Massively Parallelized DNA Motif Search on FPGA

Yasmeen Farouk[1], Tarek ElDeeb[2] and Hossam Faheem[1]
[1]*Faculty of Computer and Information Sciences, Ain Shams University,*
[2]*Faculty of Engineering, Cairo University,*
Egypt

1. Introduction

Understanding the mechanisms that regulate gene expression is a major challenge in biology. Motif finding problem is considered an important task in this challenge. Addressing the complexity nature of the problem together with being very data intensive has encouraged introducing field programmable gate arrays (FPGAs) to the problem. FPGAs are very powerful in such computationally intensive tasks.

Many Algorithms are introduced to solve this problem. They can be categorized into pattern-based and profile-based algorithms [1]. Pattern-based algorithms include PROJECTION[4], MULTIPROFILER[6], and MITRA[3]. Profile-based algorithms includes CONSENSUS[7], MEME[2] and Gibbs sampling[5]. Although these algorithms show good performance, they still can fail to identify all the possible motifs in the sequences. They also show poor performance when trying to solve the challenge problem presented by Pvzner and Sze[8]. Some of them fail due to local search, others which are based on statistical measures fail to separate the motif from the background sequences.

We can also categorize Motif finding algorithms due to the solution they provide. Some algorithms provide exact solution others provide approximate one. Brute Force algorithm is an exact algorithm but it suffers from the intractability of its running time. It increases exponentially with the size of the required motif. This makes the Brute Force unsuitable for long motifs.

Our enhanced Brute Force algorithm, skip Brute Force, can predict the quality of the computed motif. The algorithm skips those iterations which will lead to a poor scored motif, thus leads to a better running time than the original Brute Force. This enhancement guarantees the same exactness of the Brute Force. But, it still suffers from the intractable running time for long motifs.

Many approaches can be applied to speed up the running time of any algorithm using hardware; examples include chip multiprocessors, graphics processing units (GPUs) and (FPGAs). GPUs are inexpensive, commodity parallel devices and have already been employed as powerful coprocessors for a large number of applications. However, GPUs have limited instructions and limited parallelism relative to FPGA's configurability. The research in [10] employed acceleration using GPU. Another approach uses clusters of workstations [12]. However, clusters typically have high maintenance and energy costs when compared to single node solutions. Others use special hardware [9][11], where a cost performance ratio would be fairer for comparison [9].

The repetitive nature of the algorithm and the locality of the data encourage the use of FPGAs. Many operations can be done concurrently to enhance the running time. FPGAs proved to successfully accelerate sequential algorithms minimum by one or two orders of magnitude. They also have been widely used to accelerate bioinformatics problems such as Smith-Waterman and BLAST algorithms. This research offers an enhanced Brute Force algorithm hardware accelerated using Field Programmable Gate Arrays (FPGAs). Our research leads to a speed up by 1.5MX and thus boosting the running time without sacrificing the accuracy.

The rest of this chapter is organized as follows: In Section 2 we describe the motif finding problem and presents our enhanced Brute Force algorithm; skip Brute Force. Section 3 presents the hardware implementation of our novel approach with a detailed view to its components. Performance evaluation is presented in section 4. Finally, section 5 concludes our work and presents future enhancements.

2. Skip brute force algorithm

Brute-force search or exhaustive search, also known as generate and test, is a very general problem solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.

The motif finding problem can be summarized as follows:

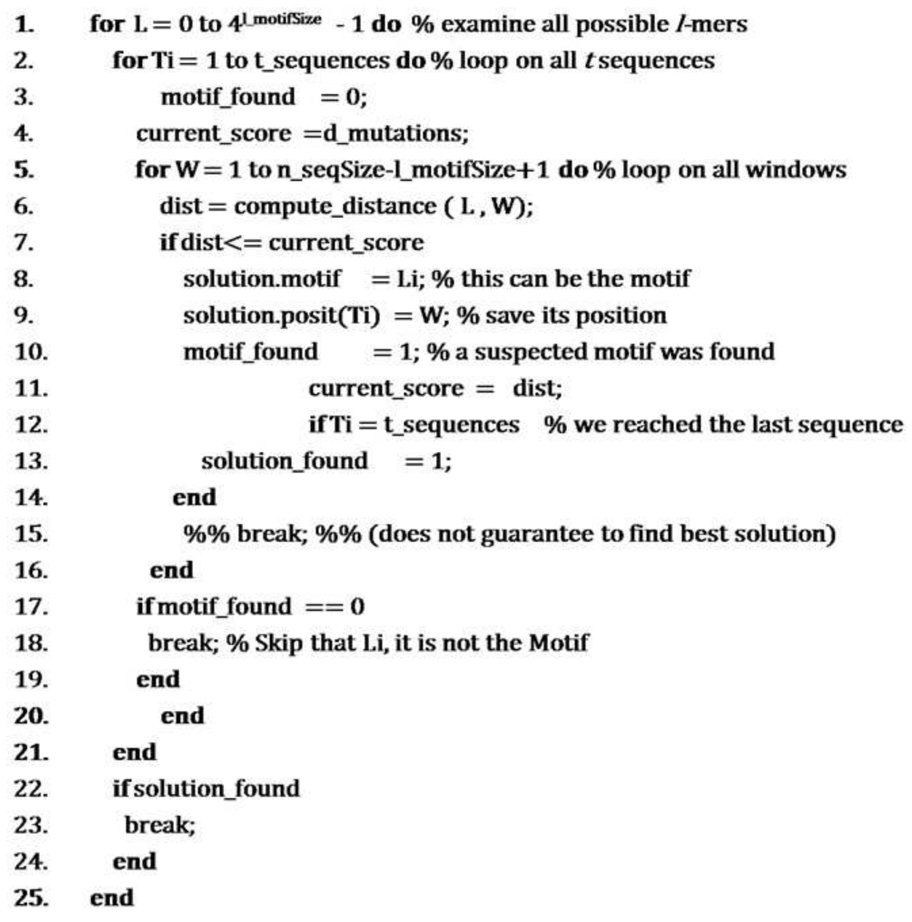
Planted *(l,d)*- Motif Problem: Find the motif consensus *M* which is a fixed but unknown nucleotide sequence of length *l*. Suppose that *M* occurs once in each of *t* background sequences of common length *n*. Each occurrence of *M* is mutated by exactly *d* point substitutions in positions chosen independently at random. Given the *t* sequences, recover the motif occurrences and the consensus *M*.

