Parallel Implementation of a Quartet-Based Algorithm for Phylogenetic Analysis

B. B. Zhou[1], D. Chu[1], M. Tarawneh[1], P. Wang[1], C. Wang[1],
A. Y. Zomaya[1], and R. P. Brent[2]

[1]School of Information Technologies
University of Sydney
NSW 2006, Australia
bbz@it.usyd.edu.au

[2]Mathematical Science Institute
Australian National University
Canberra, ACT 0200, Australia
rpb@rpbrent.co.uk

Abstract

This paper describes a parallel implementation of our recently developed algorithm for phylogenetic analysis on the IBM BlueGene/L cluster [15]. This algorithm constructs evolutionary trees for a given set of DNA or protein sequences based on the topological information of every possible quartet trees. Our experimental results showed that it has several advantages over many popular algorithms. By distributing the quartet weights evenly across the processing nodes and making effective use of a fast collective network on the IBM BlueGene/L cluster, we are able to achieve a close to linear speedup even when the number of processors involved in the computation is large.

1. Introduction

The quartet-based method is one of the very important approaches for reconstruction of a large evolutionary tree from a set of smaller trees. It constructs a tree for a given number of molecular sequences based on the topological properties of each subset of four molecular sequences. The main advantage of this method is that there is a one-to-one correspondence between a tree topology and a set of four-sequence or quartet trees. If we can correctly identify the tree topology of each individual subset of four sequences, we are able to reconstruct the entire evolutionary tree for a given problem in polynomial time. In practice, however, there exist situations that the correctly resolved quartet trees are very difficult to obtain by using currently existing methods [1,18]. Therefore, the main concern in designing a good and practical quartet-based algorithm is how to tolerate errors in the quartet trees when reconstructing the entire tree topology. Different methods have been introduced in the literature to deal with the problem of quartet errors, for example, those in [3,4,5,6,7, 8,9,10,11,13,14,16,19,20,22].

Recently, we developed a new quartet-based algorithm for reconstruction of evolutionary trees [25]. In this algorithm trees are recursively constructed using a quartet weight matrix which contains collective topological information from all possible quartets. Our experimental results show that this algorithm outperforms many existing methods for phylogenetic analysis.

One major disadvantage associated with most quartet-based algorithms is that they require $O(n^4)$ computational steps to complete where n is a given number of molecular sequences in the analysis. This is simply because they need to generate $O(n^4)$ quartet trees in order to obtain a reasonably good result. When the problem size n is large, we also need a large-size memory to store these trees during the computation. For example, a well-know parallel program package for a quartet-based algorithm, TreePuzzle [20], is only able to handle problems of size smaller than or equal to 250 regardless of the number of processors involved in the computation. The current trend is to design fast algorithms for phylogenetic analysis, e.g. those described in [12,24]. However, it should be noted that most of these algorithms use NP-hard reconstruction criteria (mostly maximum likelihood). Theoretical studies [21] and our recent experimental results [26] show that even if we are able to find a truly globally optimal tree under the maximum likelihood criterion, this tree may not necessarily be the correct phylogenetic tree! Because of its excellent theoretical properties, the quartet-based method should never be overlooked. High-performance computing machines can be adopted to handle higher computational and memory requirements for quartet-based algorithms. In this paper we show that, by evenly distributing the quartets across the processing nodes and making effective use of a fast collective network on the IBM BlueGene/L cluster

1-4244-0054-6/06/$20.00

[15], our quartet-based algorithm is able to achieve a close to linear speedup even when the number of processors involved in the computation is large.

The paper is organized as follows: our quartet-based algorithm is briefly described in Section 2. Its parallel implementation is discussed in Section 3. In Section 4 we present some experimental results obtained on a 128 node (256 CPUs) IBM BlueGene/L cluster. Conclusions are given in Section 5.

2. The Sequential Algorithm

In this section we briefly describe our quartet-based algorithm. A more detailed description can be found in [25,26].

