Generalized Performance Model for Flexible Approximate String Matching on a Distributed System

Panagiotis D. Michailidis and Konstantinos G. Margaritis

Department of Applied Informatics, University of Macedonia
{panosm, kmarg}@uom.gr

Abstract

This paper proposes a generalized and practical parallel algorithm for flexible approximate string matching which is executed for several kinds of clusters such as homogeneous cluster and heterogeneous cluster. This parallel algorithm is based on the master - worker paradigm and it implements different partitioning schemes such as static and dynamic load balancing cooperating with different data allocation techniques such as the allocation of texts and allocation of text pointers. Furthermore, the parallel algorithm is analyzed experimentally using the Message Passing Interface (MPI) library for different strategies of load balancing and data allocation onto two kinds of clusters. Further, we propose a general performance model that can be used to predict the performance of the parallel flexible approximate string matching algorithm for both types of clusters. The theoretical performance model has been validated against experimental results and it is shown that the model is able to predict the parallel performance accurately.

Keywords: approximate string matching, performance model, cluster of workstations, MPI

1. Introduction

Approximate string matching problem received much attention over the years due to its importance in various applications such as information retrieval, computational biology and intrusion detection. All those applications require highly efficient algorithm to find all the occurrences of a given pattern in the text. It is defined as follows: given a large text collection $t = t_1t_2...t_n$ of length n, a short pattern $p = p_1p_2...p_m$ of length m and a maximal number of errors allowed k, we want to find all text positions where the pattern matches the text up to k errors. Errors can be substituting, deleting, or inserting a character. Recent surveys and experimental results of well known sequential algorithms for simple approximate string matching can be found in [Navarro (2001)].

The basic problem can be extended to include more complicated patterns, including patterns with a “don’t care” symbol, patterns with a complement symbol and patterns

with a class symbol. Some recent publications, which also provide surveys of previous work, are [Navarro et al. (2000)]. Therein sequential algorithms are proposed for the solution of several aspects of the flexible approximate string matching problem.

A few attempts for implementing approximate string matching and other similar problems have been made on a cluster of workstations. In [Michailidis et al. (2001)] an exact string matching algorithm was parallelized and modeled on a homogeneous platform giving positive experimental results. Further, in [Lavevier et al. (1997), Yap et al. (1998)] presented parallelizations of a biological sequence analysis algorithm on a homogenous cluster of workstations and on an Intel iPSC/860 parallel computer, respectively. Further, we have been proposed four parallel implementations for simple approximate string matching on a cluster of heterogeneous workstations [Michailidis et al. (2003)]. These implementations are based on static and dynamic master - worker paradigm. In the same paper we have been proposed a performance prediction model of four implementations. Recently, in [Boukerche et al. (2005)] three parallel strategies were developed to run a biological sequence alignment algorithm such as the Smith-Waterman algorithm on a cluster of homogeneous workstations. Previous research is based on parallelization of a simple dynamic programming algorithm on a cluster of workstations.

This paper makes the following research contributions:

- A generalized parallel flexible approximate string matching algorithm to execute different load balancing strategies which cooperate with other data allocation policies on different kinds of high performance clusters (such as homogeneous and heterogeneous) in an unified way.
- A practical performance prediction model for evaluation of the general parallel algorithm.
- The proposed general parallel algorithm implements not only simple approximate string matching but also it executes searching for complex patterns. Most text searching applications require these types of patterns, therefore algorithms that cannot support them have limited applicability. Further, this parallel algorithm can be implementing any sequential flexible approximate string matching algorithms quite easily [Navarro et al. (2000), Myers (1999)].

2. A General Parallel Implementation and Performance Analysis

2.1 General parallel implementation

The general parallel implementation is based on a general master - worker programming paradigm and it takes into account two criteria: the load balancing strategy and the data allocation strategy. The load balancing strategy is divided into

two categories: static and dynamic. In static load balancing strategy, the entire text collection is partitioned into a number of the subtext collections according to the number of workstations allocated. The amount of the subtext collection depends on the type of cluster computing environment. In dynamic load balancing strategy, the text collection is partitioned into small chunks of text and these chunks are assigned dynamically to idle workstations in order to keep all the workstations busy. The size of each chunk is *a* successive characters. This 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. This block size can be applied to both types of clusters.