Pevzner and Sze[8] presented the challenge problem(15,4) which makes a particular parameterization to the panted motif problem. The motif we are searching for is of length l=15, the allowed mutations d=4 and the number of sequences we are searching in is t=20 each of size n=600. The parameters of the challenge problem are typical values for finding transcription factor binding sites in co-regulated gene promoter regions yeast [4].

The Brute Force algorithm solves the motif finding problem by considering the set of all 4^l possible l-mers. It computes the total distance of each l-mer in that set to all other l-mers in all t sequences. The correct motif is the one that have the smallest total distance along all the other l-mers. The run time of this algorithm is $O(4^l nt)$. The running time for finding a motif of l=11 is about 5hrs and it fails to handle longer motifs in reasonable time. To solve the challenge problem, the running time of the Brute Force algorithm would obviously be too slow.

The idea behind our skip Brute Force algorithm is that it skips all the iterations that will not lead to a correct solution. The algorithm is forced to skip over the remaining iterations in two cases. The algorithm generates all possible 4^l l-mers. It then iterates over all the sequences examining that generated l-mer with all the windows in each sequence. For each sequence iteration, the current score is initialized with the allowed mutation and then the score of each window is computed; i.e. the hamming distance between that window and the current l-mer. If this distance beats the current score then we would suspect the current window to be an implanted motif until another window in the same sequence with a higher score beats it.

The planted motif problem guarantees to find the motif in each sequence. Based on this fact the skip algorithm skips the iterations over the remaining sequences if it reached the end of the current sequence without finding any window that matches the current *l*-mer (this *l*-mer can not be the motif) and jumps to the next *l*-mer. Assuming a single solution, the algorithm also skips the iterations over the remaining *l*-mers if it reaches the last sequence (*t*=20) without skipping any iteration (the solution is found).
A pseudo code of the skip Brute Force algorithm is shown below in Figure 1.



```
1.   for L = 0 to $4^{l\_motifSize}$ - 1 do % examine all possible l-mers
2.     for Ti = 1 to t_sequences do % loop on all t sequences
3.         motif_found   = 0;
4.       current_score =d_mutations;
5.       for W = 1 to n_seqSize-l_motifSize+1 do % loop on all windows
6.         dist = compute_distance ( L , W);
7.         if dist<= current_score
8.           solution.motif    = Li; % this can be the motif
9.           solution.posit(Ti)  = W; % save its position
10.          motif_found      = 1; % a suspected motif was found
11.                    current_score =  dist;
12.                    if Ti = t_sequences   % we reached the last sequence
13.            solution_found    = 1;
14.          end
15.          %% break; %% (does not guarantee to find best solution)
16.        end
17.      if motif_found  == 0
18.        break; % Skip that Li, it is not the Motif
19.      end
20.        end
21.    end
22.    if solution_found
23.      break;
24.    end
25.  end
```

Fig. 1. Pseudo Code of the skip Brute Force Algorithm. If the commented break command is applied, then algorithm will skip-more.

2.1 Skip-more brute force

In our early implementation of the skip algorithm, we did not consider scores for the motifs found. We forced to skip the current sequence if a single motif is found that has *d* mutations within the allowed range (line 15). Here the algorithm fails to find the best motif as more windows in the current sequence can reveal occurrences of motifs with lower mutations.
The complexity of this algorithm is $O(4^l nt)$ at its worst case, just as the Brute Force.

3. Hardware implementation of skip brute force

Our design benefits from the concurrent nature of the FPGAs as a hardware platform; control, multiplexing, matching and decision making are all occurring on the same clock edge. We used VHDL to model our design preserving its extendibility for more complex challenging problems in future. Figure 2 shows the system block diagram.

