Heterogeneous HW/SW FPGA-Based Embedded System for Database Sequencing Applications

Talal Bonny
Department of Electrical and Computer Engineering
University of Sharjah, Sharjah, UAE

***Abstract*—Database sequencing applications including sequence comparison, searching, and analysis are considered among the most computation power and time consumers. Heuristic algorithms suffer from sensitivity while traditional sequencing methods, require searching the whole database to find the most matched sequences, which requires high computation power and time. This paper introduces a dynamic programming technique based-on a measure of similarity between two sequential objects in the database using two components, namely frequency and mean. Additionally, database sequences that have the lowest scores in the comparison process were excluded such that the similarity algorithm between a query sequence and other database sequences is applied to meaningful parts of the database. The proposed technique was implemented and validated using a heterogeneous HW/SW FPGA-based embedded system platform. The implementation was partitioned into (1) hardware part (running on logic gates of FPGA) and (2) software part (running on ARM processor of FPGA). The validation study showed a significant reduction in computation time by accelerating the database sequencing processes by 60% comparing to traditional known methods.**

***Keywords*—Database; sequence comparison; dynamic programming; FPGA**

I. Introduction

Sequence analysis, comparing, alignment, or any sequence computing application are common concepts in a variety of research fields. The rapid analysis of Protein and DNA sequences, in Biology, is performed on large databases of sequences in order to search for close matches in specific sequences, such as a protein that has been discovered recently [1], [2], [3], [4]. If the sequences were correlated, then new drugs would be created, and the invention of better techniques could be possible in order to treat the disease.

"String Editing" [5], which is a form of sequence comparison, is used, in Computer Science, as error correction mechanism similar to the one found in spell checkers and file comparators. The way it works is by comparing and searching through a large sequence database of words for a particular one. Sequence comparison is also used to find the Longest Common Subsequence (MLCS) between two input strings [6]. Using Sequence Comparison in Social Science [7], [8], [9], [10], involves a broad selection of topics, such as national histories and daily life careers. In video processing, a video's temporal and spatial info that is contained in a frame sequence, are aligned to find the repeated contents in a video's stream [11]. In the previously mentioned applications, the time consumption is high, since they rely on the comparison of a specific sequence with a huge sequences database. Approximate solutions can be found via the use of Heuristic algorithms, which are problematic since they are sensitive and my trim searches which may result in missing some important homologies, unexpectedly. However, deterministic algorithms can guarantee that the optimal comparison result is returned from the two sequences as they are based on dynamic programming principles. In these algorithms, a query sequence, which is the sequence under search, is compared with every sequence in the database. A similarity score will be computed for every comparison process. The higher the score, the closer is the database sequence to the query.

Dynamic programming based algorithm breaks down the large computing problems into a smaller subset of problems, where each one's result depends dynamically on the other. The end results are presented in a time which is proportional to the product of the two lengths of both sequences under comparison, such as if n represents the query sequence's length, and m represents that of the database sequence, the optimal alignment from the previous algorithms will be provided in n x m steps. Hence, we can conclude that searching a whole database will grow the computational time in a linear fashion with respect to the size of the database.

Powerful and efficient techniques have been suggested to compute these huge amounts of data in a more realistic time using the FPGAs [12], [13]. The authors of [14] proposed a Recursive Variable Expansion (RVE) based technique and implemented it on the FPGA. In [15], the authors partitioned the database sequences into two sections based on the sequence length by running the short sequences on the CPU and the long sequences on the GPU. In [16], the authors combined a sequence alignment algorithm with linear space complexity using a GPU. The authors in [17] have suggested a measurement of similarities across two web pages, as well as a clustering method of the web sessions via a Fast Optimal Global Sequence Alignment algorithm (FOGSAA). In [18], the authors provided a comparative analysis of the various optimization strategies of the Smith-Waterman algorithm and contributed to the dynamic programming of sequence alignment and the implementation in FPGA. [19] presented a run-time efficient hardware-software partitioning technique for FPGAs. [20] presented a method to approximate dynamic programming for direct model predictive control (MPC) of current reference tracking in power electronics and the FPGA implementation is validated on a variable speed drive system with a three-level voltage source converter.

Every previous application and submission has used the similarity measures to find how close the objects are to each other. This object may be a sequence database, a string file, a stream of a video, or a webpage. Objects consist of various frequently reoccurring letters (codes) and are sequences in a database.

In this research, we suggest new similarity measures (similarity functions as we refer to them) that are based upon the mathematical parameters; the mean of the codes in the sequences on the database, and their frequency. We can reduce the required time to measure the similarity of two objects (database sequences) by using our similarity functions. Also, we will present a new efficient technique that reduces the computational time required to compute similarities between the entire database sequences and the query sequence by the exclusion of the sequences which obtain a low score in the comparison process. In such cases, we have to apply dynamic programming algorithm on part of the database and not on the entire database. We also develop a heterogeneous HW/SW FPGA-Based Embedded System which exploits the new features of the Xilinx ZYNQ-7000 series, by partitioning the implementation into (1) hardware part (running on logic gates of FPGA) and (2) software part (running on ARM processor of FPGA). As the sorting part of the technique requires more computations, we run it on ARM processor of FPGA, while we leave the part which can be parallelized to be run on logic gates of the FPGA. Using our technique, we can size-down the comparison application time by 60% with respect to the traditional methods. The selling point in this technique is the ability to use it in conjunction with the currently available methods to prove their validity.

