New Processor Array Architecture for DNA Sequence Alignment Using Stevens-Song Algorithm

ATEF IBRAHIM
Prince Sattam Bin Abdulaziz University
Department of Computer Engineering
Saudi Arabia
attif_ali2002@yahoo.com

HAMED ELSIMARY
Prince Sattam Bin Abdulaziz University &
Electronics Research Institute
Department of Computer Engineering &
Microelectronics Department
Saudi Arabia & Egypt
hamed@eri.sci.eg

ABDULLAH ALJUMAH
Prince Sattam Bin Abdulaziz University
Department of Computer Engineering
Saudi Arabia
aljumah88@hotmail.com

FAYEZ GEBALI
University of Victoria
Department of ECE
Canada
fayez@ece.uvic.ca

Abstract: This paper proposes a new processor array architecture for an optimized parallel sequence alignment algorithm. This architecture is extracted by applying a nonlinear mapping methodology to the Stevens-Song optimized algorithm after expressing it as Regular Iterative Algorithm (RIA). This methodology uses a data scheduling and node projection techniques to explore the processor array architectures of the algorithm. The proposed architecture is one of the explored architectures and has the advantage that it can be modified to be reused for multiple pass processing in order to increase the number of processing elements that can be packed into a single FPGA and to increase the number of sequences that can be aligned in parallel in a single FPGA. This resolves the potential problem of many FPGA resources left unused for designs that have large values of short read length when using the previously published conventional hardware design. FPGA implementation results show that, for large values of short read lengths ($M > 128$), the proposed design has a slightly higher speed up and FPGA utilization over the the conventional one.

Key–Words: Systolic array, Bioinformatics, Genome sequence alignment, Resequencing applications, Sequencing Technology, Biological computation.

1 Introduction

The coming of the most recent generations of sequencing technologies [1] has opened a lot of new research chances in the fields of science (biology) and medication, including cell Deoxyribonucleic Acid (DNA) sequencing, gene disclosure and evolutionary connections. These sophisticated technologies have helped the exponential development of biological information that is accessible for specialists. For example, the Genbank [2] has multiplied its information measure at regular intervals (approximately 18 months) and in its latest release of February 2014 it included over 158×10^9 base-pair (bp) from a few distinctive species.

To aid the researcher in the extraction of handy data and in the understanding of the immense estimated sequence databases, a set of alignment algorithms (e.g. the generally utilized Smith-Waterman (S-W) [3] and Steven-Song optimized [4, 5] algorithms) have been produced to take care of numerous open issues in the field of bioinformatics, for example, (1) DNA re-sequencing, where genome gathering is carried out against a reference genome; (2) Multiple Sequence Alignment (MSA), where various genomes are adjusted to perform genome annotation; and (3) Gene discovering, where Ribonucleic Acid (RNA) sequences are adjusted against the living being genome to recognize new genes.

At present, a basic sequencing methodology is dependent upon the provision of High Throughput Short Read (HTSR) technologies [6], to decrease the expense of the sequencing procedure. This methodology comprises of cutting the DNA pieces under analysis into shorter sections called short reads, which are exclusively sequenced and aligned against a reference sequence.

Currently, the three most essential HTSR sequencing platforms are: the Solexa 1g Sequencer (llumina), the GS FLX Genome Analyzer (454), and the Solid Sequencer (Applied Biosystems). The biochemistry engineering underlying each of these platforms prompts altogether different attributes, in terms of throughput, short reads length, and raw errors. In any case, freely of the embraced platform, the length of the short reads processed by these platforms is small when contrasted with past generation sequencing technologies and much smaller than the original completed DNA sequence. Though, the big volume of information that is created and the requirement to align these short reads to huge reference genomes constrains an immediate and credulous provision of standard Dynamic Programming (DP) methods. One straightforward example of a common challenge comes from the necessity to align up to 100 million short reads against a reference genome that could be as substantial as 3 Gbp. For the SOLiD sequencer, with reads as short as 30 bps, this relates to the processing of 100 million grids of size $3 \times 10^9 \times 30$, which brings about a computational assignment that is unfeasible actually for a standard high execution machine. Hence, the computational requests for the analysis of the genetic information handled by the different sequencing technologies has lead to the advancement of a few accelerating methodologies that aims at parallelizing the execution of the alignment algorithms. Some of these methodologies are programming based, while others utilize devoted hardware implementations. Among the programming based methodologies, an optimized software implementation utilizing Single-Instruction Multiple-Data (SIMD) instructions for current general purpose processors [7] is normally adopted in sequence alignment programs, in the same way as SSEARCH35. Other software implementations make utilization of the exceedingly parallel execution competencies exhibited by Graphics Processing Unit (GPU) to attain a high alignment throughput [8]. Concerning the hardware implementations, these incorporate both Application Specific Integrated Circuit (ASIC) [9–12] and Field Programmable Gate Array (FPGA) [13–17] implementations. Despite the recognized implementation, the most well-known and productive hardware architectures map alignment computations to a systolic array of Processing Element (PE). Moreover, in spite of the fact that some bi-dimensional processor arrays have been exhibited [18], the most widely recognized implementations adopt a uni-dimensional (linear) systolic array [9–17], [19–23]. Actually, the principle contrasts among the different implementations attributed to the design of the singular PE.

