American Journal of Bioinformatics 1 (1): 50-63, 2012
ISSN 1948-9862

Using Threads to Overcome Synchronization Delays in Parallel Multiple Progressive Alignment Algorithms

[1]Evandro A. Marucci, [1,2]Geraldo F.D. Zafalon, [1]Julio C. Momente,
[1]Alex R. Pinto, [2]Jose R.A. Amazonas, [3]Yang Shiyou, [2]Liria M. Sato and [1]Jose M. Machado
[1]Departamento de Ciencias de Computacao e Estatistica,
Sao Paulo State University, Sao Jose do Rio Preto, SP, Brazil
[2]Escola Politecnica, University of Sao Paulo, Sao Paulo, SP, Brazil
[3]College of Electrical Engineering, Zhejiang University, Hangzhou, 310027, China

Abstract: Problem statement: The parallelization of multiple progressive alignment algorithms is a difficult task. All known methods have strong bottlenecks resulting from synchronization delays. This is even more constraining in distributed memory systems, where message passing also delays the interprocess communication. Despite these drawbacks, parallel computing is becoming increasingly necessary to perform multiple sequence alignment. **Approach:** In this study, it is introduced a solution for parallelizing multiple progressive alignments in distributed memory systems that overcomes such delays. **Results:** The proposed approach uses threads to separate actual alignment from synchronization and communication. It also uses a different approach to schedule independent tasks. **Conclusion/Recommendations:** The approach was intensively tested, producing a performance remarkably better than a largely used algorithm. It is suggested that it can be applied to improve the performance of some multiple alignment tools, as CLUSTALW and MUSCLE.

Key words: Sequence alignment algorithms, Hidden Markov Models (HMM), performance remarkably, progressive approach, computational complexity, represent genetic

INTRODUCTION

An important field of bioinformatics is the analysis of amino acid sequences, which is the major form to produce data about evolutionary processes. Geneticists use protein alignments to gather information about genetic relationships among species or individuals. One method that is extensively applied is the progressive alignment of multiple sequences. Several multiple sequence alignment algorithms have been proposed but they present, usually, a high computational complexity (Rashid *et al*., 2009). Among these algorithms a major category includes the progressive approach. Although the solutions achieved by a progressive approach are not optimal, its computational cost is lower than the cost from exact methods. Indeed, exact methods are not viable when the number of sequences to be aligned reaches the hundreds.

Approximate methods, such as the progressive ones, are crucial since the amount of amino acid sequences available for comparison becomes larger and larger. Globally available sequence databases allow for accurate multiple alignments, requiring the development of faster alignment tools using high performance computing. Several such tools have been developed to support the genetic researches. However, the parallelism provided by them either are restricted to expensive shared-memory multiprocessors or does not consider fundamental aspects of parallel computing.

A common problem found in such tools is that they simply replicate processes to reach parallelism. This works well for certain applications, but fails miserably for multiple alignment in distributed memory systems. Since progressive multiple alignments demand frequent communication and synchronization, the use of simple replication creates a huge latency. This latency makes many hosts to remain idle waiting for data, what is worsened by poor task scheduling mechanisms. An often used algorithm, presented by (Luo *et al*., 2005) suffers from this problem. In that algorithm, a host may remain idle when its input data, stored in another host, cannot be sent immediately.

In this study a new parallel algorithm, aimed for distributed memory systems, is presented. This new algorithm circumvents the problems created by synchronization, communication and poor scheduling through the use of multihtreaded programming.

Corresponding Author: Geraldo F.D. Zafalon, Departamento de Ciencias de Computacao e Estatística, Sao Paulo State University, Sao Jose do Rio Preto, Sao Paulo, Brazil

Separated threads are created to deal with alignment processing and with data communication. The task scheduling is performed by a priority algorithm based on data location. This approach was tested and its results show a noticeable improvement when compared with Luo's algorithm, as will be seen in results.

During the remaining sections the reader finds a description of related works, followed by a detailed discussion of the proposed algorithm and the obtained results. The discussion is concerned with the experimental evaluation of this algorithm. Finally, some conclusions and prospective directions are presented in the last section.

Current research in multiple sequence alignment algorithms: The progressive alignment consists in the alignment of pairs of sequences or pairs of alignments in a progressive way, through a previously constructed phylogenetic tree, which is a binary tree where each leaf node is a single genomic sequence, while intermediary and root nodes represent genetic relationships between families of sequences. The parallelization of alignment algorithms may be made through two distinct approaches, with different granularity: the first one considers each tree node as a single problem and the data in each node is decomposed and distributed among parallel processing elements; the second one parallelizes the tree nodes, managing each node as an atomic block.

