

Accelerating Motif Finding Problem using Grid Computing with Enhanced Brute Force

H.M. Faheem, PhD.

Associate Professor, Computer Systems Dept.,
Faculty of Computer and Information. Sciences, Ain Shams University,
Cairo, Egypt
hmfaheem@eun.eg

Abstract— Motif finding problem is a major task to understand the mechanisms of gene expression regulation. Motif is generally defined as a recurring pattern in the sequence of nucleotides or amino acids. In the DNA sequence, it is usually a short segment that occurs frequently, but not required to be an exact copy for each occurrence. This property of motif makes motif mining very difficult. In fact, motif finding problem is proven to be NP-Complete. Many algorithms fail to solve the well-known challenge problem of finding a motif of length 15 and has at most 4 mutations (15, 4) due to the huge runtime needed. Others fail to distinguish the motif from background sequences. Brute Force is one of the exact algorithms that never fail to find the motif, but it suffers from an intractable running time. In this paper, we present a novel approach to accelerate the motif finding problem using grid computing. As grid computing makes available low-cost high-performance computing (HPC) infrastructures, the best candidates for using such infrastructures are applications that require high computational power, large storage capacity, or fast and high-throughput networking. The nature of motif finding problem is considered one of the perfect scenarios for grid computing. We deployed an enhanced version of the Brute Force; skip-Brute Force running on EUMEDGRID infrastructure (Co-Funded project by the European Commission in the framework of FP6, with the aim of supporting the development of a Grid e-Infrastructure in the Mediterranean Area and promoting the porting of computationally intensive applications on the Grid platform). The idea behind our skip Brute Force algorithm is that it *skips* all the iterations that will not lead to a correct solution. The message passing programming paradigm is deployed since it assumes a partitioned address space and supports explicit parallelization. Our experimental results showed boosting in the performance without sacrificing the exactness of the Brute Force.

Keywords- Motif findin; DNA sequence analysis; Grid Computing.

I. 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 grid computing to the problem. Grid computing is 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 [1-3]. Pattern-based algorithms include PROJECTION [4,5], MULTIPROFILER [6], and MITRA [3]. Profile-based algorithms including CONSENSUS [7], MEME [2] and Gibbs sampling [5], try to find the starting positions of the occurrences of the motif in each sequence.

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 Pevzner 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. It never misses a motif, as it always searches in all the possible locations. What makes Brute Force an impossible solution is 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.

Many approaches can be applied to speed up the running time of any algorithm using hardware; examples include chip multiprocessors, graphics processing units (GPUs), (FPGAs), and special hardware solutions. These solutions require hardware specialists who are able to design such systems. Moreover, it is not realistic to design architecture for every single bioinformatics problem. In fact, the grid as an emerging infrastructure that provides seamless access to computing power and data storage capacity distributed over the different grid computing nodes is ideally qualified to knock the door of computationally intensive problems like motif finding problem.

The rest of this paper is organized as follows: In Section II we describe the motif finding problem and the Brute Force algorithm. In Section III, a brief explanation of the skip brute force algorithm is presented in addition to the explanation of the parallel nature of the algorithm. Section IV describes the

grid-based implementation of the suggested approach. Performance evaluation is presented in section V. Finally, section VI concludes our work and presents future enhancements.

II. Motif finding problem

The 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 corrupted 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 planted 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 this 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 high.

III. Enhanced Brute Force Algorithm- the skip Brute Force

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 generates all possible 4^l l -mers. It then iterates over all the sequences examining the generated l -mer with all the windows in the first sequence. The algorithm then excludes all the generated l -mers that are not matched with at least one window of the sequence. Matching means that the hamming distance between the l -mer and the window is less than or equal to 4. In other words; a generated l -mer of length 15 with 4 permitted mutations is considered matched with a window if at least 12 identical positions at both of them are matched. The algorithm then starts checking the next sequences and consequently excludes unmatched l -mers from the search in the next sequence. The algorithm repeats this mechanism till the end of the sequences. A pseudo code of the sequential *skip* Brute Force algorithm is shown in Fig.1.

