FINE GRAIN PARALLEL CONSTRUCTION OF NEIGHBOUR-JOINING PHYLOGENETIC TREES WITH REDUCED REDUNDANCY USING MULTITHREADING

Biswajit Sahoo[1], Ashutosh Behura[2], and Sudarsan Padhy[3]

[1]KIIT University, Department of CSE, Bhubaneswar, India
sh_biswajit@yahoo.co.in
[2]KIIT University, Department of CSE, Bhubaneswar, India
ashutoshbehura_2006@yahoo.com
[3]Utkal University, Department of Mathematics, Bhubaneswar, India
spadhy04@yahoo.co.in

ABSTRACT

In biological research, scientists often need to use the information of the species to infer the evolutionary relationship among them. The evolutionary relationships are generally represented by a labeled binary tree, called the evolutionary tree (or phylogenetic tree). The phylogeny problem is computationally intensive, and thus it is suitable for parallel computing environment. In this paper, a fast algorithm for constructing Neighbor-Joining phylogenetic trees has been developed. The CPU time is drastically reduced as compared with sequential algorithms. The new algorithm includes three techniques: Firstly, a linear array A[N] is introduced to store the sum of every row of the distance matrix (the same as SK), which can eliminate many repeated (redundancy) computations, and the value of A[i] are computed only once at the beginning of the algorithm, and are updated by three elements in the iteration. Secondly, a very compact formula for the sum of all the branch lengths of OTUs (Operational Taxonomic Units) i and j has been designed. Thirdly, multiple parallel threads are used for computation of nearest neighboring pair.

INDEX TERM

Pair wise alignment, operational taxonomic unit, phylogenetic tree, multiple sequence alignment.

1. INTRODUCTION

An understanding of evolutionary relationships is at the heart of modern pharmaceutical research for drug discovery, and is also the basis for the design of genetically enhanced organisms. Evolutionary history is typically represented by an evolutionary tree [13-15]. An evolutionary tree is a leaf-labeled binary tree which tracks the genetic similarities of a set of closely related species.

The task of constructing the evolutionary tree for a set of species is known as the phylogeny problem. The difficulty of such problem is that the number of possible evolutionary trees is very large. Construction of an evolutionary history for a set of contemporary taxa based on their pairwise distance is computationally intractable (i.e., NP-complete) for various optimality

criteria [2, 3]. Existing methods for solving the phylogeny problem include parsimony, maximum likelihood, and distance matrix methods [15-17].Various heuristics have been proposed to search for solutions of desired quality, and the majority of these methods are greedy. Among the greedy approaches, the Neighbor-Joining (NJ) method [4, 5] is widely used by molecular biologists due to its efficiency and simplicity.

In recent years, NJ method is very popular especially in multiple sequence alignment (MSA) because it is applicable to any type of evolutionary distance data. The output of NJ method is used to direct the grouping of sequences during the multiple alignment process [8]. The most popular MSA tool CLUSTALW [7] uses NJ algorithm for constructing phylogenetic trees and the progressive alignment. Most recent famous MSA software such as T-COFFEE [8], MAFFT [9], MUSCLE [10] *et al* also use NJ method as main alternative option. But the big-O time complexity of the original NJ algorithm is $O(N^5)$, where N is the number of OTUs (Operational Taxonomic Units), which limits the application of NJ algorithm when N is very big. For a part of this reason, MUSCLE and MAFFT use UPGMA [11] to construct phylogenetic tree since UPGMA reduces the time complexity to $O(N^3)$. However, UPGMA does not build the true evolutionary tree to guide a progressive alignment in line with biological expectations though it may get higher SP score than NJ method sometimes. If the time complexity of NJ method is reduced, we believe that it will be more welcome by molecular biologists.

Studier and Kepler [5] have succeeded to reduce the time complexity of NJ from $O(N^5)$ to $O(N^3)$. But there is still a scope to reduce the execution time for NJ algorithm. In this paper, we propose a parallel algorithm which is verified to be much faster than the original one, and about 2, 2.2, and 2.3 times faster than Studier and Kepler's algorithm on a single thread, dual thread, and quad thread respectively by implementing with sufficient experimental data. We also reduce the time complexity from $O(N^5)$ to $O(N^3)$ by an extra $O(N)$ space in comparison to original NJ algorithm.