As a case study, we apply our technique to the dynamic programming based algorithm the Needleman-Wunsch [21].

Our contributions are as follows:

1) Proposing new similarity functions to measure the similarity between two objects based on mathematical parameters.

2) Minimizing the computation time required for database sequence computing application by proposing a novel technique.

3) Heterogeneous HW/SW FPGA-Based embedded system is proposed which executes the hardware part of the technique on the logic gates of FPGA, and the software part on the ARM processor of FPGA

The following sections in this paper will be organized as the following. Section II will demonstrate the sequential applications via applying the traditional methods. Section III demonstrates our similarity functions. Section IV will introduce the time complexity and the reduction of the computational time of our technique.

The proposed FPGA implementation is presented in Section V. The experimental results are presented in Section VI. We conclude this paper in Section VII.

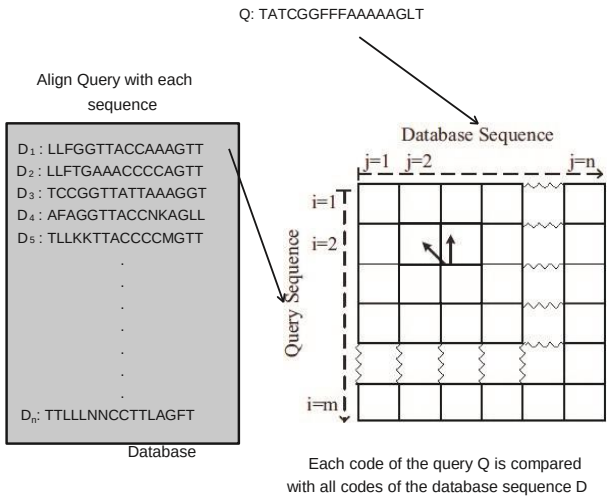

Fig. 1. The Sequential Alignment, where every Element (Code) of the Query Sequences has to be Compared to every Element in the Database Sequences.

II. Sequencing Applications using Traditional Methods

Traditional methods align two sequences together and compute an alignment score (AS) that represents the amount of similarity between them. In order to search for a query sequence in a database, the query should be aligned with every sequence in a database. Every alignment process compares every element (code) of a query sequence with every element in a database (as mentioned in Figure 1). Based on the results of the comparison, an alignment score is calculated, where it can be a match or a mismatch. In the case of a mismatch, any of the three operations can be performed; insertion, deletion, or substitution. A gap can be added to the sequences to make them closer to each other. The scores of these operations are predefined. The alignment score (AS) of every alignment process between the database and query is computed using the following formula:

$$AS = (\#of\ matches \times match\ score) + (\#of\ gaps \times gap\ score) + (\#\ of\ mismatches \times mismatch\ score) \quad (1)$$

Mismatches are usually negative along with the gap scores, however matches are positive scores. Hence, we can understand that the matches increase the alignment score, whereas gaps and mismatches decrease it. The scores of each are given as input parameters. The optimal number of every score can be calculated via the use of any dynamic programming based algorithm, just like the Needleman-Wunsch algorithm [21]. Then, a matrix containing the scores is generated. It has the dimensions of m x n (where m is the query sequence length and n is the database sequence length). The optimal score in every element of this matrix is calculated by adding the previous score to the current match score while subtracting the gap penalties. Every element can be positive, negative, or 0 value, according to the predefined score.

When the T matrix is full, we use a method called trace back to determine the optimal sequence alignment score from the scoring matrix. This method remembers the position in the scoring matrix the provided the best score so far. This position can align, or be next to a gap, depending on the traceback matrix's information. Multiple maximum alignments may exist. As mentioned earlier, the time required to obtain the optimal alignment of both sequences (one database sequence and the query) is proportional to the product of their lengths, which is n x m steps.

III. Our Proposed Similarity Functions

This section shows the new similarity measures we proposed, or similarity function, as we call them. These functions rely on the mathematical parameters; the mean of each database sequence's codes, and their frequency. Since every sequence contains different reoccurring codes, we introduce the **"Frequency Function",** which is our first similarity function. It relies on the code frequencies for every sequence. This frequency represents the number of the repeated codes in a sequence. This indicates the similarity between both sequences. As an example, when the frequency of the codes is close one to another, then the two sequences may be similar.

Most of the time, the sequence may contain multiple varying codes, and in order to realize the similarities between the two sequences, we find the frequency difference score (FDS). This score is the sum of the absolute values of the differences between the two sequences for each type of code.

