Journal of information and communication convergence engineering
http://jicce.org

J. Inf. Commun. Converg. Eng. 14(1): 26-34, Mar. 2016 **Regular paper**

Optimized and Portable FPGA-Based Systolic Cell Architecture for Smith–Waterman-Based DNA Sequence Alignment

Hurmat Ali Shah[1], Laiq Hasan[2], and Insoo Koo[1*], *Member, KIICE*

[1]School of Electrical Engineering, University of Ulsan, Ulsan 44610, Korea
[2]Department of Computer Systems Engineering, University of Engineering & Technology, Peshawar, Pakistan

Abstract

The alignment of DNA sequences is one of the important processes in the field of bioinformatics. The Smith–Waterman algorithm (SWA) performs optimally for aligning sequences but is computationally expensive. Field programmable gate array (FPGA) performs the best on parameters such as cost, speed-up, and ease of re-configurability to implement SWA. The performance of FPGA-based SWA is dependent on efficient cell-basic implementation-unit design. In this paper, we present an optimized systolic cell design while avoiding oversimplification, very large-scale integration (VLSI)-level design, and direct mapping of iterative equations such as previous cell designs. The proposed design makes efficient use of hardware resources and provides portability as the proposed design is not based on gate-level details. Our cell design implementing a linear gap penalty resulted in a performance improvement of 32× over a GPP platform and surpassed the hardware utilization of another implementation by a factor of 4.23.

Index Terms: Bioinformatics, DNA sequence alignment, FPGA architecture, Smith–Waterman algorithm, Systolic cell

I. INTRODUCTION

Bioinformatics is the convergence of diverse and often divergent fields such as biology, mathematics, statistics, computer science, and computer engineering. Bioinformatics seeks to find answers to the all-relevant and all-important question of the origin of life and the processes of life. Bioinformatics takes a different perspective on life depending on the scale or scope on which life is considered. From the perspectives varying from molecular to organism to evolution, one process lies at the heart of it all: sequence alignment.

Sequence alignment is concerned with finding the relatedness of one sequence to another sequence. Relatedness indicates whether the sequences are homologous and whether they share a common domain and motifs or not. Finding relationships among biological sequences helps significantly in knowing concepts such as how proteins are related to each other in an organism and to other organisms with which this organism shares an evolutionary history, thus providing significant insight into the meaning and function of life. Moreover, sequence alignment has diverse applications such as drug engineering, disease diagnostics, detection of autoimmune disease, genetic engineering of plants and animals, determination of the structure of proteins, evolution tracing, and determination of protein

Received 17 September 2015, **Revised** 23 September 2015, **Accepted** 26 October 2015
***Corresponding Author** Insoo Koo (E-mail: iskoo@ulsan.ac.kr, Tel: +82-52-259-1249)
School of Electrical Engineering, University of Ulsan, 93 Daehak-ro, Nam-gu, Ulsan 44610, Korea.

Open Access **http://dx.doi.org/10.6109/jicce.2016.14.1.026** print ISSN: 2234-8255 online ISSN: 2234-8883

functions [1]. Sequence alignment compares two sequences by calculating the distance between the sequences. This distance represents the minimal cost at which one sequence can be converted into another. A score is assigned to each distance computed, and from this distance matrix, the degree of similarity or correlation between the sequences is inferred. For computing the abovementioned distance, two elementary operations are used, namely substitution and insertion/deletion (indel). Substitution signifies that one base in the sequence is replaced by another, while indel represents the insertion/deletion of a base into/from the sequence.

DNA is composed of four bases, namely adenine, cytosine, thymine, and guanine. When aligning two sequences, these bases are represented by the letters A, C, T, and G, respectively. Strings of these letters form a DNA sequence. Further, the basic alignment tool is the dot matrix. In this type of alignment, one sequence is arranged row-wise, while the other is arranged column-wise. A dot is placed in the cell that corresponds to a match of the letters of the row and the column.

As can be seen, DNA sequence alignment can be a very tedious and resource-intensive operation because DNA sequences can be millions of characters long and aligning them by conventional methods is not possible. For the alignment process to be meaningful, it needs to be accelerated. Such acceleration can be carried by using either software or hardware. Software acceleration employs heuristic methods and hence, is not optimal. For optimal acceleration, exact methods that compare each character against every character of the other sequence are required. Exact methods are very time consuming and resource intensive. To achieve optimality, they need to be carried out on dedicated hardware.

