Multithreaded Implementation of a Biomolecular Sequence Alignment Algorithm - Software/Information Technology

Weiwei Gao
Computing and Software Department
McMaster University
Hamilton, Ontario L8S 4K1
weiwei@cas.mcmaster.ca

Sanzheng Qiao
Computing and Software Department
McMaster University
Hamilton, Ontario L8S 4L7
qiao@cas.mcmaster.ca

Abstract

This paper describes a parallel implementation of a sequence alignment algorithm for biomolecular sequence analysis. It uses multiple threaded programming for the most time consuming functions and works in X Window based interactive systems. Its sequence alignment operations include Pairwise Alignment, Star Alignment, Phylogeny Reconstruction and Generalized Tree Alignment. Both of fast and optimal modes are provided. The algorithms for Phylogeny Reconstruction, Generalized Tree Alignment, and Tree Alignment are based on heuristic stepwise addition and internal node sequence alignment induction methods.

PTAR can be used for DNA, RNA, and PROTEIN sequence analysis. In general, the system can carry out the alignments for any sequences composed of characters a-z and A-Z.

1 Introduction

Ever since the structure of DNA was unraveled in 1953, molecular biology has witnessed tremendous advances. A new scientific field: Computational Molecular Biology is created upon this. One of its major direction is pattern recognition and comparison on molecular sequences. In 1994, SAT [CHE1994] system was developed with the handling of Pairwise Alignment, Star alignment, and Tree alignment with a given phylogeny. New algorithms including Phylogeny Reconstruction and Generalized Tree Alignment were added in 1999 by TAAR [LIU1999] to make comparison of the sequences as well as construct an evolutionary tree to reveal the molecular sequence development history and mutation secrets among different creatures and/or categories. However, due to the huge amount of data involved in processing, the computation is intensive. The traditional sequential system is too slow to be applied to real world. In this paper, we present a parallel algorithm for biomolecular sequence alignment using multithread programming.

This paper is organized as follows. Section 1 introduces the alignment problem and the system development histories. Section 2 presents the main algorithms deployed in the serial alignment system. The TAAR system analysis is included in section 3 where we explore the computation bottleneck in this serial version. Section 4 presents our parallel algorithms for the most time consuming procedures and the performance analysis.

2 Serial Algorithms

Pairwise Alignment and Star Alignment are the fundamental algorithms for all the alignments. The other main algorithms are Tree Alignment and Phylogeny Reconstruction, which are exclusively called to perform alignments. In addition, Generalized Alignment also includes Multiple Sequence Alignment algorithm, which extends every sequence to the same length after doing phylogeny alignment. Since it does not take significant time in the system, we leave it out of our discussion.

2.1 Tree Alignment

First, we present some notations.
For a (rooted) tree T and each node v of T:

. $r(T)$ denotes the root of T;

. When all leaves have been assigned sequences, we define $S(v)$ to be the set of sequences assigned to the descendant leaves of v;

. When a tree's nodes are associated with sequences, we call it a *loaded* tree;

. A *loaded* tree is called a *lifted* tree if the sequence associated with each node equals the sequence associated with some child of the node.

Our Tree Alignment problem is:
Given a set $X = \{s_1, \ldots, s_k\}$ of sequences and a phylogeny T, which is defined as a binary tree such that each leaf is assigned a unique sequence of X, we need to construct a sequence for each internal node such that the total cost of the tree is minimized.

The problem of Tree Alignment with a given phylogeny is NP-hard even if the phylogeny is a binary tree [WJ1994]. Some heuristic algorithms have been proposed [HEI1989, SCL1976]. Here we use an approximation algorithm based on the results of Jiang, Lawler and Wang [JLW1994].
Step 1: Initialize the internal nodes.

Algorithm Initialization(X,T)
Input: $X = \{s_1, \ldots, s_k\}$, T (a phylogeny for X).
Output: Lifted tree $\overline{T}$.
begin
 for each pair(i, j), $1 \leq i < j \leq k$
  compute $dist(s_i, s_j)$.
 Select a virtual root for T and construct $\overline{T}$.
 for each level of $\overline{T}$, with bottom level first
  for each node v at the level
   for $i = 1$ **to** k
    if $s_i \in S(v)$ **then** compute $D[v, s_i]$.
 Select an $s \in X$ such that $D[r(\overline{T}), s]$ is minimized.
 Assign innode sequences to $\overline{T}$ by back-tracing.
end.

