Parallel Processing Approach for Pattern Matching using MPI

Rashmi C.
High Performance Computing Project
Dept. of Studies in CS
University of Mysore, Mysuru, India

Hemantha Kumar G.
High performance Computing Project
Dept. of Studies in CS
University of Mysore, Mysuru, India

ABSTRACT
Bioinformatics is one of the field where high performance computation widely used. Pattern matching is essential task in Bio-informatics. A powerful technique for searching sequence patterns in the biological sequence databases is the pattern recognition. Significant increase in the number of protein sequences and DNA expanded the need for the enhancement of performance of pattern matching. Hence fast and high performance algorithms are highly demanded in many applications of computational molecular biology and bio-informatics. In this paper we present a parallel processing approach for pattern matching algorithm using distributed parallel programming paradigm Message Passing Interface (MPI). The focus of the research is the implementation of basic algorithm naïve for pattern matching by utilizing compute nodes of high performance computing server optimally. The parallel algorithm finds correct matches and experimental results show very high performance gain over sequential approach.

Keywords
Pattern recognition, DNA, Parallel Processing, MPI

1. INTRODUCTION
Biological information is increasing from past few years. Larger computations are required for the complex interactions for determining the biological processes [1]. Different processing elements with different characteristics on same machine are becoming main stream in high performance computing platforms and seem able to cope with these requirements. As the growth rate of biological sequence databases increased, the demand for advanced and high performance computational method for comparing and searching biological sequences have also increased. In DNA sequence alignment [2], the performance of comparison and alignment affect a lot of application processes such as vaccines design, drugs, disease detection and curing method. Hence with the high performance and high sensitivity DNA sequences alignment or comparison the vaccines, drugs, disease detection and disease curing method can be designed and defined in a faster way. To satisfy this need, high performance and sensitive DNA sequence matching algorithms are very important for research and application of molecular biology today. Biological sequence alignment is a computationally expensive application in the field of bioinformatics and computational biology as its computing and memory requirements grow quadratic ally with the size of the datasets. It aims to find out whether two or more biological sequences are related or not. Pattern matching focuses on finding the occurrences of a particular pattern in a text file. The problem in pattern discovery is to determine how often a candidate pattern occurs, as well as possibly some information on its frequency distribution across the sequence/text. In general, a pattern will be a description of a set of strings, each string being a sequence of symbols. Hence, given a pattern, it is usual to ask for its frequency, as well as to examine its occurrences in a given sequence/text. The main objective of this paper is to discuss and present about the parallel algorithm for the pattern matching which were implemented to achieve the improvements in the reduction of execution time in bio-informatics. Parallel computation serve as a guideline for other projects in bioinformatics for data analysis and computer science. Multicore clusters are the most popular option for the deployment of High Performance Computing (HPC) infrastructures, due to their scalability and performance/cost ratio. Message-passing interface (MPI) [3] is the distributed memory programming and shared memory programming model Open specification for multi-processing (OpenMP) are the two models for parallel programming multi-core architectures. The most commonly used paradigm that can be employed not only within a single processing mode but also across several connected ones is the Message Passing Interface (MPI). To bridge the gap between the performances offered by a parallel distributed architecture and also to enhance portability in parallel applications MPI standard has been designed. The standard defines semantics and syntax for writing portable message passing programs in Fortran, C and C++. A clearly defined base set of routine can be implemented efficiently by parallel hardware provided by MPI. A networks of workstations, shared memory multiprocessors, distributed memory and a combination of these elements can be used by MPI. This distributed memory programming paradigm can be applied in multiple settings and are independent of network speed or of memory architecture.

Memory is used for programming models. An application runs as a collection of autonomous processes each with its local memory and processes will communicate by sending and receiving the messages in message passing model but shared access space is accessed by each processes in shared memory model. Universality, Simplicity, Performance, ease of debugging and expressivity are the advantages of message passing model. Source-code portability of message-passing programs written in Fortran or C across a variety of architectures are provided by MPI. MPI allows the development of the code on one architecture before running it on the target machine and protecting investment in a program. Since Brute force pattern matching is widely used for matching the patterns, hence it is considered for parallelization using a distributed memory programming paradigm in Bio-informatics for DNA sequence.

The rest of the paper is organized as follows Section II is Literature survey, Section III is Parallel Approach for Pattern Matching, Section IV is Experimental System Requirements and we make some concluding remarks in Section V.

2. LITERATURE SURVEY

Study of pattern matching algorithms the literature describes various traditional pattern matching methodologies like Naive Brute force, Boyer Moore, Knuth Morris Pratt and Dynamic algorithms along with their performance issues when applied for sequence analysis. Pattern matching is used in various processes like DNA sequencing, Intrusion Detection System.