The Smith–Waterman algorithm (SWA) is an exact sequence matching algorithm that performs optimally as it can return local alignments and thus, provide considerably accurate information about the similarity between sequences that are aligned against each other [2]. SWA is a dynamic programming-based algorithm that decomposes the alignment problem into sub-problems and iteratively solves the given problem [3]. SWA has the space and time complexity of $O(MN)$, where M and N denote the lengths of the sequences to be aligned. The space and time complexity is the best among other alignment methods and provides accurate and local alignment [3].

SWA is a resource-intensive operation and thus, needs dedicated hardware for efficient and fast operation. Different hardware implementation candidates can be found, such as FPGA, GPU, and cell BE platforms. In [4], a comparison between the FPGA, GPU, and cell BE reconfigurable platforms for aligning biological sequences on the basis of speed, energy consumption, and developmental costs has been presented. Further, it has reported that FPGA outperforms the other two platforms in terms of performance per watt and performance per dollar spent. The advantage of FPGA in high-performance computing is that, if it is used as an accelerator, the computing density can be increased significantly as FPGA benefits from the increased clock speed as opposed to general microprocessors that have reached their maximum clock speed [5].

Because of the low cost and speed-related advantages, all recent hardware-based SWA implementations are carried out on FPGAs [6]. The performance of an implementation on FPGA depends upon the cell design that does the calculation for SWA. This work focuses on designing an efficient systolic cell for implementing SWA on FPGA. The main objective of this work is to design a systolic cell that uses relatively few hardware resources but at the same time, is not designed on a low-level device/platform and does not depend on the specific details of the low-level device/platform. Therefore, the design can be implemented on any other FPGA device or platform. Our implementation has a trade-off between complexity and performance. Moreover, it avoids both oversimplification and wastage of hardware resources by avoiding a direct mapping of the iterative equations to the hardware. This work should be considered in a broader framework developed by [7] and focuses particularly on designing a cell that optimally fits the system design of [7].

The performance issues associated with the transfer of the required data to FPGA and back has been studied by [8]. GPU platforms are explored in the same capacity as FPGA. The authors in [9] studied the well-known seed-and-extend algorithm to accelerate sequence alignment. Meaningful alignment often requires the alignment of short DNA sequences. The mechanisms and specifics of the alignment of short DNA sequences are sometimes divergent with the goals of the alignment of large sequences. The authors in [10] developed a platform that through GPU could accelerate the alignment of short sequences. Other architectures are also explored for accelerating the alignment of DNA sequences. In [11], a single instruction multiple data (SIMD)-based architecture is implemented on an application-specific instruction-set processor (ASIP) for aligning DNA sequences.

The rest of this paper is organized as follows: Section II discusses the SWA in detail. Section III presents the principles on which the proposed cell design is based. Section IV describes the proposed cell design. Section V presents the results of this study, and Section VI concludes the paper.

II. SMITH–WATERMAN ALGORITHM

Needleman–Wunch [12] presented an efficient algorithm

(NW algorithm) for aligning two sequences. The method was exact and thus, produced optimal results. However, it performed global alignments. Global alignments are good when the sequences to be aligned are closely related as end-to-end alignment highlights the similarity across their lengths but they fail to perform optimally when the sequences to be aligned do not have large homology. Smith and Waterman presented a variant of the NW algorithm in 1984 and called it the Smith–Waterman algorithm (SWA) [3]. SWA finds the local alignment and works well for sequences that are distantly related as such sequences do not manifest the overall similarity but have regions that have concentrated local similarities.

Local alignments are carried out by making small changes to the equations that perform global alignments. SWA like the NW algorithm consists of three stages for performing alignment, namely initialization, matrix fill, and trace back. The initialization and matrix fill stages resemble those of the NW algorithm, while the trace back stage differs from that of the NW algorithm. These three stages can be described as follows:

Initialization: Assume that we have two sequences *x* and *y* of length *i* and *j*, respectively, are to be aligned. The initialization step is different for the NW algorithm and SWA in that the top row and the leftmost column are initialized to 0 in the SWA, whereas in the NW algorithm, the upper-rightmost element (0,0) is set to zero and the top row and the leftmost column are initialized with the cost of gap penalties of length *i* and *j*.