The aforementioned proposed systolic array architecture is not effectively enhanced to manage the short reads sequences got from current HTSR sequencing platforms (e.g. Illumina). In this paper, the authors propose a novel semi-systolic array architecture for Stevens-Song optimized algorithm [4, 5] for DNA resequencing that is slightly efficient in speed and area - for large values of short read length - than the conventional systolic array architectures reported in the literature [9–17], [19–23] and also it can be easily optimized to deal with short read sequences, got from current HTSR sequencing platforms. This is achieved by applying a nonlinear mapping methodology to the Stevens-Song optimized algorithm after expressing it as Regular Iterative Algorithm (RIA). This methodology uses a data scheduling and node projection techniques to explore the processor array architectures of the algorithm. One of the explored architectures is new and to the best of our knowledge is not reported before in the literature, while the other one is identical to the conventional systolic array that is reported in the literature [9–17], [19–23]. Also, this paper presents the hardware implementation of the processing elements (PEs) of the explored processor array architectures. The new processor array architecture (called semi-systolic array architecture) can be modified to be reused for multiple pass processing in order to increase the number of processing elements that can be packed into a single FPGA and to increase the number of sequences that can be aligned in parallel in a single FPGA.

This paper is organized as follows. Section 2 presents the optimized parallel genome alignment algorithm of Stevens-Song [4, 5]. Section 3 presents the proposed methodology employed to explore the new semi-systolic array architecture and other architectures. Section 4 shows the Multiple short-read alignment in a single FPGA using the proposed architecture. Section 5 compares the resulting semi-systolic array to the previously published conventional systolic array in terms of area, speed. Finally Section 6 concludes the paper.

2 An optimized parallel sequence alignment algorithm

In this research work, we are concerned with the acceleration of DNA resequencing. Resequencing is the task of sequencing DNA of a an individual when provided with a reference sequence to the species. In resequencing, the prepared data is composed in small fragments, called short reads, with currently normal length lies in the range of $30 \sim 200$ base-pairs and it is likely doubles in the next few years. The sequence alignment of many short-reads of length

M ($30 \leq M \leq 200$) against a long reference sequence of length N ($10^8 \leq N \leq 3 \times 10^9$) is the predominant consumer of computational resources. Stevens and Song [4, 5] have prescribed a sequence alignment algorithm that is sufficiently adaptable for resequencing applications, yet is additionally efficiently implementable on FPGAs.

The notations used in this paper are as follows:

- $R = r1r2 \ldots rj \ldots rN$: reference sequence of length N, $rj \in \{$A, G, C, T $\}$.
- $S = s1s2 \ldots si \ldots sM$: short-read sequence of length M, $si \in \{$A, G, C, T$\}$.
- C : similarity matrix.
- $c(i, j)$: the optimal similarity score for the alignment of the first i characters of S against the first j characters of R.
- α : penalty for deletion in S.
- β: penalty for insertion in S.
- $\gamma(si, rj)$: base-pair mismatch penalty.

The core part of the algorithm is the calculation of an $(N + 1)$ by $(M + 1)$ similarity matrix C, often called the dynamic programming table. The following recursion computes the matrix elements (optimal scores).

$$c(i,j) = \min \begin{cases} C(i, j-1) + \alpha \\ C(i-1, j) + \beta \\ C(i-1, j-1) + \gamma(si, rj) \end{cases}$$

The authors of algorithm supposed that the penalties of the α, β, and γ are restricted to positive integers in the range $[0, 3]$ and there is no charge penalty at the beginning of the read. The restriction to positive values implies that the matrix definition should select a minimum rather than maximum value as in global Needleman-Wunsch [24] and local Smith-Waterman (S-W) [3] algorithms. Another difference between conventional sequence alignment algorithms and the one presented in the paper of [5] is that the output values are taken from the final row in the matrix. In fact for this application, it is sufficient to output only the column positions where the value in the final row is below some threshold T. In our implementation γ is chosen to be as follows:

$$\gamma(si, rj) = \begin{cases} 0, & \text{if } si = rj \\ \delta, & \text{otherwise} \end{cases}$$

Therefore no penalty is applied if the base-pairs are the same, and a standard penalty of δ is applied if they are not. where δ should be in the range 1 to 3. Therefore, it can be represented using only two bits.

3 A Systematic Methodology for Processor Array Design

Systematic methodologies to design processor arrays allow for design space exploration for optimizing performance according to certain specifications while satisfying design constrains. Several methodologies were proposed earlier [25], [26], [27], [28]. However, most of these methodologies were not able to deal with algorithms that have dimensions more than two. The third author proposed a systematic methodology that deals with algorithms of arbitrary dimensions [28, 29]. The author proposed a formal algebraic procedure for processor array design starting from a Regular Iterative Algorithm (RIA) for a three-dimensional digital filter which gives rise to a dependency graph in six-dimensional space. In this work, we used this formal technique to develop processor arrays for Stevens and Song optimized algorithm.

3.1 Obtaining the Algorithm Dependency Graph (DG)

The Stevens and Song Algorithm explained in Section 2 can be easily defined on a two dimensional (2D) domain since there are two indices (i, j). The DG is shown in Fig. 1. The computation domain is the convex hull in the 2D space, where the algorithm operations are defined as indicated by circles in the 2D plane [28–30]. Also, from this figure we notice that the input variables si and $c(i, j-1)$ are represented by horizontal lines, the input variables rj and $c(i-1, j)$ are represented by vertical lines, and the output variable $c(i, j)$ is represented by the slanted lines. the zero inputs at the left and upper borders of DG represents the initial values of the score matrix, $c(i, 0)$ and $c(0, j)$.

