

Parallelization of Multiple String Matching on a Cluster Platform

Panagiotis D. Michailidis and Konstantinos G. Margaritis

Abstract

This work proposes four parallel methods for multipattern matching which are executed on a heterogeneous cluster. These parallel methods are based on the master - worker paradigm and they implement different partitioning schemes such as static and dynamic load balancing. Furthermore, the parallel methods are analyzed experimentally using the Message Passing Interface (MPI) library on a cluster of heterogeneous workstations. Further, we propose a performance modeling of the parallel methods that can be used to predict the parallel performance on a cluster of workstations. The results by the theoretical performance models have been validated against experimental results of the four parallel methods.

Index Terms

multiple pattern matching, MPI, cluster of workstations, performance modeling

I. INTRODUCTION

STRING matching is one of the main problems in classical string algorithms, with applications to text searching, computational biology, pattern recognition, etc. Given a text string $T_{1\dots n}$ of length n and a pattern string $P_{1\dots m}$ of length m where both the pattern and the text are sequences of characters from Σ with $m \ll n$, we want to find all the text positions where the pattern matches the text.

A natural extension to the basic problem consists of multipattern searching, that is, searching for r patterns $P_1\dots P_r$ simultaneously in order to report all their occurrences. This has also several applications such as virus and intrusion detection, spelling, speech recognition, optical character recognition, handwriting recognition, text retrieval under synonym or thesaurus expansion, computational biology, multidimensional approximate matching, batch processing of single-pattern searching, etc. Moreover, some single-pattern search algorithms resort to multipattern searching of pattern pieces. Depending on the application, r may vary from a few to thousands of patterns. The naive approach is to perform r separate searches, so the goal is to do better. The multipattern problem has received much less attention, not because of lack of interest but because of its difficulty. There exist sequential algorithms that search too few patterns [1], [2], [8], [9]. However, no effective algorithm exists to search for many patterns in a reasonable amount of time.

PC- or workstation-based computing technique provides a computationally cost-effective way to single-pattern search algorithms [5], [6], [12], [7], [4], [3]. In this work, we implement four parallel computing methods to accelerate the multipattern searching, where the Message Passing Interface (MPI) library [10] is used to perform the data communication on a PC-based Linux cluster. These parallel methods are based on master - worker paradigm and these search r patterns using any sequential single-pattern matching algorithm. This work also presents a performance modeling of the four parallel computational methods on a cluster of heterogeneous workstations. We must note that the parallel implementations of multipattern matching problem on a cluster of workstations are not available in the literature.

The rest of this paper is organized as follows: Section II briefly presents heterogeneous computing model and the metrics. Section III presents four parallel implementations and performance analysis for multipattern matching problem. Section IV discusses the experimental and theoretical results of the parallel multipattern implementations. Finally, Section V contains our conclusions.

II. HETEROGENEOUS COMPUTING MODEL

A heterogeneous network (HN) can be abstracted as a connected graph $HN(M,C)$, where

- $M=\{M_1, M_2,\dots,M_p\}$ is set of heterogeneous workstations (p is the number of workstations). The computation capacity of each workstation is determined by the power of its CPU, I/O and memory access speed.
- C is standard interconnection network for workstations, such as Fast Ethernet or an ATM network, where the communication links between any pair of the workstations have the same bandwidth.

Based on the above definition, if a cluster consists of a set of identical workstations, the cluster is homogeneous.

A. Metrics

Metrics help to compare and characterize parallel computer systems. Metrics cited in this section are defined and published in previous paper [13]. They can be roughly divided into characterization metrics and performance metrics.

Panagiotis D. Michailidis and Konstantinos G. Margaritis are with the Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece.

