Available online at www.sciencedirect.com
ScienceDirect
Parallel Computing 32 (2006) 441–446
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Short communication

pNJTree: A parallel program for reconstruction of neighbor-joining tree and its application in ClustalW

Zhihua Du [a,*], Feng Lin [b]

[a] College of Information Engineering, ShenZhen University, Biological School, NanShan District, Shenzhen, GuangDong 518060, PR China
[b] BioInformatics Research Centre, Nanyang Technological University, Nanyang Avenue, Singapore 639798, Singapore

Received 27 February 2006; accepted 29 May 2006
Available online 2 August 2006

Abstract

Neighbor-joining (NJ) is a distance-based method for tree construction. It is the most widely used method with polynomial time complexity at present. However, a fundamental problem with the previous implementations of this method is its limitation to handle large taxa sets within a reasonable time and memory resources. In this paper, we present a parallel implementation, pNJTree, for fast construction of very large phylogenetic trees. In comparison, pNJTree gets near-linear speedup for large taxa sets. It can be used to improve the speedup of the parallelized ClustalW methods.

Keywords: Neighbor-joining; Multiple Instruction Multiple Data Architecture (MIMD); Phylogenetic tree; Multiple sequence alignment

1. Introduction

NJ method is a distance-based method for reconstructing phylogenetic trees, and computing the lengths of the branches of this tree. The algorithm was originally written by Saitou and Nei [3]. Studier and Keppler made improvement for the algorithm [4]. It firstly gives the proof of the algorithm, and second of all made a slight change to the algorithm which brought the efficiency down to $O(n^3)$. The NJ method is a greedy heuristic algorithm that joins at each step, the two closest neighbors that are not already joined. It is based on the minimum evolution principle. The NJ method is currently very popular, largely due to its application in the ClustalW [5]. The ClustalW algorithm has become the most popular method for multiple alignments. In the ClustalW program, the first phase is to do pairwsie alignment (Pairalign). Secondly, it generates a neighbor-joining 'guide tree' from pairwise distances (NJtree). This guide tree gives the order in which the progressive alignment will be carried out as the third phase (Malign). A serious problem, however, is that, for large taxa sets, the runtime of the NJ method becomes prohibitive. The need to analyse large taxa sets is necessary and increasing, for example, the National Center for Biotechnology Information's GenBank has more than 37

* Corresponding author. Tel.: +86 13428761373.
E-mail addresses: duzhihua@pmail.ntu.edu.sg, du_zhihua@yahoo.com.cn (Z. Du).

0167-8191/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2006.05.001

million sequence records as of August 2004, which has nearly doubled in size each year for the past decade.

Recently, Li [2], Ebedes and Datta [1] reported parallelization of the ClustalW algorithms for distributed memory architectures by using MPI [7]. They concluded that the sequential ClustalW implementation spends almost 96% running time in the first stage for pairwise alignment of the n input sequences. As each pairwise alignment is completely independent of all other pairwise alignments, the two programs has parallelized the phase of pairwise alignment and achieved near-linear speedup. As the phase of building NJ tree, Li has parallelized the phase of construction of NJ tree but did not get much better speedup. Ebedes and Datta did not parallel this phase at all. However, when handling larger taxa set, such as $n > 5000$, the second phase of construction NJ tree spends more than 30% running time instead of 4%. This is one of the reasons that the speedup of the two programs decreases when employing more processors for large taxa sets.

In this application note, we present a parallel algorithm, pNJTree, for implementation of the NJ method on a MIMD architecture[6] such as a cluster of compute nodes, using the Message Passing Interface. These clusters play an important role due to the advent of commodity high-performance processors, low-latency/high-bandwidth networks and powerful development tools in bioinformatics research labs and we would like to see our program be widely used. In the experimental results, we show that pNJTree can get near-linear speedup and its application in the parallelization of the ClustalW algorithm that gets better performance than other published methods. Our experimental results show that, for large data sets, we can achieve considerable reduction in computational time and inter-node communication overhead.

