A Proposed Approach for Motif Finding Problem Solved on Heterogeneous Cluster with Best Scheduling Algorithm

Abdullah Barghash, Ahmed Harbaoui
Department of Computer Science, King Abdulaziz University, Jeddah, KSA

***Abstract*—The Motif Finding Problem (MFP) is the problem of finding patterns in sequences of DNA. This paper discusses and presents an enhanced scheduling approach to solve the motif problem on the Heterogeneous Cluster by making a comparison between exact algorithms. The method that was followed is to analyze several exact algorithms, compare them within specific points to measure, and improve performance by comparing the number of devices and peripheral units used in every situation and running time in every method. Our experimental results show that the use of the scheduling approach that use different algorithms on Heterogeneous Cluster make a significant difference in the speed of completing the problem and in a shorter record time with less resources, and that this proposed approach is more effective than the traditional method of distributing tasks to solve the motif problem.**

***Keywords*—Motif finding problem; scheduling algorithm; heterogeneous; high-performance computing**

I. Introduction

Motif finding is a well-known problem in bioinformatics that involves identifying patterns or motifs within a set of DNA or protein sequences [1]. These motifs, which can be as short as a few base pairs or amino acids, can provide important insights into the function and regulation of genes and proteins [2]. The motif finding problem is computationally intensive, as it requires analyzing large datasets and generating many potential motif candidates. To accelerate the motif finding process, researchers have increasingly turned to high-performance computing (HPC) techniques, which involve the use of specialized hardware and software tools to distribute and parallelize the computational workload [3]. Within this paper, we present our proposed approach, which not only improves time performance but also requires fewer resources, making it a valuable contribution to the field.

HPC clusters can be constructed with a variety of computing resources, including Central Processing Units (CPUs), Graphics Processing Units (GPUs), Many Integrated Core (MIC) Architecture, and other computing resources. As the domain of processors continues to evolve, that will lead to more heterogeneity among them.

One HPC approach for solving the motif finding problem is the use of CPUs and GPUs [4, 5]. By using CPUs and GPUs in combination, researchers can harness the power of both types of processing units to analyze large datasets and search more efficiently for motifs.

Das and Dai [6] proposed another HPC approach where the use of many integrated core (MIC) architectures, such as Intel's Xeon Phi. MIC architectures are designed to provide high levels of parallelism and are well-suited for tasks that can be easily parallelized, such as motif finding. Zymbler and Kraeva [7] explained that by using MIC architectures, researchers can further increase the computational power available for solving the motif finding problem by following the proposed algorithm which showed high scalability, especially in the case of high computational load due to greater motif length.

Durbin et al. [8] found that effective use of HPC techniques for motif finding requires the implementation of appropriate scheduling strategies. Scheduling strategies determine how the computational workload is distributed among the available processing units and can significantly impact the efficiency of the motif finding process. For example, a scheduling strategy may involve dividing the dataset into smaller chunks and distributing them among the available CPUs, GPUs and MICs for parallel processing. Alternatively, Jones and Pevzner [9] determined that scheduling strategy may involve using machine learning techniques to optimize the allocation of computational resources.

HPC users may not always fully utilize the resources available to them on a cluster. This can be due to a variety of factors, including a lack of knowledge about the cluster's capabilities, the complexity of the HPC environment, limited time and resources, insufficient data, incorrect configuration, and inefficient algorithms. To overcome these challenges and fully utilize a cluster in their experiments, HPC users can seek support from experts, optimize their algorithms and data processing pipelines, and allocate sufficient resources. Additionally, HPC users can take advantage of cluster management tools and techniques, such as job scheduling and resource allocation, to better utilize the cluster's resources and improve the efficiency of their experiments. By addressing these challenges, HPC users can effectively leverage the power of HPC to solve complex problems and advance their research. Due to the difficulty of installing programs, as most software packages are not available by default for this environment, and the programs must be improved to take advantage of the capabilities of this device and match its high specifications, the same problems and challenges may be present through the administrator's view of the high-performance computer system. But, as software engineering advances, these problems are overcome by allowing software to optimize how it operates in a HPC environment. Another problem that is encountered from the point of view of the system administrator is that most users use by default one type of available resource, and this leads to a long waiting time in the queue for it to be released

for use by another user. This problem can be overcome by enabling scheduling algorithm that will direct users through different channels from resources to reduce waiting time in queue.