3.2 Data Scheduling

Pipelining or broadcasting the variables of an algorithm is determined by the choice of a timing function that assigns a time value to each node in the DG. A simple but useful timing function is an affine scheduling function of the form [28].

$$t(p) = Gp - g \tag{1}$$

where the function $t(p)$ associates a time value t to a point p in the DG. Value of G is chosen to ensure that

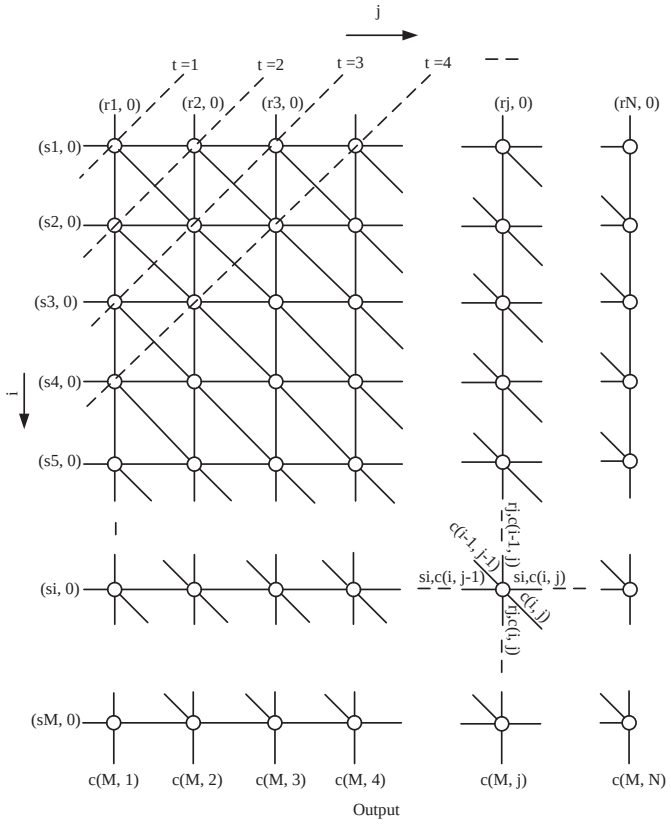

Figure 1: Stevens-Song algorithm dependency graph.

only positive time index values are obtained. The row vector $G = [g_1\ g_2]$ is the scheduling vector and g is an integer. The affine scheduling function must satisfy several conditions. From Fig.1, we observe that in each column the output variable $c(i, j)$ of each node depends on the output variable $c(i-1, j)$ form the previous node in the same column, thus we can write

$$t(p(i-1, j)) < t(p(i, j)) \tag{2}$$

Applying our scheduling function in Equation (2) to this inequality, we get

$$\begin{bmatrix} g_1 & g_2 \end{bmatrix} \begin{bmatrix} i-1 \\ j \end{bmatrix} < \begin{bmatrix} g_1 & g_2 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \tag{3}$$

$$ig_1 - g_1 + jg_2 < ig_1 + jg_2 \tag{4}$$

Which could be simplified to

$$g_1 > 0 \tag{5}$$

Similarly From Fig.1, In each row we observe that the output variable $c(i, j)$ depends on the previous output value $c(i, j-1)$ of the same row, thus we can write

$$\begin{bmatrix} g_1 & g_2 \end{bmatrix} \begin{bmatrix} i \\ j-1 \end{bmatrix} < \begin{bmatrix} g_1 & g_2 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} \tag{6}$$

$$ig_1 + jg_2 - g_2 < ig_1 + jg_2 \tag{7}$$

Which could be simplified to

$$g_2 > 0 \tag{8}$$

From Equations 5 and 8 there are many solutions for G, the most reasonable and simplest one is

$$G = [1\ 1] \tag{9}$$

If we want to pipeline or allocate a variable whose nullvector is θ , we must have

$$G\theta^t \neq 0 \tag{10}$$

where θ is the nullvector of the variable dependence matrix [28]. On the other hand, if we want to broadcast a variable whose nullvector is θ , we must have [28]

$$G\theta^t = 0 \tag{11}$$

To study the timing of the variables si, rj, and $c(i, j)$, we first find their nullvectors

$$\theta_{si} = [0\ 1] \tag{12}$$

$$\theta_{rj} = [1\ 0] \tag{13}$$

$$\theta_{c(i,j)} = [-1\ -1] \tag{14}$$

The product of G and these nullvectors gives

$$G\theta^t_{si} = 1 \tag{15}$$

$$G\theta^t_{rj} = 1 \tag{16}$$

$$G\theta^t_{c(i,j)} = -2 \tag{17}$$

Therefore, The input and output variables will be pipelined or allocated.

3.3 DG Node Projection

The projection operation is a many-to-one function that maps several nodes of the DG onto a single node, which constitutes the resulting processor array. Thus, several operations in the DG are mapped to a single PE. The projection operation allows hardware economy by multiplexing several operations in the DG on a single PE. The third author [28] explained how to perform the projection operation using a projection matrix P. To obtain the projection matrix we need to define a desired projection direction d. The vector

d belongs to the null-space of P. Since we are dealing with a two-dimensional DG, matrix P is a row vector and d is a column vector [28]. A valid projection direction d must satisfy the inequality [28]

$$Gd \neq 0 \tag{18}$$

In the following, we will discuss design space explorations based on the timing function G obtained in Equation 9. There are many projection vectors that satisfy Equation 18 for the scheduling function in Equation 9. For simplicity we choose four of them as follows:

$$d_1 = [1\ 0]^t \tag{19}$$

$$d_2 = [0\ 1]^t \tag{20}$$

$$d_3 = [1\ 1]^t \tag{21}$$

The corresponding projection matrices are given by

$$P_1 = [0\ 1] \tag{22}$$

$$P_2 = [1\ 0] \tag{23}$$

$$P_3 = [-1\ 1] \tag{24}$$

Our processor design space now allows for three systolic/semi-systolic array configurations for each projection vector for the timing function G. In the following subsections, we study the systolic/semi-systolic arrays associated with each design option.

