Accelerating Motif Finding Problem Using Skip Brute-Force on CPUs and GPU's Architectures

M. M. Al-Qutt*[γ], H. Khaled*, Rania ElGohary[β], H. M. Faheem*, Iyad Katib[¥], Nayif Al-Johani[¥]
*Computer Systems Department, Faculty of Computer & Information Science, Ain Shams University, Cairo 11566, Egypt, Email: mmalqutt@cis.asu.edu.eg, heba.khaled@cis.asu.edu.eg, hmfaheem@cis.asu.edu.eg
[β]Information Systems Department, Faculty of Computer & Information Science, Ain Shams University, Cairo 11566, Egypt, Email: dr.raniaelgohary@fcis.asu.edu.eg
[¥]King Abdulaziz University, [γ] Corresponding Author

Abstract - - *Motif Finding Problem (MFP) aims to discover unknown motifs that are expected to be common in a set of sequences. MFP is considered as one of the most computationally intensive problems in the field of bioinformatics which requires a large amount of memory and it categorized as Nondeterministic Polynomial time (NP-complete) problem. Both software and hardware accelerators have been proposed and implemented to accelerate MFP algorithms. In this paper, SKIP brute-force algorithm is accelerated using different High Performance Computing (HPC) paradigms such as CPUs, CUDA GPUs. Different sequence lengths are used to conduct the experiments on 2500 CUDA core on TESLA K20 kepler GPU ACCELERATOR and Intel(R) Xeon(R) CPU E5-2695 v2. The results show that CUDA implementation outperformed Intel implementation, execution time is reduced.*

Keywords: Motif Finding Problem (MFP), High Performance Computing, Power Consumption, GPU, CUDA.

1 Introduction

DNA sequences implicate genes, its main function is to decode genetic information in human body and living creatures, Motifs are short patterns consisting of nucleotide it appears close to these genes. They are repeated in the sequence with mutations in several of their nucleotide locations. [1,2]. MFP concerned about finding patterns in DNA sequences, the found patterns in DNA sequences are very useful for genetics related researches because they highlight the sequences that take the control to activate specific genes [3].

The bioinformatics related problems in general are well known as computationally intensive problems, this embraces MFP [4]. To clarify the complexity of MFP we consider the following example, if we are searching for a motif of length (L) then there are (4^L) possible motif with 4 mutations (M) accepted and searching in (T) sequences each of length (N) then there are (N-L+1) possible window in each sequence. Using an exact searching algorithm such as Brute-Force algorithm requires (4^L*(N-L+1)*T) comparisons, running time of such an algorithm is O (4^L N T) and it will take almost 70 days to produce results on a duel core, 2.5GHz and 2 GB RAM machine[1]. All known exact search algorithms used for solving the MFP searches all locations for all possible L-mer so they have an exponential worst case runtime [2][3]. One intractable instance of planted MFP is the (15,4) motif problem which known as searching for the motif of length 15 with 4 mutations in random set of DNA sequences each is a 600 nucleotide long. This problem was raised by Pevzner [4]. Despite of the intensive expected computational effort, the nature of MFP has a high potential for parallelization; MFP requires concurrent independent set of comparisons as same as most of the DNA sequences operations [5, 6].

The power of high performance computing techniques can be used and utilized to speed up the solution of MFP by considering adequate distributed memory utilization combined with a good implementation of parallel algorithm. The contribution in this paper concentrates on the implementation of different homogeneous parallel paradigms with several hardware architectures: CPU Multi-core and GPU, compare the results based on both speed up and power consumption.

Recently multicore CPUs are being used to speed-up the implementation of many applications that require an extensive data processing, one major example of these applications is string matching, and it has gained significant throughput [7]. Accordingly, it is worthwhile to use of multicore CPUs to speedup MFP and should accomplish a privileged performance compared to the use of a grid network.