The effectiveness of using various types of resources in a heterogeneous environment depends on the parallel approaches chosen by the developer. The implementation of a scheduling strategy is a crucial aspect of performance. However, HPC users often utilize only one computing resource at a time for their experiments due to their expectations and behaviors. This paper presents a modified scheduling strategy for the planted motif finding problem that can achieve significant performance while using fewer computational resources.

II. Motif Finding Problem and Algorithms

The Motif-Finding problem (MFP) is the problem of finding patterns in sequences of DNA. Finding the common patterns in sequences is challenging as the DNA is a huge set [10]. This common pattern is called Motif and is usually a short segment that occurs frequently, see Fig. 1. These patterns considered a scientific interest in bioinformatics domain and those Motifs can be correspond to sequences of DNA that control the activation of specific genes [11]. MFP discovery algorithms can be classified into three categories based on its variants as Simple Motifs Search (SMS), Edited Motif Search (EMS) and Planted (l,d)-Motif Search (PMS)[12, 13]. This paper will consider (PMS) exact algorithms due to its high complexity [12] which makes it suitable to be solved on HPC systems as MFP is well known to be computationally intensive problem [14].To detect a Motif of length L with allowed mutation d and all possible L-$mers$ (4^L) is compared for all possible Motifs of length L and a sequence with size N we will get $(N-L+1)$ using Brute-Force Algorithm. We will present our parameters same as used in [14, 15] and for such intensive resources computations could be implemented using heterogenous platforms [4], [16]-[21] and for our experiment it will be conducted on a cluster containing CPUs, GPUs and MIC.The DNA constructed of nucleotides which is cytosine [C], guanine [G], adenine [A] or thymine [T] and can be represented using the regular expression in (1) . The length L and its possible L_{mers} represented in (2). The sequence set on (3) and the $match$ function used to compare Motifs A and B each of them has a size L represented on (4) the i_{th} position is represented where $AiBi$ for A and B Motifs. The counting of existence of L-mer in T sequences done by using $score$ function shown in (5).

Motif finding problem can be solved in many algorithms that researchers have investigated in the past two decades [12, 13] and these algorithms are influenced by the length and the allowed mutations [6]. Some techniques are using variants of brute force algorithm that requires hundreds of hours. Faheem et al. [15] proposed an algorithm that can benefit from different architectures to split the MFP into smaller sub problems that can be solved on heterogeneous architectures with minimal communications. They proposed a speed-based scheduling algorithm to split the work and they proved that the problem is a data parallel problem given that they are using only brute force algorithm which may not be the most suitable algorithm for such problem; also it's hard to for a normal user to reserve a full cluster due to the fact that most of HPC centers are a shared resource and they apply some resource limitation per user.

A Brute Force Algorithm solves this problem by considering all possible sets all 4^l possible $l-mers$. Compute the total distance of each $l-mer$ in that set from all other $l-mers$ in all t sequences. The correct Motif is the one with the smallest distances along all other $l-mers$. The running time of this algorithm is $O\ (4^l nt)$. To find a motif of $l=11$ is about 5 hours and longer motifs cannot be processed in a reasonable amount of time [4]. Although the execution time of the brute force algorithm is clearly too long to solve the challenge problem, but it's of the exact algorithms that never fail to find the motif [4, 12].

The original Brute Force can be improved to be an upgraded version of Brute-Force algorithm called "SKIP Brute-Force" (SKIP BF) [4, 5, 22] as proven by Faheem [4] by using a grid computing and the enhanced version of Skip Brute Force has better execution time. This approach can be implemented in parallel on different compute recourses as done by M. Al-Qutt et al. [5]. They implement the Skip Brute Force and solve MFP on CPU,GPU and MIC and the parallel version of their solution has significant execution time. The core enhancement on this SKIP BF Algorithm (Fig. 2) is to skip all iterations that won't lead to a correct solution. The algorithm behaves as Brute-Force algorithm by generating all possible 4^l $l-mers$ then a generated $l-mer$ of length L with d permitted mutations is considered matched if at least $(L-d)+1$ identical positions at both are matched. Then the algorithm starts looking to the next sequence and excludes any unmatched $l-mers$ from the search for the next sequence [5, 23]. SKIP BF leads to a better running time against the original Brute-Force Algorithm by skipping those irrelevant iterations and it's shown a high speedup performance on hardware accelerators like FPGA and shown a good chance for parallelization due to its fact of repetitive nature [22]. The complexity of this algorithm is $O\ (4^l nt)$ at its worst case [4].

