A New Scheduling Strategy for Solving the Motif Finding Problem on Heterogeneous Architectures

H. M. Faheem
Professor of Computer Systems,
Ain Shams University, Cairo, Egypt

B. König-Ries
Chair of Distributed Information Systems,
Jena University, Jena, Germany

ABSTRACT

The Motif Finding Problem is a well-known computationally intensive problem in bioinformatics. Exact solutions to finding a motif such as brute-force suffer from the intractability of their run time. To ameliorate this, different solutions relying on different hardware technologies have been proposed including the usage of FPGA or multicore architectures. Recently, deploying heterogeneous architectures that involve CPUs, GPUs, and FPGA kits seems to be very promising. The main goal is to speed up processing in order to achieve the result in a moderate time. The scheduling strategy dealing with such heterogeneity is one of the most important factors affecting the performance of the heterogeneous systems. In this paper, we introduce a new scheduling strategy that depends on the expected finish time when the problem is solved by only one type of architecture. The proposed scheduling strategy provides the fastest available processor type with a specific size of data items and instructs it to perform the parallel operations. Other slower architectures will get different sizes of data that can be processed exactly in the same time granted to the fastest architecture. This operation will be iterated until the finish of all the processing operations required to get the motif. Evaluation results of solving the motif finding problem using our proposed approach on a set of heterogeneous architectures versus the implementation on an individual architecture are compared. Results show that the new scheduling strategy yields much more advantageous results over the implementation on individual architectures. We believe that the proposed strategy is a step towards a well-defined framework for implementing run-time systems that support heterogeneous architectures.

Keywords

Motif Finding Problem; Heterogeneous architectures; Scheduling Strategies

1. INTRODUCTION

One of the most important problems in bioinformatics is the understanding of gene regulatory networks. This is due to the fact that gene regulatory networks help the researchers better identify how genes cooperate to perform functions, how any species respond to diseases or environmental insults, and how organisms are affected by genes disorders. In this context, finding regulatory elements, especially the binding sites for transcription factors is crucial and constitutes a major challenge. The binding sites for expressed genes are called motifs. In the DNA sequence, a motif is usually a short segment that occurs frequently, but is not required to be an exact copy for each occurrence. This property of motifs makes motif finding very difficult. Despite considerable efforts to date, finding these motifs remains a complex challenge for biologists and computer scientists.

The Motif Finding Problem (MFP) can be simply considered as a string matching problem. Solving the MFP to find a motif of length L with permitted mutation d can be implemented using a brute-force algorithm. All the possible *L-mers* (4^L) are compared with each possible motif of length *L*. If we have a sequence of size N then we can have *(N-L+1)* motifs. Pevzner and Sze [1] presented the challenge problem (15,4) and (16,4) where the first number is a specific length *L* and the second number a specific mutation d. In this paper, we present a problem in which the motif has a length *L=16*, allowed mutations *d=4*, and the number of sequences we are searching in is *T=20* each of size *N=600*.

Solving such computationally intensive problems can be implemented using a set of heterogeneous platforms [2, 3, 4, 5, 6, 7, 8]. Currently, almost all our home PCs have heterogeneous architectures since they have at least CPUs and GPUs. In fact, multicore architectures are now playing an important role in solving complex problems. Moreover, FPGA cards could be also deployed in our home environment to add more computing power to our PCs.

The main concern of such heterogeneity is related to the scheduling strategy needed to benefit from the capabilities of such architectures. Recent scheduling strategies for heterogeneous architectures include heterogeneous earliest finish time (HEFT) [9], and predict earliest finish time (PEFT) [10]. HEFT is performed in two phases. The first phase is intended to prioritize the tasks while the second phase assigns tasks to the workers based on its priorities. PEFT depends on an Optimistic Cost Table OCT that is used to rank tasks and for processor selection.