In the rest of this paper, sections 2 and 3 describe the sequential NJ algorithm and its time complexity respectively. In section 4, we propose a fine grain parallel fast algorithm. Experimental results are given in section 5. Section 6 contains the conclusion of the paper.

2. SEQUENTIAL NJ ALGORITHM

The NJ method was initially proposed by Saitou and Nei [4], and later modified by Studier and Kepler [5]. Neighbor-Joining seeks to build a tree which minimizes the sum of all branch lengths, i.e., it adopts the minimum-evolution (ME) criterion. Many studies have corroborated NJ's performance in reconstructing correct evolutionary trees. For small numbers of taxa, NJ solutions are likely to be identical to the optimal ME tree [6]. Neighbor-Joining begins with a star tree, then iteratively finds the nearest neighboring pair (i.e. the pair that induces a tree of minimum sum of branch lengths) among all possible pairs of nodes (both internal and external). The nearest pair is then clustered into a new internal node, and the distances of this node to the rest of the nodes in the tree are computed and used in later iterations. The algorithm terminates when $N-2$ internal nodes have been inserted into the tree (i.e., when the star tree is fully resolved into a binary tree). The framework of NJ is defined by three components: the criterion used to select pairs of nodes; the formula used to reduce the distance matrix; and the branch length estimation formula.

Fig.1 shows an example of NJ tree, which indicates the evolutionary relationships of 8 OTUs and which is an un-rooted tree. The rectangles denote the terminal nodes or OTUs and the rectangles denote the internal node. The characters beside the triangle denote OTUs, numbers and the rectangle denote the orders that OTUs are inserted into the tree and the data above the lines are the branch lengths of two neighbors. It is obvious that OTUs 1 and 2 are clustered first, generating a new combined OTU. The branch lengths between 1 and the new OTU and 2 and the new OTU are 5 and 2 respectively. After OTUs 7 and 8 are clustered, there are only two OTUs, and they are clustered. The tree is built now.

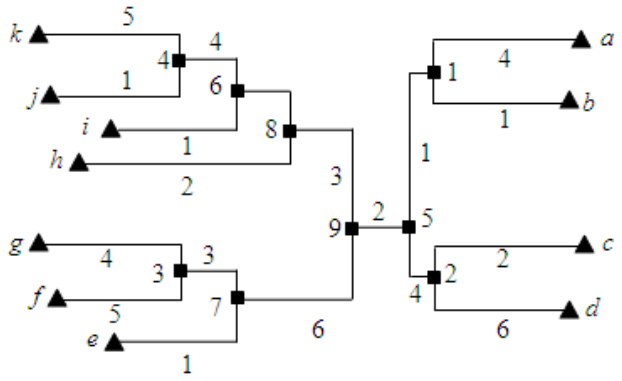

Figure 1. Neighbor-Joining tree

The NJ algorithm is described briefly as follows. S_{ij} is the sum of the least-squares estimates of branch lengths [4] and is defined by

$$S_{ij} = \frac{1}{2(N-2)} \sum_{k=0}^{N-1} (D_{ik} + D_{jk}) + \frac{1}{2} D_{ij} + \frac{1}{N-2} \sum_{m<n} D_{mn} \qquad (1)$$

where $i=0 \sim N-2$, $j=i+1 \sim N-1$, $k,m,n \neq i, j$.

At every iterative step, the minimum S_{ij} indicates that nodes i and j are the nearest evolutionary neighbors. Suppose that OTUs i and j are the neighbors to be joined in the tree. They are clustered as a new one, denoted by X. Define the symbol D_{ij} as the edit distance between OTUs i and j.

