Computing and Informatics, Vol. 29, 2010, 1233–1250

A PARALLEL ALGORITHM FOR LARGE-SCALE MULTIPLE SEQUENCE ALIGNMENT

Heitor S. Lopes, Carlos R. Erig Lima, Guilherme L. Moritz

Bioinformatics Laboratory/CPGEI
Federal University of Technology – Paraná
Av. 7 de setembro, 3165
80230-901, Curitiba, Brazil
e-mail: {hslopes, erig}@utfpr.edu.br

Manuscript received 16 October 2008; revised 22 January 2009
Communicated by Viera Šipková

Abstract. Multiple sequence alignment is a central topic of extensive research in computational biology. Basically, two or more protein sequences are compared to evaluate their similarity and to identify conserved regions. 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 is run in multilevel parallelism. Extensive experiments were done to evaluate performance and scalability of the reported method. Results suggest that the proposed method is very promising for large-scale multiple protein sequence alignment.

Keywords: Bioinformatics, parallel algorithm, multiple sequence alignment

Mathematics Subject Classification 2000: 68W10: Parallel algorithms; 92-08: Computational Methods; 92D20: Protein sequences

1 INTRODUCTION

In biological systems, proteins are the most abundant and functionally diverse molecules. Almost all vital processes depend on these macromolecules that are composed by amino acids chains. The standardized 20 different types of amino acids can

be combined in a linear sequence having the necessary information for the generation of an unique tri-dimensional structure. The comparison of two protein sequences is known as pairwise alignment (or, sometimes, as string editing), and a group of them is known as multiple sequence alignment (MSA). Such alignment consists of an ordered and systematic comparison of the amino acids belonging to the sequences throughout their whole extension (or specific regions), and then computing a similarity score. From the computational viewpoint, the MSA of proteins or DNA is a very difficult problem and was proved to be NP-complete [10, 20]. However, alignment is still the most important tool for discovering and representing similarities between sequences, capable of unraveling the evolutionary history, critical preserved motifs, details of the tertiary structure and important clues about function. Real-world proteins can have up to several hundreds of amino acids and researchers often have to align a large number of them. Consequently, MSA is a recurrent topic of extensive research in bioinformatics and computer science, aiming at finding more efficient algorithms, as well as speeding-up existing ones [1, 3, 4, 12, 13, 14].

Several efficient algorithms have been proposed for the pairwise alignment problem (either for local or global alignment) [2, 3, 7, 18], including some that use parallel approaches. Due to the high level of complexity of the MSA, when compared with pairwise alignment, it has been addressed much more sparsely in recent literature, usually by using heuristic approaches [14, 16, 21]. In general, apart from the computational efficiency, the main difference between alignment algorithms is the quality of the alignment. Frequently, algorithms that give good alignments (regarding the similarity score) are computationally expensive and tend to be unfeasible for a large number of sequences.

Parallel computing has been used do deal more efficiently with several problems in bioinformatics. Amongst the many parallel processing resources, PVM (Parallel Virtual Machine) [6] is an environment frequently used for scientific applications. PVM is a software library that offers message-passing support and allows the exploitation of distributed computing across a network of heterogeneous computers. 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.

In this work, a methodology for the parallelization of a MSA task was developed. It was designed to establish a tradeoff between the quality of the alignment and the processing speed. The methodology is inspired by Clustal [9], a well-known algorithm for MSA, and is based on a parallel processing environment.

1.1 The ClustalW Algorithm

The Clustal algorithm [9] 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 original Clustal algorithm: the ClustalW [19].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 a dynamic programming algorithm for global alignment. It builds a $m \times n$ matrix (m and n are the lengths 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 [11]. 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 [17]. 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 [12, 18]. However, its time complexity $O(2^N L^N)$ (N is the number of sequences and L the average length of the sequences) makes it unfeasible even for a moderate number of sequences.

2 METHODOLOGY

2.1 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 computer, controls the PVM environment and the data flow to/from the Slaves. It also 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. Such dynamical adaptation to hosts availability is illustrated in the two possible configurations of the system, shown in Figure 1. The system can operate with a Master controlling several Slaves, which can be either active or inactive in a given moment (Figure 1-left). Alternatively, a given Slave can be transformed into a kind of "second-class" Master (running as Din2-Master, see below) to control some active Slaves, thus dividing tasks (1-left)-right).

Figure 2 shows a flowchart of the process in both Master and Slave hosts. The interrupted lines represent actions that change the current state in another process (host). Full lines represent the program flow within a given process (host). At the end of this process, the Manager will have the score matrix (see below) required for the next step.