1) *Characterization Metrics*: To compute the power weight among workstations an intuitive metric is defined as follows:

$$W_i(A) = \frac{\min_{j=1}^p \{T(A, M_j)\}}{T(A, M_i)} \quad (1)$$

where A is an application and $T(A, M_i)$ is the execution time for computing A on workstation M_i . Formula 1 indicates that the power weight of a workstation refers to its computing speed relative to the fastest workstation in the network. The value of the power weight is less than or equal to 1. However, if the cluster of workstations is homogeneous then the values of the power weights are equal to 1.

To calculate the execution time of a computational segment, the speed, denoted by S_f of the fastest workstation executing basic operations of an application is measured by the following equation:

$$S_f = \frac{\Theta(c)}{t_c} \quad (2)$$

where c is a computational segment, $\Theta(c)$ is a complexity function which gives the number of basic operations in a computational segment and t_c is the execution time of c on the fastest workstation in the network.

Using the speed of the fastest workstation, S_f , we can calculate the speeds of the other workstations in the system, denoted by S_i ($i = 1, \dots, p$), using the computing power weight as follows:

$$S_i = S_f * W_i, i = 1, \dots, p, \text{ and } i \neq f \quad (3)$$

where W_i is the computing power weight of M_i . So, by equation 3, the execution time of a segment c across the heterogeneous network HN, denoted by $T_{cpu}(c, HN)$, can be represented as

$$T_{cpu}(c, HN) = \frac{\Theta(c)}{\sum_{i=1}^p S_i} \quad (4)$$

where $\sum_{i=1}^p S_i$ is the computational capacity used which is obtained by summing the individual speeds of the workstations. Here, T_{cpu} is considered the required CPU time for the segment.

2) *Performance Metrics*: Speedup is used to quantify the performance gain from a parallel computation of an application A over its computation on a single machine on a heterogeneous network system. The speedup of a heterogeneous computation is given by:

$$SP(A) = \frac{\min_{j=1}^p \{T(A, M_j)\}}{T(A, HN)} \quad (5)$$

where $T(A, HN)$ is the total parallel execution time for application A on HN, and $T(A, M_j)$ is the execution time for A on workstation M_j , $j=1, \dots, p$.

III. PARALLEL IMPLEMENTATIONS OF MULTIPLE STRING MATCHING

We follow the master-worker programming model to develop our distributed multiple string matching implementations under the MPI library [10]. This model consists of a master workstation and a collection of worker workstations. The master workstation is used to partition a given text collection into a set of several smaller subtext collections, distribute them to all worker workstations and collect the local results from the worker workstations. The worker workstations are mainly performed a sequential string matching algorithm on their respective subtext collections. The string matching algorithm can be easily replaced by other string matching algorithms to perform string searching. In this paper, each worker performs the SO string matching algorithm [2]. Hence, the master-worker implementation is general enough to be used in a wide range of string matching applications. Static and dynamic master-worker strategies are implemented and presented in next subsections.

A. Static Master-Worker Implementation

In order to present the static master-worker implementation we make the following assumptions: First, the workstations have an identifier $myid$ and are numbered from 1 to p , second the documents of our text collection are distributed among the various workstations and stored on their local disks and finally the multiple patterns is stored in main memory for all workstations.

The partitioning strategy of this approach is to partition the entire text collection into a number of subtext collections according to the number of workstations allocated. The size of each subtext collection contains $\lceil \frac{n}{p} \rceil + m - 1$ successive characters of the complete text collection. There is an overlap of $m - 1$ pattern characters between successive subtexts, i.e., a redundancy of $p(m - 1)$ characters. Therefore, the static master-worker implementation that is called P1, is composed of six phases. In first phase, each worker reads its subtext collection from the local disk in main memory. In second phase, the master broadcasts the first pattern string to all the workers. In third phase, each worker executes the preprocessing phase of the string matching algorithm [2]. In fourth phase, each worker performs the searching phase of the string matching algorithm to generate the number of occurrences. In fifth phase, the master collects the number of occurrences from each worker. In sixth phase, if there are still any patterns of the set of patterns left, the master broadcasts the next pattern to workers and loops back to the third phase.

