Reconstruction of Large Phylogenetic Trees: A Parallel Approach

Zhihua Du and Feng Lin
BioInformatics Research Centre, Nanyang Technological University, Nanyang Avenue, Singapore 639798
Usman W. Roshan
College of Computing Sciences, Computer Sciences Department, New Jersey Institute of Technology, University Heights, Newark, NJ 07102

ABSTRACT

Reconstruction of phylogenetic trees for very large datasets is a known example of a computationally hard problem. In this paper, we present a parallel computing model for the widely used Multiple Instruction Multiple Data (MIMD) architecture. Following the idea of divide-and-conquer, our model adapts the Recursive-DCM3 decomposition method (Roshan *et al*., 2004) to divide datasets into smaller subproblems. It distributes computation load over multiple processors so that each processor constructs subtrees on each subproblem within a batch in parallel. It finally collects the resulting trees and merges them into a supertree. The proposed model is flexible as far as methods for dividing and merging datasets are concerned. We show that our method greatly reduces the computational time of the sequential version of the program. As a case study, our parallel approach only takes 22.1 hours on four processors to outperform the best score to date (found at 123.7 hours by the sequential Rec-I-DCM3 program (Roshan *et al*., 2004)) on one dataset. Developed with the standard message-passing library, MPI, the program can be recompiled and run on any MIMD systems.

Keywords: phylogenetic tree, maximum parsimony, parallel, divide-and-conquer, MIMD

Availability: This program is available from the authors.

Contact: duzhihua@pmail.ntu.edu.sg

INTRODUCTION

A phylogenetic tree illustrates the evolutionary relationships among a group of organisms, or among a family of related nucleic acid or protein sequences; e.g. how this family might have been derived during evolution. It plays a fundamental role in many biological problems such as multiple sequence alignment, protein structure and function prediction, and drug design (Bull and Wichman, 2001).

There are two general categories of methods for calculating phylogenetic trees: distance-based and character-based. The distance-based methods compute a matrix of pairwise distances between sequences in an alignment, and then construct a tree based entirely on the odistance computations. Neighbor-Joining (Saitou and Nei, 1987), WEIGHBOR (Bruno *et al*., 2000), BIONJ (Gascuel, 1997), FASTME (Desper and Gascuel, 2002) and a latest approach considering maximum-likelihood estimated triplets of sequences (Ranwez and Gascuel, 2002) belong to this category. The disadvantages of distance-based methods include the inevitable loss of evolutionary information when a sequence alignment is converted to pairwise alignment (Steel, 1988) and bad performance on large datasets.

Character-based methods examine each column of the alignment separately and look for the tree that best accommodates all of this information, such as maximum parsimony (MP) (Camin and Sokal, 1965) or maximum likelihood (ML) (Felsentein, 1981). MP chooses tree that minimizes number of changes required to explain data. ML, under a model of sequence evolution, finds a tree that gives the highest likelihood of the observed data. Character-based methods are information rich for there is a hypothesis for every column in the alignment. However, the MP method is NP-hard. ML has unknown complexity (Steel, 1994) and is hard to solve in practice. Primary sources of phylogenetic tree construction software include PHYLIP (available at http://evolution.genetics.washington.edu/phylip.html), MrBayes (Huelsenbeck and Ronquist, 2001), PAUP (Swofford, 2002), and TNT (Goloboff 1999).

Reconstructing optimal MP or ML phylogenies on large datasets is a particularly challenging task. Many of these datasets involve thousands of taxa. Among the current heuristic techniques for solving MP on large datasets, TNT performs the best (Goloboff 1999, Roshan 2004b, Hovenkamp 2004, Meier 2005, Giribet 2005). In addition to a very fast implementation of hill-climbing heuristics, TNT implements other search strategies, such as divide-and-conquer and genetic algorithms, which allow the analysis of large datasets in a reasonable time limit (much faster than other software packages).

A different class of methods for solving MP (and ML) on large datasets are Disk Covering Methods (DCM) (Huson *et al*., 1999; Warnow *et al*., 2001; Nakhleh *et al*., 2001; Roshan *et al*., 2004). DCMs are divide and conquer methods which divide the problem into smaller subsets, reconstruct trees on the subsets using a *base method*, and then merge the subtrees to obtain a tree on the full dataset. DCMs are *booster* methods in the sense that they improve upon the base method by applying it smaller instances of the subproblem. It was previously shown that Rec-I-DCM3 was able to improve upon the unboosted default heuristics of TNT (Roshan 2004,2004b). The default TNT heuristic is a combination of its own divide-and-conquer strategy and genetic algorithmic techniques (Goloboff 1999).

In this paper, we present a parallel model for constructing phylogenetic tree in MIMD architecture

following the idea of DCM. The goal of our parallel model is to exploit the computational power of clusters with a distributed memory architecture, high network bandwidth and low message passing latency. Clusters of compute nodes have become popular in bioinformatics research labs and we would like our program to be widely used. In this connection, our model is designed to be flexible to employ any other methods for dividing and merging dataset.