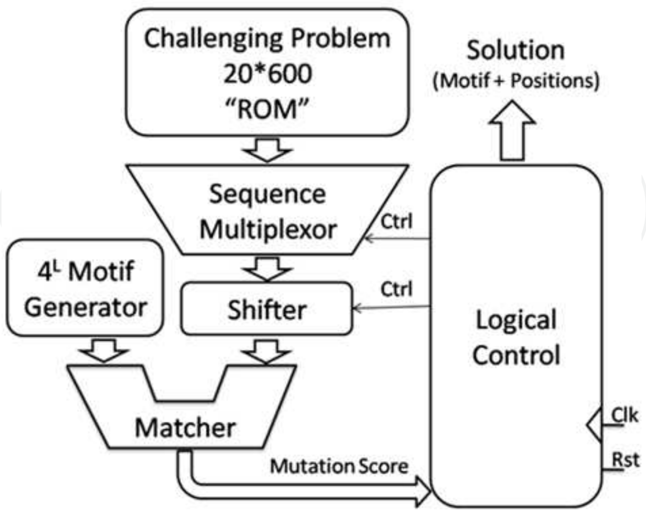


Fig. 2. Block diagram of the skip Brute Force - running on an FPGA with one matching unit.

All *t*-sequences are first loaded into an on-chip read-only memory 'ROM' as shown in Figure 3. On the contrary, the set of all 4^l *l*-mers are not stored, but locally generated. Gaining from encoding each nucleotide into 2-bit symbol, the 4^l Motif Generator is a simple controlled binary counter. The shifter block is fed by the currently needed sequence and only reveals a sliding *l*-sized window of it at a time. The matching block compares the revealed window to the generated *l*-mer and outputs the hamming distance as the mutation score. The logical control unit synchronizes the system to properly implement the skip Brute Force algorithm. More details are found in the following subsections.

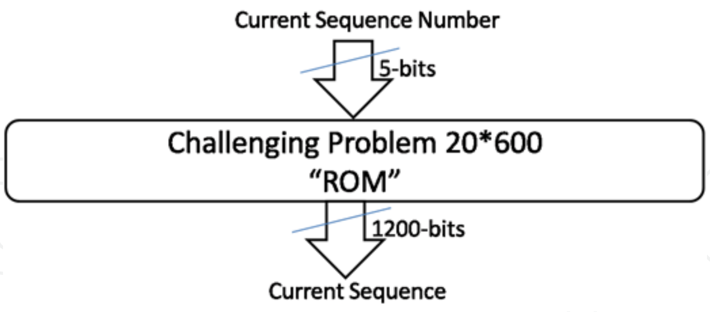


Fig. 3. The ROM Block holding the challenging problem sequence.

3.1 Sequence multiplexor

The sequence multiplexor gets one sequence at a time. The Logical control issues the signal to the multiplexor to load the sequence from the ROM and feed the shifter.

3.2 Sequence shifter

The sequence shifter block has the following inputs: clk, reset and the sequence to be shifted. The shifter outputs an *l*-sized motif each clock cycle through a windowing approach.

The shifter outputs (n-l+1) motifs for each sequence unless it is interrupted by resetting it. Our skip Brute Force resets the shifter in one case; when the shifter has generated all the (n-l+1) l-mers for this sequence. The shifter is reset to be fed with new sequence to generate the newly suspected motifs (l-mers) from this sequence.

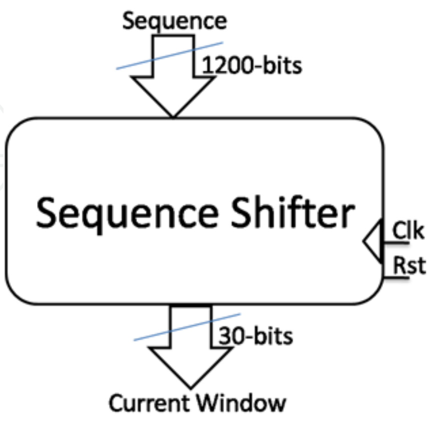


Fig. 4. Sequence Shifter block diagram.

The skip-more algorithm resets the shifter in two cases. The first case is the one previously explained. The second case happens when the matching unit finds an l to be within the d allowed mutations. In this case the system resets the shifter as the motif is considered to be found. Block diagram of Sequence Shifter is shown in Figure 4.

3.3 Motif generator

The set of all 4^l l-mers starting with AA ... A to TT ...T is not stored in the system. The four DNA nucleotides {A,C,G,T} are easily encoded into the 2-bit symbols 00,01,10 and 11 respectively. The system locally generates all the possible l-mers by a simple controlled binary counter of size l bits.

That is, in a system with l=3 we would like to generate AAA, AAC, AAG, AAT, ...,TTT. According to the encoding mentioned above; we would like to generate a series of 6-bits each as follows 000000, 000001, 000010, 000011, ..., 111111. The relation between these encoded bits can be obtained by a simple binary counter of size l bits.

3.4 Matching block

The matching block consists of many sub-blocks; xoring units, an l-bit adder and a comparison block. The matching block takes two l-sized sequences and compares them. If the difference between the two sequences is less than or equal to the allowed mutation (the two sequences have less than or equal to d different nucleotides), it outputs a match signal.

The matching block uses a series of xoring gates to determine if two l nucletoids are identical. The l-bit adder is used to count the differences between them. Finally, a comparison block is used to compare the value obtained from the adder with the d allowed mutation.