HEFT and PEFT are thinking of the scheduling problem from the absolute scheduling mechanism point of view without considering other factors that may affect the implementation. For example, when implementing an algorithm using CUDA paradigm on NVidia GPUs we have to consider the shared memory usage, register usage, and thread block size. Also, we have to carefully consider the data movement mechanism in order to always process local data not to move it among blocks.

In this paper, we think of the scheduling problem from a different point of view. We will consider the speed differences in architectures in solving a problem such that the faster the architecture is, the larger the number of chunks of tasks assigned to it. A chunk here means a complete unit of work that we can predict the execution time of on a specific architecture. We will assume that the tasks of each chunk assigned to a specific architecture will be executed by this architecture only. In doing so, we eliminate the factors that may affect the overall system performance such as shared memory, registers, etc. This is due to the fact that we can use

specific parallel computing paradigms that can handle these issues efficiently when proper code is developed.

The rest of this paper is organized as follows: Section II describes the task dependency analysis of the MFP. Section III illustrates the proposed scheduling strategy. Section IV presents the environment used for the implementation and provides the evaluation results. Section V presents the conclusion and directions for future work.

2. TASK DEPENDENCY ANALYSIS OF MOTIF FINDING PROBLEM

MFP has a stereotypical type of comparison operations. The process starts with data expansion such that a total number of *N–L+1* windows each of length *L* are derived from a sequence of size *N*. This operation is sequential in nature such that a sliding window technique with a one position shift is required to obtain each window. To implement the expansion operation in parallel we can work on all the available sequences concurrently such that we can extract the first window W_1 from each sequence. If we have T sequences then we can obtain *T* windows (W_1 of each sequence) at the first clock CLK0. A total number of *N-L+1* units of time are required to expand all the *T* sequences to get a total number of *[(N-L+1) * T]* windows each of size *L*.

In principle, parallel comparisons between all possible 4^L *L-mers* and the generated *T* windows can start as long as there are windows ready to be compared after the expansion operations. A total number of $4^L * T$ comparisons can be implemented concurrently. This means that a total number of $(4^L * T) * (N-L+1)$ can be implemented in *N-L+1* units of time. Time slots that describe the concurrent operations for both data expansion and parallel comparisons are shown in Fig.1. It is clear that the comparison operations will start at CLK_1 since T windows (W_1 of each sequence) are derived in CLK_0. Consequently, deriving *T* windows (*Wi* of each sequence) at CLK_i will be followed with comparison operations at CLK_{i+1}.

