The CSI Journal on
Computer Science and Engineering
Vol. 13, No. 1, 2015
Pages 38-50
Regular Paper

A Fast Hardware Implementation of STON Algorithm for Comparing 3-D Structure of Proteins in FPGA

Somayeh Kashi Morteza Saheb Zamani

Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran

Abstract

Protein is an important molecule that performs a wide scope of functions in biological systems. Protein structures change little during evolution and structural comparison provides more accurate information about the evolution history. Therefore, comparing three-dimensional structure of proteins is one of the most fundamental problems in bioinformatics. In recent years, various algorithms have been proposed to solve this problem efficiently. The proposed algorithms are very time-consuming due to high complexity of these structures. In this paper, an FPGA-based hardware accelerator is proposed for the first time to speed-up the execution of a high-performance algorithm, called STON. Two classes of approaches are used to reduce the execution time of the algorithm, namely, coarse-grain and fine-grain. In the coarse-grain approach, due to the independency of for-loop cycles, parallelism is used for the iterations of these loops. In the fine-grain approach, the square root, division and trigonometry functions are performed in a fast mode. A software/hardware co-design of the algorithm was implemented based on two different Xilinx FPGA devices from Vertex family. An average speed-up of 10x is achieved compared to the software execution.

Keywords: Bioinformatics, Protein Structure Comparison, STON Algorithm, Hardware Acceleration, FPGA.

1. Introduction

Since the function of every living organism can be determined by the physical structure of its protein, research on proteins has attracted researchers in systems biology. Proteins have three physical structures, called protein sequence, protein second structure and protein three-dimensional structure. The first structure is formed from amino acids; proteins are synthesized as linear chains of amino acids. Then they form secondary structures along the chain, such as α helices, β sheets and loops. The condensed multiple secondary structural elements lead to tertiary structure. The tertiary structure is stabilized by efficient packing of atoms in the protein interior (Figure 1). Physical, chemical and biological characteristics of proteins are extracted from the differences and similarities of the linked three-dimensional structures of the protein. Protein structures change less than protein sequences during evolution [14]. Therefore, structural comparison provides more accurate information about the evolution history than sequence alignment.

About 123273 protein structures are available from protein data bank on Oct 2016, and the number of structures is growing rapidly (http://www.rcsb.org). A newly determined structure of a protein is compared with the previously classified structures to determine the family, evolutionary relationship and function of the protein.

Protein structure comparison is an NP-hard problem [2]. Two categories of various heuristic algorithms have been developed for this problem during the last decades, namely, intermolecular and intramolecular. In the first approach, one protein structure rotates in 3-D space, and therefore, its related parts are laid on another protein structure and their intermolecular distances are minimized [3-7]. In the second approach, the interior distances between atoms within each

protein are computed and these distances in the two proteins are compared [8-13]. Although many methods have been developed, different results have been reported for the same benchmarks. One of the basic criteria for choosing a proper algorithm is Root Mean Square Deviation (RMSD). The algorithm which produces a small RMSD for a large protein is preferred.

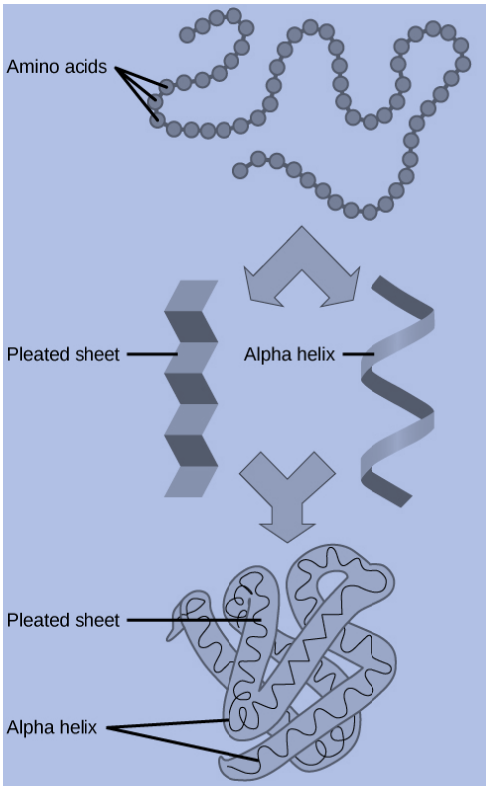

Figure 1. Protein structures (https://figures.boundless-cdn.com, Retrieved on 10/6/2016)

STON is one of the high-performance and parameterized algorithms [14] which has the ability of finding weakly- and strongly- related proteins and near or remote structural homology can be detected. This algorithm generates a two-dimensional zero/one matrix based on the result of comparing the angles of the two proteins and then finds the largest locally similar parts of the proteins. Then it computes the distance matrix and rotates the smaller protein (the protein with smaller radius) around the other one in several steps. The RMSD and number of equivalent residues are computed and then the data with large size and small RMSD is selected as a desired result. Due to the large database and high complexity of the STON algorithm, its long run-time is one of the most major problems in using the algorithm in practice. In fact, the run-time of the algorithm grows significantly with the size of the input proteins and user-defined parameters. There are two challenges in solving this problem. First, achieving practical speed and second, preserving the required precision. To ameliorate these problems, hardware acceleration solution is considered to implement the whole algorithm or some parts of it on a suitable hardware platform [15].

In this paper, an FPGA-based implementation of the STON algorithm is presented for comparing the three-dimensional structures of proteins while significantly preserving the accuracy of the algorithm. Overally, the key attributes and contributions of this hardware acceleration method are as follows:

- We described the STON algorithm in verilog language and then synthesized, placed and routed it on two different Xilinx FPGA devices from Vertex family, namely, Virtex5:xc5vsx240t and Virtex7:xc7vx1140. These devices are rich in memory and DSP block resources compared to the other members of these families.
- To minimize the execution time of time-consuming functions in the STON algorithm, two approaches were used, namely coarse-grain and fine-grain. The fine-grain approach was proposed in our previous paper [1]. This paper presents more detailed description of this approach; the square root, division and trigonometry functions are performed in a fast mode.
- The coarse-grain approach has three parts; namely, parallelism, pipelining and reconfiguration. Due to the independency of for-loop cycles, three modes of parallelism are used for the iterations of these loops. The parallelism modes are limited by the resources of the FPGA device. Moreover, other techniques such as reconfiguration and pipelining are used due to specific capabilities of FPGA devices and compared the results of each technique.
- Some functions of the STON algorithm generate large matrices to preserve the required values. These matrices limit the usage of memory resources in the FPGA device. We attempt to compute the required values during the execution time and remove the area consuming matrices of the software implementation.

The rest of the paper is organized as follows: Section 2 briefly reviews existing state-of-the-art hardware acceleration schemes. Section 3 describes a detailed review of the STON algorithm and presents the first hardware implementation of this algorithm. Section 4 reports experimental. Finally, Section 5 concludes the paper.