2. Methods

The NJ method begins with a distance matrix between the sequences. In each stage, the two nearest nodes are chosen and defined as neighbors in a tree. This is done recursively until all of the nodes are paired together. The nearest nodes can be defined by minimizing the expression $M_{ij} = (n-2)d_{ij} - (R_i + R_j)$, where d_{ij} is the distance between node i and j shown in distance matrix, R_x is the row sum over row x of the distance matrix. $R_x = \sum_{1 \leqslant k \leqslant n} d_{xk}$, and n is the remaining number of nodes adjacent to the root. When nodes i and j are joined, they are replaced with a new node, y, with distance to the remaining nodes given by $d_{yk} = (d_{ik} + d_{jk} - d_{ij})/2$. The method performs a search for minimum value of M_{ij}.

The pNJTree algorithm on a cluster of compute nodes is outlined by the flowchart in Fig. 1. pNJTree's task-scheduling strategy of parallel implementation is based on a master/slave model. One of the processor acts as master, scheduling and dispatching blocks of tasks to slave processors. Slave processors perform calculations specified by the algorithm.

The implementation of the pNJtree algorithm is described by the pseudo code below:

- Input: Distance matrix d of dimension $n \times n$
- Output: A NJ tree T leaf-labeled by $\{1, 2, \ldots, n\}$
- Algorithm: Parallel Construction of Neighbor-Joining Tree
  1 Master processor divides matrix d of dimension $n \times n$ into $p-1$ parts and distributes them to $p-1$ slave processors (p is the number of processors). Each slave processor has $n \times n/(p-1)$ elements of matrix d. n is the number of nodes, $N = n$.
  2 For $N > 2$ do
  {
    2.1 Each slave processor computes a block of R_x's for each x on its own matrix (block's length $= n/(p-1)$) $R_x = \sum_{k=1}^{n} d_{xk}$ (x represents the node for which we are computing now) ($d_{ii} = 0$). After finish the task, slave processors send back the block of R_x to master processor.
    2.2 Master processor collects all the nodes' R_x. And broadcast the whole $R_x (1 \leqslant x \leqslant n)$ to all slave processors.
    2.3 Each slave processor computes M_{ij} on its own matrix (for each $i < j$, since the matrix is symmetric) $M_{ij} = d_{ij} - (R_i + R_i)/(N-2)$. Find the minimum M_{ij}: $M_{\min i, \min j}$. And send back minimal min$i$,minj and $M_{\min i, \min j}$ to the master processor.