The major concern with such approaches is the allocation of new tasks to idle hosts. This allocation must consider the dependence from primitive tasks, as well as the load balance and the communication overheads. This is a hard problem and has strong influence in the implementation performance.

Both approaches present advantages and drawbacks. For instance, strategies acting in the node level will produce a finer granularity, increasing communication but reducing the latency from data dependence, since a single tree node is dependent only from its children. Some strategies following this scheme were proposed in the literature, including (Lopes and Moritz, 2005; Luo *et al*., 2005). They were incorporated in alignment software tools, such as ClustalW-MPI (Li, 2003) and Muscle-SMP (Deng *et al*., 2006). These strategies usually keep a single process in each processor, which is responsible for both processing and data communication. This approach creates significant communication latency, since a process cannot communicate and work at the same time. Although priority lists can be used to minimize the latency (Luo *et al*., 2005), there is not a known approach which completely eliminates it.

Other examples for alignment approaches include the use of distributed architectures (Ebedes and Datta, 2004; Strumpen, 1995), simulated annealing (Ishikawa *et al*., 1993; Zola *et al*., 2006), or Markov chain decomposition (Bhandarkar *et al*., 1998; Keibler *et al*., 2007).

(Strumpen, 1995) presented a geographically distributed parallel computing approach over the internet (MIMD) which improves the biological sequence analysis. He showed that sequence analysis might be done over several countries with low requirements on communication bandwidth, achieving better results than those executing in a sequential machine. (Du *et al*., 2005) used MIMD architecture to perform the reconstruction of large phylogenetic trees, dividing the datasets into smaller subproblems and distributing the computation load over multiple processors so that each processor constructs subtrees on each subproblem within a batch in parallel. It finally collects the resulting trees and merges them into a supertree.

(Ebedes and Datta, 2004) showed a variation of clustalW distributed memory implementation. It is showed that this approach results in a significant speedup. The experimental results obtained by Ebedes showed a speedup of over 5.5 on six processors for most inputs.

(Zola *et al*., 2006) proposed a method that simultaneously performs multiple sequence alignment and phylogenetic tree inference for large input data sets. It is described a parallel implementation of the method that utilizes simulated annealing metaheuristic to find locally optimal phylogenetic trees in reasonable time. In a different approach (Ishikawa *et al*., 1993) used simulated annealing heuristics in a parallel system to solve the problem of multiple sequence alignment. Basically, they solve the problem of multiple sequence alignments calculating the annealing temperature in parallel, in order to improve the simulated annealing algorithm result at a reasonable execution time.

Another application of parallelism in bioinformatics can be seen in (Bhandarkar *et al*., 1998), where it is presented practical experience with the design and implementation of a suite of parallel algorithms for chromosome reconstruction via ordering of DNA sequences. It uses parallel simulated annealing algorithms for physical mapping of the sequences and they are based on Markov chain decomposition. A different application of Markov chains was suggested by (Keibler *et al*., 2007), who presented two approaches to decode human chromosomes without turning the memory usage and running time prohibitive. They developed the Treeterbi and Parallel Treeterbi methods and implemented them in the TWINSCAN/ NSCAN gene-prediction system. Both methods use generalized Hidden Markov Models (HMM).

More different approaches includes the inversion of the conventional order for progressive alignment, as

proposed by (Kleinjung *et al*., 2002), use of decentralized cache (Trystram and Zola, 2005), protein foldings (Palu *et al*., 2007; Arcuri *et al*. 2010).

MATERIALS AND METHODS

The multithreaded approach: In distributed memory systems each host usually process one task at time. While this avoids overcharging the host's memory and processor, it opens the door for poor cpu utilization, depending on the application. As briefly discussed during the introduction, the progressive multiple alignment technique demands frequent synchronization and data exchange. These demands turn this technique somewhat inefficient when running on distributed systems.

The inefficiency created when performing the alignment in parallel comes basically from two conditions: Need for data synchronization and poor task scheduling. The data synchronization is characteristic from the progressive alignment, where alignments produced by one node are input data for its parent node in the phylogenetic tree. This problem cannot be eliminated but can be minimized through improved communication and task allocation. The poor scheduling is originated by the use of simple bag-of-tasks scheduling, which implies in severe idle times waiting for remote data. This problem is also overcome by an improved allocation mechanism.