2. Related Work

There are significant studies which report co-design methodology to build hardware accelerators for different bioinformatics applications; DNA or protein sequence/multiple sequence alignment, RNA secondary structure alignment, protein secondary structure prediction, phylogenetic likelihood and protein folding [16-40]. Sometimes, if the algorithm is composed of simple operations on extended data, single instruction multiple data (SIMD) platforms can be useful. Hardware platforms such as graphics processing units (GPU) are SIMD platforms. However, when the algorithm has high complexity with a variety of operations, GPU is not an appropriate choice. These algorithms may be more suitable for being implemented on field programmable gate arrays (FPGAs). For the best hardware implementation, the design flow must consider all factors such as programmability, performance, programming cost and sources of overhead. In general, FPGAs provide better performance and power consumption, while GPUs are easier to program [42]. Both platforms were used in different applications in bioinformatics; RNA secondary-structure alignment (FPGA) [21-22], comparing DNA and protein sequences (FPGA) [23], sequence

alignment (GPU) [24], DNA sequence Alignment (FPGA) [37-38], multiple sequence alignment (GPU) [25], multiple sequence alignment (FPGA) [26], protein structure prediction (FPGA) [27], phylogenetic likelihood (FPGA) [28] protein sequence search (GPU) [29], molecular docking (FPGA and GPU) [30], protein folding (FPGA) [31], protein sequence analysis [32], protein 3D structure alignment (GPU) [33-36] and protein search (FPGA) [40]. The 3D structure alignment methods of [33-35] use different representations of protein structures and different computational procedures and are implemented on GPU. In [33], a GPU-based implementation of the CASSERT algorithm [36] for efficiently scanning a database of protein structures was presented in order to identify structural similarities. The CASSERT algorithm is based on the two phase alignment of protein structures when matching fragments of the compared proteins. In [34], authors demonstrated a new heuristic for tableau-based protein structural and substructure searching based on simulated annealing and proposed a parallel implementation of it on GPUs. Reference [35] presented a parallel structure alignment for large-scale proteins and designed it to exploit the parallelism of GPUs. Paper [39] provides a survey on the use of hardware accelerators such as FPGAs and GPUs in the domain of computational genomics. Moreover, it describes the role of recently developed or soon to be released accelerator technologies.

In this paper, an FPGA-based accelerator is presented for comparing the three-dimensional structures of proteins based on the STON algorithm whereas significantly preserving the accuracy of the algorithm. The implemented hardware executes timing-critical parts of the design in parallel or pipeline manner. In addition to this, time-consuming operations such as division and square root are implemented in a fast mode.

3. Materials and Methods

3.1. STON Algorithm

In this section, a detailed review of the STON algorithm is described. Let A be a protein with n amino acids. As shown in figure 2, for every four consecutive amino acids i, i+1, i+2 and i+3; i<= n-3, a triple of angles $(\emptyset_1^A, \emptyset_2^A, \emptyset_3^A)$, measured in radian, is assigned to each residue i.

Figure 3 shows the pseudo code for the STON algorithm. Let p1 and p2 be two proteins with n and m residues, respectively.

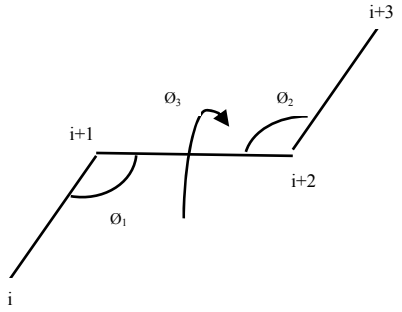

Figure 2. Definition of the three angles Ø1, Ø2 and Ø3 for four Consecutive amino acid atoms which are numbered from i to i+3

The make_degs_matrix function computes a zero/one matrix. Consider a small number ε and a matrix $M = [m_{ij}]$ such that for each i and j,

$$m_{ij}: \begin{cases} 1 & \text{if } \dfrac{\left|\emptyset_{i_1}^{P1} - \emptyset_{i_1}^{P2}\right| + \left|\emptyset_{i_2}^{P1} - \emptyset_{i_2}^{P2}\right| + \left|\emptyset_{i_3}^{P1} - \emptyset_{i_3}^{P2}\right|}{3} < \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

Non-zero entries in the matrix correspond to locally similar structures in the two proteins. The goal of the algorithm in this function is to find the longest consecutive locally similar parts of the two structures, namely, nodes and cores vectors.

In the next step, this algorithm executes in a two-dimensional space (Figure 4) where rows follow residue i from p1 and columns follow residue j from p2. Every cell in this space is calculated independent of other cells. In the set_center function, every amino acid of each protein is used to set as the new center of the protein. Therefore, these proteins will be different from the original ones.

```
Algorithm STON
Input:
  Input proteins: p1 and p2.
  Number of amino acids of the proteins: p1.size and p2.size.
  User defined parameter: distance_delta (δ), rotation_ delta (σ).
Output:
  Data with small RMSD and large size.
  ----------
  begin
    (nodes, cores) = make_degs_matrix (p1, p2);
    for i in 0 to p1.size do
      p1.set_center_axis(i);
      for j in 0 to p2.size do
        p2.set_center_axis(j);
        For k in 0 to nodes. size do
          check distance (p1, p2, nodes[k], cores[k]);
          count check will be true or false into conditions
        end for
        if (count check)
          protein _compare (p1, p2, δ, σ);
          x_axis_rotation (p1, p2);
        end if
      end for
    end for
  end
```

Figure 3. The pseudo code of the STON algorithm

j\i	0	1	2	3	4	….	m-2	m-1
0								
1								
2								
. .								
n-1								

Figure 4. Two-dimensional space of execution time of STON algorithm

The protein_compare function generates a two-dimensional matrix according to the distance between every residue of two proteins (comp-list) and distance_delta (Figure 5). This function is performed in $O(n^2)$.

In the x_axis_rotation function, the protein with smaller radius is determined and rotated by kσ angles $(0 \le k\,{}^{2\pi}/_{\sigma})$ around the other one where σ is a parameter to

be set to trade-off precision vs. speed (Figure 6). In other words, the rotation is made in σ units. For each kσ rotation, the algorithm receives the distance matrix from the protein_compare function and generates a graph called $G_{m\sigma}$. The vertex set of the graph is the set of residues of the two proteins where the Euclidean distance between the corresponding residues is smaller than distance_delta. Then the size and RMSD are computed. This function is also performed in $O(n^2)$.

```
Function: angle_check
Input:
  Input acids: a1 and a2
  ------------------------------
  Begin
    a1-size = sqrt(x1^2 + y1^2 + z1^2).
    a2-size= sqrt(x2^2 + y2^2 + z2^2).
    if (abs(arcos(x1/a1-size) − acos (x2/a2-size))<asin(δ/a1-size))
  then
    return True
  else
    return False
    end if
  end
```

```
Function: protein _compare
Input:
  Input proteins: p1 and p2.
  Number of the amino acids of the proteins: p1.size and
  p2.size.
  Distance of amino acid i from centre of the p1: p1 [i].size.
  Distance of amino acid i from centre of the p2: p2 [j].size.
Output:
  two-dimensional distance matrix: comp_list
  --------------------------------
  begin
     for i in 0 to p1.sizedo
       value1 = p1[i].size;
       for j in 0 to p2.sizedo
         value2 = p2[j].size;
         comp_value = abs(value1 - value2);
        new_check= angle_check(p1[i], p2[j]);
       if comp_value<δ and new _heck is True then
           push j in temp_vect vector;
      end for
      push temp_vect in comp_list vector;
   end for
  end
```

Figure 5. The pseudo code of protein_compare function to construct the distance matrix

The algorithm continues with the next amino acids as the centre until the end position of the proteins is reached. The protein_compare and x_axis_rotation functions must be repeated for each residue i from protein p1 and each residue j from protein p2. Therefore, the time complexity of the algorithm is $O(n^4)$, where n is the size of the proteins. This shows the long execution time of the algorithm.