The matching block also outputs the score of the matching process. This score is used by the logical control to determine the quality of the motif obtained. The Matching block diagram is shown in Figure 5. Detailed Matching block diagram is shown in Figure 6.

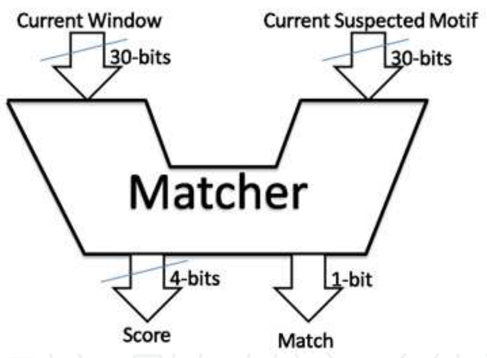


Fig. 5. Matching block diagram.

Our design is meant to be extendible by instantiating more of the matching units. Thus, its circuit implementation has to be highly optimized. Classical hamming distance circuits start with an array of XOR gates to determine matching nucleotides, followed by *l* sequential adders to compute the required distance. This approach leads to long circuit delays that will cause the system maximum frequency to drop, degrading the performance.
Our design replaces the sequential adders with a specially designed adders tree. For the (15, 4) problem, the proposed design shortens the critical path from fifteen 4-bit adders to only four full adders and two half adders. Figure 7 shows the optimized adder tree.

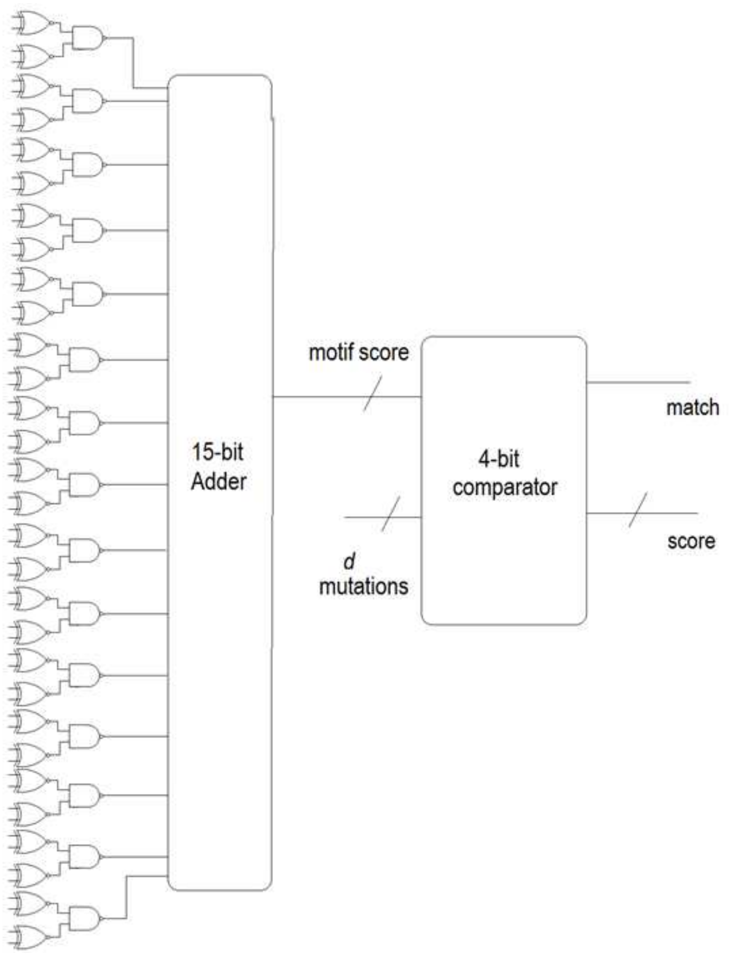


Fig. 6. Matching block components - xoring units are double the size of the motif.

3.5 Adder tree

The *l*-bit adder takes a pattern of size *l*, calculates the number of ones in this pattern and outputs the count in a $\log_2 l$ bits. For *l*=15, the adder would accept a 15 bit input signal and ouputs a 4-bit output signal. A 15-bit input signals needs five full adders; this would be stage 0. Stage 0 outputs 5 sum signals and 5 carry signals. Stage 1 needs 1 full adder and 1 half-adder for the output sum signals and the same for the output carry signals. Accordingly, stage 2 needs only 4 half adders, stage 3 needs 2 full adder and stage 4 needs 1 half adder. The final stage needs 1 full adder.