The distance between this combined OTU (X) and another OTU k is defined by

$$D_{Xk} = (D_{ik} + D_{jk})/2, \; k = 0 \sim N-1, \; k \neq i,j \qquad (2)$$

L_{ab} is defined as the branch length between nodes a and b. L_{iX} and L_{jX} are estimated by

$$L_{iX} = (D_{ij} + D_{iZ} - D_{jZ})/2 \qquad (3)$$
$$L_{jX} = (D_{ij} + D_{jZ} - D_{iZ})/2$$

where

$$D_{iZ} = \sum_{k=0}^{N-1} D_{ik} / (N-2), D_{jZ}$$
$$= \sum_{k=0}^{N-1} D_{jk} / (N-2), k \neq i, j.$$

Here, Z represents a group of OTUs including all but *i* and j, and D_{iz} and D_{jz} are the distances between *i* and Z and *j* and Z, respectively. At last OTUs *i* and *j* are deleted from the current OTUs lists and *X* is added to the list, then the number of current OTUs reduces by one and a new iteration starts. The pseudo code of NJ algorithm is described as follows:

```
/*Input: N (the number of OTUs) and D_ij (the edit distance between pair OTUs i and j),
Distance Matrix, D */
/* Output:    NJ tree */

1.   While N > 2 do
1.1 For i=0 to N-1 do
1.2        For j=i+1 to N do
1.2.1     Compute S_ij according to formula (l).
           EndFor
     EndFor
1.3 Search the minimum S_ij to get neighbors i and j.
1.4 OTUs i and j are clustered as a new OTU X.
1.5 Compute D_Xk according to formula (2).
1.6 Compute branch lengths L_iX and L_jX according to formula (3).
1.7 Delete OTUs i and j, and add X to current OTU lists.
1.8 N=N-1.
1.9 End while
2 Cluster the last two OTUs.
```

3. TIME COMPLEXITY ANALYSIS OF NJ ALGORITHM

According to the pseudo code of NJ algorithm, the time complexity of the outermost loop is O*(N),* lines 1.1 and 1.2 are O*(N^2),* line 1.3 is O*(N^2),* line 1.4 is O(1), line 1.5 is O*(N),* line 1.6 is O*(N),* line 1.7 is O(1) and line 2 is O*(N).* As for line 1.2.1, because formula (1) includes three parts, we will analyze the time complexity of which separately. The first part contributes the time O*(N),* the second is O(1) and the third is O*(N^2)* because we should consider all the *m* and *n* but *i* and *j.* So the most expensive step is formula (1), which induces that the total time of NJ is O*(N^5)* if we compute it directly. So the key point to optimize NJ algorithm is to optimize the computation of formula (1).

For the three parts of formula (1), in fact there are many repeated computations of the same data in part 1 and 3. For example, in part 1, when computing S_{12}, D_{13} is added; when computing S_{14}, D_{13} is added too. In part 3, when computing S_{12}, D_{34} is added; when computing S_{15}, D_{34} is added too. In other words, there are close relations between the computations of S_{ij}. In other words, NJ

algorithm should compute the triangular matrix (S_{ij}) iteratively to search the nearest neighbors. But it is fortunate that the computations of S_{ij} only need the distance matrix (D_{ij}), which motivates us to design fast parallel algorithm to avoid the repeated computations of D_{ij}.

4. PARALLEL FAST ALGORITHM FOR CONSTRUCTING NJ TREE

Our motivation is very simple, we try to reduce the number of computations in the inner most loop (1.2.1), which is the most expensive of the computations for the time complexity of the whole algorithm. Pre-processing outside the double loops of lines 1.1 to 1.2 of the pseudo code has been done, the results of which are used for the computations of S_{ij} (line 1.2.1).