3.2. Hardware Implementation

The input data for this program are generally in pub (protein database) format that consist of the coordinates of the amino acids in the proteins in the three-dimensional space. These data are floating point numbers with a maximum of three digits for the fraction part. Therefore, fixed-point representation can be used to simplify the calculations in hardware.

```
Function2:x_axis_rotation
Input:
  Input proteins: p1 and p2.
  Two-dimensional matrix: comp_list
Output:
  The best data with small RMSD and large size for the selected
residues as the centre.
  ------------------------------
  begin
    max_deg = 360 / σ ;
    form in 0 to max_deg do
      fore in 0 to comp_list.size do
        for j in 0 to comp_list[i].size do
          temp_index = comp_list[i][j];
          value = distance meter(p1[i];p2[index]);
          if sort(value) <= δ then
            push value in weight matrix;
            push j in graph matrix;
          end if
        end for
        if int_row.size ≠ 0 then
          push i in ripple vector;
      end for
      generate the Gmσ then choose the data with small RMSD
      and large size according to the weight, graph and ripple
      matrices
    end for
  end
```

Figure 6. The pseudo code of x_axis_rotation function

Table 1. Data representation in hardware: 10 bits for integer part and f bits for fraction part

S/W	H/W		
	f=8	f=10	f=12
4.499	4.49609375 00100.01111111	4.49609375 00100.01111111	4.498779296875 00100.011111111011
3.188	3.1875 00011.00110000	3.1875 00011.00110000	3.18798828125 00011.001100000010
2.243	2.33984375 00010.00101011	2.33984375 00010.00101011	2.342773435 00010.001010111110
16.594	16.58984375 10000.10010111	16.58984375 10000.10010111	16.593505859375 10000.100101111111

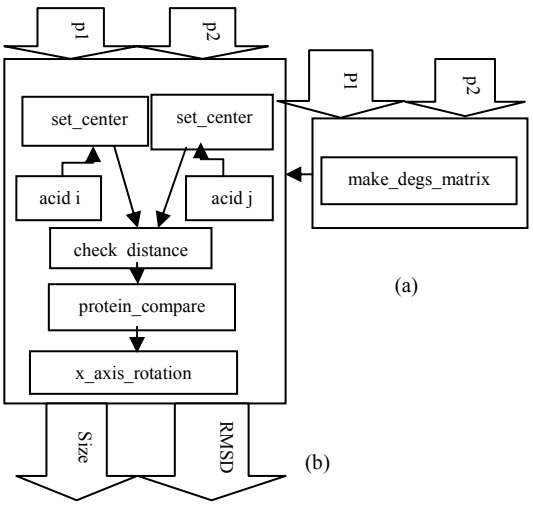

Figure 7. One cycle of hardware block for STON algorithm (a) Software implementation of make_degs_matrix function (b) Hardware implementation of the rest

The width of data was chosen by experiments such that the desired precision of comparison can be obtained. The

representation of real coordinates in sufficient precision with 22 bits was showed in table 1. Increased precision uses larger bit widths, which in turn requires more hardware resources.

Figure 7 shows top-view of the proposed architecture which implements the STON algorithm without any optimization. Each block in the diagram implements one step of the algorithm as described in the previous section, except for the make_degs_matrix function which is implemented by a software program running on the host computer.

The structure of the algorithm has two major bottlenecks, namely, protein_compare and x_axis_rotation functions in terms of run-time (Figure 8). Two classes of approaches are used in this paper to minimize the execution time of these functions, namely, coarse-grain and fine-grain.

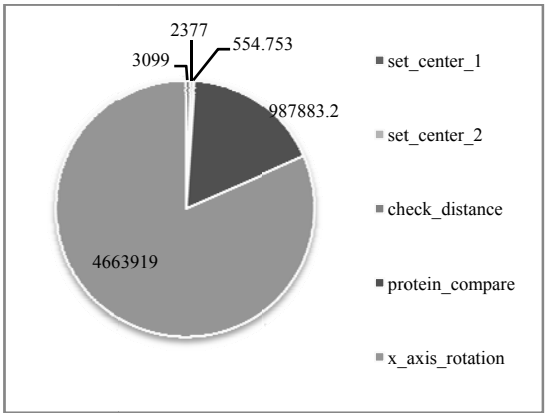

Figure 8. Execution time of functions of STON algorithm

3.2.1. Fine-Grain Approach

The protein_compare function requires n × m × p cycles to finish, where p is the number of cycles for computing division, square root and trigonometry functions. Also n and m are size of pairs of proteins. The fine-grain techniques attempt to minimize the execution time of these arithmetic functions.

Square Root: Two options were attempted to implement square root. First, ISE core generator from Xilinx toolset (http://www.xilinx.com) was used. The square root core can perform this operation in 25 cycles for high precision data. As mentioned before, the latency of square root operation has a significant effect on the run-time of the algorithm. Therefore, this operation should be performed in a faster mode. Next, the logarithm and antilogarithm functions were used to compute the square root as follows:

$$\sqrt{x} = x^{1/2} = 2^{1/2 \log x} \quad (1)$$

The logarithm and antilogarithm of a range of x values are stored in memory (Table 2), and therefore there is no need to compute them during run-time. Another operation in (1) is division by 2 that can be performed by a single right shift operation. The square root operation requires 10 cycles for 44-bit input data and 22-bit output data to compute while preserving the required precision (Figure 9). To achieve fast operation with nearly equal results as the software implementation, the second option was chosen.

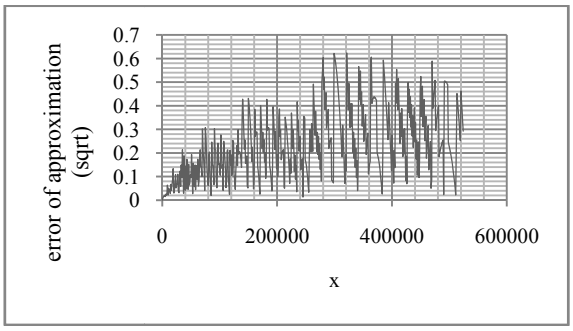

Figure 9. Error approximation of square root(x), x is a float number in [0, 500000)

Division: There are several options to compute the division operation. Due to the time criticality of this operation, we decided to choose the fixed point division which was implemented based on subsequent shifts and subtractions. Figure 10 shows the approximation of this implementation compared to the real values of division results that requires at most 32 cycles for 22-bit input data.

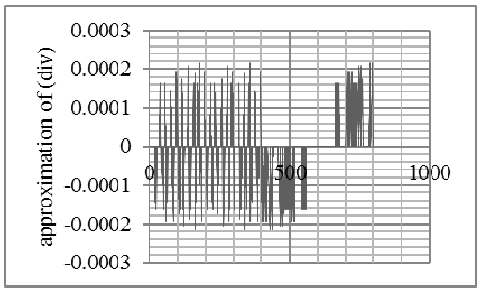

Figure 10. Difference between real and the implemented division for 800 inputs in [-500, 500]

Table 2. Required memory for tables of possible values for the square root and trigonometry functions

Function	Address and Data	Required Memory (K bit)
Tan^{-1}	$2^{18} \times 15$	3840
Cos^{-1}	$2^{14} \times 15$	240
Sin^{-1}	$2^{14} \times 15$	240
Sin	$2^{15} \times 14$	448
Coos	$2^{15} \times 14$	448
Log	$2^{9} \times 10$	0.512
Anti-log	$2^{11} \times 16$	32

Trigonometry Functions: These functions can be implemented by CORDIC classes of ISE core generator from Xilinx toolset (http://www.xilinx.com). CORDIC functions require many cycles for high precision data and use large amount of logic resources in FPGA devices [41]. Due to the limitations in logic resources to be used for parallel execution of other operations, this option was not adopted. Instead, we used some tables of possible values for the trigonometry functions previously stored in the memory available on the device. Due to the large size of these tables, the bottleneck of this method is the limited size of memory resources in FPGA devices. Therefore, as table 2 shows, approximate values of the trigonometry functions can reduce the required amount of memory.