Matrix fill: In the matrix fill stage, two characters are compared to each other and a score is assigned to the comparison on the basis of (1), where $F(i,j)$ denotes the element that is computed presently, $s(xi,yj)$ represents the similarity score that is selected on the basis of the similarity between the two characters, while d indicates the gap penalty.

$$F(i,j) = max \begin{cases} 0, \\ F(i-1,j-1) + s(xi,yj), \\ F(i-1,j) - d, \\ F(i,j-1) - d, \end{cases} \quad (1)$$

(1) uses a linear gap penalty scheme as every gap is penalized in the same way. A more biologically relevant method for handling gaps is the affine gap model that penalizes opening a gap more than extending an already opened gap. Such a scheme is given in (2).

$$H(i,j) = \max(0, H(i-1,j-1) + S(i,j), E(i,j), F(i,j)), \quad (2)$$

$$F(i,j) = \max(H(i-1,j) - o, F(i-1,j) - e), \quad (3)$$

$$E(i,j) = \max(H(i,j-1) - o, E(i,j-1) - e). \quad (4)$$

In (3) and (4), *o* denotes the gap opening penalty, while *e* represents the gap extension penalty. The matrices $F(i,j)$ and $E(i,j)$ contain the trace of opening and extending a gap, respectively. $F(i,j)$ stores the cost for opening a gap and extending a gap on sequence *x*, while $E(i,j)$ stores the cost for opening and extending a gap on sequence *y*.

Trace back: In SWA, trace back starts from the element that has the maximum score. Moving in the direction of the cell that has the highest value, 0 is encountered and the alignment is stopped. Indels are created; i.e., "—" is inserted in the alignment for the vertical or horizontal selection of the cell instead of the diagonal movement. In the affine gap model, three separate matrices need to be stored, whereas in trace back, all the matrices are searched to find the highest value.

III. SYSTEM MODEL

Before presenting the system model, we will evaluate the performance of the SWA used in the proposed design.

A. Performance Evaluation of SWA

SWA performs optimally on many performance indicators such as accuracy, parallelism, and space and time complexity [13]. SWA is considered the most optimal because it performs the alignment of every individual character in one sequence against every character in the other sequence. This results in the massive space complexity of $O(MN)$, where M and N denote the sizes of the two sequences, respectively. However, SWA also computes the cell's value independent of the reset of the matrix and dependent only upon the three previous cells and the similarity score.

Both fine-grained and coarse-grained parallelism can be exploited in SWA. Fine-grained parallelism works on making the required data available for a cell and then computing the values of these cells in parallel; the values of the three previous cells are available for this computation. Coarse-grained parallelism is achieved by dividing the alignment problem into sub-problems and solving the sub-problems in parallel, thus reducing the overall time complexity.

In calculations that are not parallel, computing the entire matrix takes $M \times N$ cycles, which increases tremendously with an increase in the sequence length. From Fig. 1, we can see that diagonal elements in the matrix can be computed in parallel; therefore, the time complexity can be reduced to the number of diagonals in a matrix. There exist $M + N - 1$ diagonals in an array; therefore, two sequences having lengths of M and N can be aligned in $M + N - 1$ cycles.

B. Systolic Cell Design for SWA

This section presents an efficient PE design for aligning two sequences by using a linear systolic array (LSA) on FPGA. The systolic array was introduced by Kung and Leiserson [15]. Systolic arrays have proven to be significantly efficient in parallelizing computing designs. These arrays have been used for performing the complex function of multiplying two long sequences: the sequences are arranged in rows and columns, and two elements from the sequences are multiplied by the PE in a systolic array. The idea of systolic arrays was used by Lipton and Lopresti [16] in 1985 for implementing the edit distance algorithm, a standard global alignment algorithm for DNA sequences. In 1987, on the basis of the initial results, which showed an improvement in speed in the order of hundreds to thousands of times, Lopresti developed the first full-fledged system for aligning nucleic sequences. This system was called P-NAC. Further, BISP, Bio-Scan, B-Sys, Splash/Splash 2, and Kestrel employ systolic arrays for aligning two sequences.

