COMP 7850 Advances in Parallel Computing

Project No. COMP 7850 2010
S.M. Al Mamun
Constructing Phylogenetic Tree using Neighbour Joining Algorithm with MPI

S.M. Al Mamun
Depatment of Computer Science
University of Manitoba*

December 30, 2010

Abstract

Phylogenetic tree construction is one of the most important and interesting problems in bioinformatics. Constructing an efficient phylogenetic tree is always been a research issue. It needs to consider both the correctness and the speed of the tree construction. There are several strategies to determine the best one, based on a certain criteria. In this report, I implemented the Neighbour-Joining algorithm, using MPI for constructing the phylogenetic tree. Performance is quite impressive, comparing to the best sequential algorithm.

1 Introduction and Motivation

A phylogenetic tree shows the evolutionary relationship among the biological species. There are several well-established strategies to build a phylogenetic tree. Neighbour Joining algorithm is one of them, which is frequntly used

*mamun@cs.umanitoba.ca

to reconstruct a phylogenetic tree based on the distance between each pair of taxa, where taxa is group of organisms. Parallel implementation of this algorithm can make computationally efficient tree for huge amount of data.

2 Background and Related Work

Saitou et al. [3] first designed neighbour joinging approach, which is an iterative algorithm. Neighbour joining is also used in several multiple sequence alignment package. Thompson et al. [6] used neighbour joining algorithm in Clustalw, which is widely used for aligning multiple sequences. Li [2] developed a parallel version of Clustalw, using message passing interface. RapidNJ and NINJA are another two implementations of neighbour joining algorithm. Stamatakis et al. [5] developed another package named RaxML for phylogenetic tree construction using neighbour joining algorithm. Sheneman et al. [4] designed a faster version, named as Clearcut which is based on relaxed neighbour joining algorithm.

3 Specific Problem Statement

Constructing the phylogenetic tree with high performance has become one of the major problems in computational biology. As the sequence becomes larger, huge amount of time is needed to calculate. For example, an input with 50,000x50,000 may not even computed by a week. If the same input is given to a parallel implementation, it may be done by a day. It may be finished by several hours even, if the code is highly optimized.

4 Solution Strategy and Implementation

Neighbouring algorithm is hierarchical clustering algorithm. It takes a distance matrix,D as input where d_{ij} is the distance between two clusters named i,j. Neighbour joining repeatedly joins pair of closest clusters. In some of the previous works, biological sequences were used as input. From the sequences, they calculated the distance matrix. In some other implementations, designers directly provided distance matrix. In this project, I also emphasized on the neighbour joining algorithm only and considered distance matrix as direct input. It made my work easier and durable.

A NxN distance matrix is used as input where each value is a distance between two species. The root processor reads the size of matrix and broadcasts it to other processors. Each processor then computes the local sum, r_j

and exchanges the sums with MPI all-to-all. Again, all the processors search a minimum distance, $D_{i,j}$ and exchange their vlaue with another all-to-all MPI call. A global minimum distance is calculated to construct the tree. Each processor computes the node, comparing with all of it's distances and the received minimum distance. At the end, root processor draws the tree. Here, I did not put any new idea. This approach is well-known in the field of phylogeny.

5 Experimental Framework

Programs were run in the Helium machine, which is a cluster of linux servers. It consists of one head node(Sun Fire X4200 machine) and five other computing nodes(Sun Fire X4600 machines), operating system is CentOS 5. Each computing node features 8-processor dual-core ccNUMA SMP (cache-coherent Non-Uniform Memory Access Symmetric Multiprocessing) computing model.

6 Results

Several distance matrices were used as input. I used 3 datasets of size 500x500, 1130x1130 and 2700*2700 from `http://www.daimi.au.dk/~cstorm/courses/AiBTaS_e05/project2/distance-matrices/`. For these datasets, the overall scenario can be found in Table 1,2 and 3.