METHOD

In the following parts, we call our method Parallel REC-I-DCM3 for ease of writing and reference, but note that other division and conquer ideas can be used. A schematic flowchart of the Parallel Rec-I-DCM3 is shown in Figure 1.

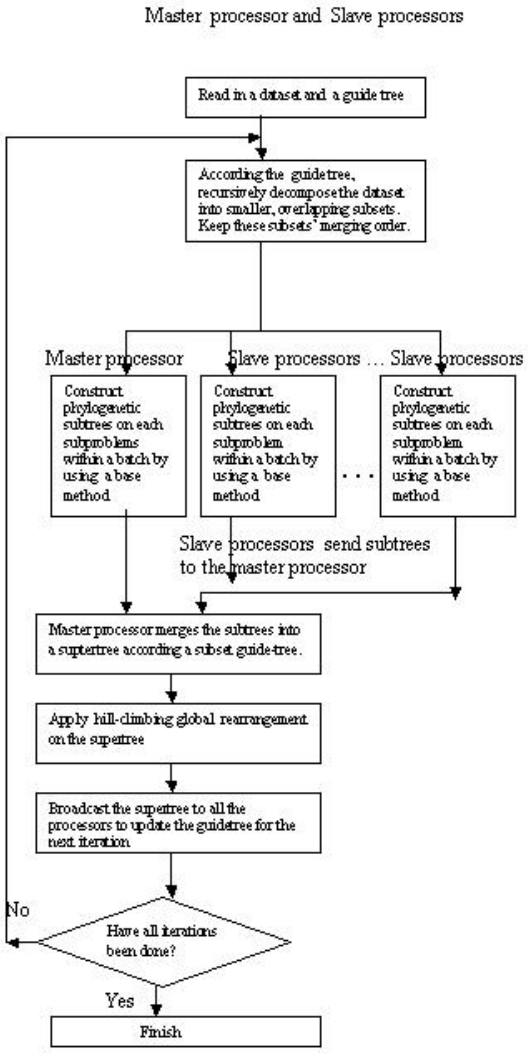


Fig. 1. Flowchart of Parallel Rec-I-DCM3

Parallel Rec-I-DCM3 is based on a master-slave architecture. It adopts recursive-DCM3 decomposition to recursively divide the datasets into smaller subproblems. Meanwhile, it keeps a subset guide-tree imposed by the recursive calls as the topology of the final merge. This part of code was executed both on a master processor and slave processors in order to reduce communication between processes. Then it uses a scheduling strategy called fixed-size chunking (Hagerup, 1997) to allocate batches of subproblems of one fixed size to available processors. After that, each processor computes subtrees on each subproblem within a batch by using a base MP method, TNT (Goloboff, 1999), and sends back these subtrees to a master processor. The master processor collects and combines the set of subtrees into a supertree. In our approach, the subtrees are merged according subset guide-tree order by using Strict Consensus Merger (SCM) (Huson *et al*., 1999) from the bottom up.). Following that, we apply a hill-climbing MP search, TBR (Maddison, 1991), on the supertree in order to do a global rearrangement. Finally, the supertree will be broadcast to all slave processors for iterative improvement.

The implementation of the Parallel Rec-I-DCM3 and subroutine RecDcm3, is presented in the pseudo code below.

Algorithm of Parallel Rec-I-DCM3

1. Problem Initialization
   1.1 Set S={s_0, …, s_{k-1}} of aligned biomolecular sequences. Set *k*=number of sequences, *n*=number of iteration, *b*=base heuristic (TNT), *T*=starting tree, *MS*=maximum subproblem size.
   1.2 Initialize variables, *myrank*=processors' rank, *nprocesors*=numbers of available processors.
   Initialize a subset guide-tree, *rurTree*, to record recursive calls as the topology for merge subtrees.
   Initialize, *allsubsets*, to save a total set of subproblems.
   1.3 Initialize MPI environment.
   MPI_Init(&argc, &argv);
   /*Find out my identity, *myrank*, in the default communicator*/
   MPI_Comm_rank(MPI_COMM_WORLD, &*myrank*);
   /*Find out how may processors, *nprocessors*, there are in the default communicator*/
   MPI_Comm_size(MPI_COMM_WORLD,&*nprocesses*);
2. For *n* iteration do
   2.1 /*Construct a recursive DCM3 decomposition using T|S as the guide tree, producing a total set of subproblems, *allsubsets* =A_0, A_1 ,…, A_{m-1} (*m* is the total number of subsets). Produce a subset guide-tree, *rurTree*, to keep the merge order. The *rurTree* is expressed in a string format that uses parenthesis to start and end subtree groups, commas to separate group members, and subproblems names to name tree leaves. */
   Call RecDcm3(*S*, *MS*, *b*, *T*).
   2.2 Do parallel step
      2.2.1 Each available processor reads a fixed-size chunk (*l*) of *allsubsets*, *l=m/nprocessors*..
      2.2.2 Each processor applies the base heuristic *b* to construct subtrees for subproblem A_i, using *T*| A_i as the starting tree and

letting the resulting tree be t_i. ($l * myrank \geq i > l * (myrank + 1)$).