Mathematically, the FDS of the database sequence 'D' and the query sequence 'Q' is defined using the following formula, considering that both of them have an alphabet of 'n' codes:

$$FDS = |Freq\_code\_1(Q) - Freq\_code\_1(D)| + |Freq\_code\_2(Q) - Freq\_code\_2(D)| + |Freq\_code\_3(Q) - Freq\_code\_3(D)| + \cdots + |Freq\_code\_n(Q) - Freq\_code\_n(D)| \quad (2)$$

Where "*Freq _code* $1(Q)$" is the frequency of code 1 in the query sequence. "*Freq code* $1(D)$" is the frequency of code 1 in the database sequence, etc.

The first function we proposed does not always yield a correct output. For example, when two sequences have a close number of code frequencies, but they are distributed differently among these two sequences, the frequency score will not be the correct measurement of similarity.

So, we introduce a 2nd function, dubbed as the **"Mean Function"**. The mean, or the average, is obtained via the division of the sum of codes by the number of observations, as defined in the following formula:

$$\bar{X} = \frac{\sum_{i=1}^{i=n} X_i}{n} \quad (3)$$

In our case, it is better to locate a central code concentration location. This is a good indication of similarities between two sequences. For example, if the mean is close to that of another sequence, it is a good indicator of the possibility of a similarity. When the sequence has varying codes, we compute the mean difference score (MDS) to find if a similarity exists among them, as defined in the FDS. This function, as well, does not always yield the correct output. As an instance, when two sequences have a close mean to each other but the number is not the same, the "Mean Score" is not correct to identify the similarity.

Because of the two problems, we suggest our third function, which we refer to as the **"Frequency+Mean Function".** The new score FMDS is the sum of both, the FDS and MDS. This is a good indication of a similarity since it considers both, the number of codes and their concentration.

IV. Sequencing Applications using Our Technique

In this section, we are going to propose our new efficient technique in order to size-down the required computational time for sequencing applications. Our technique uses the similarity functions we introduce in section III in order to compute the similarity between a query sequence with every database sequence.

The database in a sequencing application has huge amounts of sequences (as described in Section I). In order to align the sequence of the query (Q) to everyone in the database (D), we apply dynamic programming based algorithms on every pair, as described in section II.

Our technique filters the database where it gets rid of the sequences that are not close to the query from the searching process, so the algorithm is applied to the sequences that are similar to the query.

Our technique identifies the similarity functions and keeps them close to each related sequence. This procedure may require huge amount of time since the database may include huge amount of sequences, which is done off-line (independent from the comparison process). Hence, the time does not matter since we perform it only once to prepare our database for future comparison processes.

Our technique computes the scores of FDS, MDS, and FMDS when the query sequence has to be compared with the database sequences. Next, it sorts the sequences in the database in accordance to the difference score (DS) they produce, where the lower scores (closer to the query sequence) are at the top of the database.

Next, it applies dynamic programming-based algorithms on the sequences that have a low score, which was identified in the previous step. The sequences resulting in high difference scores are omitted from the search. This gives us the best alignment in a reasonable amount of time. The upcoming section will demonstrate how fast this technique is with respect to the traditional methods by demonstrating the time complexity.

A. Complexity of our Technique

As the dynamic programming based algorithm uses dynamic programming, the complexity that results to align one sequence is *O(m x n)*. This can be multiplied by the number of sequences 's' in a database, which will result in *O(m x n x s).*

In the case of our presented technique, let's assume that we have *c* different codes. In order to compute the distribution of these *c* codes in the query sequence, we have to scan it along its length. If its length is assumed to be m, then m steps are required to perform the scan. Computing the DS between the query and a sequence on the database requires c steps to subtract for the *c* codes, and *c-1* steps for the summation process. If *s* is the sequences amount, then the steps we need are *((2c-1) x s)* steps to calculate the difference score. Sorting the *s* scores via the use of QuickSort requires (*s x log s*) steps.

If we select 50% of the sequences in the database to apply the dynamic programming based algorithm on, then performing this step requires m x n x s/2 steps.

The total number of steps required in our technique is

$$(m + (2c - 1) \times s + s \times \log s) + (m \times n \times \frac{s}{2}) \quad (4)$$

i.e., the complexity is $O(m \times n \times s/2)$.

As an example, assuming a query length m of 500, and s = 10000, each sequence in this database has a length n of 500. Performing the comparison on the query sequence with all the sequences in the database using the traditional methods requires 500 x 500 x 10000 = 2500000000 steps, 2,500 Million. However, via the use of our technique, codding the data with 4 different codes requires:

500 + (7 x 10000) + (10000 x log 10000) + (500 x 500 x 5000) = ≈ 1250 Million steps.

Our technique saves 50% of the required time to align the sequences using the traditional methods.

V. FPGA-BASED EMBEDDED SYSTEM DESIGN

The ZedBoard FPGA prototyping board, from Xilinx[1], is used to implement our embedded design. The board contains Zynq-7000 All Programmable SoC FPGA[2], which is released by Xilinx into the market, as a new series of products. The Zynq-7000 FPGA combines ARM dual-core 1GHz Cortex-A9 MPCore Processing System (PS) that comes with a high-performance memory system, with Xilinx 28 nm Programmable Logic (PL). Our technique has some parts which can be run in parallel, such as computing FDS, MDS, or FMDS for each sequence. These computations are sequence independent and therefore, they can be run in parallel using the logic gates of the FPGA. On the other hand, our technique has another part which needs powerful computations such as sorting the database sequences based on their similarity score. This part can be run on the ARM processor of the FPGA. By partitioning the implementation, we exploit the new features of Xilinx Zynq7000 series to design heterogeneous HW/SW FPGA-Based embedded system.

