International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-4, February 2020

Parallel Algorithms for Discovering Planted (*l*, d) Motif

Satarupa Mohanty, Biswajit Sahoo

Abstract*: In computational biology, motifs are short, recurring patterns of biological sequences that possess the principal character for the analysis and interpretation of various biological issues like human disease, gene function, drug design, etc. The major objectives of the motif search problem are the management, analysis, and interpretation of huge biological sequences using computational techniques from computer science and mathematics. However, detection of the motif leads to computational problems whose solutions require a substantial amount of time in one uniprocessor machine and thus, remains as one challenging problem. In this chapter, two parallel algorithms are proposed, along with its implementation detail which crucially enhances the performance of the PMSP motif search algorithm. The first approach enhances the existing algorithm by eliminating the redundant process of the computation and also, minimizes the execution time by the use of both process-level and thread-level parallelism in the implementation. The second approach is the improvement over the first one, where not only the time of computation is reduced further but also the best space utilization is achieved..

***Keywords* : Planted Motif Search, Process-level parallelism, Thread-level parallelism, Bit Vector Map.**

I. INTRODUCTION

In computational molecular biology and bioinformatics, the discovery of motif, the approximate patterns in DNA sequences has given rise to important solutions in the domain of many biological problems. Motifs are the sequence of common patterns present in transcription factor binding sites that exhibits eventful character in gene regulation and expression, understanding of human diseases, drug design, gene function, etc. As a result, in biological studies, the discovery of motifs considered as important as well as one fundamental problem.

In literature, extensive study has made on several categories of the motif search problem.

For instance, they include Simple Motif Search (SMS), Edit distance based Motif Search (EMS), and Planted (*l*, d) Motif Search(PMS). In this work, our focus is on Planted (*l*, d) Motif Search (PMS) problem.

Revised Manuscript Received on February 28, 2020.
***** Correspondence Author
Satarupa Mohanty*, Associate Prof., School of Computer Engineering, KIIT University, Bhubaneswar, India. **satarupafcs@kiit.ac.in,**
Biswajit Sahoo, Professor, School of Computer Engineering, KIIT University, Bhubaneswar, India. **bsahoofcs@kiit.ac.in**

The definition of PMS problem is as follows: Inputs to the PMS are n sequences of length m each, two positive integers *l* and d. The objective is to extract strings of length *l*, such that any such string M is present in all n sequences with maximum d mismatches.

PMS algorithms are categorized into two approaches: exact and approximation algorithms, based on heuristic search and exhaustive enumeration search respectively. Generally, PMS approximation algorithms are more widespread and faster compared to exact algorithms, however, there no assurance of getting always the correct motif.

The probabilistic approach [1], [2], [3], [4] is utilized by the approximate algorithm, which depends on the Position-Specific Scoring Matrix (PSSM) representation [5], or a combinatorial approach [6], [7], [8], [9], [10]. Extensively used analytical algorithms are the stochastic GibbsDNA algorithm [1], AlignACE [2], PhyloGibbs [3], BioProspector [4], TEIRESIAS [6], WINNOWER [7], Random Projection [8], MULTIPROFILER [9], Pattern Branching [10], CONSENSUS[11], MEME [12], MCEMDA[13] and Vine [14]. The WINNOWER algorithm, by Pevzner and Sze[7] builds a graph considering the *l*-mers as nodes and connection of the pair of *l*-mers as the edge, where the pair differs at maximum 2d distance. Then he systematizes PMS into the problem of figuring out a large clique, an NP-Hard problem. Random projections [8], by Buhler and Tompa, grouped *l*-mers based on the similarity of the projections by considering k locations from the entire *l*-mers. The probability of obtaining the desired motif is more in the groups having a maximum number of *l*-mers. Pattern Branching [10] algorithm uses the scoring method to assign a score to each neighbor of all *l*-mers available in the sequence and then the best-scored neighbors are determined by the local search. The greedy CONSENSUS [11] employs statistical measures for the alignment of *l*-mers and finds probable motifs from the alignment while GibbsDNA [1] uses Gibbs sampling.

MEME (Multiple Expectation-Maximization Elicitation) [12] is among the widely used approximation algorithms, where the technique of expectation minimization is used. A Monte Carlo algorithm, MCEMDA[13], initiates from a starting model and then it repeatedly executes the Monte Carlo simulation with parameter update till the convergence. Vine [14], a polynomial time heuristic method based on WINNOWER[7]. Exact algorithms take exponential time to compute, but there is a guarantee of getting motifs.

For the NP-hard nature of the exact algorithm, it is impractical to run on a very larger instance and consequently required to develop the exact algorithm with shorter time and space overhead. Based on search space the exact algorithms built on two approaches, namely: sample-driven and pattern-driven.

The sample-driven approach generates the common neighbor by experimenting (m-*l*+1)n achievable combinations of *l*-mers of distinct strings. Pattern-driven approach verifies all possible 4^l patterns to target a particular pattern that available in every sequence with the least possible deviation. Some of the exact algorithms for solving PMS are SMILE [15], CENSUS [16], MITRA [17], PMS1 [18], PMS2 [18], Voting [19], and RISOTTO [20], three algorithms namely PMSi, PMSP, and PMSprune by Davila et al. [21], Pompa[22], PMS4[23]. Many of these approaches use a tree data structure like a suffix tree or a mismatch tree or tries to steadily generate motifs one character at a time. CENSUS [16] constructs a trie out of *l*-mers of every sequence. In the process of the trie generation, the nodes go on storing its distance value from the motif, which helps it to potentially prune several branches of the trie. The algorithms SMILE[15] and RISOTTO[20] uses the suffix trees. Among this RISOTTO [20] performs efficiently and uses the suffix tree. MITRA [17] considers the mismatch tree data structure. He divides the space of patterns into several sub-spaces those starts with a stated prefix and then employs pruning to each of those sub-spaces. PMS1, PMSi, PMS2 [18] are built on the radix sort. They intersect efficiently the sorted d-neighborhood *l*-mers of all in the input sequences using a unique technique. Voting [19] uses the hash table to store all possible *l*-mers and thus the space requirement is huge for larger instances. Davila et al. [21] proposed a series of PMS algorithms that are fast as well as comparatively economical on space. PMSP [21] explores the d-neighborhood *l*-mers of the initial input sequence and then applies an extensive search among the rest of the input sequences to compute the particular *l*-mers in the found d-neighborhood called as motifs. Practically it performs better than the PMS1 instead of its worse theoretical worst-case time complexity. PMSprune [21] uses the branch and bound approach to explore all d-neighborhood and uses dynamic programming to compute the distance among them. Pampa [22] added the concept of wildcard characters over PMSprune to compute approximate motif patterns and then find the actual motifs by doing a thorough mapping within the computed approximate pattern. PMS4 [23], a generic speedup approach that can improve the execution time of any exact algorithms. The algorithm PMS5 [24], PMS6[25] and qPMS7[26] compute all the common neighbors of the three *l*-mers iteratively using an integer linear programming formulation. PairMotif[27] selects a candidate *l*-mer from the input string and then alter the alphabets one by one. Some algorithm utilizes an efficient combination of sample-driven and pattern-driven approach such as the algorithm of [28], PMS8 [29] and qPMS9[30].

