Parallel Homologous Search with Hirschberg Algorithm: A Hybrid MPI-Pthreads Solution

NURAINI ABDUL RASHID, ROSNI ABDULLAH & ABDULLAH ZAWAWI HJ. TALIB
School of Computer Sciences,
Universiti Sains Malaysia
11800 USM Pulau Pinang
MALAYSIA

Abstract: - In this paper, we apply two different parallel programming model, the message passing model using Message Passing Interface (MPI) and the multithreaded model using Pthreads, to protein sequence homologous search. The protein sequence homologous search uses Hirschberg algorithm for the pair-wise sequence alignment. The performance of the homologous search using the MPI-Pthread is compared to the implementation using pure message passing programming model MPI. The evaluation results show that there is a 50% decrease in computing time when the parallel homologous search is implemented using MPI-Phtreads compared to when using MPI.

Key-Words: - Parallel Programming Model, MPI, Pthreads, Homologous search, Hirschberg algorithm, Protein sequence data

1 Introduction

The current technology in parallel computing has made it possible to achieve better computing time with less cost. We have take advantage of the parallel methods in achieving our goal of producing a fast database sequence homologous search algorithm.

Shared Memory Multiprocessors (SMP) machines are now available to the mass with the introduction of cheap multiprocessor personal computers. These low-cost multiprocessors can be clustered together to create a new parallel computing platform call CLUMPs (Clusters of SMPs)[1] which is a hybrid of shared memory and distributed computing platform. The hybrid parallel computing platform allows user to implement data parallelism at large and medium grain level. The large grain parallelism is implemented using MPI, a message-passing interface for communication between processors, and the medium grain parallelism is implemented using Phtread/OpenMP, a shared address space programming model.

Protein sequence database data are growing exponentially. This sets the needs for faster tools to do sequence analysis. This process includes protein sequence homologous search given a query sequence. The search process uses the protein sequence alignment algorithm as its basic operation. The most optimal protein sequence alignment algorithm for protein sequence homologous search uses the dynamic programming method. One known algorithm is Smith-Waterman Algorithm which is O(mn) in time and space. The space saving algorithm for sequence alignment is Hirschberg Algorithm. We take advantage of the hybrid parallel computing platform to implement a protein sequence homologous search algorithm.

In this paper, we present our work on parallel Hirschberg on the Hybrid parallel computing platform using the hybrid parallel programming model. We show the advantage of our work compared to pure message passing programming model. The rest of the paper is organised as follows. Section 2 briefly describes the existing parallel programming model and introduces the hybrid parallel programming model. Section 3 discusses Hirschberg algorithm and the methods we use to parallelise the algorithm. Section 4 shows the experimentation, which includes the experiment environment, performance metrics and numerical results of the experiment. The conclusion and future work are presented in Section 7.

2 A hybrid Programming Model : MPI-Pthread

The hybrid model in this research is implemented at two levels of parallelism following closely to the implementation by [3]. The message

passing model is used to pass message between the SMP nodes and the shared memory programming model is used to create threads in the processors inside SMP.

3 Protein Sequence Homologous Search

The advancement in DNA sequencing technology has produced a large amount of sequence data. Raw protein sequences data are being deposited in this databases before being processed to produce other forms of database such as the domain or family databases. The growth of protein sequence data in Swiss-Prot is exponential between 1995 and 2007 .Because of these phenomena, computers have become indispensable to biological data analysis. One of the branches in biological data analysis is protein sequence analysis.

One of the processes in protein sequence analysis is protein sequence homologous search. The output of this homologous search, which is categorised as homologous sequences, is used to determine the structure and function of a newly found sequence. Homologous sequences are sequences that share the same ancestors. Homology cannot be quantified but it is usually based on percentage similarity between two sequences but two similar sequences do not imply that they are homologous [3]. Protein sequence comparison algorithms are used to find similar sequences in the database. Similar sequences are determined by the number of matching characters between the two compared sequences divide by the length of the longer sequence of the two. The most used protein sequence comparison algorithm for homologous search is pair-wise protein sequence alignment. The input to sequence homologous search engine are sequence database, new protein sequence as query and scoring matrices. The outputs are sequences that are similar to the input query sequence above certain threshold, which is determined by the user. These sequences are known as homologous sequences to the input query sequence.

Protein sequence homologous search is a simple algorithm that employs a pair-wise sequence alignment algorithm when comparing a sequence to the database. A flow-chart for protein sequence homologous search is given in Figure 1[4].