On the other hand, the data allocation strategy also is divided into two categories: allocation of texts and allocation of text pointers. In allocation of texts, the subtexts that are obtained by load balancing strategy are distributed to corresponding workstations. In this case the entire text collection is required to be stored on the local disk of the master workstation. In allocation of text pointer, some master workstation of the cluster has a text pointer that shows the current position in the text collection and the master distributes the text pointers instead of the subtexts to corresponding workers in order to reduce the communication overhead. In this case the entire text collection is required to be stored on the local disks of all workstations.

Our distributed approximate string matching algorithm is based on the following assumptions: First, the number of workstations in the cluster is denoted by *p* and we assume that *p* is power of 2. Further, the workstations have an identifier *myid* and are numbered from 1 to *p*. Second, the length of text *n* is much longer than the length of the pattern string *m*. Finally, the complete text collection is stored on the local disks of all workstations irrelative of the data allocation policy is used. The proposed general parallel implementation is shown below.

```
main()
{
        1. Initialize message passing routines;
        2. for(i = 1; i <= p; i++) bs[i] = (l[i] * n) * A + a * A';
        3. if (process == master) then call master() else call worker();
        4. Exit message operations;
}

master()
{
        1. bcast(pattern);
        2. bcast(k);
        3. count = active = offset = 0;
        4. for(i = 1; i <= p; i++) {
        5.          if (R == 1)
        6.                  send(&offset,Pi,WORKTAG);
        7.          else {
        8.                  read a text chunk of size bs[i] starting from the position offset of file;
        9.                  send(text,Pi,WORKTAG);
        10.          }
```

```
            11.         active++;
            12.         offset += bs[i] - m + 1;
            13. }
            14. do {
            15.         recv(&count);
            16.         active--;
            17.         sender = Pany;
            18.         if (offset < (n - (m + 1))) {
            19.                 if (R == 1)
            20.                     send(&offset,Psender,WORKTAG);
            21.                 else {
            22.                     read a text chunk of size bs[sender] starting from the pos. offset of file;
            23.                     send(text,Psender,WORKTAG);
            24.                 }
            25.                 active++;
            26.                 offset += bs[sender] - m + 1;
            27.         } else
            28.                 send(0,Psender,DIETAG);
            29. } while (active>0);
}

worker()
{
            1. bcast(pattern);
            2. bcast(k);
            3. while(TRUE) {
            4.         if (R == 1) recv(&offset,Pmaster,sourcetag);
            5.         else recv(text,Pmaster,sourcetag);
            6.         if (sourcetag == DIETAG) break;
            7.         if (R == 1) read a text chunk of size bs[myid] starting from the pos. offset of file;
            8.         count = string_search(pattern,text,m,n,k);
            9.         send(&count,Pmaster);
               }
}
```

The parallel implementation consists of two stages: the preprocessing and the processing stage. The preprocessing stage of the parallel algorithm corresponds to the line 2 of the function *main()*, whereas the processing stage corresponds to the master and worker procedures. The preprocessing stage consists of computing the amount of the text collection that is assigned to each workstation. The size of the text collection that is distributed to each workstation depends on the load balancing policy and the type of cluster computing environment. Therefore, if we want to run the static load balancing policy on heterogeneous cluster then the amount of text that is distributed to each workstation is proportional to its processing capacity compared to the entire network i.e. is equal to $l_i * n$ characters where l_i is equal to $S_i / Sum_j^{p-1}{}_{=0} S_j$ and S_j is the speed of the workstation j. If we substituting $S_j = 1$ in above equation l_i for homogeneous cluster then the amount of text that is assigned to each workstation is equal to the size of the text collection divided by the number of allocated workstations. Finally, if we want to run the dynamic load balancing policies on both types of clusters then the size of text that is assigned dynamically to workers will be an optimal block size a which minimizes the communication overhead.

The processing stage consists of five phases. In first phase, the master broadcasts the pattern string and the number of errors *k* to all workers which corresponds to the lines 1-2 of the master procedure. In second phase, the master sends the first text pointers to corresponding workers or the master reads from the local disk the several chunks of the text collection and sends these chunks to corresponding workers. The second phase corresponds to the lines 4-13 of the master procedure. In third phase, the master receives the number of occurrences from each worker and if there are still any chunks of the text collection left, the master reads and distributes next chunks (or pointers) of the text collection to workers. The master sends a terminator message when all the chunks of text or pointers have been taken. This phase corresponds to the lines 14-29 of the master procedure. In fourth phase, when each worker receives a pointer then reads from the local disk the *bs[j]* characters of text starting from the pointer that receives and performs a sequential flexible approximate string matching procedure between the corresponding chunk of text and the pattern. However, when each worker receives a chunk of text then performs a sequential flexible approximate string matching algorithm between the corresponding chunk of text and the pattern. This fourth phase corresponds to the lines 4-8 of the worker procedure. In fifth phase, each worker sends the result i.e. the number of occurrences back to master. This phase corresponds to the line 9 of the worker procedure.

