstrate that the linear RVE implementation performs up to 2.33 times better than the traditional linear systolic array implementation, at the cost of utilizing 2 times more resources.**

***Index Terms*— Bioinformatics, Sequence Alignment, Smith-Waterman Algorithm, FPGAs, Systolic Arrays, Recursive Variable Expansion**

I. Introduction

Based on *dynamic programming (DP)* [1], the S-W algorithm [2] 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). Each element of the matrix $H_{i,j}$ is calculated according to the following equation:

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

where $S_{i,j}$ is the similarity score of comparing sequence A_i to sequence B_j and d is the gap penalty. 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 time of the S-W algorithm. Various approaches have been adopted to accelerate the S-W algorithm in hardware [5], [6], [7], [8], [9], [10]. An overview of such approaches is given in [11]. In [12] 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 [13], 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 [14] 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 [15], 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 this paper, we present an efficient and high performance linear implementation of the S-W algorithm based on the RVE approach and compare it 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 remainder of the paper is organized as follows: Section II presents an implementation based on the linear systolic array approach. Section III presents an implementation based on the linear RVE approach. Section IV discusses and compares the results obtained from the two implementations. Section V gives a brief conclusion.

II. 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 Figure 1. The cells are used repeatedly during each clock cycle and the computed values are stored in registers for further manipulations.

Figure 2 shows the block diagram representation of a basic cell design, for computing the elements of the $H_{i,j}$

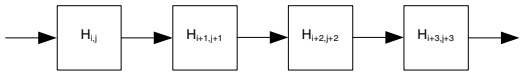

Fig. 1. An example of a four-element linear systolic array

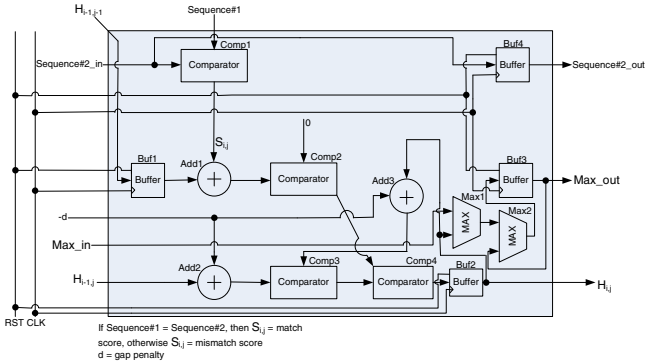

Fig. 2. Cell design for the linear systolic array implementation of the S-W algorithm

matrix of Equation 1, according to the linear systolic array implementation approach. In the cell design of Figure 2, Comp1 compares the corresponding characters of the two input sequences and generates a similarity score. If the corresponding characters are similar, 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 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. Add2 adds the left element $(H_{i-1,j})$ 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 input. Max2 compares the output of Max1 with the previous max value. The output of Max2 is stored back in Buf3. Buf4 delays the Sequence2 input by one clock cycle for the next element of the array. The cell design of Figure 2 is used as a building block for the implementation of the four-element linear systolic array, shown in Figure 3, where in addition to the four basic blocks, a De-Mux and a Mux blocks are also used. The De-Mux block is used to provide the corresponding characters of the search sequence (referred to as Seq1) to all cells. The De-Mux runs at a frequency, which is 4 times higher than the frequency of the cell itself. Thus the De-Mux is able to input all characters in one clock cycle, while minimizing the *Input Output Buffers (IOBs)* utilization by a factor of 4. In the same way, a Mux unit, running at a 4 times higher frequency than the operating frequency of the cell is used at the output as well, which reduces the IOBs utilization for the outputs by a factor of 4. For scaling up the design to a larger size, the total number of IOBs is calculated according to Equation 2, where the four-