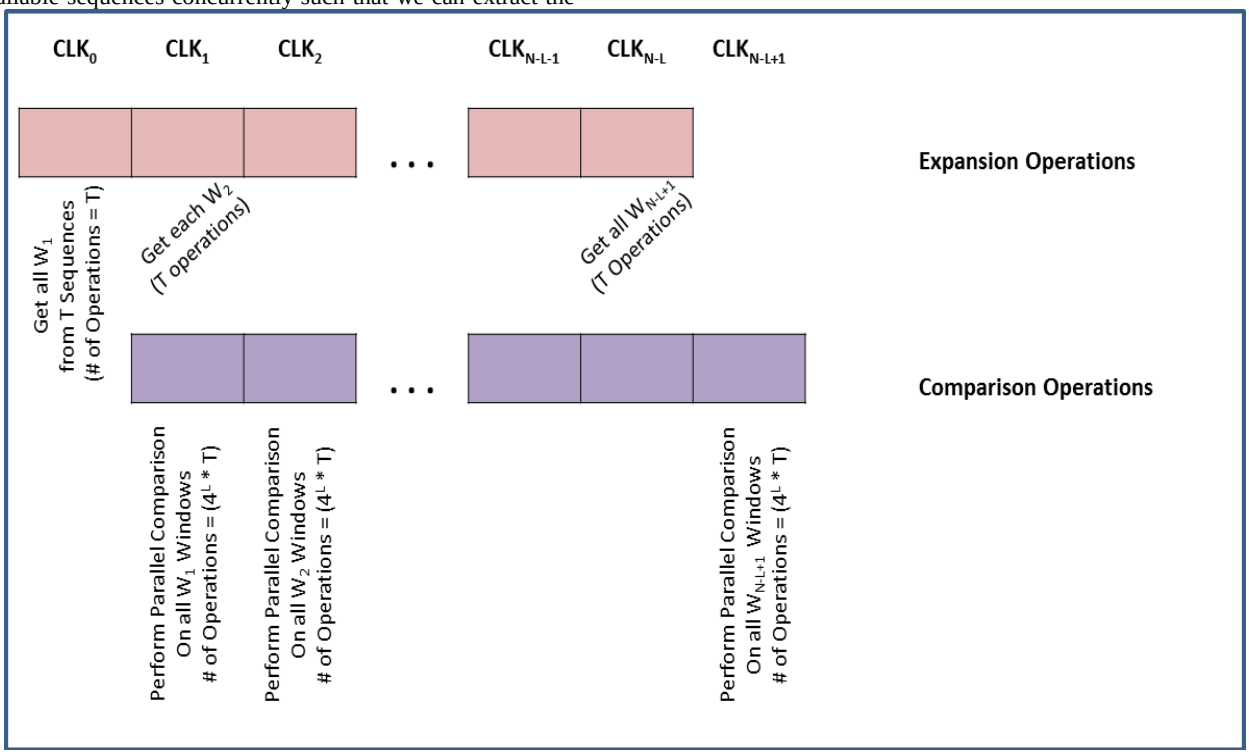

Fig.1 Data Expansion and Comparison Operations

3. TASK SCHEDULING STRATEGY

Due to the nature of the operations required to implement the MFP, a simple scheduling strategy can be suggested. The main idea benefits from the fact that the number of the operations required to solve the MFP can be divided into *C* chunks each of $((4^L * T) + T)$ operations in almost all cases except the first and the last ones which are *T* and $(4^L * T)$ respectively.

Consider that we have *p* heterogeneous architectures $(A_1, A_2, ..., A_p)$ capable of performing the operations of each chunk in different amounts of time $(t_1, t_2, ..., t_p)$ where $t_i < t_{i+1}$ then: If the total execution time required by the fastest architecture A_1 to find the motif which is approximately equal to $(t_1 * C)$ where $(C = (N–L+1)$ in case of MFP) is greater than t_i (for *i=2* to *p*) then allow A_i to perform the operations for a number of chunks that is relative to its execution time with respect to t_1. The pseudo code for the proposed scheduling strategy is shown in Fig.2.

The *distribute_chunks()* function initially decides the architectures that will take part in solving the problem. This function distributes the chunks proportionally according to the execution time of the architectures. Of course, the architecture with less execution time for the chunk will get more chunks to perform. The function eventually returns with the number of chunks individually assigned to the architectures.

Pseudo Code for the Scheduling Strategy

Input ($t_1, t_2, \ldots, t_p$) ; The execution times for performing the operations of a
; single chunk on $A_1, A_2, \ldots, A_p$ respectively in ascending order

Input (C) ; The number of chunks $C = N-L+1$ in case of MFP

Output ($C_1, C_2, \ldots, C_p$) ; The number of chunks assigned to each architecture

Function *Distribute_Chunks*(t_i, C)

For each i:=1 to p
if (($t_1 * C$) > t_i) ; for each architecture satisfies this condition where
; ($2 \leq i \leq p$) Decide which architectures will be eligible
; to perform operations on chunks
then $R_i := C / t_i$; find the weight of each eligible architecture
else $R_i := 0$; or exclude ineligible architecture
End

$R := R_1 + R_2 + \ldots + R_p$; find the total weights

$R_u := C / R$; find the unit assigned for each weight

For each i:=1 to P

$C_i := R_i * R_u$; assign number of chunks C_i to each eligible A_i
End

Return ($C_1, C_2, \ldots, C_p$)