2.2.3 /*Slave processors, *myrank* > 0*/
If (*myrank* > 0)
Send the resulting trees t_i, ($l * myrank \geq i > l * (myrank + 1)$) to the master processor.
/*master processor, *myrank*=0*/
Else
Receive the resulting trees from all other processors.

2.3 /*On the master processor, it uses a stack structure to read out subtrees t_0, t_1 ,…, t_{m-1} according subset guide-tree, *rurTree*.*/
Initialize char* *ptr*=*rurTree*;
Initialize tree *T*' ;
While(**ptr*!='\0'){
Switch(**ptr*){
Case '(':
/*push '(' into the stack*/
Push ( );
Break;
Case 't_i':
/*Push subtree t_i into the stack*/
Push(t_i) ;
Case ')':
Do{
/*pop out subtrees between '(' and ')' from stack*/
Set *temp*=Popout();
}while(*temp*!='(')
T'=Merge the subtrees using SCM
Push(*T*');
break;
default:
break;
}
treeptr++;
}//end of while

2.4 Apply tree bisection and reconnection (TBR) (Maddison, 1991) search starting from *T*' until we reach a global optimum. Let *T*' be the resulting global optimum.
2.5 Set *T*=*T*'.
2.6 Broadcast the new *T* to every available processor.

3. MPI_Finalize
Terminates the MPI environment.

Function RecDcm3(*S, MS, b, T*)

1. Problem Initialization
Input: Set of *k* sequences *S*, S={s_0, …, s_{k-1}}
Maximum subproblem size *MS*
Base heuristic *b*
Starting tree *T*
2. Construct a DCM3 decomposition using *T*|*S* as the guide tree, producing subproblems A_0, A_1 ,…, A_{x-1}.
For A_i ($0 \geq i \geq x$ -1)
If (*Ai*'s size>*MS*){
Let *T*|*Ai* be the result of restricting tree *T* to Ai for i.
/*Recursively compute the subsets for Ai*/
Call RecDCM3(*Ai, MS, b, T*|*Ai*).
}
Else{
Add *Ai* to *allsubsets*.
Update *rurTree*;
}

EXPERIMENTAL DESIGN

Overview We compare Parallel Rec-I-DCM3 to the sequential Rec-I-DCM3 in our experiments. We studied the performance in the initial 24 hours to determine which method finds better phylogenetic trees faster.

Datasets The experiments were done on six large datasets, some of which are available from obtained from http://www.cs.njit.edu/usman/RecIDCM3.html. The datasets we used are (1) 4114 16s rRNaA(1263 sites), (2) 6281 Eukaryotes ssu rRNA sequences from the European rRNA database, (1661 sites), (3) 6458 firmicutes bacteria 16s rRNA sequences from the RDP (1352 sites), (4) 6722 three-domain rRNA sequences from Robin Gutell (1122 sites) (Maidak et al., 2000), (5) 7769 three-domain + 2 organelle rRNA sequences from Robin Gutell (851 sites), (6) 11361 set of all bacteria ssu rRNA sequences from the European rRNA database (1360 sites) (Wuyts *et al.*, 2002), and (7) 13921 proteobacteria 16s rRNA sequences from the RDP (1359 sites) (Maidak *et al.*, 2000).

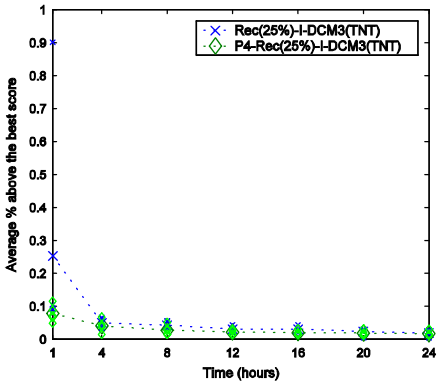
Implementation and Platform We have implemented the parallel algorithm on a cluster of 4 customized compute nodes, each with 4 Intel Itanium 733MHz processors, PCI-66 MHz I/O bandwidth and 266MHz data bus frequency. (Note that the full-instrumented parallel Rec-I-dcm3 requires a minimum of two processors).