$$A_i = \sum_{j=0}^{N-1} D_{ij} (j \neq i) \tag{4}$$

$$total = \sum_{i=0}^{N-1} A_i \tag{5}$$

Define a linear array *A[N]* and a buffer *total* the former is used to store the sum of every row of D_{ij} and the later to store all the sum of D_{ij} .Note that (D_{ij}) is a symmetric square matrix but not a triangular matrix as (S_{ij}). Then the first and third part of S_{ij} *sum1* and *sum3* can be written respectively as

$$sum1 \overset{def}{=} \left(A_i + A_j - 2D_{ij} \right) / 2\left(N \ -2 \right) \tag{6}$$

$$sum3 \overset{def}{=} (total - 2(A_i + A_j) + 2D_{ij}) \ /2(N-2) \tag{7}$$

and

$$S_{ij} = sum1 + \ \frac{1}{2} D_{ij} + sum3 \tag{8}$$

$$= (total - \ A_i - A_j) / 2(N - \ 2) \ + \ \frac{1}{2} D_{ij}$$

Compared with formula (1), this modified formula (8) is simpler and efficient. To compute *Ai* and *total,* the time complexity is O*(N^2)* and O*(N)* respectively. And the time complexity of formula (8) is only O(1).

Proof of equations (6), (7), and (8)

$$A_i - D_{ij} = \sum_{k=0}^{N-1} D_{ik} (k \neq i) - D_{ij} = \sum_{k=0}^{N-1} D_{ik} (k \neq i, j)$$

$$A_j - D_{ij} = \sum_{k=0}^{N-1} D_{jk} (k \neq j) - D_{ij} = \sum_{k=0}^{N-1} D_{jk} (k \neq i, j)$$

$$(A_i - D_{ij}) + (A_j - D_{ij}) = A_i + A_j - 2D_{ij}$$

$$= \sum_{k=0}^{N-1} (D_{ik} + D_{jk})(k \neq i, j)$$

(2) For *sum3,*

$(A_i + A_j)$ denotes the sums of rows and cols of i and j respectively. $(total - 2\ (A_i + A_j\ ) + 2D_{ij})$ denotes the sums of the remaining elements of (D_{ij}) except those elements related to i and j. $2D_{ij}$ is added because D_{ij} and D_{ji} are deleted twice from rows and cols of i and j. Then

$$total - 2(A_i + A_j) + 2D_{ij} = \sum_{m<n} D_{mn}(m, n \neq i, j).$$

Consider formula (8) again, *total* and N are constants for every S_{ij} at each iteration, so they need not be computed to search the minimum S_{ij}, and the constant factor 1/2 and $1/(N-2)$ are ignored too, then we can get a more simple form of S_{ij} (formula (9)) compared with formula (8). We can search minimum S_{ij} using S'_{ij} where

$$S'_{ij} = D_{ij}(N-2) - (A_i + A_j) \qquad (9)$$

Formula (9) does not include *total,* which implies that we need not compute the sum of A_i any more as formula (5) iteratively. This saves O*(N)* time computation.

Consider the distance matrix (D_{ij}), suppose that OTUs i^* and j^* are the neighbors to be joined in the tree, then the two rows and cols of i^* and j^* are deleted from D_{ij} , and a new row and column (denoted by *X)* are added. Then D_{Xk} is computed according to formula (2). In fact, we need not compute A_i again as formula (4), because most values of the remaining A_i are unchanged between every two successive iteration step. A_x can be computed in O*(N)* time. For the remaining $A_{i'}$ only D_{ii^*} and D_{ij^*} are deleted, and D_{iX} is added. So

$$A_i = A_i - D_{it^*} - D_{ij^*} + D_{iX} \qquad (10)$$

Which saves *O(N)* time for each iteration.
And the formulas for D_{iZ} and D_{jZ} in formula (3) are computed as

$$D_{iZ} = (A_i - D_{ij})/(N - 2) \qquad (11)$$
$$D_{jZ} = (A_j - D_{ij})/(N - 2)$$

Which saves O*(N)* time compared with formula (3).