1) Naive Brute force

It is one of the simplest algorithms having complexity O(MN). In this, first character of pattern P (with length m) is aligned with first character of text T (with length n).Then scanning is done from left to right. As shifting is done at each step it gives less efficiency [4].

2) Boyer-Moore Algorithm

It performs larger shift-increment whenever mismatch is detected. It differs from Naïve in the way of scanning. It scans the string from right to left; unlike Naive i.e. P is aligned with T such that last character of P will be matched to first character of T. If character is matched then pointer is shifted to left to very rest of the characters of the pattern.

If a mismatch is detected at say character c, in T which is not in P, then P is shifted right to m positions and P is aligned to the next character after c. If c is part of P, then P is shifted right so that c is aligned with the right most occurrence of c in P. The worst complexity is still O (m+n) [5].

3) Knuth-Morris-Pratt

This algorithm is based on automaton theory. Firstly a finite state automata model M is being created for the given pattern P. The input string T with Σ= {A, C, T, G} is processed through the model. If pattern is present in text, the text is accepted otherwise rejected. But the only disadvantage of the KMP algorithm [6] is that it doesn't tell the number of occurrences of the pattern [7].

4) Dynamic programming Algorithms

Dynamic programming is the oldest and mostly used algorithm. Basically Needleman Wunsch and Smith waterman algorithm [8] come under this approach. These are much more complex than the exact pattern matching. It involved solving successive recurrence relations recursively i.e. smaller problems are solved in succession to solve the main problem. A)Smith-Waterman (local alignment)[8]

• Accuracy: good with gapped pairs

• Processing: Computationally expensive O (N2) and with trace-back a lot of memory is required; this is slow

• Limitations: indexing to find targets is required.

B) Needleman-Wunsch (global alignment)[8]

• Good for small genomes and long matching alignments

• Processing: Computationally expensive O (N2) Talk today showed novel pruning technique for in large matches.

• Limitations: requires hard left hand bound known query and target size.

3. DISTRIBUTED MEMORY PROGRAMMING

MPI is a standardized distributed memory programming, specification for clusters, Parallel computers and heterogeneous networks as depicted in figure 1. It primarily addresses the message passing parallel programming model. Data is moved from the address space of one process to that of another process through co-operative operations on each process. Providing a widely used standard for writing message passing programs including interface attempts such as practicability, portability, efficient, flexibility and ability to run transparently on heterogeneous systems, a collection of processors with distinct architectures. It implements asynchronous, global and local. MPI has two modes of communication collective and point to point communication. Collective communication allows large number of processes to communicate, they are of two kinds data movement operations and collective computation operations. A value is computed from data located in different processes for example sum, maximum, logical OR and so forth in collective communications. Data are rearranged in among the processes in data movement operations. Point to Point is the simplest form of message passing. Only two processes are involved in communications for sending and receiving the messages but have a some different versions which represent different semantics in the communication. To program the parallel computers, MPI is the language independent communication protocol. The solution most widely used for shared memory programming is openMP since an easy parallel application development is achieved through compiler directives. As this model is limited to shares memory architectures, computational power of a single system is bounded by the performance. Hybrid system with both shared and distributed memory is used in order to avoid the limitation, both OpenMP combined with MPI can be programmed for multi-core clusters. However this hybrid model can make the parallelization.

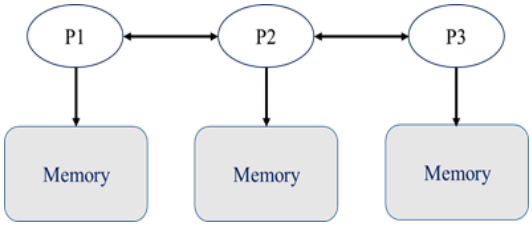

Fig 1: Distributed Memory Programming Paradigm

MPI programming model exhibits SPMD (Single Program Multiple Data) to distinguish it from MPMD(Multiple Program and Multiple Data) in which same program is executed by every or different processor. Many versions of high performance open source MPI library for IOGig/iWARP, RoCE (RDMA over converged enhanced Ethernet such as MAVAPICH(MPI-1), MAVAPICH2 (MPI-2.2 and 3.0) that support for GPGPUs and MIC delivering scalability and best performance to MPI applications. MVAPICH with MPI-1 semantics and MVAPICH2 with MPI-2 semantics are the two current versions of MPI Library supporting different computation with communication platforms. C/C++ and Fortran programming languages are supported by these versions. Built network topology support makes an efficient use of MPICH2 on LINUX and UNIX platforms in contrast to MPICH2 on windows.