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.

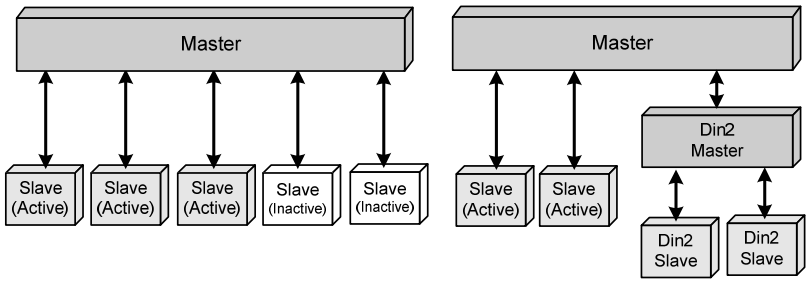

Fig. 1. Two possible operational configurations of the system

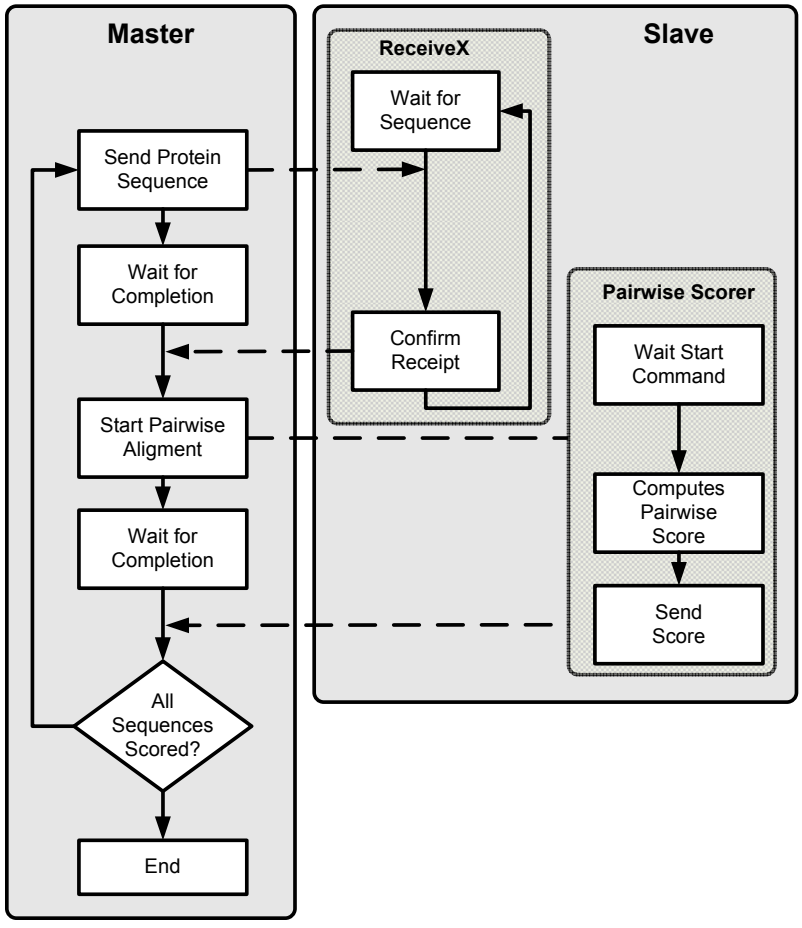

Fig. 2. Flowchart of the process in both Master and Slave hosts

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 [18], using up to three hosts. It is he heart of the system and shall be explained later.

2.2 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 ready to run tasks [6]. It initializes the ReceiveX and Pairwise-scorer modules in the Slaves. In principle, the system will use all available Slave hosts. Then 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 able 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 the Equation (1):

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

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, except the one where Manager runs; and
- $rand(0.1)$ is a random generated number in the range 0 to 0.1.

The small perturbation in N, that is, $\pm 10\,\%$, is aimed at giving a small, but important, difference in load distributed between hosts. When running in a network of homogeneous processors, the purpose of this procedure is to avoid Slaves finishing their jobs at about the same time, thus overwhelming the communication with the Master. For the experiments described in Section 3, we used $\pm 10\,\%$ of random variation in the load balancing. This value was obtained empirically, and turned out to be adequate for the range of experiments done. However, more efficient mechanisms for load balancing shall be investigated in future implementations.