Computing S_{ij} for all pair of protein/DNA sequences have been parallelized. As formula (9) for S_{ij} is a simple one, fine grain parallelism is suitable. The elements of the upper triangular matrix (S_{ij}) are mapped onto a one dimensional array for equal load distribution among the threads. The indexes i and j of the matrix (S_{ij}) can be mapped onto the index r of the one dimensional array where

$$r = (i - 1)(2n - i) / 2 + j - i$$

Step 1.2 of our algorithm uses master-slave parallelization method to compute minimum S_{ij}. In each iteration, master thread creates multiple slave threads. Each thread calculates minimum S_{ij} independently. Then master thread waits till all slaves complete finding their respective local minimum. The number of S_{ij}s' computed by each thread (both master and slaves) are evenly distributed among them. Finally, master declares the global minimum S_{ij} and continues computation of D_{kx}, L_{ix}, L_{jx}, A_i and A_x. The master thread repeats the above steps till *N–2* number of internal nodes are inserted.

Now the new algorithm is described as follows: Compute A_i according to formula (4).

```
1 While N > 2 do
1.0  For i=1 to N – 1 do
1.1    For j=i+1 to N   do
          r = (i – 1) (2n – i) / 2 + j – i
           /*map the indexes i and j of upper triangular matrix (Sij) to a linear array index r*/
        EndFor;
     EndFor;
     /* Step 1.2: Find minimum Sij by parallel threads (both master and slave) */
1.2  For i=1 to (N × (N – 1)) / (2p) do parallel
          /* p is the number of processes */
           Compute Sij according to formula (9).
          Find local minimum Sij
        EndFor;
1.3    Search the global minimum Sij from all local minimum to get neighbors i* and j*
        by the master thread
        /* Step 2: Update Dkx, Lix, Ljx, Ai and Ax by master thread*/
1.4       OTUs i* and j* are clustered as a new OTU X
1.5       Compute DXk according to formula (2).
1.6       Compute branch lengths LiX and LjX according to formulas (3, 11).
1.7       Delete OTUs i* and j*, and add X to current OTU lists.
1.8        Update the remaining Ai according to formula (10) and compute Ax.
1.9       N=N-1.
     EndWhile
2 Cluster the last two OTUs.
```

The time complexity of the new algorithm is $O(N^3)$.

The implementation of multithreaded fine grain parallel algorithm can combine with either OpenMP, Pthreads or thread libraries. We implemented the fine grain parallel portion using Pthread. In our parallel algorithm, a scheduling strategy called *fixed-size chunking* [18] where tasks of one fixed size are to be allocated to available cores.

5. EXPERIMENT RESULTS:

We implement our algorithm in C programming language and utilize the Pthread library of Linux Operating System. The experiment was conducted in a computer system having 2.0 GHz Quad core Xeon CPU with 16 GB of primary memory and 12 MB cache memory. This computer runs Linux OS with no other applications installed. All measured times include the times that

were taken to read the input data from a file and write the solution into a file. Furthermore, all measured times were measured when there is no one using the system except this experiment.

The distance matrix (D_{ij}) is generated by three methods: (1) Global pairwise alignments by dynamic programming, which is slowest but the most accurate. (2) K-mer distance [10, 12], which is linear time complexity for pairwise distance estimation. (3) Random generation by computer pseudo number, which is just to provide a fast method to generate data.
Experiments measured the elapsed times for SK and our proposed single (1T), dual (2T), and quad (4T) thread implementation of NJ algorithm as a function of the number of sequences as shown in the Table 1. Number of sequences varies from 300 to 900 in our experience. All measured times are in millisecond.

Table 1
Elapsed times of our NJ phylogenetic trees construction algorithm running in single(1T) dual(2T), quad(4T) threads and SK algorithm implementation with respect to no. of sequences. Times are in millisecond.