3.2.2. Coarse-Grain Approach

This approach has three parts; namely, parallelism, pipelining and reconfiguration. In all of them, we attempt to optimize the execution time of the algorithm and use the device resources efficiently. Some functions of this algorithm generate huge matrices for preserving the required values whereas these limit the usage of the memory resources in FPGA device (e.g. weight_matric). We require to remove this area consuming parts of the software implementation and attempt to achieve required values in the code (e.g. the bio_matching function). So, there is no necessity to store these values in large memory.

Parallelism: The STON algorithm was analyzed carefully to identify parts that can be implemented in parallel. In the following, several modes of parallelism are proposed. The top function has many independent for-loops, and therefore parallel execution of the iterations in this function can improve the run-time considerably. This approach is limited by the memory size and logic resources of the FPGA device.

According to figure 4, all involved functions must be repeated for each residue i from protein p1 and each residue j from protein p2 which results in a long execution time. The first mode of parallelism executes different residues of two proteins simultaneously to speed up the algorithm by the affordable degree of parallelism that is 6 in figure 12. In other words, every 6 rows of the two-dimensional space matrix begin executing at the same time. This method reduces the run-time of the function by a factor of 6 in comparison with the basic implementation (Figure 11).

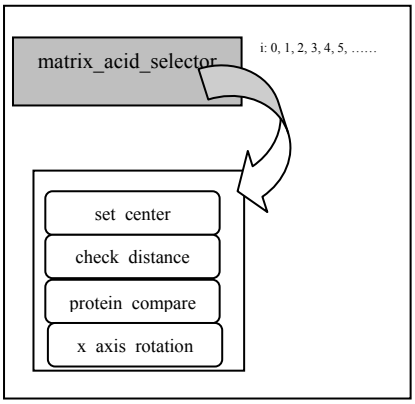

Figure 11. Basic implementation of matrix_acid_selector without any parallelism

In the x_axis_rotation function, the number of iterations of the outer for-loops depends on rotation-delta parameter (Figure 13).

Some iterations of this for-loop can be done in parallel which can reduce the execution time of the x_axis_rotation function by a factor of l, where l shows the degree of parallelism. For example, round 0 to round $\sigma \times (l-1)$ start at the same time. In the experiments, l and σ were chosen as 8 and 5º, respectively (Figure 14).

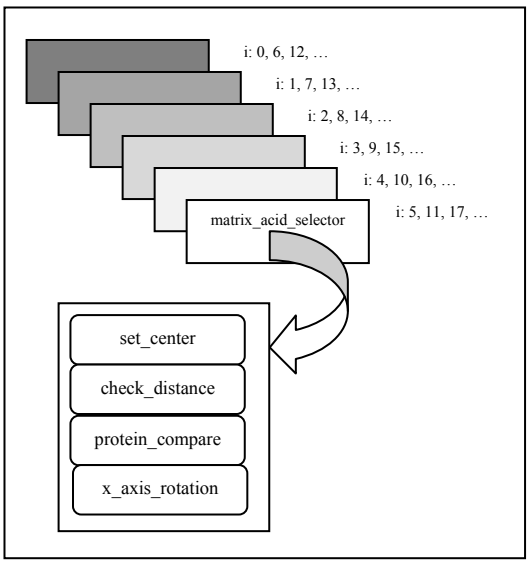

Figure 12. The first mode of parallelism:
Parallel implementation of matrix_acid_selector function

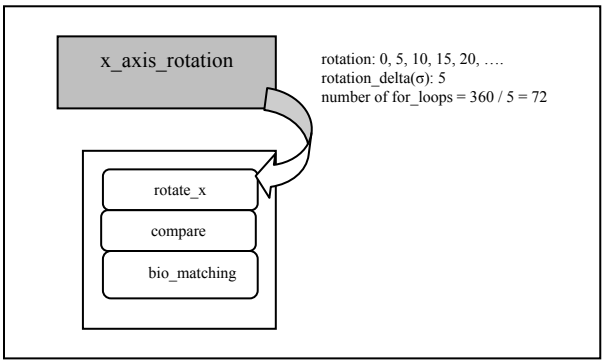

Figure 13. Basic implementation of x_axis_rotation function without any parallelism

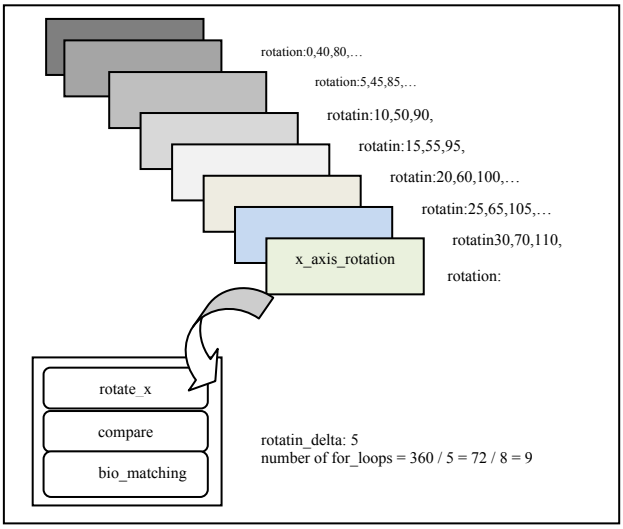

Figure 14. The second mode of parallelism:
Parallel implementation of x_axis_rotation function

The x_axis_rotation function calls rotate_x, compare and bio_matching functions. The run-time of rotate_x for every rotation angle is equal and they finish at the same time. This is the case for compare function too. However, the run-time

of bio-matching may be different for different angles (Figure 15). Therefore, if the bio_matching function for a rotation angle finishes much later than another, the parallelism may not help much. Consequently, reducing the run-time of the bio-matching function can effectively decrease the total run-time of the x_axis_rotation function.

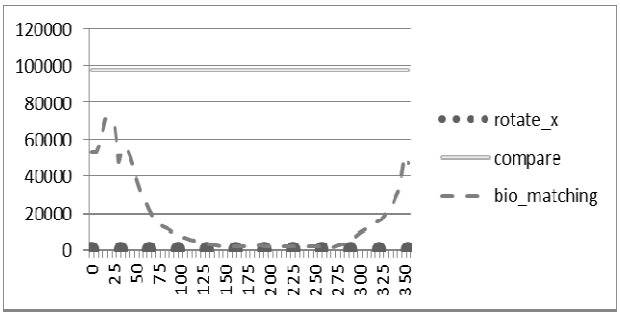

Figure 15. Comparison of the execution times of the three functions

The bio-matching function can be performed in about half of its normal run-time by executing this function for two different indices at the same time. The input data for this function are read from memory resources. Therefore, using dual-port memories of the FPGA device accelerates the run-time of this function. This is the third mode of parallelism in the coarse-grain approach that benefits from dual-port memories on FPGA devices.