Next, Manager enters in an idle state, waiting for further communication from Slaves. ReceiveX modules on Slaves send 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 [19], when dynamic programming is used to compute the pairwise evolutionary distance between two sequences using the

Kimura approach [11]. 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 the information necessary to build a score matrix.

2.3 Distance Tree Building

After scoring all the pairwise alignments, the next step of the ClustalW algorithm is the construction of the distance tree. This procedure is, by itself, essentially sequential and is executed only 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. Then, the branches of this level are analyzed to find the next highest scores, which, in turn, will constitute an internal node of the tree, at a level just above the leaves. 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 one traverses the tree upwards, the size of the profiles to be aligned increases, 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 grow exponentially and become unfeasible even for a moderate number of sequences.

Another way to construct the tree is that 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. On the other hand, only a single process can be run at a time, since its branch depends upon the previous one. In this work we used a modified version of this approach, as shown later.

2.4 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 follows.

2.4.1 Dynamic Programming for Profile Alignment

The dynamic programming algorithm is based on the Smith-Waterman algorithm [18] and uses a $H \times W$ matrix, where H is the length of the profile sequence placed in the column, and W is the length of the profile sequence placed in the row, both with a gap added in the first cell. Due to the data dependency inherent to the algorithm, the elements of the matrix ($M(i,j)$) are computed sequentially. The computation of $M(i,j)$ is based on the sum of all evolutionary distances between all possible combinations between amino acids of the row and the column. In this work we used BLOSUM62 [8] as the evolutionary distance matrix. It should be noted that other evolutionary distance matrices could be used as well, depending on the nature of the sequences to be aligned.

The elements of $M(i,j)$ are computed as follows. Cell $M(1,1)$ is set to zero, since the evolutionary distance between any number of gaps is null. Let Gp be the gap penalty, Naa the number of elements in the row (row) or column (col) profiles, $NG(i)$ the number of gaps in the i-th position of a profile. Cells $M(1,i)$, $i = 2, \ldots, W$ are calculated using Equation (2), and cells $M(i,1)$, $i = 2, \ldots, H$ are calculated using Equation (3):

$$M(1,i) = Gp.Naa_{col}.[Naa_{row} - NG_{row}(i)] + M(1,i-1) \tag{2}$$

$$M(i,1) = Gp.Naa_{row}.[Naa_{col} - NG_{col}(i)] + M(i-1,1) \tag{3}$$

The inner cells of the matrix are calculated using Equation (4) below. In this equation, $D(i,j)$ is defined as the sum of all evolutionary distances between all possible combinations between amino acids of the row and the column. For example, in Figure 3, the value of $M(2,2)$ would be the sum $B(-,S) + B(W,S) + B(I,S) + B(I,S) + B(-,K) + B(W,K) + \ldots + B(I,S)$, where $B(x,y)$ is the value of the evolutionary matrix for a pair of amino acids x and y. The value of a gap penalty is an user-defined parameter. By default, it is set to -10, if a value is a gap and the other is an amino acid, and set to zero if both are gaps.

$$\begin{cases} M(i,j) = max(A,B,C) \\ A = [M(i,j-1) + Gp.Naa_{row}] \\ B = [M(i-1,j) + Gp.Naa_{col}] \\ C = [M(i-1,j-1) + D(i,j)] \end{cases} \tag{4}$$

2.4.2 Parallelizing the Dynamic Programming Algorithm

For the leaf nodes of the distance tree constructed in Section 2.3, several processes run in parallel, each one in a Slave host. For the remaining levels of the tree, besides the regular parallelizing approach, a further level of parallelization is possible: 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 Slave hosts available at the moment. When all leaf nodes of the tree are already processed, the Pairwise-aligner

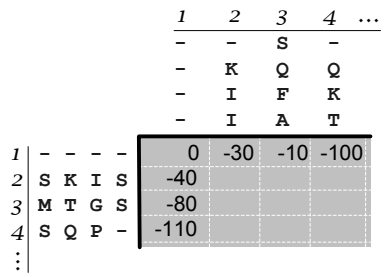

Fig. 3. A detail of the dynamic programming matrix for profile alignment

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.

2.4.3 Partition of the Dynamic Programming Matrix

The construction of the dynamic programming matrix for profile alignment is a recursive process (following [18]) and, therefore, massive 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 4-left shows how the matrix can be partitioned in three regions. In region 1 (in white), edge cells 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.