3.1 Protein Sequence Comparison

Protein Sequence comparison is the most important basic operation in sequence homologous search engine. It is defined as the problem of finding which part of a sequence is similar and which part is different.

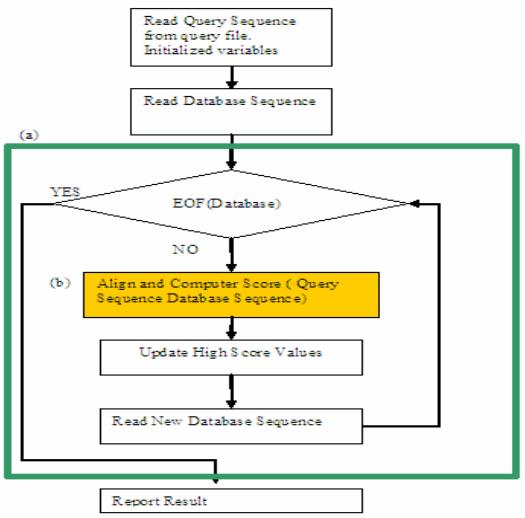

Figure 1: General flow-chart for Protein sequence Homologous search.

All living organisms are related by the process of evolution, a fact that motivates computer scientists to use sequence comparison to search for similar sequences in the database [5]. This fact implies that protein sequences of a closely related species have high similarity in terms of the amino acids that build up the sequences.

The most used sequence comparison algorithm is pair-wise sequence alignment algorithm. Smith-Waterman algorithm, which is a dynamic programming based algorithm, is the most optimal protein sequence alignment algorithm but the most compute intensive and uses large space to create a similarity matrix. In this research we implement Hirschberg algorithm which is the space saving version of dynamic programming based sequence alignment.

3.1 Hirschberg Algorithm

Hirschberg algorithm **[**6**]** is a recursive algorithm but still uses the dynamic programming technique in solving sequence comparison. It divides the similarity matrix into smaller blocks and calculates each portion differently. The basic operations for this algorithm is similar to the Smith Waterman algorithm [7]. The difference is Hirschberg uses divide and conquer to calculate the matrix hence uses less space. The first step in Hirschberg algorithm is to divide the similarity matrix into two main parts . After splitting, the Smith-Waterman algorithm is execution on both portion of the similarity matrix from two different

directions. The first half is executed normally from cell SIM[0,0] to cell SIM[m/2,n]. The second of the matrix is executed in reverse from cell SIM[m,n] to cell SIM[m/2+1,0]. The filling of each cell in the similarity matrix is as follows:

$$M(x, y) = \max \begin{cases} M(x-1, y) + g \\ M(x, y-1) + g \\ M(x-1, y-1) + s \\ 0 \end{cases}$$

3.2 Parallel Implementation

Parallelizing protein sequence homologous search can be done at two points

1 The main: at this point, the database is partitioned and distributed to different processors and a search algorithm is executed against this data. This level uses MPI to distribute the data.
2 At the pair-wise sequence alignment :at this level, the similarity matrix used in the protein sequence comparison algorithm is partitioned and distributed to different threads. This level uses Pthread which is a shared memory programming model.

3.2.1 Large Grain Parallelism using MPI

The pseudo-code for the parallel version for the first implementation using MPI is shown in Algorithm 1 (for master node) and Algorithm 2 (for slave node). The major changes from the sequential algorithm shown in Figure 3 is at the main loop . The parallel version first partitioned the data before the loop for comparing the protein sequence is executed.

The data parallel method is used in parallelizing the search algorithm at the database level. The target database is partitioned using static partitioning into portions of similar size and distributed among the processor nodes. Each processor only has to search its portion of the database. When the search finishes only one node will keep the search results and a global sort is done on this results. This is static load balancing.

The master node has two tasks to handle. At the initial stage, it will partition the database. The second task is to calculate the similarity values for the sequences belong to the master node. At the same time the master node waits for slaves to send results and request new workloads. The issue arises in this implementation is what is the best grain size to balance between communication time and computation time.

To overcome the problem of load imbalance when some workstations have more workload, a manager-worker approach has been taken. In this approach, the database is partitioned into smaller portion (usually the number of blocks is more than the number of nodes available) and distributed to workers and when a worker has finish its task it will signal the manager for more task. In this approach, the faster processor will have more task than the slower processor. However, the granularity has to be taken in consideration, the smaller the granularity the better compute time but with the increase of communication time.