GPUs were originally used to support graphics and then they are becoming increasingly popular for different types of applications for the goal of achieving high performance. The Advances of Computer Unified Device Architecture (CUDA) and the simplicity of programming for GPU were factors for GPU spread. A GPU is a massively parallel, multi-threaded, many-core processor with hundreds of cores and extensive computational capabilities. CUDA processors are designed around a fully programmable scalable processor array, organized into a number of streaming multiprocessors. The

wide spread of GPUs encouraged the development of multiple discrete independent software systems based on Multi-Agent systems [8-10]

This paper exploits different homogeneous parallel paradigms applied for "SKIP brute Force" Algorithm proposed by Faheem [7]; for solving planted MFP (15,4): the designated architecture are (MPI + OpenMP) for multi-core system 16x24 XEON dual physical processor, each has 12 core. The second is (CUDA) for GPU-based architecture TESLA K20 Kepler GPU ACCELERATOR, it has 13 Multiprocessors each of 192 CUDA Cores. The experiments are conducted on random-generated sequences that have two varying parameters: number of sequences (T) and sequence length (N). Number of sequences varies from 20 to 240 while the sequence length varies from 100 to 1200. Moreover this paper implements an existing algorithm and many researchers exploited multi-core and GPU based architectures, the main contribution of this paper is the study of running-time, number of comparisons and power consumption against both number of computational resources and sequence lengths. This study could be a step toward a MFP recommender system of hybrid heterogeneous solutions for computational resources management bounded by running time and power consumption. The rest of this paper is organized as follows Section 2 illustrates the related work, section 3 shows the implemented systems which is consists of two systems the first is based on MPI and Open MP and the second is based on CUDA. Section 4 illustrates the comparison of both the performance and power consumption of the implemented systems. Finally, the conclusion is summarized and future work is proposed.

2 Problem Definition

Due to the importance of MFP, many researchers paid attention to explore different solutions for it. Solution approaches can be categorized from different point of views; some approaches find exact motifs[7,11,12,13] such as Brute Force, others find approximate motifs [4, 14, 15] such as probabilistic approaches.

Algorithms for motif finding can be classified into two main categories: exact (deterministic or word-based enumerative) algorithms and non-exact (nondeterministic or probabilistic sequence models). The word-based methods depend on exhaustive counting, enumeration and comparing nucleotide frequencies. These algorithms include the brute-force, branch and bound techniques and suffix trees [13-15]. Word-based methods guarantee global optimal solution and are suitable for short motifs. Meanwhile, the probabilistic sequence models consider the parameters that are estimated using Likelihood calculation and Bayesian [16]. This type contains method such as Expectation Maximization [17], Gibbs Sampling [18] and Random Projection [19]. Probabilistic models are suitable for relatively long motifs also, this type requires fewer search. However, the probabilistic models do not guarantee the global optimal solution, but these models are significantly faster than the exact models. Another classification point of view: approaches can be either iterative or combinatorial approaches. Iterative approaches such as, Gibbs sampling and expectation maximization consider the position weight matrices. Meanwhile, combinatorial approaches such as MITRA, WINDOWER rely on hamming distances.

This paper considers an upgraded version of Brute-Force algorithm which is called "SKIP Brute-Force" (SKIP BF) Algorithm suggested by Faheem [7, 20] to be accelerated, Figure 1. The main idea of SKIP BF algorithm is based on the fact that some paths and iterations won't lead to a correct solution so SKIP BF simply skips all these irrelevant iterations.

Why SKIP BF

Skip BF obtains the same exactness of the original Brute Force with better running time by skipping irrelevant iterations. Also, it has been implemented on H/W accelerator such as FPGA and achieved a remarkable speedup. In addition to it has a high potential of parallelization due its repetitive nature [20].

```
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   = L; % 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 L, it is not the Motif
19.     end
20.       end
21.   end
22.   if solution_found
23.    break;
24.   end
25. end
```

Figure 1: SKIP Brute-Force Algorithm [17]