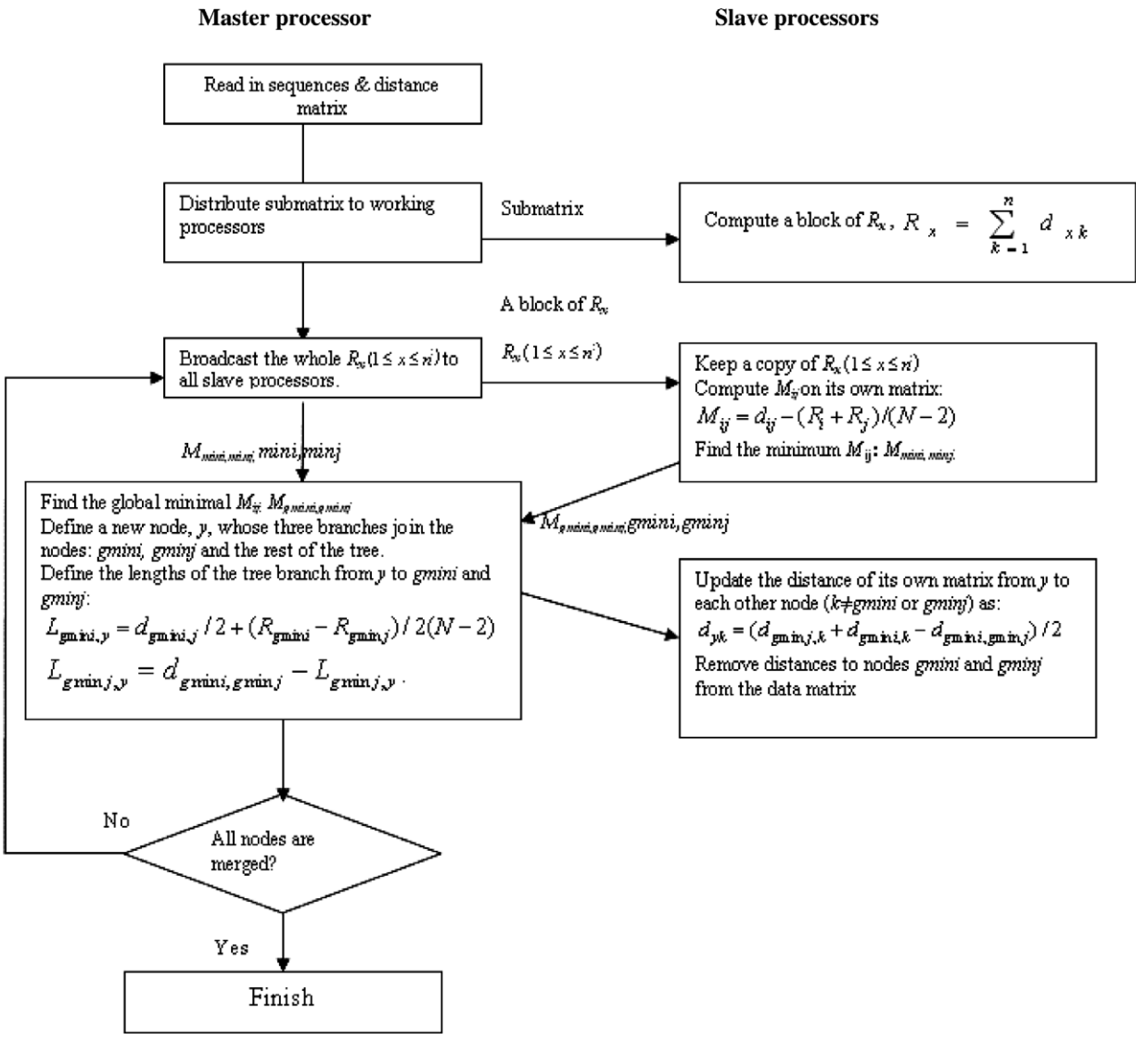

Fig. 1. Flowchart of pNJtree.

2.4 Master processor collects minimal min$i$, min$j$ and $M_{\min i,\min j}$ from $p-1$ slave processors, finds the global minimal M_{ij}.$M_{\text{gmin}i,\text{gmin}j}$, and broadcast $M_{\text{gmin}i,\text{gmin}j}$, gmin$i$,gmin$j$, to slave processors.

2.5 Master processor defines a new node, y, whose three branches join i, j and the rest of the tree. Define the lengths of the tree branch from y to i and j: $L_{\text{gmin}i,y} = d_{\text{gmin}i,j}/2 + (R_{\text{gmin}i} - R_{\text{gmin}j})/2(N-2)$ $L_{\text{gmin}j,y} = d_{\text{gmin}i,\text{gmin}j} - L_{\text{gmin}j,y}$. These distances are the lengths of the new branches.

2.6 Slave processors define the distance of its own matrix from y to each other node (k $\neq$ i or j) as: $d_{yk} = (d_{\text{gmin}j,k} + d_{\text{gmin}i,k} - d_{\text{gmin}i,\text{gmin}j})/2$.

2.7 Remove distances to nodes i and j from the data matrix, and decrease N by 1.

}

At the Bioinformatics Research Centre (BIRC), Nanyang Technology University, we have set up a multi-node compute cluster for our implementation. The cluster uses 8 customized AlphaServer ES45 compute

nodes, or server, each with 4 Alpha-EV68 1 GHz processors, 8 GB/s memory bandwidth and an interconnect PCI adapter capable of over 280 MB/s sustained bandwidth. Installed on each nodes is the Tru64™ UNIX v5.1a operating system. At the heart of the cluster is the Quadrics 128-port interconnect switch chassis, delivering up to 500 MB/s per node, with 32 GB/s of cross-section bandwidth and MPI application latencies less than 5 μs. A management server, based on an AlphaServer DS20E system, is used for redundancy and availability.