When SWA is mapped to a systolic array, each PE computes one element of the alignment matrix at a time, and an entire column is computed during the time span of the entire operation. Further a PE array is mapped to the anti-diagonal lines of the alignment matrix. A PE that is composed of elements that perform simple operations such as addition, subtraction, and comparison makes the design very fast. The focus is always on making the PE simpler, faster, and thus, more efficient. The overall system design is adopted from our previous work as presented in [7]; the corresponding block diagram is illustrated in Fig. 1.

To achieve the goal of designing an efficient PE, it is imperative to reduce the area that a PE uses. Besides being slow, oversized PEs waste significant hardware resources. This reduces the available resources that can be used for realizing a large number of PEs and hence, reducing the size of the LSA. The size of the LSA controls the degree of parallelism and thus, the speed-up achieved. Oversized PE also adversely affects the clock frequency through which the LSA operates. As the number of logic units increases in a PE, the clock will need more time to traverse to the end, subsequently affecting the rate at which the computation of the PE can be completed.

Previous design methods of PE can be broadly categorized into three categories: The first one oversimplifies the design of PE. The second approach employs a direct implementation of an iterative equation to the LSA. The third approach attempts to carry out optimizations at the VLSI level by integrating two PEs. All these approaches have problems of their own. The first method calculates only the edit distance between two sequences and thus, results in non-used hardware resources. The second results in an overuse of hardware resources, while the third results in compatibility issues as such a design cannot be migrated to other platforms.

Therefore, we need to come up with a design that avoids all of the abovementioned pitfalls. A design which is not overly-simplified and does not overuse resources, and which is also compatible with other platforms is needed. The next section looks at such a design for aligning two sequences by using SWA with linear gap penalties.

IV. CELL DESIGN

This section presents the proposed design for SWA. First, the cell design for SWA with a linear gap penalty is presented, and then, the cell design for SWA with an affine gap penalty is discussed.

A. PE Design for SWA with Linear Gap Penalty

A portable design that judiciously uses hardware is possible if the iterative equations of SWA can be simplified and captured in standard hardware description languages (HDLs) such as Verilog. The major bottleneck with the recursive equation of SWA is that the data precision of the computed elements is very high. For the alignment of DNA sequences, a data precision of 16 bits is used. The logic units that a PE uses are very basic, such as adders and comparators, but the size of these units increases the hardware resources consumed by an individual PE. Therefore, the size of the logic units needs to be reduced to achieve an overall improvement in the hardware utilization. If Lopresti's observation is realized in the field of PE design, significant savings in hardware usage can be achieved.

In 1985, Lipton and Lopresti [16] observed that the values of the top element b and the left element c are restricted to a certain distance from the corner element, i.e., $a + 1$ or $a - 1$. Therefore, the iterative equations of SWA can be modified to accommodate this fact as in (5):

$$a = \begin{cases} a & if\ b\ or\ c = a - 1\ or\ Si = Ti \\ a + 2 & if\ c = a + 1\ and\ Si \neq Ti \end{cases}. \quad (5)$$

In (5), S and T denotes the two sequences to be matched, and i represents the index of both the sequences. In (5), the intermediate values that are used for calculating the final $F(i,j)$ can be derived from the corner element. These new values differ from the corner values by +1 or –1 or by 0. Therefore, two bits can be used for representing them. The calculations of the difference vectors are expressed in (6) and (7);

$$D(i-1,j) = F(i-1,j)_{15:4} - F(i-1,j-1)_{15:4}, \quad (6)$$

$$D(i-1,j) = F(i-1,j-1)_{3:0} + Similarity\ score. \quad (7)$$

The values as computed in (6) and (7) are used as intermediate values, and thus, the adders and comparators involved compare two inputs of two bits each instead of two inputs of 16 bits each. The output $D(i,j)$ is added to $F(i-1,j-1)_{15:4}$ in order to obtain $F(i,j)$. Fig. 2 shows the basic PE design for the SWA linear gap penalties. The sequence comparison unit compares two characters, one from each sequence, and generates a similarity score on the basis of this comparison. The sequence Ns is shifted by one, and the current Ns is buffered by one cycle as it is used in the next PE after one clock cycle. The difference bits from the left element and the diagonal elements are separated in the first clock cycle. A subtraction unit subtracts the diagonal element from the left element to obtain the difference vector. This difference vector is used in the rest of the calculations. The remaining 12 bits from the diagonal element are added to the resultant difference vector in order to obtain the value of the present cell. The final difference vector is buffered for one clock cycle, and this buffered value is used as the top element in the next clock cycle, as in LSA. Then, in the next clock cycle, the lower value in the column is computed by the PE. The value of the cell is compared with Max_in, i.e., the global maximum, to find the global maximum. This global maximum is then