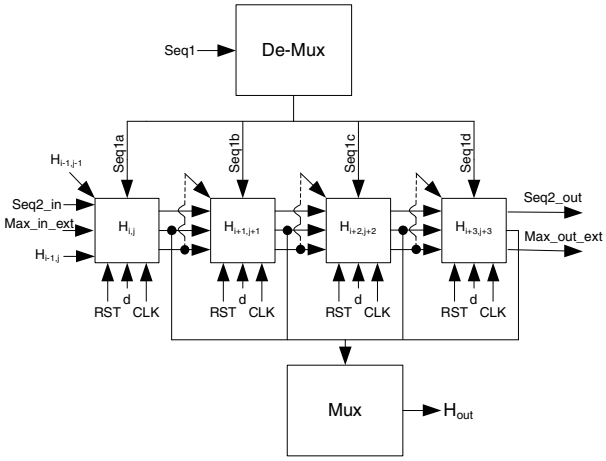

Fig. 3. A four-element linear systolic array implementation of the S-W algorithm

element linear systolic array is considered as one *Processing Element (PE)*.

$$\begin{aligned}\text{Total number of IOBs} \;=\; & \\ & \text{IOBs utilized by one PE} \;+ \\ & (\text{Number of PEs} \;-\; 1) \;* \\ & \text{IOBs utilized by } H_{out}\end{aligned} \tag{2}$$

The four elements linear systolic array, shown in Figure 3 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 700 ns, whereas the slices utilized are 127 out of 13696. The platform used for implementation is Xilinx Virtex II Pro.

III. Linear RVE Implementation

This section presents an implementation of the S-W algorithm based on the linear RVE design. Figure 4 shows the

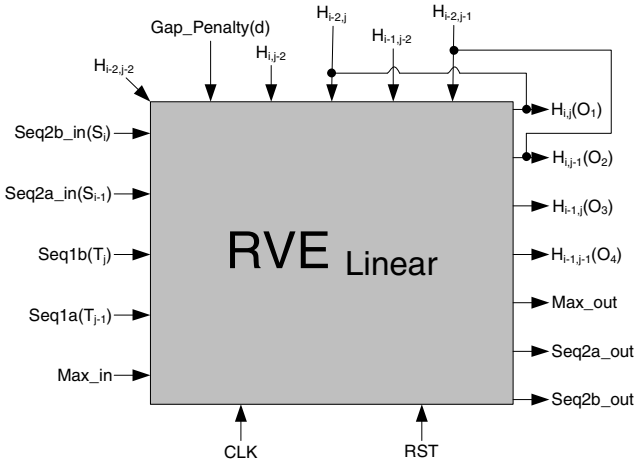

Fig. 4. Block diagram representation of the linear RVE design

block diagram representation of the linear RVE design that

implements a 2x2 array (hereafter called the RVE block). The advantage of this approach is that all four elements in the 2x2 array are computed in parallel, without waiting for the previous elements to be computed. Thus the data dependencies are minimized, as compared to the linear systolic array implementation. In [13], a detailed discussion and mathematical derivations for computing the four elements of $H_{i,j}$ matrix are given. The RVE block depends on the search and target sequences, the gap penalty, 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 $H_{i,j}$ 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, Seq2a and Seq2b, which become inputs for the next block, when the array is extended. Using this linear RVE design as a building block, we implemented a two-block linear RVE array as shown in Figure 5, where the blocks are connected in a linear systolic fashion. In addition to the two linear RVE blocks, a De-Mux block and two Mux blocks are also used in the extended design of Figure 5. The De-Mux block inputs the corresponding characters of Seq1 to both RVE blocks, in the same way as in the case of linear systolic array design. The Mux blocks are used to output the calculated values of the $H_{i,j}$ matrix, such that the IOBs utilization is minimized. Both the De-Mux and Mux blocks are running at a frequency, that is 4 times higher than the frequency of the RVE design.