	300	**400**	**500**	**600**	**700**	**800**	**900**
SK	202	368	710	1190	1805	2590	3610
IT	130	230	390	620	930	1330	1840
2T	120	205	350	550	820	1150	1610
4T	110	170	330	510	770	1105	1535

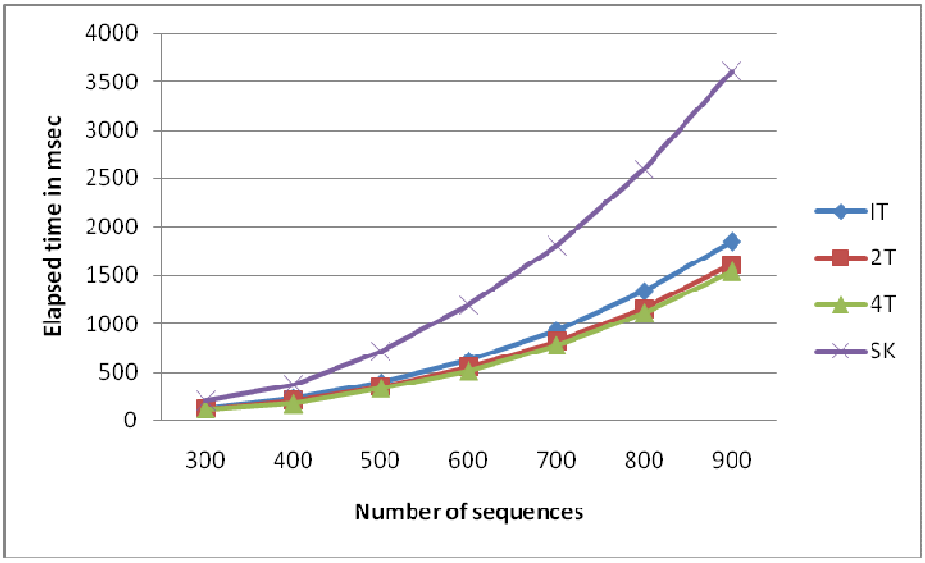

Figure 2. Elapsed time for SK sequential(SK), our single(1T), dual(2T), and quad(4T) threads implementation of Neighbor Joining algorithm as a function of number of sequences.

Figure 2 shows the elapsed times for the sequential SK and our threaded (single, dual and quad) Neighbor joining phylogenetic tree algorithm as a function of number of sequences. All those threaded algorithms successfully reduce the execution time.

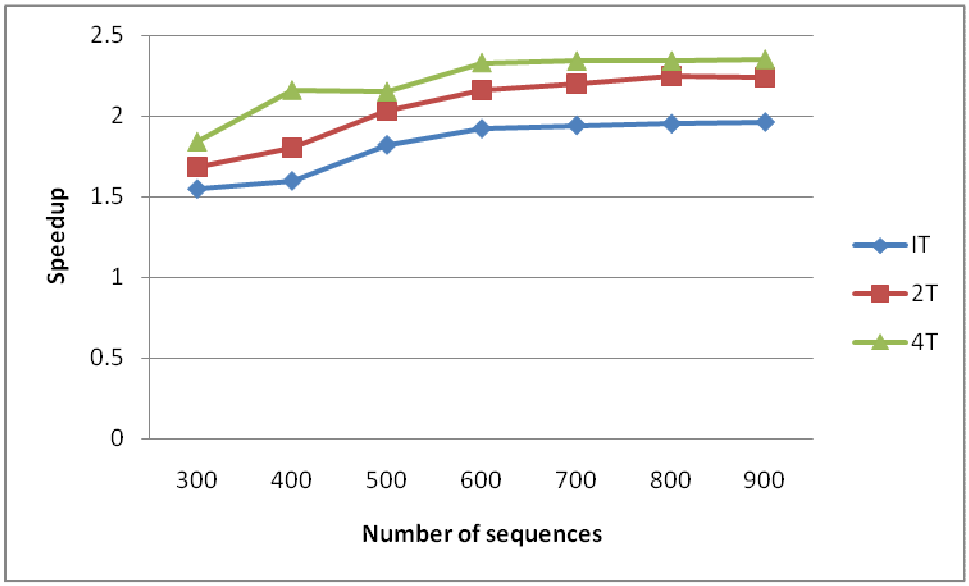

Figure 3. Speedup of our Neighbor Joining algorithms running in single(1T), dual(2T), and quad(4T) threads comparison to SK as a function of number of sequences.