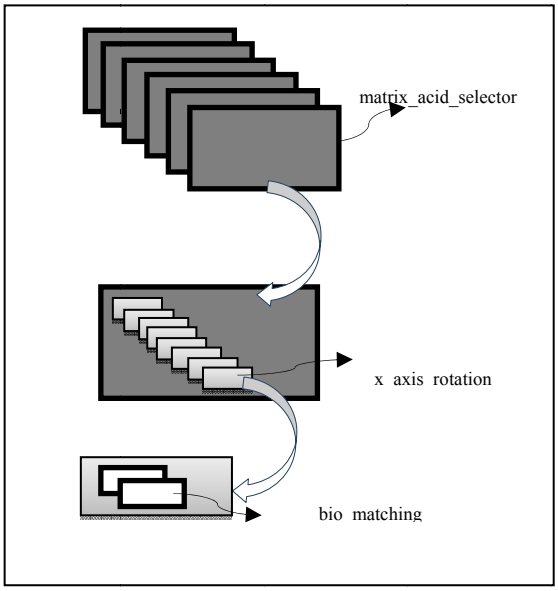

Figure 16. All modes of parallelism approach

All modes of parallelism are shown in figure 16. Six instances of matrix_acid_selector function execute simultaneously and in each module of this function, eight instances of x_axis_rotation function run at the same time. Moreover, in each instance of x_axis_rotation, two instances of bio_matching function run in parallel.

On the other hand, computing the two-dimensional matrix in protein_compare function in parallel can decrease the execution time of this function substantially. The computation part of this function has been done in angle_check (ac) function that is called in two-dimensional space. For example, instead of calling them one by one, twenty four instances of ac function are executed simultaneously (Figure 17). This technique is combined with other methods explained in the following sections, to improve the execution time of this function. It is noteworthy that two last mode, parallelism in the x_axis_rotation and bio_matching functions, were proposed in our paper [1] and are discussed with more details here.

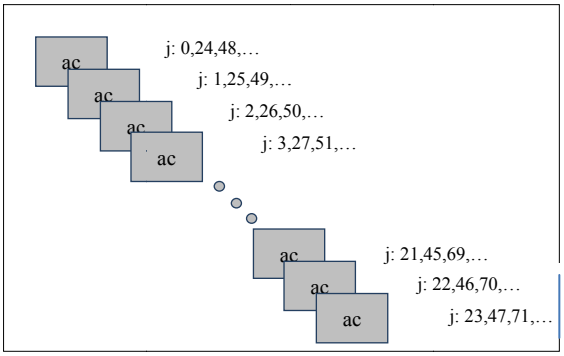

Figure 17. The fourth mode of parallelism method: parallel Implementation of protein_compare function

Pipelining: Pipelining is a technique that can be used to increase the throughput of synchronous circuits. In a pipelined design, the advantage of parallel processing capabilities of FPGA devices can be used to improve the performance of an application program developed by sequential code. To implement a pipelined design for our implementation of STON algorithm, we divide protein_compare function into three steps; 1. Read, 2. Compute and 3. Write (Figure 18). The angle_check function computes the results and during the write stage, previously computed results are written in the complist_ram matrix.

Figure 18. Three stages of pipeline implementation of protein_compare function

In the pipelining technique, each stage of an instruction is run simultaneously with the next stage of the previous instruction (Figure 19). In this paper, the protein_compare function was implemented using a combination of the parallelism explained in previous Section and the pipelining technique.

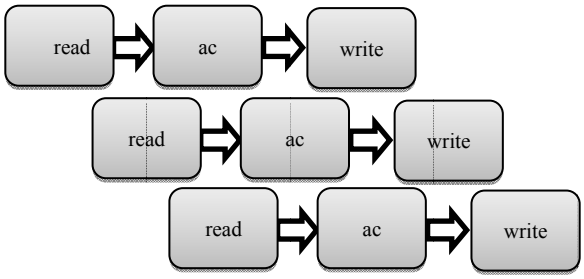

Figure 19. Implementation of protein_compare in pipelining manner

Reconfiguration: Some FPGA devices can be reconfigured dynamically during run time. We take this advantage of FPGA devices to accelerate the execution of the algorithm. The modules implementing the set center and protein_compare functions are first loaded on the device. In other words, the FPGA device is configured such that it can run those two functions. The indices of amino acids and two-dimensional matrices are computed and saved in external memories. In the next step, the device is reconfigured to implement x_axis_rotation function. In this step, for each pair of residues of two proteins, the matrices and coordinates of proteins must be read from memories and stored in the device memories and then RMSD and size of compared proteins are computed in the x_axis_rotation function. This method helps have a full available device to implement every stage of the algorithm. So, parallelism can be more effective for each stage but with overhead of reconfiguration time, the total result is not satisfactory. More discussion is included in section 4.

4. Results

The software implementation of the STON algorithm [13] was compiled and run on a 3.6 GHz AMD PhenomTM II×6 1090T system with 7.650 GB of memory. The run-time was measured for a number of structure pairs that are extracted from (http://www.rcsb.org) (Table 3). This table shows that the runtime of the algorithm is not related only to the size of proteins. It is affected by the size of core and nodes vectors and the complexity of the comp-list matrix. For the hardware implementation, the coordinates of the amino acids in the proteins in the three-dimensional space are floating point numbers with a maximum of three digits for the fraction part. Therefore, fixed-point representation can be used to simplify the calculations in hardware and save valuable hardware resources compared to floating-point representation. The width of data was chosen by experiments such that the desired precision of comparison can be obtained. Table 1 shows the representation of real coordinates with 22 bits. We chose 10 bit for integer part and 12 bit for fraction part. We described the STON algorithm in Verilog language and then synthesized, placed and routed it on two different Xilinx FPGA devices from Vertex family, namely, Virtex5:xc5vsx240t and Virtex7:xc7vx1140.

Table 3. Number of proteins and their size

p1:p2	p1-size	p2-sizse	Time(S\W)
1CEW : 1MOL	108	94	2m6.722s
1FXI : 1UBQ	96	76	2m40.204s
3HHR : 1TEN	195	89	3m42.658s
3HLA : 2RHE	99	114	3m44.429s
1CID : 2RHE	177	114	7m29.215s
1DSB : 2TRX	188	109	7m38.938s
1BGE : 2GMF	159	121	7m43.086s
2AZA : 1PAZ	129	120	6m49.721s
2MTA: 1YCC	147	108	7m54.485s
1TIE : 4FGF	166	124	11m37.162s

These devices are rich in memory and DSP block resources compared to the other members of these families. Our first design was implemented on the Virtex5 device but this device was not appropriate for implementing our other approaches as they need more resources. Virtex7 family is the high-end FPGA devices of Xilinx company (http://www.xilinx.com) that has been optimized for system performance with a large amount of hardware resources.

4.1. Experiments with Matrices Using Virtex5

At first, the design was implemented on the xc5vsx240t device with the matrices which consume large amount of memory. The x_axis_rotation function was implemented for parallelism degrees of 8 and 14. These different degrees of parallelism depend on the size of proteins and lead to different usage of memory resources for storing the matrices. The results of RMSD and size of the proteins p1 and p2 when executing the algorithm on the software and hardware platforms are reported in table 4 for the distance_delta and rotation_delta equal to 5 Å and 5°, respectively. The results of RMSD and size are almost equal for the software and hardware platforms. This is due to the proper precision adopted for the square root and division functions in the hardware implementation. The run-time of the software and hardware implementations is shown in table 5. In addition to preserving the required precision, the speedups of 1.2 to 1.6 were achieved for the hardware implementation in this case. These different speedups are due to the different run-time of the algorithm for the pairs of proteins that resulted from their various structural characteristics.