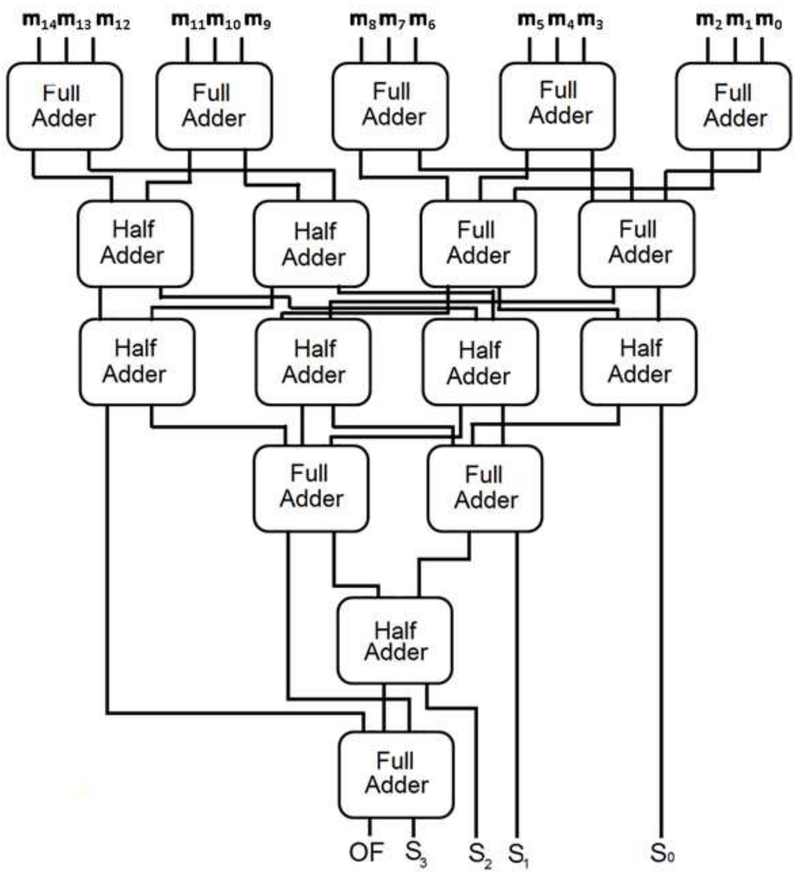


Fig. 7. The six stages adder tree - The critical path involves 4 full adders and 2 half adders.

3.6 Logical control

The system is managed by the logical control. Reset signals are issued to the motif generator and to the sequence shifter to control the flow of the sequences to be compared. As explained earlier, the logical control issues this signal under certain events. The logical control outputs the best motif which is determined by the scoring function.

3.7 Multiple matching units

It is clear that scaling up the design by utilizing more matching units in parallel will speed up the overall performance by the factor of extra units. Slight modifications and some logic duplication will be introduced for proper functionality and synchronization. The only limiting factor to the performance boost is the FPGA resources.

Figure 8 shows the block diagram of the skip Brute force running on an FPGA with multiple matching units. All *t* sequences are also loaded into an on-chip read-only memory ROM as the previous architecture. The sequence multiplexor feeds *n* series of sequence shifter followed by a matching unit. The matching unit takes its two *l*-sized sequences one from the shifter and the other from the logical unit which contains the motif generator. The outputs of the matching unit in each series are ANDed to determine the value of solution found. The number of the series of sequence shifter followed by matching unit is equal to *n*, where *n* is the number of the examined sequences. In the previous architecture, the system has to loop over all the sequences for each generated motif. This corresponds to $n.t.4^l$ loops. In this enhanced architecture, the system loops only $n \,.\, 4^l$.

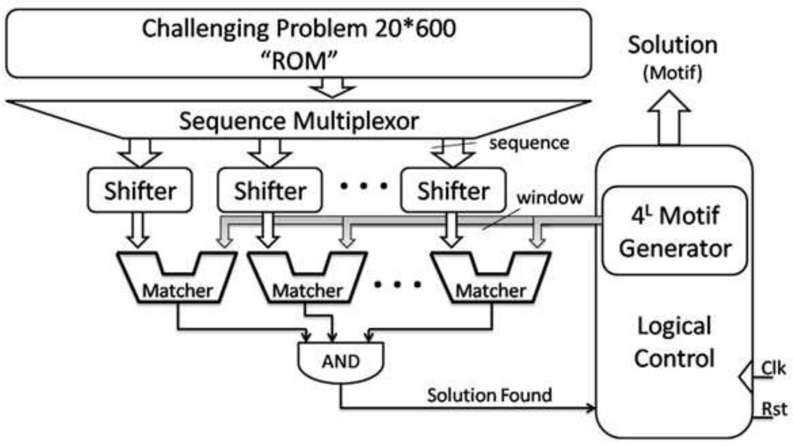


Fig. 8. Block diagram of the skip Brute Force - running on an FPGA with multiple matching units.

4. Performance evaluation and results

We tested the performances of Brute Force algorithm and skip Brute Force on synthetic problem instances generated according to the planted (*l*,*d*)-motif model. We followed the FM model described by Pvzner and Sze [8] to generate synthetic data to test our work. We produced problem instances as follows:

First, a motif consensus M of length *l* is chosen by picking *l* bases at random. Second, *t*= 20 occurrences of the motif are created by randomly choosing *d* positions per occurrence (without replacement) and mutating the base at each chosen position to a different, randomly chosen base. Third, we construct *t* background sequences of length *n*=600 using *n***t* bases chosen at random. Finally, we assign each motif occurrence to a random position in a background sequence, one occurrence per sequence. All random choices are made uniformly and independently with equal base frequencies.