The multithreaded algorithm proposed here reduces the time needed for task synchronization through the separation between alignment processes and data exchange processes. It also improves the scheduling through a task allocation policy that priorizes the allocation accordingly the data location. Combining both modifications it is possible to improve the parallel progressive alignment by a large factor. These modifications are detailed in the next section.

An empirical evaluation of these improvements can be made considering that the total time for alignment is composed by three components: the time to align pairs of sequences (τ_{align}), the time to transfer a pair of sequences to another host ($\tau_{transfer}$) and the time that host must wait for the sending hosts to become available to transmit (τ_{synch}). This is represented in the following Eq. 1:

$$ExecTime = \tau align + \tau transfer + \tau synch \quad (1)$$

It is known that τ_{align} cannot be affected by the scheduling policy. Therefore, the multithreaded algorithm has its performance related to the other two parcels.

First, $\tau_{transfer}$ is the sum of each individual data transfer between hosts. The number of transfers is, at most, equal to the number of edges in the phylogenetic tree. The size of each transfer is proportional to the size of the input sequences in the sending node, being larger in the upper portion of the tree. Since the scheduling policy proposed here avoids these transfers, it is conceivable that for each node that is not a leaf node in the tree at least half of the transfers can be avoided. This is true in the upper portion of the tree, where the algorithm may use fewer nodes to finish the alignment.

Second, τ_{synch} is the sum of the delay times to start a transfer that appears because the sending host is busy doing some computation. With the multithreaded approach this time can be neglected since the computing thread does not preclude the communication thread to begin the transmission.

It is impossible to have an exact measure of the number of data transfers and the time spent on them. However, it is conceivable that the reductions introduced by the scheduling policy and the multithreading are important. This is even more admissible for alignments involving larger number of sequences.

Threads description: The core of this new algorithm is composed by two threads. One is in charge of actual processing, executing the alignment procedures over its input data. The other is in charge of data exchange, freeing the alignment thread from dealing with interprocess communication and synchronization.

The basic structure for this approach is shown in Fig. 1. From that it is possible to see that threads running in one host communicate to each other through the local memory. Remote hosts communicate through the communication threads residing on each host.

The use of separate threads for data munching and data exchange allows every host to keep processing a given task while another task is waiting for data. Similarly, a host that receives a request for data (already available in the host memory) might send it back immediately, even if it is processing a running task.

As a consequence of the data exchange optimization, whenever a processor becomes idle, any active task may be scheduled, independently from the locality of its input data. If the task requires data that is not locally stored, a message is sent by this task to the host keeping the data, through the involved communication threads, ordering the immediate data transfer to the requiring task.

Although the improvements made by the separation of threads are significant, the algorithm can be further improved through the use of a scheduling policy more efficient than the conventional bag-of-tasks. The goal here is to reduce the amount of data exchanges that are needed in order to process the whole alignment.