Figure 3 shows the speedup for the single, dual, and quad thread execution of the proposed algorithm as a function of number of sequences as compared to SK algorithm. It can be observed that higher speedup can be achieved with more number of sequences. This happens due to the overhead of thread creation, which is fixed for any number of sequences.

Now, we focus on the speedup as a function of number of threads. In Figure 4, the speedup increases from single thread to dual threads and from dual threads to quad threads. However, speedup may fall if both the numbers of sequences and number of thread do not increase proportionately. This is because of the overhead incurred in creating threads. The speedup seems to be lower than steady state when executed using large number of threads.

The proposed method fully utilizes the multithreading feature along with reducing redundant computation in comparison to sequential SK algorithm. This achieves better speedup especially when the number of sequences is large. This is compatible with the massive work for the tree construction and gains more throughput.

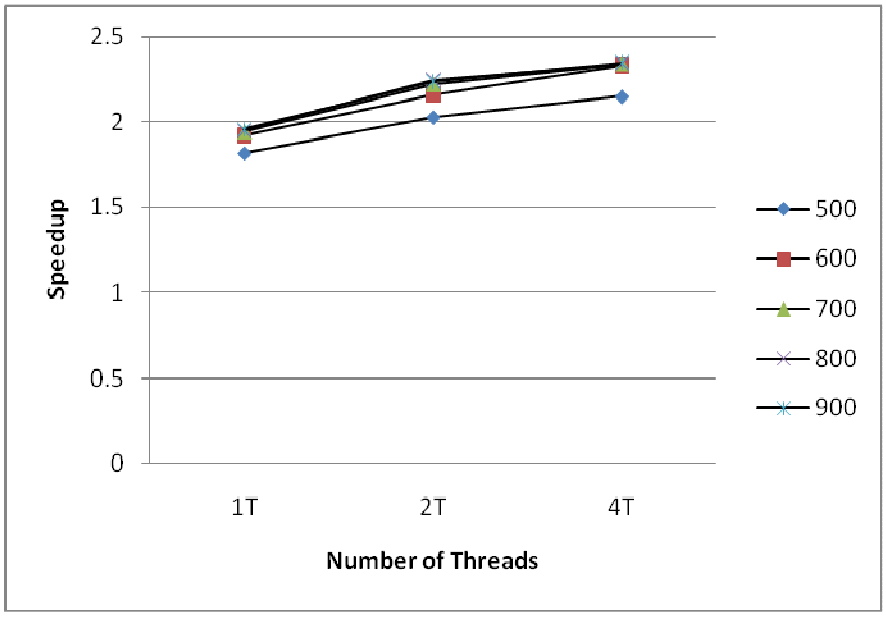

Figure 4. Speedup of our Neighbor Joining results for different number of sequences in comparison to SK as a function of number of threads.

6. CONCLUSIONS

In this paper, a fast parallel algorithm for constructing Neighbor – Joining phylogenetic trees has been developed. The new algorithm has $O(N^3)$ time complexity compared with $O(N^5)$ of SN, the same as SK. Further a fine grain parallelism is incorporated using multiple threads in the inner most loop. The experimental results show that running time of our algorithm using a single thread is from tens to hundreds times faster than SN and about two times faster than SK for large N. It is also verified that Speedup of our algorithm using dual threads is 2.2 times faster and that using quad threads is 2.3 times faster than SK algorithm on a desktop computer. The key point of our algorithm is to reduce the repeated computations and to parallelize the computation. We tried the above computation in the inner most loop.

REFERENCES

[1] C. Yang and S. Khuri, "PTC: An interactive tool for phylogenetic tree construction," in IEEE Proc. of the *Computational Systems Bioinformatics*, Stanford University, 2003,8, pp. 476-477.

[2] L. R. Foulds and R. L. Graham, "The steiner problem in phylogeny is NP-complete," *Advances Appl. Math.* vol. 3, 1982, pp. 43-49.

[3] W. Day, "Computational complexity of inferring phylogenies from dissimilarity matrices," *Bull. Math. BioI.,* voI.49,no.4, 1987, pp.461-467.