A. FPGA Implementation

To implement the embedded design of our technique on the FPGA prototyping board, different software/hardware tools are used. The software tools are the MATLAB and the SDK (Software Development Kit), and the hardware tool is the Vivado from Xilinx[3]. The MATLAB software is used to compute the frequency and mean of each code of the sequences.

The Xilinx Vivado design suite is used to instantiate all required IPs for our embedded design and to build interfaces between them. It synthesizes the complete design, implements it, and generates the bit-stream to be downloaded on the FPGA for verification. The SDK is used to write the software application (sorting algorithm), in C programming language, to be run on one of the two ARM processors. It initiates the IPs and transfers the required information to/from DDR memory.

Figure 2 shows the schematic of our embedded design which contains different IP blocks:

- "DMA IP" (Direct Memory Access)
- "Processing System IP"
- "FMDS IP"

The "DMA IP" is used to transfer Data from the DDR memory to other parts of the system, and vice versa through the interface "M AXI HP0" of "Processing System IP". This will increase the data throughput and will offload the processor from tasks that involve memory transfers.

The "Processing System IP" includes the ARM processor, the processor System Reset, the DDR memory controller, and the AXI interconnects. All these components are combined together to simplify the schematic as shown in Figure 3.

The "FMDS IP" (Frequency+Mean Difference Score) includes the FDS (Frequency Difference Score) IP and the MDS (Mean Difference Score) IP. Each of these blocks is a two-input subtractor. Their outputs are summed up using two-input adder to give the Frequency+Mean Difference Score. The computed frequency and mean values of each sequence are stored in the DDR memory of the ZedBoard. The implementation starts by initiating the FMDS and DMA IPs by the ARM processor through the interfaces M00 AXI and M01 AXI, of the Processing System IP, respectively (see Figure 2). The ARM processor transfers the stream of frequency and means values from the DDR memory to the "FMDS IP" through M AXIS MM2S interface of the DMA. The "FMDS IP" computes the absolute values of the frequency difference scores using FDS IP and the mean difference scores using MDS IP.

[1] Digilent, Inc. Website: www.zedboard.org. 2016
[2] Xilinx, Inc. 7 Series FPGAs Overview. Volume 1. Number 15. 2014
[3] Xilinx, Vivado Design Suite - HLx Editions. www.xilinx.com/products/design-tools/vivado.html. 2016

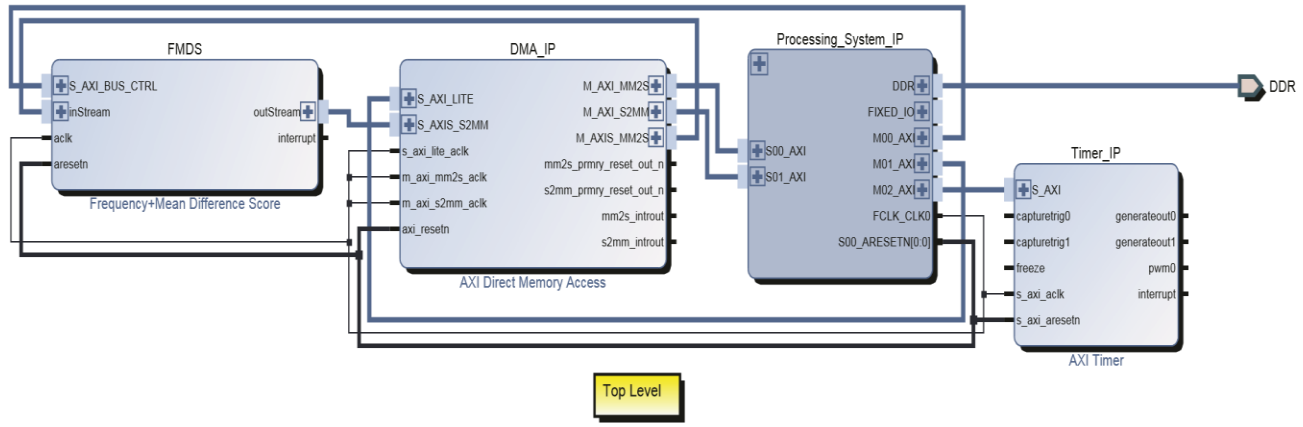

Fig. 2. Schematic of our Embedded Design Implemented in Xilinx Vivado.