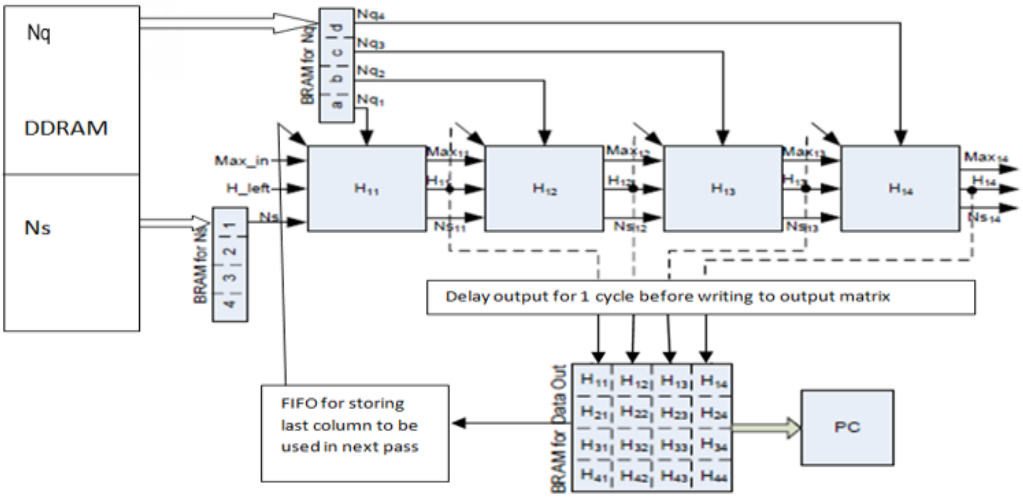

Fig. 1. Block diagram of the overall system design.

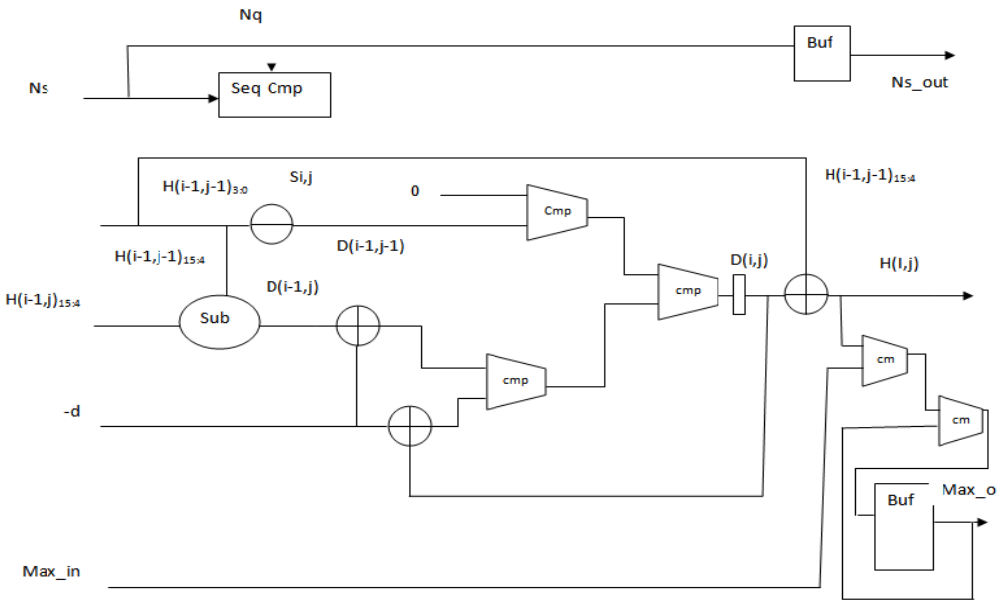

Fig. 2. Cell for the linear gap penalty.