End

Fig.2 Scheduling Strategy for Solving MFP

4. IMPLEMENTATION

In order to evaluate the proposed strategy we have worked on the MFP (16, 4). The block diagram of the used system is shown in Fig.3 while the architectures used are listed in Table 1. We have deployed a PC machine (core i5, 2.3 GHz, 4 GB RAM) having both an FPGA card (Stratix III FPGA) interfaced with it and a GPU (NVidia GT240, 96 cores, 1.34 GHz core clock). In this case, we have three types of architectures CPU, GPU, and FPGA. CUDA SDK4.0 is used to solve the MFP on the GPU while MPICH2 for x64 running on 2 nodes (cores) is examined on the CPU. The FPGA design used in [11] was selected to be investigated since we have developed its VHDL code. The block diagram of the FPGA system used is shown in Fig. 4. The FPGA has a ROM that holds the T sequences under investigation. The Sequence multiplexor selects the sequence Ti from the set of sequences T. The shifter is responsible for selecting windows starting from W1 and ending at W_{N-L+1} from a given sequence. The motif generator is responsible for generating all the possible 4^L *L-mers* each of size *L*. The matching unit is responsible for comparing the window W provided by the shifter with the *L-mer* generated by the motif generator. A single matching unit is used in this block diagram while a modified version of this architecture utilizes 20 matching units to accelerate the comparison process. The FPGA with multiple matching units is shown in Fig.5.

Table 1 Hardware Configuration

Architecture	Vendor	Model
CPU	Intel	Core i5, 2.3 GHz
GPGPU	NVidia	GT240, 96 Cores, 1.34 GHz core clock
FPGA	Altera	Stratix III

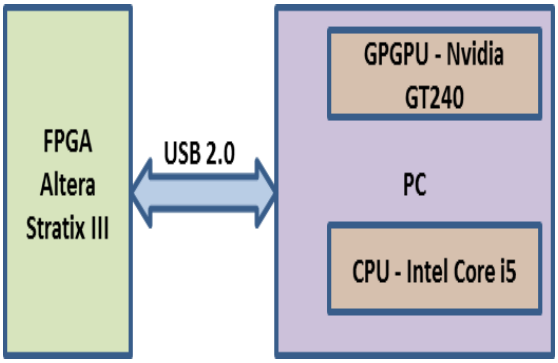

Fig. 3 Block Diagram of the system consisting of heterogeneous architectures (CPU, GPGPU, and FPGA)

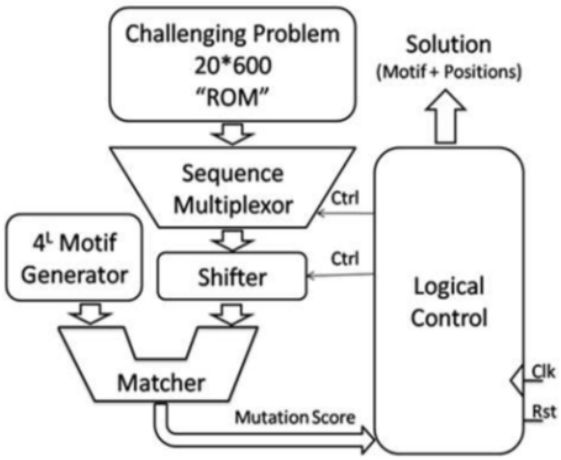

Fig. 4 Block diagram of the Brute Force - running on an FPGA with one matching unit

When implementing the scheduling strategy using the FPGA architecture with 20 matching units, the execution time of the chunk is (approximately 23 seconds). Consequently, when applying the scheduling strategy a new chunk distribution will take place. This is shown in Table 5. A total execution time of 12,489 sec. is achieved (3.46 hours approximately). Relative speed up when solving the MFP on heterogeneous architectures as compared to the individual architectures is shown in Fig. 6 and Fig. 7 for (CPU, GPU, and FPGA(1)) and (CPU, GPU, and FPGA(20)) respectively.