The cost $D[v, s_i]$ is computed as follows. For each leaf v, we define $D[v, s_i] = 0$ if s_i is assigned to v. Let v be an internal node, and v_1 and v_2 its children. For each $s_i \in S(v)$,

$$D[v, s_i] = \begin{cases} \min_{S_j \in S(v_2)} & \{D[v_1, s_i] + D[v_2, s_j] \\ & +dist(s_i, s_j)\} \\ & \text{if } s_i \in S(v_1) \\ \min_{S_j \in S(v_1)} & \{D[v_1, s_j] + D[v_2, s_i] \\ & +dist(s_i, s_j)\} \\ & \text{if } s_i \in S(v_2) \end{cases}$$

Step 2: Local Optimization.

Starting with a lifted tree obtained from step 1, we use an iterative improvement method to update and improve the sequences assigned to the internal nodes to get a better approximation. Because the degree of each in-node of the phylogeny under our consideration is 3, based on each in-node, we can construct a *3-component* which is a subtree consisting of the in-node and three edges connecting to it. Then on each *3-component* local optimization is performed by Star Alignment.

To illustrate these two steps, let us consider the phylogeny in Figure 1, which contains nine species on its nine leaves. To construct an evolutionary tree, we assign one of the nine given sequences to each internal node by applying the algorithm in Step 1. Then we divide the phylogeny into seven *3-component* shown by the circles. Local optimization is done for every *3-component* as follows. For the *3-component* in Figure 2 (a), from the labels (sequences) s_1, s_2 and s_3 of the three terminals, we can compute the sequence of the center labeled c_1 using *dynamic programming* to minimize the cost of the component. The revised c_1 can then be used to update the center label c_2 of an *overlapping 3-component* (Figure 2 (b)). The algorithm converges since each local optimization reduces the cost of the tree by at least one. Thus, if the process is repeated long enough, every *3-component* will become optimal. Although this algorithm does not necessarily result in an optimal tree, it seems that it can produce a reasonably well loaded tree after 5 iterations.

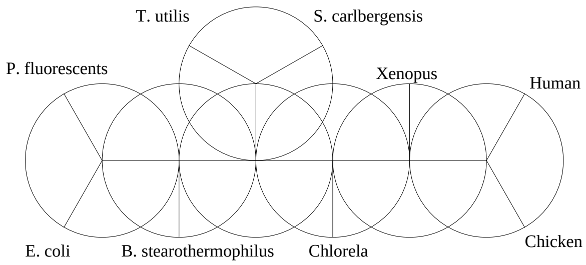
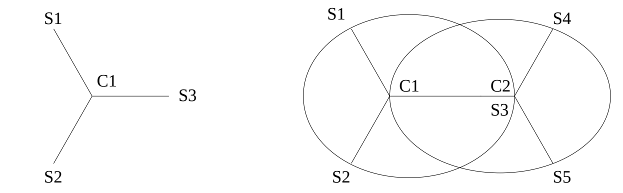


Figure 1: A phylogeny with 9 species that is divided into 7 3-components.

2.2 Phylogeny Reconstruction

For a given set of biological sequences, we usually infer and construct a phylogenetic tree that represents the evolutionary relationship among them. In the context of molecular biology, we can only



(a) A 3-component (b) Two overlapping 3-components

Figure 2: Example of 3-component.

postulate a best chosen phylogeny which represents the evolutionary relationship. In our system, we define criteria for comparing alternative phylogenies to another one and select a better one (or they are equally good) [GUS1997]. However, we can not exhaustively compare all the trees, since the total number of trees for K input sequences (taxa) [SOW1996] is: $B(K) = \prod_{i=3}^{K}(2i-5)$.

Here we use heuristic stepwise addition of taxa to a growing tree to reduce computing time. First, any three taxa are chosen for the initial star (three branch) tree. Next, one of the unplaced taxon is selected for next addition. Each of the three trees that would result from joining the unplaced taxon to one of its three branches is evaluated, and the tree with optimal cost is saved for the next round. In the next round, another unplaced taxon is connected to the tree, yet this time to one of the five possible branches on the tree saved from the previous round. This process terminates when all taxa have been joined to the tree. Figure 3 shows the whole process.