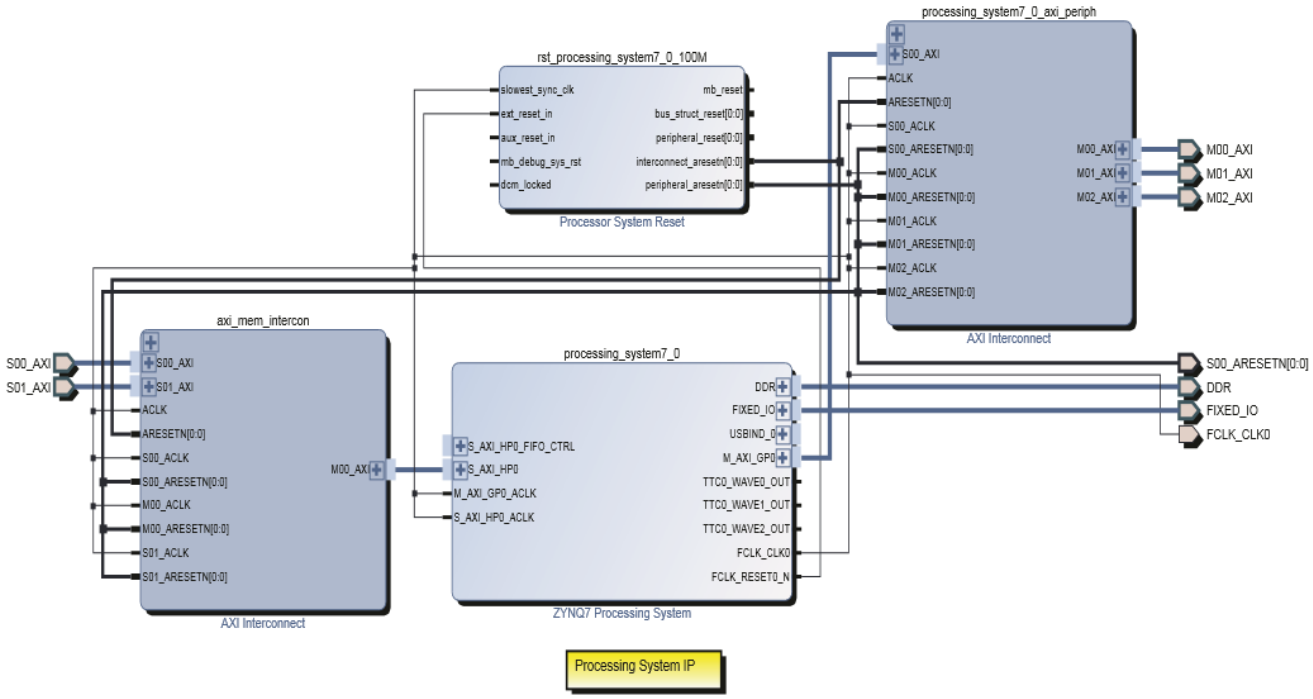

Fig. 3. Schematic of our Embedded Design Implemented in Xilinx Vivado.

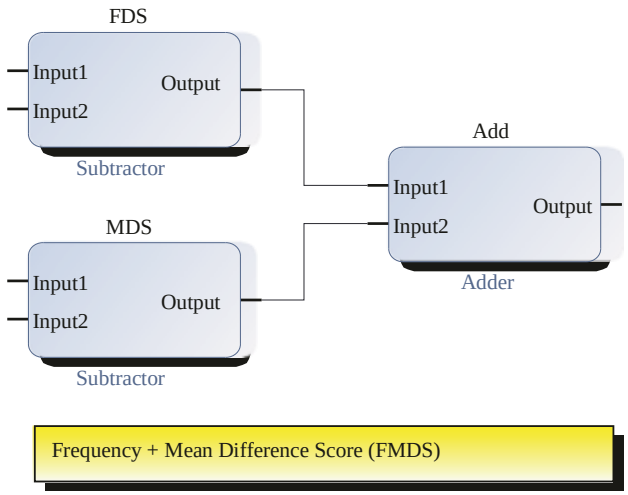

Fig. 4. Schematic of our Embedded Design Implemented in Xilinx Vivado.

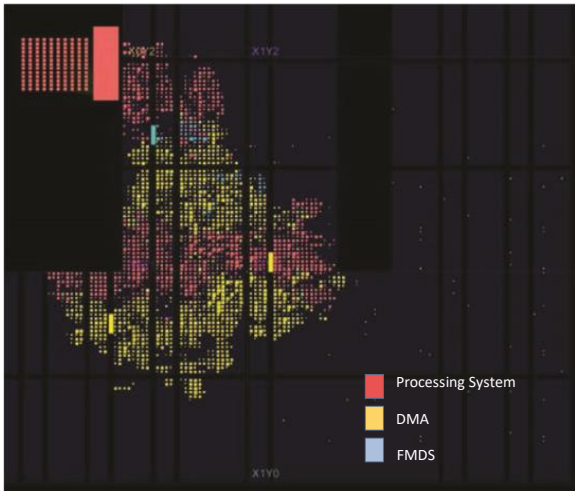

Fig. 5. FPGA Floorplan for the Embedded System Design.