We introduced two Boolean parameters A and R in the proposed parallel implementation which control different policies of load balancing and data allocation for both types of clusters. More specifically, the parameter A controls the load balancing strategy whereas the parameter R controls the data allocation policy. In Table 1 we show the values of the parameters A and R which corresponds to different parallel schemes and variations.

Table 1. *Protocol of the parallel implementation*

Parallel Schemes	A	R
Static allocation of texts (in short, P1)	1	0
Dynamic allocation of texts (in short, P2)	0	0
Dynamic allocation of pointers (in short, P3)	0	1
Static allocation of pointers (in short, P4)	1	1

2.2 General performance model

We give the execution time for each phase of the general parallel implementation. Therefore, the execution time can be broken up into four terms:

- T_a: It is the execution time of the first phase and it is given by:

$$T_a = \frac{m+1}{S_{comm}} \quad \textbf{(1)}$$

where S_{comm} is communication speed.

- T_b: It is the execution time of the second and third phases. This time can be broken up into three sub-terms T_{b1}, T_{b2} and T_{b3} and they are given by:

$$T_{b1} = \frac{\sum_{j=1}^{p}(bs[j] * R' + 1 * R)}{S_{comm}} \quad (2)$$

$$T_{b2} = \frac{n * R'}{(S_{I/O})_{master}} \quad (3)$$

$$T_{b3} = \frac{(\frac{n}{bs[j]+m-1} - p) * ((bs[j] + m - 1) * R' + 1 * R) * A'}{S_{comm}} \quad (4)$$

where S_{comm} is communication speed and $(S_{I/O})_{master}$ is the I/O capacity of the master workstation. We note that the term T_{b1} corresponds to the lines 4 - 13 of the master procedure and it is include the communication time to send the first p pointers or chunks of text to workers. The term T_{b2} corresponds to the lines 8 and 22 of the master procedure and it is include the I/O time to read the entire text collection. Finally, the term T_{b3} corresponds to the lines 14-29 of the master procedure and it is include the other communication time to send next chunks (or pointers) of text collection to workers.

- T_c: It is the execution time of the fourth phase. This time can be broken up into two sub-terms T_{c1} and T_{c2} and they are given by:

$$T_{c1} = (\frac{(\frac{n}{bs[j]+m-1} - p)(bs[j] + m - 1)}{\sum_{j=1}^{p}(S_{i/o})_j} + max_{j=1}^{p}\{\frac{bs[j] + m - 1}{(S_{i/o})_j}\}) * A' * R +$$

$$+max_{j=1}^{p}\{\frac{bs[j] + m - 1}{(S_{i/o})_j}\} * A * R \quad (5)$$

$$T_{c2} = (\frac{(\frac{n}{bs[j]+m-1} - p)[\Theta(bs[j] + m - 1, m, k)]}{\sum_{j=1}^{p}(S_{search})_j} + max_{j=1}^{p}\{\frac{\Theta(bs[j] + m - 1, m, k)}{(S_{search})_j}\}) * A' +$$

$$+max_{j=1}^{p}\{\frac{\Theta(bs[j] + m - 1, m, k)}{(S_{search})_j}\} * A \quad (6)$$

where $\sum_{j=1}^{p}(S_{i/o})_j$ and $\sum_{j=1}^{p}(S_{search})_j$ is the I/O and searching capacity of the heterogeneous network when p workstations are used, respectively. The term T_{c1} corresponds to the line 7 of the worker procedure and it is average I/O time to read each worker from local disk the *bs[j]* characters of the text in the case of allocation of text pointers. The term T_{c2} corresponds to the line 8 of the worker procedure and it is average string searching time. We include the first max term in the equations 5 and 6 which define 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 is used in the case of the dynamic load balancing policies. Also, there is a second max term in equations 5 and 6 which define the maximum time by all workstations of the heterogeneous cluster when static load balancing policies are used. Finally, *Θ(bs + m - 1, m, k)* is complexity of any approximate string searching algorithm between a chunk of text and a pattern.