The advantage of this simple approach is low communication overhead. This advantage was achieved, a priori, by the search computation, assigning each worker to search its own subtext independently without having to communicate with the other workers or the master. However, the main disadvantage is the possible load imbalance because of the poor partitioning technique. In the other words, there is a significant idle time for faster or more lightly loaded workstations in a heterogeneous environment.

1) *Performance Analysis for Static Master-Worker Implementation:* According to the previous description, the execution time of the static master-worker implementation P1 can be broken up into five terms:

- T_a : It is the I/O time each worker to read its subtext collection with size $\lceil n/p \rceil + m - 1$ characters from the local disk in the main memory. Then, the time T_a is given by:

$$T_a = \max_{j=1}^p \left\{ \frac{\lceil n/p \rceil + m - 1}{(S_{i/o})_j} \right\} \quad (6)$$

where $(S_{i/o})_j$ is the I/O capacity of the heterogeneous network when j workstation is used.

- T_b : It includes the communication time to broadcast the first pattern string to all worker workstations involved in processing of string searching. Therefore, the broadcast transfers m bytes to the other $p - 1$ workstations. The time T_b is given by:

$$T_b = \frac{m}{S_{comm}} \quad (7)$$

where S_{comm} is communication speed.

- T_c : It is the preprocessing time for processing of the pattern string from each worker. For executing the preprocessing phase requires $m|\Sigma|$ steps. Then, the time T_c is given by:

$$T_c = \max_{j=1}^p \left\{ \frac{m|\Sigma|}{(S_{prep})_j} \right\} \quad (8)$$

where $(S_{prep})_j$ is the preprocessing capacity of the heterogeneous network when j workstation is used.

- T_d : It is the string matching time across the heterogeneous cluster. Each worker performs string matching of an m pattern string in a subtext collection with size $\lceil n/p \rceil + m - 1$ characters that requires $(\lceil n/p \rceil + m - 1)m$ steps using the SO string matching algorithm [2]. Then, the time T_d is given by:

$$T_d = \max_{j=1}^p \left\{ \frac{(\lceil n/p \rceil + m - 1)m}{(S_{search})_j} \right\} \quad (9)$$

where $(S_{search})_j$ is the text searching capacity of the heterogeneous network when j workstation is used.

- T_e : It includes the communication time to gather p results resulting from the string matching carried on the subtext collections by p workstations concurrently. Each worker sends back one value (in our case, the number of occurrences). Therefore, the time T_e is given by:

$$T_e = \frac{p}{S_{comm}} \quad (10)$$

where S_{comm} is communication speed.

The total execution time of our static string matching implementation, T_p , using p workstations for r patterns is given by:

$$T_p = T_a + r * (T_b + T_c + T_d + T_e) \quad (11)$$

B. Dynamic Master - Worker Implementation

In this subsection, we implement two versions of the dynamic master-worker model. The first version is based on the dynamic allocation of the subtexts and the second one is based on the dynamic allocation of the text pointers.

1) *Dynamic Allocation of Subtexts:* The dynamic master - worker strategy that we adopted is a known parallelization strategy and is known as "workstation farm". Before we present the dynamic implementation, we make the following assumption: the entire text collection is stored on the local disk of the master workstation. The dynamic master-worker implementation that is called P2 is composed of seven phases. In first phase, the master broadcasts the first pattern string to all workers. In second phase, each worker performs the preprocessing phase of the algorithm. In third phase, the master reads from the local disk, several chunks of the text collection. The size of each chunk is $sb + m - 1$ successive characters, where sb is the optimal block size. The block size is an important parameter which can affect the overall performance. More specifically, this parameter is directly related to the I/O and communication factors. In fourth phase, the master sends the first chunks of the text collection to corresponding worker workstations. In fifth phase, each worker workstation performs the searching phase of the string matching algorithm between the corresponding chunk of the text and the pattern, in order to generate the number of occurrences. In sixth phase, each worker sends the number of occurrences back to the master workstation. In seventh phase, if there are still any chunks of the text collection left, the master reads and distributes the next chunks of the text collection to workers and loops back to the fifth phase. In eighth phase, if there are still any patterns left, the master broadcasts the next pattern string to workers and loops back to the second phase except of the third, fourth and seventh phases. We must note that during the fifth phase each worker stores the subtexts (that are received) in memory so that they are used for searching next patterns. This operation occurs during the broadcast of the first pattern string.