The skip Brute Force achieves an average speedup of 9.11X. Both Brute Force and skip Brute Force algorithms were modelled and implemented on MatlabR2006b[15]. All the experiments ran on an AMD 5500 X2+ processor with 2GB RAM. For fair comparison, it is reported in literature that the Matlab platform is about 5 to 6 times slower than an optimized C coded program.

To evaluate the hardware implementation; we need to define the expected number of matching operations. First, we define the probability to find a random *l*-mer in a given sequence with up to *d* mutations as:

$$P_d = \sum_{i=0}^{d} \binom{l}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{l-i}$$

Additionally, we define the expected number of required matching operations to find the correct implanted motif as:

$$E(l,d) = \frac{4^l}{2}(n-l+1)(1+\sum_{i=1}^{t-1} P_d^i)$$

We then deduce for a problem of size *n*=600, *t*=20, the expected matching operations to be as shown in table 1.

L	D	Expected Matching Operations
9	2	7.7699×10^7
11	3	1.2388×10^9
12	3	4.9428×10^9
13	4	1.9750×10^{10}
14	4	7.8813×10^{10}
15	4	3.1464×10^{11}
17	5	5.0170×10^{12}

Table 1. Expected matching operations for different (*l*,*d*) problems.

We synthesized our design for multiple matching units (MU). Synthesis results of one, five, ten and twenty matching units need further analysis. Figure 9 shows the area utilization of the FPGA. The FPGA utilization increases almost linearly with increasing the number of MUs.

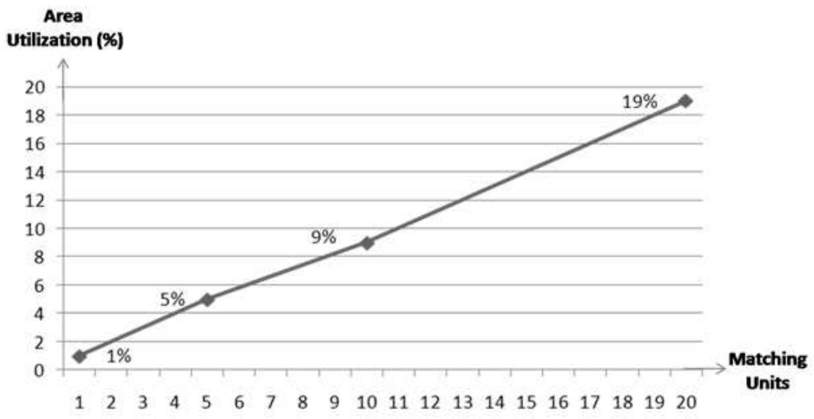


Fig. 9. FPGA area utilization - increases almost linearly.

The design of multiple MUs inherits parallelization; this means the system critical path remains the same even after increasing the number of MUs. Unfortunately, the system maximum frequency decreases with increasing the number of MUs. This is due to the increased complexity of the FPGA interconnects. Over 80% of transistors inside the FPGA are dedicated to the programmable routing network as programmable switches and buffers. The increased complexity of the interconnects leads to FPGA resource starvation.

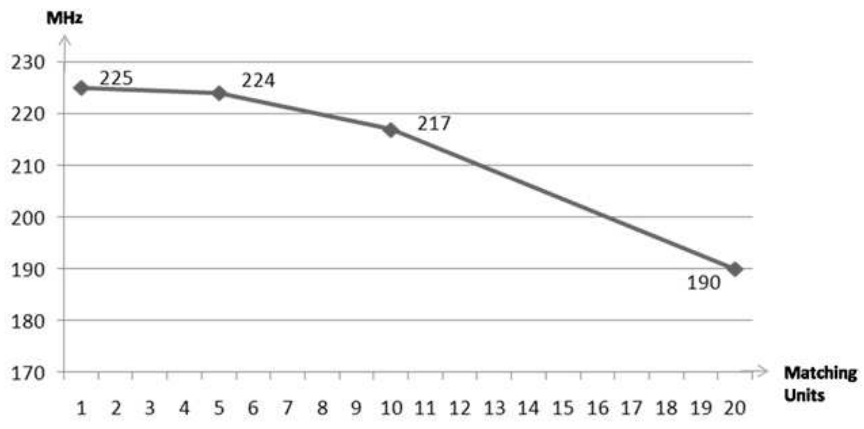


Fig. 10. Maximum system frequency - decreases due to interconnects complexity.

Furthermore, It is well known that interconnects in FPGA dominate the system performance and power consumption.
Depending on the architecture, 60% to 80% of the FPGA critical path delay is due to the routing between logic blocks. Long interconnects exhibit a substantial delay and often lead to timing violation and require further optimizations. In a recent study [13], it was found that FPGA interconnects is poorly scaled. Based on the extrapolation of future device performance, interconnects will become the performance bottleneck, of which the clock rate will be slowed down to 17 MHz in a 13 nm process. Figure 10 shows degradation in the maximum frequency of the system with increasing the number of matching units.
We define the system throughput as the number of matching operations per second. Figure 11 shows the curve of the system throughput. The throughput increases by increasing the number of MUs. The curve tends to be linear but the degradation in the maximum frequency alters this linearity.