3.3.1 Design1: using $d_1 = [1\ 0]^t$

A point in the DG $p = [i\ j]^t$ will be mapped by the projection matrix $P_1 = [0\ 1]$ onto the point

$$p' = P_1 p = j \tag{25}$$

The resulting semi-systolic array corresponding to the projection matrix P_1 consists of N PEs. Only at most M PEs are active at each time step. Fig. 2 shows the processor activity for the case $N = 6$ and $M = 3$, where the black nodes represent active PEs and white nodes represent idle PEs. Since only maximum M PEs are active at a given time step, the PEs are not well utilized. To improve PE utilization, we need to reduce the number of processors. We note from Fig. 2 that PE_j and PE_{j+M} are active at non-overlapping time steps. Thus, each pair of PEs (PE_j and PE_{j+M}) can be mapped to a single PE without causing time conflicts. This can be achieved by mapping PEs with indices j (in Fig. 2) to PEs with indices $j^{'}$ using the following nonlinear projection operation:

$$j^{'} = j \bmod M \tag{26}$$

The resulting semi-systolic array for different values of N and M is shown in Fig. 3. To the best of our knowledge, This semi-systolic array is new and was not reported before in the literature. The semi-systolic array consists of M PEs. Input bits of reference sequence $r(kM + j)$ should be allocated to each processing element in the semi-systolic array, where k has values in range from 0 to $\lceil N/M \rceil - 1$. since each PE in the semi-systolic array needs the reference sequence bits at different time (one time step difference between consecutive PEs as shown in activity graph, Fig. 2). Thus, these reference bits are made available for all PEs (broadcasted) with internal control in each PE that loads the signal at the right time. On the other hand, Input bits of the short-read sequence si and intermediate out bits of $c(i, j)$ are pipelined between adjacent PEs. A tristate buffer at the output of each PE ensures that it is the only output fed to the output bus.

To reduce the size of the PE of the semi-systolic array to maximize parallelism, we used the compact offset encoding scheme of [5] to reduce the bandwidths of its data paths. In this scheme, the authors labeled the three inputs and the one output (a quartet) of the scoring matrix recursion starting from left to right as x, y, z, w, where x represents the upper left input and w is the output. They proved that in any quartet of the scoring matrix there are only two possible offsets of the value of x, $x - 1$ and $x + 1$, for the uniform scoring function: $\alpha = 1$, $\beta = 1$, $\delta = 1$ (see Table 1). As shown in Table 1, the recursion contains only three possible relative values $(x, x - 1, x + 1)$. Therefore, only two bits are required to represent the scores.

Table 1: Offset encoding scheme using the uniform scoring function ($\alpha = 1$, $\beta = 1$, $\delta = 1$) [5].

x	$y = x - 1, x, x + 1$
$z = x - 1, x, x + 1$	$w = x, x + 1$

Figure 4 shows the hardware implementation of each PE of Design1 depending on the compact offset encoding scheme. The input $r(kM + j)$ represents a base pair of the reference sequence R. It is loaded in a register one clock cycle before a computation starts and it is applied for the whole computational cycle (M clock cycles). The input si represents the base pair of the short read sequence S that is pipelined to the next PE through register. The resulted score $c(M, kM + j)$ will be available on the output bus after M clock cycles through a tristate buffer. There are two data path control signals. The first one is the "first-clk-cycle", that indicates the first cycle in each iteration. The sec-

ond one is the "Last-clk-cycle", that indicates the last cycle in each iteration. These control signals, in each PE, is delayed one clock cycle before propagating to the next PE. This is done by propagating these signals through registers between the PEs.

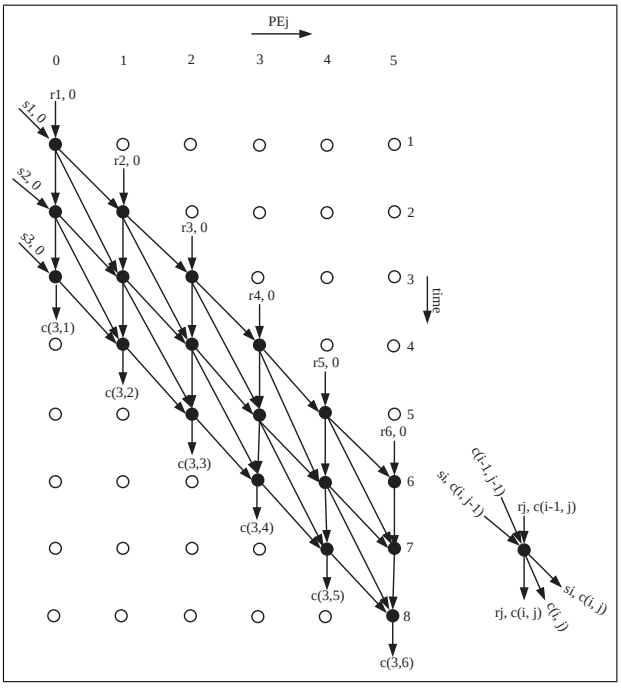

Figure 2: Processor activity at different time steps for $d_1 = [1\ 0]^t$, $N = 6$, and $M = 3$.

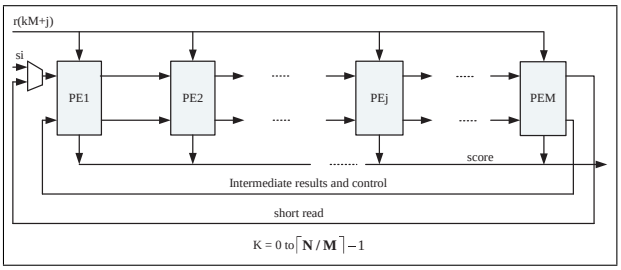

Figure 3: Processor array for $d_1 = [1\ 0]^t$ for different values of N and M.