[4] N. Saitou, and M. Nei, "The n.eighbor-joining method: a new method for reconstructing phylogenetic trees," *Mol. BioI. Evol.,* vol. 4, no. 4, 1987, pp. 406-425.

[5] J. Studier and K. Keppler, "A note on the neighbor-joining algorithm of Saitou and Nei," *Mol. BioI. Evol.,* vol. 5, no. 6, 1988, pp. 729-731.

[6] N. Saitou and T. Imanishi, "Relative efficiencies of the Fitch-Margoliash, maximum-parsimony, Maximum likelihood, minimum-evolution, and neighbor-joining methods of phylogenetic tree construction in obtaining the correct tree," *Mol. BioI. Evol.,* vol. 6, no. 5, 1989, pp. 514-525.

[7] J.D. Thompson, D.G. Higgins, and T. J. Gibson, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Res.,* vol. 22, no. 22, 1994, pp. 4673-4680.

[8] C. Notredame, D.G. Higgins, and J. Heringa, "T-Coffee: a novel method for fast and accurate multiple sequence alignment," *J. Mol. BioI.,* vol. 302, 2000, pp. 205-217.

[9] K. Katoh, K. Misawa, K. Kuma and T. Miyata, "MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform," *Nucleic Acids Res.,* vol. 30, no. 14, 2002, pp.3059-3066.

[10] R.C. Edgar, "MUSCLE: multiple sequence alignment with high accuracy and high throughput," *Nucleic Acids Res.,* vol. 32, no. 5, Mar. 2004, pp. 1792-1797.

[11] P.H.A. Sneath and R.R. Sokal, Numerical Taxonomy. Freeman, San Francisco, 1973.

[12] R.C. Edgar, "Local homology recognition-, Jan. and distance measures in linear time using compressed amino acid alphabets," *Nucleic Acids Res*., vol. 32, no. 1, Jan. 2004, pp.380-385.

[13] H. O. Andrzej Lingas and A. Ostlin. Efficient merging and construction of evolutionary trees. *Journal of Algorithms*, volume 41, 2001, pages 41–51.

[14] M. Csuros. Fast recovery of evolutionary trees with thousands of nodes. *In the proceeding of the International Conference on Research in Computational Molecular Biology*, number 01, page 2001, 1997.

[15] R. C. T. Lee. Computational biology. *Department of Computer Science and Information Engineering, National Chi-Nan University*, 2001.

[16] L. A. Salter. Algorithms for phylogenetic tree reconstruction. In *Proceeding of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, volume 2, 2000, pages 459–465.

[17] K. Strimmer and A. von Haeseler. Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies. In *Molecular Biology and Evolution*, volume 13(7), 1996, pages 964–969.

[18] T. Hagerup. Allocating independent tasks to parallel processors: an experimental strudy. *Journal of Parallel and Distributed Computing,* 1997, Vol. 47, pp. 185-197.

Authors :

Biswajit Sahoo received Bachelor Degree in Electronics & Communication Engineering from NIT Durgapur, India, M.Tech Degree in Computer Engineering from B.E College, Shibpur & pursuing Ph.D. in Computer Science under Utkal University, India. He is presently working as Associate Professor Department of Computer Science & Engineering, KIIT University, Bhubaneswar, India. He has published papers in the areas relating to Parallel Algorithms and Bio Informatics.

Ashutosh Behura received Bachelor degree in Computer Science and Engineering from KIIT University, Bhubaneswar, India. He is presently working as Research Associate in the Department of Computer Science & Engineering, KIIT University, Bhubaneswar, India. His area of interest is Parallel Algorithms and Bio Informatics.

Sudarsan Padhy is a Professor of Mathematics at Utkal University, India. He obtained his Ph.D. degree in Mathematics from Utkal University in 1979 and Postdoctoral research at University of Freiburg, Germany during 1980-81. He has over fifty published research papers and five books to his credit extending over Fluid dynamics, Finite difference and Finite element method for solving partial differential equations, Operation research, Parallel algorithms, Computational finance and Computational biology.