Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2013) 21: 246 – 262
© TÜBİTAK
doi:10.3906/elk-1105-22

System designs to perform bioinformatics sequence alignment

Çağlar YILMAZ[1,*], Mustafa GÖK[2]
[1]Department of Computer Engineering, Çukurova University, 01330 Adana, Turkey
[2]Department of Electrical Engineering, Çukurova University, 01330 Adana, Turkey

Received: 12.05.2011 • **Accepted:** 01.10.2011 • **Published Online:** 27.12.2012 • **Printed:** 28.01.2013

Abstract: The emerging field of bioinformatics uses computing as a tool to understand biology. Biological data of organisms (nucleotide and amino acid sequences) are stored in databases that contain billions of records. In order to process the vast amount of data in a reasonable time, high-performance analysis systems are developed. The main operation shared by the analysis tools is the search for matching patterns between sequences of data (sequence alignment). In this paper, we present 2 systems that can perform pairwise and multiple sequence alignment operations. Through the optimized design methods, proposed systems achieve up to 3.6 times more performance compared to the previous designs. The proposed systems are modeled with VHDL; these models are simulated and then mapped on a Mezzanine card that contains an FPGA chip. The systolic processing core implemented on the Mezzanine card processes the data obtained from the PC and sends the results back to the PC. The practical functionality of the systems is tested and verified by comparing the system results with pure software results.

Key words: Bioinformatics, sequence alignment, FPGA

1. Introduction

The common definition of bioinformatics is research, development, or application of computational tools and approaches for expanding the use of biological, medical, behavioral, or health data, including those to acquire, store, organize, archive, analyze, or visualize such data [1]. This rapidly developing branch is expected to provide solutions from finding cures for diseases to minimizing global warming, and from extending the lifetime of humans to developing agricultural production techniques to end starvation [2]. Therefore, bioinformatics has become one of the most important research areas in developed countries. The budget of the National Center for Biotechnology Information (NCBI) [3] (founded by the United States to support bioinformatics research) was US$83 million in 2009. This foundation provides databases to store biological data and online tools to search these databases [4]. A good example for appreciating the size of the total data is the Human Genome Project. The project was completed in 2003 with the help of 6 developed countries. The rate of growth in biological data can be observed from GenBank statistics, as well. This database contained 680,000 base pairs in 1982. It was reported in August 2009 that the total number of records in the databases of GenBank was over 250 billion. GenBank and similar databases are expected to grow with this exponential trend as several genome projects continue. It is obvious that fast and accurate analysis of biological data is demanded for high-performance systems. The most crucial step of the analysis is the search for matching patterns among sequences of data (sequence alignment). This work aims to contribute to this goal by presenting new sequence alignment systems.

[*]Correspondence: cagyil@cu.edu.tr

The fundamental assumption in bioinformatics is that there is an evolutionary and functional linkage between similar sequences. This similarity can be local or global. The similarity of the sequences can be found by using sequence alignment operations. These operations can be briefly defined as follows: global sequence alignment attempts to align every residue in every corresponding sequence. It is only useful when sequences in a query set are similar and have approximately equal length. Local sequence alignment aligns every subsequence in one sequence to every subsequence in another sequence. One of the systems presented in this paper performs local sequence alignment, since it is used more commonly than global alignment. Multiple sequence alignment performs sequence alignment operations among 3 or more sequences. It can also be global or local; however, it is much more important to identify similar subregions among sequences than to identify the global similarity among them.

Several algorithms have been developed to align sequences. The most popular ones are FASTA [5], BLAST [6], and Smith–Waterman (SW) [7]. The FASTA and BLAST algorithms are included in several software packages. These algorithms produce alignments in short amounts of time; however, the quality of these alignments is low since they use heuristic methods. On the other hand, the SW algorithm produces alignments with optimal quality. Therefore, it is guaranteed that the alignment identifies the maximum possible similarity between sequences. However, this algorithm suffers from its computational complexity and requires more time and storage resources.

Multiple sequence alignment is generally applied to protein sequences and identifies functional and evolutionary relationships among 3 or more sequences. Multiple sequence alignment operation is much more complex than pairwise sequence alignment and requires more sophisticated methods. Generally, statistical methods [8, 9] and progressive methods such as CLUSTALW [10] and T-COFFEE [11] are used to perform multiple sequence alignment.

Sequence alignment algorithms are executed on high-performance systems to achieve high-quality results in a reasonable amount of time. Two types of hardware platforms can be selected: supercomputers (in this context, computer clusters can be considered as equivalent to supercomputers) or application-specific processors. Supercomputers have high design and maintenance costs. Access to these systems is also usually difficult, since they are shared by many researchers. The main advantage of the supercomputer platforms is that they can execute a large variety of algorithms with suitable software support.

Sequence alignment systems implemented on field-programmable gate arrays (FPGAs) have much lower design and maintenance costs. A systolic array of processing elements implemented on an FPGA can execute a dedicated algorithm with a performance equivalent to that of a supercomputer. Designs can also be upgraded to new platforms effortlessly when a new FPGA technology is made available. The first FPGA processor for bioinformatics applications was Splash-2 [12], which was designed with a common architecture for text searching, sequence alignment, and edge detection. It can compare 12 million symbols per second and can compute edit distances between these symbols. An updated version of Splash-2 is provided by Xilinx [13], which is optimized for their FPGA platforms. That improved version of Splash-2 is 20 to 100 times faster than the original Splash-2. However, both versions of Splash-2 compute only edit distances between 2 sequences, which are rarely used in practice. Most alignment algorithms used in practice require gap penalties and substitution tables. Yamaguchi et al. presented the first FPGA design that computes these parameters [14]. This work also addresses the problem of fitting long query sequences on FPGAs. On the other hand, state-of-the-art FPGAs have enough capacity to fit even the longest query sequences into a single chip. One of the most recent FPGA designs was reported by Oliver et al. [15]. The novelty of their work was the efficient computation of the maximum score values.

In addition to the basic sequence alignment hardware accelerators, a relatively low number of FPGA hardware accelerators are presented for multiple sequence alignment algorithms. Dawning 4000H, designed by Lin et al., aimed to improve the first step of the CLUSTALW algorithm [16]. However, they altered the first step of the CLUSTALW algorithm to simplify the hardware implementation. This approach might reduce the quality of the biological deduction. The system can only perform the alignment of nucleotide sequences. Another recent system was designed by Oliver et al. [17]. This system stays loyal to the original algorithm and supports protein sequence alignment.

All the previously cited designs for both pairwise sequence alignment and multiple sequence alignment directly map the SW algorithm in their core processing units, due to the SW algorithm's capacity of finding the optimal alignment. However, unoptimized mapping reduces the number of processing elements that can be fitted on the FPGAs, resulting in a significant decrease in performance. To overcome this problem, in [18] efficient cell design methods for systolic SW applications were presented. These methods were applied on protein sequence alignment applications in [19]. An optimized system for multiple sequence alignment was also presented in [20]. This paper extends the previous research presented in [18]–[20]. It differs from the previous work of the authors in the sense that it presents 2 fully functional systems (including the software interfaces) that perform pairwise and multiple sequence alignment operations, whereas the previous work was focused on the development of the hardware. The main contributions of this paper are as follows:

- The proposed pairwise sequence alignment system has a systolic architecture core, which is designed by using the optimized processing elements. The optimization reduces the area of the processing elements and also increases the speed of the overall system.

- The proposed multiple sequence alignment system has a systolic architecture core, which is optimized for the first step of the CLUSTALW algorithm. Similarly, this optimization provides better performance for multiple sequence alignment.

- Proposed systems are designed with whole hardware and software support. Software implementation is generally neglected in the previous work. However, the functional verification of a sequence alignment system cannot be fully achieved without providing the software hardware integration.