RESULTS

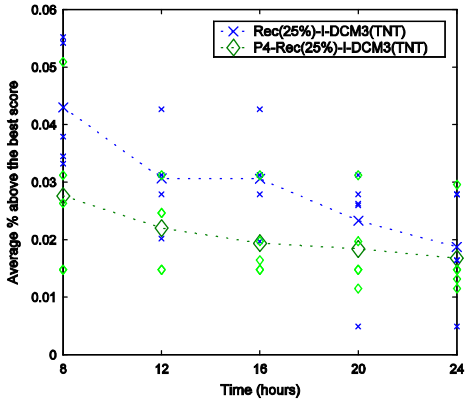
Comparing MP scores as a function of time

At first, we evaluated the performance of our parallelization by comparing the MP scores. In this experiment, our criterion is to look at how quickly a method gets within 0.01% of the best score for there is little change in the tree topologies if below the 0.01% threshold (Williams *et al.*, 2004). We ran five trials of the parallel program on 4 processors (P4-Rec-I-DCM3) and sequential program (Rec-I-DCM3) for up to 24 hours each and reported average scores. In our study, we defined the "optimal" MP score on each dataset to be the best score found over all five runs by the two methods in the 24 hours. We defined the "best" MP score on each dataset to be the best score found to date (http://www.cs.njit.edu/usman/dcm3/recidcm3_csb04_data.html). On dataset 1, 2, 3, 4 and 5, we took 1/4th of the full dataset size as maximum subset sizes. On dataset 6 and 7,

the maximum subset sizes are $1/8^{th}$ of the full dataset sizes. Our results in Figure 2 through 8 show the average MP scores above the best score, as a percentage of the best score on the given datasets. The results show that P4-Rec-I-DCM3 is better than Rec-I-DCM3 on every point in time. For a quick view, we only show one run's scores of P10-Rec-I-DCM3 on larger Dataset 4, 5, 6 and 7 in Figure 5, 6, 7 and 8. The same as we expected, P10-Rec-I-DCM3 is more quickly move close to the best score than P4-Rec-I-DCM3, especially in first eight hours.

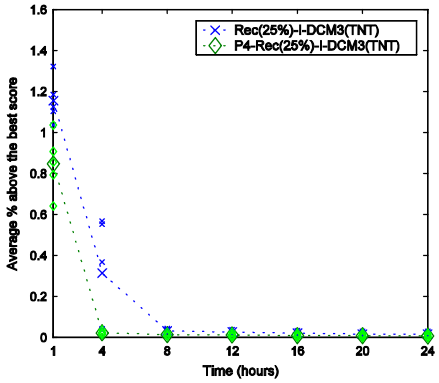


Dataset 1(4114 taxa, starting from hour 1 to hour 24)

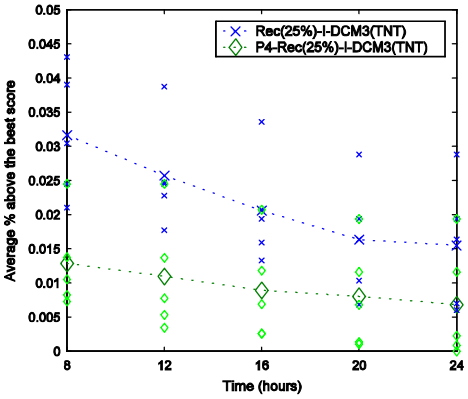


Dataset 1(4114 taxa, starting from hour 8 to hour 24)

Fig. 2. P4-Rec-I-DCM3 VS. Rec-I-DCM3 on Dataset 1. The graphs show the datapoints of all five runs of both methods.

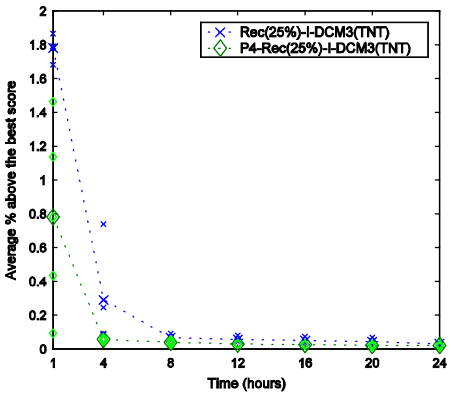


Dataset 2 (6281 taxa, starting from hour 1 to hour 24)

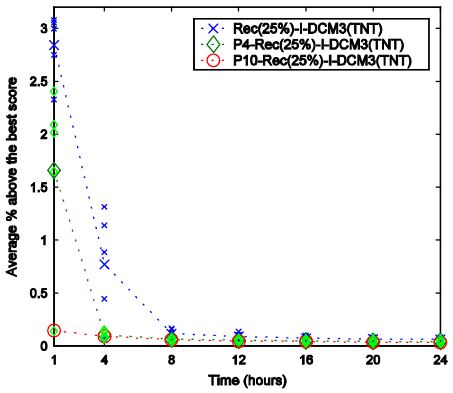


Dataset 2 (6281 taxa, starting from hour 8 to hour 24)

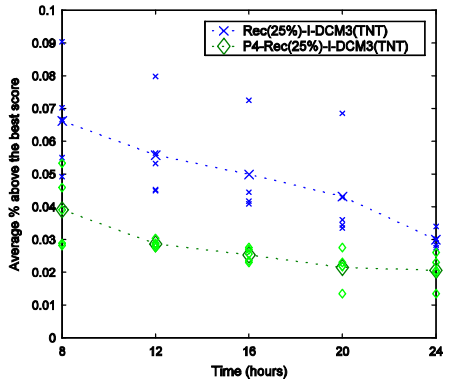
Fig. 3. P4-Rec-I-DCM3 VS. Rec-I-DCM3 on Dataset 2. The graphs show the datapoints of all five runs of both methods.