Master: Partition_Data(Database,Number of Processors
1 Get database size
2 portion_size = databasesize/numberofprocessor
3 for(i =1 to number of processors)
i. send(startadd[i] to slave[i]);
4 Get query sequence and Broadcast to slaves
5 For all sequence belong to master
i. Hirschberg(querysequence, database sequence)
ii. Save results where similarity value > threshold
6 Recv(results from other slave)
7 Combine results and output

Algorithm 1 : Master algorithm for Parallel Protein Sequence Search Algorithm

Slave
1 Receive(start_address, portion_size)
2 Receive querysequence
3 For all sequences belongs to this slave
a. Hirschberg(querysequence, databasesequene)
4 Send (similarity result) to master

Algorithm 2 : Slaves algorithm for Parallel Protein Sequence Search Algorithm

In our approach, the database is partitioned statically at the beginning of the computation with fixed size according to the number of nodes. However when the portion involved has a border line cases then the portion before it will overtake the computation.

3.2.2 Medium Grain Parallelism using Pthread

The challenge in implementing parallelism at the similarity matrix is data dependency. After the first row and first column has been initialized, the calculation of all other entries in the similarity matrix is dependent on the previous entries. The

three previous cells are one from the cell above (same column but previous row) one from cell on the right (same row but previous column) and one from the diagonal cell (previous row and previous column). Given the data dependency, works has to be done to parallelized row by row, column by column and anti-diagonal by anti-diagonal. All the parallelization methods lead to an expensive communication overhead [8].

In our solution, we propose a method to divide the similarity matrix into two blocks which is to be distributed to two processors on shared memory multiprocessor. Each block is independent of each other. The calculation of all cells in the first block is done from the first row and first column to cell in the last row and last column. The calculation of the second block is from the cell in the last row and last column to the cell in the first row and the first column. Finally the main thread will merge the the last row of the first portion and the first row in the second portion and produce a maximal value.

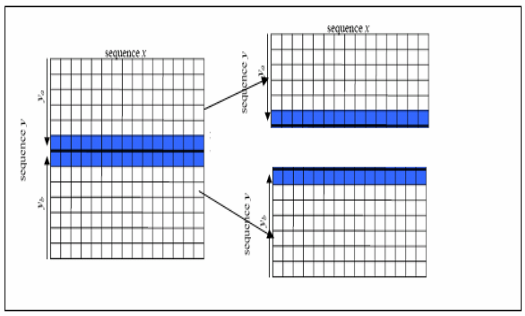

Figure 2: Distribution of similarity matrix to threads

3.3 Data Distribution

The input database is divided into p-size portions and the starting address and the size of the database to be processed is sent to the slaves. MPI structure is used to keep the information to avoid multiple send and receive and this will decrease the communication between master and slaves. This type of partitioning is appropriate when the load of each processor is balanced. Though the length of each protein sequence in each portion is different, we argue that the load balance is maintained with the static partitioning. This is at par with the bucket partitioning[9]. This is possible as the cluster of SMPs we used is homogenous and all the nodes are of the same CPU power and the memory are of the same size. Each static portion has different number of sequences. When the length of the protein sequence is long then the number of sequences is in the portion is less. In Figure 3, we presented of how we partitioned the database and distributed among the precessors. The query is first broadcast to all the processors. Then the starting address of each portion and size of each portion is distributed to the slaves.

The problem with static partitioning method is that a protein sequence might be cut in the middle which makes the alignment of the first sequence produces false result. To overcome this problem, each individual processor checks for this border case and skips this sequence if the sequence is not a full protein sequence. This problem also occurs at the end of the portion. To solve this problem, the processor will keep accessing the protein sequence until the end which is beyond its end address. The processor (n- 1) handles the borderline case. The load imbalance occurs only when the border string is very long.

The communication overhead at this level is very minimal. The communication is at two points, when sending the data to be processed by slaves and when receiving the results from slaves. The first communication is the point of sending the data to all processor as shown at point ***a*** in Figure 4. The second point is at point ***b*** in Figure 4 when the slaves send the result back to the master processor.