This paper is organized as follows: Section 2 explains pairwise and multiple sequence alignment algorithms. Section 3 presents the proposed designs. Section 4 discusses the synthesis and simulation results. Section 5 presents the conclusions.

2. Sequence alignment algorithms

Four nucleic acid pair bases construct DNA and RNA, whereas 20 different amino acids are the basic building blocks of all proteins. Therefore, DNA and RNA can be represented using a string derived from a 4-letter alphabet, and proteins can be represented using a string derived from a 20-letter alphabet. In biology, this kind of representation of proteins is called the primary structure of proteins, which neglects the 3-dimensional organization of the molecules. The following example is the protein sequence of influenza A/H1N1 (the infamous swine flu) virus gathered from the NCBI database [21].

Table 1. Similarity score matrix.

	Φ	**S**	**L**	**Q**	**I**	**F**	**R**	**Y**	**N**	**S**	**H**
Φ	0	0	0	0	0	0	0	0	0	0	0
S	0	4	0	0	0	0	0	0	1	4	0
A	0	1	3	0	0	0	0	0	0	2	2
Q	0	0	0	8	3	0	1	0	0	0	2
L	0	0	4	3	10	5	0	0	0	0	0
G	0	0	0	2	5	7	3	0	0	0	0
I	0	0	2	0	6	5	4	2	0	0	0
F	0	0	0	0	1	12	7	7	2	0	0
T	0	1	0	0	0	7	11	6	7	3	0
S	0	4	0	0	0	2	6	9	7	11	6
H	0	0	1	0	0	0	2	8	10	6	19

```
  1 | DTVLEKNVTV   THSVNLLEDK   HNGKLCKLRG   VAPLHLGKCN   IAGWILGNPE
 51 | CESLSTASSW   SYIVETSSSD   NGTCYPGDFI   DYEELREQLS   SVSSFERFEI
101 | FPKTSSWPNH   DSNKGVTAAC   PHAGAKSFYK   NLIWLVKKGN   SYPKLSKSYI
151 | NDKGKEVLVL   WGIHHPSTSA   DQQSLYQNAD   AYVFVGSSRY   SKKFKPEIAI
201 | RPKVRDQEGR   MNYYWTLVEP   GDKITFEATG   NLVVPRYAFA   MERNAGSGII
251 | ISDTPVHDCN   TTCQTPKKTS   LPFQNIHPIT   IGKCPKYVKS   TKLRLATGLR
301 | NVPSIQSRGL   FGA
```

As seen from this example, the protein sequence of the virus is represented using a string of letters, where each letter is an abbreviation for an amino acid. This sequence and similar sequences are stored in biological databases in text format. It is possible to search these databases to find similar strings to a query sequence using sequence alignment algorithms. In the rest of this section, 2 popular algorithms used to perform pairwise and multiple sequence alignment operations are discussed.

2.1. Pairwise sequence alignment and SW algorithm

Pairwise sequence alignment is the quest of finding similarities between 2 sequences. Basically, this task is used to find similar sequences in a biological database to a query sequence. For example, the H1N1 virus sequence is compared with other virus sequences to help develop a vaccination or a treatment. The following illustrates a pairwise sequence alignment between 2 sequences, P and Q. This alignment is produced by the SW algorithm using the BLOSUM62 substitution matrix.

```
Before alignment
P  =  S  L  Q  I  F  R  Y  N  S  H
Q  =  S  A  Q  L  G  I  F  T  S  H
After alignment
P  =  S  L  Q  I  -  -  F  R  Y  N  S  H
Q  =  S  A  Q  L  G  I  -  -  F  T  S  H
```

Table 1 presents the calculated similarity matrix for these sequences. In general, the alignment is formed by following the track of the maximum scores, which starts from the right-bottom square to the left-top square in the similarity matrix. Here, the track starts from the cell containing 19 and continues with 11, 7, and so on.

In the alignment example, dashes represent the deleted or inserted residues in the evolutionary process and are named as *gaps*. Same or different letters in columns represent the *matches* or *substitutions*, respectively. Each of these 3 evolutionary processes has a score calculated using statistical methods. The maximum score in the alignment algorithm indicates the degree of similarity between sequences. Using this score, the databases are scanned for similar sequences to a query sequence.

The SW algorithm [7] is derived from the Needleman–Wunsch algorithm [22], which is a well-known global sequence alignment algorithm. Smith and Waterman modified this algorithm to perform local alignments. Since these 2 algorithms are very similar, with a little effort, hardware designed to perform local alignment operation can be modified to perform global sequence alignment operation. The main idea of the SW algorithm is to calculate a similarity score for each possible subalignment. The alignment with the maximum similarity score is then returned as the optimal alignment. The computational complexity of the algorithm prevents its usage in online applications. However, when it is necessary to obtain the best alignments, the SW algorithm must be used. The algorithm is explained as follows.

Let $Q = q_1q_2\ldots q_m$ and $P = p_1p_2\ldots p_n$ be the sequences to be aligned. $H(i,j)$ is the similarity score for the alignment of the subsequences $q_1q_2\ldots q_i$ and $p_1p_2\ldots p_j$. The SW algorithm calculates these $H(i,j)$ scores for every possible subalignment using the recursive equation set presented in Eq. (1), Eq. (2), and Eq. (3).

$$\begin{aligned} F(i,j) &= max\{H(i-1,j)-g_o, F(i-1,j)-g_e\} && (1)\\ E(i,j) &= max\{H(i,j-1)-g_o, E(i,j-1)-g_e\} && (2)\\ H(i,j) &= max\{0, H(i-1,j-1)+S(q_i,p_j), E(i,j), F(i,j)\} && (3)\\ & \quad for \quad 1 \le i \le m \quad and \quad 1 \le j \le n \end{aligned}$$

Initial values for these recursive equations are $H(i,0) = E(i,0) = F(i,0) = 0$ for $1 \le i \le m$ and $H(0,j) = E(0,j) = F(0,j) = 0$ for $1 \le j \le n$. g_o represents the gap opening penalty, and g_e represents the penalty for the extension of an already opened gap. Because of the nature of the biological sequences, the gap opening penalty is much greater than the gap extension penalty. If gap opening and gap extension penalties are different, then the alignment is called a *affine-gap-penalty* alignment. Affine-gap-penalty alignments are generally more meaningful for biological purposes. On the other hand, in some applications these penalties are assumed as equal to reduce the time and space requirements of the algorithm. This kind of alignment is called a *linear-gap-penalty alignment*, and Eq. (1), Eq. (2), and Eq. (3) can be simplified as follows.

$$\begin{aligned} H(i,j) &= max\{0, H(i-1,j-1)+S(q_i,p_j), H(i-1,j)-g_o, H(i,j-1)-g_o\} && (4)\\ & \quad for \quad 1 \le i \le m \quad and \quad 1 \le j \le n \end{aligned}$$

The $S(q_i,p_j)$ term in the given equations is the score for matches and substitutions, which is obtained from a substitution matrix. The substitution matrix is derived from biological databases using statistical methods. Two popular substitution matrices, the Dayhoff PAM250 [23] and the BLOSUM62 [24], are shown in Figure 1. The top halves of the matrices are not shown, since they are symmetric matrices.