The complexity of this algorithm is $O(4^l nt)$ at its worst case, just as the Brute Force. The performance coefficient introduced by Pevzner and Sze to measure the performance is defined as follows: If K denotes the set of t, l base positions in t occurrences of a planted motif, and P denotes the corresponding set of base positions in the t occurrences predicted by the algorithm. Then the performance coefficient equals $K \cap P / K \cup P$. If the algorithm succeeds to recover all the positions of the planted motif correctly, this will give a performance coefficient of 1, which is typically the case of Brute Force. The performance coefficient gives an indication of how many positions of the motif occurrences in the sequences have been predicted correctly by the algorithm.

```

1. for L = 0 to  $4^{l_{\text{motifSize}}}$  - 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.      end
16.    if motif_found == 0
17.      break; % Skip that Li, it is not the Motif
18.    end
19.  end
20. end
21. if solution_found
22. break;
23. end
24. end

```

Fig. 1 Pseudo Code for Skip Brute Force Algorithm

The brute force or skip-brute force algorithms are inherently parallel algorithms with well-defined points of synchronization and are thus well suited to implementation of Bulk Synchronous Parallel (BSP) computer. A computer constructed with this paradigm is made up of a number of components each comprising three parts: parallel threads, known as virtual processors; a point-to-point routing mechanism; and facilities for providing synchronization between some or all of the threads at a regular time interval. 2. The solution of a problem on the BSP computer consists of a number of super-steps, at one or more time intervals apart. Fig.2 shows the parallel comparisons between the generated l -mers and the windows of a given sequence on the components

(c) of the BSP model [9]. Of course a preliminary data preparation step should be performed before the start up of comparison processes. This step extracts all the possible windows from a given sequence. Hence each l -mer of the 4^l l -mers will be compared with all the possible $(n-l+1)$ windows. The total number of comparisons is $(n-l+1) * 4^l$. Actually there is no dependency between these comparison processes.

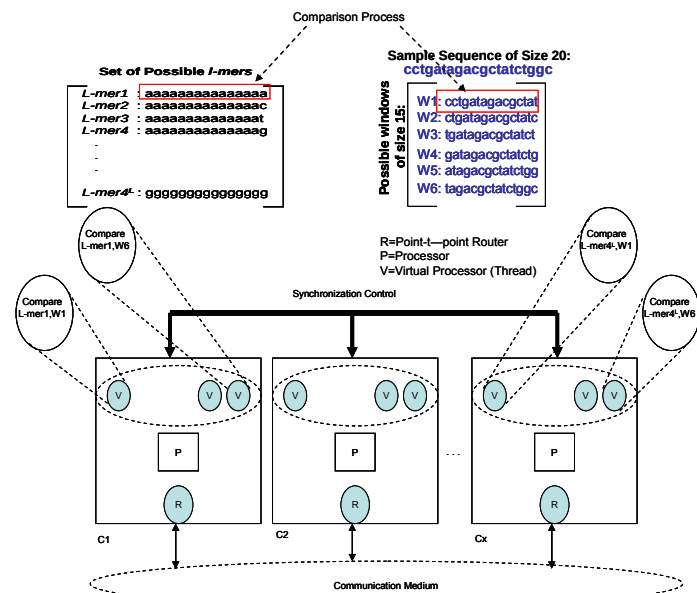


Fig.2 Comparison processes using the BSP model

IV. Grid-based Implementation

The BSP model was introduced to provide a bridge between software developed for parallel machines and the architectures on which this software is ultimately implemented. BSP was proposed as a possible standard to facilitate the development of general purpose parallel machines and increase the portability of software between these machines. In order to implement the motif finding problem on a specific grid node consisting of multiple computers, we have to correlate the elements of the grid node to the components of the BSP [9].

The message-passing programming paradigm is one of the oldest and most widely used approaches for programming parallel computers. Its wide spread adoption can be attributed to the fact that it imposes minimal requirements on the underlying hardware.

The two key attributes that characterize the message-passing programming paradigm are: it assumes a partitioned address space; and it supports only explicit parallelism. The message-passing interface MPI defines a standard library for message-passing that can be used to develop portable message-passing programs using either C or FORTRAN. Most message-passing programs are written using the single program multiple data (SPMD) approach. In the SPMD programs the code executed by different processes is identical except for a

small number of processes. We can obviously see that the BSP model components have the same attributes of the message-passing programming paradigm. This makes the message-passing programming paradigm is the optimum choice for implementing the skip brute force algorithm on the grid [10].