Table 4. Results of RMSD and size for the structural alignment with 5 Å distance_delta and 5° rotation_delta on the software (S/W) and hardware (H/W) platforms

p1:p2	RMSD/ size (S/W)	RMSD/ size (H/W)
1BGE, 2GMF	2.926/90	2.926/90
1CID, 2RHE	2.842/90	2.845/90
1CEW,1MOL	2.317/77	2.309/77
3HLA, 2RHE	3.010/70	3.104/73
2AZA, 1PAZ	2.811/79	2.904/78

Table 5. Results of run-time for the structural alignment with 5 Å distance_delta and 5° rotation_delta on the S/W and H/W platforms

p1:p2	Time (S/W)	Time (H/W)	Speedup
1BGE, 2GMF	7m43.086s	5m29.301s	1.406
1CID, 2RHE	7m29.215s	6m31.284	1.148
1CEW,1MOL	2m6.722s	1m17.169s	1.642
3HLA, 2RHE	3m44.429s	2m19.312s	1.611
2AZA, 1PAZ	6m49.721s	5m29.967s	1.242

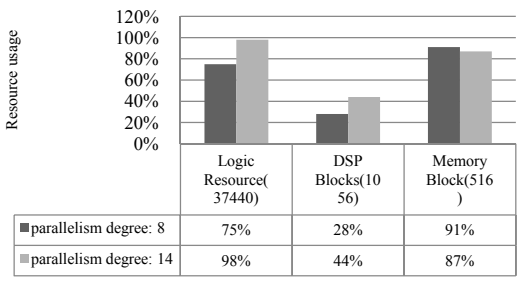

Figure 20. Average resource usage on the Virtex5 device for 5 benchmarks for two degree of parallelism

As a large amount of the logic and memory resources is used, these degrees of parallelism (8 and 14) are the maximum values which are feasible for this device (see figure 20).

4.2 Experiments without Matrices Using Virtex7

Then, our design was implemented on the second FPGA devices (xc7vx1140) and the memory consuming matrices were removed from the algorithm. In this step, the protein_compare (pc) and x_axis_rotation (xar) functions were implemented once with parallelism degree of 24 and 12 respectively (pc: 24, xar: 12) and another time with 32 and 16 respectively (pc: 32, xar: 16). The results of RMSD and size values of the protein pairs on the software and hardware platforms are reported in table 6 for distance_delta and rotation_delta equal to 5 Å and 5°, respectively.

Table 6. Results of RMSD and size for the structural alignment with 5 Å distance_delta and 5° rotation_delta on the S/W and H/W platforms

p1:p2	RMSD/ size (S/W)	RMSD/ size (H/W)
1CEW,1MOL	2.317/77	2.311/77
3HLA, 2RHE	3.010/70	3.010/70
1BGE, 2GMF	2.926/90	2.926/90
1CID, 2RHE	2.842/90	2.875/91

The run-time of the algorithms on the software and hardware platforms is reported in table 7 and table 8. Comparing table 7and table 8 shows that increasing the degree of parallelism brings about speedup improvement. According to figure 21, the parallelism of (pc: 32, xar: 16) is critical due to 80% of memory usage of FPGA device.

Table 7. Results of run-time for the structural alignment with 5 Å distance_delta and 5° rotation_delta on the S/W and H/W platforms (pc:24 and xar:12)

p1:p2	Time (S/W)	Time (H/W)	Speedup
1CEW,1MOL	2m6.722s	23.331s	5.431
3HLA, 2RHE	3m44.429s	1m5.67s	3.418
1BGE, 2GMF	7m43.086s	2m16.286s	3.398
1CID, 2RHE	7m29.215s	2m24.478s	3.109

Table 8. Results of run-time for the structural alignment with 5 Å distance_delta and 5° rotation_delta on the S/W and H/W platforms (pc: 32 and xar: 16)

p1:p2	Time (S/W)	Time (H/W)	Speedup
1CEW,1MOL	2m6.722s	20.321s	6.236
3HLA, 2RHE	3m44.429s	56.401s	3.979
1BGE, 2GMF	7m43.086s	1m59.184s	3.855
1CID, 2RHE	7m29.215s	1m40.936s	4.450

These results show that the required precision is properly preserved and speedup of 3x to 6x is achieved (Table 7 and Table 8) but this implementation is not satisfactory as large amount of logic and DSP resources are wasted (Figure 21).

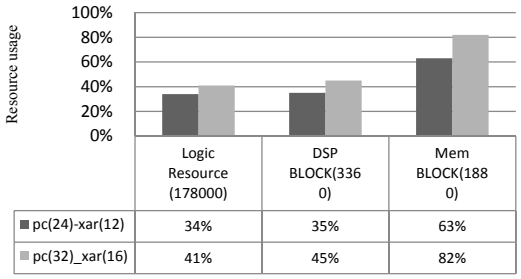

Figure 21. Average resource usage on the Virtex7 device for 4 benchmarks for two pairs of parallelism degree

4.3. Experiments with Run-Time Reconfiguration

In this experiment, the reconfiguration capability of FPGA devices was exploited and applied in addition to the previous methods in order to increase the degree of parallelism. In the first configuration, the protein_compare function was implemented with parallelism degree of 48 and the results such as two-dimensional matrix and set-centered proteins were stored in external memories. In the second configuration, the required input data were downloaded from the external memories to the internal memories and the x_axis_rotation function was implemented with parallelism degree of 24. The run-time of each configuration of the algorithm on the hardware platform is reported in table 9for the distance_delta and rotation_delta equal to 5 Å and 5°, respectively.

Table 9 Run-time of each step of reconfiguration technique with 5 Å distance_delta and 5° rotation_delta for the H/W platform (pc:48 and xar:24)

p1:p2	Conf1:Pc_48	Conf2:Xar_24	reconfiguration time
1CEW,1MOL	1.754s	15.433s	25s
3HLA, 2RHE	3.059s	39.333s	33s
1BGE, 2GMF	5.533s	1m13.355s	39s
1CID, 2RHE	17.393s	1m7.585s	53s

Total run-time is the sum of the time required for conf1, conf2 and reconfiguration time that are reported for the software and hardware platforms in table 10. Comparing table 8 and table 10 shows that the reconfiguration techniques help to increase the degree of parallelism but can't obtain desirable speedup because of the time overhead of reconfiguration in the current FPGA technology.

Table 10 Results of run-time for the structural alignment with 5 Å distance_delta and 5° rotation_delta for the S/W and H/W platforms (with reconfiguration, pc: 48 and xar: 24)

p1:p2	Time(S/W)	Time (H/W)	Speedup
1CEW,1MOL	2m6.722s	42.187s	3.004
3HLA, 2RHE	3m44.429s	1m15.392s	2.977
1BGE, 2GMF	7m43.086s	1m57.888s	3.928
1CID, 2RHE	7m29.215s	2m7.978s	3.510

The total amount of resource usage is reported in figure 22. For this degree of parallelism, if no reconfiguration is performed, the total usage of memory resources would be more than the available resources on the FPGA device. Therefore, the reconfiguration technique makes this parallelism feasible but the technique used in Section 4.2 leads to better speedup in practice.

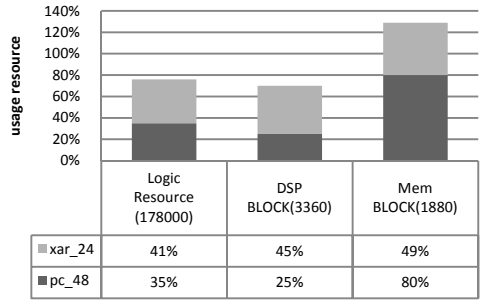

Figure 22. Average resource usage on the Virtex7 device for 4 benchmarks for two configurations

To show the time overhead of the reconfiguration method, we assumed to have a hardware with adequate resources according to figure 22 and implemented protein _compare and x_axis_rotation functions with the required parallelism degrees (pc:48 and xar:24). Table 11 shows that the speedup increases more than 50% compared to table 10. Therefore, the reconfiguration method have 50% reduction in speedup because additional time is needed to do the reconfiguration.