TABLE I. FPGA RESOURCES UTILIZATION OF OUR EMBEDDED DESIGN

Resources	Utilization	Available	Utilization (%)
FF	7303	106400	6.86
LUT	5921	53200	11.13
Memory LUT	280	17400	1.61
BRAM	3	140	2.14
BUFG	1	32	3.12

Those scores are added to get the frequency+mean difference scores (see Figure 4). The output of the "FMDS IP" is then transferred back to the DDR memory through the M AXIS S2MM interface of the DMA. The sorting algorithm, which is running on the ARM processor, is used to sort the sequences based on their score.

B. FPGA Resources Utilization

We implement our design on a Xilinx Zynq-7000 All Programmable SoC (XC7Z020-CLG484) Artix-7 FPGA using Zedboard. All IPs are clocked using a 100 MHz frequency clock signal which is generated by the processing system. Table I summarizes the FPGA resources utilization of our embedded system. For example, less than 12% of the slice Look-up tables and less than 7% of the Flip-Flops registers are utilized. In the fact, the most resources are dominated by the processing system IP and the DMA, as shown in Figure 5. This figure shows the floorplan placement of the different IPs: the processing system IP is marked in red, the DMA in yellow and the FMDS IP in blue.

VI. EXPERIMENTAL RESULTS

In this section, we present the experimental results of our technique. For evaluation, DNA sequences of the database DNA Data Bank of Japan (ddbj)[4] were used. 100000 sequences of the length of 400 nucleotides were selected as a case study. Our technique is compared with the optimal sequence alignment Needleman-Wunsch Algorithm[5]. Random sequences were selected as a query to test our technique using different similarity functions.

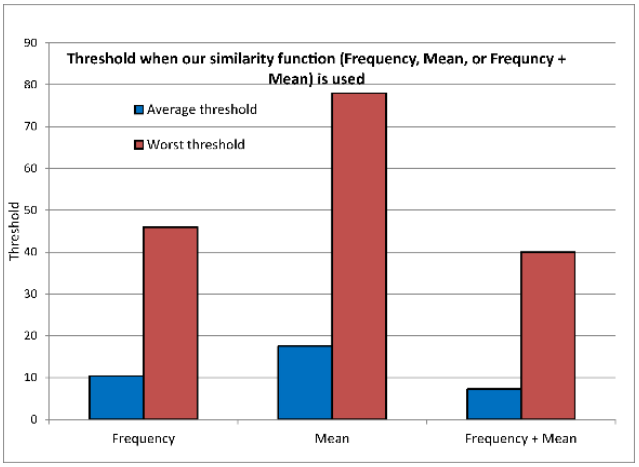

Fig. 6. Average and Worst thresholds when our Similarity Functions: Frequency, Mean, or Frequency + Mean is used.

[4] DDBJ Center, http://www.ddbj.nig.ac.jp/

5 National Center for Biotechnology Information, "Basic Local Alignment Search Tool", blast.ncbi.nlm.nih.gov/Blast.cgi

Figure 6 shows the threshold when our similarity functions, Frequency, Mean or Frequency + Mean is used. In this figure, we define a new parameter called "Threshold". The threshold refers to the number of sequences we need to apply Needleman Wunsch Algorithm on them (using our technique).

The first similarity function used in Figure 6 is "Frequency". The maximum threshold among all tested queries is 46. This is the worst case in which we need to apply the Needleman-Wunsch Algorithm on 46 sequences. The average threshold of the "Frequency" similarity function is 10.4.

When our similarity function "Mean" is used (second bars in Figure 6), the maximum threshold (worst case) and the average among all other queries is 78 and 17.5, respectively.

Looking at the obtained results, we find that using a similarity function alone (no combinations) does not yield good results. This is because two sequences are to have a close number of code frequencies (but distributed differently among both of them). Here, the FDS is not a correct measurement of similarity. Also, when two sequences have close Mean, but their frequency is different. Here, MDS is not a correct measure. Therefore, a combination of both yields better results, as shown in the third bars of Figure 6. The worst case threshold (maximum) among all the other cases in this function, and the average threshold become 40 and 7.2 respectively. Hence, the use of FMDS yields the best outcome. Or, when our technique is to be used on the best 40% sequences, applying the Needleman-Wunsch algorithm is enough to obtain a maximum alignment score. Figure 7 shows the details of distributing the threshold through all experiments when the similarity function "Frequency + Mean" is used.

As shown, in 66% of the experiments, the maximum threshold is less than 10, i.e., it exists in the top 10% of the database. It is between 10 and 20 in 15% of the experiments, while, it is between 20 and 30 in 10% of the experiments, and between 30 and 40 in the rest.