- T_d: It is the execution time of the fifth phase and it is given by:

$$T_d = \frac{\frac{n}{bs[j]+m-1} * A' + p * A}{S_{comm}} \quad \textbf{(7)}$$

where S_{comm} is the communication speed.

We must note that the values of the parameters A and R are given in Table 1 and the parameters A´ and R´ are defined as the complement of A and R. Finally, the total execution time of the general parallel implementation, T_p, using p workstations, is given by:

$$T_p = T_a + T_{b1} * A + T_{b2} * R' + T_{c1} * A * R + (T_{c2} + T_d) * A + (max\{T_{b1} + T_{b3} + T_d, T_{c1} * A' * R + T_{c2}\}) * A' \quad \textbf{(8)}$$

In the general equation 8 we consider the maximum value between the communication time and the computation time since in dynamic load balancing policies, there is parallel communication and computation.

3. Experimental and Theoretical Results

The proposed parallel algorithm is implemented in C programming language using the MPI library. In order to show the flexibility and the performance of the parallel algorithm we used the values of Table 1 for implementing different parallel schemes. Further, we incorporated the MYE flexible approximate string matching algorithm [Myers (1999)] in each worker of the proposed parallel implementation for executing string matching operation.

3.1 Experimental Results

The target platforms for our experimental study are two kinds clusters of workstations connected with 100 Mb/s Fast Ethernet network. The homogeneous cluster consists of 9 Pentium workstations based on 100 MHz with 64 MB RAM. 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 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 10 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 workstations for different parallel schemes using the MYE algorithm on both types of clusters. 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 [Michailidis et al. (2003)]. We observe from the experiments that all parallel schemes produce similar speedups. However, the schemes with allocation of texts like P1 and P2 present a communication overhead slightly compared to the schemes with allocation of pointers such as P3 and P4.

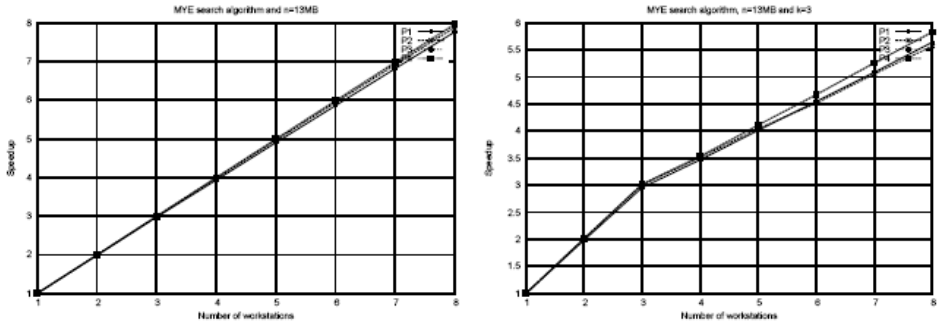

Figure 1. *Speedup of parallel flexible approximate string matching with respect to the number of workstations for text size of 13MB and k = 3 using several pattern lengths on a homogeneous cluster (left) and on a heterogeneous cluster (right)*

3.2 Theoretical Results

In this subsection, we validate our proposed general performance model presented in the previous section with results obtained by experiments. The performance estimated results for different schemes of parallel implementation was obtained by the equation 8. In order to get these estimated results, we must to be 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 [Yan et al. (1996)] 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} were measured for different text sizes as follows, $S_{I/O}$ = 31021855,26 chars/sec, S_{prep} = 4407511,178 chars/sec and S_{search} = 892402,9763

chars/sec. Finally, the communication speed was measured for different text sizes and block sizes as follows, S_{comm} = 9378629,434 chars/sec.

Figure 2 presents the speedups obtained by the experiments and those by the equation 8 for different parallel schemes on both types of clusters. We can see that the estimated results of the parallel implementation for different schemes confirm well the computational behavior of the experimental results.