The other algorithm to solve MFP was proposed by [3] called Recursive Brute Force Algorithm (RBF) and simply considered a searching technique that aims to search among the highest occurrence of the patterns of length L in set of characters and in some cases this algorithm allow us to accept the result of allowed mutation which means a non-exact match which is valid for MFP. RBF shown good time performance but its required a huge memory to be implemented using parallel methods as shown by Marwa Radad et al. [16, 25], Memory allocation for recursive algorithms is major point and for RBF proposed by [24] they use the static memory allocation technique to avoid memory management and it was implemented in two phases- the initialization phase begins with candidate initialization as shown in Fig. 4; then test the candidate's list to search for possible candidates. Then extend the target for a good Motif and all motifs that have grater mutations than d are called bad candidates. The second phase is candidate generation phase and the extension and addition phase. RBF algorithm use a parallelization layer and distribute the workload using parallel paradigms MPI (Fig. 3). The search is finished in (level 1) with no expansion in time of $O\ (4^l nt)$. It has same complexity as the original Brute Force Algorithm [16].

Nikolaos et al. [28] introduced a parallel algorithm aimed at

efficient Top-k Motif Discovery in Weighted Networks. Their approach demonstrated commendable scalability and speedup, especially with an increase in the number of CPU cores. In a separate study, Theepalakshmi et al. [29] developed an enhanced solution for planted motif by applying the Freezing FireFly (FFF) algorithm. Their method outperformed existing state-of-the-art optimization algorithms in terms of time efficiency. Despite these advancements, this paper will primarily concentrate on the work referenced as [4], [5], [15], [16] given that the same platform was used for our experimental procedures.

$$V \rightarrow A|C|G|T \tag{1}$$

$$Possible\ L_{mers} \rightarrow V^{l} \tag{2}$$

$$S = \{s1, s2, \ldots, sT\} \tag{3}$$

$$match(A, B, l, d) = \begin{cases} 1, l - d \geq \sum_i \begin{cases} 1, & Ai = Bi \\ 0, else \end{cases} \\ 0, else \end{cases} \tag{4}$$

$$score(L_{mers}, S, d) = \sum_{i=1}^{T} \sum_{k=0}^{N-l+1} match(L\_mer, s_i[k, \ldots, k+l], l, d) \tag{5}$$

$$motif = \{m \mid m = MAX\ (score(L\_mer, S, m)\ \forall\ L\_mer \in Possible\ L_{mers})\} \tag{6}$$

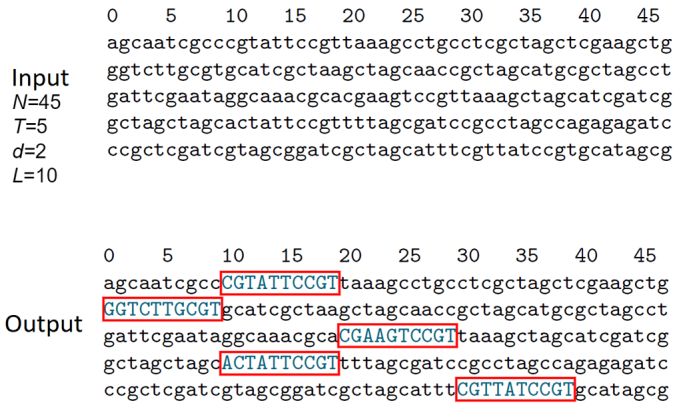

Fig. 1. Example of founded planted motif-[10, 2].