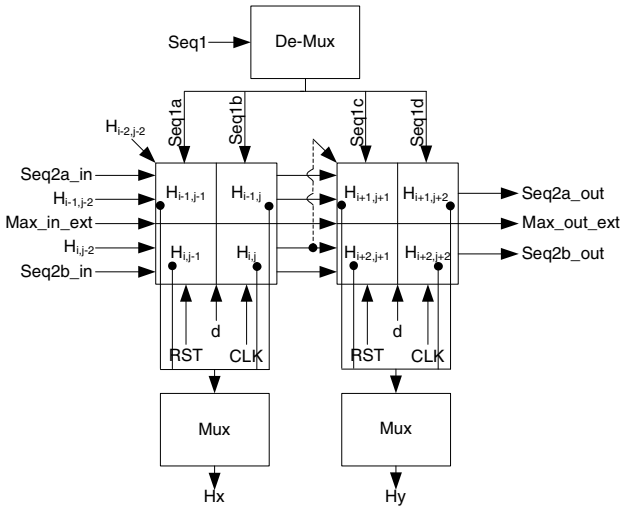

Fig. 5. Two-block linear RVE design

For scaling up the design to a larger size, the total number of IOBs are calculated according to Equation 3, where the two-block linear RVE array is considered as one PE.

$$\begin{aligned}\text{Total number of IOBs} \;=\; & \\ & \text{IOBs utilized by one PE} \;+ \\ & (\text{Number of PEs} \;-\; 1) \;* \\ & \text{IOBs utilized by} \;\; (H_x + H_y)\end{aligned} \tag{3}$$

The two-block linear RVE design shown in Figure 5, which is equivalent to the four-element linear systolic array design, 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 300 ns, whereas the slices consumed are 254 out of 13696. The platform used for implementation is Xilinx Virtex II Pro. Table I presents the filled matrix obtained, using both the linear systolic array and linear RVE implementations. The same inputs are used in both the cases and the same correct results verify the correctness of both designs. The bold digits in Table I, indicate the trace back path.

TABLE I

FILLED MATRIX OBTAINED USING THE LINEAR SYSTOLIC ARRAY AND LINEAR RVE IMPLEMENTATIONS

		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**

IV. EXPERIMENTAL RESULTS

Table II summarizes the results presented in Section II and Section III. It demonstrates that the four-element linear systolic array implementation consumes 700 ns with a 10 MHz clock frequency and utilizes 127 out of 13696 slices, when implemented on a Xilinx Virtex II Pro FPGA, whereas the IOBs utilization is 27 out of 556. Thus a maximum of 107 PEs can be implemented on the same device, thereby consuming most of the available slices on the 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 two-block linear RVE implementation consumes 300 ns with a 10 MHz clock frequency and utilizes 254 out of 13696 slices, when implemented on a Xilix Virtex II Pro FPGA, whereas the IOBs utilization is 55 out of 556. Thus a maximum of 53 PEs can be implemented, using the same device. Thus in comparison with a traditional four-element linear systolic array implementation, the two-block linear RVE implementation improves the performance by a factor of 700/300 = 2.33, at the cost of utilizing 254/127 = 2 times more resources. The table further demonstrates that when both designs are extended to fifty PEs each (where a linear systolic PE contains four elements and a linear RVE PE contains two linear RVE blocks), then the 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 more resources. The IOBs utilization for the fifty-PE implementations are calculated according to Equations 2 and 3, such that the IOBs utilized by the fifty-PE linear systolic array implementation = 27 + (50 - 1)x4 = 223 and the IOBs utilized by the fifty-PE linear RVE implementation = 55 + (50 - 1)x(4 + 4) = 447. A full scale linear systolic array implementation fits

TABLE II

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	Number of IOBs	Hardware utilization cost
Four-element linear systolic array	700 ns	10 MHz	1	127 out of 13696	27 out of 556	1
Two-block linear RVE	300 ns	10 MHz	2.33	254 out of 13696	55 out of 556	2
Fifty-PE linear systolic array	39900 ns	10 MHz	1	6350 out of 13696	223 out of 556	1
Fifty-PE linear RVE	19900 ns	10 MHz	2.01	12700 out of 13696	447 out of 556	2

a maximum of 107 PEs, whereas a full scale linear RVE implementation fits a maximum of 53 PEs, 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. The IOBs for the full scale implementations can be calculated according to Equations 2 and 3, such that the IOBs utilized by a full scale linear systolic array implementation and a full scale linear RVE implementation are 451 and 471, respectively. From Table II, it can be concluded that the linear RVE implementation is preferred in case high performance is desired and hardware cost is not a big concern. The chart in Figure 6 shows a graphical comparison between various linear systolic array and linear RVE implementations, where the factors considered for comparison are time consumed in nanoseconds and the number of slices utilized.