Table 1: Execution Time and load per processor for 500x500 dataset

No. of processors	Execution Time	Load per processors
1	2.43186	500
2	2.61519	250
5	2.24316	100
10	2.08031	50
20	2.43741	25
50	3.50614	10

Table 2: Execution Time and load per processor for 1130x1130 dataset

No. of processors	Execution Time	Load per processors
1	25.4642	1130
2	17.3297	565
5	10.3612	226
10	7.52645	113
113	16.3894	10

Table 3: Execution Time and load per processor for 2700x2700 dataset		
No. of processors	Execution Time	Load per processors
1	337.428	2700
2	214.586	1350
5	102.67	540
10	57.05	270
20	60.0495	135
50	58.3676	54
100	73.2529	27

When the load per processor is too high then the performance is poor. As the load decreases performance gets better. Again, when load per processor is too low, the performance is again poor. That means, an average load gives the best performance. In figure 1, different timing are plotted for all these 3 datasets and also for the sequential program. For sequential program, I used quicktree [1] as it can build tree amazingly fast. For the same three datasets, quicktree gives the execution time of 0.55 second, 2.91 seconds, 27.11 seconds.

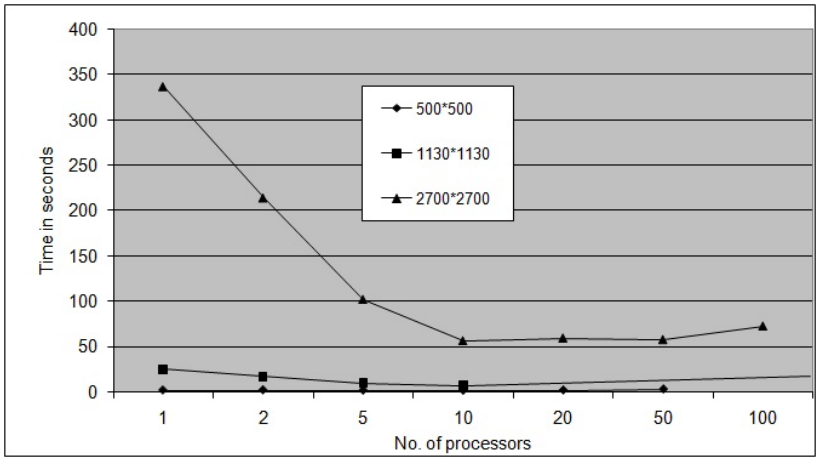

Figure 1: Processors Vs Time(in seconds)

Figure 2 shows the speedup and 3 shows the efficiency for various number of processors. In figure 2, the speedup is highest for 10 nodes. Then speedup starts decreasing. In figure 3, efficiency also decreases with the number of increasing nodes. From both of these figures, it can be said that parallel programming is a better choice for larger datasets. As it is highly iterative algorithm, the speedup and efficieny may be a bit low. And

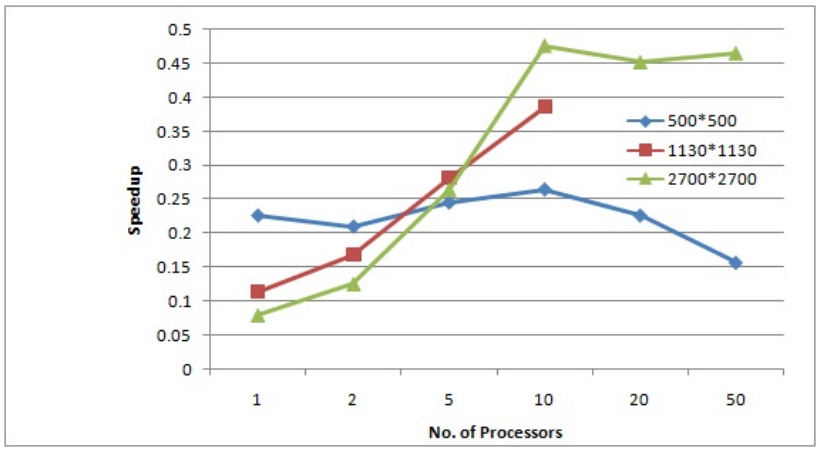

Figure 2: Processors Vs Speedup

figure 4 shows the cost for different datasets. Here, cost increases with the increse of processors.