From the intensive survey of discovering planted (*l*, d) motif, the observation is that Rajsekharan's PMSP algorithm is the pedestal of his series of algorithms from PMS5 to PMS9. However, despite the efficiency of PMSP, its inexhaustible memory consumption leads to a failure in the implementation. In this paper, we work more efficaciously by developing two parallel algorithms that minimize the motif algorithm's search space as well as search time. The first approach enhances the existing algorithm by eliminating the redundant process of the computation and also, minimizes the execution time by the use of both process-level and thread-level parallelism in the implementation. The second approach is the improvement over the first one, where not only the time of computation is reduced further but also the best space utilization is achieved.

This paper is structured into eight sections. Section 2 describes the existing sequential exact PMSP algorithm with its limitation in section 3. The motivation for the proposed work is addressed in section 4. The proposed parallel algorithm along with its implementation detail is described in section 5. Section 6 delineates the improvement over the proposed parallel algorithm with its implementation. Performance evaluation and discussion on the experimental results are carried out in section 7. Finally, the chapter concludes in section 8.

II. EXISTING PMSP ALGORITHM

To describe the proposed parallel PMSP algorithm systematically, this section presents some definition of the related concepts and the existing planted motif search algorithm relied on these concepts. The simple idea followed by the PMSP is as follows: for individual length *l* substring x of sequence S_1, it generates the set of d-neighborhoods and outputs the *l*-mer of this neighborhood that are present in every sequence S_i for i=2,..,t within d Hamming distance.

Definition 2.1. Given a set of t DNA sequences S={S_1, ..., S_t} of length n each, over an alphabet of interest Σ = {A, G, C, T} and two nonnegative integers *l* and d such that $0 \le d < l < n$, the planted (*l*, d) motif search is defined to search a string z with $|z| = l$ such that each DNA sequence S_i contains at least a substring z_i with $|z_i| = l$ and z differs from z_i by at most d positions.

Definition 2.2. Given are two strings s and x of length n and *l* respectively, with $l < n$. String x is said to be an *l*-mer of s, if x is a length *l* substring of s and is denoted as $x <_l s$.

Definition 2.3. Given are two strings x and y with $|x| = |y| = l$, the Hamming distance between them is denoted by $d_H(x, y)$, which is the number of mismatches between them.

Definition 2.4: Given a string x[1,..,*l*], the d-neighborhood of it can be denoted as $B_d(x)$, where $B_d(x)$={y: $|y| = l$ and $d_H(x, y) \le d$}.

If Σ is the alphabet of interest then note that $|B_d(x)| = N(l,d) = \sum_{i=0}^{d} \binom{l}{d} (|\Sigma|-1)^i$

Definition 2.5: Given are two strings s and x of length n and *l* respectively, with $l < n$. Let C(x,s) be the collection of the *l*-mers y of string s such that $d_H(x, y) \le 2d$ or C(x, s) ={y: $x <_l s$ and $d_H(x, y) \le 2d$}

It is inferred that, for any motif M, if there exist two instances x and y, then they will differ by maximum 2d Hamming distance. Here, to search the motif M derived from *l*-mer x, the set C(x,s) will only contain the *l*-mer of sequence s which differs at the maximum 2d distance.

Definition 2.6: Given a set of sequences $S=\{S_1,S_2,\ldots,S_t\}$ of length n each, then the set of potential (*l*, d) motifs of S is denoted by $M_{l,d}(S)$.

For the given *l*, d, and S the aim of PMS is to determine $M_{l,d}(S)$.

Algorithm PMSP

Input: Integers t, n, *l*, d and t number of DNA sequences as S.
Output: Potential motif set $M_{l,d}(S)$
PMSP (t, n, *l*, d, S)

1. $M_{l,d}(S) = \emptyset$ // Ø symbolizes the empty set
2. For every x, where $x <_l S_1$ do
3. For every S_i, where $2 \le i \le t$ do
4. $C(x, S_i) = \emptyset$
5. If (($y <_l S_i$) && ($d_H(x, y) \le 2d$)) then
6. $C(x, S_i) = C(x, S_i)$ Y y
7. For every $z \in B_d(x)$ do
8. If (for every i, where $2 \le i \le t$, there exist $y \in C(x, S_i)$ such that $d_H(z, y) \le d$) then
9. $M_{l,d}(S) = M_{l,d}(S)$ Y z

Return $M_{l,d}(S)$.

In the pseudocode, lines 3 to 6 collect the *l*-mers for set $C(x, S_i)$ by calculating the Hamming distance of *l*-mer x of sequence S_1 from each *l*-mer of sequence S_i for $2 \le i \le t$. Line 7 and 8 check the validity of each neighbor z, for being an (*l*, d) motif. Line 9 includes all motifs into the potential motif set $M_{l,d}(S)$ and line 10, finally, output all potential planted motif satisfying the definition of planted (*l*, d) motif. .

III. LIMITATION

1. T This algorithm reasonably works for 20 number of DNA sequences with challenging (*l*, d) instances as (11,3), (13,4), (15,5), (17,6). However, discovering motifs for the PMS problem with larger Hamming distance d with longer *l* value becomes impractical.
2. With the increase in d value, the volume of d-neighborhoods yields from every *l*-mer of sequence S_1 will increase tremendously, which will become unserviceable with the use of any list data structure.
3. To realize the line 8, the sequential search of PMSP will become tedious with the larger in *l* and d values.
4. It is assumed that the likelihood of the appearance of two *l* -mers at a different position of the string S_1 is different from each other. This may not be happening in a larger string. Again when two *l* -mers are in close proximity, then duplicate neighbors can be generated. In PMSP, there is a lack of provision of expelling these duplicate or redundant *l*-mers generated in the process of motif search and thus there is a wasted effort taken place for such type of *l*-mers.