2.2. Multiple sequence alignment and CLUSTALW algorithm

Multiple sequence alignment is a more complex operation than pairwise sequence alignment. This operation is used for the following purposes: a) determination of functional or structural relationships between sequences, b) building phylogenetic trees of related sequences, and c) identifying evolutionary conserved regions in sequences.

$$\begin{array}{c|cccccccccccccccccccc}
 & A & R & N & D & C & Q & E & G & H & I & L & K & M & F & P & S & T & W & Y & V \\
\hline
A & 2 \\
R & -2 & 6 \\
N & 0 & 0 & 2 \\
D & 0 & -1 & 2 & 4 \\
C & -2 & -4 & -4 & -5 & 12 \\
Q & 0 & 1 & 1 & 2 & -5 & 4 \\
E & 0 & -1 & 1 & 3 & -5 & 2 & 4 \\
G & 1 & -3 & 0 & 1 & -3 & -1 & 0 & 5 \\
H & -1 & 2 & 2 & 1 & -3 & 3 & 1 & -2 & 6 \\
I & -1 & -2 & -2 & -2 & -2 & -2 & -2 & -3 & -2 & 5 \\
L & -2 & -3 & -3 & -4 & -6 & -2 & -3 & -4 & -2 & 2 & 6 \\
K & -1 & 3 & 1 & 0 & -5 & 1 & 0 & -2 & 0 & -2 & -3 & 5 \\
M & -1 & 0 & -2 & -3 & -5 & -1 & -2 & -3 & -2 & 2 & 4 & 0 & 6 \\
F & -3 & -4 & -3 & -6 & -4 & -5 & -5 & -5 & -2 & 1 & 2 & -5 & 0 & 9 \\
P & 1 & 0 & 0 & -1 & 3 & 0 & 1 & 0 & 0 & 2 & 3 & 1 & 2 & 5 & 6 \\
S & 1 & 0 & 1 & 0 & 0 & -1 & 0 & 1 & -1 & -1 & -3 & 0 & -2 & -3 & 1 & 2 \\
T & 1 & 2 & 0 & 0 & -2 & -1 & 0 & 0 & -1 & 0 & -2 & 0 & -1 & -3 & 0 & 1 & 3 \\
W & -6 & 2 & -4 & -7 & -8 & -5 & -7 & -7 & -3 & -5 & -2 & -3 & -4 & 0 & -6 & -2 & -5 & 17 \\
Y & -3 & -4 & -2 & -4 & 0 & -4 & -4 & -5 & 0 & -1 & -1 & -4 & -2 & 7 & -5 & -3 & -3 & 0 & 10 \\
V & 0 & -2 & -2 & -2 & -2 & -2 & -2 & -1 & -2 & 4 & 2 & -2 & 2 & -1 & -1 & -1 & 0 & 6 & 2 & 4
\end{array}$$

(a) The Dayhoff PAM250 matrix

$$\begin{array}{c|cccccccccccccccccccc}
 & A & R & N & D & C & Q & E & G & H & I & L & K & M & F & P & S & T & W & Y & V \\
\hline
A & 4 \\
R & -1 & 5 \\
N & -2 & 0 & 6 \\
D & -2 & -2 & 1 & 6 \\
C & 0 & -3 & -3 & -3 & 9 \\
Q & -1 & 1 & 0 & 0 & -3 & 5 \\
E & -1 & 0 & 0 & 2 & -4 & 2 & 5 \\
G & 0 & -2 & 0 & -1 & -3 & -2 & -2 & 6 \\
H & -2 & 0 & 1 & -1 & -3 & 0 & 0 & -2 & 8 \\
I & -1 & -3 & -3 & -3 & -1 & -3 & -3 & -4 & -3 & 4 \\
L & -1 & -2 & -3 & -4 & -1 & -2 & -3 & -4 & -3 & 2 & 4 \\
K & -1 & 2 & 0 & -1 & -3 & 1 & 1 & -2 & -1 & -3 & -2 & 5 \\
M & -1 & -1 & -2 & -3 & -1 & 0 & -2 & -3 & -2 & 1 & 2 & -1 & 5 \\
F & -2 & -3 & -3 & -3 & -2 & -3 & -3 & -3 & -1 & 0 & 0 & -3 & 0 & 6 \\
P & -1 & -2 & -2 & -1 & -3 & -1 & -1 & -2 & -2 & -3 & -3 & -1 & -2 & -4 & 7 \\
S & 1 & -1 & 1 & 0 & -1 & 0 & 0 & 0 & -1 & -2 & -2 & 0 & -1 & -2 & -1 & 4 \\
T & 0 & -1 & 0 & -1 & -1 & -1 & -1 & -2 & -2 & -1 & -1 & -1 & -1 & -2 & -1 & 1 & 5 \\
W & -3 & -3 & -4 & -4 & -2 & -2 & -3 & -2 & -2 & -3 & -2 & -3 & -1 & 1 & -4 & -3 & -2 & 11 \\
Y & -2 & -2 & -2 & -3 & -2 & -1 & -2 & -3 & 2 & -1 & -1 & -2 & -1 & 3 & -3 & -2 & -2 & 2 & 7 \\
V & 0 & -3 & -3 & -3 & -1 & -2 & -2 & -3 & -3 & 3 & 1 & -2 & 1 & -1 & -2 & -2 & 0 & -3 & -1 & 4
\end{array}$$

(b) The BLOSUM62 matrix

Figure 1. Two popular substitution score matrices.

An example of a multiple sequence alignment among 4 DNA sequences is given below.

A	A	T	C	G	-	-	A	T	G	T
A	C	T	C	G	T	-	T	A	G	T
A	C	T	C	A	A	T	T	G	G	T
T	A	T	C	G	-	-	-	T	G	T

Dynamic programming algorithms are not feasible for performing optimum multiple sequence alignment since they have $O(L^N)$ complexity, where N is the number of sequences in multiple alignment and L is the average length of these sequences. Progressive algorithms have been developed to overcome this NP-complete problem [25]. Two well-known progressive alignment algorithms/software are the CLUSTALW [10] and the T-COFFEE [11]. In this work, CLUSTALW software performance is increased by hardware support. The CLUSTALW software was developed by Higgins et al. 20 years ago [26] and is still the most popular software solution for multiple sequence alignment. CLUSTALW [10] has the following steps.

1. **Distance matrix computation:** In the first step, the algorithm performs pairwise alignment between every pair of sequences. After the preliminary pairwise alignment of sequences, the exact matches of the corresponding positions are counted. The computational complexity of this stage is $O(N^2L^2)$. For example, the number of matches in the alignment given below is 6. Exact matches are shown with bold font.

A	A	**T**	**C**	**G**	-	-	A	T	**G**	**T**
A	C	**T**	**C**	**G**	T	-	T	A	**G**	**T**

The following formula is used to calculate distance scores between the pair of sequences named as P and Q.

$$D(P,Q) = 1 - \frac{N(P,Q)}{min(P_l,Q_l)} \tag{5}$$

In Eq. (5), $N(P,Q)$ denotes the number of exact matches, and $min(P_l,Q_l)$ denotes the minimum length of 2 sequences. The distance score $D(P,Q)$ for the given example is $3/9$.

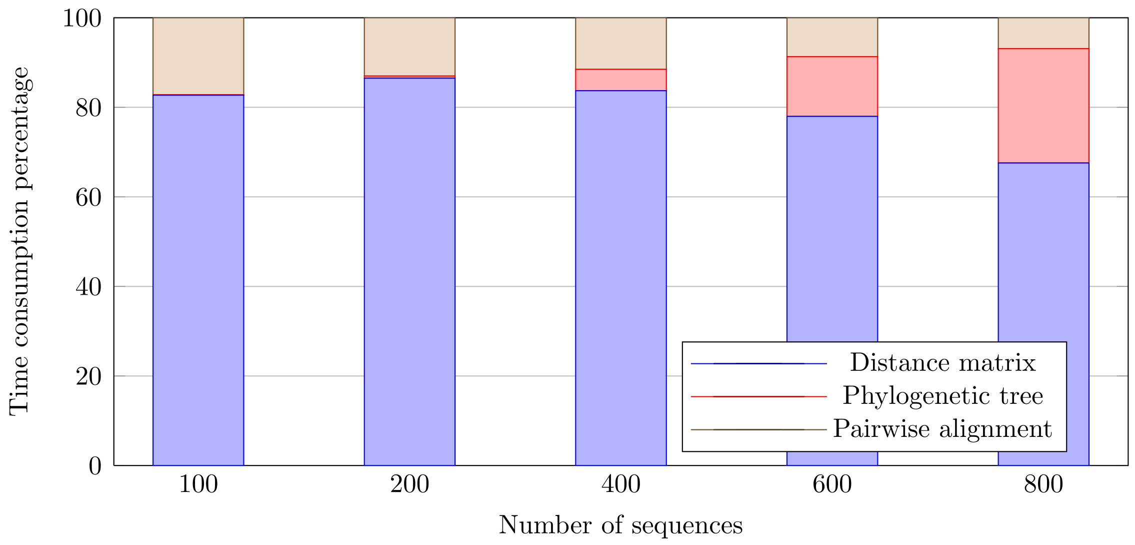

Figure 2. The execution time percentages of CLUSTALW steps.