Our quartet-based algorithm for phylogenetic analysis consists of two major stages. In stage one we calculate quartet weights for every possible quartet trees from a given number of sequences. In stage two we first generate a global quartet weight matrix to gather the quartet topological information from the quartet weights calculated in stage one and then reconstruct a full size tree using this quartet weight matrix. We can use any existing method for phylogenetic analysis to calculate the weights of quartet trees [17]. In the following we only discuss the computations in stage two. We first discuss how a quartet weight matrix is generated from a set of quartets defined by a given tree topology and give an efficient algorithm for reconstruction of the tree topology from the generated quartet weight matrix. This one-to-one mapping between a given tree topology and its associated quartet weight matrix forms the basis of our quartet-based algorithm for phylogenetic analysis. We next discuss the tree reconstruction algorithm and show how to deal with the problem of inaccurate quartet weight matrices.

2.1. Basic concept

A quartet, or a set of four sequences is associated with three possible fully resolved trees, as shown in Figure 1. In the figure $(xy \mid zt)$ indicates how the sequences, represented by leaf nodes, are divided into two pairs by cutting the middle edge (so-called bi-partitioning), and thus shows the neighbourhood relations of the quartet in terms of topology. One way to measure which of the three possible trees is more likely to be the true tree is to use Bayes weights [17], or quartet weights in this paper. The quartet weights for three possible trees of a quartet is obtained by first calculating the likelihood value for each tree and then transforming these likelihood values into posterior probabilities, or quartet weights w_i for i = 1, 2, and 3, by applying Bayes' theorem assuming a uniform prior for all three possible trees.

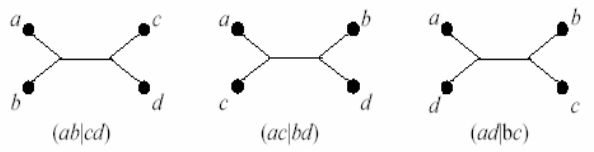

Figure 1. Three possible fully resolved trees for a quartet {a, b, c, d}.

Let $(ij \mid kl, w)$ denote a possible quartet tree with a quartet weight w. Our global quartet weight matrix is generated by adding each w to entries ij, ji, kl and lk, using a complete set of quartets from a given number of sequences. This matrix is symmetric and its size is $n \times n$, where n is the total number of sequences and each row or column corresponds to one particular sequence. (Note this quartet weight matrix is called score matrix in [11] and it was generated using discrete weights (or scores) from a distance matrix.)

Given a tree topology of n leaves, we can uniquely determine a set of $\binom{n}{4}$ quartet trees which are consistent with the original tree, i.e., each quartet tree separates the four leaves into two pairs in the same way as the original tree through bi-partitioning. For each quartet there can only be one fully resolved tree in the set of these quartet trees and it is described as $(ab \mid cd, 1.0)$. For a given tree topology a global quartet weight matrix is also uniquely determined. This is because our matrix is generated using the quartet weights of the associated quartet trees.

We can also reconstruct the tree topology from its generated quartet matrix by using an efficient $O(n^2)$ algorithm, as shown in Figure 2. This algorithm is derived using the dynamic programming technique [25]. In the figure each row in quartet weight matrix corresponds to a particular leaf node and is associated with a variable m_i. One node in each sub-tree (or a row in the quartet weight matrix) is chosen as the representative node for the sub-tree which is also associated with a variable n_i. Initially every leaf node is considered as a sub-tree. Therefore, each row in the matrix will be associated with two variables which are set to $m_i = 0$ and $n_i = 1$.

Each pair of sub-trees is associated with a confidence value c_{ij}. To calculate the confidence value, we first assume that two sub-trees are connected together in the original tree and calculate the total number of quartets with

For each row i, set $n_i = 1$ and $m_i = 0$ (initialise n sub-trees)
For each row i, $(1 \le i < n)$
  Do (initialise g_i which stores the index j of a sub-tree to be merged with, or otherwise set the value to zero.)
  $g_i = 0$
  For each column j $(i < j \le n)$
    Do calculate c_{ij}. **If** $c_{ij} = 1$, **then** set $g_i = j$ (there is only one such c_{ij} for any two sub-trees)
While the number of sub-trees does not equal to one (main loop of the recursion)
  Do check g_i for a nonzero entry $g_i = j$
  Add an internal node to merge sub-trees i and j
  Update m_i for the representative node of sub-tree i: $m_i = m_i + \binom{n_j}{2}$
  Update m_j for the representative node of sub-tree j: $m_j = m_j + \binom{n_i}{2}$
  Choose the representative node of sub-tree i as the representative of the new sub-tree
  Update n_i: $n_i = n_i + n_j$
  $g_i = 0$
  For each other sub-tree j
    Do calculate c_{ij}. **If** $c_{ij} = 1$, **then** set $g_i = j$ (or $g_j = i$ if $j < i$)