IV. MOTIVATION

The exhaustive survey of the computational methods of discovering planted (*l*, d) motifs are performed and also their biological significances are studied thoroughly. The efficiency is found to be minimal in the serial implementation of the exact solution by a single-core computer. The existing algorithm contains lots of repeating as well as data-independent operations. This motivates us to propose a parallel way of implementing the existing exact planted (*l*, d) motif search algorithm with some modification to avoid repeated operations. Additionally, a larger memory is also available with the multiprogramming environment to store the huge number of generated *l* -mers.

In parallelization, with a multicore processor, the issue of sharing cache for different core simultaneously endangers with enough memory traffic. This can be resolved by designing one algorithm where the different core will operate on the separate memory areas. Again, employing abstraction of assigning separate memory area for the shared data to be used by different core will cause the dispute of "false sharing" which can be resolved through a multithreading programming approach.

To address the problem of the existence of the duplicate or redundant *l*-mer, the bit vector approach is used where for every *l*-mer there will be a corresponding bit present in response to its availability. Again, the use of the bit vector reduces the time of searching and time of merging in the process of solving the problem.

V. PROPOSED PARALLEL ALGORITHM

In the proposed parallel procedure, some additional features are used which improves the functioning of the existing algorithm. Firstly, the parallelism present in the PMSP algorithm is exploited and secondly, the bit vector mapping technique is employed to avoid the ambiguous and repeated operations. The proposed algorithm is as follows.

Algorithm 1. Proposed Parallel Algorithm

Input: integers t, n, *l*, d and t number of DNA sequences as S
Output: Potential motif set $M_{l,d}(S)$
Proposed_parallel_PMSP (t, n, *l*, d, S)

1. $M_{l,d}(S) = \emptyset$, $M_c = \emptyset$ // Ø symbolizes the empty set
2. For every $x <_l S_1$ do parallel using process
   Par Begin
3. For every S_i, $2 \le i \le t$, do parallel using thread
4. $C(x, S_i) = \emptyset$
5. If (($y <_l S_i$) && ($d_H(x, y) \le 2d$)) then
6. $C(x, S_i) = C(x, S_i)$ Y y
7. Find $B_d(x)$
   Par End
   Par Begin
8. For every $z \in B_d(x)$ and $z \notin M_c$ do parallel using thread
9. Mc = Mc Y z

10. For every i, where $2 \le i \le t$, if (there exist $y \in C(x, S_i)$ such that $d_H(z, y) \le d$) then
11. $M^{th}_{l,d}(S) = M^{th}_{l,d}(S)\ \Upsilon\ z$
// Motif set for each thread
Par End
12. $M^{P}_{l,d}(S) = \Upsilon\ M^{th}_{l,d}(S)$ //Motif set for each process
13. $M_{l,d}(S) = \Upsilon\ M^{P}_{l,d}(S)$ //Synchronize the results of the process
14. Return $M_{l,d}(S)$.

A. Implementation of Proposed Parallel Algorithm

The server uniformly distributes the task among the clients and does the thoroughgoing coordination and synchronization among the clients. Figure 1 depicts the flow of the proposed algorithm. The client accepts the task from the master process and implements it in a parallel way. The SSh key technique is used to avoid the requirement of a password during the interaction between the master system and client (slave) systems for the execution of the program. This helps easy and fast communication between systems.

All the given sequences S_i for $2 \le i \le t$ are evenly distributed among the processors along with the portion of the sequence S_1. Each process will determine the possible planted (*l*, d) motifs from the allotted subset independently. Every *l*-mer is equalized to a unique integer and is represented as a binary number. If $l_1, l_2, \ldots, l_l$ is an *l*-mer, then it is expressed as $(r_1, r_2, \ldots, r_l)$, a sequel of residues. For $|\Sigma| = 4$, two bits are sufficient to represent the residues of the *l*-mer. Assuming A=0, G=1, C=2, T=3, residues A, G, C, T are represented as 00, 01, 10, 11 respectively. Thus any *l*-mer can be considered as an array of residues and can be derivable to a unique integer in a natural way. For illustration, if $l = 15$ and the word size of an integer in accordance with the machine of interest is of 32bits, then every *l*-mer can act as a unique integer. In the proposed algorithm, the bit vector instead of the list is used. For every possible *l*-mer of length *l*, there is a corresponding bit in the bit vector. The bit represents whether the *l*-mer is present (1) or not (0). For a fixed-length *l*, increasing d gives rise to increase the list tremendously in the existing PMSP algorithm, whereas in the proposed algorithm the bit vector size will remain the same irrespective of the size of d.

In the proposed algorithm, line 2 splits the *l*-mers x of sequence S_1 into several subsets, and allocate each subset to each process. For every l-mer x of the sequence S_1, Line 3 to 7, constructs the lists $C(x, S_i)$ containing the *l*-mers of S_i for $2 \le i \le t$, those are present by Hamming distance 2d from x and also, constructs the d-neighborhoods of l-mers x in parallel using thread. Line 8, 9 and 10 jointly find the neighborhoods z of *l*-mer x, which is present in each of the list $C(x, S_i)$ for $2 \le i \le t$ within a d Hamming distance, in parallel using threads and in affirmative, include that z into the bit vector $M^{th}_{l,d}$ in line 11. Using bitwise OR operation, line 12 merges the bit vectors $M^{th}_{l,d}$ of every thread to get the processor's bit vector $M^{p}_{l,d}$ and line 13 merges the bit vectors $M^{p}_{l,d}$ of every processor to get the final bit vector. Finally, line 14 outputs the desired motif.

VI. PROPOSED PARALLEL ALGORITHM

This section improves the proposed parallel algorithm in crucially as follows.

1. The use of (t-1) number of lists, $C(x, S_i)$, is replaced with the introduction of a single bit vector M_p of size in several bytes. The content of $C(x, S_i)$ in the bit vector, now will correspond to the segment of the bit vector M_p, as $M_p[(i-1)\times r, \ldots, i\times r-1]$ where $r = \frac{n-l+1}{32} + 1$ assuming an integer size to be of 32bits. For every *l*-mer of S_i, there is a corresponding bit in the segment $M_p[(i-1)\times r,\ldots,i\times r-1]$ which can be set (1) or reset (0) depending on their Hamming distance value from *l*-mer x. This implies that, if any *l*-mer y of S_i is within Hamming distance 2d from *l*-mer x, then the corresponding bit of M_p will be set to one otherwise it will remain zero.
2. Processors' communication overhead is eliminated using thread-level parallelism only. To achieve this, two levels of parallelism are used, one outer level and another inner level. The outer level parallelism exploits the parallel motif search on different *l*-mers of the first sequence and the inner level parallelism exploits the parallel motif search of the different d-neighborhood of the sequence one *l*-mer.