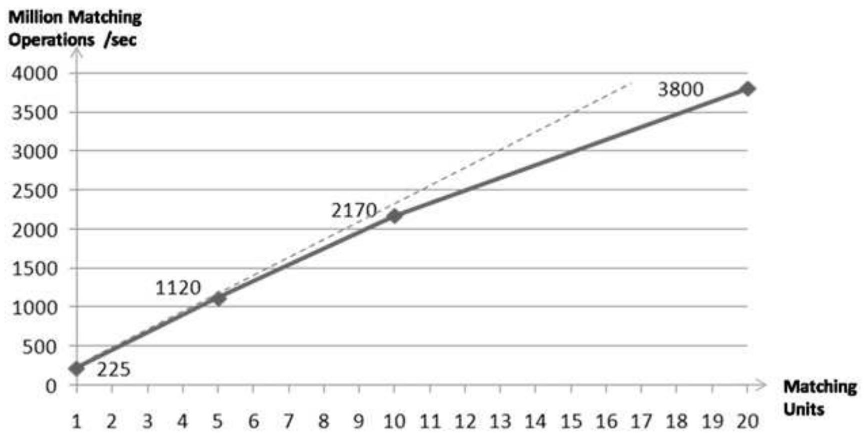


Fig. 11. System throughput - increases almost linearly.

Figure 12 compares the running time of Brute Force, skip Brute Force, skip Brute Force running on FPGA with one matching unit and with 20 matching units of different challenge problems. The running time of Brute Force in all challenge problems is the highest. Our skip Brute Force algorithm running on an FPGA has the best running time.

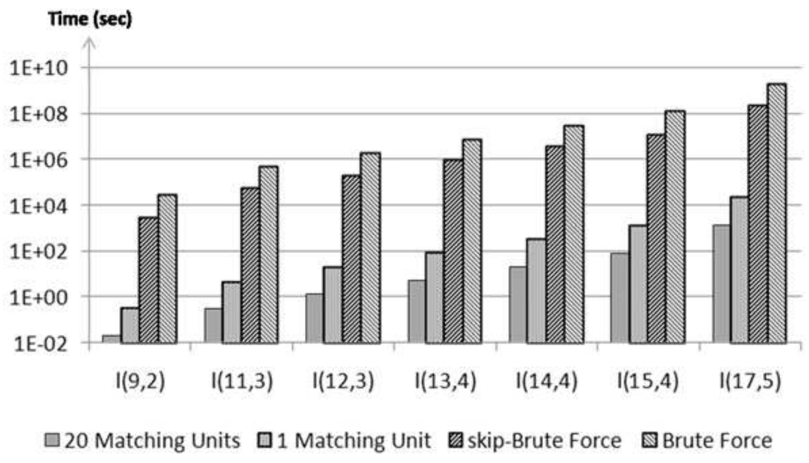


Fig. 12. Running time of various challenge problems - skip Brute Force running on an FPGA based architecture with 20 matching units has the fastest running time.

Utilizing one matching unit leads to a speedup by 9800X over pure software running time of skip Brute Force. It is clear that scaling up the design by utilizing more matching units in parallel will speed up the overall performance nearly by the factor of extra units. We used 20 matching units and achieved a speed up factor 16.88X over one matching unit.
Thus, applying the skip Brute Force (9.11X) on 20 matching units (16.88X) running on an FPGA-based architecture (9800X) would offer 1.5MX boosting in the performance.
Figure 13 illustrates these observations.

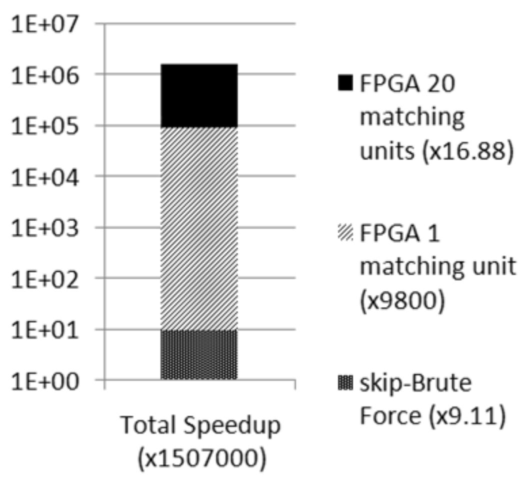


Fig. 13. Speedup factors of our accelerating designs - Total speedup is 1.5MX.

RTL synthesis and Place and route were accomplished using Quartus tool on the Stratix III FPGA technology, a product from Altera[14]. The skip Brute Force FPGA design does not use any of the FPGA memory blocks. The PowerPlay tool showed a total of power consumption of 400mW.

5. Conclusion and future work

This chapter presents a proof-of-concept parallization of motif finding on FPGA to achieve high performance at low cost. Among all Motif Finding Algorithms, Brute Force is known to be the most accurate. This is mainly because it searches the space of all possible motifs. The major drawback of Brute Force is the intractability of its running time. The algorithm running time grows exponentially with the length of the motif. This makes the Brute Force unsuitable for long motifs. The algorithm can not be used to solve the (15,4) challenge problem in a reasonable time.

In order to find the correct solution for the planted motif problem; we have to over-come two main problems. We have to be able to identify the motif from background sequences by applying an exact algorithm such as the Brute Force that guarantees to always find the correct motif. We also have to overcome its running time and memory complexities through acceleration by enhancement in the algorithm itself and by hardware implementation. Our research presented here addresses these two issues.