III. Performance Analysis of Motif Finding Algorithms on Different Architectures

To evaluate the performance of HPC for solving planted motif finding problems, we conducted a series of experiments using CPUs, GPUs, and MIC architectures. The experiments were designed to compare the performance of different algorithms on each type of resource. We used three different exact algorithms for solving the planted motif finding problem: Brute Force (BF), Skip Brute Force (SBF) and Recursive Brute Force (RBF).

```
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; %% (dows 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. 2. Pseudo code algorithm for SBF.

For the experiments, we used a Synthetic DNA sequences [13] as database with a data parameters that proposed by Pevzner [14] for Planted (l, d) Motif where the input sequences of size $N = 600$ each from the set of alphabets (1) and a motif M of size $l = 15$ and $d = 4$ allowed mutation.

For the CPU experiments, we used a cluster that with a limited recourses per user of eight nodes, each equipped with two Intel Xeon processors. For the GPU experiments, we used a cluster of 2 nodes, each equipped with one NVIDIA GPGPU K20 graphics card. For the MIC experiments, we used a cluster of two nodes, each equipped with one Intel Xeon Phi 7250 processor. We measured the performance of the algorithms by running time in seconds that reflected the time required to complete the planted motif finding process.

Examining the results in Table I gives an indication of the improvement in total run time with different algorithms. For instance, implementing the three algorithms using one regular node with OpenMP will reduce the run time from 11368 seconds for brute force algorithm on a single regular node to 2343 seconds for skip brute force algorithm with a speedup factor of 4.8 and recursive brute force algorithm run time 420 seconds that's comes with a speed factor of 28.2 to the original brute force algorithm run time while using eight regular nodes, will reduce the run time for brute force algorithm from 1416 seconds on pure 8 regular nodes to 73 seconds for skip brute force algorithm with a speedup factor of 19.3 and recursive brute force algorithm run time 40 seconds that's comes with a speed factor of 35.4 to the original brute force algorithm run time. For MIC and GPU, architectures are designed to

TABLE I. MFP RESULTS WITH DIFFERENT ARCHITECTURES

Trial No.	Platform	BF Results (sec)	Skip Results (sec)	RBF Results (sec)
1	1 Regular node (OpenMP)	11368	2343	420
2	1 Regular node (MPI+OpenMP)	11303	538	400
3	2 Regular node (MPI+OpenMP)	5627	273	120
4	4 Regular node (MPI+OpenMP)	2821	140	70
5	8 Regular node (MPI+OpenMP)	1416	73	40
6	MIC	11287	541	Not supported
7	GPU	10614	540	Not supported