The proposed algorithm is described as follows. (Note: For the convenience, the set of sequences S is considered as $\{S_i\}_{i=0}^{t-1}$)

Algorithm 2 Improved Parallel PMSP

Input: integers t, n, *l*, d and a set of S of t input sequences S = $\{S_i\}_{i=0}^{t-1}$.
Output: Potential motif set $M_{l,d}(S)$
Improved_parallel_PMSP (t, n, *l*, d, S)

1. $M_{l,d} = \emptyset$, $M_p = \emptyset$, $M_c = \emptyset$, $r = \frac{n-l+1}{32} + 1$
2. For each $x <_l S_1$, do parallel using outer thread T[out]
3. For each i, $1 \le i \le t-1$ do
4. If $((y <_l S_i)$ && $d_H(x, y) \le 2d)$ then
5. $M_p[(i-1)\times r, \ldots, i\times r-1] = M_p[(i-1)\times r, \ldots, i\times r-1]\ \Upsilon\ y$
6. Find $B_d(x)$
7. For each $((z \in B_d(x))$ && $(z \notin M_c))$ do parallel using inner thread T[out][in]
8. $M_c = M_c\ \Upsilon\ z$
9. For each i, $1 \le i \le t-1$ do
10. flag = 0
11. For each $y \in M_p[(i-1)\times r, \ldots, i\times r-1]$
12. If$(d_H(z, y) \le d)$ then
13. flag=1
14. break;
end for
15. If(flag = 0) then
16. break;
17. else if(i = t-1) then

18. $M_{l,d}(S) = M_{l,d}(S)\ Y\ z$
19. end for
20. Output $M_{l,d}(S)$

A. Implementation of the Improved Parallel Algorithm

The implementation of the improved parallel algorithm is realized using the pthread library running on the Linux environment over the Intel 4-core machine where the individual core runs at 2.4GHz. Figure 2 depicts the flow of the proposed algorithm. The parallelism present in the improved algorithm is exploited through two levels (outer and inner). Only the additional work made concerning the proposed parallel PMSP approach is described here. Initially, line 2, using the thread block T[out], achieves the outer level parallelism on motif search due to different *l*-mers of sequence S_0. Then, line 5, using multiple threads T[out][in], achieves the inner level parallelism on motif search due to the different neighbors of *l*-mer x. The total number of threads depends on the availability of both memory and number of cores in the system. Each outer thread T[out] has its own inner threads T[out][in]. The outer threads cooperate to find the motifs for different *l*-mers x of sequence S_0 and inner threads cooperate to find the motifs for neighbors of a particular *l*-mers of S_0. In the improved parallel algorithm, a single bit vector is used instead of several lists C(x, S_i). For every possible *l*-mer, there is a corresponding bit in the bit vector that can be set to specify the availability of the *l*-mer. Three numbers of bit vectors M_p, M_c, and $M_{l,d}$ are used in the algorithm. Bit vector M_p stores the *l*-mers of S_i for $1 \leq i \leq t\text{-}1$ in the segment $M_p[(i\text{-}1)\times r, \ldots, i\times r\text{-}1]$, those are within 2d Hamming distance from the *l*-mer x of sequence S_0, the bit vector M_c keeps the record of the *l*-mers those are encountered in the process of motif search and the bit vector $M_{l,d}(S)$ stores the resultant motifs. The purpose of the bit vector M_p is to minimize the storage space of different lists C(x, S_i) into a size, in several bytes only and to avoid any redundant operation in the process of motif search. The break statement in the line 14 and line 16, restrict the unnecessary loop iteration to occur, which subsequently minimizes the computation process.

VII. EXPERIMENTAL RESULTS AND DISCUSSION

The effectiveness of the proposed parallel model and improved parallel model is examined through the computational experiments. Twenty random sequences each of consisting 600 are considered. A planted motif of length *l* is then mutated randomly in d selected positions. Despite the fact of the analogous behavior of the proposed algorithm with the existing algorithm in the worst case, the proposed algorithm outperforms the existing one in all other cases.

A. Performance Analysis of Parallel Algorithm

The proposed parallel algorithm has been implemented in the C environment using the MPI library and the Pthread library. The experiment has performed in an environment of two nodes in clusters with each of 2.4GHz Intel Pentium-IV processor having 4GB RAM running under Red Hat Linux. One Gigabits/s Ethernet switch is used to connect the nodes. The parallel algorithm has been examined on all nodes and the starting and ending execution time has been evaluated with the wall clock time. The running time of the program is the sum of its execution time, communication delay and time of fetching input data from the file. The algorithm allocates the sequence pairs based on the number of symmetric multiprocessor (SMP) nodes available. Due to the LAN connection, the distance of transferring data or intermediate results between the server and the clients can be ignored.
The running time of PMSP and the proposed parallel algorithm on different clusters is depicted in Table1.

The running time of planted (*l*, d) motif algorithm derived from data set having variable motif length and a variable number of Hamming distance is discussed below. The proposed parallel algorithm is carried out by creating two numbers of processes on a two-node cluster. This distributes one process to each node. In this experiment, motif lengths are (11,3), (13, 4), and(15,5) for 20 sequences of each 600 alphabets drawn from an alphabet set $\sum = \{A, G, C, T\}$.

The first set of experiments is aimed at observing how the length of the planted motif affects the performance of the proposed parallel algorithm compared to the existing PMSP sequential algorithm as shown in Figure 3. It is observed in all cases that correct planted motifs are found using the proposed parallel algorithm successfully.

The next experiment investigates how the processing time is influenced by varying Hamming distance for the length of the motif 11, 13, 15 as shown in Figures 4, 5, and 6 respectively. Running time of a parallel algorithm on a two-node cluster is discussed with two nodes and one node, with the increasing Hamming distance. Curves indicate that the processing time has linear growth as the number of processors increase.