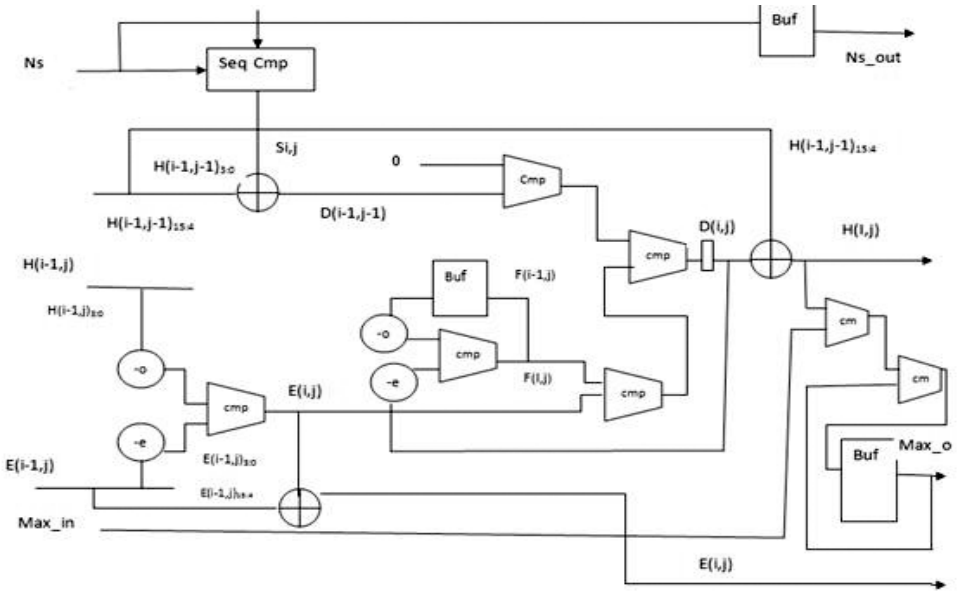

Fig. 3. Cell for affine gap penalty.

compared with the current maximum value of the cell, which is stored in a buffer, and the greater of the two is stored. This value is also forwarded to the next PE as the global maximum. This optimized cell uses two-bit units for computing the iterative equations of SWA. The last adder that adds the bits from the diagonal element to the difference vector is a 16-bit adder, and the comparators that compute the global maximum and the maximum value of the cell are 16 bit wide.

B. PE Design for SWA with Affine Gap Penalty

Affine gap models are used for aptly representing the biological fact of the costliness of opening a gap than that of extending it. An affine gap model penalizes opening a gap more than extending an already opened one. (2)–(4) present the Smith–Waterman iterative equations for affine gap models. Fig. 3 presents the block diagram for an affine gap penalty cell.

Two new units are added to calculate $E(i,j)$ and $F(i,j)$. $E(i,j)$ and $F(i,j)$ amortize the costs of opening new gaps on the first sequence and the second sequence, respectively. Difference bits are taken from the left element and are subtracted from the value of the gap opening penalty, i.e., o, while from $E(i-1,j)$, the previous PE gap extension penalty e is subtracted; then, both these values are compared to find the difference vector for $E(i,j)$. The value of $E(i,j)$ is forwarded to the next PE. The value of $F(i,j)$ need not be taken as an input from the previous PE; instead, it is generated in the current PE by itself. The current value of $F(i,j)$ is buffered as the value of the present difference vector is delayed by one clock cycle in order to be used in the next cycle for generating $F(i,j)$. From these values, o and e are subtracted to generate the difference vectors of $F(i,j)$. The maximum values of $E(i,j)$ and $F(i,j)$ are compared with the maximum of 0 and the difference vector of the left element to find the current difference vector. To this difference vector, the remaining bits of the diagonal element are added to get the present value of the cell. The calculation of the maximum value is the same as that of the linear gap penalty cell.

V. RESULTS AND DISCUSSION

In order to reduce the cost incurred by acceleration through FPGA, we implemented the design on low-cost Spartan-3 FPGA, a product of Xilinx. Spartan-3 is a low-priced device of Xilinx but is sufficient for implementing a design of low–mid-level complexity. It does not contain many logic slices to realize a high-performance design. It does not allow the implementation of PEs in the order of hundreds. The implementation of only a limited number of PEs is allowed. This affects the overall speed-up achieved. The speed-up achieved is at a cost incurred such as energy consumption and hardware resources used, thus, a balance has to be maintained between speed-up achieved and the cost incurred.

