Proceedings of the 2005 IEEE
Engineering in Medicine and Biology 27th Annual Conference
Shanghai, China, September 1-4, 2005

A distributed approach for a multiple sequence alignment algorithm using a parallel virtual machine

Heitor S. Lopes and Guilherme L. Moritz

Abstract— **Multiple sequence alignment is a central topic of extensive research in computational biology. Basically, two or more protein sequences are compared so as to evaluate their similarity. This work reports a methodology for parallel processing of a multiple sequence alignment algorithm (ClustalW) in an environment of networked computers. A detailed description of the modules that compose the distributed system is provided, giving special attention to the way a dynamic programming algorithm can be executed in parallel. Extensive experiments were done to evaluate performance and scalability of the method. Results show that the proposed method is efficient and offers a real advantage for large-scale multiple protein sequence alignment.**

I. INTRODUCTION

In biological systems, proteins are the most abundant and functionally diverse molecules and almost all vital processes depend on these macromolecules, which are composed by amino acids chains. The common 20 different types of amino acids can be combined in a linear sequence having the necessary information for the generation of a unique tri-dimensional structure. The comparison of two protein sequences (or a group of them) is known as alignment. It consists in the systematic comparison of the amino acids compounding the sequences throughout their whole extension (or only definite regions), and then computing a similarity score. From the computational viewpoint, the multiple alignment of sequences (proteins or DNA) is a very difficult task and it is NP-complete [2]. However, alignment is the most important tool for discovering and representing similarities between sequences, and can unravel the evolutionary history, critical preserved motifs, details of the tertiary structure or important clues about function. Therefore, multiple sequence alignment is a central topic of extensive research in computational biology [1]. There are known computational algorithms that allow finding suitable alignments among sequences (local/global alignment; pairwise/multiple alignment). The main difference between them is the quality of the alignment. Frequently, algorithms that give good alignments are computationally expensive or even unfeasible for a large number of sequences. In this work, a methodology for parallel multiple sequence alignment was developed. It was designed to address both the quality of the alignment and the processing speed. For this purpose, we choose an efficient algorithm, Clustal [3], and a parallel processing environment using networked computers, PVM (Parallel Virtual Machine) [4].

PVM is a software library that offers message-passing support and allows the exploitation of distributed computing across a network of heterogeneous computers [4]. PVM is efficient and easy to use and, thanks to its explicit communication model and process-based computation, it has become a standard in parallel computation.

A. The ClustalW algorithm

The Clustal algorithm [3] is based on a progressive alignment of sequences, using a distance tree between related alignments. In this work, we used a further improvement of the Clustal algorithm: the ClustalW [5].The algorithm is divided into three basic steps, as follows: The first step is the pairwise alignment, where a pair of sequences is aligned using the well-known dynamic programming algorithm for global alignment. It builds a $m \times n$ matrix (m and n are the length of the two sequences) and computes a score by means of a backward walk in the matrix, looking for the minimal cost associated with substitutions, insertions and deletions. This step is repeated iteratively for all $n(n-1)/2$ pairs of sequences to be aligned. The computed score is meant as the similarity degree between two sequences. The scores are computed as evolutionary distances using the model of Kimura [6]. In the next step a distance tree (a kind of phylogenetic tree) is constructed using all pairs of alignments. This tree shows the evolution of the sequences, grouped in pairs of minimum distances (scores) using the neighbor-joining clustering algorithm by Saitou and Nei [7]. This tree is a profile of the order in which sequences should be aligned for maximal efficiency of the next step. Finally, a progressive alignment of the previous alignments is done, traversing the distance tree in order of decreasing similarity: sequence-sequence, sequence-profile, and profile-profile alignment, thus reaching the final result. This result does not have a score and it is not necessarily the best possible alignment for the sequences under study. The optimum multiple sequence alignment can be obtained with multidimensional dynamic programming, but the time complexity is $O(2^N L^N)$, where N is the number of sequences and L the average length of the sequences [1][8].

II. METHODOLOGY

A. Architecture of the system

The proposed architecture is based on a Master-Slaves approach. At the low level, the system is divided into modules. A central module (Manager), running in the Master

This work was supported by the Brazilian National Research Council – CNPq, under grants 350053/03-0 and 475049/03-9.

H.S. Lopes and G.L. Moritz with Bioinformatics Laboratory/CPGEI, Federal Center for Technological Education of Parana, Av. 7 de setembro, 3165 – 80230-901 Curitiba, Brazil hslopes@cpgei.cefetpr.br and moritz@cpgei.cefetpr.br

computer, controls the PVM environment and the data flow to and from the Slaves, and constructs the distance tree (see below). The Manager enquires Slaves cyclically for completion of every task allowing a dynamical adaptation of the system to the load. In the Slaves, the running modules are:

- ReceiveX: It manages the computation of pairwise scoring in each Slave host.
- ReceiveP: It receives two files that can be either sequence files (to be aligned) or alignment files. The latter file type is used by Pairwise-aligner and Din2 modules to create a new alignment.
- Pairwise-scorer: This module operates over a pair of sequences and returns only the score of the alignment.
- Pairwise-aligner: A modified version of the dynamic programming algorithm that operates over a pair of sequences, but returning a file with the alignment itself.
- Din2: This module is responsible for managing the parallelization of the Smith-Waterman algorithm [8], using up to three hosts. It is he heart of the system and shall be explained later.

B. Scoring Pairwise Alignments

When the Manager is started in the Master host, it will seek for the Slaves hosts added to the PVM environment that are up to run [4]. It initializes the ReceiveX and Pairwise-scorer modules on the Slaves. In principle, the system will use all available Slave hosts. Following, the Manager sends to the Slaves a file with all the sequences to be aligned. Next, it computes the total number of score-computing operations, that is, the number of all possible pairs of sequences. This value is divided by the current number of Slaves up to be allocated. Therefore, each Slave will receive a vector of N ordered pairs, corresponding to the sequences the current host is in charge of aligning. N is obtained according to equation 1:

$$N = int\left\{\frac{N_{Proc}}{N_{Slaves}} \times [1 \pm rand(0.1)]\right\} \qquad (1)$$

where: $int\{.\}$ is a function that returns the integer part of the argument; N_{Proc} is the number of processes; N_{Slaves} is the number of available hosts to perform the computations, except the host where Manager runs; and $rand(0.1)$ is a random generated number in the range 0 to 0.1. The small variation in N, that is, $\pm 10\%$) is aimed at giving a small different load distributed between hosts, so as to avoid Slaves finish their jobs about at the same time, thus overwhelming the communication with the Master.

In the sequence, Manager gets idle waiting for further communication from Slaves. ReceiveX modules on Slaves sends a pair of sequences (pointed by the vector of ordered pairs) to the Pairwise-scorer module. Recall that Pairwise-scorer is based on the first step of CLUSTALW algorithm [5], when dynamic programming is used to compute the pairwise evolutionary distance between two sequences using the Kimura approach [6]. The computed evolutionary distance is returned back to ReceiveX module.

This process is repeated until all pairs of sequences are processed. Then, ReceiveX contacts the Manager and sends back a triplet composed by: a vector containing the same N ordered pairs previously received and the corresponding evolutionary distance for each pair. After sending those data, the Slave gets idle. After receiving all replies from Slaves, Manager has all information necessary to build a score matrix.

C. Distance Tree Building

The next step of the ClustalW algorithm is the construction of the distance tree. This procedure is essentially sequential and, fortunately, it is not computationally intensive, even for a large number of sequences. Therefore, it is executed by the Master. The result of this step is a guide to the successive alignments of the next step (Profile Alignment).

There are two possible ways to construct the distance tree. The construction of the branches of the tree follows the decreasing order of scores obtained before. Therefore, the highest scores will be leaf nodes of the tree. Next, the branches of this level are analyzed to find the nearest ones, which, in turn, will constitute an internal node of the tree, at a higher level. This process is repeated until all branches converge to the root. Constructing the distance tree in this way enables the parallelization of the profile alignment (next step), since each pair of leaf nodes can be processed separately in a Slave, and several processes can be done in parallel. Notwithstanding, this procedure imposes an important drawback. As the process advances, the size of the profiles to be aligned increase, since more sequences are added. Sequences to be aligned are added $(2n-1)$ by $(2n-1)$, where n is the depth of the node in the tree. This fact makes the computational effort grows exponentially and become unfeasible even for a moderate number of sequences. Another way to construct the tree is the one used by ClustalW. The two most similar sequences are connected by the first branch of the tree. The next sequence most similar to the previous ones is then connected by the second branch and so on until reaching the root. By using this procedure, the computational effort grows linearly with the number of sequences but, on the other hand, only one process can be run at a time, since it branch depends upon the previous one.

D. Profile Alignment

Profile alignment uses the distance tree built in the previous step. It is divided into phases that use different modules based on the dynamic programming algorithm adapted for profile alignment, as follow.

1) Dynamic programming for profile alignment: The dynamic programming algorithm is based on the Smith-Waterman algorithm [8] and it is base on a $H \times W$ matrix, where H is the length of the profile that was placed in the column and W is the length of the profile sequence that was placed in the line, both with an added gap in the first cell. The elements of the matrix ($M(i,j)$) are computed sequentially, due to the data dependency inherent to the algorithm. The

computation of $M(i, j)$ is based on the sum of all evolutionary distances between all possible combinations between amino acids of the line and the column. In this work we used BLOSUM62 [9] as the evolutionary distance matrix.

2) Parallelizing the dynamic programming algorithm: For the leaf nodes of the distance tree constructed in section II-C, several processes run in parallel, each one in a Slave host. For the remaining levels of the tree, besides the regular parallelizing approach, a single process can be run in more than one host. In the beginning, the Manager starts modules ReceiveP and Pairwise-aligner in all the available Slave hosts. When all leaf nodes of the tree are already processed, the Pairwise-aligner is deactivated. In the subsequent levels of the tree (inner branches), a new way to process multiple sequences is accomplished by means of module Din2.

3) Partition of the dynamic programming matrix: The construction of the dynamic programming matrix for profile alignment is a recursive process (following [8]) and, therefore, direct parallelization is not possible. In this section we show how the matrix can be partitioned so as to allow concomitant processes to compute it.

Figure 1-left shows how the matrix can be partitioned in three regions. In region 1 (in white), cells in the borders must be calculated first. Next, cell $M(2, 2)$ is computed. At this point it is possible to start two additional parallel processes, for computing cells in regions 2 and 3 at the same time. Using such approach, up to three processors can be employed, exploiting the potential of parallel computation. When these processes are started, the first processor takes control of the task division and computation of the cells necessary for the recursivity (top row, leftmost column and main diagonal) and other two are responsible for computing cells in regions 2 and 3.

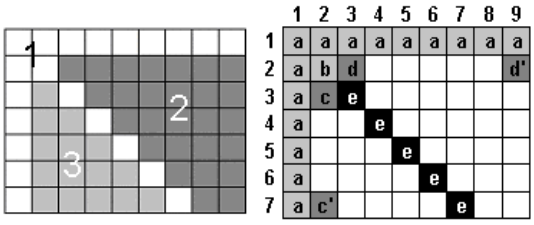

Fig. 1. Partition of the dynamic programming matrix for parallel processing (left). Steps for computing cells of the matrix (right).

4) Din2 module: In the original algorithm, sequences to be aligned were added $(2n - 1)$ by $(2n - 1)$, where n is the reverse depth of the node in the tree. This fact makes the computational effort grows exponentially. To circumvent such problem, it is necessary to change the scheme used before and, therefore, module Din2 was developed. Actually, this module applies the dynamic programming algorithm in parallel, adapting itself dynamically to the availability of processors for the current level of the distance tree. This module can start up to three processors to do the task, as explained before, and behaves transparently to the Manager. For the progressive alignment, cells marked with "a" are computed first (see figure 1-right), by the Din2 module started the host. As soon as cell marked with "b" is computed, two parallel processes (in the Slaves) can be started, just to compute either cells in the column (marked with "c") or those in the row (marked with "d"). As soon as the first elements of column and row are computed, the next diagonal cell can be processed (marked with "e") and so on.

III. Computational Experiments and Results

Several experiments were done to evaluate the performance of the proposed system. Experiments focused on the number of sequences to be aligned and their length. The reduction of processing time due to parallelism was also investigated. Although PVM supports heterogeneous networks, we used a homogeneous set of computers in this work, formed by desktop PC computers with AMD Athlon ®XP 2.4+ processors and 512 MB RAM, running Microsoft Windows 2000-SP4, connected in a switched 100 Mbps local area network. Since PVM uses the Remote Shell protocol (RSH) (native for the UNIX operating system), some adaptations in the environment were needed to run under Windows 2000.

The first experiments are aimed at discovering how the number of sequences affect the performance of the system, as a function of the number of hosts available for processing the Pairwise-score module. It should be noted that a minimal system comprises 2 hosts: the Master and one Slave, because the former does not process (just manages) and the latter does not process without the first. In this experiment, the number of sequences to be aligned (N_{seq}) was varied between 40 to 800, and the length of profiles were arbitrarily fixed in 1200 columns. Processing time was recorded for 2, 3 and 4 hosts. Figure 2 shows how the use of parallel computation speeds up the process, by using 3 and 4 hosts, relative to a basic 2-hosts system.

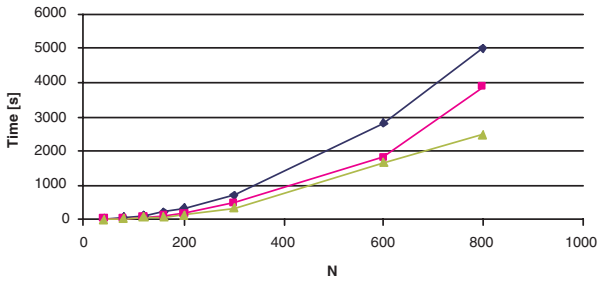

Fig. 2. Processing time of Pairwise-score as function of the number os sequences, for 2 hosts (upper line), 3 hosts (middle line) and 4 hosts (lower line).

The next experiments investigated how the processing time is affected as the number of sequences per profile grows. Again, the length of profiles were arbitrarily fixed in 1200 columns and the number of sequences in each profile was changed between 20 and 1000. The results for this experiment is shown in figure 3, where N is the number of sequences per group to be aligned. The time needed by

the parallel process using two hosts besides the Master was about the same as when using one host and the Master (lower line in figure 3).

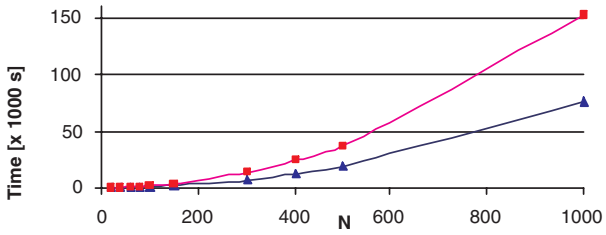

Fig. 3. Processing time of Din2 as function of the number of sequences per profile, using only the Master host (upper line) and using one or two hosts besides the Master (lower line).

The next experiments investigated how the processing time was affected as the length of the profiles grow. For these experiments, the length of both profiles were changed and the number of sequences per profile was kept constant in 50 (that is, two profiles of 50 sequences each were aligned at a time). Profiles used in this experiment come from the previous step of the multiple sequence alignment process, and so, they do not represent the final alignment. Results are presented in figure 4, for profile lengths in the range 50 to 2500. This figure shows the time needed by the sequential process, and the time needed by the parallel process simulated in a single host.

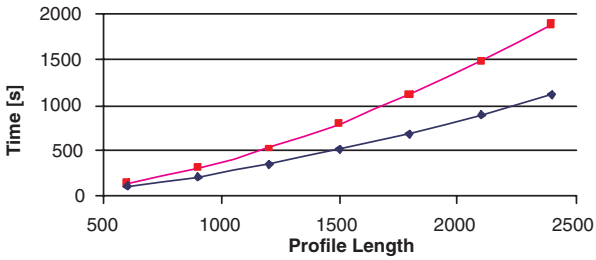

Fig. 4. Processing time as function of the length of profiles, running sequentially (upper line) and simulating parallel processing in a single host (lower line).

IV. Discussion and Conclusions

We have proposed a methodology for parallelizing a multiple sequence alignment algorithm using a network of PCs. This system self-adjusts to the number of available hosts (from one to three), managing all necessary operations for the global alignment of sequences.

For a small number of sequences to be aligned (say, $<$200), there is no significant difference between parallel and sequential processing. The same holds for profile length and number of sequences per profile. This is due to the communication overhead between hosts, necessary for data and control flow. For all experiments, as the size of the problem increases (that is, the number os sequences, or length of profiles), the difference in performance along the number of available hosts become more significant. A deeper analysis of this performance reveals that this improvement by using the parallel algorithm can achieve almost 50% of speed-up, when compared with the sequential processing. In fact, the speed-up does not grows monotonically as the problem size grows.It displays a asymptotic behavior, tending to a maximum gain of around 50%. Future investigation will address this issue. Furthermore, curves shown indicate that running the algorithm with three hosts, the processing time tends to grow polynomially as the size of the problem increases (complexity analysis is not shown here). This is a remarkable fact, since for large-scale problems, processing time can become prohibitive.

Multiple sequence comparison by alignment is an important, and still opened question, in computational biology. We believe that the proposed algorithm is a useful contribution to the area of research. In the next future we intend to do more extensive experiments and put the system freely available for research purposes.

References

[1] A.R. Leach, *Molecular Modelling: Principles and Applications*, 2^{nd} ed. Prentice-Hall, Dorset, 2001.

[2] L. Wang and T. Jiang, On the complexity of multiple sequence alignment, *J. Comp. Biol.*, vol. 1, 1994, pp. 337-348.

[3] D.G. Higgins and P.M. Sharp, CLUSTAL: a package for performing multiple sequence alignments on a microcomputer, *Gene*, vol. 73, 1988, pp. 237–244.

[4] A. Geist, A. Beguelin, J. Dongarra et al. *PVM - Parallel Virtual Machine*, MIT Press, Cambridge, 1994.

[5] Thompson, J.D., Higgins, D.G., Gibson, T.J. CLUSTALW: improving the sensitivity of progres-sive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice, *Nucl. Ac. Res.*, vol. 22, 1994, pp. 4673-4680.

[6] M. Kimura, *The Neutral Theory of Molecular Evolution*, Cambridge University Press, New York, 1983.

[7] N. Saitou and M. Nei, The neighbor-joining method: a new method for reconstructing phylogenetc trees, *Mol. Biol. Evol.*, vol. 4, 1987, pp. 406-425.

[8] T.F. Smith and M.S. Waterman, Comparision of bio-sequences. *Adv. Appl. Math.*, vol. 2, 1981, pp. 482-489.

[9] S. Henikoff and J.G. Henikoff, Amino acid substitution matrices from protein blocks, *PNAS*, vol. 89, 1992, pp. 10915-10919.