3.3.2 Design2: using $d_2 = [0\ 1]^t$

A point in the DG $p = [i\ j]^t$ will be mapped by the projection matrix $P_2 = [1\ 0]$ onto the point

$$p' = P_2 p = i \tag{27}$$

The resulting systolic array corresponding to the projection matrix P_2 consists of M PEs. Fig. 5 shows the processor activity for the case $N = 6$ and $M = 3$,

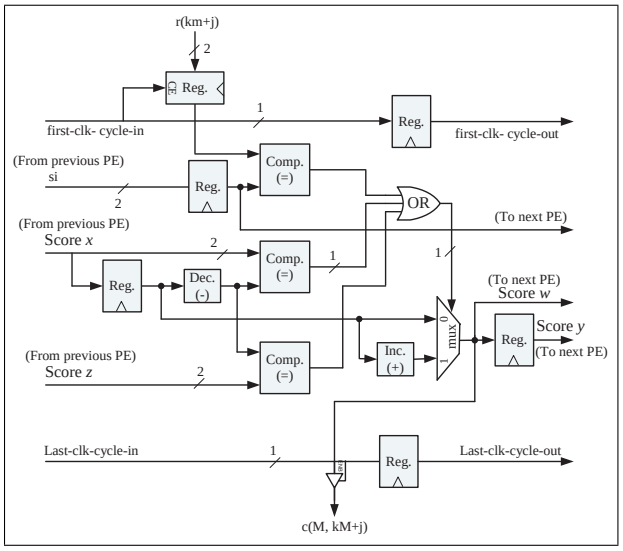

Figure 4: Design1 PE logic diagram for uniform scoring function ($\alpha = 1$, $\beta = 1$, $\delta = 1$).

where the black nodes represent active PEs and white nodes represent idle PEs. At most time steps, the maximum number of PEs, M, are active and this results in a good utilization of PEs. The resulting systolic array, for different values of N and M is shown in Fig. 6. This systolic array is identical to the one reported in [9–17], [19–23]. The systolic array now consists of M PEs. Input bits of short-read si should be allocated to each PE (as reported in the previous publications). Since each PE in the processor array needs the short read bits at different time (one time step difference between consecutive PEs as shown in activity graph, Fig. 5). Thus, these bits are made available for all PEs (broadcasted) with internal control in each PE that loads the signal at the right time. By this way, the query sequence bits can be fed serially to the systolic array as the subject sequence bits. On the other hand, input bits of the reference sequence rj and the intermediate output bits of $c(i, j)$ of each PE are pipelined to the next PE with higher index. The output score is obtained serially from the last PE (PEM) of the array after latency of M clock cycle.

To reduce the size of the PE of the systolic array to maximize parallelism, we used the compact offset encoding scheme of [5] to reduce the bandwidths of its data paths. This scheme was discussed before in Subsection 3.3.1.

Figure 7 shows the hardware implementation details of each PE of Design2 depending on the compact offset encoding scheme. The input si represents a base pair of the short read sequence S. It is loaded in a register one clock cycle before a computation starts and it is applied for the whole computational cycle (M

clock cycles). The input rj represents the base pair of the reference sequence R that is pipelined to the next PE through register. The resulted score $c(M, j)$ will be available on the output bus, after M clock cycles, from the last PE in the systolic array. .

There is one data path control signal, "first-clk-cycle", that indicates the first cycle in each iteration. This signal is delayed one clock cycle before propagating to the next PE. This is done by propagating this signal through registers between the PEs.

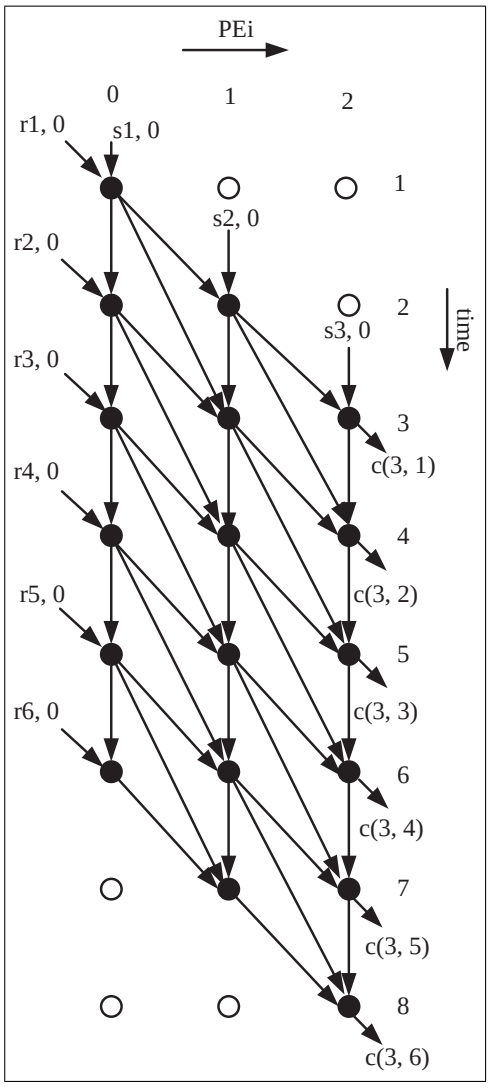

Figure 5: Processor activity at different time steps for $d_2 = [0\ 1]^t$, $N = 6$, and $M = 3$.

3.3.3 Design3: using $d_3 = [1\ 1]^t$

A point in the DG $p = [i\ j]^t$ will be mapped by the projection matrix $P_3 = [-1\ 1]$ onto the point

$$p' = P_3 p = i - j \tag{28}$$

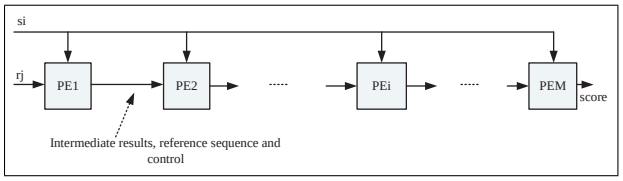

Figure 6: Processor array for $d_2 = [0\ 1]^t$ for different values of N and M.