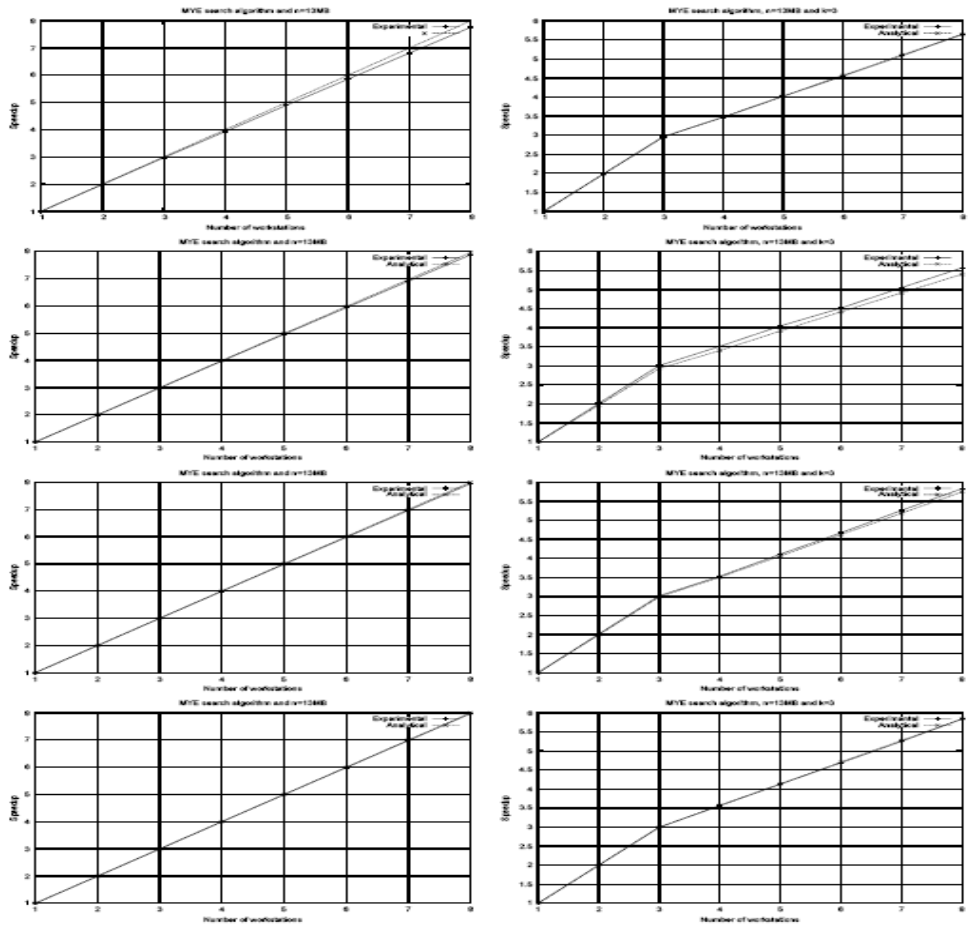

Figure 2. *Experimental and theoretical speedup for four schemes P1, P2, P3 and P4 on a homogeneous cluster (left) and on a heterogeneous cluster (right)*

4. Conclusions

The advantage of the proposed parallel implementation is that there is a single algorithm to execute different load balancing and data allocation choices for both

types of cluster in a unified and efficient way. Further, the proposed performance model is general and it can be incorporated in a kernel of a application. This model in a kernel can be predict the performance for some workstations of the cluster a priori and it will run the suitable load balancing and data allocation strategy. Therefore, as a general conclusion we can say that this parallel implementation and performance model provide a flexible environment to execute the string matching in a huge text base using different policies of load balancing and data allocation. When user submits a request with the values of the parameters A and R, the cluster platform is adapted easily in order to execute the string matching in a large text collection. Further, the parallel implementation has adaptability since it is suitable for parallelizing other sequential flexible approximate string matching algorithms and multipattern string matching algorithms. Finally, the performance model can be adapted to other important applications such as bioinformatics and scientific computing.

References

Boukerche, A., Melo, A. C. M. A., Ayala-Rincon, M., Santana, T.M. (2005), *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).

Lavenier, D., Pacherie, J.L. (1997), *Parallel processing for scanning genomic databases*, in Proc. PARCO'97, pp. 81-88.

Michailidis, P.D., Margaritis, K.G. (2003), *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.

Michailidis, P.D., Margaritis, K.G. (2001), *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.

Myers, G. (1999), *A fast bit-vector algorithm for approximate string matching based on dynamic programming*, Journal of the ACM, vol. 46, no. 3, pp. 395-415.

Navarro, G. (2001), *A guided tour to approximate string matching*, ACM Computer Surveys, vol. 33, no. 1, pp. 31-88.

Navarro, G., Raffinot, R. (2000), *Fast and flexible string matching by combining bit-parallelism and suffix automata*, ACM Journal of Experimental Algorithmics, vol. 5, no. 4.

Yap, T.K., Frieder, O., Martino, R.L. (1998), *Parallel computation in biological sequence analysis*, IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 3, pp. 283-293.

Yan, Y., Zhang, X., Song, Y. (1996), *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.