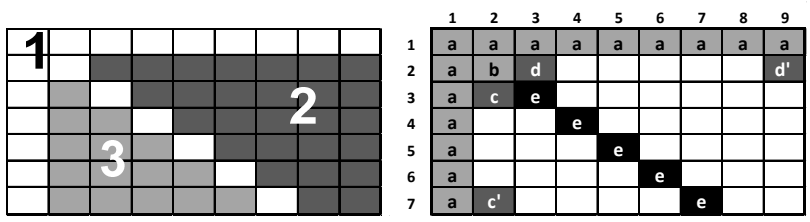

Fig. 4. Partition of the dynamic programming matrix for parallel processing (left). Steps for computing cells of the matrix (right)

2.4.4 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. As mentioned before,

this fact makes the computational effort grow exponentially. To circumvent such problem, 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 is able to 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 4-right), by the Din2 module started the host. As soon as the 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.

Figures 6 and 5 show the state machines that control the communication between processes. Tables 1 and 2 show the finite state machines (FSM) corresponding to Figures 6 and 5, respectively. In these tables, CS is the current state, SD is the state description, EV is the event that causes a transition, ED is the event description, and NS is next state. Figure 6 illustrates the behavior of the Manager module when it commands a profile alignment process using a Din2 module and when it commands pairwise alignments using Pairwise-Aligner module. Figure 5 shows the behavior of the Din2 module operating like a Din2-Master or a Din2-Slave.

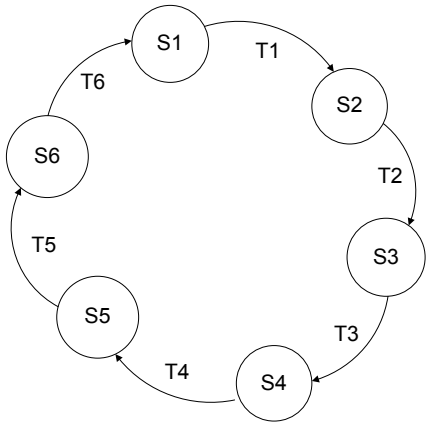

Fig. 5. State machine of Manager module

In the beginning of the second step of profile alignment, a Din2 module is started in all hosts currently available in the PVM environment. Then, start commands are sent out by the Manager to them. While the number of alignments is larger than the number of available hosts, everything happens exactly like the Pairwise-aligner mentioned before. However, as the alignments are done, more hosts get idle. At this point, tasks are divided to be run in parallel. The parallel processing is controlled by Din2 module, although hosts are allocated by the Manager.

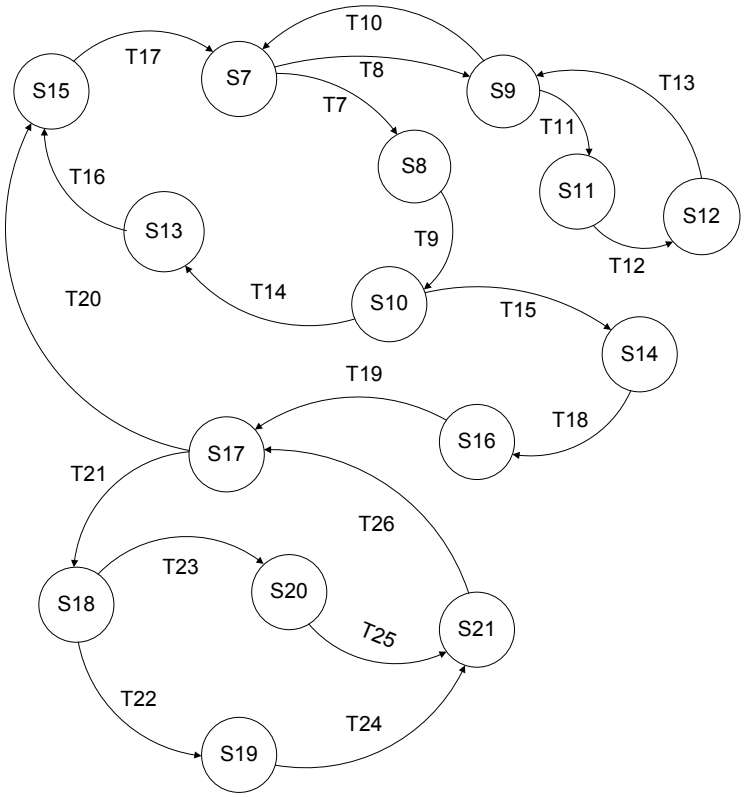

Fig. 6. State machine of Din2 module

Firstly, all hosts are idle waiting for a command from Manager, and they all have the same hierarchical level, that is, Slaves of the main process (Manager) – state $S7$ in Figure 6. Then, Manager sends out two files to a given host (state $S2$), with profiles to be aligned. Next, Manager sends a start command (state $S3$) to module Din2 of the host to whose profile files were sent. After receiving the acknowledgement of the command, Manager sends a list of the hosts currently idle