The advantage of this dynamic approach is low load imbalance, while the disadvantage is higher interworkstation communication overhead.

2) *Performance Analysis for Dynamic Allocation of Subtexts*: According to the previous description, the execution time of the dynamic master-worker implementation P2 can be broken up into six terms:

- T_a : It includes the communication time for broadcasting of the first pattern string to all workers involved. The amount of this time is similar to the T_b of the previous implementation.
- T_b : It includes the preprocessing time for processing of the pattern from each worker. The amount of this time is similar to the T_c of the previous implementation.
- T_c : It is the total I/O time to read the text collection into several chunks of size $sb + m - 1$ bytes from the local disk of the master workstation. The sb is the optimal block size. So, the master reads n bytes totally of the text collection. Then, the time T_c is given by:

$$T_c = \frac{n}{(S_{i/o})_{master}} \quad (12)$$

where $(S_{i/o})_{master}$ is the I/O capacity of the master workstation.

- T_d : It is the total communication time to send all chunks of the text collection to all workers. The size of each chunk is $sb + m - 1$ bytes. Then, the total time T_d is given by:

$$T_d = \frac{n}{S_{comm}} \quad (13)$$

- T_e : It is the average string matching time across the heterogeneous cluster. Each worker performs string matching of an m pattern string in a chunk of text with size $sb + m - 1$ characters that requires $(sb + m - 1)m$ steps using the SO string matching algorithm [2]. Then, the time T_e is given by:

$$T_e = \frac{(\frac{n}{sb+m-1} - p)[(sb + m - 1)m]}{\sum_{j=1}^p (S_{search})_j} + \max_{j=1}^p \left\{ \frac{(sb + m - 1)m}{(S_{search})_j} \right\} \quad (14)$$

where $\sum_{j=1}^p (S_{search})_j$ is the text searching capacity of the heterogeneous network when p workstations are used. We include the second max term in the equation 14 which defines the worse case load imbalance at the end of the execution when there are not enough chunks of the text collection left to keep all the workstations busy. This term corresponds to the seventh phase of the P2 implementation.

- T_f : It includes the communication time to receive $n/(sb + m - 1)$ results from all workers. Each worker sends back one value. Therefore, the time T_f is given by:

$$T_f = \frac{n}{S_{comm}} \quad (15)$$

Further, we note that this dynamic implementation in practice there is parallel communication and computation and this reason we take the maximum value between the communication time i.e. $T_d + T_f$ and the computation time i.e. T_e . Therefore, the total execution time of our dynamic string matching implementation, T_p , using p workstations for r patterns, is given by:

$$T_p = T_a + T_b + T_c + \max\{T_d + T_f, T_e\} + (r - 1) * (T_a + T_b + T_e + T_f) \quad (16)$$

3) *Dynamic Allocation of Pointers*: Before we present the dynamic implementation with the text pointers, we make the following assumptions: First, the complete text collection is stored on the local disks of all workstations and second the master workstation has a text pointer that shows the current position in the text collection. The dynamic allocation of the text pointers that is called P3 is composed of seven phases. In first phase, the master broadcasts the first pattern string to all workers. In second phase, each worker performs the preprocessing phase of the algorithm. In third phase, the master sends the first text pointers to corresponding workers. In four phase, each worker reads from the local disk $sb + m - 1$ characters of the text starting from the pointer that receives. In fifth phase, each worker performs the searching phase of the string matching procedure between the corresponding chunk of the text and the pattern in order to generate the number of occurrences. In sixth phase, each worker sends the result back to the master. In seventh phase, if the text pointer does not reach the end of the text, then master updates the text pointers for the next position of next chunks of text and sends the pointers to workers and loops back to fourth phase. In eight phase, if there are still any patterns left, the master broadcasts the next pattern string to workers and loops back to the second phase except of the third, fourth and seventh phases. We must note that during the fifth phase each worker stores the subtexts in memory so that they are used for searching next patterns. This operation occurs during the broadcast of the first pattern string.