```
Input:
l; d; t; S[1....,t]
Output:
ActualMotif [1...., x]
Score [1...,x]
BEGIN
    Allocate CandiateMotifs [4^(L/RANKS)] [L]
    CandiateMotifs ← Provision (4^(d+1))/RANKS Motif each of
    length d+1.
    Foreach motif in CandiateMotifs
    BEGIN
        Foreach seq in S
        BEGIN
            Foreach window in seq:
            BEGIN
                If (mismatch (motif, window) <d)
                BEGIN
                MatchCount++;
                Break;
                END
            END
            If MatchCount == n and Len (Motif) < L
                CandidateMotifs ←
                ProvisionNewMotifs (Motif)
            Else if MatchCount == n and len (Motif) == L
                Actual Motif ← Append (Motif)
        END
    END
    Return (ActualMotifs, Score (ActualMotifs)
END
```

Fig. 3. Pseudo code MPI algorithm for RBF.

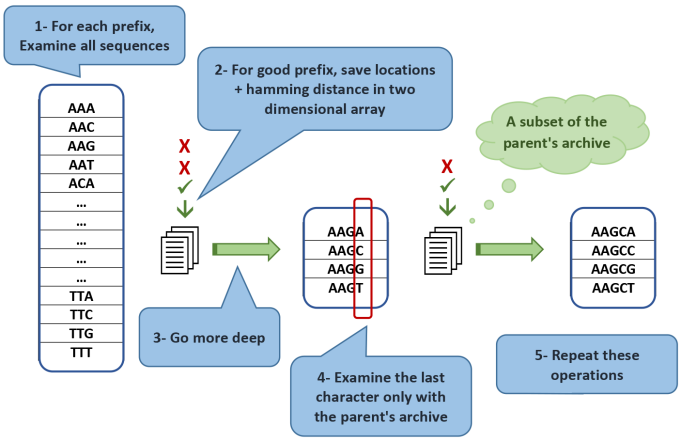

Fig. 4. RBF steps to find candidate motif.

provide high levels of parallelism and well-suited for tasks that can be easily parallelized. They are not well-suited for tasks that require sequential execution or complex control flow that use recursive function calls. Therefore, parallel recursive algorithms are generally not supported on MIC architectures or GPUs while results show that skip brute force algorithm has better running time than original brute force algorithm with speed up factor 5.9 to GPU and 20.8 for MIC.

The results of the experiments showed that the performance of the algorithms varied significantly depending on the type of resource and the algorithm deployed on each one of them. Overall, the GPU and MIC architectures outperformed the CPU architecture in terms of running time in some algorithms while CPU has better running time for recursive brute force algorithm, these results will provide valuable insights into the performance of different algorithms on different types of resources and will help to identify the most effective algorithm for each architecture and generate the assign map Table II. We will be able to optimize the use of computational resources in Table I and II to improve the scheduling strategy efficiency of the motif finding process in detail in the coming section.

TABLE II. ASSIGNED ALGORITHM TO EACH TYPE OF ARCHITECTURE

Architecture	Assigned Algorithm
CPU	RBF
GPU	SBF
MIC	SBF

IV. SCHEDULING STRATEGY AND PROPOSED APPROACH

A scheduling strategy is a plan or approach for distributing computational tasks among the available resources in a HPC environment. In the context of motif finding, a scheduling strategy involve dividing the dataset into smaller chunks and distributing them among the available CPUs, GPUs, and MICs for parallel processing. There are several approaches that can be used to develop a scheduling strategy for solving motif finding problems on HPC systems that have different types of computing nodes.

HPC systems are typically shared resources among multiple users, each with their own research goals and computational needs. In this context, it is often the case that one user cannot fully utilize the entire cluster for their own use and must share the resources with other users. This can lead to a lack of access to the resources that a user needs for their research, or to longer wait times for access to the resources. Furthermore, HPC users often rely on only one type of computing resource, such as CPU or GPU, for their experiments which can lead to underutilization of other resources and less efficient use of the available resources.

A scheduling strategy can help to mitigate these issues by more efficiently allocating the resources of the HPC system among the different users and their tasks. By using a scheduling strategy that takes into account the characteristics of the tasks and the available resources, it is possible to improve the overall performance of the HPC system and increase the number of users that can be accommodated on the system.

We use a Task-based scheduling strategy that assigns tasks to the computing node that is most suitable for solving them based on the best performed algorithm from previous results. These types of strategies are often used to optimize the performance of HPC systems by ensuring that tasks are executed on the most appropriate computing resources [26].

We have developed a scheduling strategy that can be used to divide tasks based on the best-performed algorithm for solving the motif finding problem on CPUs, GPUs, and MICs with similar approach described in [15], [27] . The PBS script used to construct the scheduler and designed to optimize the performance of HPC systems by selecting the best algorithm for each task on each type of resource based on its performance, then will divide the workload among other computing recourses as shown in Fig. 5. The scheduler can help to ensure that tasks are executed efficiently and accurately, improving the overall performance of the system. In addition, the PBS script can be easily modified to support new algorithms or to adapt to changes in the characteristics of the data, making it a flexible and versatile tool for solving the motif finding problem on HPC systems with different types of resources.

V. Experiment Results and Validation

After running the scheduler on a heterogeneous cluster consisting of CPUs, GPUs, and MICs, we obtained the following results.

TABLE III. Results of the Previous Papers

Platform	CPU Ratio %	CUDA Ratio %	MIC Ratio %	Results
1 Regular Node+ 1 CUDA+1 MIC	40.321	29.867	29.812	2330.11
2 Regular Node+ 1 CUDA+1 MIC	69.250	15.389	15.361	1978.20
4 Regular Node+ 1 CUDA+1 MIC	79.427	10.296	10.277	1533.72
8 Regular Node+ 1 CUDA+1 MIC	86.549	6.732	6.719	1056.87

TABLE IV. Results of the Proposed Approach

Platform	CPU Ratio %	CUDA Ratio %	MIC Ratio %	Result
1 Regular Node+ 1 CUDA+1 MIC	40.321	29.867	29.812	363
2 Regular Node+ 1 CUDA+1 MIC	69.250	15.389	15.361	182
4 Regular Node+ 1 CUDA+1 MIC	79.427	10.296	10.277	121
8 Regular Node+ 1 CUDA+1 MIC	86.549	6.732	6.719	77

The scheduler was able to effectively divide the tasks based on the best-performed algorithm for each type of resource, resulting in an overall improvement in the performance of the system. With more resources that selected the more workload, is assigned as the case for CPU as shown in Table I .