Table 11. Results of run-time for the structural alignment with 5 Å distance_delta and 5° rotation_delta for the S/W and H/W platforms (without careconfiguratioin, pc:48 and xar:24)

p1:p2	Time (S/W)	Time (H/W)	Speedup
1CEW,1MOL	2m6.722s	17.187s	7.373
3HLA, 2RHE	3m44.429s	42.392s	5.294
1BGE, 2GMF	7m43.086s	1m18.888s	5.870
1CID, 2RHE	7m29.215s	1m24.978s	5.286

4.4. Selected Parallelism

In the next experiment, the matrix_acid_selector was implemented with parallelism degree of 6. Moreover, the x_axis_rotation function was implemented with parallelism degree of 8 but the protein_compare function was not parallelized to save resources for the other functions. The results of RMSD and size of the protein pairs for the software and hardware implementations are reported in table 12. With obtaining much better speedups, more protein pairs of (http://www.rcsb.org) were attempted in these experiments.

The run-time of the algorithm for software and hardware platforms is reported in table 13. The parallel implementation of the matrix_acid_selectror (top) with the parallelism degree of 6 and the x_axis_rotation (xar) with the degree of 8 (top-6: xar-8) resulted in speedup of 7.1 to 11.5. Comparing this speedup with the previous results indicates that parallel implementation of the top-module of the algorithm (e.g., matrix_acid_selector) is more effective than the other functions (e.g., protein_compare).

Table 12 Results of RMSD and size for the structural alignment with 5 Å distance_delta and 5° rotation_delta on the S/W and H/W platforms

p1:p2	RMSD/size(S\W)	RMSD/size(H\W)
1CEW : 1MOL	2.317/77	2.310/77
1FXI : 1UBQ	3.08756/55	3.084/55
3HHR : 1TEN	2.43139/83	2.433/82
3HLA : 2RHE	3.010/70	3.104/73
1CID : 2RHE	2.842/90	2.845/90
1DSB : 2TRX	2.67874/76	2.738/78
1BGE : 2GMF	2.926/90	2.926/90
2AZA : 1PAZ	2.81128/79	2.904/78
2MTA: 1YCC	2.64973/81	2.650/81
1TIE : 4FGF	3.03966/108	3.031/108

Table 13 Results of run-time for the structural alignment with 5 Å distance_delta and 5° rotation_delta on the S/W and H/W platforms (matrix_acid_selector: 6 and x_axis_rotation: 8)

p1:p2	Time(S\W)	Time(H\W)	Speedup
1CEW : 1MOL	2m6.722s	11.229s	11.285
1FXI : 1UBQ	2m40.204s	13.868s	11.552
3HHR : 1TEN	3m42.658s	29.468s	7.556
3HLA : 2RHE	3m44.429s	24.123s	9.304
1CID : 2RHE	7m29.215s	58.791s	7.641
1DSB : 2TRX	7m38.938s	53.536s	8.573
1BGE : 2GMF	7m43.086s	43.36s	10.680
2AZA : 1PAZ	6m49.721s	47.759s	8.579
2MTA: 1YCC	7m54.485s	46.919s	10.113
1TIE : 4FGF	11m37.162s	97.141s	7.177

Figure 23 shows the average resource usage on the FPGA device for the ten attempted benchmarks. Comparing figure 21 with figure 23 shows that by this selection of functions, the resource usage for memory, logic and DSP blocks are at same level. On the other hand, in figure 21 memory usage is about twice the logic resource usage and much of the FPGA resources are wasted. Therefore, parallel implementation of the protein_compare function is not preferred because of using more memory resources versus logic and DSP resources. Moreover, its parallel implementation is not efficient in terms of run-time of the algorithm.

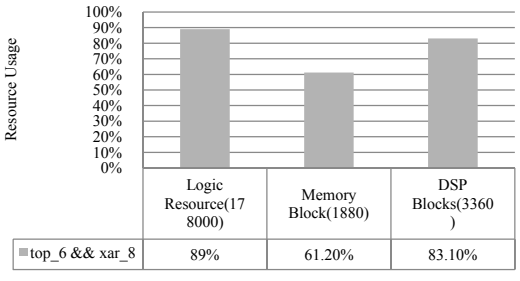

Figure 23. Average resource usage on the Virtex7 device for 10 benchmarks

It is worth noting that the hardware accelerator was clocked at most at 100 MHz while the software

implementation was executed on a PC with 3 GHz clock frequency. The same values for threshold and user-defined parameters were used in both hardware and software implementations. As shown in the reported tables, the speedup gained from our FPGA implementation compared to software ranges from 2x to 12x. The reason behind this speedup of the FPGA implementation, in spite of the huge differences in the clock frequency, is mostly due to the parallelism used in the FPGA. As comparing three-dimensional structures of proteins is normally run a large number of times to find the weak or strong linked proteins in databases, this speedup can be very advantageous in the real world experiments. On the other hand, the low frequency of clock for FPGAs is mainly due to their programmable structures. If the proposed algorithm is implemented on application-specific integrated circuits (ASIC), the speedup of the proposed architecture is expected to be much more.

5. Conclusions

In this paper, a hardware accelerator was proposed for comparing three-dimensional structures of proteins. Since the software implementation of the procedure takes a long time to complete for large databases, hardware acceleration can make it more practical. In this paper, the detailed FPGA-based implementation of the STON algorithm was presented. To the best of our knowledge, this is the first hardware implementation of the STON algorithm. The results show that the proposed hardware accelerator leads to the speedup of 10x on average. The proposed architecture can be implemented on ASICs to improve the speedup even more.

References

[1] S. Kashi, and M. Saheb Zamani, "Hardware Acceleration of STON Algorithm for Comparing 3-D Structure of Proteins," Euro micro Conference on Digital System Design, 2012.

[2] R. H. Lathrop, "The protein threading problem with sequence amino acid interaction preferences is NP-complete," Protein Engineering, vol. 7, no. 9, pp. 1059–1068, 1994.

[3] A. Falicov, and F. E. Cohen, "A surface of minimum area metric for the structural comparison of proteins," Journal of Molecular Biology, vol. 258, no. 5, pp. 871-892, 1996.

[4] M. Gerstein, and M. Levitt, "Comprehensive assessment of automatic structural alignment against a manual standard, the SCOP classification of proteins," Protein Science, vol. 7, no. 2, pp. 445-456, 1998.

[5] W. R. Taylor, "Protein structure comparison using iterated double dynamic programming," Protein Science, vol. 8, no. 3, pp. 654-665, 1999.

[6] A. L. Jewett, C. C. Huang, and T. F. Ferrin, "MINRMS: an efficient algorithm for determining protein structure similarity using root-mean-squared-distance," Bioinformatics, vol. 19, no. 5, pp.625-634, 2003.

[7] I. N. Shindyalov, and P. E. Bourne, "Protein structure alignment by incremental combinatorial extension (CE) of the optimal path," Protein Engineering, vol. 11, no. 9, pp. 739-747, 1998.

[8] L. Holm, and C. Sander, "Protein Structure comparison by alignment of distance matrices," Journal of Molecular Biology, vol. 233, no. 1, pp. 123-138, 1993.

[9] W. R. Taylor, and C. A. Orengo, "Protein Structure alignment," Journal of Molecular Biology, vol. 208, no. 1, pp. 1-22, 1989.

[10] A. Godzik, J. Skolnick, and A. Kolinski, "Regularities in interaction patterns of globular proteins," Protein Engineering, vol. 6, no. 8, pp. 801-810, 1993.