Dataset 3 (6458 taxa, starting from hour 1 to hour 24)

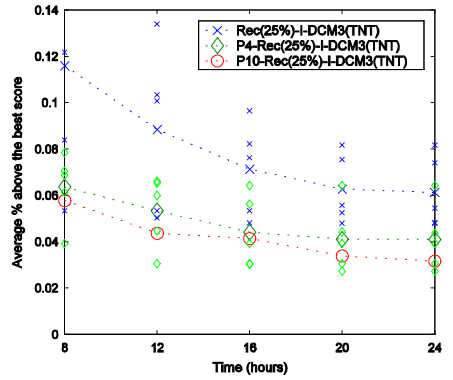


Dataset 4 (6722 taxa, starting from hour 1 to hour 24)



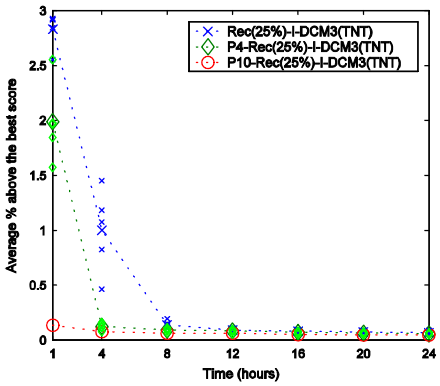
Dataset 3 (6458 taxa, starting from hour 8 to hour 24)

Fig. 4. P4-Rec-I-DCM3 VS. Rec-I-DCM3 on Dataset 3. The graphs show the datapoints of all five runs of both methods.

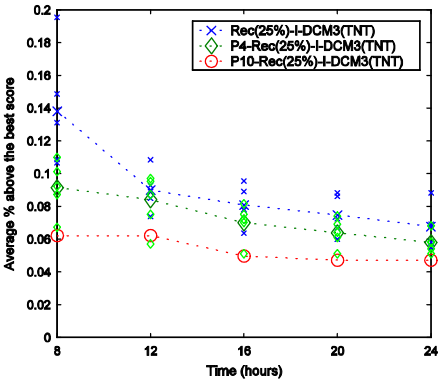


Dataset 4 (6722 taxa, starting from hour 8 to hour 24)

Fig. 5. P4-Rec-I-DCM3 VS. Rec-I-DCM3 on Dataset 4. The graphs show the datapoints of all five runs of both methods. P10-Rec-I-DCM performs better than the average P4-Rec-I-DCM and best Rec-I-DCM3 trials at all time points shown.

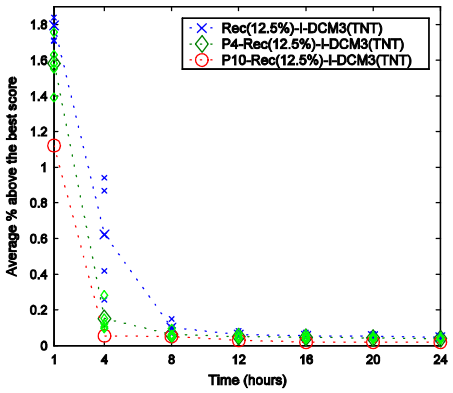


Dataset 5 (7769 taxa, starting from hour 1 to hour 24)

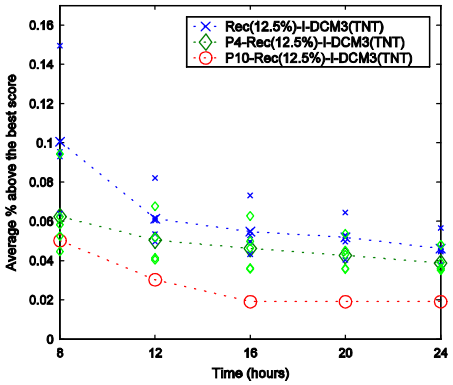


Dataset 5 (7769 taxa, starting from hour 8 to hour 24)

Fig. 6. P4-Rec-I-DCM3 VS. Rec-I-DCM3 on Dataset 5. The graphs show the datapoints of all five runs of both methods. P10-Rec-I-DCM performs better than the average P4-Rec-I-DCM and best Rec-I-DCM3 trials at all time points shown.