2. **Phylogenetic tree construction:** In the second step, a phylogenetic tree is built based on the distance scores. In this way, distantly related sequences are separated from each other. This stage has $O(N^3)$ complexity.

3. **Pairwise alignment :** In the final step, related pairs of sequences are aligned based on their positions in the tree to finalize the multiple alignment. The complexity of this stage is $O(NL^2)$.

In [16] and [17], it was reported that 90% of the overall execution time is spent in the first step. To verify this claim, the CLUSTALW algorithm is executed on a computer that has an AMD Athlon X2 64-bit processor with 1 GB RAM and Windows XP operating system. Figure 2 shows the percentage execution time of each step for 100 to 800 sequences. It is also verified by our tests that the majority of the overall execution time is spent in the first step. Thus, our optimization effort is focused on this step.

3. Proposed designs

This section presents the proposed hardware designs for pairwise sequence alignment and multiple sequence alignment operations. In the next subsections, first the general organizations of the designs are described, and then the implementation details are explained.

3.1. Pairwise sequence alignment system

The general organization of the pairwise alignment system is presented in Figure 3. The proposed system consists of 2 main units. The first unit is the host PC, which stores the biological databases and executes the interface program. The second unit is the FPGA card, which is connected to the host PC via the peripheral component interconnect (PCI) data bus. The systolic architecture, which calculates necessary scores for pairwise alignment, resides on the FPGA. Since the design implements the SW algorithm, the cells of the systolic architecture are labeled as SW cells. The properties and data flow of the system are explained as follows:

1. The user inputs the query sequence via interface software. The interface software encodes the biological database sequences to an appropriate processing format.

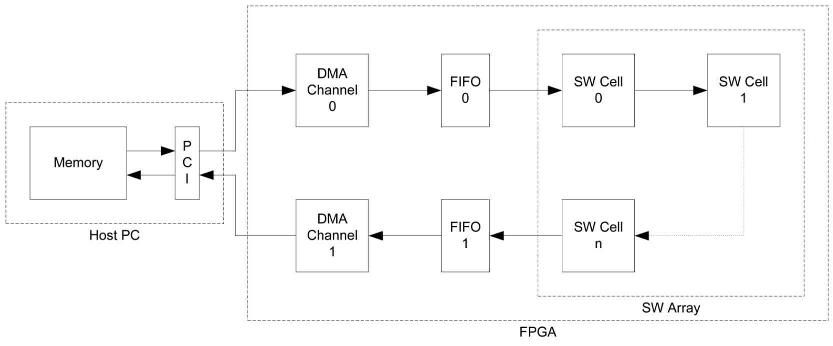

Figure 3. The general organization of the pairwise alignment system.

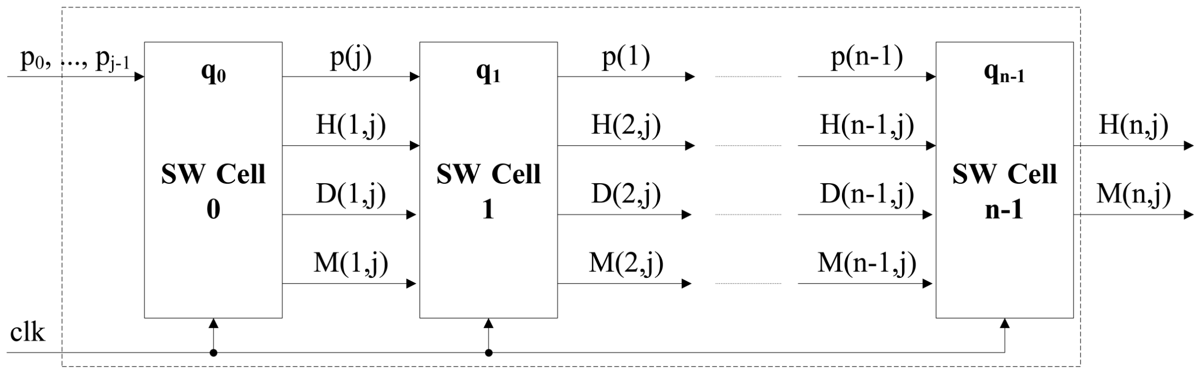

Figure 4. The details of the systolic architecture [19].

2. The encoded data stored in the host PC's memory are transferred to FIFO 0 via the PCI bus using direct memory access (DMA) protocol.

3. Each residue of the query sequence in FIFO 0 is placed into one SW cell of the systolic architecture. These residues remain in SW cells until the entire database is scanned.

4. The systolic architecture calculates similarity scores and stores them in FIFO 1.

5. When FIFO 1 is about to become full, similarity scores are transferred to the PC via the PCI bus.

6. The sequences with maximum similarity scores are aligned with the query sequence and displayed to the user by the software.

The connections between neighboring SW cells are shown in Figure 4. All SW cells are identical and cascaded as a single-dimensional array. Here, p_j denotes the residues of the sequences from the database, and q_i denotes the residues of the query sequence. $H(i,j)$ and $M(i,j)$ represent the similarity scores and the current maximum scores, respectively. $D(i,j)$ denotes the intermediate values used by the proposed method. p_j values pass through all SW cells, and their most significant 3 bits are used as flag bits to control the systolic architecture. Therefore, all control signals are transferred to the next cell in every clock cycle. In this way, installation of long control lines is prevented and an efficient way of controlling the signal distribution is achieved. The cell design is the most important part of the systolic architecture, since the size and speed of the cells determine the overall system performance. Adders, registers, and maximum score generators in the SW cells implement the calculations given in Eq. (1), Eq. (2), and Eq. (3). However, direct mapping of these equations to the hardware results in inefficiencies, as explained below.

1. Precision of $H(i,j)$ values determines the size of adders, registers, and maximum generators in SW cells. These units are the main factors that determine the cell area and clock frequency. The higher the precision of the operands entering these units, the greater the hardware requirement of the system and the lower the performance.

2. Direct implementation of Eq. (1), Eq. (2), and Eq. (3) causes significant use of resources. Optimization of these equations for hardware implementation provides higher performance with less hardware.

The drawbacks stated above can be overcome by methods presented in [18] and [19]. Examining the equations reveals that the precision of similarity scores is always greater than the precision of gap costs. For example, $S(q_i,p_j)$ substitution values, gap opening and extension costs (g_o and g_e), can be represented using 4-bit numbers, while the precision of $H(i,j)$ similarity scores is at least 16 bits. Therefore, the *H(i,j)* value differs from $H(i,j-1)$, $H(i-1,j)$, and $H(i-1,j-1)$ values by a small amount. In fact, this small amount can be at most equal to the difference between the gap opening cost and the maximum value in the substitution matrix. Assuming that the precision of $H(i,j)$ is 16 bits and the precisions of gap penalties and substitution values are 5 bits (as in BLOSUM62), then the difference values can be calculated using the following equations.