[11] D. P. Yee, and K. A. Dill, "Families and the structural relatedness among globular," Protein Science, vol. 2, no. 6, pp. 884-899, 1993.

[12] J. D. Szustakowski, and Z. Weng, "Protein structure alignment using a genetic algorithm," Proteins, vol. 38, no.4, pp. 428-440, 2000.

[13] H. M. Grindley, P. J. Artymiuk, D. W. Rice, and P. Willett, "Identification of tertiary structure resemblance in proteins using a maximal common sub graph isomorphism algorithm," Journal of Molecular Biology, vol. 229, no. 3, pp.707-721, 1993.

[14] C. Eslahchi, H. Pezeshk, M. Sadeghi, A. M. Rahimi, H. Maboudi Afkham, and S. Arab, "Ston: a novel method for protein three-dimensional structure comparison," Computers in Biology and Medicine, vol. 39, no. 2, pp. 166-172, 2009.

[15] J. T. Dudley, and A. J. Butte, "A Quick Guide for Developing Effective Bioinformatics Programming Skills," PLoS Comput Biol, vol. 5, no. 12, 2009.

[16] L. Hasan, Z. Al-Ars, and S. Vassiliadis, "Hardware Acceleration of Sequence Alignment Algorithms-An Overview," Design and Technology of Integrated Systems in Nan scale Era, 2007.

[17] A. Wirawan, CK. Kwoh, NT. Hieu, and B. Schmidt, "CBESW: Sequence Alignment on Play station 3," BMC Bioinformatics, vol. 9, 2008.

[18] P. L. McMahon, "Accelerating Genomic Sequence Alignment using High Performance Reconfigurable Computers," University of Cape Town, Master's Thesis, 2008.

[19] S. Lloyd, and Q. O. Snell, "Hardware accelerated sequence alignment with trace back," International Journal of Reconfigurable Computing, 2009.

[20] G. M. Striemer, and A. Akoglu, "Sequence Alignment with GPUZ; Performance and design challenges," Parallel & Distributed Processing, 2009.

[21] J. Moscola, R. K. Cytron, and Y. H. Cho, "Hardware-Accelerated RNA Secondary-Structure Alignment," ACM Transactions on Reconfigurable Technology and Systems, vol.3, no.14, pp. 1–44, 2008.

[22] F. Xia, Y. Dou, G. Lei, and Y. Tan, "FPGA accelerator for protein secondary structure prediction based on the GOR algorithm," BMC Bioinformatics, vol. 12, 2011.

[23] P. Faes, B. Minnaert, M. Christiaens, E. Bonnet, Y. Saeys, D. Stroobandt, and Y. Van de Peer, "Scalable hardware accelerator for comparing DNA and protein sequences," International Conference on Scalable Information Systems, 2006.

[24] M. Schatz, C. Trapnell, A. Delcher, and A. Varshney, "High-Throughput Sequence Alignment Using Graphics Processing Units," BMC Bioinformatics, vol. 8, no. 474, 2007.

[25] Y. Liu, B. Schmidt, and D. L. Maskell, "MSA-CUDA: Multiple Sequence Alignment on Graphics Processing Units with CUDA," Application specific Systems, Architectures and Processors, 2009.

[26] T. F. Oliver, B. Schmidt, D. Natehan, R. Clemens, and D. Maskell, "Multiple Sequence Alignment on an FPGA," Parallel and International Systems, 2005.

[27] J. Advait, G. Pulkit, J. Priyanka, M. Balakrishnan, and P. Kolin, "FPGA accelerator for protein structure prediction algorithm," Southern Conference on Programmable Logic, 2009.

[28] S. Zierke, and J. D. Bokos, "FPGA Acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods," BMC Bioinformatics, vol. 11, 2010.

[29] J. Zhang, H. Wang. H. Lin, and W. C. Feng, "BLASTP: Fine-Grained Parallelization of Protein Sequence Search on a GPU," Parallel and Distributed Processing, 2014.

[30] I. Pechan, and B. Feher, "Molecular Docking on FPGA and GPU Platforms," Field Programmable Logic and Applications, 2011.

[31] W. T. Sung, "Efficiency Enhancement of Protein Folding for Complete Molecular Simulation via Hardware Computing," Bioinformatics and Bioengineering, 2009.

[32] J. Fernando Eusse, N. Moreano, A. C. M. A. de Melo, and R. P. Jacobi, "A Protein Sequence Analysis Hardware Accelerator Based on Divergences," International Journal of Reconfigurable Computing, 2012.

[33] D. Mrozek, M. Brożek, and B. Małysiak-Mrozek, "Parallel implementation of 3D protein structure similarity searches using a GPU and the CUDA," Journal of Molecular Modeling, vol. 20, no. 2, 2014.

[34] A. D. Stivala, P. J. Stuckey PJ, and A. I. Wirth "Fast and accurate protein substructure searching with simulated annealing and GPUs," BMC Bioinformatics, vol. 11, 2010.

[35] B. Pang, N. Zhao, M. Becchi, D. Korkin, and C-R Shyu, "Accelerating large-scale protein structure alignments with graphics processing units," BMC Research Notes, vol. 5, 2012.

[36] D. Mrozek, and B. Małysiak-Mrozek, "CASSERT: a two-phase alignment algorithm for matching 3D structures of proteins," Computer Networks, vol. 370, p. 334-343, 2013.

[37] W. Abou El-Wafa, A. G. Seliem, and H. F. A. Hamed, "Hardware Acceleration of Smith-Waterman Algorithm for Short Read DNA Alignment Using FPGA," Computer Software and Applications, 2016.

[38] J. Arram, T. Kaplan, W. Luk, and P. Jiang, "Leveraging FPGAs for Accelerating Short Read Alignment," IEEE/ACM Transactions on Computational Biology and Bioinformatics, accepted for publication.

[39] S. Aluru, and N. Jammula, "A Review of Hardware Acceleration for Computational Genomics," IEEE Design & Test, vol. 31, no. 1, pp. 19-30, 2014.

[40] E. Rucci, C. Garcia, G. Botella, A. E De Giusti, M. Naiouf, and M. Prieto-Matias, "Smith-Waterman Protein Search with Open CL on an FPGA," TRUSTCOM-BIGDATASE-ISPA, 2015.

[41] "Logy CORE IP CORDIC v5.0," Product specification, DS858 October 19, 2011.

[42] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating Compute-Intensive Applications with GPUs and FPGAs," Symposium on Application Specific Processors, 2008.

Somayeh Kashi received the B.Sc. degree in Computer Engineering from Isfahan University of Technology in 2009 and the M.Sc. degree in Computer Engineering from Amirkabir University of Technology in 2012. She is currently working towards Ph.D. degree in School of Computer Engineering, Iran University of Science and Technology. Her research interests are bioinformatics, network on chips, reconfigurable systems, and electronic design automation.
E-mail: s-kashi@aut.ac.ir

Morteza Saheb Zamani received the B.Sc. degree in Computer Engineering from Isfahan University of Technology in 1989, and the M.Eng.Sc. and Ph.D. degrees in Computer Engineering and Computer Science from the University of New South Wales, Australia in 1992 and 1996, respectively. He joined Amirkabir University of Technology in 1996 and is now an associate professor at Computer Engineering and IT department. His research interests are biological design automation, reconfigurable systems, and electronic design automation.
E-mail: szamani@aut.ac.ir

Paper Handling Data:

Submitted: 01.08.2016
Received in revised form: 16.09.2016
Accepted: 03.11.2016
Corresponding author: Dr. Morteza Saheb Zamani, Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran.