An Efficient Parallel Processing Approach For Multiple Biological Sequence Alignment

Md. Maruf Monwar, *Member IEEE*, and Siamak Rezaei, *Member, IEEE*

Abstract—**Multiple biological sequence alignment is a challenging task due to its high demands for computational power, memory capacity and bandwidth and a number of novel algorithms have been developed for this. In this paper, an MPI based parallel multiple sequence alignment (MSA) algorithm is implemented with the Divide-and-Conquer approach. With this approach, the sequences are first cut down into smaller subsequences to minimize the computational space. Then these subsequences are run in parallel on different available processors using MPI. Each of those processors first builds an individual guide tree and then aligns the subsequences by Needleman-Wunsch algorithm for biological sequence comparison. After aligning, the results are then sent to the main processor where they concatenate to produce the final alignment. Because of the creation of multiple guide trees, this approach achieves a significantly better speed up than a simple MPI based parallel MSA algorithm. But some quality of the alignment is compromised for the introduction of gaps at the start or end of subsequence alignments. Therefore, some heuristic methods for fixing the cut points were suggested for future improvement.**

Index Terms— **Multiple sequence alignment, Guide tree, Divide-and-Conquer technique, Alignment sensitivity.**

I. Introduction

Multiple sequence alignment (MSA) continues to be an active field of research in Computational Biology and a number of novel approaches have been developed during the last years. Until some years ago, research on sequence alignment was mainly concerned with aligning proteins or single genes. During the last few years, however, comparison of genomic sequences became a crucial tool for uncovering functional elements such as genes or regulatory sites. Consequently, the focus of alignment research shifted to large genomic sequences. MSA helps us to organize, visualize and analyze sequence data, to estimate evolutionary distance, to highlight conserved sights or regions, and to uncover changes in gene structures etc.

Alignment of sequences in the order of hundreds of kilobases or megabases is computationally demanding and classified as an NP-hard problem [1]. Some extremely efficient tools have been developed that are able to align entire chromosomes or genomes. These approaches, however, work best on aligning closely related species; they are unable to compare sequences with larger evolutionary distances, i.e. less than 30% similarity

There are different approaches for sequence alignment, such as, exact algorithm, progressive algorithm, Divide-and-Conquer algorithm, and iterative algorithm. In this work, progressive and Divide-and-Conquer approaches are collectively used for sequence alignment.

A. Progressive Alignment

The most commonly used heuristic methods are the tree-based progressive alignment strategy. The idea is to establish an initial order (i.e. a guide tree) for joining the sequences and to follow this order in gradually building up the alignment.

The difficulty with progressive alignments is that they depend upon the initial pair-wise sequence alignments. If the sequences are closely related, then the likelihood is high that the initial alignment contains relatively few errors. However, if the initial sequences are distantly related, then there will be more errors in the alignment, which will propagate through the rest of the alignments. Furthermore, suitable scoring matrices and gap penalties must be chosen to apply to the sequences as a set [2].

B. Divide-and-Conquer Alignment

The general idea of DCA is based on Carillo and Lipman algorithm [3] to limit the computation to a smaller space and to cut each sequence in two behind a suitable *cut position* somewhere close to its midpoint. This way, the problem of aligning one family of (long) sequences is divided into the two problems of aligning two families of (shorter) sequences, the prefix and the suffix sequences. This procedure is reiterated until the sequences are sufficiently short - say, shorter than a pregiven stop size L which is a parameter of DCA - so that they can be aligned optimally by MSA. Finally, the resulting short alignments are concatenated, yielding a multiple alignment of

Manuscript received March 6, 2006. This work has been done as a requirement of an M.Sc course project.

Md. Maruf Monwar is from the University of Northern British Columbia, Prince George, Canada. He is a graduate student of Computer Science. He is also a member of IAENG and a student member of IEEE.
(phone: 1-250-612-0859; fax: 1-250-960-5544; e-mail: monwar@unbc.ca).

Siamak Rezaei is an Associate Professor of Computer Science of the University of Northern British Columbia, Prince George, Canada.
(phone: 1-250-960-6263; fax: 1-250-960-5544; e-mail: siamak@unbc.ca).

the original sequences. The following figure sketches this general procedure [1].

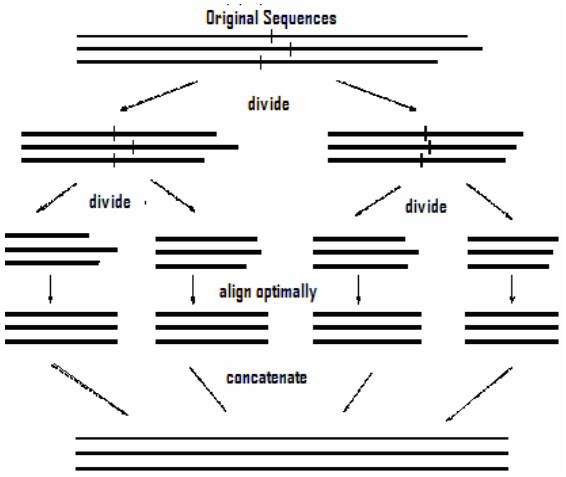

Fig. 1: General approach for Divide-and-Conquer algorithm

II. MPI-BASED DIVIDE-AND-CONQUER ALGORTITHM FOR MULTIPLE SEQUENCE ALIGNMENT

A. Basic Idea

Progressive alignments use an approximation of a guide tree between the sequences as a guide tree that dictates the alignment order. The progressive strategy is appropriate for many alignment problems, but also suffers from its greediness. Errors made in the first alignments during the progressive protocol cannot be corrected later as the remaining sequences are added in. attempts to minimize such alignment errors have generally been targeted at global sequence weighting [4], where the contributions of individual sequences are weighted during the alignment process. However, such global sequence-weighting schemes carry the risk of propagating rather than reducing error when used in progressive multiple-alignment strategies [5].

Simultaneous alignments are high quality heuristics that deliver an alignment usually very close to optimality. They nonetheless remain an extremely CPU and memory-intensive approach, applicable only to about nine sequences of average length for the fastest implementation (DCA). From Figure 1 one can easily notice that the divide-and-conquer technique actually provides a perfect structure for parallel programming, and each sub problems can be computed independently. Another major advantage of using divide-and-conquer technique is that extremely long sequences can be also acceptable by a multiple sequence alignment program as long as the sequences can be cut into small enough pieces.

Based on the characteristics of both progressive and divide-and-conquer alignments, long sequences will be first cut into several sets of sub-sequences, and each of these sub-sequences will be aligned progressively and independently by a number of processors. In some cases, the sequences are extremely long and cannot be feed very well into a simultaneous alignment program even after they are divided into several shorter pieces. That is why sometimes progressive alignment is still considered for the sub-sequences alignments.

This MPI based multiple sequence alignment approach actually combines the idea of divide-and-conquer alignment and progressive alignment. In order to check the alignment speed and sensitivity, two different alignment programs, depending on how the guide tree(s) would be applied, are done for getting a better sense. One program is called the single-tree alignment, in which a guide tree is built for the full-length sequences at the beginning, and after the sequences are cut into sub-sequences, all the sub-alignments will follow the single, uniformed tree. Multiple-tree alignment is thus the other implementation, in which sequences will be cut first and each of the sub-sequences will build their own guide tree to guide their individual alignments.

B. Algorithm

In our implementation, we have developed two programs for MSA using C++ language. One program is for single-tree implementation and the other program is for multiple-tree implementation. In the single-tree implementation, the main processor will build the guide tree first and then divide the sequences into n subsequences by using Divide-and-Conquer technique. Then one of these subsequences are kept by the main processor and the rest of the sub sequences are sent to n-1 processors in parallel. Message passing technique is used for sending these n-1 subsequences to n-1 processors. Then those processors (including main processor) will execute the MSA module of the program according to the previously built guide tree in parallel but independently [6]. After doing the subsequence alignment, all of these n-1 processors will send the alignment results to the main processors. The main processor will merge the sub sequence alignment, sent by n-1 processors, to complete the final alignment.

On the other hand, in multiple-tree implementation, the Divide-and-Conquer technique is applied first to the sequences to make subsequences. Then one of these subsequences are kept by the main processor and the rest of the sub sequences are sent to n-1 processors in parallel. Message passing technique is used for sending these n-1 subsequences to n-1 processors. Then each of those processors (including main processor) will build guide tree from their own subsequences and execute the MSA module of the program according to their own guide tree in parallel. After doing the subsequence alignment, all of these n-1 processors will send the alignment results to the main processors. The main processor will merge the sub sequence alignment, sent by n-1 processors, to complete the final alignment.

The computers used in this implementation are UNIX networked workstations from the laboratory of Computer Science department of University of Northern British Columbia, Canada. The laboratory has nearly 80 workstations. We have used maximum 10 workstations for our testing. But it can be tested with more than 10 processors. The number of processors depends on the number of cuts in the initial sequences. The minimum size that the processors can handle is

two ($L=2$) i.e., there must be at least two characters in very subsequences.

The pseudo code for our implementation (multiple tree approach) is given below.

```
Po: Read sequence
    Break sequence (sequence, subsequence[n])
    Parallel do t=0 to n do
       Send (processor[n], subsequence[n])
       Pi: Parallel do s = 0 to m do
          Break sequence (subsequence,
                          blockwisesubsequence[m])
          Send (processor[m],
                blockwisesubsequence[m])
             Pj:   Make scoring matrix
                   Send (processor, scoring matrix)
          Receive (scoring matrix[m],)
          Build guide tree(scoring matrices)
          Analyze guide tree for independency
          Send independent tree alignments
            (processor[m], independent tree elements)
             Pj:  Receive independent tree element
                  Perform alignment
                  Send partial alignment of sequence to
                                  (alignment, processor)
          Receive partial alignments from Pj
          Make alignments for subsequences by
                              progressive alignment
          Send alignments (alignment, processor)
      Receive alignment (alignment, processor[n])
  End do
  Make final alignment
End
```

III. Simulations and Comparison Results

The main idea behind implementing parallelism deployed in the program was based on the divide-and-conquer technique structure and on the creation of single or multiple-tree. On one hand it is clear that optimal cut positions exist; on the other hand it is clear that it is NP-hard to find them [1]. In this implementation, all the initial sequences are chopped into same size sub sequences according to the number of available processors.

Since progressive alignment only performs global alignment and match sequences over their full lengths, problems with this approach can arise when highly dissimilar sequences are compared. Especially when there is a large difference in the lengths of two sequences to be compared, global alignment routines become unwarranted. This is because highly similar internal regions may be overshadowed by dissimilar regions and the high gap penalties normally are required to achieve proper global matching. Moreover, many biological sequences are modular and show shuffled domains, and the repeats of internal sequence can also severely limit the applicability of global methods. Therefore, in our simulations, only sequences with similar length and over 40% identical are tested for the single-tree and multiple-tree alignment programs.

A. Speed Comparison

The major advantage of MPI programming is program speedup in terms of time, because each process processes a different piece of the same job simultaneously and independently. However, this is obviously the case for the multiple-tree alignment program, but not quite true for the single-tree alignment, which does not gain any speed improvement after some point. The reason is that in single-tree implementation, no matter how many processors are used, every time a single guide tree for the full length sequences are built at the beginning for the processors sub-alignments, and it appears to be the most time-consuming part of all alignment procedures.

For testing the implementation over various length sequences, three sample input files are chosen from a publicly available database BAliBASE (Refs 2) [7], which have benchmark alignments. The first file consists of 15 sequences each approximately 57 characters long, second file consists of 15 sequences each approximately 404 characters long and the third file consists of 34 sequences each approximately 1400 characters long. After simulations, for multiple-tree implementation we obtained a very good speedup for large sequences compared to single-tree implementation. We also obtained sufficiently satisfactory speed up for small and medium sequences. The results of the simulations are shown in the following figures:

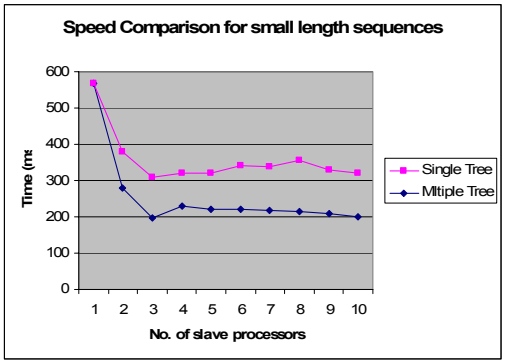

Fig. 2: Speed up with respect to the no. of processors for small length sequences

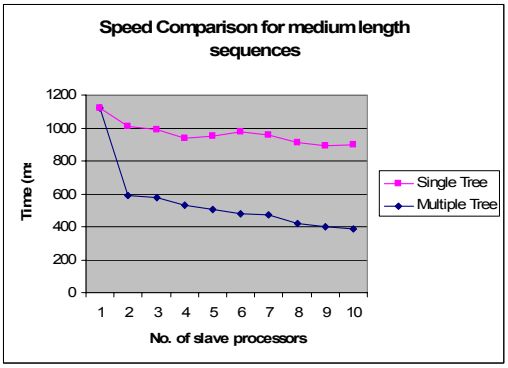

Fig. 3: Speed up with respect to the no. of processors for medium length sequences