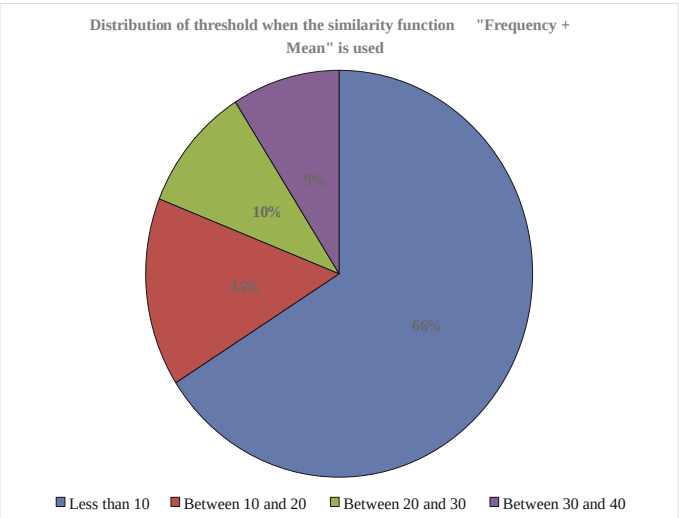

Fig. 7. Distributing the threshold through all Experiments when the Similarity Function: Frequency + Mean is used.

Fig. 8. The Error Rate Resulted from Removing Sequences from the Database.

When the technique is applied to less than 40% of sequences in the database, the result will not be correct for all experiments, because the sequence that has the lowest DS is not the sequence that has the highest similarity score. The result will be different depending on how many sequences are omitted from the database.

Figure 8 provides the rate of error that results when we omit sequences from the database. The x-axis represents the percentage of the sequences which have been removed from every database for all experiments. The y-axis represents the number of the wrong cases that result when we omit sequences from the database. For instance, when 99% of database sequences are removed, there will be 61% of wrong cases and only 39% cases with correct results. Decreasing the percentage of removed sequences decreases the error rate, and the correct cases number increases. If the percentage of the omitted sequences is 60%, and 40% of the database remains, then no wrong cases will be available. This is considered as the best case in accordance with the database size and time of execution. Omitting fewer sequences will not have an effect on the results, but will increase the size of the database, which increases the analyzation time.

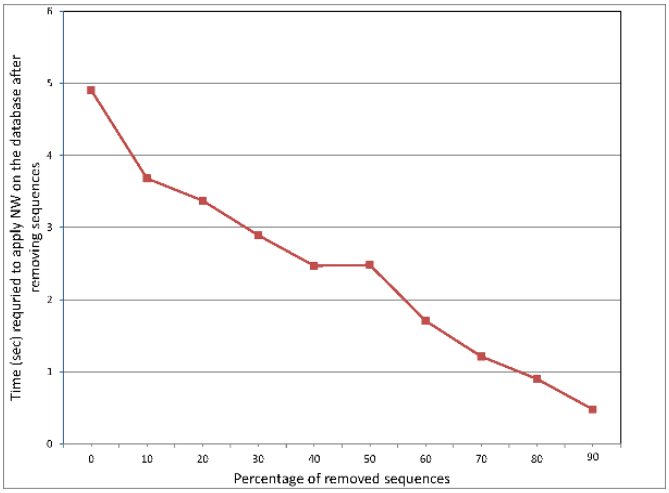

Fig. 9. The Effect of Removing the Sequences on the Execution Time.

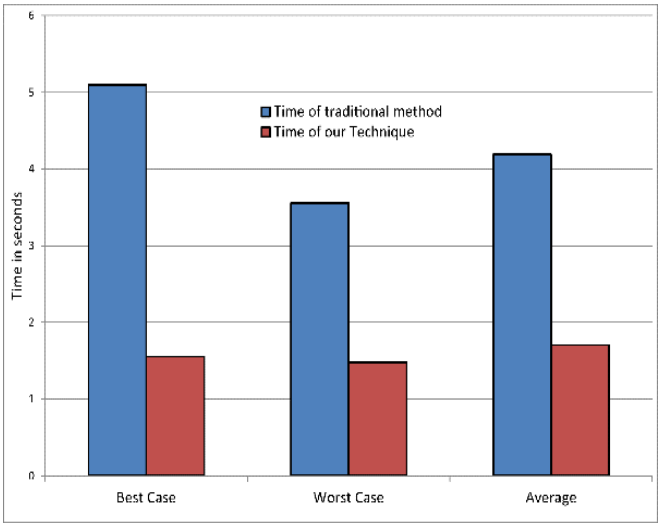

Fig. 10. Execution Time Comparison between Traditional Methods and the Proposed Technique.

Figure 9 demonstrates the impact of omitting more sequences on the time needed to obtain the highest similarity score sequences. Using traditional methods by applying the Needleman-Wunsch Algorithm on the entire sequences in the database, requires 5.9 seconds to obtain the best solution. Increasing the number of sequences omitted will not reduce execution time, however, it will increase the error rate, as seen in Figure 8.

Figure. 10 shows a comparison between the execution time of traditional methods and our technique. In this figure, The x-axis shows the best, worst and average cases through all sequences of the database. The y-axis shows the execution time in seconds. The blue bar shows the time for traditional methods while the red one shows the time for our technique which applies NW algorithm on selected 40% of the database sequences. The first bars show the best case in which the time difference is the best (70% improvement), while the worst difference is shown in the second bars (58% improvement). The third bars represent the average of all experiments. Here, the time of execution via the use of our technique is improved by 60% in respect to that of the traditional methods. (The average of the traditional methods is 4.18 seconds, while our technique has a 1.7 seconds average time.) We obtained this result because we have excluded 60% of the sequences before applying the Needleman-Wunsch Algorithm.