The advantage of this simple implementation is that it reduces the interworkstation communication overhead since each workstation in this scheme has an entire copy of the text collection on the local disk. However, this scheme requires more local space (or disk) requirements, but the size of the local disk in parallel and distributed architectures is large enough.

4) *Performance Analysis for Dynamic Allocation of Text Pointers*: According to the previous description, the execution time of the dynamic implementation with the text pointers P3 can be broken up into six terms:

- T_a : It includes the communication time for broadcasting of the pattern string to all workers involved. The amount of this time is similar to the T_a of the previous dynamic implementation.
- T_b : It includes the preprocessing time for processing of the pattern from each worker. The amount of this time is similar to the T_b of the previous implementation.

- T_c : It is same with the time T_d of the P2 implementation but this term includes the total communication time to send all text pointers instead of chunks of text to all workers. Therefore, the time T_c is given by:

$$T_c = \frac{\frac{n}{sb+m-1}}{S_{comm}} \quad (17)$$

where S_{comm} is the communication speed.

- T_d : It is the average I/O time to read the text collection into several chunks of size $sb + m - 1$ bytes from the local disks of the worker workstations. We note that each worker reads from the local disk $sb + m - 1$ characters of the text starting from the pointer that receives. Then, the time T_d is given by:

$$T_d = \frac{\left(\frac{n}{sb+m-1} - p\right)(sb + m - 1)}{\sum_{j=1}^p (S_{i/o})_j} + \max_{j=1}^p \left\{ \frac{sb + m - 1}{(S_{i/o})_j} \right\} \quad (18)$$

where $\sum_{j=1}^p (S_{i/o})_j$ is the I/O capacity of the heterogeneous network when p workstations are used. We include the second max term in the equation 18 which defines the worse case load imbalance at the end of the execution when there are not enough chunks of the text collection left to keep all the workstations busy. This term also corresponds to the seventh phase of the P3 implementation.

- T_e : It includes the average string matching time across the heterogeneous cluster. The amount of this time is similar to the time T_e of the P2 implementation.
- T_f : It includes the communication time to receive the results of the string matching from all workers. The amount of this time is same with the time T_f of the P2 implementation.

We take the maximum value between the communication time i.e. $T_c + T_f$ and the computation time i.e. $T_d + T_e$, since in this P3 implementation there is parallel communication and computation. Therefore, the total execution time of our dynamic string matching implementation, T_p , using p workstations, for r patterns is given by:

$$T_p = T_a + T_b + \max\{T_c + T_f, T_d + T_e\} + (r - 1) * (T_a + T_b + T_e + T_f) \quad (19)$$

C. Hybrid Master - Worker Implementation

Here, we develop a hybrid master-worker implementation that combines the advantages of the three previous parallel implementations in order to reduce the load imbalance and communication overhead. This implementation is based on the optimal distribution strategy of the text collection which is performed statically. To avoid the slowest workstations to determine the parallel string matching time, the load should be distributed proportional to the capacity of each workstation. The goal is to assign the same amount of time, which may not correspond to the same amount of text collection.

To achieve a good balanced distribution among heterogeneous workstations, the amount of text distributed to each workstation should be proportional to its processing capacity as compared to the entire network:

$$l_i = \frac{S_i}{\sum_{j=1}^p S_j} \quad (20)$$

Therefore, the amount of the text collection that is distributed to each workstation M_i , ($1 \leq i \leq p$) is $l_i \times (n + m - 1)$ successive characters. There is an overlap of $m - 1$ pattern characters between successive subtexts.