B. Performance Analysis of Improved Parallel Algorithm

The improved parallel algorithm is implemented with a different degree of outer level and inner level parallelism for different motif instances. When the value of *l* is small, let for *l* =11 or 13 the performance is constrained by the memory bandwidth of the computer. The increase in the degree of parallelism has less effect on the execution time due to the stalls of threads for memory access. However, for the larger *l* value, the availability of the system memory decides the number of thread blocks. Table 2 indicates the degree of outer level and inner level parallelism for different challenging instances with their run time. The algorithm is designed in such a way that between the threads the work to be done is distributed uniformly. In this experiment, the motif of length (11,4), (13,4), (15,5), (16,6) and (17, 6) has been considered for 20 numbers of sequences of length 600 each.

Figure 7 shows how the lengths of the planted motif affect the performance of the improved parallel PMSP algorithm compared to the proposed parallel algorithm and the existing sequential PMSP algorithm. It is observed that in all the cases the correct planted motifs are found using the proposed parallel algorithms successfully.

VIII. CONCLUSION

In computational biology, accuracy and high-performance computing are challenging as well as an important issue. In this work, the design, implementation, and analysis of two parallel algorithms are discussed.

One is a parallel model to PMSP with some additional features to accelerate the process of search and another is further improvement over the proposed model.

In the simple parallel approach, the use of both coarse grain and fine grain parallelism justifies the claim of the proposed parallelization method on the SMP cluster system improves over the existing sequential algorithm. The proposed algorithm is implemented on two nodes SMP cluster system with each of 2.4GHz Intel Pentium-IV having 4 GB RAM running under Red Hat Linux. The bit vector implementation minimizes the storage of a huge number of *l*-mers generated during intermediate steps and also encourages the result of motifs for the higher length of larger Hamming distance. In improved parallel PMSP, multithreading parallelism is used by adding some modifications to the proposed parallel PMSP. The use of the bit vector in this approach reduces the memory requirement further by eliminating the use of multiple lists by a single bit vector of size in several bytes only and also, reduces the computation time by omitting the redundant process in motif search. The use of bit vector for storage and parallelization (both coarse grain and fine grain) turns it into space and cost-optimal.

REFERENCES

1. G. O. Young, "Synthetic structure of industrial plastics (Book style with paper title and editor)," in *Plastics*, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.
2. C. Lawrence, S. Altschul, M. Boguski, J. Liu, A. Neuwald, and J. Wootton, " Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment, " Science, vol. 262, 1993, pp.208-254,
3. F.P. Roth, J.D. Hughes, P.W. Estep, and G.M. Church, "Finding DNA Regulatory Motifs within Unaligned Noncoding Sequences Clustered by Whole-Genome mRNA Quantization", Nature Biot echnology, vol. 56, 1998, pp.939-945.
4. R. Siddharthan, E.D. Siggia, and E. van Nimwegen , "Phylogibbs: A Gibbs Sampler Incorporating Phylogenetic Information, " PLoS Computational Biology, vol. 5, 2005, pp. 534- 556.
5. X. Liu, J.S. Liu, and D.L. Brutlag, "BioProspector: Discovering Conserved DNA Motifs in Upstream Regulatory Regions of Co-Expressed Genes, " Proceeding of Pacific Symposium Biocomputing, 2005, pp. 527.
6. P.A. Evans, A. Smith, and H.T. Wareham, " On the Complexity of Finding Common Approximate Substrings" , Theoretical Computer Sc., vol. 306, 2003, pp 407-430.
7. I. Rigoutsos and A. Floratos, " Combinatorial Pattern Discovery in Biological Sequences: The TEIRESIAS Algorithm", Bioinf., vol. 54, 1998, pp. 56-57.
8. P.A. Pevzner and S.H. Sze, " Combinatorial Approaches for Finding Subtle Signals in DNA Sequences" , Intelligent Sys. For Molecular Biology, 2000, pp. 269-278.
9. J. Buhler and M. Tompa, " Finding Motifs Using Random Projections, " J. Computational Biology, vol. 9, 2002, pp. 125-242.
10. U. Keich and P.A. Pevzner, " Finding Motifs in the Twilight Zone," Bioinformatics, vol. 4, 2002, pp. 374-385.
11. A. Price, S. Ramabhadran, and P. Pevzner, "Finding Subtle Motifs by Branching from Sample Strings," Proceeding. 2nd European Conf. Comput. Biology (ECCB 03), Bioinformatics, suppl. ed., 2003, pp. 1-7.
12. G.Z. Hertz and G.D. Stormo, " Identifying DNA and Protein Patterns with Statistically Significant Alignments of Multiple Sequences," Bioinformatics. Vol. 55, 1999, pp. 563-577.
13. T. Bailey and C. Elka, " Fitting a Mixture Model by Expectation Maximization to Discover Motifs in Biopolymers" , Proc. Second International Conf. Intelligent System.
14. Bi CP A monte carlo EM algorithm for De Novo motif discovery in bio molecular sequences. IEEE/ACM Trans. on Computational Biology and Bioinformatics-6: 2009, pp. 370– 386.
15. CW Huang, WS Lee, SY Hsieh An improved heuristic algorithm for finding motif signals in DNA sequences. IEEE/ACM Trans. on Computational Biology and Bioinformatics, 2011, pp. 959– 975.
16. L. Marsan and M.F. Sagot, "Extracting Structured Motifs Using a Suffix Tree Algorithms and Application to Promoter Consensus Identification " , Fourth Annual International Conference on Computational Molecular Biology (RECOMB), 2000.
17. P. A. Evans and A. D. Smith , " Toward optimal motif Enumeration" , Proc. Eighth Intenational Workshop Algorithm and Data structures (WADS 03), 2013, pp. 47-58.
18. E. Eskin and P.A. Pevzner, " Finding Composite Regulatory Patterns in DNA Sequences," Bioinformatics, vol. 4, 2002, pp. 354-363.
19. S. Rajasekaran, S. Balla, and C.-H. Huang, " Exact Algorithms for the Planted Motif Problem," J. Computational Biology, vol. 12(8), 2005, pp. 1117-1128.
20. F.Y.L. Chin and H.C.M. Leung, " Voting Algorithms for Discovering Long Motifs", Proceeding 3rd Asia-Pacific Bioinfor. Conf., 2005, pp. 261-271.
21. N. Pisanti, A.M. Carvalho, L. Marsan, and M.F. Sagot, " RISOTTO: Fast Extraction of Motifs with Mismatches" , Proceeding 7th Latin Am. Theoretical Symp., 2006, pp. 757-768.
22. J. Davila, S. Balla, and S. Rajasekaran, " Fast and Practical Algorithms for Planted(*l*, d) Motif Search," IEEE/ACM Trans. Comp. Biology and Bioinf. vol 4, 2007, pp. 544-552.
23. J. Davila, , S. Balla, and S. Rajasekaran, Pampa: An Improved Branch and Bound Algorithm for Planted (*l*, d) Motif Search, Tech Report, University of Connecticut, 2007.
24. S. Rajasekaran, H. Dinh, A speedup technique for (*l*, d) motif finding algorithms, BMC Research Notes, Vol. 4(54), 2011, 1-7.
25. H. Dinh, , S. Rajasekaran, and V. Kundeti, PMS5: an efficient exact algorithm for the (*l*, d) motif finding problem, BMC Bioinformatics, 2011, Vol. 12(410),.
26. S. Bandyopadhyay, S. Sahni, and S. Rajasekaran, " PMS6: A Fast Algorithm for Motif Discovery," Proc. IEEE Second Int' l Conf. Computational Advances in Bio and Medical Sciences (ICCABS ' 12), IEEE, 2012, pp. 1-6.
27. H. Dinh, S. Rajasekaran, and J. Davila, " qPMS7: A Fast Algorithm for Finding(*l*, d)- Motifs in DNA and Protein Sequences," PLoS One, 2012, vol. 7(7), article e41425.
28. Q. Yu, H. Huo, Y. Zhang, and H. Guo, " PairMotif: A New Pattern-Driven Algorithm for Planted (l, d) DNA Motif Search," PLoS One, 2012, vol. 7(10), article e48442.
29. S.H. Sze, , S. Lu and J. Chen, "Integrating sample-driven and pattern-driven approaches in motif finding. Proceedings of the International Workshop on Algorithms in Bioinformatics, , Springer, Berlin, Germany, 2004, pp. 438-449.
30. M. Nicolae, and S. Rajasekaran, "Efficient sequential and parallel algorithms for planted motif search" BMC. Bioinf., 2014, Vol 15 pp 1-34,.
31. M. Nicolae, and S. Rajasekaran, " qPMS9: An efficient algorithm for quorum planted motif search", MSc Thesis, University of Connecticut, Mansfield, Connecticut, 2015.