3 Parallel MFP and Related Work

To speedup MFP, various H/W and S/W accelerators are exploited. The repetitive nature and data locality have encouraged researchers to use (FPGA) to speedup MFP. Faheem et al. [20] have implemented Skip-Brute Force search on FPGA and have achieved speed up more than 10x over the serial version. Chen et al. [21] have accelerated "Expectation Maximization for Motif Elicitation (MEME)", which is an efficient approximate algorithm, using GPUs and have reached a significant speed up. They concluded that more speed up can be achieved by using a cluster of GPUs. Liu et al. [22] have raised this work by accelerating it in (CUDA) enabled GPUs and also using multiple GPUs. They compared

the performance with the parallel MEME running in CPU clusters and concluded that GPU based implementation is much faster than use of CPU clusters. Naga. S. D et al. [23] proposed another parallel implementation for the planted (l, d) MFP on GPU using Bit Based. They succeeded to solve the challenging instance (21,8) of planted problem in 1.1 hours. Faheem et al. [24] has proposed a heterogeneous architecture that combines GPUs, Multi-core CPUs and MICs. They proved that appropriate scheduling mechanism on heterogeneous system can significantly improve speedup.
In this paper, SKIP BF algorithm is parallelized using different homogeneous architectures: Multi-core CPUs and the Implementation were using Open MP and MPI and GPUs using CUDA.

4 Implementation

This section shows the proposed implementation of the MFP solution based on two different architectures:

- The first implementation exploits Intel Compiler 2015 and Intel MPI V5. This implementation uses the MPI with 16x24 XEON dual physical processor, each has 12 core and 24 threads per CPU with thermal design power (TDP) equals to 115 W, usually is called thermal design point, which is the maximum amount of heat generated per processor. Also, an InfiniBand network is used to connect different compute nodes.
- The second implementation uses a TESLA K20 Kepler GPU ACCELERATOR, The GPU has 13 Multiprocessors each of 192 CUDA Cores, so it has a total 2496 processor core which offers a Total 5 GB of memory, 1310720 bytes L2 Cache Size, Total 65536 bytes of constant memory, a total 49152 bytes of shared memory per block, a total 65536 registers available per block, with maximum 2048 threads per multiprocessor and maximum 1024 threads per block, and with thermal design power (TDP) equals to 225 W.

One major concern of this work is studying the effect of the DNA sequence length on performance of the search algorithm on different parallel paradigms; all generated DNA sequences files have the same size on desk 21 Kbytes. Table 1 shows the generated DNA Sequences files where T is the Number of Sequences and N is Sequence Length.

Table 1: Generated sequences

T * N	20 * 1200	24 * 1000	30 * 800	40 * 600	60 * 400	80 * 300	120 * 200	160 * 152	240 * 100

4.1 Multi-core CPU Implementation

OpenMP is a compiler-directive-based Application Program Interface (API) that could be used to explicitly direct multithreaded, shared memory parallelism [25, 26]. OpenMP fundamental idea is the existence of multiple threads in the shared memory programming paradigm which facilitates data-shared parallel execution. OpenMP programs start with one thread "*Master thread*" and other threads *"Worker threads*". In parallel parts in code, Worker threads executes and in between the Worker threads are put to sleep.

MPI is a standard for developing multi-platform parallel applications using the message passing mechanism [25]. MPI programs work on both distributed and shared memory architectures and platform2s, offering a highly portability features of parallel programming on a various hardware topologies.

Multi-Core CPU implementation depends on a hybrid implementation using OpenMP and MPI according the following coordination:

- The main thread divides the space (all possible motifs) into subspace based on MPI Rank
- Each processor searches for its own assigned motifs subspace within all of the (T) DNA sequences using OpenMP to activate all cores.
- Each returns the motif of the maximum score
- The it uses the MPI reduction to return the overall maximum

The detailed pseudo code for the proposed implementation is illustrated in Figure 2 and the proposed Architecture is illustrated in Figure 3

```
1. PROGRAM MFP_OMP_MPI
2. Input: space(start,…,end)
3. Input: S[1…T]
4. BEGIN
5. motif ← 0
6. score_max ← 0
7. rank ← MPI_Rank
8. mpi_start ← rank*((end-start)/MPI_Size)
9. mpi_end ← (rank+1)*((end-start)/MPI_Size)
10.  $OpenMP Directive
11.  FOR ALL motif x in space [mpi_start,….,mpi_end]
12.        BEGIN
13.           score ← score (x,s)
14.           IF score > score_max THEN
15.              motif ←x
16.              Score_max ← score
17.           ENDIF
18.        END
19.  return MPI_Reduction(MAX, score_max)
20. END
```