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

$$D(i-1,j) = H(i-1,j)_{15:4} - H(i-1,j-1)_{15:4} \tag{7}$$

$$D(i,j) = H(i,j)_{15:4} - H(i-1,j)_{15:4} \tag{8}$$

Here, subscript 15:4 means that the most significant 12 bits of value are used. $D(i-1,j-1)$ is defined as the value of the carry or borrow out of the addition $H(i-1,j-1)_{3:0}+S(q_i,p_j)$. All of these difference values can be calculated using 2-bit subtractors, and they can be only -1, 0, +1. The difference values are concatenated to the similarity scores as $D(x,y)\&H(x,y)_{3:0}$. The new 6-bit representation reduces the size of the adders, registers, and maximum generators. Moreover, the sizes of these units remain the same even if the score precision increases. Similarly, the calculation of $E(i,j)$ and $F(i,j)$ values can be simplified. According to Eq. (1), Eq. (2), and Eq. (3), $H(i,j-1)$ is always greater than $E(i,j-1)$. Therefore, $Z = H(i,j-1) - E(i,j-1) \geq 0$ is always true. This inequality can be used for the following analysis:

1. If $Z = 0$, then $H(i,j-1) - g_o < E(i,j-1) - g_e$, and,
   $E(i,j) = H(i,j-1) - g_e = E(i,j-1) - g_e$ $(g_o > g_e)$.

2. If $Z > 0$, there are 2 cases to be considered:

   (a) If $Z \geq g_o - g_e$, then $E(i,j) = H(i,j-1) - g_o$.

   (b) If $Z < g_o - g_e$, then $E(i,j) = E(i,j-1) - g_e$.

If $(g_o - g_e = 1)$, then the case in 2(b) never occurs, and only $H(i,j-1)$ is used to generate $E(i,j)$. On the other hand, if $(g_o - g_e > 1)$, 2(b) can occur; because of that, $Z < g_o - g_e$ is tested and the result of the test and $E(i,j-1)$ are stored. A similar analysis can be made for the generation of $F(i,j)$.

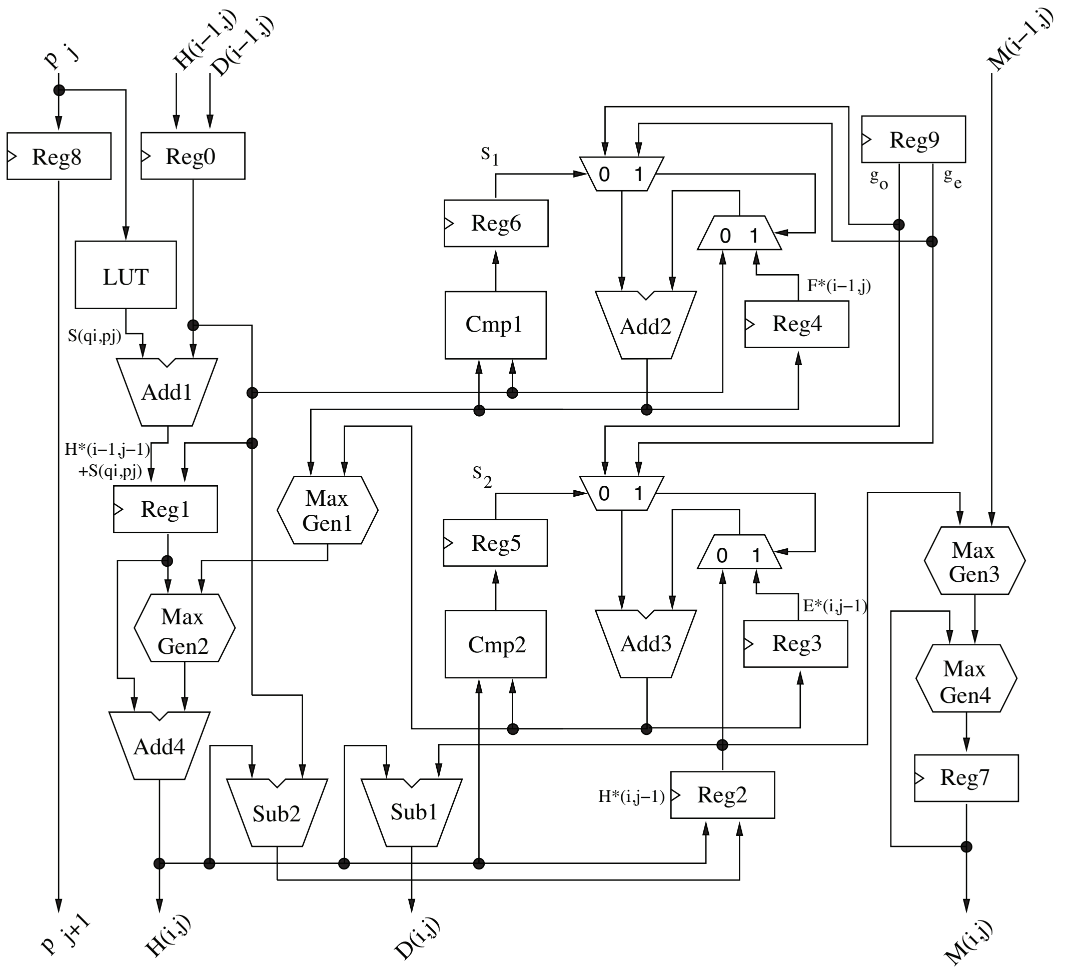

Figure 5. Pairwise alignment cell (SW-Cell) [19].

The optimized cell shown in Figure 5 is designed by using the discussed optimizations. Here, the cell represents the i^{th} cell in the linear systolic architecture. Therefore, inputs of this cell are the outputs of the $(i-1)^{th}$ cell. The input signals carry encoded residues p_j, intermediate similarity score $H(i-1,j)$, current maximum score $M(i-1,j)$, and difference value $D(i-1,j)$. The outputs of the cell are passed to the next cell in the systolic array. Functions of the units in the cell are explained as follows.

- *Add1*, *Add2*, and *Add3* are 4-bit adders. *Add1* adds the amino acid substitution score, $S(q_i,p_j)$ (obtained from the LUT), with $H(i,j-1)_{3:0}$, and generates $H^*(i-1,j-1) = D(i-1,j-1)$ & $H(i-1,j-1)_{3:0}$, where $D(i-1,j-1)$ is the carry or borrow out from the adder, and $H(i-1,j-1)_{3:0}$ is the 4-bit sum generated by the adder. *Add2* adds g_o or g_e with $H^*(i-1,j)$ or $F^*(i-1,j)$ based on the value of the control signal $S1$, and generates $F^*(i,j)$. *Add3* adds g_o or g_e to $H^*(i,j-1)$ or $E^*(i,j-1)$ based on the value of control signal $S2$, and generates $E^*(i,j)$.

- *MaxGen1* and *MaxGen2* are 6-bit maximum generation units. *MaxGen1* outputs the maximum of $F^*(i-1,j)$ and $E^*(i,j-1)$. *MaxGen2* compares the output of *MaxGen1* with $H^*(i-1,j-1)$ and outputs the maximum of the 2.

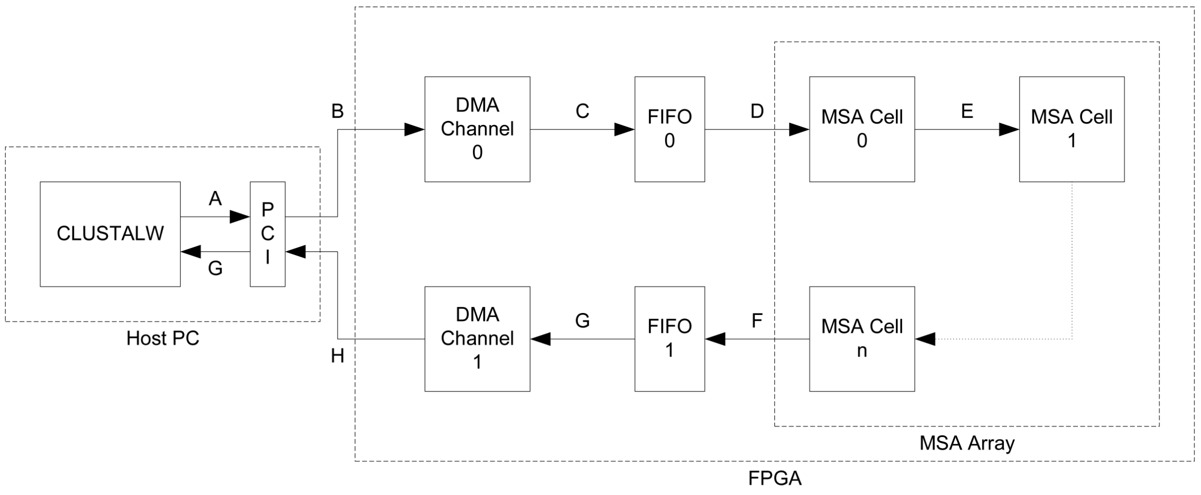

Figure 6. Block diagram of multiple sequence alignment system [20].