AUTHORS PROFILE

Dr. Satarupa Mohanty, is currently working as the Associate Professor, in School of Computer Engineering, KIIT University, Bhubaneswar. Her research area includes: Algorithm Analysis, Parallel Computing and Bioinformatics

Dr. Biswajit Sahoo, is currently working as a Senior Professor, in School of Computer Engineering, KIIT University, Bhubaneswar. His research area includes: Parallel Computing, Bioinformatics and Advanced Computer Architecture.

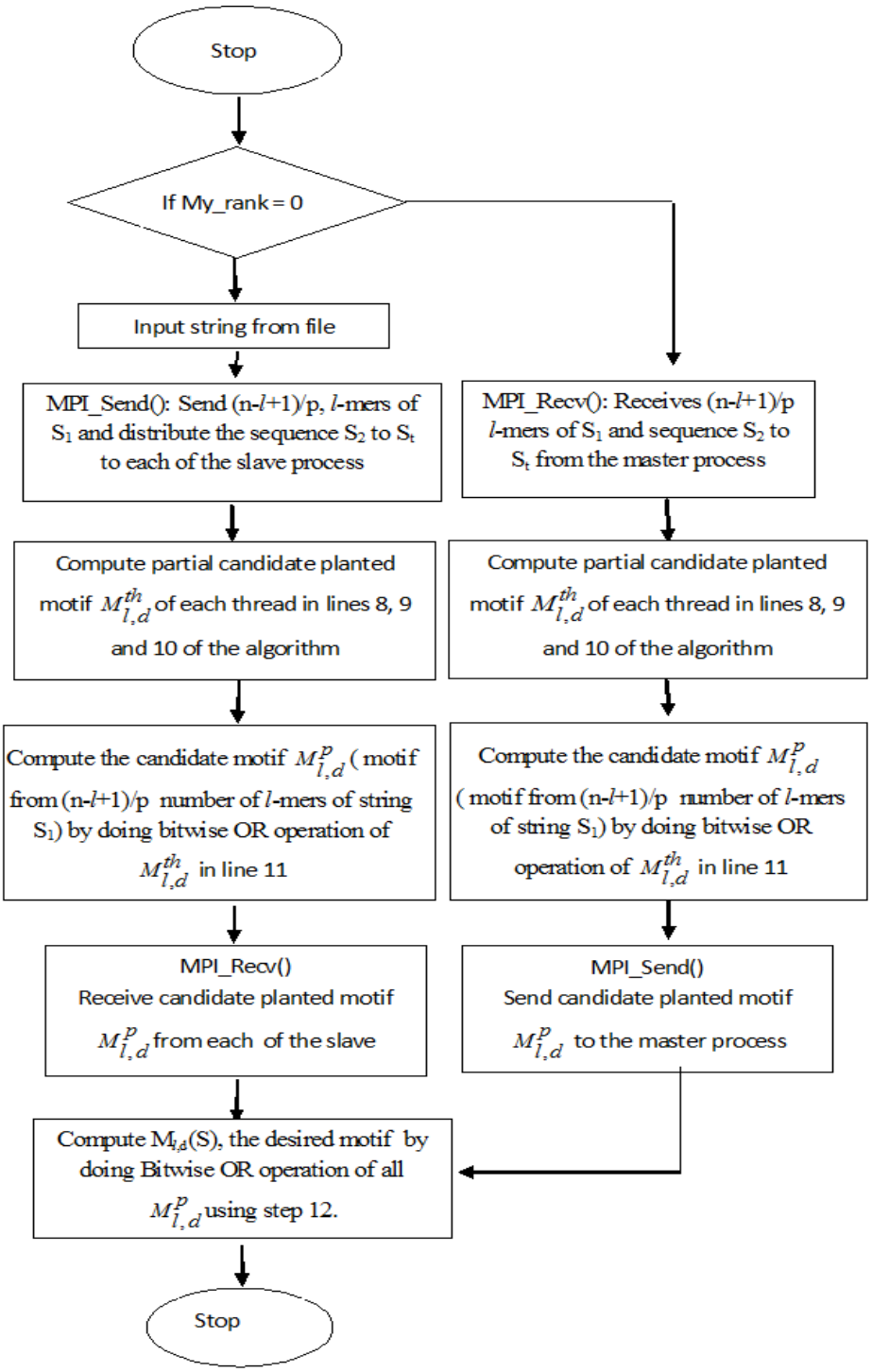

Figure 1: Flow chart depicting the message passing among the master and slaves.