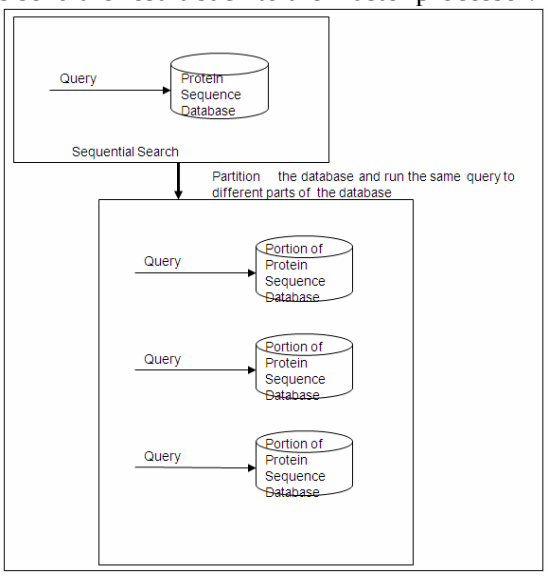

Figure 3: Parallel Protein Sequence Search Algorithm using Static Database Partitioning.

Homologous search algorithm is an embarrassingly parallel computation algorithm. The algorithm exploits the MIMD architecture by distributing the search algorithms among the nodes (processor) in the clusters. Each of the nodes in the clusters of SMPs independently processes the query against a portion of the database. There is no communication involve except at the initial level and final stage. The master slave model of parallel computation is used at this stage.

At the initial stage, a master process will enquire the size of the database to be queried and the

number of processor involved in the processing. Then the master process statically partitioned the database into portions of equal size. This is static partitioning as in [12]. The size is strictly on the byte size. When the partitioning is done, there will be cases when the partitioning begins at the middle of the protein sequence and there will be cases when the partitioning ends at the middle of the protein sequences. These cases are handled individually during the processing each individual portion.

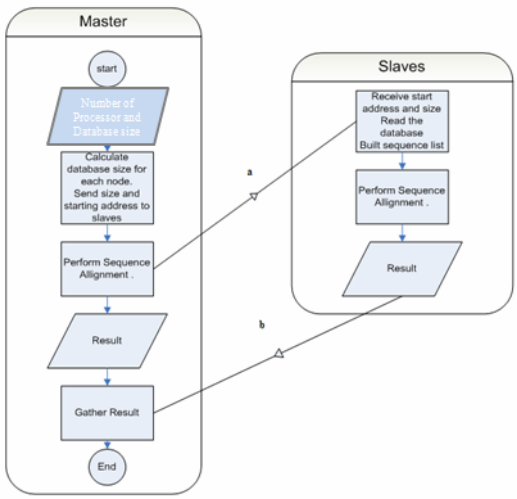

Figure 4 : Communication between Master and Slaves

4 Experimentation

4.1 Experimentation Environment

The experiments were executed on the Sun Fire cluster system available at the Parallel and Distributed System Lab at the School of Computer Sciences, Universiti Sains Malaysia. The Sun Fire Cluster consist of five machines where one machine is the server node and the other four are child nodes. Each machine is a SMP machines with two multiprocessors. The cluster is interconnected using the Ethernet. The limitation of Sun Fire cluster is that it is shared by many users around the campus, so the result is unstable when the load of the machine is high. To get a stable results, we tested the algorithms during midnight and set a priorities to the algorithm manually and run a few times and the results are then averaged out.

We downloaded a version of Swiss-Prot and keep it on the server to avoid using the Internet to access public databases,. The largest database we downloaded is Swiss-Prot version 54, which is downloaded in 2006 that consists of 280,000 protein sequences. All the input data are selected randomly from this database.

4.2 Performance Metrics

The study of the performance of parallel algorithms involved one performance metric. Execution time of a sequential algorithm T_1 is defined as the elapsed time between the beginning and end of running program on a single processor. The execution time of a parallel program T_p is the elapsed time between the beginning of a parallel execution and the moment the last processor stops processing. The elapsed time of the process is the time to do protein sequence comparison between the query sequence and the sequences in the database.

Performance Gain

Performance gain(PG) of an algorithm (called algorithm2) from another algorithm (called algorithm1) is a measure of percentage of performance difference in terms of computing time when running the algorithm1 and algorithm2 . It is calculated by getting the difference between computing time of algorithm1, $T_{algorithm1}$ and computing time of algorithm2, $T_{algorithm2}$. This difference is divided by the computing time of algorithm1 mathematically,

$$PG(\text{Algorithm2}) = \frac{T_{algorithm2} - T_{algorithm1}}{T_{algorithm1}} \times 100$$

4.3 Numerical Results

Figure 5 shows the compute time of MPITH and MPIH for query length of 200 and 1000. The compute time improvement when query length is 1000 is much better than the query time improvement when the query length is 200.