CS	SD	EV	ED	NS
S1	Master-Start	T1	New sequences	S2
S2	Send sequences	T2	ReceiveP Ack	S3
S3	Send Master Start	T3	Request Idle machine list	S4
S4	Send Idle host list	T4	End of S3	S5
S5	Lock Din2-slave hosts	T5	Results received	S6
S6	Unlock Din2-slave hosts	T6	End of S5	S1

Table 1. Tabular description of the FSM of the Manager process

CS	SD	EV	ED	NS
S7	Din2-Start	T7	Receive sequences from Master	S8
		T8	Receive sequences from Din2-master	S9
S8	Send Ack	T9	Receive Master Start	S10
S9	Wait Command	T10	Command = End	S7
		T11	Command = Coordinates	S11
S10	Request Idle machine list	T14	Idle host list (4) and Host idle	S13
		T15	Idle host list (4) and Host is not idle	S14
S11	Compute Vertical Cells	T12	End of S11	S12
S12	Send Cells	T13	End of S12	S9
S13	Compute alignment sequentially	T16	End of S13	S15
S14	Start Slave	T18	End of S14	S16
S15	Send Results	T17	End of S15	S7
S16	Send sequences	T19	End of S16	S17
S17	Compute Recursive Cells	T20	End of Cells	S15
		T21	More Cells	S18
S18	Send vertical CR command	T22	More hosts	S19
		T23	No more hosts	S20
S19	Send Horizontal CR command	T24	End of S19	S21
S20	Compute horizontal cells	T25	End of S20	S21
S21	Wait for Vertical and/or Horizontal Cells	T26	Receive Vertical and/or Horizontal Cells	S17

Table 2. Tabular description of the FSM of the Din2 process

(state $S4$). Module Din2 that has just received the list sends out a "ping" command to all listed hosts until it finds two ones available for processing (if possible). Those hosts that reply the "ping" will be used as auxiliary in the computation of dynamic programming for profile alignment, thus working at a lower level of parallelization. The same Din2 module requests Master to lock the auxiliary hosts (state $S5$), ending its iteration with the Master. Notice that now on Din2 modules do not have the same hierarchical level as before. The first Din2 module (that contacted by the Manager) is considered as a "Din2-Master" and those that replied the "ping" command are considered as "Din2-Slaves".

Now, the configuration phase begins, and the behavior of module Din2-Master depends on the number of hosts that replied the "ping" command (state $S10$). If no reply was obtained, the process will be essentially sequential. In this case, Din2-Master needs only to read the file with the sequences to be aligned and to start the computation (state $S11$). In the case that one or two auxiliary hosts have replied, they have to be configured as follows. Din2-Master sends the file with sequences to be aligned to the Din2-Slaves (state $S12$). After receiving the file, they will be in standby mode until receiving a command from Din2-Master. This command can be

a "Compute Request" (CR) to start computation, or "End of Process" (EOP) to go back to the beginning (state $S9$).

The progressive alignment is done as follows. Din2-Master computes all edge penalty values of the dynamic programming matrix. The corresponding cells of the matrix are marked with "a" in Figure 4-right. These cells are necessary for recursivity and must be calculated prior to the remaining cells, by the Din2-Master (state $S17$ in Figure 5). Next, Din2-Master starts sending CR commands to the enabled Din2-Slaves (state $S18$). In Figure 4-right it is supposed that two auxiliary hosts are available for computation of the second row and second column of the matrix. Hence, Din2-Master will issue two CR commands, one to each Din2-Slave. A CR command is composed by three elements:

- The value of the cell necessary for recursivity (marked with "b" in figure 4-right);
- The initial coordinates of the matrix from where Din2-Slave have to start computing cells (marked with "c" in Figure 4-right, in the case of computing a column; or with "d", in the case of computing a row);
- The final coordinates of the matrix up to where Din2-Slave have to compute cells (marked with "c" in Figure 4-right, in the case of computing a column; or with "d", in the case of computing a row).