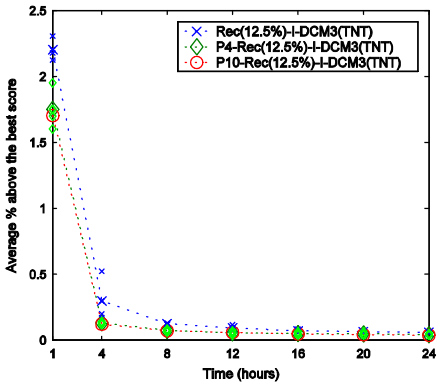


Dataset 6 (11361 taxa, starting from hour 1 to hour 24)

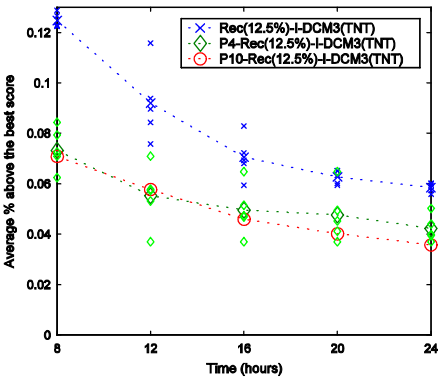


Dataset 6 (11361 taxa, starting from hour 8 to hour 24)

Fig. 7. P4-Rec-I-DCM3 VS. Rec-I-DCM3 on Dataset 6. The graphs show the datapoints of all five runs of both methods. P10-Rec-I-DCM performs better than the average P4-Rec-I-DCM and best Rec-I-DCM3 trials at all time points shown.



Dataset 7 (13921 taxa, starting from hour 1 to hour 24)



Dataset 7 (13921 taxa, starting from hour 8 to hour 24)

Fig. 8. P4-Rec-I-DCM3 VS. Rec-I-DCM3 on Dataset 7. The graphs show the datapoints of all five runs of both methods. P10-Rec-I-DCM performs better than the average P4-Rec-I-DCM and best Rec-I-DCM3 trials at all time points shown.

Comparing the score of the best run of serial and parallel

In Table 1 we compare the optimal score in 24 hours found by the five runs of Rec-I-DCM3 and P4-Rec-I-DCM3. From the table, we can see that except Dataset 1, the optimal score was always found by P4-Rec-I-DCM3.

Table 1. The optimal scores were found by two methods of five runs in 24 hours.

Dataset No.	Rec-I-DCM3	P4-Rec-I-DCM
1	60895	60899
2	232618	232580
3	156235	156213
4	91918	91899
5	99870	99866
6	272157	272142
7	241064	241010

Speedup of parallel over serial

Scaling of Parallel Rec-I-DCM3 Besides comparing MP score, we also conducted parallel speedup tests with these datasets range from 4114 sequences to 13,921 sequences. On dataset 1, 2, 3, 4 and 5, we took 1/4 of the full dataset size as maximum subset sizes. On dataset 6 and 7, the maximum subset sizes are 1/8 of the full dataset sizes. In order to show the scaling of Parallel Rec-I-DCM3, we executed it on 2, 4, 10 processors because the number of subsets from dataset 1 to 5 is ten. For uniprocessor performance, we used the sequential version as a baseline. We ran one iteration of the parallel and sequential code 5 times for each dataset and calculated the average execution time and maximum parsimony scores in order to explore the effect of non-determinism on program performance. The scaling behavior of parallel Rec-I-dcm3 is shown in Figure 9 and 10.

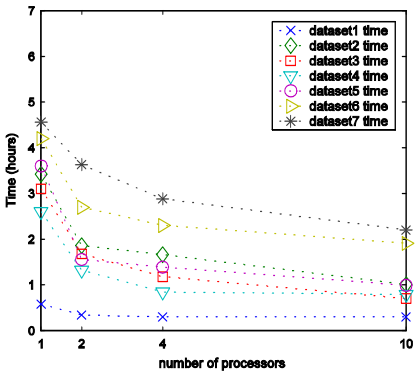


Fig. 9. Time to complete inferring phylogenetic trees from Dataset 1 to Dataset7 on 1, 2, 4 and 10 processors.

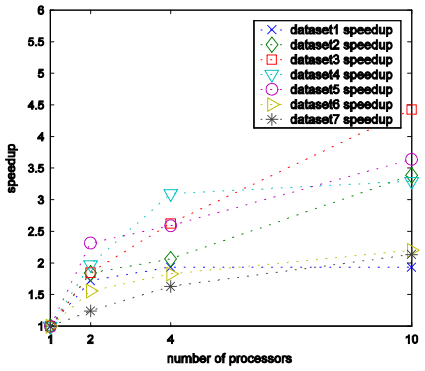


Fig. 10. Speedup of inferring phylogenetic trees from Dataset 1 to Dataset7 on 1, 2, 4 and 10 processors.