The most often used performance metric for computational biology applications is the number of cells updated per second (CUPS), and it can be calculated as expressed in(8):

$$Performance\ in\ CUPS = No.of\ PEs \times Operating\ frequency \quad (8)$$

The device that we used is Spartan 3 XC3S250E. We could instantiate an array of PEs of size 35 on the said device. The operating frequency shown by the post-place and route simulation model was 50 MHz. The proposed cell design implemented in the framework results in 1.75 GCUPS for the affine gap penalty cell and 2.75 for the linear gap penalty cell. This is a 20× and 32× performance improvement over the GPP implementation described in [4]. Table 1 presents the performance improvement.

The authors of [17] built highly customizable PEs for the alignment of biological sequences on FPGA. They could implement an array of PEs of size 13 on a Spartan 3 1500 chip. Table 2 presents the efficiency of the proposed design in terms of the hardware utilization. The proposed design more efficiently utilizes hardware as it can realize more PEs by using limited hardware resources in contrast to the implementation described in [5] that realized fewer PEs by using a lot of the hardware resources available.

The number of PEs realized can be increased if our memory utilization logic is not implemented and an UART that directly sends the output is used. Performance can be improved by using a device that has a higher clock speed and has more hardware resources available.

Table 1. Performance improvement in terms of CUPS

Implementation	Performance
DNA sequence alignment over GPP [4]	0.085 GCUPS
Proposed implementation with affine gap penalty scheme	1.75 GCUPS
Performance improvement	20×
Proposed implementation with linear gap penalty scheme	2.75 GCUPS
Performance improvement	32×

GCUPS: giga cell updated per second, GPP: general-purpose processor.

Table 2. Hardware utilization improvement

Implementation	Device	Size of PE array
Customized processor for biological sequence alignment on FPGA [17]	Spartan 3 1500	13
Proposed affine gap penalty scheme design	Spartan 3 XC3S250	35
Proposed linear gap penalty scheme design	Spartan XC3S250	55
Hardware utilization improvement (linear gap penalty scheme)		4.23
Hardware utilization improvement (linear gap penalty scheme)		2.69

FPGA: field programmable gate array, PE: processing element.

Implement-ations such as the implementation described in [18] achieve very high performance but at a very high cost; further, the design cannot be implemented on any other platform because it incorporates the VLSI-level details as in contrast, the proposed system achieves comparable performance at a very low cost.

VI. CONCLUSION

In this paper, we presented an efficient and low-hardware-resource-consuming systolic cell design for implementing SWA on FPGA. Cell design is the most important part of designing an efficient and fast implementation of SWA on FPGA because the actual matching takes place here. Previous designs either oversimplified the cell design for implementing the iterative equations or avoided optimization altogether and directly mapped the iterative equations to the hardware. This resulted in either inflexibility or the wastage of hardware resources.

The proposed design aimed at a trade-off between flexibility and performance. Equivalent equations of the iterative equations that require fewer iterations for the implementation were used instead of the original iterative equations. The results showed performance improvement over a GPP implementation in the order of 32× for the linear gap penalty scheme and of 20× for the affine gap penalty scheme. Moreover, the results were compared with those of a design that was implemented on a device that had more hardware resources than our Spartan XC3S250. Further, we showed that the proposed design surpassed the hardware resource utilization of the other design by a factor of 4.29 in the case of the linear gap penalty scheme.

REFERENCES

[1] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, no. 3, article no. R25, 2006.

[2] L. Hasan, Z. Al-Ars, and S. Vassiliadis, "Hardware acceleration of sequence alignment algorithms: an overview," in *Proceedings of IEEE International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, Rabat, Moroco, pp. 92-97, 2007.

[3] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195-197, 1981.

[4] K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu, and X. Tian, "High performance biological pairwise sequence alignment: FPGA versus GPU versus cell BE versus GPP," *International Journal of Reconfigurable Computing*, vol. 2012, article no. 7, 2012.

[5] R. Wain, I. Bush, M. Guest, M. Deegan, I. Kozin, and C. Kitchen, *An Overview of FPGAs and FPGA Programming: Initial Experiences at Daresbury*. Cheshire: Council for the Central Laboratory of the Research Councils, 2006.

[6] M. Gok and C. Yilmaz, "Efficient cell designs for systolic smith-waterman implementations," in *Proceedings of IEEE International Conference on Field Programmable Logic and Applications (FPL'06)*, Madrid, Spain, pp. 1-4, 2006.