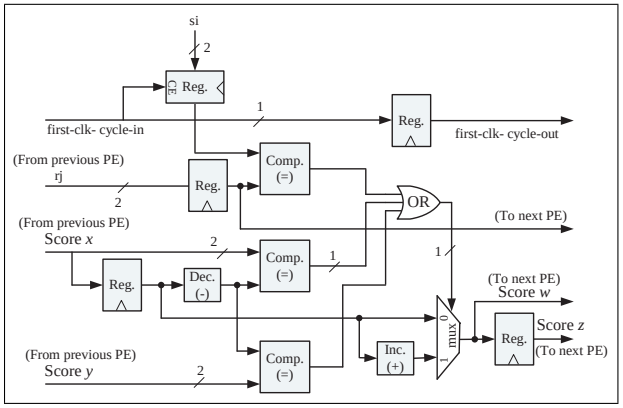

Figure 7: Design2 PE logic diagram for uniform scoring function ($\alpha = 1$, $\beta = 1$, $\delta = 1$).

The resulting semi-systolic array corresponding to the projection matrix P_3 consists of M PEs, after adding a fixed increment to all PE indices to ensure non-negative PE index values [28, 29]. Only at most M PEs are active at each time step. Fig. 8 shows the processor activity for the case $N = 6$ and $M = 3$ where the black nodes represent active PEs and white nodes represent idle PEs. Since only maximum M PEs are active at a given time step, the PEs are not well utilized. To improve PE utilization, we need to reduce the number of processors. We notice from Fig. 8 that all PEs whose indices are given by (28) can be mapped to PEs with indices j' as

$$j' = i - j \bmod M \tag{29}$$

without any timing conflicts. This statement is true as long as the inequality $Gd_3 \neq M$ is satisfied. The resulted semi-systolic array of this design is not regular and will have high complexity that makes it not suitable for VLSI implementation. Thus, we will ignore this design in this research paper.

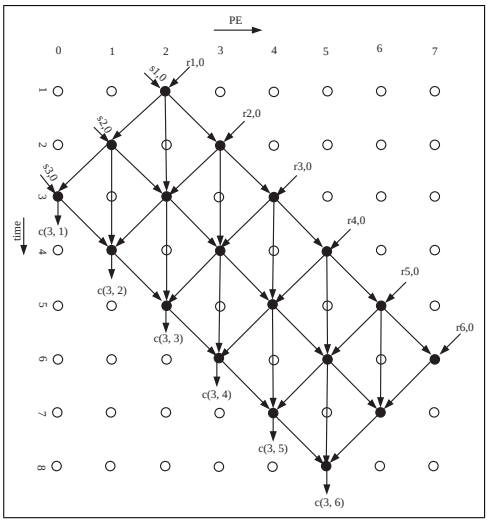

Figure 8: Processor activity at different time steps for $d_3 = [1\ 1]^t$, $N = 6$, and $M = 3$.

4 Multiple short-read alignment in a single FPGA using the proposed design

The proposed semi-systolic array, shown in Fig. 3, computes a single short read against a reference sequence. However, it is possible to fit several independent semi-systolic arrays onto an FPGA, and by doing so it is possible to align several short-reads in parallel. The extension of the design from a single alignment processor to one that can perform P alignments in parallel is shown in Fig. 9. The semi-systolic array is simply replicated as many times as we can fit it onto a single FPGA. The DRAM interface and serializer feeds all the systolic arrays in parallel. The score output buses in the semi-systolic arrays are passed to a thresholding output stages that conditionally write the corresponding reference base-pair index to a FIFO buffer if the score is below a user-specified value. The round-robin reader cycles through the FIFOs at the end of each systolic array to read their values and write those values to a single shared FIFO. The hardware details of the thresholding output stage and the round-robin reader are given in [4].

Due to the on-chip memory and logic limitations of FPGA, the number of PEs that could be packed into a single FPGA is limited. Also, there is another potential problem to designs that have large values of short read length M ($M \geq 128$). This problem is the large amount of FPGA resources that left unused due to the number of semi-systolic arrays P should be a whole number. It is possible to alleviate this problem by making semi-systolic arrays with different length, but this solution is not desirable to the user, who has constant-length short reads. To increase the number of PEs that can be packed into a single FPGA and hence increasing the number of sequences that can be aligned in parallel in it, this design architecture should be modified to be reused for multiple pass processing. This is also resolves the potential problem of many FPGA resources left unused for designs that have large values of short read length M. The design modification starts by partitioning the algorithm in hand into small alignment steps and map the partitioned algorithm on a fixed size linear semi-systolic array of size v, where $M > v$. This fixed size array is called folded semi-systolic array and can be extended to perform $Q = pq$ alignments in parallel, where $q = \lceil M/v \rceil$ is the number of folds. In each fold of semi-systolic array, $M - v$ short read base-pairs should be kept in a feed back FIFO to be used for the next fold. We still working on the design of