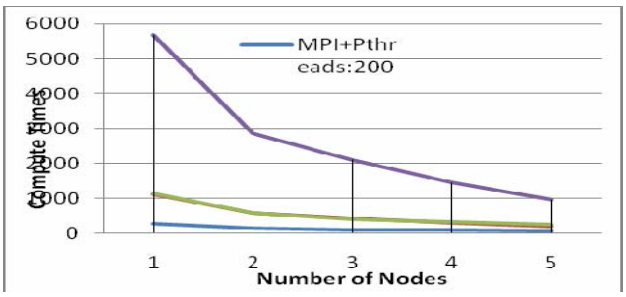

Figure 5: Compute time for query length of 200 and 1000 for MPI and MPI+Pthread implementation

We compare the performance gain for querying the database with varying length. The comparison is made with the following condition:
When MPI is running on two processors(p), we run MPI+Pthreads on one node(n). Mathematically

MPI: MPI+Pthreads is 2p: n.

Table 1: Performance gain of MPIMTH over MPIH

	MPIH:4 MPIMTH:2	MPIH:6 MPIMTH:3	MPIH:8 MPIMTH:4	
Query Length	Gain(%)	Gain(%)	Gain(%)	Gain(%)
200	53.85	53.52	52.94	47.89
400	57.84	57.45	56.68	52.82
600	59.06	59.12	58.38	53.95
800	59.85	59.81	59.18	54.96
1000	60.33	60.18	59.66	55.76

The reason is that the SMP nodes has two processors for each node. Table 2 shows the performance gains of MPI+Pthread over MPI. From the results, there is over 50% gain in performance when using the hybrid implementation. Although the percentage gain decreases as the number of nodes /processors increases, there is still a performance gain. This is due to the portion size gets smaller that fits into the cache of each node and can lead to linear speed up even by using MPI alone.

5 Conclusion

Base on our experimental results, we can conclude that for the dynamic programming based Hirschberg algorithm, the hybrid implementation give a better performance over the MPI solution. We obtain these results using MPI and Pthread on a cluster of SMPs to take advantage of the share memory architecture of Sun SMP machines. With the coming trends in Personal Computer with multiprocessors, this provides a cheap solution to the scientist community.

Currently we have managed to secure a grant that would allow us to buy low cost server with 2 X quadcore processors. We plan to extend this work to Smith-Waterman algorithm that is the most used sequence alignment algorithm with optimum solution but the most compute intensive.

6 Acknowledgements

This research is funded by E-Science titled " Parallel Sequence Alignment and Clustering Algorithms for Sequence Analysis of Fish Species" account number 01-01-05-SF0052 granted by the Ministry of Science, Technology and Innovation, Malaysia.

References:

[1] Frank Cappello and Olivier Richard, *Intra node parallelization of MPI programs with OpenMP*, Report TRCAP, http: // citeseer.ist.edu /cappello98intrahtml , 1998.

[2] Holger Brunst and Bernd Mohr, *Performance Analysis of Large Scale OpenMP and Hybrid MPI/OpenMP Applications with VampirNG*, In Proc. For IWOMP, 2005

[3] Lisa Holm and Chris Sander, Removing near-neighbour redundancy from large protein sequence collections, Bioinformatics, Vol14, 1998, pp 423-429.

[4] Vipin Chaudary,Feng Lui,Vijay Matta, and Lawrence T. Yang, *Parallel Implementation of Local Sequence Alignment:Hardware and Software* in *Parallel Computing for Bioinfomatics and Computational Biology* edited by Albert Y. Zomaya, Wiley Series on Paralle and Distributed Computing 2006.

[5] J. Cohen, *Molecular Biology Viewed from A Computer Scientist Perspective*, ACM computing Survey, 1999,pp 122-158

[6] D.S. Hirschbegr, *A linear Space Algorithms for Computing Maximal Common Sub-Sequences*, Communication of the ACM, Vol 18, No.6, 1975,pp 341-343.

[7]T.F. Smith and M. Waterman, *Identification of common moolecular subsequences*, Journal of Molecular Biology,147,1981,pp1950197

[8] W.S. Martin, J.B. del Cuvillo, F.J. Useche, K.B. Theobald, and G.R. Gao, *A Multithreaded Parallel Implementation of a dynamic programming algorithm for sequence comparison*, Pasific Symposium on Biocomputing, 2001.

[9] T.K. Yap,O. Frieder and R.L. martino,Parallel Computation in Bio Sequence Analysis,IEEE Transaction on Parallel and Distributed Computing, 9(3), 1998, 283-294.