[7] H. A. Shah, L. Hasan, and N. Ahmad, "An optimized and low-cost FPGA-based DNA sequence alignment: a step towards personal genomics," in *Proceedings of 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Osaka, Japan, pp. 2696-2699, 2013.

[8] T. Moorthy, J. M. Correa, and S. Gopalakrishnan, "Gigabyte-scale alignment of biological sequences: a case study of IO bandwidth reconfiguration for FPGA acceleration," in *Proceedings of 26th Annual IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, Regina, SK, pp. 1-4, 2013.

[9] R. Wilton, T. Budavari, B. Langmead, S. J. Wheelan, S. Salzberg, and A. Szalay, "Faster sequence alignment through GPU-accelerated restriction of the seed-and-extend search space," *BioRxiv*, 2014 [Internet], Available: http://dx.doi.org/10.1101/007641.

[10] Y. Liu and B. Schmidt, "GSWABE: faster GPU-accelerated sequence alignment with optimal alignment retrieval for short DNA sequences," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 4, pp. 958-972, 2015.

[11] N. Neves, N. Sebastiao, A. Patricio, D. Matos, P. Tomas, P. Flores, and N. Roma, "BioBlaze: multi-core SIMD ASIP for DNA sequence alignment," in *Proceedings of 24th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, Washington, DC, pp. 241-244, 2013.

[12] S. B. Needleman and C. 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, no. 3, pp. 443-453, 1970.

[13] L. Hasan, "Hardware acceleration of biological sequence alignment application," Ph.D. dissertation, Delft University of Technology, The Netherlands, 2011

[14] C. W. Yu, K. H. Kwong, K. H. Lee, and P. H. Leong, "A Smith-Waterman systolic cell," in *New Algorithms, Architectures and Applications for Reconfigurable Computing*. New York, NY: Springer, pp. 291-300, 2005.

[15] H. T. Kung and C. E. Leiserson, "Algorithms for VLSI processor arrays," in *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, pp. 271-292, 1980.

[16] R. J. Lipton and D. Lopresti, "A systolic array for rapid string comparison," in *Proceedings of the Chapel Hill Conference on VLSI*, Chapel Hill, NC, pp. 363-376, 1985.

[17] T. Oliver, B. Schmidt, and D. Maskell, "Hyper customized processors for bio-sequence database scanning on FPGAs," in *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, pp. 229-237, 2005.

[18] P. Zhang, G. Tan, and G. R. Gao, "Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform," in *Proceedings of the 1st International Workshop on High-Performance Reconfigurable Computing Technology and Applications*, Reno, NV, pp. 39-48, 2007.

Hurmat Ali Shah
received his M.Sc. and B.Sc. in Computer Systems Engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2013 and 2010, respectively. He is currently pursuing his Ph.D. degree at the Multimedia Communications System Laboratory, University of Ulsan, South Korea. His research interests include secure spectrum sensing in cognitive radio networks, next-generation communications, and wireless sensor networks.

Laiq Hasan
received his Ph.D. in Computer Engineering from Delft University of Technology, Delft, The Netherlands, in 2011. He received his M.Sc. and B.Sc. (with honors) in Electrical Engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2003 and 2000, respectively. He worked as Visiting Researcher in Faculty of Electrical Engineering Mathematics and Computer Science at Delft University of Technology, Delft, The Netherlands. Currently, he is Chairman of the Department of Computer Systems Engineering at the University of Engineering and Technology, Peshawar, Pakistan. His research interests include logic design, computer architecture, bioinformatics, performance analysis, and optimization. Besides publishing a technical book and a book chapter, he has authored and co-authored many research papers in journals of international repute. He has also presented his research work at various international conferences.

Insoo Koo
received his B.E. degree from Konkuk University, Seoul, Korea, in 1996, and his M.S. and Ph.D. degrees from Gwangju Institute of Science and Technology (GIST), Gwangju, Korea, in 1998 and 2002, respectively. From 2002 to 2004, he worked with Ultrafast Fiber-Optic Networks (UFON) Research Center in GIST, as a research professor. For one year from September 2003, he was a visiting scholar at Royal Institute of Science and Technology, Sweden. In 2005, he joined University of Ulsan, where he is now a full professor. His research interests include next-generation wireless communication systems and wireless sensor networks.