Figure 2. A recursive algorithm for tree reconstruction from its generated quartet matrix.

a concerned form $(ab \mid cd, 1.0)$ where a is a leaf node from one sub-tree and b from the other, but c and d are leaf nodes not in either of these two sub-trees. The confidence value is then obtained by dividing the actual number accumulated directly from the quartet sets and stored in the matrix by this calculated value. If two sub-trees are truly connected together in the original tree, the corresponding confidence value for each pair of leaf nodes, one from each sub-tree must be equal to one. In the algorithm we use one additional variable g_i for representative node i to store index j when $c_{ij} = 1$. In each step we first try to find two sub-trees to merge by checking the variable g_i and then update the variables m_i and n_i in accordance with the merge; Next c_{ij}s are re-calculated and g_is updated for the next merge step. The process continues until all sequences are merged into a single tree.

2.2. Tree construction from inaccurate global weight matrices

In the previous subsection we discussed an algorithm for reconstructing the original tree from its generated global weight matrix. The same algorithm may be used to construct an evolutionary tree for a given set of n sequences if all the associated quartets are fully and correctly resolved. Unfortunately, this is only an ideal case and in reality it is very hard for us to have all the quartets fully and correctly resolved. Therefore, the global weight matrix generated from a set of quartet weights is inaccurate and the algorithm for tree topology reconstruction discussed above cannot be used without modification. To deal with inaccurate weight matrices we make three major changes to the original algorithm.

Average confidence value $\bar{c}_{ij}$: Since the entry values of the global weight matrix are no longer ideal, different node pairs, one from each of the two sub-trees, may produce different confidence values. A simple way to alleviate this problem is to calculate the confidence values for every leaf node pairs, to average them and then to use this averaged value as the confidence value $\bar{c}_{ij}$ for each pair of sub-trees.

Since the entry values of the weight matrix are inaccurate, we may not obtain $\bar{c}_{ij} = 1$ for a pair of sub-trees during the computation. In addition to the three variables, namely, g_i, m_i and n_i, associated with each sub-tree, we need a new variable $\bar{c}_i$ to record the highest average confidence value for sub-tree i with another sub-tree j for $i<j$. At each step we compare the stored values in $\bar{c}_i$ and choose to merge the two sub-trees which have the highest average confidence value.

Quartet weight correction: After two sub-trees are merged, we take an additional step to restore the associated entries in the matrix to their "true" values, i.e., change the quartet weights based on the currently reconstructed sub-trees and update the weight matrix accordingly. In particular, after each merge we need to correct the weights of all those quartets containing four nodes $\{i, j, p, q\}$ to $(ij|pq, 1.0)$ where i is a leaf node in one merged sub-tree, j is a leaf node in the other merged sub-tree and p and q the leaf nodes from the rest. If the weights are not corrected, the distributed errors may significantly affect the correct decision making in the following merge steps.

With the above two modifications we can have an algorithm which is able to deal with inaccurate quartet weights, as shown in Figure 3.

At each merge step the three major contributors to the total computational cost are: (1) the updating of m_i values, (2) calculation of average confidence values for each sub-tree to find the highest one, and (3) the quartet weight correction. The total costs for updating m_i values and for calculating average confidence values are $O(n^2)$ and $O(n^3)$, respectively. However, each quartet weight is corrected once and only once during the entire computation. The total number of quartets for a given set of n sequences is $\binom{n}{4}$ and obviously the total cost for quartet correction is $O(n^4)$. Therefore, the total

computational cost for this algorithm will be $O(n^4)$. It should also be noted that this cost is much less than the cost for computing quartet weights using the maximum likelihood which requires $O(sn^4)$ operations where s is the length of the sequences, usually a few hundreds to a few thousands.