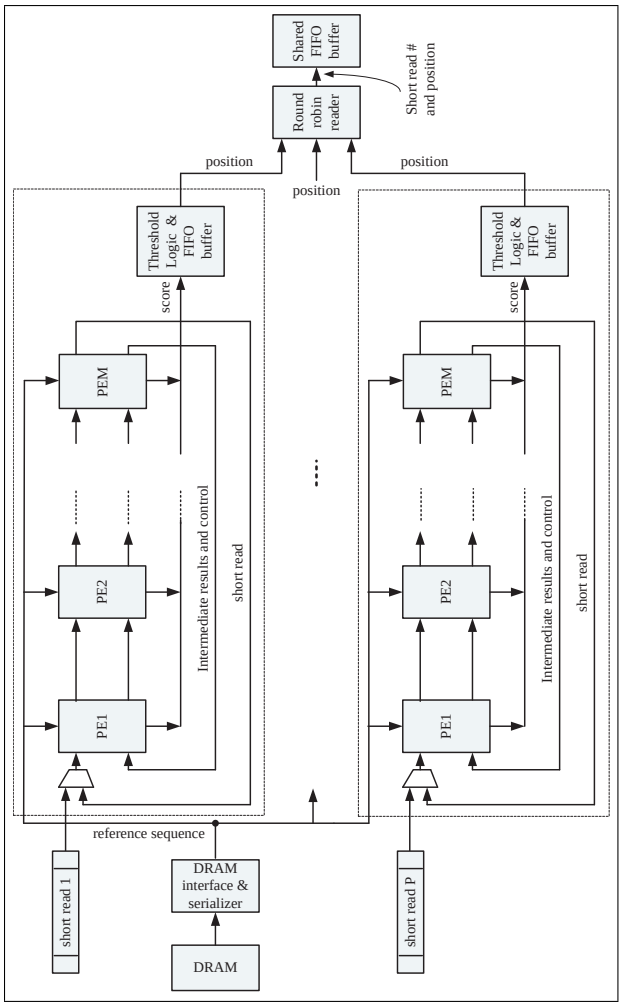

Figure 9: Multiple short-read alignment in a single FPGA using proposed semi-systolic array.

the fixed size folded semi-systolic array and it will be put in a future work.

The folded systolic array solution for the conventional systolic array is infeasible. This is due to the resulted folded systolic array would have a feed back FIFO depth of $N - v$ that depends on the length N of the reference sequence ($10^8 \leq N \leq 3 \times 10^9$). Because of the large size of this feed back FIFO, the memory resources of the FPGA will not be sufficient to implement it.

5 Implementation Results

This section discusses the resource utilization and performance evaluation of the proposed novel design (Design1) and the conventional design (Design2). The semi-systolic array of Design1, Fig. 3, and the systolic array of Design2 , Fig. 6, are described in VHDL language and implemented on Xilinx Vertix-5 FPGA (XC5VLX110) using Xilinx ISE8.1 tools. The semi-systolic array of Design1 utilizes 473 slices for $M = 64$, with a total of 17,280 slice, a maximum of 36 semi-systolic arrays can be fitted onto the XC5VLX110 FPGA. These utilization values are increased as the short read length increases as shown in Table 2. On the other hand, the systolic array of Design2 utilizes 470 slices for short read length of $M = 64$ with a maximum value of 36 systolic arrays can be fitted on the same FPGA. Also, this utilization increases as the short read length increases as shown in Table 2.

In Table 2, The columns entitled "#PE-D1" and "#PE-D2" represent the total number of PEs that can be fitted on the FPGA by the multiple parallel semi-systolic arrays of Design1 and the multiple parallel systolic arrays of Design2, respectively. The columns entitled "f_max-D1" and "f_max-D2" represent the maximum clock frequency obtained from the synthesis results for Design1 and Design2, respectively. The columns entitled "GCUPS-D1" and "GCUPS-D2" represent the Giga cell updates per seconds obtained for Design1 and Design2, respectively. The GCUPS design metric is a convenient measure to performance in sequence alignment implementation that it does not depend on the lengths of the two sequences being aligned. It is calculated by multiplying the number of PEs that are fitted on the the FPGA by the maximum clock frequency obtained for each design. The "speed up" and "FPGA utilization ratio" design metrics are calculated using the synthesis results in order to measure the degree of optimization achieved in each design. The "speed up" is calculated by dividing the GCUPS of Design2 "GCUPS-1" by the GCUPS of Design1 "GCUPS-D2", while the "FPGA utilization ratio" is calculated by dividing "#PE-D1" by "#PE-D2".

We notice from Table 2 that for large values of short read lengths ($M > 128$), the proposed design (Design1) has a slightly higher speed up over the conventional design (Design2) by ratios ranging from 0.07% to 0.15% as M increases. Also, it has a slightly better FPGA utilization over the conventional design by ratio of 0.005%. In spite of this slight increase in performance of the proposed design, the folded fixed size version of the proposed design - that we are working on it now - will achieve a significant increase in speed and FPGA utilization.

6 summary and conclusion

This paper presented a new semi-systolic array architecture for Stevens-Song optimized algorithm for DNA resequencing applications. This proposed architecture is efficient in area and speed - for large values of short read length - than the conventional systolic array architecture previously reported in the literature. Also, it can be easily optimized to deal with short read sequences, got from current HTSR sequencing platforms. Moreover, it can been modified to be reused for multiple pass processing in order to increase the number of processing elements that can be packed into a single FPGA and to increase the number of sequences that can be aligned in parallel in a single FPGA. This resolves the potential problem of many FPGA resources left unused for designs that have large values of short read length. The implementation results showed that the proposed design has a slight higher speed and higher FPGA utilization over the conventional design as short read length increases. In spite of this slight increase in performance of the proposed design, the folded fixed size version of the proposed design - that we are working on it now - will achieve a significant increase in speed and FPGA utilization.

Acknowledgements: The authors would like to acknowledge the support of the deanship of scientific research at Prince Sattam Bin AbdulAziz university under the research project # 2014/01/2025.

References:

[1] J. Shendure and H. Ji, "Next-generation dna sequencing," *Nat. Biotechnology*, vol. 26, 2008, pp. 1135–1145.

[2] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, and E. Sayers, "Genbank," *Nucleic Acids Research*, vol. 39, 2010, pp. D32–D37.

Table 2: Resource utilization and performance of Design1 and Design2 for different short read length M.

short read length (M)	Design1			Design2			speed up	FPFA utilization ratio
	f_max-D1 (MHZ)	#PE-D1	GCUPS-D1 (S)	f_max-D2 (MHZ)	#PE-D2	GCUPS-D2 (S)		
64	152	2117	321.8	150	2115	317.2	1.015	0.9988
128	151	2051	309.7	149	2050	304.4	1.014	0.9999
256	149	1944	289.7	148	1943	287.5	1.007	1.0005
512	147	2001	294.1	145	2000	290.0	1.014	1.0005
1024	144	2004	288.6	142	2003	284.4	1.015	1.0005

[3] T. Smith and M. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol*, vol. 147, 1981, pp. 195–197.