We presented an enhanced Brute Force algorithm; skip Brute Force, which can predict the quality of the obtained motif. The algorithm skips those iterations which will lead to a poor scored motif, thus leads to a better running time. This enhancement guarantees the same exactness of the Brute Force. Our enhanced algorithm showed a speedup factor of average 9.11X.

The repetitive nature of the algorithm and the locality of the data encourage the use of FPGAs. Many operations can be done concurrently to enhance the running time. FPGAs proved to successfully accelerate sequential algorithms minimum by one or two orders of magnitude. They also have been widely used to accelerate bioinformatics problems such as Smith-Waterman and BLAST algorithms. This research offers an enhanced Brute Force algorithm hardware accelerated using Field Programmable Gate Arrays (FPGAs).

We designed an FPGA-based architecture to accelerate our skip Brute Force algorithm. The core of the skip Brute Force algorithm is its matching unit. Utilizing one matching unit leads to a speedup by 9800X over pure software running time of skip Brute Force. It is clear that scaling up the design by utilizing more matching units in parallel will speed up the overall performance nearly by the factor of extra units. We used 20 matching units and achieved a speed up factor 16.88X over one matching unit.

Thus, applying the skip Brute Force (9.11X) on 20 matching units (16.88X) running on an FPGA-based architecture (9800X) would offer 1.5MX boosting in the performance.

Obviously, the real boosting in the performance (9800X) is achieved by introducing FPGA to the algorithm. It is neither the effect of enhancing the Brute Force algorithm, nor the effect of applying more matching units.

Many motif finding algorithms achieves better running time on the expense of the motif accuracy obtained. We succeeded to accelerate the motif finding problem without sacrificing the accuracy by applying an exact algorithm; skip Brute Force.

Our work can be extended to accelerate other motif finding algorithms that have shown better performance to solve the motif finding problem. Algorithms such as Projection [4]

and MEME [2] proved to have high accuracy and much better running time. Introducing these algorithms to hardware acceleration will offer more boosting to its running time.
An embedded processor can be added on the FPGA to run the algorithm on chip. This approach will eliminate the communication overheads which is the bottleneck in most hardware-software co-designs.
Furthermore, our approach can be applied to other biological applications. One of the most important problems in the biological research is the tertiary structure prediction of a protein using amino acid information. This is particularly important in the context of designer proteins in the area of drug discovery. Graph analysis of biological networks is also computationally intensive.

6. References

[1] Rajasekaran, S., Balla, S. and Huang, C.H.: Exact algorithm for planted motif challenge problems, Proceedings of Asia-Pacific Bioinformatics Conference, 249–259 (2005)
[2] Bailey, T.L., Williams, N., Misleh, C., Li, W.W.: MEME: discovering and analyzing DNA and protein motifs. Nucleic Acid Research 34, W369–W373 (2006)
[3] A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen, Predicting gene regulatory elements in silico on a genomic scale, Genome Research 15, 1202-1215(1998)
[4] Buhler, J., Tompa, M.: Finding motifs using random projections. J. Comput. Biol. 9, 225–242 (2002)
[5] Lawrence, C.E., Altschul, S.F., Boguski, M.S., Liu, J.S., Neuwald, A.F., Wootton, J.C.: Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. Science 262, 208–214 (1993)
[6] E. Eskin and P. Pevzner, Finding composite regulatory patterns in DNA sequences, Bioinformatics S1, 354-363(2002)
[7] Hertz, G., Stormo, G.: Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. Bioinformatics 15(7-8), 563–577 (1999)
[8] Pevzner, P., and Sze, S.-H.: Combinatorial approaches to finding subtle signals in DNA sequences. Proc. 8th Int. Conf. Intelligent Systems for Molecular Biology, 269–78(2000)
[9] Jan SchrOder, Lars Wienbrandt, Gerd Pfei□er, and Manfred Schimmler: Massively Parallelized DNA Motif Search on the Reconfigurable Hardware Platform COPACOBANA. PRIB 2008 LNBI 5265, 436-447(2008)
[10] Chen Chen, Bertil Schmidt, Liu Weiguo, and Wolfgang Müller-Wittig: GPU-MEME: Using Graphics Hardware to Accelerate Motif Finding in DNA Sequences. PRIB 2008 LNBI 5265, 448-459(2008)
[11] Sandve, G.K., Nedland, M., Syrstad, B., Eidsheim, L.A., Abul, O., Drablas, F.: Accelerating motif discovery: Motif matching on parallel hardware. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 197–206. Springer, Heidelberg (2006)
[12] Grundy, W.N., Bailey, T.L., Elkan, C.P.: ParaMEME: A parallel implementation and a web interface for a DNA and protein motif discovery tool. Computer Applications in the Biological Sciences (CABIOS) 12, 303–310 (1996)
[13] Terrence Mak The Future Looks Gloomy for FPGA Interconnects Technical Report Series NCL-EECE-MSD-TR-2009-145, 2009.

[14] Altera Inc., http://www.altera.com/
[15] Matlab Product Family. http://www.mathworks.com.