Figure 2: MPI-OpenMP implementation of the MFP using the CPUs based architecture

4.2 GPU Implementation

CUDA is a parallel programming model that facilitates programming for scalable applications to be run on GPU. A CUDA program is formed of a sequential host code that is executed on CPU and calls to operations "kernels" that is executed on GPU. Designing a CUDA program is very a critical issue; it should be appropriately designed to take advantage of the available resources to achieve a better performance.

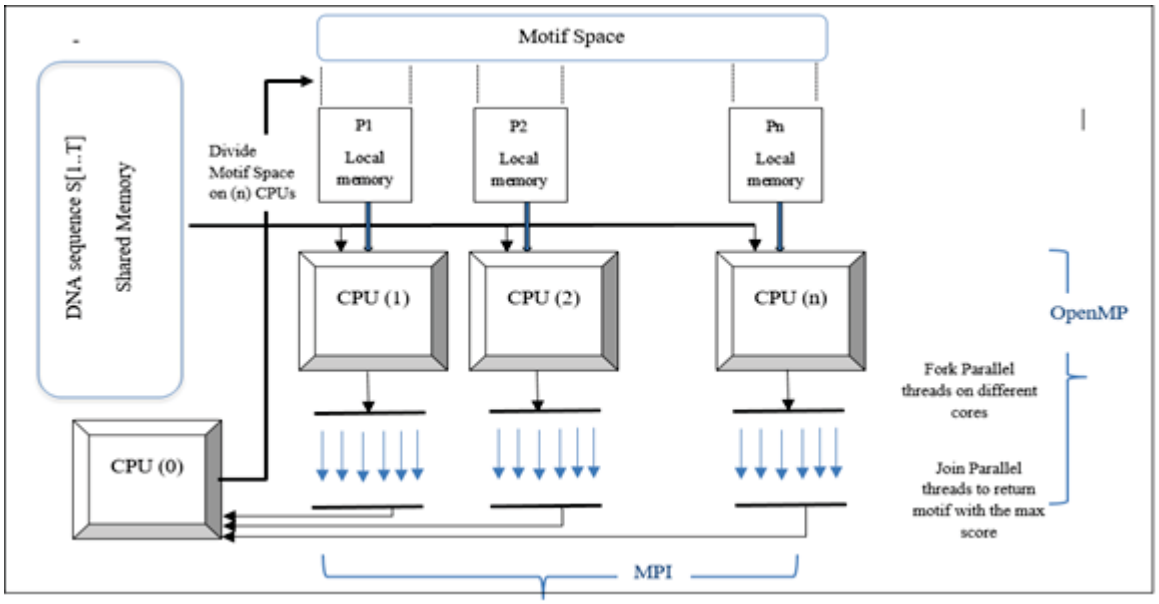

Figure 3: Multi-Core CPUs based architecture

The fact that GPU uses a Single instruction, multiple thread (SIMT) architecture concluded that the optimal results can be gained when all the threads in a warp execute without diverging. Threads are executed serially whenever they diverge causing degrading performance. Figure 4 shows a pseudo code for the proposed MFP with a CUDA implementation.