A. EUMEDGRID Structure

A set of grid nodes is implemented at different Mediterranean countries. EUMEDGRID deploys glite as a stack of generic middleware. Main glite components include user interface gliteUI, virtual organization management service VOMS, information service, workload management system gliteWMS, logging and bookkeeping utility gliteLB, computing services, data management, monitoring services, and secure access services. Typical grid node structure is shown in Fig.3.

User Interface allows users to access Grid functionalities. It is simply a machine where users have a personal account and where the user certificate is installed. It is also a gateway to grid services. The user interface provides a command line interface to perform some basic Grid operations such as: list all the resources suitable to execute a given job, submit jobs for execution, show the status of submitted jobs, cancel one or more jobs, retrieve the logging and bookkeeping information of jobs, retrieve the output of finished jobs, and copy, replicate and delete files from grid.

The resource broker (RB) is responsible for the acceptance of submitted jobs and for sending those jobs to the appropriate computing element. The resource broker retrieves information from Information catalogues so as to find the proper available resources depending on the job requirements.

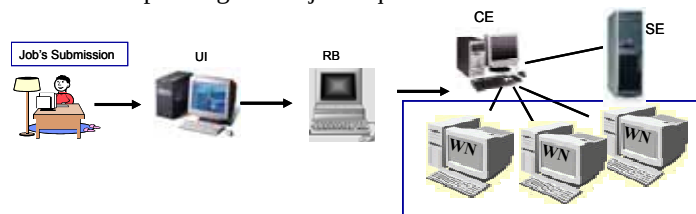


Fig.3 Grid Node Structure

The computing element (CE) is considered the "Grid interface". It is built on a farm of computing nodes called Worker Nodes. It also executes the basic queues functions. The computing element is responsible for accepting jobs and dispatching them for execution to the worker nodes. The state of an executing job is being watched by the Computing Element. LCGCE is the current CE service implementation of the production infrastructure, based on Globus toolkit.

The submitted jobs are being executed in the worker nodes (WN). Only basic services of middleware are required in the Worker nodes such as: application libraries, Application Programming Interfaces (API), and commands for performing actions on grid resources and data.

The storage element (SE) provides uniform access to storage resources. It may control simple disk servers, large disk arrays or Mass Storage Systems.

Information service provides information about the Grid resources and their status. Monitoring service has a set of monitoring tools designed for grid systems. It provides basic data about Grid like the number of each site's CPUs, the number of submitted jobs, the number of executing jobs, the number of waiting jobs, the number of each site's free CPU, the available storage space, etc. Secure access service decides resource access policy, authentication, an authorization mechanisms.

B. MPI Program

Based on the nature of the problem and the huge number of concurrent comparisons, we introduced a novel approach. Since all the implementations of algorithms similar in nature to brute force are based on the generation of the possible l-mers through loops, we here assumed that the possible l-mers are distributed evenly among the different worker nodes. On the other hand, all the sequences will be submitted to all the worker nodes. This means that each worker node will have a complete set of all the sequences and a portion of the possible l-mers. Typical execution steps after data distribution of both the l-mers and the sequences are listed in Fig. 4.

- Step 1:* For all worker nodes; expand each sequence to provide $(n-l+1)$ windows

Step 2: For all worker nodes; perform the comparison process between each window of the first sequence and the resident l-mers of the worker node.

Step 3: For all worker nodes; exclude the unmatched l-mers

Step 4: Repeat steps 2 and 3 on the next sequence until the end of sequences

Step 5: Collect results and then display it

Fig.4 MPI Program steps for implementing skip brute force

V. Performance Evaluation and Results

A test of the performance of the Skip Brute Force algorithm on synthetic problem instances is performed. Problem instances are generated according to the planted (l, d) -motif model. We followed up the FM model described by Pevzner and Sze[8]. Problem instances are produced 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. For each set of

parameter l and d , we ran 10 test cases and recorded the average running time of each algorithm.