After managing the computation of cells of the current row and column by the Din2-Slaves, Din2-Master computes the next cell required for recursivity and the whole process is repeated. The required cells for recursivity at each step are marked with "e" in Figure 4-right. The number of cells computed by each of the two auxiliary hosts is given by Equation (5) when the horizontal (row) profile is larger than the vertical (column) one. Similarly, Equation (6) gives the number of cells computed when the vertical (column) profile is larger than the horizontal (row) one.

$$\begin{cases} \mathit{Master} = 2(H-1) + W \\ \mathit{Vertical} = H(H-3)/2 + 1 \\ \mathit{Horizontal} = W(H-1) - H(H+1)/2 + 1 \end{cases} \tag{5}$$

$$\begin{cases} \mathit{Master} = 2(W-1) + H \\ \mathit{Vertical} = H(W-1) - W(W+1)/2 + 1 \\ \mathit{Horizontal} = W(W-3)/2 + 1. \end{cases} \tag{6}$$

The number of CR commands issued at each step of the MSA process depends upon the number of hosts available. For instance, if two slave hosts are available, two CRs are issued at a time, as in state $S19$ of Figure 5, totalizing, at the end of the process, $(H-2)$ CR for each host. In the case that only one slave host is available, Din2-Master issues only one CR, corresponding to a row in the matrix, while the column is computed by Din2-Master itself – see (state $S20$). After sending CRs, Din2-Master waits for completion of the task (state $S21$) and, further, increments indices i and j and restarts the procedure until all the cells of the matrix are computed.

As soon as this procedure is finished, Din2-Master releases all Din2-Slaves (state $S20$), which, in turn, get idle and wait for commands from the Manager or other Din2 modules. Finally, a signal is sent to the Manager causing the end of the algorithm (state $S9$).

3 COMPUTATIONAL EXPERIMENTS AND RESULTS

Several experiments were done to evaluate the performance of the proposed parallelization methodology. Experiments focused on the number of sequences to be aligned and on 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 Server-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.

For all experiments, the data set used was constructed using protein data randomly extracted from the PDB – Protein Data Bank [5]. The length of sequences was properly adjusted for the experiments either by clipping sequences being too long, or by copying and pasting sequences being too short. For all experiments, results reported are the average of 10 independent runs, under the same conditions and with the same computers. No processes were running in the hosts other than those native of the operating system.

The first experiment is aimed at discovering how the number of sequences affects 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 a single 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 was arbitrarily fixed in 1 200 columns. Processing time was recorded for 2, 3 and 4 hosts. Figure 7 shows how the use of parallel computation speeds up the process, by using 3 and 4 hosts, relative to a basic 2-hosts system.

The next experiment investigated how the processing time is affected as the number of sequences per profile grows. Again, the length of profiles was arbitrarily fixed in 1 200 columns and the number of sequences in each profile varied between 20 and 1 000. The results for this experiment are shown in Figure 8, where N is the number of sequences per group to be aligned.

The next experiment investigated how the processing time was affected as the length of the profiles grow. For this experiment, the lengths of both profiles were changed and the number of sequences per profile was kept constant at 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 9,

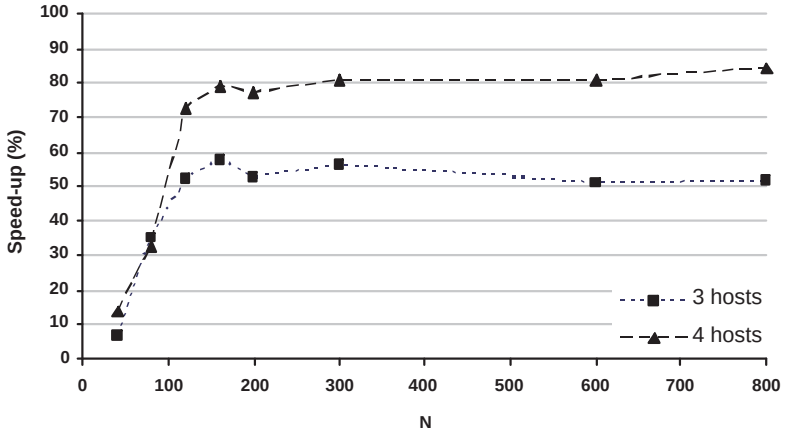

Fig. 7. Speedup of Pairwise-score, relative to a 2-hosts system: 4 hosts (upper line), 3 hosts (lower line)

for profile lengths in the range 500 to 2 500. This figure shows the time needed by the sequential process (ClustalW running sequentially), and the time needed by the parallel process simulated in a single host.