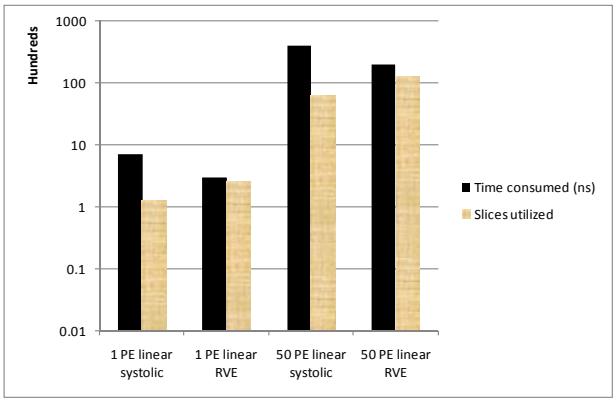

Fig. 6. Comparison between various linear systolic array and linear RVE implementations on a logarithmic scale

V. CONCLUSIONS

In this paper, we presented an efficient and high performance implementation of the S-W algorithm based on the linear RVE approach and compared it with a traditional linear systolic array implementation. The linear RVE implementation is efficient in terms of hardware utilization (both slices and IOBs) and high performance in terms of time consumption (latency). The results demonstrate that the linear RVE implementation is upto 2.33 times faster than a traditional linear systolic array implementation at the cost of utilizing 2 times more resources.

REFERENCES

[1] R. Giegerich, "A systematic approach to dynamic programming in bioinformatics", *Bioinformatics*, vol. 16, pp: 665–677, 2000.

[2] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences", *Journal of Molecular Biology*, vol. 147, pp: 195–197, 1981.

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

[4] S. F. Altschul, Gish, W. Miller, W. Myers and D. J. Lipman, "A Basic Local Alignment Search Tool", *Journal of Molecular Biology*, vol. 215, pp: 403–410, 1990.

[5] J. Chiang, M. Studniberg, J. Shaw, S. Seto and K. Truong, "Hardware Accelerator for Genomic Sequence Alignment", *Proceedings of the 28th IEEE EMBS Annual International Conference*, Aug 30–Sept 3, 2006, New York City, USA.

[6] Y. Yamaguchi, Y. Miyajima, T. Maruyama, and A. Konagaya, "High Speed Homology Search Using Run-Time Reconfiguration", *FPL 2002*.

[7] M. Borah, R. S. Bajwa, S. Hannenhalli and M. J. Irwin, "A SIMD Solution to the Sequence Comparison Problem on the MGAP", *Proceedings of the International Conference on Application Specific Array Processors*, 1994.

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

[9] A. Schroder et. al., "Bio-Sequence Database Scanning on a GPU" *HICOMB*, 2006.

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

[11] L. Hasan, Z. Al-Ars and S. Vassiliadis, "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, September 2–5, 2007, Rabat, Morocco.

[12] Z. Nawaz, O. S. Dragomir, T. Marconi, E. M. Panainte, K. Bertels and S. Vassiliadis, "Recursive Variable Expansion: A Loop Transformation for Reconfigurable Systems", *proceedings of International Conference on Field-Programmable Technology 2007*, Kokurakita, Kitakyushu, JAPAN, December 2007.

[13] Z. Nawaz, M. Shabbir, Z. Al-Ars, K.L.M. Bertels, "Acceleration of Smith-Waterman Using Recursive Variable Expansion", *proceedings of 11th Euromicro Conference on Digital System Design 2008*, Parma, Italy, September 2008.

[14] 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.

[15] 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 Inernational Design and Test Workshop IDT08*, Monastir, Tunisia, December 2008.