Our results show that the program performs well and exhibits good speedup. For example, the elapsed time of dataset3 is reduced from 3 hours on a single processor to 1.8 hours on 2 processors, 1.27 hours on 4 processors and less than 1 hour on more than 4 processors. The fairly flat curve of the elapsed time at the high end of processor number suggest that computational gain from further distribution of the subsets will be discounted by the overhead communication between the processes. Another factor limiting the scalability of this algorithm is the relatively few sequential portions of the program. One portion is global rearrangement by TBR search on the complete phylogenetic tree in order to reach a global optimal, which will take about half time of one iteration. The other is that SCM merges subtrees into a complete phylogenetic tree sequentially.

Comparison of best Rec-I-DCM3 against P4-Rec-I-DCM3 at 24 hours We want to see how long P4-Rec-I-DCM3 would take to attain the performance of Rec-I-DCM3. We also ran five trials of the parallel program on 4 processors and sequential program for up to 24 hours each and reported average scores. In the figure 11, we look at the ratio of time taken by P4-Rec-I-DCM3 to find the best average score to the time taken by Rec-I-DCM3 in 24 hours On Datasets 2-7, the improvement is about 2 times.

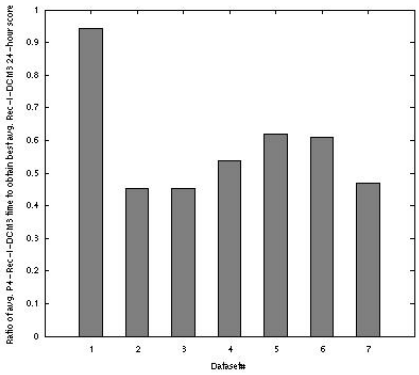


Fig. 11. Ratio of time taken by P4-Rec-I-DCM3 to find the best average score to the time taken by Rec-I-DCM3 in 24 hours.

Comparison of #iterations in the time limit We finally compared the number of iterations of P4-Rec-I-DCM3 with those obtained by the sequential program. Figure 12 shows that it obtained more iterations upon sequential program.

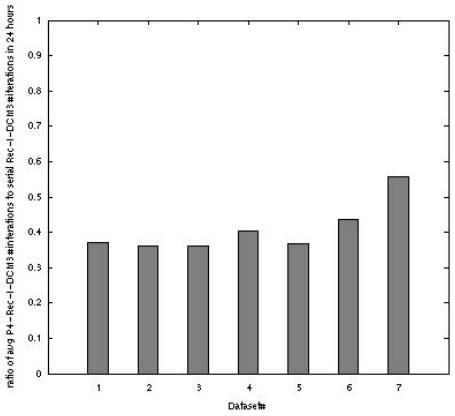


Fig. 12. Ratio of avg P4-Rec-I-DCM3 #interations to serial Rec-I-DCM3 #iterations on Dataset 1 to 7 in 24 hours (average over five runs)

This study explains why Parallel Rec-I-DCM3 can quickly come within a certain range of the best score. It runs more iterations than the sequential version.

Conclusion and discussion

We present an efficient parallel approach, Parallel Rec-I-DCM3, for constructing phylogenetic tree. It is targeted at clusters with distributed memory architecture, high network bandwidth and low message latency. It provides a fast and practicable approach for parallel inference of large phylogenetic trees containing up to 10,000 sequences. Based on our tests, Parallel Rec-I-DCM3 always finds a more optimal tree in less time than Rec-I-DCM3. Because Parallel rec-I-dcm3 is based on well-known and trusted software, it is easier for the research community to adopt.

Global rearrangement by TBR search on the complete phylogenetic tree is one of the factors to affect speedup. It would be interesting to propose a parallel method on this part.

Until now, Parallel Rec-I-DCM3 technique described in this paper mainly focuses on MP methods for constructing phylogenetic trees. ML is a harder problem than MP because it is not known how to compute the ML score of a given tree in polynomial time. By using our parallel model one could speed-up the tree construction with ML-based methods. According to our preliminary (and still unpublished) tests, the parallel Rec-I-DCM3 (ML) leads not only to a significant reduction of response times for large trees but also to a great improvement of final tree quality. To our knowledge the parallel Rec-I-DCM3 is among the fastest (if not literally the fastest) and most accurate approaches for inference of large phylogenetic trees with ML. Furthermore, it appears to be the only currently available program that is capable of handling

huge alignments (over 5.000 taxa) with relatively modest memory requirements and reasonably short inference times.

REFERENCES

Bruno,W.J., Socci,N.D. and Halpern,A.L. (2000) Weighted Neighbor Joining: A Likelihood-Based Approach to Distance-Based Phylogeny Reconstruction. *Mol Biol Evol*, 17: 189-197.

Bull,J.J. and Wichman,H.A. (2001) Applied evolution. *Annual Review of Ecology and systematics*, 32:183-217.