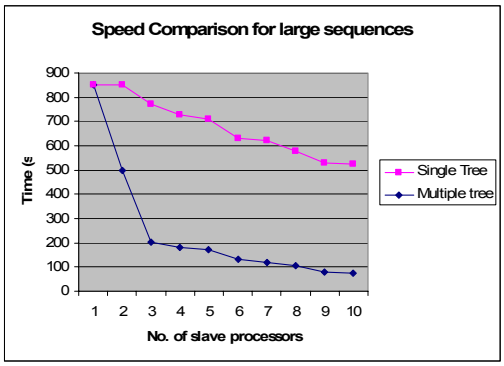

Fig. 4: Speed up with respect to he no. of processors for large sequences

B. *Alignment Sensitivity Comparison*

BAliBASE provides a module (BaliScore) that defines two scores. SP is the ratio of the number of correctly aligned pairs of positions in the test (predicted) alignment to the number of aligned pairs in the reference (structurally informed) alignment. TC is the ratio of the number of correctly aligned columns in the test alignment to the number of aligned columns in the reference alignment. Both SP and TC range from 1.0 for perfect agreement to 0.0 for no agreement. The designers of BAliBASE recommend SP as the best quality score for Refs1, 2 and 3, TC as the best score for Refs4 and Refs5.

Currently, our tests were done mainly based on Refs2, thus following figure (fig. 5) reflects the average SP scores calculated by BAliScore in terms of the number of processors for the single-tree and multiple-tree alignment programs. It turns out that the quality of alignments drops down for both approaches when the number of processors increases, as unwanted gaps are inserted at the start or the end positions of the sub-alignments, thus bring more gaps in the final full-length alignments and infects the values of SP scores.

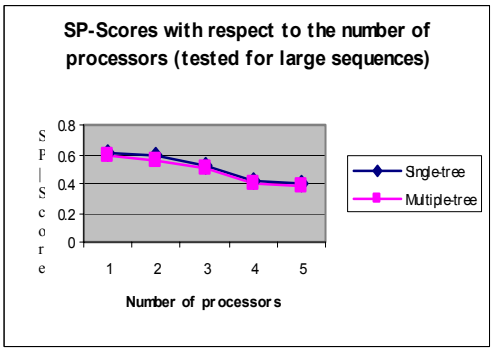

Fig. 5: SP-scores with respect to the number of processors, tested for large sequences

IV. Conclusions

Divide-and-Conquer technique with the progressive alignment approach is implemented to incorporate more parallelism in biological sequence alignment algorithm to face the problem of growing computational power in the biomedical field as the complexity and the volume of data increases. Multiple-tree alignment seems to have a better speedup performance than single tree alignment for large sequences. It also obtains sufficiently satisfactory speed up for small and medium sequences. But both approaches decrease alignment sensitivity as the number of processors increases.

V. Future Suggestions

To overcome the problem of unwanted gaps introduced at the start and end of the sub sequence alignments, which affect the sensitivity performance, the following three ways can be thought of and deployed in the future. Firstly, in stead of cutting the sequences into same size subsequence, some more effective calculations should be found and performed to decide the cut points. For this, overlapping alignment or sliding window cut points calculation approach can be used. Secondly, weights could be considered and given to the sequences – downweighting the sequences that are very similar to other ones in the data set and upweighting the most divergent sequences. The weights will be calculated directly from the branch lengths in the initial guide tree for single-tree implementation and all the guide trees for multiple-tree implementation. Thirdly, affine gap penalties and varying substitution matrices may be applied dynamically in the progressive alignment.

References

[1] A.W.M. Dress, G.FÃllen and S.W. Perrey, **"**A Divide-and-Conquer approach to multiple alignment**,"** *in **the** proceedings of the 3rd International Conference on Intelligent Systems for Molecular Biology (ISMB 95),* AAAI Press, Menlo Park, CA, USA, 1995, pp. 107-113.

[2] Joanne Bai and Siamak Rezaei, "Parallelized multiple sequence alignments with multi threads," *in the proceedings of the 27th Annual Conference of the IEEE Engineering in Medicine and Biology Society (EMBS)-2005*, Sanghai, China, Sptember 1-4, 2005.

[3] H. Carillo. and D. Lipman, "The multiple sequence alignment problem in biology", *SIAM Journal of Applied Mathematics*, vol. 48(5), 1998, pp. 1073–1082, 1998.

[4] Higgins,D.G. and Gibson,T.J., "CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice**,"** *Nucleic Acids Research,* vol. 22, 1994, pp. 4673–4680.

[5] J. Heringa, "Two strategies for sequence comparison: Profile-preprocessed and secondary structure-induced multiple alignment**,"** *Computer Chemistry*, vol. 23, 1999, pp. 341-364.

[6] J. Thompson, F. Plewniak, et al., "BAliBASE: A comprehensive comparison of multiple alignment programs," *Nucleic Acids Research*, vol. 27(13), 1999, pp. 2682-2690.

[7] J. Thompson, F. Plewniak, et al, "BAliBASE (version 2.0): A benchmark alignment database, including enhancements for repeats, transmembrane sequences and circular permutations," 1999.