Processing all the search space (L-mer) consumes very long time causing a card time out. So, the proposed module calls the CUDA kernel multiple times. The Thread index is used to indicate which motif is looked up within all the (T) sequences. An offset is also used to differentiate the 8 chunks of the data. The scores are collected in a global array and finally, we search the array to find the maximum score. The module uses the Offload directive to run this implementation on the MIC co-processor. The coordination main idea can be summarized as following:

- The program divides all possible motifs space into 8 subspace, cause processing all the search space (L-mer) consumes very long time causing a card time out, the offset indicates the start point of each subspace
- The motifs subspace is divided into blocks each of size 512
- For each block a 512 thread will be generated, so each motif a thread will be initialized to look up this motif
- A CUDA kernel is launched like this GPUMotifKernel<numBlocks,threadsPerBlock> (S, offset, score)

The score represents the quality of the proposed motif, in other terms; it calculates the frequency of each motif, given $\mathbf{s} = (s_1, s_2 \ldots s_t)$ a set of patterns each of length (l). The score function first aligns the patterns by their start indexes "Alignment". Then, it constructs the profile matrix to record the frequency of each nucleotide in each column position. Finally, it constructs the consensus where its nucleotide in each position has the highest score in column. The score function compares different guesses and choose the best one. The best consensus is the motif with the highest score or in other terms, it is the motif with the minimum total distance given by (1).

```
1. PROGRAM MFP_GPU
2. Input: subspace(start,…end)
3. Input: S[1,…,T]
4. BEGIN
5. Threads <- 512
6. Blocks <- ceiling ((end-start)/8/threads)
7. score[start,…,end] ←0
8. FOR n 0 to 7
9.       BEGIN
10.            Offset= n * (end-start)/8
11.               GPU_MotifKernel<blocks, threads>(S,
    offset, score)
12.                  Begin
13.                       threads ← (blockIdx * 256 +
    threadIdx) + offset
14.                        score[thead]← score(thread,S)
15.                   END
16.              return Max_{i=start}^{end} (score_i)
17.            END
18. END
```

Figure 4: CUDA implementation of the MFP using the GPUs based architecture

5 Experimental Results

It is important for an unbiased comparison between Multicore CPUs and GPUs performance for this study, we must ensure to optimize the written CPU implementation to the reasonably acceptable level. The memory access should be cache-optimized as much as possible and the code should not confuse the branch predictor. The written codes

were confirmed to be optimized for both Multicores and GPUs each.

Before exploiting the experiments, we had the prejudgment that CPU will perform better with the fewer - long sequences and the GPU will perform better with the many - short sequences. This impression came from the following reasons:

- The major limitation on the GPU side is that each thread can essentially only perform the same operation at a time, just on different parts of the data set you load into memory. The threads are "dumber" compared to CPU threads.
- The memory hierarchy of both architecture where the short sequence will fit easily into the GPU cache memory

The study begins by comparing the Execution time for both architectures against (TxN where T: Number of sequences and N: Sequence Length), Figure 5.

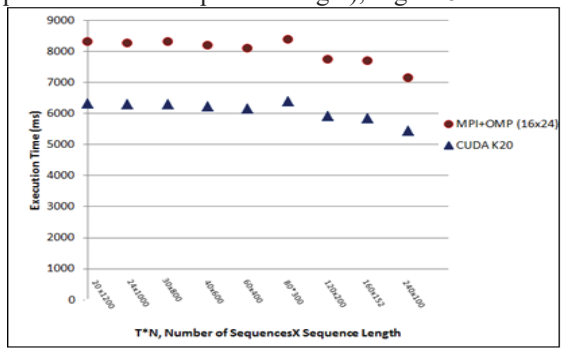

Figure 5: Execution Time for Solving MFP on different parallel paradigms

Unexpectedly GPU outperformed in both the fewer - long and the many-short sequences. These results come from the fact that with CPU threads concurrency and synchronization could slow down the running. While the GPU threads, synchronization is less of a problem, and the challenge lies in optimizing the algorithm to operate on massive arrays of data a little at a time. The detailed results are illustrated in Table 2.