- *Cmp1* and *Cmp2* units generate control signals $S1$ and $S2$, respectively. Each of these units consists of a 6-bit subtractor and a 4-bit comparator. In *Cmp1* the inequality $(H^*(i-1,j) - F^*(i-1,j)) < (g_o - g_e)$ is tested, and if it is true, $S1$ is set. In *Cmp2*, the inequality $(H^*(i,j-1) - E^*(i,j-1)) < (g_o - g_e)$ is tested, and if it is true, $S2$ is set.

- *Sub1* and *Sub2* are 2-bit subtractors that compute $D(i,j)$ and $D(i,j-1)$, respectively. $D(i,j)$ is sent to the next cell. $D(i,j-1)$ is used in the generation of $H(i,j-1)$.

- *MaxGen3* and *MaxGen4* are 16-bit maximum generation units, and these units compare 3 values: the maximum values from previous cell $M(i-1,j)$, the maximum value generated in present cell $M(i,j-1)$, and the score generated in previous iteration $H(i,j-1)$. The unit outputs the maximum of the 3 as $M(i,j)$.

- Register *Reg0* stores both $H(i,j-1)$ and $D(i,j-1)$ values. Registers *Reg1* through *Reg8* store the values $H(i-1,j-1)_{15:4} + S(q_i,p_j)$, $H^*(i,j-1)$, $E^*(i,j-1)$, $F^*(i-1,j)$, $S1$, $S2$, $Max(i,j)$, and p_{j+1}, respectively. Register *Reg9* stores both g_o and g_e values.

3.2. Multiple sequence alignment system

The general organization of the proposed multiple sequence alignment system (MSAS) is presented in Figure 6. Similar to the pairwise alignment system, the MSAS has 2 main units: a host PC and an FPGA card. In the MSAS, the host PC executes the CLUSTALW software with an integrated interface program. The systolic architecture placed on the FPGA counts the exact matches of residues. The numbers of the exact matches are used in the first step of the CLUSTALW algorithm to calculate the distance scores among the aligned sequences. Open source code of CLUSTALW (publicly available at [27]) is used. The data flow is explained as follows:

1. CLUSTALW reads the sequences to be aligned from a file stored in the host PC.

2. The sequences are encoded and stored in the host PC's main memory.

3. The interface program transfers these data to FIFO 0 with a DMA transfer. This operation is managed by the control circuit DMA Channel 0.

4. The systolic architecture counts the number of exact matches between the first sequence in FIFO 0 and the rest of the sequences.

5. Counts are stored in FIFO 1 without losing the order.

6. The data are transferred to the host PC's memory when FIFO 1 is about to become full. This operation is controlled by the interface program on the host PC and the DMA Channel 1 on FPGA.

7. The previous steps are repeated until every pair of sequences is processed.

8. CLUSTALW software performs the second and third steps of the algorithm using the numbers of exact matches computed by the hardware.

In Figure 6, interconnection buses are labeled with letters *A–I*. The values transferred by these signals are explained as follows: *A* carries encoded sequences and control signals. The control signals include system reset and synchronization flags. *B* is a 32-bit connection that transfers data and control to DMA Channel 0. *C* connection transfers the packed sequences to FIFO 0 when the PCI is ready. *D* carries the sequences to the systolic architecture as 8-bit packages. To reduce the bit size from 32 bits to 8 bits, an asymmetric FIFO is used. *E* transfers sequences and control signals from one MSA cell to the next. *F* passes the calculated values to FIFO 1. *G* transfers values in FIFO 1 to the PCI bus when it is available. *H* provides the data flow between DMA Channel 1 and the PCI bus. *I* connects the PCI bus and the host PC.

The interface program that manages the data flow has 2 threads to prevent conflicts on the PCI bus. One of the threads sends the sequences and control signals to the FPGA, while the other gets the calculated values from the FPGA. Communication between these threads is implemented with mutual exclusion using semaphores. The interface program works in compliance with the CLUSTALW software. As stated in the previous section, the design of the cell is the most crucial step for a systolic architecture system. The structure of the MSA cell is organized as presented in [20] and it uses similar optimizations as the SW cell (designed for the pairwise alignment system). However, there are 2 main differences between the SW cell and the MSA cell. The first difference is that in the MSA cell, the sequences stored in each cell are changed during the process, which requires more complex control logic. The second difference is that the MSA cell counts exact matches in the alignment. This method was also used by the previous implementations in [16] and [17]. However, previous implementations dedicated special logic for this task. On the other hand, the proposed hardware shares most of the logic with similarity score computations. Exact matches can be calculated during pairwise operation using the following equations, as presented in [17].

$$N(i,j) = \begin{cases} 0 & \text{if } H(i,j)=0 \\ N(i-1,j-1)+C(i,j) & \text{if } H(i,j)=H(i-1,j-1)+S(q_i,p_j) \\ N(i-1,j) & \text{if } H(i,j)=H(i-1,j)-g_o \\ N(i,j-1) & \text{if } H(i,j)=H(i,j-1)-g_o \end{cases} \tag{9}$$

$$C(i,j) = \begin{cases} 1 & \text{if } q_i = p_j \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$for \quad 1 \le i \le m \quad and \quad 1 \le j \le n$$

The design of the proposed multiple sequence alignment cell is illustrated in Figure 7. The functions of the main units and the data flow in the MSA cell are explained as follows.