4 DISCUSSION AND CONCLUSIONS

In this paper we proposed a methodology for parallelizing a multiple sequence alignment algorithm using a network of PCs. The parallelization is achieved by exploiting a suitable partition of the dynamic programming matrix. This system has two levels of parallelization and self-adjusts to the number of available processing 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 (that is, the number os sequences, or length of profiles) increases, the difference in performance along the number of available hosts becomes more significant. A deeper analysis of the performance reveals that this improvement by using the parallel algorithm can achieve around 80 % of speed-up, when compared with the sequential processing. In fact, the speed-up does not grow monotonically as the problem size grows. It displays an asymptotic behavior, tending to a maximum gain of around 80 %. Future investigation will address this issue.

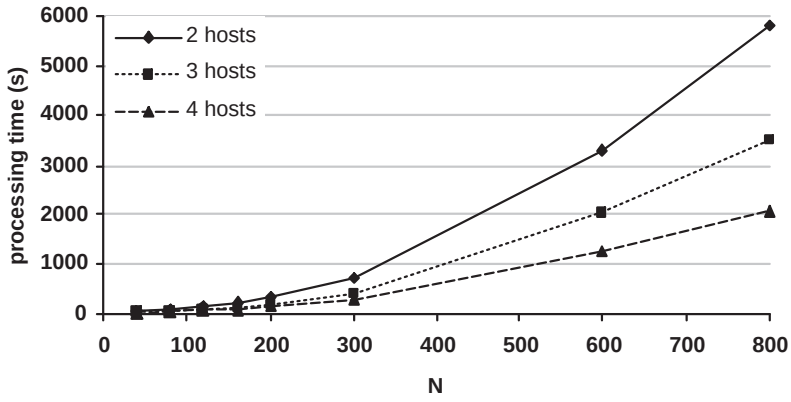

Fig. 8. Processing time of Din2 as function of the number of sequences per profile, using only the Master host (upper line) and using one or two hosts besides the Master (lower lines)

Curves shown in Figures 8 and 9 indicate that the processing time tends to grow polynomially as the size of the problem (number of sequences per profile and length of profiles) increases. This is a remarkable fact, since processing time can become prohibitive for large-scale problems, usually found in real-world problems.

No serious effort was done to optimize the running parameters of the system, since the focus of the research at this moment was the parallelization methodology

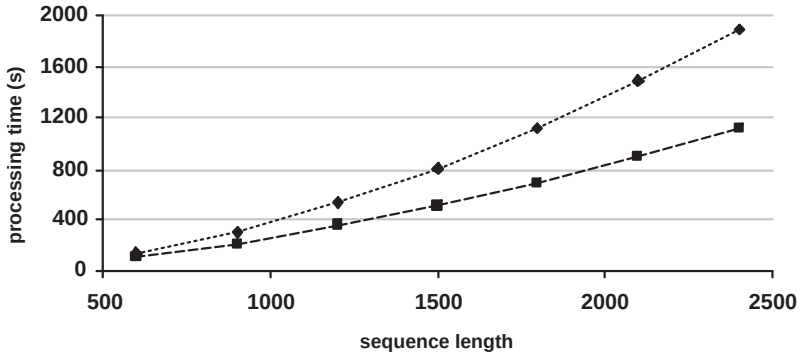

Fig. 9. Processing time as function of the length of profiles, running sequentially (upper line) and simulating parallel processing in a single host (lower line)

itself. Certainly, there are still some bottlenecks that affect performance, such as load balancing mechanisms or communication policy between processes. These and other parallelization issues will be considered in future improvements of the system, expecting better performances.

Recently, the methodology proposed here was successfully adapted to be used in a reconfigurable hardware [13]. The outstanding results encourage future work towards the use of hybrid networks, with desktops and reconfigurable logic for MSA. Such combination was proved to yield high throughoutput for bioinformatics applications [15].

Multiple sequence comparison by alignment is an important, and still open question, in computational biology. We believe that the proposed methodology is a useful contribution to the area of research, especially considering that cluster computing is a reality nowadays in many research laboratories. In the near future we intend to do more extensive experiments and put the system freely available for research purposes.

Acknowledgements

This work was partially supported by the Brazilian National Research Council – CNPq, under grants 506479/04-8 and 309262/07-0.

REFERENCES

[1] Abdeddaïm, S.—Morgenstern, B.: Speeding up the DIALIGN Multiple Alignment Program by Using the 'Greedy Alignment of BIOlogical Sequences LIBrary' (GABIOS-LIB). Lecture Notes in Computer Science, Vol. 2066, 2001, pp. 1–8.