The hybrid implementation that is called P4 is same as the P1 implementation but we use the optimal distribution method instead of the uniform distribution one.

1) *Performance Analysis for Hybrid Master-Worker Implementation*: According to the previous implementation, the execution time of the hybrid master-worker implementation P4 can be broken up into five terms:

- T_a : It is the I/O time each worker to read its subtext collection with size $l_i * (n + m - 1)$ ($1 \leq i \leq p$) successive characters from the local disk in the main memory. Then, the time T_a is given by:

$$T_a = \max_{j=1}^p \left\{ \frac{l_i * (n + m - 1)}{(S_{i/o})_j} \right\} \quad (21)$$

where $(S_{i/o})_j$ is the I/O capacity of the heterogeneous network when j workstation is used.

- T_b : It includes the communication time for broadcasting of the pattern string to all workers involved. The amount of this time is similar to the T_b of the P1 implementation.
- T_c : It includes the preprocessing time for processing of the pattern from each worker. The amount of this time is similar to the T_c of the P1 implementation.
- T_d : It is the string matching time across the heterogeneous cluster. Each worker performs string matching of an m pattern string in a subtext collection with size $l_i * (n + m - 1)$ characters that requires $(l_i * (n + m - 1))m$ steps using the SO string matching algorithm [2]. Then, the time T_d is given by:

$$T_d = \max_{j=1}^p \left\{ \frac{(l_i * (n + m - 1))m}{(S_{search})_j} \right\} \quad (22)$$

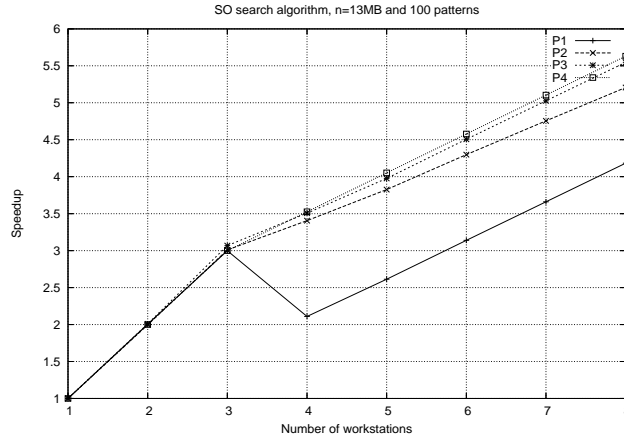


Fig. 1. Speedup of parallel multipattern matching with respect to the number of workstations for text size of 13MB and a set of 100 patterns on a heterogeneous cluster

where $(S_{search})_j$ is the text searching capacity of the heterogeneous network when j workstation is used.

- T_e : It includes the communication time to gather the results of the string matching by p workers concurrently. The amount of this time is same with the time T_e of the P1 implementation.

The total execution time of our distributed string matching implementation, T_p , using p workstations, for r patterns is given by:

$$T_p = T_a + r * (T_b + T_c + T_d + T_e) \quad (23)$$

IV. EXPERIMENTAL AND ANALYTICAL RESULTS

In this section, we discuss the experimental and theoretical results of the proposed parallel methods. These methods are implemented in C programming language using the MPI library [10].

A. Experimental Results

The target platform for our experimental study is a cluster of heterogeneous workstations connected with 100 Mb/s Fast Ethernet network. The heterogeneous cluster consists of 4 Pentium MMX 166 MHz with 32 MB RAM and 5 Pentium 100 MHz with 64 MB RAM. A Pentium MMX is used as master workstation. The middleware of the clusters is ROCKS of NPACI [11] with RedHat 7.1. The MPI implementation used on the network is MPICH version 1.2. During all experiments, the cluster of workstations was dedicated. Finally, to get reliable performance results 5 executions occurred for each experiment and the reported values are the average ones. The text collection we used was composed of documents, which were portion of the various web pages. We also selected the simple and extended patterns randomly from the same text collection.