For each row i, set $n_i = 1$ and $m_i = 0$ (initialise n sub-trees)
For each row i, $(1 \le i < n)$
  Do (initialise $\bar{c}_i$ and g_i for every sub-tree.)
    $g_i = 0$ and $\bar{c}_i = 0$
    For each column j $(i < j \le n)$
      Do calculate $\bar{c}_{ij}$. **If** $\bar{c}_{ij} > \bar{c}_i$, **then** set $\bar{c}_i = \bar{c}_{ij}$ and $g_i = j$ (store the largest $\bar{c}_{ij}$.)
While the number of sub-trees does not equal to one (main loop of the recursion)
  Do find the largest $\bar{c}_a$ and the associated $g_a = b$
    Add an internal node to merge sub-trees a and b
    (Update m_i values for every leaf node in sub-trees a and b.)
    For each row associated with a node i in a
      Do Update m_i: $m_i = m_i + \binom{n_b}{2}$
    For each row associated with a node j in b
      Do Update m_j: $m_j = m_j + \binom{n_a}{2}$
    Update n_o for the new sub-tree: $n_o = n_a + n_b$ (stored in the array entry associated with the representative node o of the sub-tree.)
    (Quartet weight correction.)
    Update the quartet weight matrix by correcting the weights of those quartets containing $\{i, j, p, q\}$ to $(ij \mid pq, 1.0)$, for i being a leaf node in sub-tree a, j a leaf node in b
    (Re-calculation of average confidence values.)
    For sub-tree i, $(1 < i < m$ for m the total number of sub-trees currently.)
      Do $g_i = 0$ and $\bar{c}_i = 0$
        For each other sub-tree j $(i < j < m)$
          Do calculate $\bar{c}_{ij}$. **If** $\bar{c}_{ij} > \bar{c}_i$, **then** set $\bar{c}_i = \bar{c}_{ij}$ and $g_i = j$

Figure 3. An algorithm for tree construction from inaccurate quartet weight matrices.

Multiple tree reconstruction: Since the matrix is not accurate, it may not always be the right decision to merge the two sub-trees that have the highest confidence value. After the highest confidence value $\bar{c}_{ij}$ is obtained, we then check whether there is another sub-tree k which has a reasonably high confidence value associated with one of the two sub-trees i and j, that is, we check whether $\bar{c}_{ik}$ (or $\bar{c}_{ki}$) $\ge \alpha \bar{c}_{ij}$, or $\bar{c}_{jk}$ (or $\bar{c}_{kj}$) $\ge \alpha \bar{c}_{ij}$ where α is a threshold which is smaller than, but close to one. (If there are several sub-trees which satisfy this condition, in our current version we simply choose the one with the highest confidence value among them.) At each of these critical points we can have three different super quartet trees with four sub-trees i, j, k, and the rest as its four super nodes at different places. The problem is which one will be the correct one leading us to find the correct tree topology. In the current version of our algorithm we keep all three different patterns. Therefore, we will reconstruct multiple trees and hope that the correct tree will be included in these generated trees. However, we need a control on the number of trees to be reconstructed. Otherwise, we may end up with about 3^n different trees if every merge step is a critical point. We use a parameter s to limit the total number of trees. Each time a critical point is encountered, two extra trees are generated until s such stages are encountered for each tree. Therefore, the maximum number of trees to be generated will be limited to 3^s.

We used the benchmarks consisting of 48,000 synthetic data sets of DNA sequences developed by the LIRMM Methods and Algorithms in Bioinformatics research group [23] to test our quartet-based algorithm. The results show that our algorithm performs much better than many existing methods [25,26], i.e., the probability for the correct tree to be among a small number of generated trees is very high and the probability of obtaining the correct tree is also high if we choose a tree from those generated trees under the maximum likelihood criterion.

3. The Parallel Algorithm

In our quartet-based algorithm we need to calculate a number of $O(n^4)$ quartet weights for all possible quartet trees in the first stage and all these weights need to be stored in the memory and used in the second stage. As discussed in the previous section, each quartet is associated with three weights for three possible resolved trees. For a given set of n sequences, the memory will be $\binom{n}{4} \times 3 \times 8$ bytes in size, assuming a double variable is used for each quartet weight. For example, we need 1.55GB to store all the quartet weights when $n = 200$. Therefore, the algorithm is both compute and memory intensive. A parallel program package of the tree-puzzle, a well-known method based on quartet weights, can only handle problems of size up to 250 regardless of how many processors used in the computation because it requires a whole set of quartet weights stored on every processor for the construction of intermediate trees in the puzzling stage. Our quartet-based algorithm does not have such a requirement. In the following we describe a parallel implementation of our algorithm which is able to tackle problems of much larger sizes. In our implementation the master-worker paradigm is adopted. Workflow graphs for the master and workers are depicted in Figures 4 and 5, respectively.