In order to evaluate the grid-based implementation; we need to define the expected number of matching operations. First we define the probability P_d to find a random l-mer in a given sequence with up to d mutations [2]. This is shown in (1). Additionally, we define the expected number of the required matching operations $E(l, d)$ to find the correct implanted motif as in (2). We then deduce for a problem of size $n=600, t=20$, the expected matching operations to be as in table 1.

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

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

Matching operations constitute the most time consuming part of the algorithms although it could be performed concurrently. Unit matching time is defined as the time required carrying out a set of l comparisons in order to decide whether an l -mer is matched with a corresponding window of a given sequence or not. Unit matching time depends of course on the length of the window. A comparison process for an l -mer of size 15 will be different from that of length 11. Typical Unit Matching Time running on Dual Core, 2.5 GHz, 2GB RAM (the default machine specifications for all the worker nodes in our experiments) for different l and d values is shown in Table 2.

Table 1: Expected matching operations for different (l, d)

l	d	Expected Matching Operations
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}

Assuming that the sequences and the possible l -mers are initially stored as data files on the computers, a typical analysis of the expected sequential implementation time T_s of Skip Brute Force algorithm on a single computer machine is given by (3).

$$T_s = T_{expand} + T_{comp} + T_{save} \quad (3)$$

Where;

T_{expand} is the time needed to expand the sequences such that a complete set of windows is generated for each sequence, T_{comp} is the time needed to perform comparison processes, and T_{save} is the time needed to save the results to a data file.

The MPI implementation time T_{MPI} on the grid is given by (4):

$$T_{MPI} = T_{expand} + (T_{comp}/P) + (T_{save}/P) + T_{comm} \quad (4)$$

Where;

P is the number of worker nodes, and T_{comm} is the communication and synchronization time of the deployed grid elements.

Table 2: Unit Matching Time

l	d	Unit Matching Time (sec.)
11	3	2.78175E-07
12	3	3.03464E-07
13	4	3.28752E-07
14	4	3.54041E-07
15	4	3.7933E-07

A. Experimental Results

Typical experimental results for the execution time of skip Brute Force algorithm on a single, 24, and 96 worker nodes are listed in tables 3, 4, and 5 respectively. It is clear that as l increases, the execution time increases. Also we can see that as the number of WNs increases, the execution time decreases while the communication time increases. Typical LAN setup of the computational grid node on which we conducted our experiments consists of 4 LAN switch each has 24 10/100 Mbps UTP ports and 4 ports each of 1Gbps (uplink modules). A typical star topology is implemented such that one switch connects the other 3 switches. Each switch connects 24 WNs with a rate of 100 Mbps for each WN.

Table 3: Implementing the skip Brute Force Algorithm on a single machine

l	d	$P=1$			
		T_{expand} (sec.)	T_{comp} (sec.)	T_{save} (sec.)	T_s (sec.)
11	3	13.2	344.60	68	425.80
12	3	13.2	1499.96	271.31	1784.47
13	4	13.2	6492.85	1084.11	7590.17
14	4	13.2	27903.02	4326.18	32242.41
15	4	13.2	119352.24	17271.16	136636.61

Table 4: Implementing the skip Brute Force Algorithm on computing grid has 24 Worker Nodes

l	d	$P=24$			
		T_{comp} (sec.)	T_{save} (sec.)	T_{comm} (sec.)	T_{MPI} (sec.)
11	3	13.784	2.72	6.25	35.95412772
12	3	59.998	10.85	24.1	108.1511767
13	4	259.71	43.36	95.98	412.2588441
14	4	1116.1	173.047	384.2	1686.568651
15	4	4774.1	690.84	1536.7	7014.836596

B. Performance Evaluation

speedup of an algorithm using P processors is defined as $S_p = t_s / t_p$ where t_s is the execution time of the *best available sequential algorithm* and t_p is the execution time of the parallel algorithm. Amdahl's law states that the speedup given P processors is as in (5).

$$S_p = T_s / (f \times T_s + (1-f) T_s / P) = P / (1 + (P-1)f) \quad (5)$$

Where; f is the fraction of the computation that is sequential and cannot be divided into concurrent tasks. It is obvious that f is directly proportional to T_{expand} , and T_{comm} . Also f is inversely proportional with the problem size since the total sequential execution time is affected by it. Hence; f is given by (6).

$$f = (T_{expand} + T_{comm}) / T_s \quad (6)$$

Table 6 shows the Speedup S_p for different l and d . It is clear that the implementation of the motif finding problem using skip brute force on the grid yields a considerable enhancement in the speed of the algorithm.