```
1.  PROGRAM MotifScheduler
2.  Input : S[1, ..., T]
3.  Input : L
4.  Input : d
5.  Input : Matrix M
6.  BEGIN
7.  t[t1,...,tp] ← load single task execution time for architectures
8.  tmin← MIN_{i=1}^{p} (ti)*4^L ; find the smallest run time
9.  Assign smallest run time to the best performed algorithm from Matrix M
10. FOR i=1 to p
11.     IF ti ≤ tmin THEN
12.         Ri ← 4^L/ ti ; find the weight of each architecture
13.     ELSE
14.         Ri ← 0 ; this architecture is very slow and will be ignored
15.     END
16. END
17. Rtotal ← R1+ R2 + ⋯ + Rp ; sum the weights
18. Ru ← 4^L / Rtotal ; find the tasks assigned to each weight unit
    offset = 0
    starti = 0
19. FOR i = 1 to p
20.     Ci= Ri * Ru ;tasks assigned to architecture
        starti = starti + offset ; determine the start index of tasks
        endi = starti +Ci – 1 ; determine the end index of tasks
        offset = Ci
21.     Scorei ← SPAWN Algorithmi(S, L, d, Ci, starti, endi)
22. END
23. return MAX_{i=1}^{p} (Scorei) ; find the motif of highest occurrence
24. END
```

Fig. 5. Pseudo code of scheduling algorithm to assign workload to architecture.

TABLE V. Comparison between Results

Platform	Previous Results	Our Results	Speed up
1 Regular Node+ 1 CUDA+1 MIC	2330.11	363	6.42
2 Regular Node+ 1 CUDA+1 MIC	1978.20	182	10.87
4 Regular Node+ 1 CUDA+1 MIC	1533.72	121	12.68
8 Regular Node+ 1 CUDA+1 MIC	1056.87	77	13.73

By comparing our results with Table III from [15] that use the same cluster we can see with one Regular Node ,one CUDA and one MIC, our approach managed to finish in 363 second instead of 2330 seconds with speedup factor 6.4 and by using 8 Regular Node ,1 CUDA and 1 MIC, as this is the maximum user limitation for the user in the cluster we gain a speedup factor 13.7 in total time of 77 seconds while in Table III its required full capacity that consist of 265 Regular Node to gain total performance of 54.91 seconds.

A comparison of our results in Table IV with those presented in Table III from study [15] shows that our approach was able to achieve a significant improvement in performance. Using a configuration of one Regular Node, one CUDA Node, and one MIC Node, our approach was able to complete the planted motif finding problem in 363 seconds, compared to

2330 seconds in Table III. This resulted in a speedup factor of 6.4. Additionally, by using a configuration of 8 Regular Node, 1 CUDA Node, and 1 MIC Node, which is the maximum user limitation for our cluster, we were able to achieve a speedup factor of 13.7 in a total time of 77 seconds as shown in Table V. This is compared to the full capacity configuration of 265 Regular Node in Table III which took 54.91 seconds. The chart in the Fig. 6 illustrates a comparison between previous studies and the proposed approach.

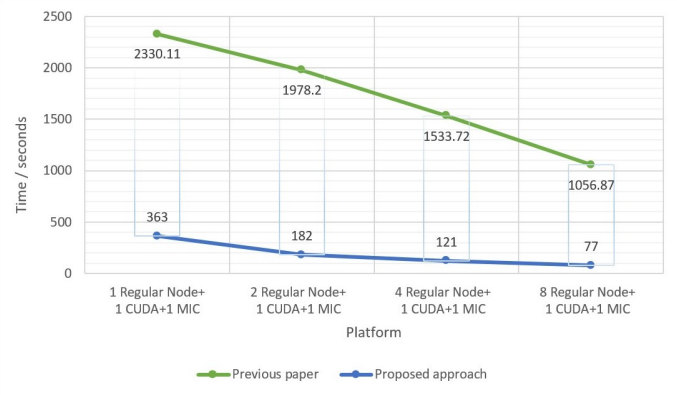

Fig. 6. Comparison between previous studies and the proposed approach.

These results demonstrate the effectiveness of our approach in achieving improved performance in motif finding problem on heterogeneous cluster. Hence, it can be seen that the waiting time has become significantly less, because the distribution of users by placing them in different channels to use the resources makes the available resources larger and does not require a large launch time to make them available to the new user.