Algorithm Phylogeny Reconstruction (X)
Input: $X = \{s_1, \ldots, s_k\}$
Output: an non-rooted phylogeny T with optimal tree score.
begin
  if $k \leq 3$
    Construct the tree.
  else
    Use s_1, s_2 and s_3 to init a start tree T.
    for $i = 4$ **to** k **do**
      for each branch of the T **do**
        Join s_i to the branch.
        Do tree initialization.
        Do local optimization for T.
        Calculate the tree cost T_c.
        if first time join s_i to the branch
          $cost = T_c$ and save tree T.
        else if $cost > T_c$
          $cost = T_c$ and save tree T.
    Output the optimal tree T and score $cost$.
end.

Figure 3: Algorithm Phylogeny Reconstruction.

3 Analysis

Denote the number of leaf sequences as n, then the number of total nodes in the tree is $2n$ for an unrooted tree. We have the following analysis for the above algorithms.

- The time complexity for Pairwise Alignment is l^2, and Star Alignment is l^3, where l is the average sequence length.
- In Initialization of each Tree Alignment, the sequence pairs distance calculation is performed for every tree and Pairwise Alignment is called to compute the cost. The total time spent for each tree is: $O(n(n-1)l^2/2) = O(n^2l^2)$.
- In Tree Alignment optimization part,
  . For fast mode, in order to get the total cost of the tree, we need to know the cost for each edge. So, we do number of tree edges $(2n-1)$ times Pairwise Alignment. The time complexity is $O((2n-1)l^2) = O(nl^2)$.
  . For optimal mode, we do OPT_TIMES times optimization for the tree. Each time, Star Alignment centered by each innode is processed. Therefore, the time complexity becomes $O(OPT\_TIMESnl^3)$.
- In the heuristic Phylogeny Reconstruction algorithm, for each new taxon starting from number 4, we probe every possible branch, then run Tree Alignment on each of such constructed tree. From adding the $4th$ sequence to nth, the list of the numbers of trees is: $3, 5, 7, \ldots, 2(n-3)+1$.
  – For fast mode, the list of the time spent for each of the tree accordingly is : $4l^2, 5l^2, 6l^2, \ldots, nl^2$. So, in total, time complexity is $O(n^3l^2)$.
  – For optimal mode, the list of the time spent for each of the tree accordingly is $OPT\_TIMES \times kl^3, k = 4, 5, \ldots, n$.

In total, time complexity is $O(n^3 l^3)$.

The above analysis describes the functions having high time complexities. Our test runs show that:

1 With same number of sequences, the difference between fast mode and optimal mode is approximately a factor of sequence length. For example, for 9 sequences with 120 characters, in fast mode, it runs 2 seconds, though in optimal mode it is 172 seconds.

2 In fast mode, for the same number of sequences, the time spent is proportional to square of sequence length. For example, for 9 sequences with length around 120, TAAR runs 2 seconds, when the length is 600, it takes 48.4 seconds; In optimal mode, the time is almost a cubic function of sequence length. For example, for 9 sequences with length around 120 TAAR takes 5344 seconds, with length around 600, it takes 796574 seconds.

3 Phylogeny Reconstruction is more than 30 times of Tree Alignment when the number of sequences is 9. The factor is about the total number of trees during the reconstruction method ($\sum_{i=4}^{9}(2(n-3)+1) = 48$).

The above observations tell us the running time is dominated by function calls Tree Alignment and Phylogeny Reconstruction. The system running profile is also consistent with our analysis. It showes that in Phylogeny Reconstruction process, Paiwise/Tree Alignment takes approximately 72.8% of the running time. And among all the functions, Tree alignment has the highest running time.

From these data and analysis, we conclude that Tree Alignment and Phylogeny Reconstruction are the computation bottlenecks of the system. Looking further into the Tree Alignment function, we find that the sequence pair distance calculation is called most frequently.

4 Parallel Implementation

4.1 Multiple Threaded Programming

Multithread programming is a parallel programming strategy where multiple lightweight threads used from a single program run on multiple processors within a single address space, and they are synchronized through a shared memory. In our case, the shared memory approach has the following major advantages:

- Threads are "lightweight" comparing with processes: thread creation, existence, destruction and synchronization primitives are very cheap.
- With shared memory, multiple threads save a lot of space comparing to the conventional multiple processes running in separate address spaces.
- The avoidance of data communication between processes also features multiple threads programming with less overhead, therefore higher efficiency.

4.2 Computation Partition

The first important step in designing a parallel algorithm is to partition the problem into small, hopefully independent pieces. In general, we can partition either computation or data or both.