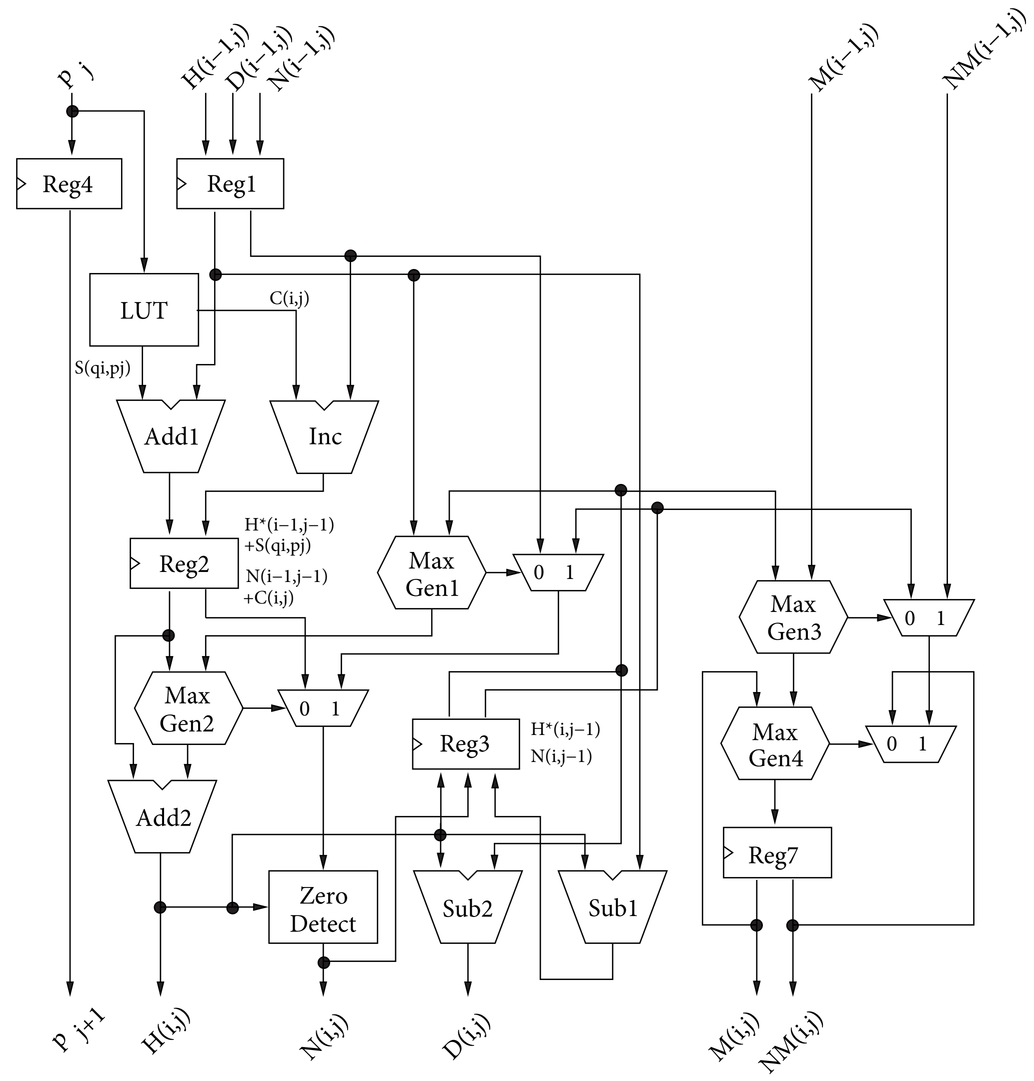

Figure 7. Multiple sequence alignment cell (MSA-Cell) [20].

- *Add1* adds the substitution score from LUT, $S(q_i, p_j)$, with $H(i, j-1)_{3:0}$, and generates $H^*(i-1, j-1) = D(i-1, j-1) \ \& \ H(i-1, j-1)_{3:0}$, where $D(i-1, j-1)$ is the carry or borrow out from the adder, and $H(i-1, j-1)_{3:0}$ is the 4-bit sum generated by the adder. *Add2* adds the calculated carry or borrow to the 12-bit output of *MaxGen2*.

- *MaxGen1* and *MaxGen2* are 6-bit maximum generators. *MaxGen1* calculates the maximum of $H^*(i-1, j)$ and $H^*(i, j-1)$, and passes either $N(i-1, j)$ or $N(i, j-1)$ value according to the result of the maximum calculation to implement the *if* statements in the equation. Similarly, *MaxGen2* calculates the maximum of the output of *MaxGen1* and $H^*(i-1, j-1)$, and passes either N output of *MaxGen1* or $N(i-1, j-1)$. *MaxGen3* and *MaxGen4* calculate the current maximum value of similarity scores and pass N value based on the maximum value.

- *Sub1* and *Sub2* are 2-bit subtractors that calculate $D(i, j)$ and $D(i, j-1)$. $D(i, j)$ is transferred to the next cell and $D(i, j-1)$ is used to calculate similarity scores in the next clock cycle.

Table 2. Synthesis results of reference and proposed designs [19].

		Linear gap penalty			Affine gap penalty		
	Precision	16-bit	24-bit	32-bit	16-bit	24-bit	32-bit
# of flip flops	Ref.	62	94	126	93	141	189
per cell	Prop.	68	84	108	76	86	124
# of LUTs	Ref.	149	265	353	314	472	632
per cell	Prop.	130	182	223	164	193	260
# of	Ref.	450	268	201	235	171	129
cells	Prop.	482	388	307	412	329	264
	Gain	7.1%	44.8%	52.7%	75.3%	92.4%	104.7%
Frequency	Ref.	91.87	78.40	72.15	68.75	68.45	62.84
(MHz)	Prop.	112.81	110.68	110.39	114.35	114.19	112.60
	Gain	22.8%	41.2%	53.0%	66.3%	66.8%	79.2%
Performance	Ref.	41.34	21.01	14.50	16.15	11.70	8.10
(GCUPS)	Prop.	54.37	42.94	33.89	47.11	37.56	29.72
	Gain	31.5%	104.4%	133.7%	191.7%	221.0%	266.9%

- *Reg1* holds $N(i-1,j)$, $H(i-1,j)$, and $D(i-1,j)$. *Reg2* stores $N(i-1,j-1)+C(i,j)$, $H(i-1,j-1)+S(q_i,p_j)$, and $D(i-1,j-1)$ values. *Reg3* holds $N(i,j-1)$, $H(i,j-1)$, and $D(i,j-1)$ values. *Reg4* stores the residues of sequences. *Reg5* keeps the current maximum value of similarity scores and exact match counts.

4. Results

Proposed pairwise and multiple sequence alignment designs with linear and affine gap penalties are implemented using Xilinx Virtex II XC2V6000 FPGA [13] on an Alpha-Data ADM-XRC-II PCI Mezzanine card [28]. The card is connected to a host PC with an AMD Athlon X2 64-bit processor and 1 GB 666 MHz memory. Interface programs are developed with C++. Functionality of the designs are verified by performing a variety of alignment operations. The following subsections evaluate the performance of the proposed designs by comparing them with the previous designs. Execution times and comparison results are given in this section for pairwise and multiple sequence alignment, respectively.

4.1. Evaluation of the pairwise sequence alignment systems

The proposed design is compared with the previous work published by Oliver et al. [15], since they have the same functionality. All of the designs are modeled using Very-Large-Scale Integrated Circuit Hardware Description Language (VHDL) to provide the same conditions as the proposed designs. Table 2 presents synthesis results for linear and affine gap penalty cells, respectively. An important performance parameter for systolic architectures is the count of cell updates per second (giga cell updates per second, GCUPS). Table 2 shows that the GCUPS value of each proposed design is higher than the corresponding reference design. Table 2 also shows the maximum number of cells that can be fitted to the aforementioned FPGA platform based on the precision. Using the 16-bit cell values given in Table 2, it can be concluded that a sequence with 4120 residues can be aligned in 10 iterations by the proposed design, while the corresponding reference design needs

Table 3. Overall performance of pairwise alignment systems.

Database	Linear gap penalty		Affine gap penalty	
size (MB)	Prop.Sys. (s)	Software (s)	Prop.Sys. (s)	Software (s)
2	1.234	77.81	1.328	84.52
10	3.125	394.78	3.218	429.34
20	5.531	779.89	5.610	844.55
40	10.297	1597.78	10.344	1746.30
80	19.765	3143.48	19.922	3583.72
100	24.578	3939.83	24.625	4324.42

Table 4. Comparison of CLUSTALW, reference, and proposed MSA systems [20].

Number of sequences	Proposed (s)	Reference (s)	Speed-up	CLUSTALW (s)	Speed-up
100	0.859	1.1	1.28	19.828	23.08
200	1.188	3.6	3.03	60.672	51.07
400	2.500	18.1	7.24	197.500	79.00
600	4.688	38.2	8.15	402.625	85.88
800	7.969	65.0	8.16	664.391	83.87

18 iterations. Note that Table 2 contains only hardware-related results. In general, the cited previous work does not include any information about performance results of the systems for practical usage. However, the end user is more interested in the wall-clock time required to get the alignment results than the hardware execution times. Therefore, wall-clock time required for alignments is also included in this work. Table 3 presents overall performances for linear and affine gap penalty alignment systems. Here, the first column is the database size to be scanned while the other columns present the performance of the proposed system and the software-only solution. As seen from Table 3, the proposed system performance increases as the database gets larger. For example, the proposed system scans a 100 MB database in less than 30 s, while the software-only application uses over 90 min to do the same operation.