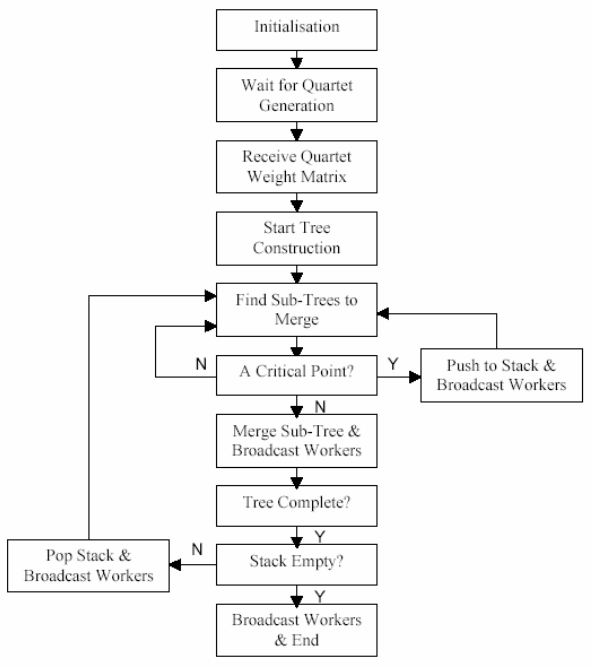

Figure 4. Master workflow graph.

Initially, the n sequences are broadcast to all the worker processors from the master for quartet weight generation in stage one. The generation of quartet weights in parallel is a good application of a well-known parallel method for combination enumeration [2]. The computation can be done embarrassingly in parallel. In our implementation each sequence is first given a number from 1 to n. This set of natural numbers is then divided into p subsets of equal size (or very close if not exactly equal) when there are p processors involved in the computation. Since there is a simple one-to-one mapping between a complete set of $\binom{n}{4}$ quartets and a set of natural numbers, with just two integer numbers, one indicating the first quartet and the other indicating the last quartet, each processor knows exactly which subset of quartets it needs to generate. After the quartet generation the subset of quartet weights is kept on each processor and used for computation in the second stage.

In stage two a global quartet weight matrix is first constructed by accumulating the weights of all possible quartet trees to the corresponding entries in the matrix. To construct this matrix each processor first builds a local weight matrix (since the quartet weights are evenly distributed among processors) and then the master processor collects the local weight matrices from the worker processors. Taking the advantage of a fast collective network on the IBM BlueGene/L, this type of collective communication can be done very efficiently.

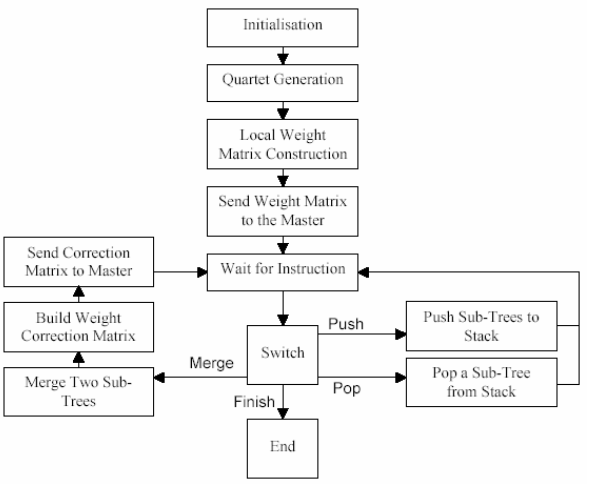

Figure 5. Worker workflow graph.

After the quartet weight matrix is built, the construction of phylogenetic trees can start using the procedure described in Figure 3. It should be noted that this procedure for tree reconstruction takes much less time to complete than the quartet weight generation in stage one. As we discussed in the previous section, it only takes $O(n^3)$ time to complete when the quartet weight correction is not concerned. We thus decide to let the master processor do the tree reconstruction and worker processors help for quartet weight correction (or quartet weight matrix correction) each time after the master has merged two sub-trees.