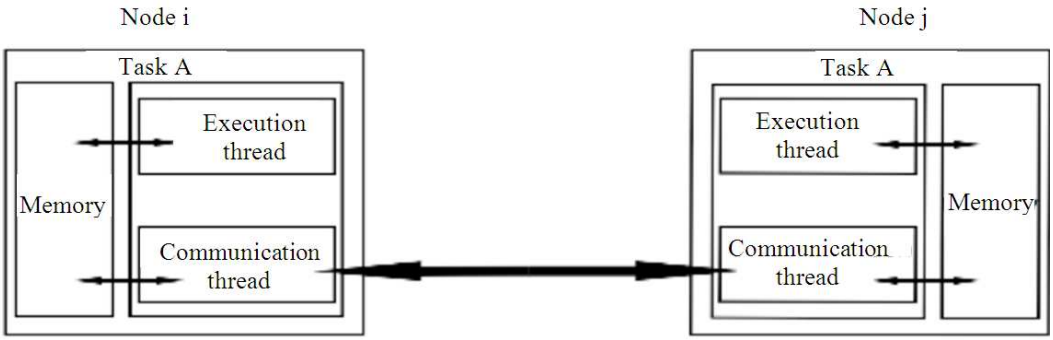

Fig. 1: Functional diagram for threads interaction

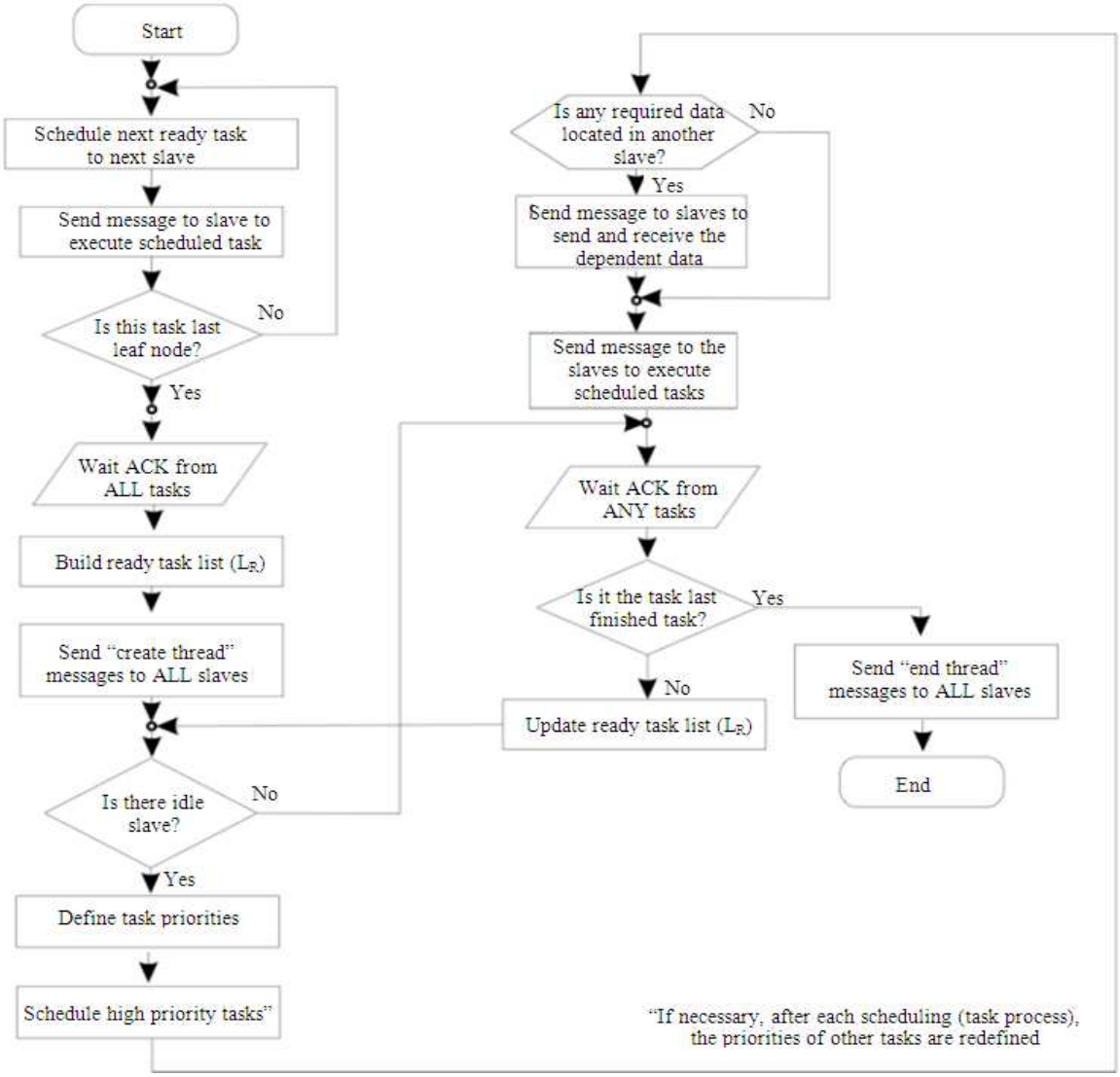

Fig. 2: Flowchart of the master process algorithm of the strategy based on the multithreaded approach