4.2. Performance evaluations of the multiple sequence alignment systems

Proposed multiple sequence alignment designs are modeled in VHDL and mapped on a Xilinx Virtex II XC2V6000 FPGA platform. There are 185 MSA cells placed on the FPGA. Each cell includes 91 flip flops and 354 LUTs mapped on 188 FPGA slices. According to the synthesis results, the clock frequency can be set to 57 MHz at maximum. Therefore, performance of the multiple sequence alignment system is 10.54 GCUPS. Table 4 presents how much the first step of the CLUSTALW algorithm is accelerated. To achieve this information, 100 to 800 sequences with 185 residues were aligned with FPGA-supported CLUSTALW and pure CLUSTALW. As is seen, the first step is accelerated over 80 times as the number of sequences surpasses 600. However, this acceleration is only for the first step of CLUSTALW, and the overall gain is still obscure. A realistic prediction can be made by using Amdahl's Law [29]. According to Figure 2, about 80% of overall time is spent on the first step. Because of that, the hardware can achieve a maximum of 5 times of acceleration over the software-only solution. For this reason, the first steps are compared in Table 4, as it was made in the previous work. The proposed design is compared with a similar design that aimed to accelerate CLUSTALW, presented in [17]. Table 4 presents results directly obtained from [17], since they used exactly the same FPGA platform. Table 4 also compares the speed of the aforementioned reference design and proposed design. In [17], the authors did not specify the names or the sizes of the globin sequences used to test their system. On the other hand, there exists approximately 8000 different globin sequences on popular databases. A rough examination of these sequences show that these sequences are 140 to 150 letters long in general. Our tests are performed on protein sequences 185 letters long obtained from the PIR database [30].

5. Conclusion

In this work, system designs to perform pairwise and multiple sequence alignment operations are presented. The proposed systems include a host PC and an FPGA Mezzanine card. Through optimizations of the designs, smaller hardware usage and higher execution speed are achieved. The proposed pairwise alignment system can extract most similar sequences in a biological database to a query sequence. Performance of this system is up to 3.5 times higher than those of reference designs. The system scans a 100 MB database in less than 30 s, while the software-only solution performs the same operation in over 90 min. The proposed MSAS accelerates the slowest step of the CLUSTALW algorithm. The MSAS design performs the first step over 8 times faster than the reference designs for 16-bit precision. In general, if the length of sequences exceeds the number of cells, then additional iterations are needed to complete the alignment. On the other hand, the current state of the art overcomes this problem, since the capacity of the FPGAs has improved to fit sequences with several thousand residues. To justify that, systems are also synthesized for the Xilinx Virtex-6 XC6VLX760 platform. Results indicate that the maximum number of pairwise alignment cells can be 3764 with 225 MHz clock frequency, and the maximum number of multiple alignment cells can be 1928 with 218 MHz clock frequency for 16-bit precision. Therefore, additional iterations are not necessary considering the length of protein sequences in databases. The performance with this state-of-the-art FPGA is expected to be about 100 GCUPS.

Acknowledgments

This work was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under Grant No: 105E075.

References

[1] National Institutes of Health, Working Definition of Bioinformatics and Computational Biology, Bethesda, NIH, http://www.bisti.nih.gov/CompuBioDef.pdf, 2000.

[2] D.W. Mount, Bioinformatics: Sequence and Genome Analysis, Cold Spring Harbor, Cold Spring Harbor Laboratory Press, 2004.

[3] National Center for Biotechnology Information, Homepage, http://www.ncbi.nlm.nih.gov/.

[4] National Center for Biotechnology Information, GenBank Overview, http://www.ncbi.nlm.nih.gov/genbank/.

[5] W.R. Pearson, D.J. Lipman, "Improved tools for biological sequence comparison", Proceedings of the National Academy of Sciences, Vol. 85, pp. 2444–2448, 1988.

[6] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman, "Basic local alignment search tool", Journal of Molecular Biology, Vol. 215, pp. 403–410, 1990.

[7] T.F. Smith, M.S. Waterman, "Identification of common molecular subsequences", Journal of Molecular Biology, Vol. 147, pp. 195–197, 1981.

[8] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Cambridge, Cambridge University Press, 2002.

[9] S.R. Eddy, "Profile hidden Markov models", Bioinformatics, Vol. 14, pp. 755–763, 1998.

[10] J.D. Thompson, D.G. Higgins, T.J. Gibson, "ClustalW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice", Nucleic Acids Research, Vol. 22, pp. 4673–4680, 1994.

[11] C. Notredame, D.G. Higgins, J. Heringa, "T-Coffee: a novel method for fast and accurate multiple sequence alignment", Journal of Molecular Biology, Vol. 302, pp. 205–217, 2000.

[12] D.T. Hoang, "Searching genetic databases on Splash 2", Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, pp. 185–191, 1993.

[13] Xilinx home page, http://www.xilinx.com.

[14] Y. Yamaguchi, T. Maruyama, A. Konagaya, "High speed homology search with FPGAs", Pacific Symposium on Biocomputing, pp. 271–282, 2002.

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

[16] X. Lin, Z. Peiheng, B. Dongbo, F. Shengzhong, S. Ninghui, "To accelerate multiple sequence alignment using FPGAs", Eighth International Conference on High-Performance Computing in Asia-Pacific Region, pp. 5–180, 2005.

[17] T.F. Oliver, B. Schmidt, D. Nathan, R. Clemens, D.L. Maskell, "Multiple sequence alignment on an FPGA", Proceedings of the 11th International Conference on Parallel and Distributed Systems, Vol. 2, pp. 326–330, 2005.

[18] M. Gök, Ç. Yılmaz, "Efficient cell designs for systolic Smith–Waterman implementations", Proceedings of the International Conference on Field Programmable Logic and Applications, pp. 889–892, 2006.

[19] M. Gök, Ç. Yılmaz, "Hardware designs for local alignment of protein sequences", Lecture Notes in Computer Science, Vol. 4263M pp. 277–285, 2006.

[20] Ç. Yılmaz, M. Gök, "An optimized system for multiple sequence alignment", Proceedings of the International Conference on Reconfigurable Computing and FPGAs, pp. 178–182, 2009.

[21] National Center for Biotechnology Information, GenBank Sequences from Pandemic (H1N1) 2009 Viruses, http://www.ncbi.nlm.gov/genomes/FLU/SwineFlu.html.

[22] S.B. Needleman, 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, pp. 443–453, 1970.

[23] M.O. Dayhoff, R.M. Schwartz, B.C. Orcutt, "A model of evolutionary change in proteins", Atlas of Protein Sequence and Structure, Vol. 5, pp. 345–352, 1978.

[24] S. Henikoff, J.G. Hanikoff, "Amino acid substitution matrices from protein blocks", Proceedings of the National Academy of Sciences, Vol. 89, pp. 10915–10919, 1992.

[25] İ.Ö. Bucak, V. Uslan, "Sequence alignment from the perspective of stochastic optimization: a survey", Turkish Journal of Electrical Engineering & Computer Sciences, Vol. 19, pp. 157–173, 2011.

[26] D.G. Higgins, P.M. Sharp, "Clustal: a package for performing multiple sequence alignment on a microcomputer", Gene, Vol. 73, pp. 237–244, 1988.

[27] ClustalW: Multiple Sequence Alignment Home Page, http://www.clustal.org.

[28] Alpha Data Home Page, http://www.alpha-data.com.

[29] G.M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities", Proceedings of the Spring Joint Computer Conference, pp. 483–485, 1967.

[30] PIR - Protein Information Resource Homepage, http://pir.georgetown.edu.