Table 5: Implementing the skip Brute Force Algorithm on computing grid has 96 Worker Nodes

l	D	$P=96$			
		T_{comp} (sec.)	T_{save} (sec.)	T_{comm} (sec.)	T_{MPI} (sec.)
11	3	3.446	0.68	6.27	23.59
12	3	15	2.71	24.6	55.51
13	4	64.92	10.84	96.98	185.94
14	4	279.03	43.26	385.2	720.69
15	4	1193.5	172.71	1555.3	2934.73

On the other hand, several implementations for solving motif finding problem on Matlab can be used in comparisons. Of course much more enhancement is achieved if we compared our results with the Matlab implementation of Brute Force algorithm running on an AMD 5500 X2+ processor and 2GB RAM. This is shown in Table 7. We can see that a boost factor of 6419 is achieved when 96 worker nodes are deployed as compared to the run time of Matlab implementation.

Table 6: Speedup S_p for different l , d , and f values

l	d	$P=24$		$P=96$	
		f	S_p	f	S_p
11	3	0.0456	11.703	0.04572	17.964
12	3	0.02090	16.207	0.021182	31.868
13	4	0.01438	18.033	0.014516	40.35
14	4	0.01232	18.699	0.012356	44.161
15	4	0.0113	19.034	0.011479	45.921

VI. Conclusion and Future Directions

Motif finding problem has been considered as one of the most challenging research topics for the scientists of bioinformatics. In this paper, we presented a grid-based solution that aimed at speeding up the solution of the problem. We introduced an enhanced version of the Brute Force algorithm called skip Brute Force. We deployed message-pass interface to parallelize the problem. We then implemented the solution on EUMEDGRID infrastructure. Results showed the achievement in terms of execution time such that the grid-based implementation provides a good speedup factor.

We can see that the experiment is conducted on a normal infrastructure that consists of traditional PCs connected to LAN switches and MPI code written in C language. This means that no need for specialized or sophisticated hardware infrastructures that costs very much.

We strongly believe that such applicability of grid computing can greatly improve the way the bioinformatics research is implemented. This paper is a step towards a complete suite of grid-based implementations in the field of bioinformatics.

Table 7: Comparison with Matlab Results

<i>l</i>	<i>d</i>	Matlab Run Time (sec.)	P=24		P=96	
			TMPI (sec.)	Boost Factor	TMPI (sec.)	Boost Factor
11	3	84000	35.95	2336.31	23.59	3559.92
12	3	237360	108.15	2194.70	55.51	4275.77
13	4	1461854	412.25	3545.96	185.94	7861.55
14	4	4655320	1686.56	2760.23	720.69	6459.51
15	4	18840000	7014.83	2685.73	2934.73	6419.66

REFERENCES

- [1] S.Rajasekaran, S. Balla, and CH. Huang, "Exact algorithm for planted motif challenge problems," in the Proceedings of Asia-Pacific Bioinformatics Conference, 249–259, 2005
- [2] T. L. Bailey, N. Williams, C. Mistleh, and W. W. Li, "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] J. Buhler, M. Tompa, "Finding motifs using random projections," Journal of Computing. Biology, Vol. 9, pp225–242, 2002.
- [5] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton, "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] G. Hertz, G. Stormo, "Identifying DNA and protein patterns with statistically significant alignments of multiple sequences," Bioinformatics 15(7-8), 563–577, 1999
- [8] P. Pevzner and S. H. Sze, "Combinatorial approaches to finding subtle signals in DNA sequences," in the Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology," 269–78, 2000
- [9] A. Chalmers and J. Tidmus, "Practical Parallel processing, An Introduction to Problem Solving in Parallel," International Thomson Computer Press, 1996.
- [10] A. Grama, A. Gupta, J. Karypis, and V. Kumar, "Introduction to Parallel Computing,, 2nd Edition," Addison Wesley Publishing Inc., 2003