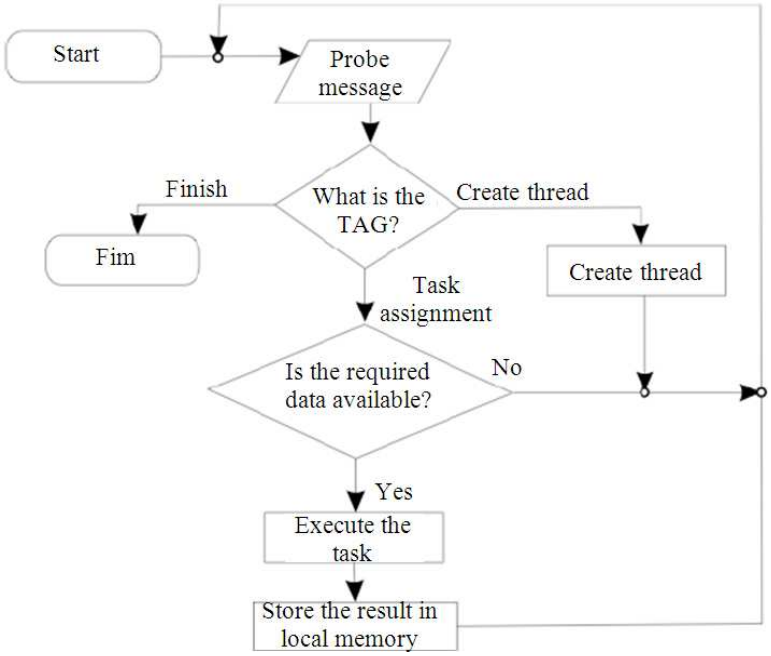

Fig. 3: Flowchart of the main process algorithm of the strategy based on the multithreaded approach

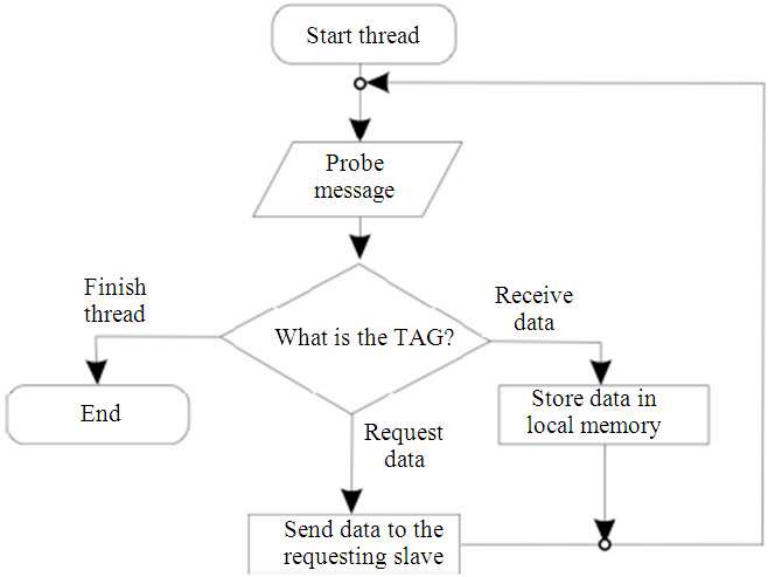

Fig. 4: Flowchart of the communication thread algorithm in the slave processes of the strategy based on the multithreaded approach

With the conventional bag-of-tasks policy, if a given processor becomes idle, any node (in the phylogenetic tree) that is already ready to processing is allocated to it. This allocation may result in zero, one or two data exchanges, depending on the place where the children of this node were executed.

In order to minimize the number of data exchanges in a given scheduling instant, three priority scheduling levels are defined. These scheduling levels of a given task depend upon the locality of the data needed for its execution. Each ready task has its priority defined as follows:

- High, when the data produced by its children are located in the same idle host
- Mean, when either only one child is in the same host or both children are located in different hosts, but at least one of them is located in an idle host
- Low, when both children are located in busy hosts

The pairing task-host is performed after priority classification, allocating the pairs with the highest priority. This procedure stops either when all hosts become busy or the tasks queue becomes empty. After each allocation the priorities are redefined, since the allocated host would be the best host for a different task. In such case, that task must receive a new priority.

Algorithm implementation: An implementation of the multithreaded algorithm was made using the Muscle alignment tool as its basic platform. Several changes were performed in order to verify several distinct possibilities of parallelization. The parallelism used the bag-of-tasks model, where a master process is in charge of distributing tasks and controlling communication. The actual processing tasks are named as slaves, being composed by the alignment and communication threads. The flowcharts presented in Fig. 2-4 represent this implementation.

Figure 2 depicts the flowchart of the master process, where the major points are the task allocation procedure (lower left) and the task preparation/execution procedure (upper right). Initially, it allocates the tasks to the slaves until all nodes are busy. After that, the master waits for each host to signal that they are ready to execute and then starts to create threads to execute tasks on the slave nodes as soon as a node becomes idle. When a node becomes idle the master process schedules the task with the highest priority to it. If all required data is located in that host, the master process sends a message to the slave to execute the task, otherwise the master sends a message to the involved hosts to move the dependent data and then the task execution is performed. After this step, the master waits the acknowledgment of each task completion, repeating the procedure until every alignment is performed.