VI. Conclusion

In this paper, we propose a scheduling algorithm to make better use of the heterogeneous cluster. It is based on the idea of dividing tasks on multiple types of resources available on the HPC. The results of this algorithm were compared with previous results, but they were using only one type of resource, which caused a burden and took a longer time to solve the problem.

Overall, the use of the scheduler resulted in a significant improvement in the performance of the system for solving the motif finding problem on a heterogeneous cluster with different algorithms. The scheduler was able to effectively select the best-performed algorithm for each type of resource, resulting in an efficient and accurate solution to the problem with much more less resources.

Acknowledgment

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

References

[1] P. A. Pevzner, Computational Molecular Biology: An Algorithmic Approach. 2000. doi: 10.7551/mitpress/2022.001.0001.

[2] M. Soskine and D. S. Tawfik, "Mutational effects and the evolution of new protein functions," Nat Rev Genet, vol. 11, no. 8, Art. no. 8, Aug. 2010, doi: 10.1038/nrg2808.

[3] W. Kim, M. Li, J. Wang, and Y. Pan, "Biological network motif detection and evaluation," BMC Systems Biology, vol. 5, no. 3, p. S5, Dec. 2011, doi: 10.1186/1752-0509-5-S3-S5.

[4] H. M. Faheem, "Accelerating motif finding problem using grid computing with enhanced Brute Force," in 2010 The 12th International Conference on Advanced Communication Technology (ICACT), Feb. 2010, vol. 1, pp. 197–202.

[5] H. Khaled, M. Al-Qutt, R. El-Gohary, H. Faheem, I. Katib, and nayif al-johani, Accelerating Motif Finding Problem Using Skip Brute- Force on CPUs and GPU's Architectures. 2017.

[6] M. K. Das and H.-K. Dai, "A survey of DNA motif finding algorithms," BMC Bioinformatics, vol. 8, no. S7, p. S21, Dec. 2007, doi: 10.1186/1471-2105-8-S7-S21.

[7] M. L. Zymbler and Ya. A. Kraeva, "Discovery of Time Series Motifs on Intel Many-Core Systems," Lobachevskii J Math, vol. 40, no. 12, pp. 2124–2132, Dec. 2019, doi: 10.1134/S199508021912014X.

[8] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, 1st ed. Cambridge University Press, 1998. doi: 10.1017/CBO9780511790492.

[9] N. C. Jones and P. Pevzner, An introduction to bioinformatics algorithms. Cambridge, MA: MIT Press, 2004.

[10] P. F. Stadler, M. E. M. T. Walter, M. Hernandez-Rosales, and M. M. Brigido, Eds., Advances in Bioinformatics and Computational Biology: 14th Brazilian Symposium on Bioinformatics, BSB 2021, Virtual Event, November 22–26, 2021, Proceedings, vol. 13063. Cham: Springer International Publishing, 2021. doi: 10.1007/978-3-030-91814-9.

[11] D. Edwards, J. Stajich, and D. Hansen, Eds., Bioinformatics. New York, NY: Springer, 2009. doi: 10.1007/978-0-387-92738-1.

[12] S. Mohanty, P. K. Pattnaik, A. A. Al-Absi, and D.-K. Kang, "A Review on Planted (l, d) Motif Discovery Algorithms for Medical Diagnose," Sensors (Basel), vol. 22, no. 3, p. 1204, Feb. 2022, doi: 10.3390/s22031204.