3. Experimental results

At first, we evaluate the performance of pNJTree with three large protein taxa sets of 6500, 7500 and 10,000 HIV sequences downloaded from NCBI. For uniprocessor performance, we used the sequential version of NJ method as a baseline. We ran five trials of our parallel program on 4, 8, 16 and 32 processors and reported average time. Note that, a processor working as a master did not involve the computation. The improvement of time scaling on different numbers of processors is presented in Figs. 2 and 3.

From this experiment, we can conclude that our parallel algorithm gets good performance with large number of taxa. In the figures, they can be observed that the parallel version scales up well. The fairly flat curve of the elapsed time at the high end of processor number suggest that computational gain from further division of the matrix will be discounted by the overhead communication between the processes. The fairly flat curve of the speedup at the high end of processor numbers suggests that computational gain from further division of the matrix will be discounted by the overhead communication between the processes. In addition, at the process of building a guiding tree, for the number of $n-1$ iterations, an array of R_x, $1 \leqslant x \leqslant n$, needs to be collected by the master process and broadcast to each process before utilizing it for computing the value, *Dist* Meanwhile, a global synchronization is needed in order to update the matrix for next iteration. So there are $n-1$ (n is the number of input sequences) rounds of communications among all the processes.

In the following experiment, we show how pNJTree improves the parallel ClustalW program on a large taxa set of 6500 sequences. The program is named ClustalW–pNJTree as we use pNJTree to parallelize the second phase of ClustalW. As the speedup achieved by Ebedes and Datta is quite similar to those reported by Li, we only compared with the ClustalW-MPI code from Li presented in Table 1.

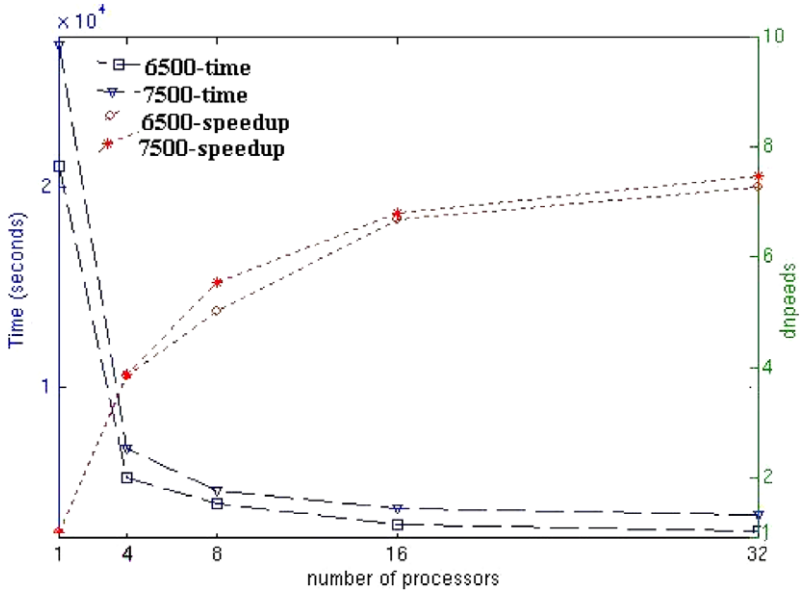

Fig. 2. Elapsed time and speedup on different numbers of processors with a 6500 and 7500 taxa sets.