Table 2: Execution Time for Solving MFP on different parallel paradigms

T*N Number of Sequences x Sequence Length	MPI+OMP (16x24) (ms)	CUDA K20 (ms)
20*1200	8308.9799	6311.1594
24*1000	8266.6473	6295.2935
30*800	8309.8822	6283.1261
40*600	8201.3555	6234.3788
60*400	8104.5923	6153.4324
80*300	8389.9444	6378.8553
120*200	7753.1210	5914.3007
160*152	7692.2951	5842.0219
240*100	7151.7948	5445.2097

Although that all files size was exactly the same, the performance varies among files. The performance on both architectures was directly proportional to the total number of comparisons needed to finish the task, and the total number of required comparisons is inversely proportional to the used sequences length as per Table 2. So, we concluded that GPUs perform better with shorter sequences as they require a fewer number of comparisons. In other words the number of comparisons required for Searching DNA sequences is function in sequence length, number of sequences and motif size Table 3. MF execution time is mainly a function in number of comparisons. It was noticed that searching short sequences requires less number of comparisons and therefore a less execution time for the same problem size as illustrated in Figure 6.

The ultimate goal for most of the researchers, scientists and engineers is to reduce the power consumption of the high performance computing architectures. One major aspect that gave the GPU privilege over CPU is the power consumed by both architectures in order to solve MFP task. Given that the thermal design power (TDP), sometimes called thermal design point, which is the maximum amount of heat generated by Intel(R) Xeon(R) CPU E5-2695 v2 is 115 W per processor then, the total power consumed to finish the motif finding problem using a Xeon dual processor of 12 cores each is (115 * 2 * 16 * Execution Time).

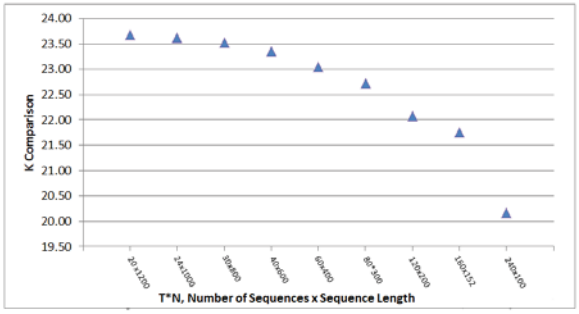

Figure 6: Number of sequences and Sequences' lengths against Number of comparisons to solve MFP.

Table 2: Number of Comparisons needed for Solving MFP for various sequence lengths.

Number of sequences T	Sequence Length N	File Size on disk [Bits]	Total Number of comparisons
20	1200	24000	23680
24	1000	24000	23616
30	800	24000	23520
40	600	24000	23360
60	400	24000	23040
80	300	24000	22720
120	200	24000	22080
160	152	24320	21760
240	100	24000	20160

Since the TDP of TESLA K20 kepler GPU accelerator is 225 W then the total power consumed by the TESLA K20 kepler GPU accelerator to finish the motif finding problem is (225 * 1 * Execution Time). Taking into consideration that the GPU is attached to a serve, we add the power consumed by the server which is (115*2* Execution Time).

Table 3 and Figure 7 show the power consumption for solving MFP by both paradigms.

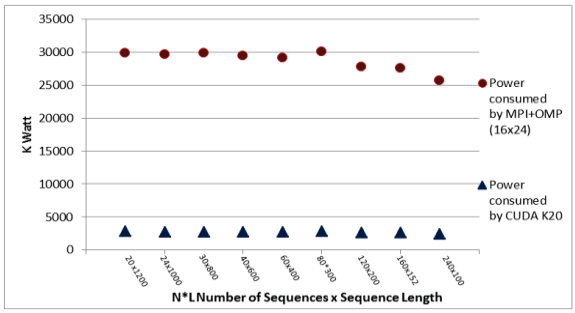

Figure 7 : Power Consumption for Solving MFP by different parallel paradigm

Table 3: MSP Power consumption for different number of comparisons on different parallel architectures

N*L	Total Number of comparisons	Power consumed by MPI+OMP (16x24) (W)	Power consumed by CUDA K20 (W)
20*1200	23.68	29860.4	2804.275
24*1000	23.62	29708.26	2797.225
30*800	23.52	29863.64	2791.819
40*600	23.36	29473.62	2770.159
60*400	23.04	29125.88	2734.191
80*300	22.72	30151.36	2834.355
120*200	22.08	27862.78	2627.936
160*152	21.76	27644.19	2595.82
240*100	20.16	25701.76	2419.502