Figure 3 and 4 show the flowcharts corresponding, respectively, to the alignment thread (named as main process) algorithm and the communication thread algorithm in the slave processes that run on client hosts.

The major aspects of the main process (alignment thread, Fig. 3) are the execution and data verification phases. During production the main process receives a message from the master indicating a new task or a finish signal. If it is a new task it checks the input data location, asking for data migration if necessary. If the data is not available it will wait for the communication thread to provide the data, performing the alignment as soon as all data become available.

The communication thread acts in two different ways. If it receives a request message it sends the data from its local memory to the host that is demanding it. If it receives migrated data it stores it in the local memory, signaling the alignment thread that the data is already available.

RESULTS

The evaluation of the multithreaded approach was performed through experimental benchmarking. The alignment was conducted over sequences extracted from the NCBI (The database is found at www.ncbi.nlm.nih.gov) (Sherry *et al*., 2001). During the benchmarks the variable parameters in the sequences were the total number of aligned sequences and the number of residues in each sequence (that is, the sequence length).

The tests were executed in a Beowulf cluster composed by 16 nodes, each with an Intel• Pentium• 4 CPU of 2.80GHz and 1 Gbyte of memory. The machines were connected through a dedicated switch of 1 Gbit/s.

In order to evaluate the algorithm the tests focused in its speedup, scalability and dependence on the phylogenetic tree. The results were compared against Luo's algorithm, since it is the only one that follows the dynamic approach for tasks assignment through clusters nodes. The first part of this section presents the results for Luo's algorithm implemented in the Muscle tool, followed by speedup and scalability tests of the multihtreaded proposal. The section ends with the investigation about the influence of phylogenetic tree structure over execution time.

Results for Luo's algorithm in muscle: In order to have a broader view of Luo'ed in the original study due to two major reasons. First they are implemented in different alignment tools (ClustalW and Muscle), second they were run over different data sets, with different number of residues and sequences (since the original set could not be used here). One important remark is that, the data set used here is much larger than the one used in Luo's original study, providing a deeper understanding about scalability.

The first test shows the performance of the Luo's algorithm. Fig. 5 shows the plot of running time for four distinct inputs. These inputs contain 500, 1000, 2000 and 4000 sequences, with approximately 1000 residues each one.