In TAAR, Sequence Pair Distance Calculation, Tree Alignment and Phylogeny Reconstruction have the same characteristics: the computation volume is huge, different computations are independent and share the same big amount of data. So our approach is computation decomposition without data distribution.

4.3 Implementations

4.3.1 Implementation of Phylogeny Reconstruction

When a new taxon is added, all the existing tree branches are tried. Instead of calculate their tree cost one after one, we distribute the computation of costs among m threads where m is the number of branches. Each thread calculates the cost of a new tree by adding the new taxon to one branch. Figure 4 shows our new phylogeny reconstruction algorithm.

Although we need to copy the tree structure for each thread function, the serial version also backs up the previously constructed tree for the next comparison. So, we just apply more space for our parallel implementation. We tested for different sets of input sequences, all the results show that in a two-processor system, the parallel version is about twice as fast as the sequential version. And the more the processors, the faster it is. However, the speed up decreases when the number of sequences and the sequence lengths become large, because large number

Algorithm New_Phylogeny Reconstruction (X)
Input: $X = \{s_1, \ldots, s_k\}$
Output: phylogeny T with optimal tree score.
begin
  if $k \leq 3$
    Construct the tree sequentially.
  else
    Use s_1, s_2 and s_3 to construct a start tree T.
    for $i = 4$ **to** k **do**
      Calculate the number of branches m.
      Fork m threads to calculate
       tree costs respectively.
      Barrier all the threads.
      Select the lowest cost.
      Construct the lowest cost tree.
    Do tree alignment for the last tree
     and get innode sequences.
    Output the optimal tree and score *cost*.
end.

Figure 4: Algorithm New Phylogeny Reconstruction.

of threads are created but the number of processors stays the same.

4.3.2 Implementation of Pair Distance Calculation

Given n input sequences, there are $n(n-1)/2$ pairs of distance. We fork m threads to carry out the task where $m = \lceil n/2 \rceil$. To balance the work load, we evenly assign each thread k pairs, therefore, $k = (n(n-1)/2)/\lceil n/2 \rceil$.

Our new pair distances function is described in Figure 5.

Algorithm Prepare_distarray
Output: *distarray* with pair costs filled in.
begin
  Calculate m and k.
  Get the indices for the first sequence pair.
  for $thread\_id = 1$ **to** m **do**
    Fork a thread to calculate k pairs distance.
    Get the indices for the next sequence pair.
  Barrier all the threads.
end.

Figure 5: Algorithm Prepare Distarray.

Similar to Phylogeny Reconstruction, the parallel version of pair distance calculation also speeds up by almost a factor of two in our two-processor system with 9 input sequences. When the number of sequences and the sequence lengths become large, the speed up decreases.

References

[CHE1994] Xiangdong Chen, *An Iteractive System for Sequence Analysis.*, M.Sc. thesis, McMaster University, Hamilton, Ontario, Canada, L8S 4K1, May 1994.

[GUS1997] D. Gusfield, *Algorithms on Strings, Trees, and Sequences.*, Cambridge University Press, 1997.

[HEI1989] J. J. Hein, *A Tree Reconstruction Method that is Economical in the Number of Pairwise Comparisons Used.*, Mol. Biol. Evol. 6(6), pp. 669-684, 1989

[JLW1994] T. Jiang, E. Lawler and L. Wang, *Aligning Sequences Via an Evolutionary Tree: Complexity and Approximation.*, Proc. of the 26th ACM Symp. on Theory of Computing, pp. 760-769, 1994.

[LIU1999] Feng Liu, *An Interactive System for Tree Alignment And Reconstruction System*, M.Sc. thesis, McMaster University, Hamilton, Ontario, Canada, L8S 4K1, March 1999.

[SCL1976] D. Sankoff, R. J. Cedergren and G. Lapalme, *Frequency of Insertion-Deletion, Transversion, and Transition in the Evolution of 5S Ribosomal RNA.*, J. Mol. Evol. 7, pp. 133-149, 1976

[SOW1996] D. L. Swofford, G. J. Olsen, P. J. Waddell, D. M. Hillis, Phylogenetic inference. In D. M. Hillis, C. Moritz, B. K. Mable, editors, *Molecular Systematics*, 2nd Edition, Sinauer Associates, Sunderland Massachusetts, 1996.

[WJ1994] Lusheng Wang and Tao Jiang, *On the Complexity of Multiple Sequence Alignment.*, Journal of Computational Biology, 1-4, 337-348, 1994.