5. CONCLUSION

A scheduling strategy for solving MFP has been introduced. The strategy can be used to assign tasks to heterogeneous architectures in a way that ensures different load balancing depending on both the computing power and the paradigm used on the architecture. Mainly, the execution times required for the architectures will be used to decide the number of chunks assigned to each one. Almost 53% reduction in the execution time as compared to pure FPGA implementation has been achieved when a single matching unit is used in the FPGA. Also, 7% reduction in the execution time has been achieved when compared to pure FPGA implementation with 20 matching units. In principle, having small gaps between the speeds of architectures will lead to significant enhancement in the execution time. The enhancement in execution time is inversely proportional with the speed gaps between architectures. We believe that the proposed strategy is a step towards much more advanced strategies for task scheduling intended for heterogeneous architectures.

Initially, we have to determine the execution time required by an individual architecture to perform a chunk of comparison operations. Results of performing (4^{16} * 20 Operations) on different architectures are listed in Table 2.

Since we have *N-L+1* chunks then we have *C=585*. Applying the rules of scheduling strategy yields the number of chunks assigned to each architecture results shown in Table 3. A total execution time of 92,610 seconds (25.7 hours approximately) is achieved. This of course provides a significant improvement when compared with the total execution time of an individual architecture as shown in Table 4.

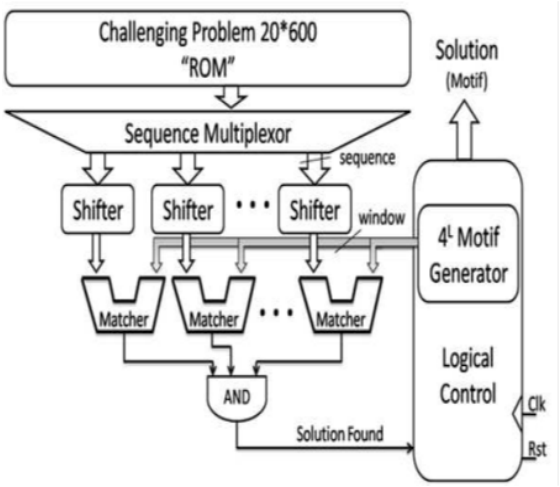

Fig.5 Block diagram of the Brute Force – running on an FPGA with multiple matching units

Table 2 Execution time of performing a chunk of comparison operations

Architecture	Execution Time (sec.) for a chunk (4^{16} *20 Operations)
FPGA	343
GPU (CUDA)	400
CPU (MPICH2)	1100

Table 3 Chunks assigned to different Architectures

Architecture	No. of Chunks Assigned
FPGA(1)	270
GPU	231
CPU	84

Table 4 Execution Time

Architecture	Individual Execution Time (Sec.)
FPGA(1)	200,655
GPU	234,000
CPU	643,500
Heterogeneous	92,610

Table 5 Chunks assigned to different Architectures (FPGA has 20 matching units)

Architecture	No. of Chunks Assigned
FPGA(20)	543
GPU	31
CPU	11

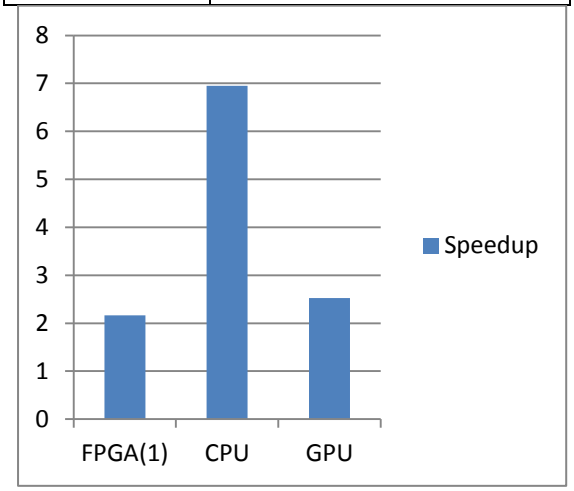

Fig. 6 Relative speedup when solving the MFP using heterogeneous architectures (FPGA with 1 matching unit, CPU, and GPGPU)