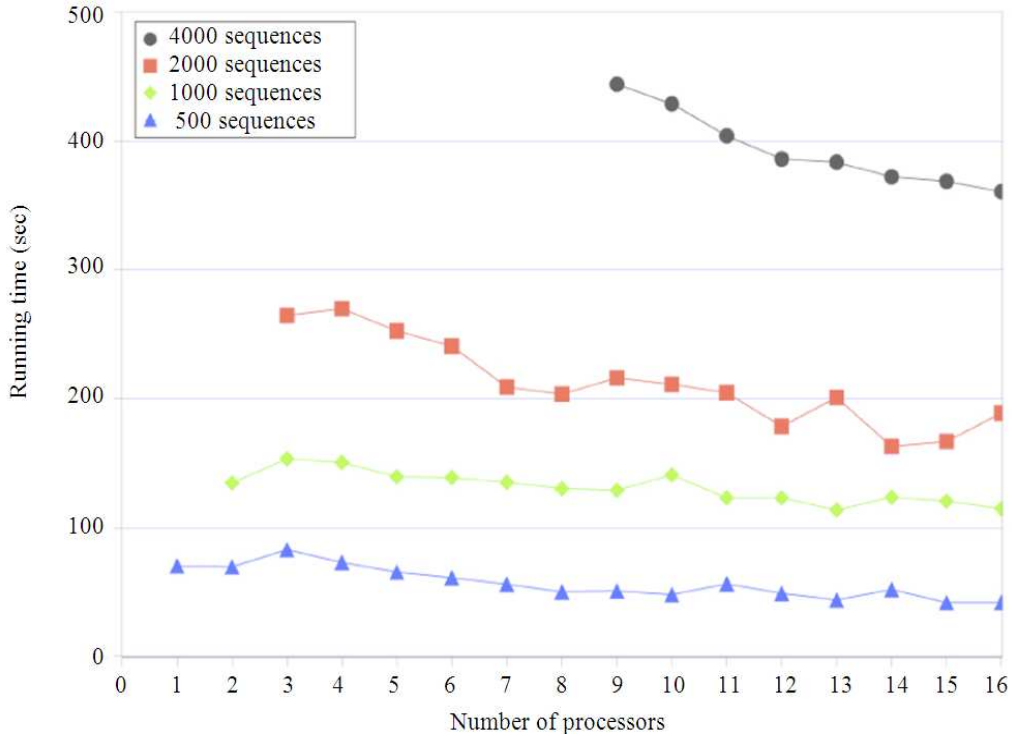

Fig. 5: Execution time of Luo's strategy for progressive alignment

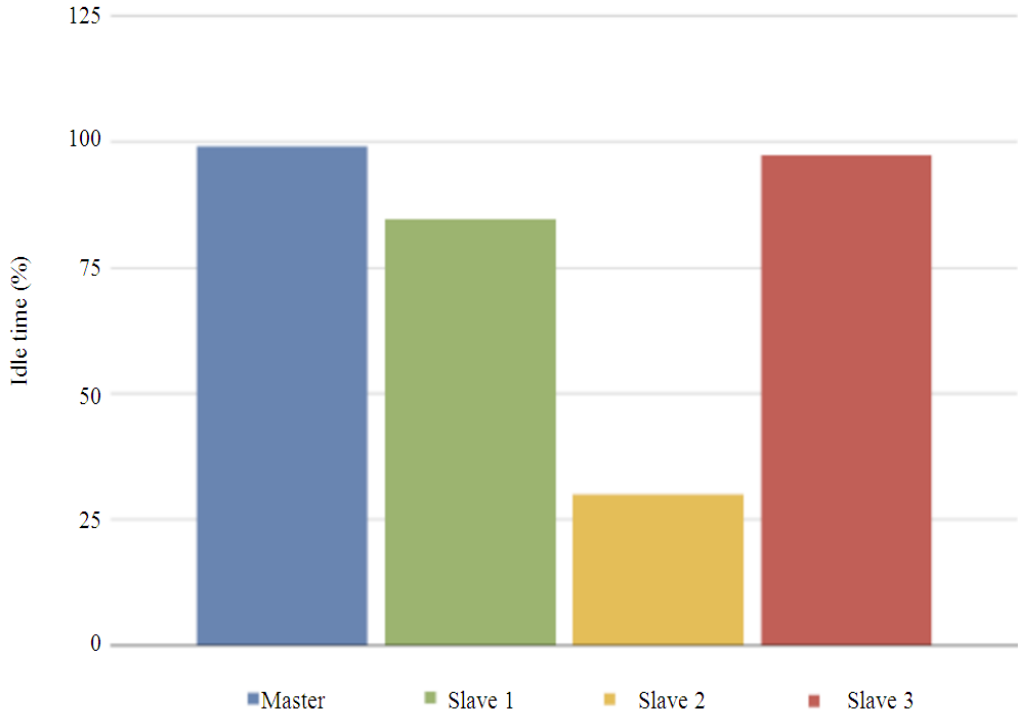

Fig. 6: Idle CPU times for Luo's algorihtm (500 sequences and 3 slaves)