[4] P. McMahon, M. Kuttel, K. Stevens, and Y. Song, "Accelerating genomic sequence alignment using high performance reconfigurable computers," Master's thesis, University of Cape Town, South Africa, 2008.

[5] K. Stevens, H. Chen, T. Filiba, P. McMahon, and Y. Song, "Seqhive: A reconfigurable computer cluster for genome re-sequencing," in *2010 International Conference on Field Programmable Logic and Applications*, 2010, pp. 442–447.

[6] M. Chaisson and P. Pevzner, "Short read fragment assembly of bacterial genomes," *Genome Res*, vol. 18, 2008, pp. 324–330.

[7] M. Farrar, "Striped smith- waterman speeds database searches six times over other simd implementations," *Bioinformatics*, vol. 23, 2007, pp. 156–161, .

[8] L. Ligowski and W. Rudnicki, "An efficient implementation of smith waterman algorithm on gpu using cuda for massively parallel scanning of sequence databases," in *Proc. Of IEEE International Symposium on Parallel & Distributed Processing (IPDPS 2009)*, 2009, pp. 1– 8.

[9] P. Guerdoux-Jamet and D. Lavenier, "Samba, hardware accelerator for biological sequence comparison," *Bioinformatics*, vol. 13, 1997, pp. 609–615.

[10] E. Chow, J. Peterson, M. Waterman, T. Hunkapiller, and A. Zimmermann, "systolic array processor for biological information signal processing," in *Proc. of the 5th International Conference on Supercomputing (ICS 91)*, (USA), 1991, pp. 216–223.

[11] C. White and etl., "Bioscan, vlsi- based system for bio- sequence analysis," in *Proc. Of IEEE International Conference on Computer Design (ICCD 91)*, 1991, pp. 504–509.

[12] T. Han and S. Parameswaran, "Swasad, an asic design for high speed dna sequence matching," in *Proc. of the 2002 Asia and South Pacific Design Automation Conference (ASP- DAC 02)*, (USA), 2002, pp. 541–546.

[13] K. Benkrid, Y. Liu, and A. Benkrid, "A highly parameterized and efficient fpga- based skeleton for pairwise biological sequence alignment,"

IEEE Trans. On VLSI systems, vol. 17, 2009, pp. 561–570.

[14] M. Gokhale and etl., "Splash: a reconfigurable linear logic array," in *Int. Conf. on Parallel Processing*, 1990, pp. 526–532.

[15] G. Caffarena, C. Pedreira, C. Carreras, S. Bojanic, and O. Nieto-Taladriz, "Fpga acceleration for dna sequence alignment," *J. Circuits Syst. Compu*, vol. 16, 2007, pp. 245–266.

[16] T. Oliver, B. Schmidt, and D. Maskell, "Hyper customized processors for bio-sequence database scanning on fpgas," in *Proc. Of 13th International Symposium Field-Programmable Gate Arrays (FPGA05)*, 2005, pp. 229–237.

[17] S. Guccione and E. Keller, "Gene matching using jbits," in *Proc. of 12th Int. Conference on Field- Programmable Logic and Applications (FPL 02)*, 2002, pp. 1168–1171.

[18] L. Hasan, Z. Al-Ars, Z. Nawaz, and k. Bertels, "Hardware implementation of the smith waterman algorithm using recursive variable expansion," in *3rd International Design and Test Workshop (IDT 2008)*, 2008, pp. 135–140.

[19] C. Bio, "White paper on clc bioinformatics cube 1.03," in *Technical Report, CLC Bio*, (Denmark), 2007.

[20] N. Sebastio, T. Dias, N. Roma, and P. Flores, "Integrated accelerator architecture for dna sequences alignment with enhanced traceback phase," in *International Conference on High Performance Computing and Simulation (HPCS)*, 2010, pp. 16–23.

[21] S. Lloyd and Q. Snell, "Sequence alignment with traceback on reconfigurable hardware," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig 08)*, 2008, pp. 259–264.

[22] N. Sebastio, T. Dias, N. Roma, and P. Flores, "Hardware accelerator architecture for simultaneous short-read dna sequences alignment with enhanced traceback phase," *Microprocessors and Microsystems*, vol. 36, 2012, pp. 96–109.

[23] J. Arram, k. Tsoi, W. Luk, and P. Jiang, "Hardware acceleration of genetic sequence alignment," *Reconfigurable Computing: Architectures, Tools and Applications, Lecture Notes in Computer Science*, vol. 7806, 2013, pp. 13–24.

[24] B. Needleman and D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, 1970, pp. 443453.

[25] S. Rao, T. Kailath, "Regular iterative algorithms and their implementation on processor arrays," *Proc. IEEE*, Vol. 76, 1988, pp.259–269.

[26] S. Kung, VLSI Array Processors, Prentice- Hall, Englewood Cliffs, N.J., 1988.

[27] E. Abdel-Raheem, "Design and vlsi implementation of multirate filter banks," *Ph.D. thesis*, University of Victoria, Victoria, BC, 1995.

[28] F. El-Guibaly, A. Tawfik, "Mapping 3d iir digital filter onto systolic arrays," *Multidimensional Systems and Signal Processing*, Vol. 7, 1996, pp.7–26.

[29] A. Ibrahim, F. Gebali, H. El-Simary, A. Nassar, "Processor array architectures for scalable radix 4 montgomery modular multiplication algorithm," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 22, 2011, pp.1142–1149.

[30] A. Refiq, F. Gebali, "Processor array architectures for deep packet classification," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 17, 2006. pp. 241–252.

[31] D. Moldovan, J. Fortes, "Partitioning and mapping of algorithms into fixed size systolic arrays," *IEEE Trans. on Computers*, Vol. 35, 1986, pp.1–12.