MPI has become a de facto standard for communication among process that model a parallel program running on a

distributed memory system, it's not sanctioned by any major standards body. Computer clusters are the distributed memory supercomputers often run such programs. The model MPI-1 model has no shared memory concept and MPI-2 has only limited distributed shared memory concept. MPI Interface provides communication functionality, synchronization and virtual topology across a set of processes [10] with language specific syntax and features in a language independent way. Programmers commonly refer to the processes as processors, MPI programs always work with processes. Single process will be assigned to each CPU(or core in a multicore machine) at runtime through the agent that starts the MPI program (MPI daemon) called mpiexec or mpirun. Computer machine that initiates MPI ring daemon will have process manager in its core CPU. Process manager identified with ID 0 and all of his worker have ID greater than 0.

4. PATTERN MATCHING

A basic algorithm for pattern matching is naïve brute force string matching that takes a string S, of size m and pattern P, of size n and scans the first mn elements of the string from left to right with pattern, looking for matches. Basically algorithm considers possible starting positions of the pattern(P) for i=0 to mn. Then for every starting position(i) the pattern (P) must exactly match S for next consecutive n-1 positions. The result of the algorithm is the set I containing all of the starting positions in S where P exactly matches the string S (using indices starting at 1) [9].The sequential form of algorithm consists of function, where it attempts to match pattern of text by scanning text from left to right. In sequential code, a single process is conducting the search and when it finds a match the algorithm will output to console the position it was found.

4.1 Sequential Algorithm

The naïve algorithm finds all valid shifts using a loop that checks the condition P[1..m]=T[s+1…. S+m] for each of the n-m+1 possible values of s.

Naïve String Matcher(T,P)

1. m=T.length

2. n=P.length

3. for s=0 to m-n

4. if P[1..m]==T[s+1..s+n]

5. print "Pattern occurs with shift"

4.2 Parallel Algorithm

Main procedure

main ()

{

1. Initialize MPI and OpenMP routines;

2. If (process==master) then call master(); else call worker();

3. Exit message passing operations;

}

Master sub-procedure

master()

{

1. Broadcast the name of the pattern set and text to workers; (MPI_Bcast)

2. Broadcast the offset of the text, the blocksize and the number of threads to workers; (MPI_Bcast)

3. Receive the results (i.e. matches) from all workers; (MPI_Reduce)

4. Print the total results;

}

Worker sub-procedure

worker()

{

1. Receive the name of the pattern set and text; (MPI_Bcast)

2. Preprocess the pattern set;

3. Receive the offset of the text, the blocksize and the

number of threads; (MPI_Bcast)

4. Open the pattern set and text files from the local disk and store the local subtext (from text + offset to text + offset + blocksize) in memory;

5. Call the chosen pattern matching algorithm passing

a pointer to the subtext in memory;

7. Determine the number of matches from each process.

8. Send the results (i.e. matches) to master;

}.

5. RESULTS

In this paper we use 16 cores on Linux-based platform to study the effect of parallel processing performance of MPI parallel implementations for pattern matching. Cluster Hardware comprises of Two Master nodes (Wipro-Netpower Datasystem) with Intel® Xeon® CPU E5-2670 @ 2.60GHz, 4 X 900GB SAS HDD, 8 X 8GB RAM with 20 compute nodes (4 X 5 Wipro Netpower Blade chassis servers) Intel® Xeon® CPU E5-2670 @ 2.60GHz, 1 X 300GB SAS HDD, 8 X 8GB RAM.

Initially explored a solution that simply split the text up so that each process would check one portion of the text; for example, using 16 processes, each process would check 1/16th of the text. It's faster than the sequential version as depicted in table I.

Table 1. Table captions should be placed above the table

Data Set	Data Size	Sequential Time(secs)	MPI Time(secs)
A1	2MB	140.213	15.62
A2	1MB	90.84	20.61
Total		231.053	36.23