VII. Conclusions

The paper introduced an efficient and novel algorithm to measure the similarity between two sequences in a large database. The proposed technique computes the difference score for each sequence of the database and selects sequences that have the highest scores. Dynamic programming is then applied on selected sequences. The proposed method was implemented and tested on the HW/SW FPGA-based embedded system using the Zedboard FPGA prototyping board. The performance of the proposed method was compared with other traditional methods. The comparison study showed that our proposed method over performed other methods by 60% in term of computation time.

REFERENCES

[1] T. F. Smith and M. S. Watermann. Identification of common molecular subsequence. Journal of Molecular Biology. Pages 196197, 1981

[2] Talal Bonny, Accuracy/Speed Trade-off Technique for Dynamic Programing Based Algorithms, IEEE 5th International conference on Electronic Devices, Systems and Applications (ICEDSA), Ras Al Khaimah, United Arab Emirates. December 2016

[3] Talal Bonny, M. A. Z. and Salama, K. N. An adaptive hybrid multiprocessor technique for bioinformatics sequence alignment. In the 5th Cairo International Conference on Biomedical Engineering. pages 112115, 2010

[4] Talal Bonny and Bassel Soudan, Filtering Technique for High Speed Database Sequence Comparison, IEEE International Conference on Semantic Computing (ICSC 2015), Anaheim, California, USA. February 2015.

[5] S. Kim, Y. J. Yoo, J. So, J. G. Lee and J. Kim., Design and Implementation of Binary File Similarity Evaluation System. International Journal of Multimedia and Ubiquitous Engineering, Vol.9, No.1. Pages 1-10, 2014

[6] Jiaoyun Yang, Yun Xu, Yi Shang, Guoliang Chen. A space-bounded anytime algorithm for the multiple longest common subsequence problem. IEEE Transactions on Knowledge and Data Engineering, 2014.

[7] B. Halpin and T. W. Chan. Class Careers as Sequences: An Optimal Matching Analysis of Work-Life Histories. European Sociological Review 14(2). Pages 111-30, 1998

[8] L. Lesnard. Optimal Matching And Social Sciences. Working Paper, Centre de Recherche en Economie et Statistique. Institut Nationale de la Statistique et des Etudes Economiques, Paris, France. 2006

[9] G. Pollock. Holistic Trajectories: A Study of Combined Employment, Housing and Family Careers by Using Multiple-Sequence Analysis. Journal of the Royal Statistical Society: Series A 170(1). Pages:167-83, 2007

[10] P. F. Marteau. Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching. IEEE Transactions on Pattern Analysis and Machine Intelligence. Pages 306-318. 2008

[11] Manal Al Ghamdi and Yoshihiko Gotoh. Alignment of nearly-repetitive contents in a video stream with manifold embedding. In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Pages 1255-1259, 2014

[12] Talal Bonny, "Performance Optimization of the Database Sequencing Applications", International Journal of Computer Applications**,** volume 112 - number 5, February 2015.

[13] Talal Bonny, "A Hybrid Heuristic/Deterministic Dynamic Programing Technique for Fast Sequence Alignment", International Journal of Advanced Computer Science and Applications**,** volume 6 - issue 8, August 2015.

[14] Z. Nawaz, M. Nadeem, J. van Someren, and K.L.M. Bertels. A parallel fpga design of the smith-waterman traceback. In Field-Programmable Technology (FPT), 2010 International Conference on, pages 454-459, Beijing, China, December 2010.

[15] M. Affan Zidan, T. B. and Salama, K. N. High performance technique for database applications using a hybrid gpu/cpu platform. IEEE/ACM 21st Great Lake Symposium on VLSI. pages 8590, 2011

[16] E. F. de O.Sandes and A.C.M.A. de Melo. Retrieving smith-waterman alignments with optimizations for megabase biological sequences using gpu. Parallel and Dis- tributed Systems, IEEE Transactions on, 24(5):1009-1021, 2013.

[17] A. Chakraborty and S. Bandyopadhyay. Clustering of web sessions by FOGSAA. In IEEE Recent Advances in Intelligent Computational Systems (RAICS). Pages 282-287. 2013

[18] Xin Chang, Fernando A. Escobar, Carlos Valderrama, Vincent Robert. Optimization strategies for Smith-Waterman algorithm on FPGA platform. International Conference on Computational Science and Computational Intelligence (CSCI), 2014.

[19] Kratika Garg, Yan Lin Aung, Siew-Kei Lam, Thambipillai Srikanthan. Modelsim simulation for real-time stereo matching using DP algorithm. The 28th IEEE International Conference on System-on-Chip (SOCC), 2015.

[20] Bartolomeo Stellato, Paul J. Goulart. Real-time FPGA implementation of direct MPC for power electronics. IEEE 55th Conference on Decision and Control (CDC), 2016.

[21] S. Needleman and C. A. Wunsch. General method applicable to the search for similarities in the amino acid sequence of two sequences. Journal of Molecular Biology. Pages 443453, 1970