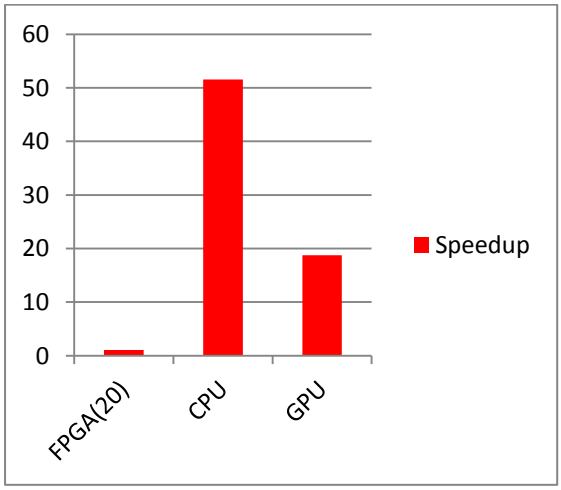

Fig. 7 Relative speedup when solving the MFP using heterogeneous architectures (FPGA with 20 matching unit, CPU, and GPGPU)

6. REFERENCES

[1] P. Pevzner and S. Sze, " Combinatorial approaches to finding subtle signals in DNA sequences," Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology, 269–78, 2000.

[2] H. M. Faheem, "Accelerating Motif Finding Problem using Grid Computing with Enhanced Brute Force," The 12th International Conference on Advanced Communication Technology (ICACT), Korea, 2010.

[3] M. Raddad, N. El-Fishawi, and H. M. Faheem, "Implementation of Recursive Brute Force for Solving Motif Finding Problem on Multi-Core," International Journal of Systems Biology and Biomedical Technologies, 2 (3):1-18, 2013.

[4] R. Inta and D. J. Bowman, "An FPGA/GPU/CPU hybrid platform for solving hard computational problems," in Proceedings of the eResearch Australasia, Gold Coast, Australia, 2010.

[5] S. J. Park, D. R. Shires, and B. J. Henz, "Coprocessor computing with FPGA and GPU," in Proceedings of the Department of Defense High Performance ComputingModernization Program:Users Group Conference—Solving the Hard Problems, pp. 366–370, Seattle,Wash, USA, 2008.

[6] M. Showerman, J. Enos, A. Pant et al., "QP: a heterogeneous multi-accelerator cluster," in Proceedings of the 10th LCI International Conference on High-Performance Cluster Computing, vol. 7800, pp. 1–8, Boulder, Colo, USA, 2009.

[7] D. B. Thomas, L. Howes, andW. Luk, "A comparison of CPUs, GPUs, FPGAs, and massively processor arrays for random number generation," in Proceedings of the 7th ACM SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '09), pp. 63–72, Monterey, Calif, USA, 2009.

[8] K. Underwood, "FPGAs vs. CPUs: trends in peak floatingpoint performance," in Proceedings of the 12th International Symposium on Field-Programmable Gate Arrays (FPGA '04), pp. 171–180, New York, NY, USA, February 2004.

[9] H. Topcuoglu, S. Hariri and M. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing", IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260-274, 2002.

[10] L.C. Canon, E. Jeannot, R. Sakellariou and W. Zheng, "Comparative evaluation of the robustness of dag scheduling heuristics", in Sergei Gorlatch, Paraskevi Fragopoulou and Thierry Priol, Grid Computing - Achievements and Prospects, pp. 73-84, Springer, 2008.

[11] Y. Farouk, T. El-Deeb, and H. M. Faheem, "Massively Parallelized DNA Motif Search on FPGA," Bioinformatics - Trends and Methodologies, INTECH, 2011