[2] Akutsu, T.—Arimura, H.—Shimozono, S.: On Approximation Algorithms for Local Multiple Alignment. In: Proceedings of 4th International Conference on Computational Molecular Biology, 2000, pp. 1–7.

[3] Aluru, S.—Futamura, N.—Mehrotra, K.: Parallel Biological Sequence Comparison Using Prefix Computations. Journal of Parallel and Distributed Computing, Vol. 63, 2003, pp. 264–272.

[4] Alves, C. E. R.—Caceres, E.—Dehne, F.—Song, S. W.: A Parallel Wavefront Algorithm for Efficient Biological Sequence Comparison. Lecture Notes in Computer Science, Vol. 2668, 2003, pp. 249–258.

[5] Berman, H. M.—Westbrook, J.—Feng, Z.—Gilliland, G.—Bhat, T. N.—Weissig, H.—Shindyalov, I. N.—Bourne, P. E.: The Protein Data Bank. Nucleic Acids Research, Vol. 28, 2000, pp. 235–242.

[6] Dongarra, J.—Foster, I.—Fox, G. et al.: Sourcebook of Parallel Computing. Morgan Kaufmann, San Francisco 2003.

[7] Edmilson, E. W.—Core, N. G.—Saltz, J. H.—Smith, R. M.: Parallel Processing of Biological Sequence Comparison Algorithms. International Journal of Parallel Programming, Vol. 17, 1988, pp. 259–275.

[8] Henikoff, S.—Nenikoff, J. G.: Amino Acid Substitution Matrices from Protein Blocks. Proceedings of the National Academy of Sciences U.S.A., Vol. 89, 1992, pp. 10915–10919.

[9] Higgins, D. G.—Sharp, P. M.: CLUSTAL: A Package for Performing Multiple Sequence Alignments on a Microcomputer. Gene, Vol. 73, 1988, pp. 237–244.

[10] Just, W.: Computational Complexity of Multiple Sequence Alignment with SPscore. Journal of Computational Biology, Vol. 8, 2001, pp. 615–623.

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

[12] Leach, A. R.: Molecular Modelling: Principles and Applications. 2nd ed., Prentice-Hall, Dorset 2001.

[13] Lima, C. R. E.—Lopes, H. S.—Moroz, M. R.—Meneses, R. M.: Multiple Sequence Alignment Using Reconfigurable Computing. Lecture Notes in Computer Science, Vol. 4419, 2007, pp. 379–384.

[14] Lopes, H. S.—Moritz, G. L.: A Graph-Based Genetic Algorithm for the Multiple Sequence Alignment Problem. Lecture Notes in Artificial Intelligence, Vol. 4029, 2006, pp. 420–429.

[15] Lopes, H. S.—Lima, C. R. E.—Murata, N. J.: A Configware Approach for High-speed Parallel Analysis of Genomic Data. Journal of Circuits, Systems, and Computers, Vol. 16, 2007, pp. 1–15.

[16] Notredame, C.: Recent Progresses in Multiple Sequence Alignment: A Survey. Pharmacogenetics, Vol. 3, 2002, pp. 131–144.

[17] Saitou, N.—Nei, M.: The Neighbor-Joining Method: a New Method for Reconstructing Phylogenetc Trees. Molecular Biology and Evolution, Vol. 4, 1987, pp. 406–425.

[18] Smith, T. F.—Waterman, M. S.: Comparison of Bio-Sequences. Advances on Applied Mathematics, Vol. 2, 1981, pp. 482–489.

[19] Thompson, J. D.—Higgins, D. G.—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.

[20] Wang, L.—Jiang, T.: On The Complexity of Multiple Sequence Alignment. Journal of Computational Biology, Vol. 1, 1994, pp. 337–348.

[21] Zhang, M.—Fang, W.—Zhang, J.—Chi, Z.: MSAID: Multiple Sequence Alignment Based on a Measure of Information Discrepancy. Computational Biology and Chemistry, Vol. 29, 2000, pp. 175–181.

Heitor Silvério Lopes is Associate Professor at the Federal University of Technology Paraná (UTFPR), Curitiba, Brazil. He is the Head of Laboratory of Bioinformatics. His research interests include bioinformatics, evolutionary computation, computational intelligence and high-performance computing.

Carlos Raimundo Erig Lima is Adjunct Professor at the Federal University of Technology – Paraná (UTFPR), Curitiba, Brazil. His research interests include computational intelligence, bioinformatics and reconfigurable computing.

Guilherme Luiz Moritz graduated in electrical engineering from the Federal University of Technology Paraná (UTFPR) and currently works as development engineer.