Since multiple trees may be constructed during the computation, in our program only quartet weight matrix, but not the original quartet weights, is modified. Since quartet weights are stored across the processors, the master processor broadcasts the merge information to worker processors each time after two sub-trees are merge. Each worker processor first updates the partially merged tree structure in accordance with the structure on the master processor and next constructs a local weight correction matrix. (It is necessary to keep a local copy of the state of sub-trees on each worker processor so that they know which quartet weights are to be corrected after each merge step and the communication cost can be minimized.) The master processor then collect all the local weight correction matrices and update its global quartet weight matrix accordingly. It takes $n-2$ steps to merge all n sequences in our algorithm. Therefore, the total communication cost for tree reconstruction is the costs for $n-2$ broadcast messages for merge information from the master to all worker processors plus the costs for $n-2$ collective messages for collection of weight correction matrices from the workers to the master processor.

For multiple tree construction we use a stack on the master processor to keep track of the possible tree merges

(critical points) waiting to be evaluated. To minimize communication costs the worker processors also maintain the same stack. Therefore, additional information will also be broadcast to the worker processors when a critical point is encountered. This message can be combined together with the message for merge information to further reduce the communication overhead.

4. Experimental Results

We tested our parallel program on a 128 node IBM BlueGene/L cluster. The IBM BlueGene/L is a new-generation massively-parallel computing system designed for research and development in computational science [15]. It is an extremely high compute-density system with relatively modest power and cooling requirements. Each BlueGene/L node consists of two 700MHz CPUs and 512 MB memory (256MB per CPU). The Blue Gene/L has implemented multiple network architectures to allow efficient communication between processors. A 3D torus network is used for node communication with neighbors. During program run some communication calls are more global than others, like all to one, one to all, and all to all. For these Blue Gene/L provides another network: the collective network. This collective network connects all the compute nodes in the shape of a tree and any node can be the tree root (originating point). The Blue Gene/L uses this network to implement MPI collective communication calls. Our algorithm makes effective use of this collective network for communication to achieve good performance. The barrier (global interrupt) network is the third dedicated hardware network the Blue Gene/L provides for efficient MPI communication.

The BlueGene/L operates in two primary modes, co-processor and virtual-node. The co-processor mode dedicates one CPU per node for computation and the other CPU for communication. This mode is particularly suited for communication bound computations. The virtual node mode allows the two CPUs in each node to act as an independent node effectively doubling the number of available processors for computation at the cost of communication speed. Due to the collective nature of communication in our parallel algorithm and length of computation time, the virtual node mode not only doubled the number of available processors, it also provided a linear speed up when compared to co-processor mode and hence was used for our experiments.

In our experiments, computation time, memory usage and communication costs of the algorithm were measured for different number of DNA sequences and across different number of CPUs. Synthetic DNA sequence data of length 4000 was used and the size of input was varied between 50 and 400 sequences. Some experimental results are presented in the following figures.

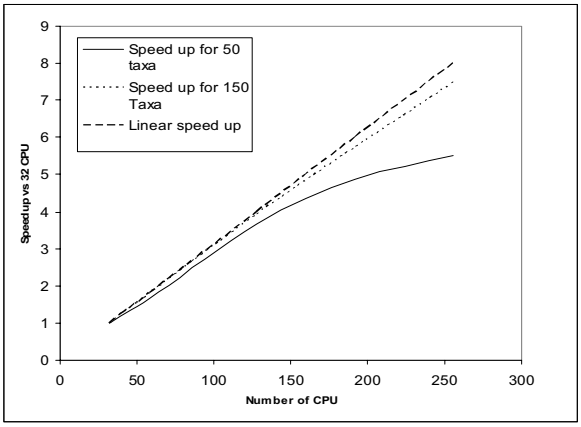

Figure 6. Speedup vs 32 CPUs for different number of sequences (or taxa).

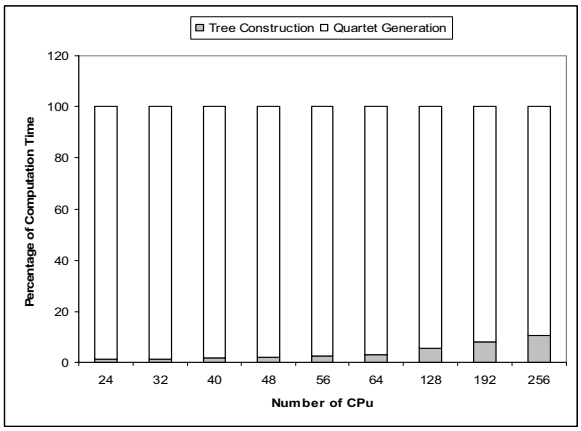

Figure 7. Percentage of Computation Time.