Another more accurate model to be used in future that calculates the total power consumed by any system (P) that simply sum up the power consumed by each device in this system, for example considering a system that has n devices then the total power consumed by the system could be calculated as follows [27, 28]:

$$P = \sum_{i=1}^{n} (P_{dev(i)})$$

Where $P_{dev(i)}$ is the power of the i^{th} device

The device power referees to static power and dynamic power, however during the execution the dynamic power may rise and fall irregularly according to the type of the operation performed by the device during the execution. The execution of any task may involve the following operations initialization, computation, communication, and output. The initialization and output phases have a little effect on the overall performance for large-scale problems that is solved in multiple nodes. Accordingly the power consumption will be during the computation phase which refers to active status or otherwise all sort of communication which refers to idle status. The device power may be obtained using the weighted sum of the power consumed during each execution phase:

$$P_{dev} = P_{active} * \frac{T_{active}}{T} + P_{idle} * \frac{T_{idle}}{T}$$

Where the total execution time $T = T_{active} + T_{idle}$

6 Conclusion

In this paper a multicore CPU based and GPU based methods to accelerate MFP using Skip-Brute Force algorithm are implemented. The parallel versions of the algorithm were implemented and run in both multicore Intel(R) Xeon(R) CPU E5-2695 v2 architectures and TESLA K20 kepler GPU ACCELERATOR. The results showed that GPU significantly reduced the execution time and improved the performance better than using Multicore architecture. Also, data files structure explicitly affects the performance of the different parallel paradigms. Hence, the searching algorithm is faster with data file that has short sequences as it requires less number of comparisons. The proposed implementation required codes optimization for unbiased comparisons. In future, this work could be extended by proposing a recommender system for different heterogeneous resources; GPUs, CPUs, MIC, etc… management for solving MFP on a heterogeneous system.

7 Acknowledgment

Running for the work described in this study was supported by High Performance Computing Center in King Abdulaziz University (Aziz Supercomputer) (http://hpc.kau.edu.sa).

8 References

[1] PatrikD'haeseleer, "What are DNA sequence motifs?", Nature Biotechnology, 2006, pp.24, 423-425

[2] N.Jones and P.A.Pavzner, "An Introduction to Bioinformatics Algorithms", 2004

[3] Das MK, Dai HK (2007),"A survey of DNA Motif Finding algorithms", BMC Bioinformatics 8.

[4] P. A. Pevzner and S.-H.Sze. Combinatorial approaches to finding subtle signals in DNA sequences. In ISMB, pages 269–278, 2000.

[5] FilippoGeraci, Marco Pellegrini, M. Elena Renda, "An efficient Combinatorial approach for solving the DNA Motif Finding problem", Ninth International Conference on Intelligent Systems Design and Applications, 2009.

[6] S. Tanaka, "Improved exact enumerative algorithms for the planted (l, d)-motif search problem," IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 11, no. 2, pp. 361–374, 2014.

[7] H.M.Faheem, "Accelerating Motif Finding Problem using Grid Computing with enhanced Brute Force", Advanced Communication Technology (ICACT),The 12th International Conference, vol. 1,2010,pp.197-210.

[8] H. M. Faheem, "Multiagent-based security for the wireless LAN," IEEE Potentials 24 (2), 19-22, 2005

[9] I. M. Hegazy, T. Al-Arif, Z. T. Fayed,and H. M. Faheem, "A framework for multiagent-based system for intrusion detection," In the Proceedings of Intelligent Systems Design and Applications, 117-125, 2003

[10] H. M. Kelash, H. M. Faheem, and M. Amoon, "It takes a multiagent system to manage distributed systems, " IEEE Potentials 26 (2), 39-45, 2007.

[11] Q. Yu, H. Huo, X. Chen, H. Guo, J. S. Vitter, and J. Huan, "An efficient algorithm for discovering motifs in large DNA data sets," IEEE Transactions on NanoBioscience, vol. 14, no. 5, pp. 535–544, 2015.