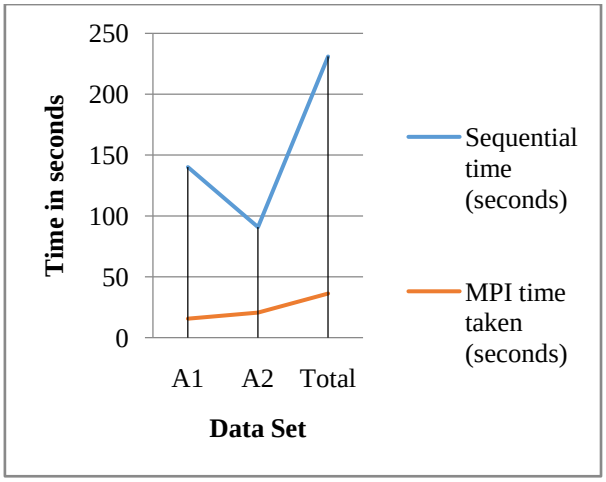

Fig 2: Graphical representation

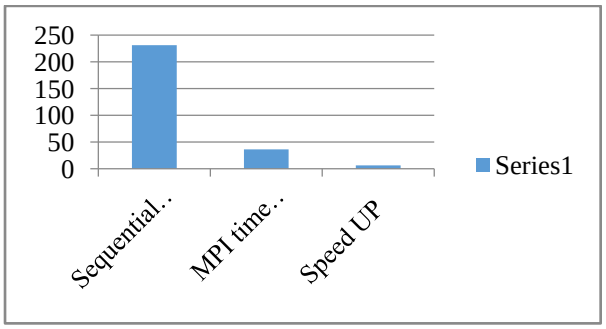

Fig 3: Performance

6. CONCLUSION

In this paper we focused on distributed programming model using MVAPICH2 as a message passing interface implementation on Linux platforms. With the performance analysis, it's evident that the new implementation seems to become more efficient with a higher number of processes. The proposed work is tested with data sets. The effect of parallel processing and also the number of cores on the performance of parallel pattern matching has been theoretically and experimentally studied. From the experimental results, it is estimated that the parallel pattern matching computation is less compared to sequential pattern matching. But MPI has some crons hidden communication takes place with collective communication and not always best to use collective communication due to synchronization.

7. ACKNOWLEDGMENTS

This work is carried by Rashmi C in High Performance Computing Laboratory, Department of Studies in Computer Science.

8. REFERENCES

[1] A. Tumeo and O. Villa. Accelerating DNA analysis applications on GPU clusters. In IEEE 8th Symposium on Application Specific Processors (SASP), pages 71–76. 2010.

[2] Mount D. Bioinformatics: Sequence and Genome Analysis, Cold Spring Harbor Laboratory(CSHL) Press,2004.

[3] M. J. Quinn, Parallel Programming in C with MPI and OpenMP, Tata McGraw Hill Publications, 2003, p. 338.

[4] http://codeapirant.worpress.com/2013/05/20/brute-force-naiveapproach-to-string-searching

[5] R.S. Boyer, J.S. Moore, "A fast string searching algorithm,"Communication of the ACM, Vol. 20, No. 10, 1977, pp.762–772.

[6] S. Rajesh , S.Prathima, Dr.L.S.S.Reddy, "Unusual Pattern Detection in DNA Database Using KMP Algorithm",2010 International Journal of Computer Applications(0975-8887) Volume 1 – No.22.

[7] KNUTH, D. E, MORRIS JR J. H , PRATT V. R,"Fast pattern matching in strings", In the procd. Of SIAM J.Comput.Vol. 6, 1, pp.323–350, 1977.

[8] Lee W-P, Stromberg MP, Ward A, Stewart C, Garrison EP, et al. (2014) "MOSAIK: A Hash-Based Algorithm for Accurate Next-Generation Sequencing Short-Read Mapping." PLoS ONE 9(3):e90581. doi:10.1371/journal.pone.0090581.

[9] Parida, Laxmi (2008) Pattern Discovery in Bioinformatics: Theory& Algorithms Boca Raton: Chapman & Hall/CRC pg. 139-182,183-212

[10] Shima Soroushnia, Masoud Daneshtalab, Tapio Pahikkala, Juhu Plosila "Parallel Implementation of Fuzzified Pattern Matching Algorithm on GPU", IEEE 23rd Euromicro International Conference on Parallel, Distributed and Network-Based processing(PDP), 2015.

[11] Kefu Xu, Wenke Cui, Yue Hu, Li Guo, "Bit-Parallel Multiple Approximate String Matching based on GPU", First International conference on Information Technology and Quantitative Management, Procedia Computer science , Vol 17,2013, pages 523-529.

[12] Daniel Luchaup, Randy Smith, Cristian Estan, Somesh Jha, "Speculative Parallel Pattern Matching" IEEE Transactions on Information Forensics and Security, Vol 6, Issue 2, 2011,pages 438-451.

[13] Spector, A. Z. 1989. Achieving application requirements. In Distributed Systems, S. Mullender

[14] Sinan Sameer Mahmood Al-Dabbagh, Nawaf Hazim Barnouti, Mustafa Abdul Sahib Naser, Zaid G. Ali, "Parallel Quick Search Algorithm for the exact String Matching problem using OpenMP", Journal of Computer and Communications, Scientific Research Publishing , Vol 4,2016, pages1-11.

[15] Gulfishan Firdose Ahmed, Nilay khare, "String Matching Algorithms using Bit Parallelism", International Journal of Advanced Engineering and Global Technology Vol-2, Issue-5, May 2014.

[16] Dale E. Parson, "Parallel reduced-instruction-set-computer architecture for real-time symbolic pattern matching", Proc. SPIE 1468, Applications of Artificial Intelligence IX, (1 March 1991); doi: 10.1117/12.45534.

[17] Saman Ashkiani, Nina Amenta, John D. Owens, "Parallel Appraoches to the String Matching Problem on the GPU", SPAA 2016.