Figure 1 presents the speedup curves of the parallel implementation with respect to the number of heterogeneous workstations. We note that all experiments for two dynamic schemes such as P2 and P3, are performed using a block size of nearly 100,000 characters because this block size is found to be optimal according to extensive study [7]. The experimental results show that the P1 and P2 parallel methods produce low performance whereas the P3 and P4 methods seem to have the best performance compared to the others ones. The P3 and P4 methods give smaller execution times and higher speedups than in case of using the P1 and P2 ones when we add workstation in the cluster heterogeneous workstations, i.e. after the third workstation.

Further, in our previous paper [7] for performance evaluation of parallel single pattern matching implementations we have observed that the performance of a single pattern can be affected by many unexpected affects, such as system load, cache/memory access, query property, etc. Evaluating parallel multiple patterns can significantly decrease these unexpected affects.

B. Theoretical Results

In this subsection, we validate four proposed performance models presented in the previous section with results obtained by experiments. In order to get these estimated results, we must to determine the values of the power weights and the values of the speeds $S_{I/O}$, S_{prep} , S_{search} and S_{comm} of the fastest workstation. The average computing power weights of the two types of workstations, Pentium MMX and Pentium, are 1 and 0.567 respectively. These weights, based on formula 1, were measured when the text size does not exceed the memory bound of any machine in the system in order to keep the power weights constant. The average speeds, $S_{I/O}$, S_{prep} and S_{search} of the fastest workstation for five pattern lengths ($m = 5, 10, 20, 30$ and 60), executing the flexible string searching application [2] (i.e., the SO string matching algorithm) are critical parameters for predicting the CPU demand times of computational segments on other workstations. The speeds were measured for different text sizes and averaged by formula 2 as follows, $S_{I/O} = 31209301,03$ chars/sec, $S_{prep} = 5149207,589$ chars/sec and $S_{search} = 2248344,298$ chars/sec. Finally, the communication speed was measured for different text sizes and block sizes as follows, $S_{comm} = 9378629,434$ chars/sec.

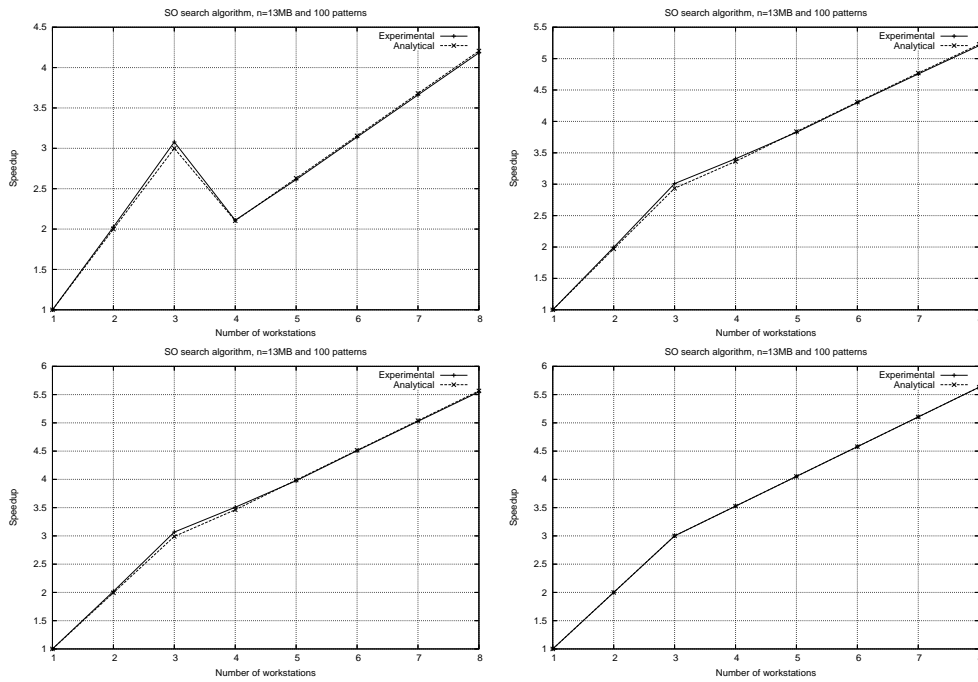


Fig. 2. Experimental and theoretical speedups for four methods P1, P2 (top), P3 and P4 (bottom) on a heterogeneous cluster