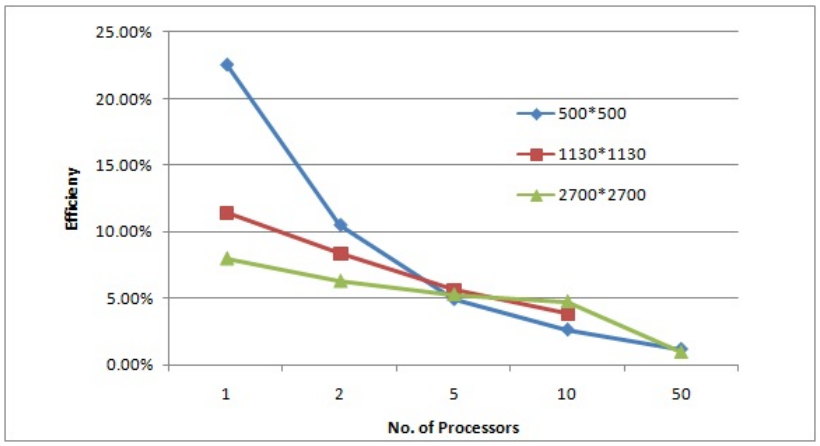

Figure 3: Processors Vs Efficiency

7 Conclusion and Future Work

I implemented the simple version of neighbour joining algorithm, even it can't handle when datasize is not divisable by the number of processors. The importance and the impact of load balance can be nicely understood

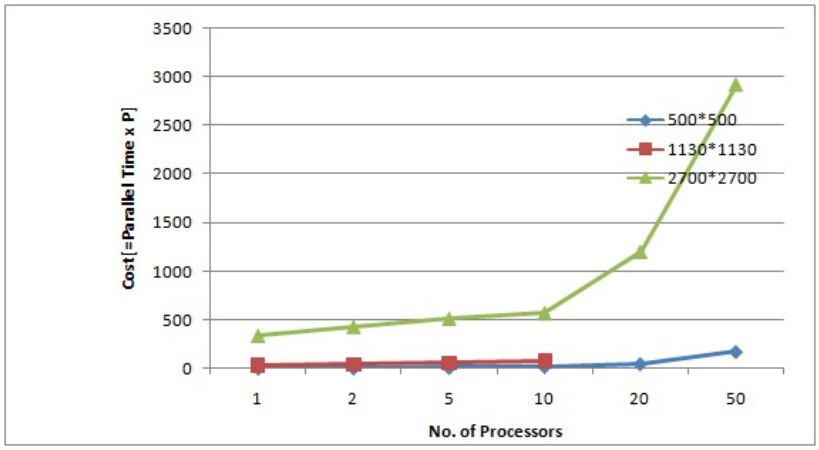

Figure 4: No. of Processors Vs Cost

through this work.

References

[1] Kevin Howe, Alex Bteman, and Richard Durbin. QuickTree: Building Huge Neighbour-Joining Trees of Protein Sequnces. *BIOINFORMATICS*, 18(11):1546–1547, 2002.

[2] Kuo-Bin Li. ClustalW-MPI: Clustalw Analysis using Distributed and Parallel Computing. *BIOINFORMATICS*, 19(12):1585–1586, 2003.

[3] Naruya Saitou and Masatoshi Nei. The Neighbour-Joining Method: A New Method for Reconstructing Phylogenetic Trees. *Mol Biol Evol.*, 4(4):406–425, 1987.

[4] Luke Sheneman, Jason Evans, and James A. Foster. Clearcut: A Fast Implementation of Relaxed Neighbour Joining. *BIOINFORMATICS*, 22(22):2823–2824, 2006.

[5] Alexandros Stamatakis. RAxML-VI-HPC: Maximum Likelihood-Based Phylogenetic Analyses with Thousands of Taxa and Mixed Models. *BIOINFORMATICS*, 22(21):2688–2690, 2006.

[6] Julie D. Thompson, Desmond G. Higgins, and Toby J. Gibson. CLUSTALW: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-Specific Gap Penalties

and Weight Matrix Choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.