[13] F. A. Hashim, M. S. Mabrouk, and W. Al-Atabany, "Review of Different Sequence Motif Finding Algorithms," Avicenna J Med Biotechnol, vol. 11, no. 2, pp. 130–148, 2019.

[14] P. A. Pevzner and S. H. Sze, "Combinatorial approaches to finding subtle signals in DNA sequences," Proc Int Conf Intell Syst Mol Biol, vol. 8, pp. 269–278, 2000.

[15] H. M. Faheem, B. Koenig-Riez, M. Fayez, I. Katib, and N. Al-Johani, "Solving the Motif Finding Problem on a Heterogeneous Cluster using CPUs, GPUs, and MIC Architectures".

[16] M. A. Radad, N. A. El-Fishawy, and H. M. Faheem, "Implementation of Recursive Brute Force for Solving Motif Finding Problem on Multi-Core," Int. J. Syst. Biol. Biomed. Tech., vol. 2, no. 3, pp. 1–18, Jul. 2013, doi: 10.4018/ijsbbt.2013070101.

[17] S. J. Park, D. R. Shires, and B. J. Henz, "Coprocessor Computing with FPGA and GPU," in 2008 DoD HPCMP Users Group Conference, Jul. 2008, pp. 366–370. doi: 10.1109/DoD.HPCMP.UGC.2008.69.

[18] M. Showerman et al., "QP: A Heterogeneous Multi-Accelerator Cluster".

[19] D. B. Thomas, L. Howes, and W. Luk, "A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation," in Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, New York, NY, USA, Feb. 2009, pp. 63–72. doi: 10.1145/1508128.1508139.

[20] K. Underwood, "FPGAs vs. CPUs: trends in peak floating-point performance," in Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, Monterey California USA, Feb. 2004, pp. 171–180. doi: 10.1145/968280.968305.

[21] H. Khaled, H. E. D. M. Faheem, and R. El Gohary, "Design and implementation of a hybrid MPI-CUDA model for the Smith-Waterman algorithm," Int J Data Min Bioinform, vol. 12, no. 3, pp. 313–327, 2015, doi: 10.1504/ijdmb.2015.069710.

[22] Y. Farouk, T. ElDeeb, and H. Faheem, "Massively Parallelized DNA Motif Search on FPGA," in Bioinformatics - Trends and Methodologies, M. A. Mahdavi, Ed. InTech, 2011. doi: 10.5772/23578.

[23] P. Perera and R. Ragel, "Accelerating motif finding in DNA sequences with multicore CPUs," in 2013 IEEE 8th International Conference on Industrial and Information Systems, Dec. 2013, pp. 242–247. doi: 10.1109/ICIInfS.2013.6731989.

[24] H. Khaled, "Enhancing Recursive Brute Force Algorithm with Static Memory Allocation: Solving Motif Finding Problem as a Case Study," in 2019 14th International Conference on Computer Engineering and Systems (ICCES), Dec. 2019, pp. 66–70. doi: 10.1109/ICCES48960.2019.9068158.

[25] M. A.Radad, N. A. El-fishawy, and H. M. Faheem, "Enhancing Parallel Recursive Brute Force Algorithm for Motif Finding," IJCA, vol. 86, no. 3, pp. 15–22, Jan. 2014, doi: 10.5120/14965-3143.

[26] O. Sinnen, Task Scheduling for Parallel Systems, 1st edition. Hoboken, N.J: Wiley-Interscience, 2007.

[27] H. M.Faheem and B. König-Ries, "A New Scheduling Strategy for Solving the Motif Finding Problem on Heterogeneous Architectures," IJCA, vol. 101, no. 5, pp. 27–31, Sep. 2014, doi: 10.5120/17685-8543.

[28] A. Papadopoulos and N. Koutounidis, "Parallel Top-K Motif Discovery in Weighted Networks," Available SSRN 4418674, 2023.

[29] P. Theepalakshmi and U. S. Reddy, "Freezing firefly algorithm for efficient planted (l, d) motif search," Med. Biol. Eng. Comput., vol. 60, no. 2, pp. 511–530, 2022.