Figure 6 shows the performance in terms of speedup (versus 32 CPUs) for two different problem sizes. When the problem size is small, i.e., $n = 50$, the total computational cost is not high in comparison with the communication. The performance is then sub-optimal when a large number of processors are involved in the computation. This indicates that there is no need to use a large number of processors to tackle small size problems. When the number of input sequences is large, e.g., $n = 150$, the overall speedup approaches linear as the computational time for quartet generation begins to dominate. Figure 7 shows the percentages of the total computation time used for the quartet generation in stage one and the tree construction in stage two. It can be seen that the total computation and communication cost for the second stage is only a very small fraction of the total computational cost for a large number of long sequences.

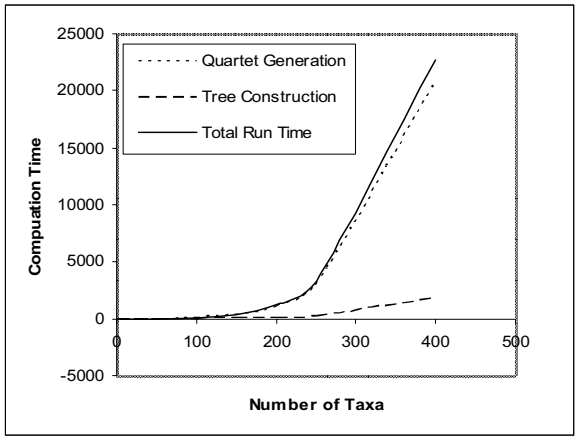

Figure 8. Computation times of program with 256 CPU.

As the number of sequences increases, we need a large memory to store the quartet weights. On a 128 node IBM BlueGene/L cluster each node has a local memory of size 512MB and the size of memory collectively is thus over 65.5GB. For a problem of size 400 the memory requirement is about 25.2 GB. Since the quartet weights are evenly distributed across the processors, we are able to easily tackle the problem of size 400 or even larger on the 128 node cluster. Figure 8 shows the computation times (in seconds) for problems of sizes from 50 to 400 using all 256 CPUs on the cluster.

Our algorithm is able to make effective use of the fast collective communication network provided by the IBM BlueGene/L cluster. We found in our experiments that for a given problem the communication time remained nearly constant as the number of processors increased. For example, the total runtime for 250 sequences running on 256 CPUs for a single tree construction is about 1.3 hours, but the total communication time is only 12 seconds, which is about the same as that when using a smaller number of CPUs.

5. Conclusions

In this paper we described a parallel implementation of an effective quartet-based algorithm we developed recently on the IBM BlueGene/L cluster. The algorithm is both compute and memory intensive and needs parallelization. By distributing the quartet weights evenly across the processing nodes and making effective use of a fast collective network on the IBM BlueGene/L cluster, we are able to tackle problems of much larger size and our experimental results show that a close to linear speedup is achievable even when the number of processors involved in the computation is large. This demonstrates that our parallel algorithm is very efficient and shows that the IBM BlueGene/L cluster is an excellent and powerful machine for scientific computing.

6. Acknowledgement

This research was partially funded by Discovery Grants (DP0557909) from the Australian Research Council. It was also partially supported by IBM, Australia.

References

[1] J. Adachi and M. Hasegawa, Instability of quartet analyses of molecular sequence data by the maximum likelihood method: the cetacean/artiodactyla relationships, *Cladistics*, Vol. 5, 1999, pp.164-166.

[2] S.G. AKL. Adaptive and Optimal Parallel Algorithms for Enumerating Permutation and Combinations. The Computer Journal. 1987 Vol 30, No. 5

[3] H. J. Bandelt and A Dress, Reconstructing the shape of a tree from observed dissimilarity data, *Adv. Appl. Math*., Vol. 7, 1986, pp.309-343.

[4] V. Berry and D. Bryant, Faster reliable phylogenetic analysis, Proceedings of 3rd *Annual International Conference on Comp. Mol. Biol*, 1999, pp.59-68.