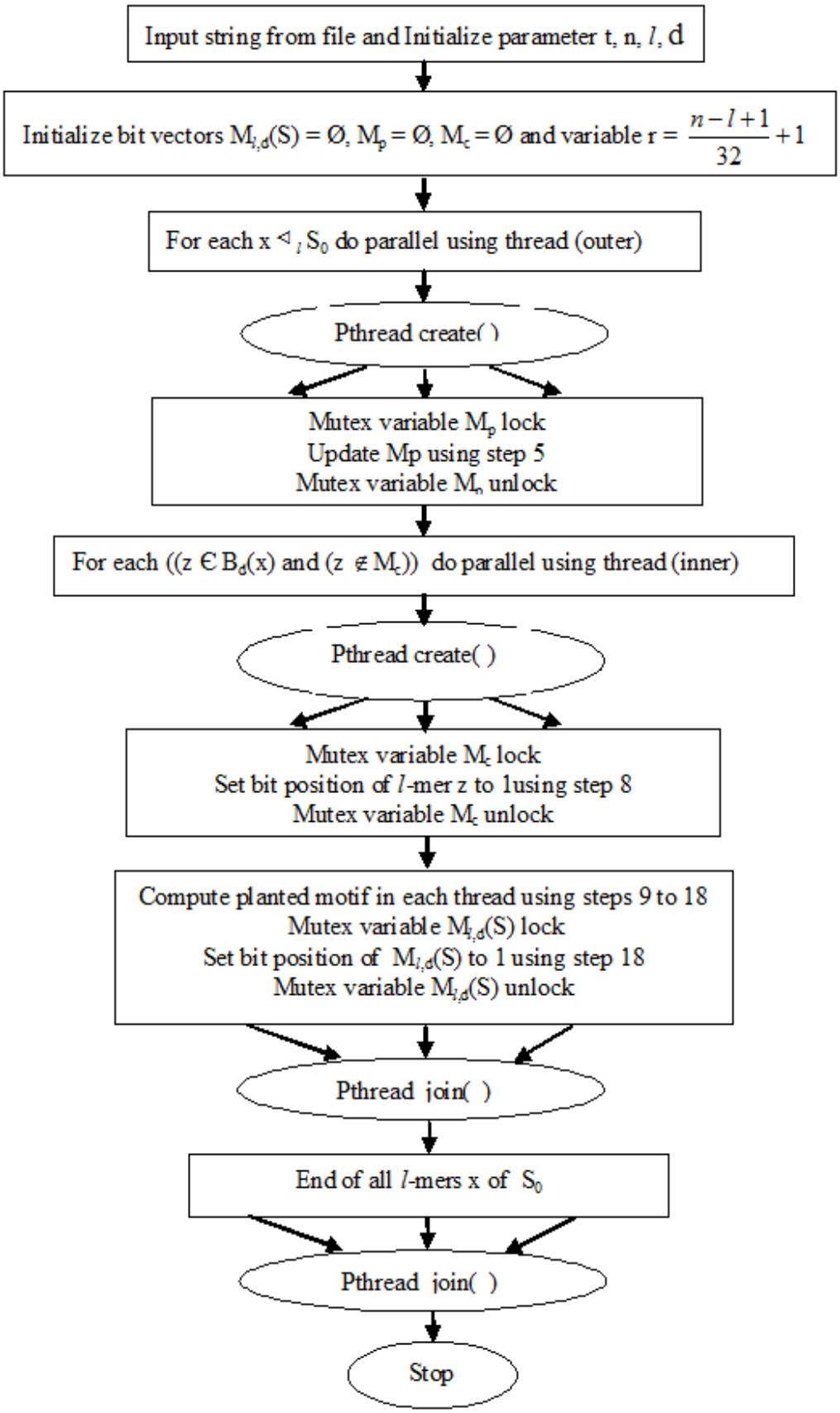

Figure 2: Flow chart showing the synchronization among worker threads and master thread

Table 1: The running time of both PMSP sequential (SA) and the proposed parallel algorithm(PA) on a two node cluster for n=600, and t=20

Motif length (*l*)	Hamming distance (*d*)	*Running time for SA (Sec)*	Running time for parallel algorithm	
			One CPU (*Sec*)	Two CPU (*Sec*)
11	3	*6.9*	4.2	3.1
11	4	-	4.9	3.6
11	5	-	6.2	4.1
13	4	*152*	86	63
13	5	-	104	78
13	6	-	156	123
15	5	*2100*	1092	666
15	6	-	1278	762
15	7	-	1320	852

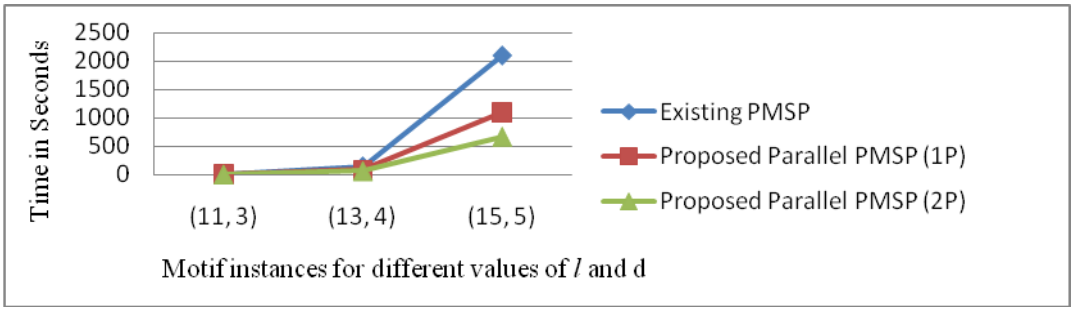

Figure 3: Running time of proposed parallel PMSP algorithms and existing sequential algorithm (SA) for t= 20, and n= 600 as a function of (*l*,d) motifs for different values of *l* and d.