Figure 2 presents the speedups obtained by the experiments and those by the equations 11, 16, 19 and 23 for the P1, P2, P3 and P4 parallel methods respectively. We can see that the estimated results of the parallel implementations confirm well the computational behavior of the experimental results.

V. CONCLUSIONS

We have presented four parallel methods for multipattern matching problem on a cluster of heterogeneous workstations. The methods are based on static and dynamic master - worker paradigm. Moreover, we have presented the performance models for these parallel methods.

We have experimentally shown that our parallel methods perform well in handling from one to a very large number of patterns. Further, the P3 and P4 methods achieve better performance results in terms of execution times and speedups than the others. Finally, we observed that the proposed performance models of four parallel methods confirm well the computational behaviour of the experimental measurements.

REFERENCES

- [1] A. Aho and M. Corasick, "Efficient string matching: an aid to bibliographic search", *Communications of the ACM*, vol. 18, no. 6, pp. 333-340, 1975.
- [2] R. Baeza-Yates and G. Gonnet, "A new approach to text searching", *Communications of the ACM*, vol. 35, no. 10, pp. 7482, 1992.
- [3] O. Beaumont, A. Legrand and Y. Robert, The master-slave paradigm with heterogeneous processors, Report LIP RR-2001-13, 2001.
- [4] A. Boukerche, A. C. M. A. de Melo, M. Ayala-Rincon, T. M. Santana, "Parallel strategies for local biological sequence alignment in a cluster of workstations", in *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS05)*, 2005.
- [5] P.D. Michailidis and K.G. Margaritis, "String matching problem on a cluster of personal computers: Experimental results", in *Proc. of the 15th International Conference Systems for Automation of Engineering and Research*, pp. 71-75, 2001.
- [6] P.D. Michailidis and K.G. Margaritis, "String matching problem on a cluster of personal computers: Performance modeling", in *Proc. of the 15th International Conference Systems for Automation of Engineering and Research*, pp. 76-81, 2001.
- [7] P.D. Michailidis and K.G. Margaritis, "Performance Evaluation of Load Balancing Strategies for Approximate String Matching Application on an MPI Cluster of Heterogeneous Workstations", *Future Generation Computer Systems*, vol. 19, no. 7, pp. 1075-1104, Elsevier-Science, 2003.
- [8] G. Navarro and M. Raffinot, "Fast and flexible string matching by combining bitparallelism and suffix automata", *ACM Journal of Experimental Algorithmics (JEA)*, vol. 5, no. 4, 2000.
- [9] G. Navarro and M. Raffinot, *Flexible Pattern Matching in Strings Practical on-line search algorithms for texts and biological sequences*, Cambridge University Press, Cambridge, 2002.
- [10] P. Pacheco, *Parallel Programming with MPI*, San Francisco, Morgan Kaufmann, CA, 1997.
- [11] P.M. Papadopoulos, M.J. Katz, G. Bruno, NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters, San Diego Supercomputer Center, University of California San Diego, 2001.
- [12] T. K. Yap, O. Frieder and R. L. Martino, "Parallel computation in biological sequence analysis", *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 3, pp. 283-293, 1998.
- [13] Y. Yan, X. Zhang and Y. Song, "An effective and practical performance prediction model for parallel computing on non-dedicated heterogeneous NOW", *Journal of Parallel and Distributed Computing*, vol. 38, no. 1, pp. 63-80, 1996.