[5] V. Berry, T. Jiang, P. Kearney, M. Li, T. Wareham, Quartet cleaning: improved algorithms and simulation, *Lecture Notes Computer Science*, Vol. 1643, 1999, pp.313-324.

[6] V. Berry, D. Bryant, P. Kearney, M. Li, T. Jiang, T. Wareham and H. Zhang, A practical algorithm for recovering the best supported edges in an evolutionary tree. *Proceedings of Symposium on Discrete Algorithms*, San-Francisco, 2000, pp.287-296.

[7] V. Berry and O. Gascuel, Inferring evolutionary trees with strong combinatorial evidence, *Theoret. Comput. Sci*., 240(2), 2000, pp. 271-298.

[8] P. Buneman, The recovery of trees from measures of dissimilarity, in: Mathematics in Archaeological and Historical Sciences (F. R. Hobson, D. G. Kendal and P. Tautum, eds.) University Press, Edinburgh, 1971, pp. 387-395.

[9] A. W. M. Dress and D. H. Huson, Constructing splits graphs, *IEEE Trans on Computational Biology and Bioinformatics*, Vol. 1, no. 3, 2004, pp.109-115.

[10] [8] P. Erdos, M. Steel, L. Szekely and T. Warnow, Constructing big trees from short sequences, *Lecture Notes Computer Science*, Vol. 1256, 1997, pp. 827-837.

[11] W. M. Fitch, A non-sequential method for constructing trees and hierarchical classifications, *J. Mol. Evol.* 18, 1981, pp. 30-37.

[12] S. Guindon and O. Gascuel, A simple, fast and accurate algorithm to estimate large phylogenies by maximum likelihood, Syst. Biol., 52, 2003, pp. 696-704.

[13] D. H. Huson, S. Nettles and T. Warnow, Obtaining highly accurate topology estimates of evolutionary trees from very short sequences, *Proceedings of The 3rd Annual Int. Conf. Comp. Mol. Biol*., 1999, pp. 198-209.

[14] D. H. Huson, Splitstree: A program for analyzing and visualizing evolutionary data, *Bioinformatics*, vol. 14, no. 10, 1998, pp. 68-73.

[15] IBM Journal of Research and Development, Vol. 49, No. 2/3, 2005; Special Issue on Blue Gene Supplementary Material.

[16] T. Jiang, P. E. Kearney and M. Li, Orchestrating quartets: Approximation and data correction, *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, 1998, pp.416-425.
[17] K. Nieselt-Struwe and A. von Haeseler, Quartet-mapping, a generalization of the likelihood-mapping procedure, *Mol. Biol. Evol.*, 18(7), 2001, pp.1204-1219.
[18] G. Olsen, H. Matsuda, R. Hagstrom and R. Overbeek, A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood, *Comput. Appl. Biosci*., Vol. 10, 1994, pp.41-48.
[19] V. Ranwez and O. Gascuel, Quartet-based phylogenetic inference: Improvements and limits, *Mol. Biol. Evol*., 18(6), 2001, pp.1103-1116.
[20] H. A. Schmidt, K. Strimmer, M. Vingron and A. von Haeseler: TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18(3), Mar 2002, pp.502-504.
[21] M. Steel, The maximum likelihood point for phylogenetic tree is not unique, *Syst. Biol.*, Vol. 43, 1994, pp.560-564.
[22] K. Strimmer and A. von Haeseler, Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies, *Mol. Biol. Evol.*, 13(7), 1996, pp.964-969.
[23] The LIRMM Methods and Algorithms in Bioinformatics research group, www.lirmm.fr.
[24] L. S. Vinh and A. von Haeseler, IQPNNI: moving fast through tree space and stopping in time, *Mol. Biol. Evol.*, Vol. 21, no. 8, 2004, pp. 1565-1571.
[25] B. B. Zhou, M. Tarawneh, C. Wang, D. Chu, A. Y. Zomaya and R. P. Brent, A novel quartet-based method for phylogenetic inference, *Proceedings of IEEE International Symposium on BIBE*, Minneapolis, Oct. 2005.
[26] B. B. Zhou, M. Tarawneh, C. Wang, D. Chu, A. Y. Zomaya and R. P. Brent, Phylogenetic Inference Using a Global Quartet Weight Matrix, submitted to *IEEE Transactions on Computation Biology and Bioinformatics*, 2005.