[12] Al-Okaily Anas and Huang Chun-Hs, "ET-Motif: Solving the Exact (l, d)-Planted Motif Problem Using Error Tree Structure" Journal of Computational Biology. 23(7): 615-623. doi:10.1089/cmb.2015.0238. June 2016

[13] Herath D, Lakmali C, Ragel R , "Accelerating String matching for Bio Computing Applications on Multicore CPUs",7th IEEE conference on Industrial and Information Systems (ICIIS), 2012

[14] FilippoGeraci, Marco Pellegrini, M. Elena Renda, "An efficient Combinatorial approach for solving the DNA Motif Finding Design and Applications, 2009.

[15] A. M. Carvalho, A. T. Freitas, A. L. Oliveira, and M.-F.Sagot.A highly scalable algorithm for the extraction of cis-regulatory regions. In APBC, pages 273–282, 2005.

[16] D. S. a. D. I. Ratne, "Use of a Probabilistic Motif Search to Identify Histidine Phosphotransfer Domain-Containing Proteins," PLOS one, pp. 1-18, 2016.

[17] Liu Y., Schmidt B., Liu W.,and Maskell D. , "CUDA-MEME: Accelerating Motif Discovery in Biological Sequences Using CUDA-enabled Graphics Processing Units",2009, Pattern Recognition Letters 31, 2170-2177

[18] Yu L., and Xu Y., "Parallel Gibbs Sampling Algorithm for Motif Finding on GPU", 2009, IEEE International Symposium on Parallel and Distributed Processing with Applications.

[19] Buhler J., and Tompa M., "Finding Motifs Using Random Projections", 2001, RECOMB '01 Proceedings of the fifth annual international conference on Computational biology.

[20] Yasmeen Farouk, TarekElDeeb, HossamFaheem, "Massively Parallelized DNA Motif Search on FPGA", Bioinformatics Trends and Methodologies, pp.107-120, 2011.

[21] Chen Chen, Bertil Schmidt, Liu Weiguo, and Wolfgang Müller- Wittig: GPU-MEME: Using Graphics Hardware to Accelerate Motif Finding in DNA Sequences. PRIB2008 LNBI 5265, 448- 459(2008).

[22] Y. Liu, B. Schmidt, W. Liu, and D. L. Maskell, "An ultrafast scalable Many-core Motif Discovery algorithm for multiple GPUs", IEEE International Parallel & Distributed Processing Symposium, 2011, pp. 423-42 1

[23] N. S. Dasari, Desh R., and Z. M, "Solving Planted Motif Problem on GPU", In Proceedings of the International Workshop on GPUs and Scientific Applications (GPUScA 2010), 2010

[24] H. M. Faheem, B. Koenig-Riez, Mahmoud Fayez, Iyad Katib, and N.AlJohani. "Solving the Motif Finding Problem on a Heterogeneous Cluster using CPUs, GPUs, and MIC Architectures", 2015. Mathematics and Computers in Sciences and Industry, CPS Published, pp. 226-232.

[25] Chanthini P, Shyamala K. A survey on parallelization of neural network using MPI and Open MP. Indian Journal of Science and Technology. 2016 May; 9(19). DOI: 10.17485/ijst/2016/v9i19/93835.

[26] Kalyani R. Application of multi-core parallel programming to a combination of ant colony optimization and genetic algorithm. Indian Journal of Science and Technology. 2015 Jan; 8(S2). DOI: 10.17485/ijst/2015/v8iS2/59091.

[27] G. Lawson V. Sundriyal M. Sosonkina Y. Shen "Modeling Performance and Energy for Applications Offloaded to Intel Xeon Phi" Proc. of the 2nd International Workshop on Hardware-Software Co-design for High Performance Computing (Co-HPC'15) 2015.

[28] G. Lawson , M. Sosonkina , and Y. Shen , " Towards modeling energy consumption of xeon phi ." CoRR , vol. abs/1505.06539, 2015 . [Online]. Available: http://dblp.unitrier.de/db/journals/corr/corr1505.html#LawsonSS15