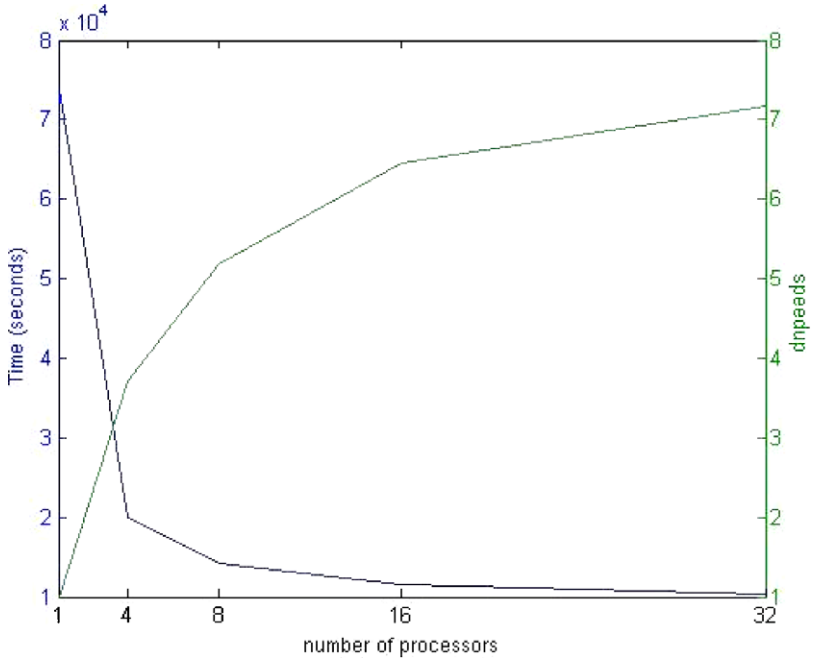

Fig. 3. Elapsed time and speedup on different numbers of processors with a 10 k taxa sets.

Table 1
Comparison of runtimes (in seconds) of ClustalW-MPI to our ClustalW–pNJTree on 4, 8, 16 and 32 processors

Methods	Num of processors	4	8	16	32
ClustalW-MPI	Overall	82845	63391	57119	50392
	Pairalign	35043	17224	10671	3788
	NJtree	**26339**	**24591**	**25278**	**25418**
	Malign	21463	21576	21170	21186
ClustalW–pNJTree	Overall	61904	42293	34846	28401
	Pairalign	34844	16525	10253	3685
	NJtree	**5474**	**4313**	**3141**	**2888**
	Malign	21586	21455	21452	21468

We compared the running time of three key steps between ClustalW-MPI and ClustalW–PNJTree. From the table, we can see that the ClustalW-MPI program does not make any speedup at the phase of building NJtree on multiple processors. While ClustalW–pNJTree make a near-linear speedup at this step. The results show that ClustalW–pNJTree is superior to ClustalW-MPI in terms of parallel performance. For example, on 32 processors, the ClustalW-MPI takes more than 7 h to build NJ tree at the second phase, while the ClustalW–pNJTree only need less than 1 h. It shows that our parallel algorithm improves the parallel ClustalW program better than previous programs. The proposed algorithm is targeted for workstation clusters with distributed memory architecture with large network bandwidth and low message latency. It also provides a fast and practicable approach for parallel reconstructing NJ tree for thousands of taxa.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China under Grant No. 60572100, foundation of State Key Laboratory of Networking and Switching Technology (CHINA) and Science Foundation of Shenzhen City under Grant No. 200408.

References

[1] J. Ebedes, A. Datta, Multiple sequence alignment in parallel on a workstation cluster, Bioinformatics. 20 (7) (2004) 1193–1195.
[2] K.-B. Li Clustal, W-MPI ClustalW analysis using distributed and parallel computing, Bioinformatics. 19 (2003) 1585–1586.
[3] N. Saitou, M. Nei, The neighbor joining method a newmethod for reconstructing phylogenetic trees, Mol. Biol. Evol. (1987) 406–425.
[4] J.A. Studier, K.J. Keppler, A note on the neighbor joining method of Saitou and Nei, Mol. Biol. Evol. 5 (1988) 729–731.
[5] J.D. Thompson, D.G. Higgins, T.J. Gibson, Clustal W: improving the sensitivity of progressive multiple alignment through sequence weighting position specific gap penalties and weight matrix choice, Nucl. Acids Res. 22 (1994) 4673–4690.
[6] M. Flynn, Some computer organizations and their effectiveness, IEEE Trans. Comput. C-21 94 (1972).
[7] MPI-Forum. Mpi standard 2.0. MPI.