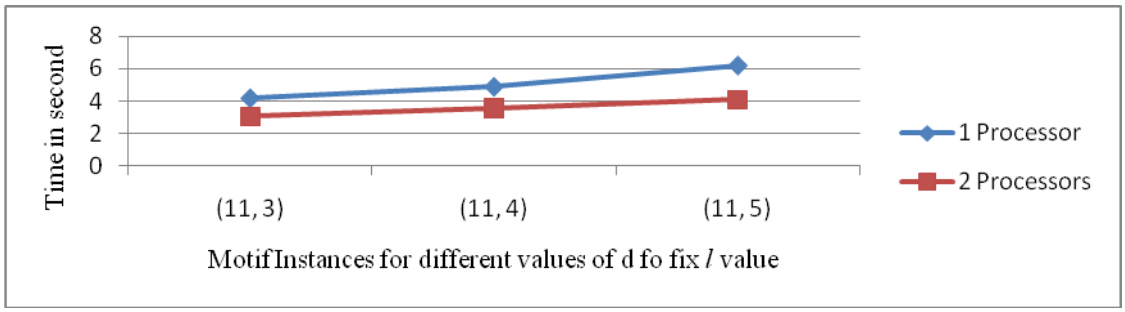

Figure 4: Running time of proposed parallel PMSP algorithm (1P, 2P) for t= 20, n= 600 and *l*= 11 as a function of Hamming distance of motifs.

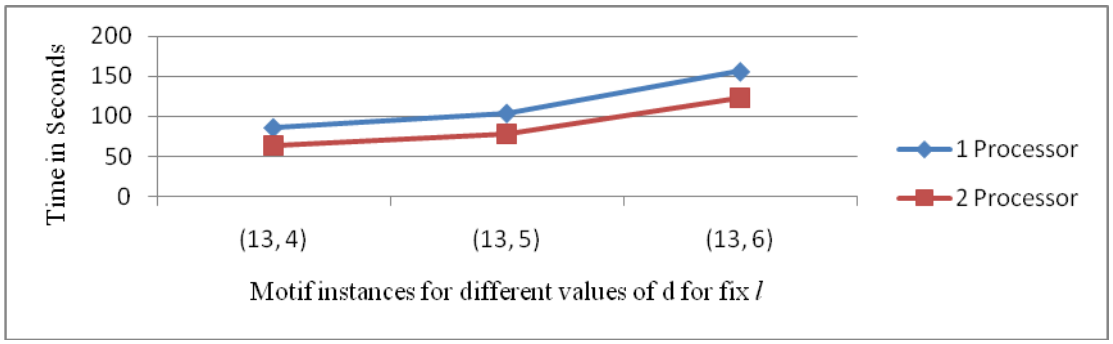

Figure 5: Running time of proposed parallel PMSP algorithm (1P, 2P) for t= 20, n= 600 and *l* = 13 as a function of (*l*, d) motifs for different values of *l* and d.

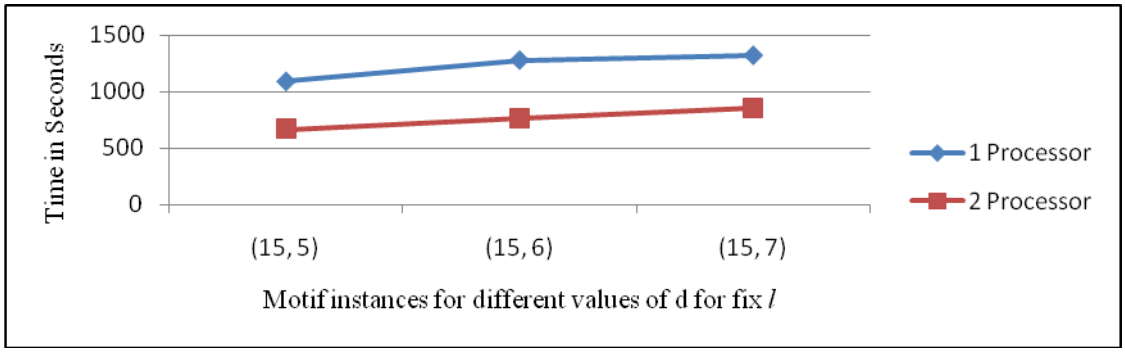

Figure 6: Running time of proposed parallel PMSP algorithm (1P, 2P) for t=20, n=600 and *l* =15 as a function of (*l*, d) motifs for different values of *l* and d.

Table 2: The running time of Improved parallel PMSP, proposed parallel PMSP and the existing PMSP on a two node cluster for n=600, and t=20 with different values of (*l*, d)

Motif instance	Thread blocks	Thread per block	Total thread	Run time of existing PMSP in sec	Run time of proposed parallel PMSP in sec	Run time of improved parallel PMSP in sec
(11, 4)	2	4	8	6.9	3.6	2.3
(13, 4)	2	6	12	152	63	47
(15, 5)	4	4	16	2100	666	378
(16, 6)	2	4	8	-	689	452
(17, 6)	2	6	12	6120	756	468

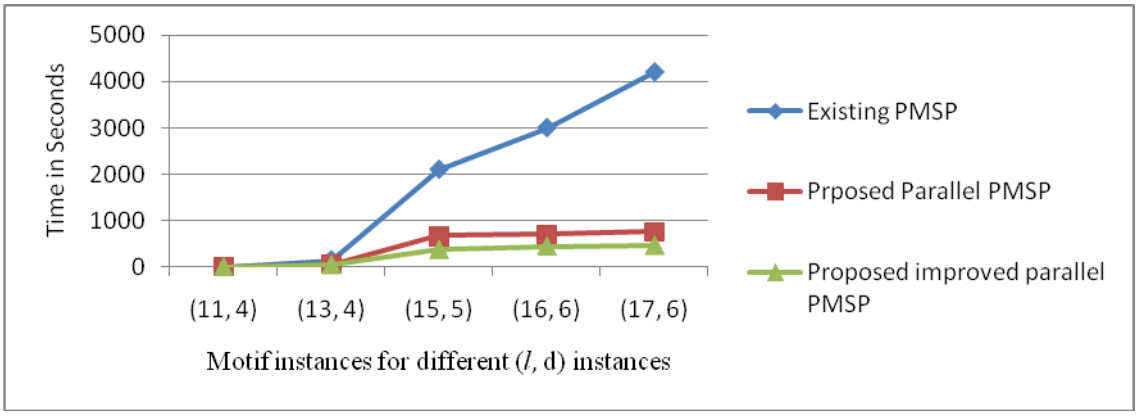

Figure 7: Comparison of running time of improved parallel PMSP algorithm with proposed parallel PMSP and existing PMSP for t= 20, and n= 600 as a function of (*l*,d) motifs for different values of *l* and d.