Camin,J. and Sokal,R. (1965) A method for deducing branching sequences in phylogeny. *Evolution*, 19:311-326.

Desper,R. and Gascuel,O. (2002) Fast and Accurate Phylogeny Reconstruction Algorithms based on the Minimum-Evolution Principle. *J Comput Biol*, 19:687-705.

Felsentein,J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *J Mol Evol*, 17:368-376.

Gascuel,O. (1997) BIONJ: an improved version of the NJ algorithm based on a simplemodel of sequence data. *Mol Biol Evol*, 14:685-695.

Giribet. G (2005) A review of "TNT: Tree Analysis using New Technology". *Systematic* Biology, 54(1): 176-178.

Goloboff,P.A. (1999) Analyzing large data sets in reasonable times: solution for composite optima. *Cladistics*, 15: 415-428.

Hagerup,T. (1997) Allocating independent tasks to parallel processors: an experimental study. *J. Parallel Distrib. Comput.*, **47**, 185–197.

Hovenkamp P. (2004) Review of TNT – Tree Analysis using New Technology, Version 1.0. *Cladistics*, 20:378-383.

Huson,D., Nettles,S. and Warnow,T. (1999) Disk-covering, a fast-converging method for phylogenetic tree reconstruction. *Journal of Computational Biology*, 6:369-386

Huson,D., Vawter,L. and Warnow,T. (1999) Solving large scale phylogenetic problems using DCM2. in proc. 7th int'l conf. *On intelligent systems for Molecular Biology(ISMB'99)*, pages 118-129. AAAI Press.

Huelsenbeck,J.P. and Ronquist,F. (2001) MYBAYES: Bayesian inference of phylogenetic trees. *Bioinformatics*, 17, 754-755.

Kimmen,S. (2004) Phylogenomic inference of protein molecular function: advances and challenges. *Bioinformatics*, 20,170-179.

Maidak,B. et al , (2000) The RDP (ribosomal database project) continues. *Nucleic Acids Research*, 28: 173-174.

Maddison,D.R. (1991) The discovery and importance of multiple islands of most parsimonious trees. *Systematic Biology*, 42(2): 200-210.

Meier R. and F. Ali (2005) The newest kid on the parsimony block: TNT (Tree Analysis using New Technology). *Systematic Entomology*, 30:179-182.

Nakhleh, L., Roshan,U., John,K.St. Sun,J. and Warnow,T. (2001) Designing fast converging phylogenetic methods. In proc. 9th Int'l Conf. On Interligent Systems for Molecular Biology (ISMB'01), volume 17 of *Bioinformatics*, pages S190-S198. Oxford U. press.

Ranwez,V. and Gascuel,O. (2002) Improvement of Distance-Based phylogenetic Methods by a Local Maximum Likelihood Approach Using Triplets. *Mol Biol Evol*, 19:1952-1963.

Roshan,U, Moret,B.M.E., Williams,T.L. and Warnow,T. (2004) Performance of suptertree methods on various dtaset decompositions. In O.R>P.Binida-Emonds, editor, Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life, Volumn 3 of *Computational Biology*, pages 301-328, Kluwer Academics.

Roshan,U, Moret,B.M.E., Williams,T.L. and Warnow,T. (2004) Rec-I-DCM3: A Fast Algorithmic Technique for Reconstructing Large Phylogenetic Trees, Proceedings of the IEEE Computational Systems Bioinformatics conference (ICSB).

Roshan,U. (2004b) Algorithmic techniques for improving the speed and accuracy of phylogenetic methods. *Ph.D. thesis*, The University of Texas at Austin

Swofford,D. (2002) PAUP*. Phylogenetic Analysis Using Parsimony(* and other mothods). Version 4.*Sinauer Associates*.

Steel,M.A. (1994) The maximum likelihood point for a phylogenetic tree is not unique. *Systematic Biology*, 43(4): 560-564.

Steel,M.A *et al*. (1988) Loss of information in genetic distances. *Nature*, 336, 118.

Saitou,N. and Nei,M. (1987) The nigehbor-joining method: a new method for reconstructing phylogenetic tree. *J Mol Evol*, 1987, 4:406-425.

Warnow,T., Moret,B.M.E. and John,K.St. (2001) Absolute convergence: True from short sequences. In proc. 12th Ann. ACM-SIAM Symp. *Discrete Algorithm* (SODA'01), pages 186-195. SIAM press

Wuyts,J., Van de Peer,Y. Winkelmans,T. and De Watchter,R. (2002) The European database on small subunit ribosomal RNA. *Nucleic Acid Research*, 30:183-185.

Williams,T.L., Moret,B.M.E., Berger-Wolf,T., Roshan,U. and Warnow,T. (2004) The relationship between maximum parsimony scores and phylogenetic tree topologies. *Technical Report TR-CS-2004-04*, Department of Computer Science, The University of New Mexico.