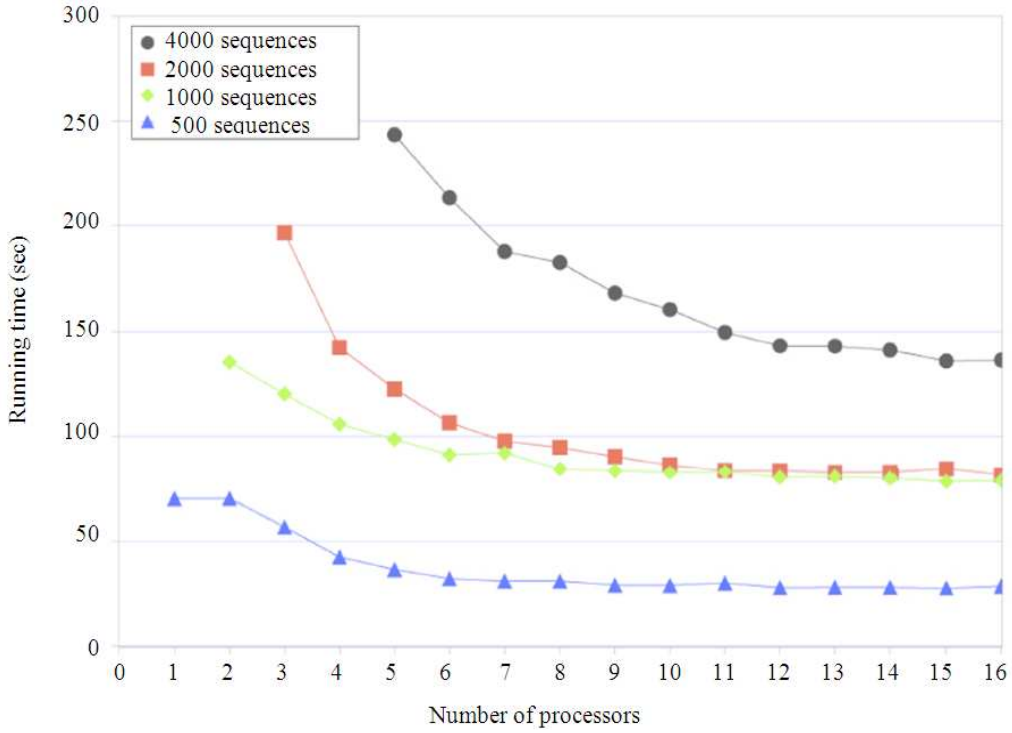

Fig. 7: Execution time of the multithreaded strategy

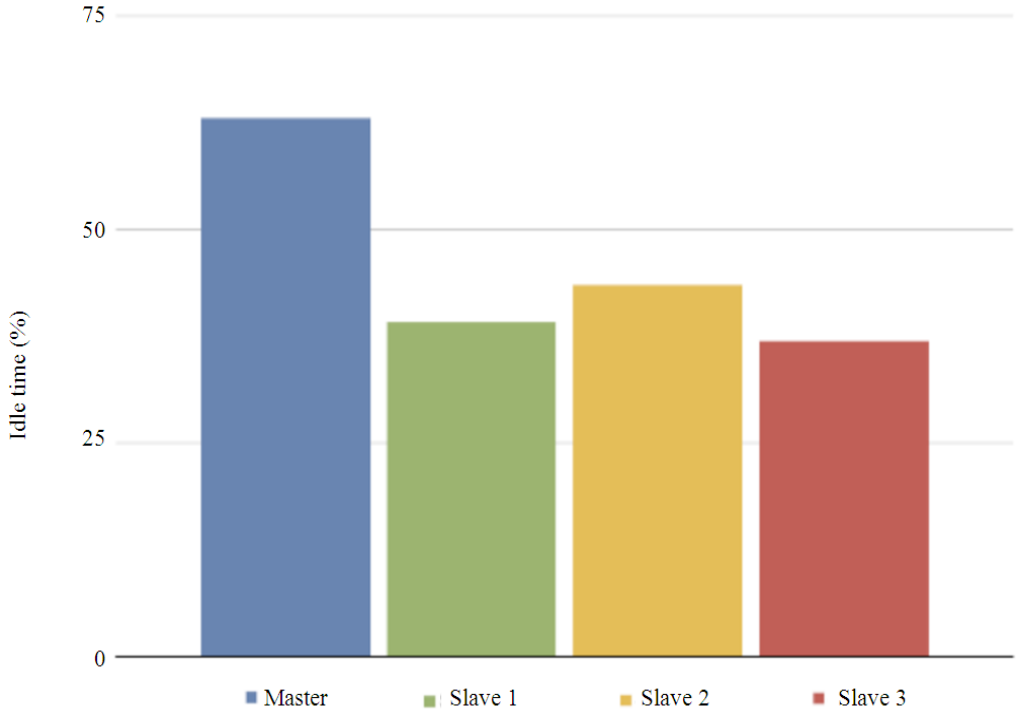

Fig. 8: Idle CPU times for the multithreaded algorithm (500 sequences and 3 slaves)

From that plot two remarks must be made:

- The size of the problem strongly constrains the algorithm application in smaller systems due to the restrictions on available memory
- The speedup is under 2 for any number of nodes, which is a very small speedup

Although the latter remark would indicate that the parallelization is not useful, the former tells that it must be used in order to solve larger problems.

These poor results come from the inefficient scheduling and load balance policies in Luo's algorithm. The plot presented in Fig. 6 shows the load measured in a test with 500 sequences running on four nodes. The bars in that plot represent how much time (in percentage) each node remained idle, waiting for data or synchronization. From that it is easy to notice that the load is unbalanced, with only one node working during most of the time. This means that the demands on memory and CPU cycles are not proportional over the system, implying in worst resources usage.

Results for multithreaded algorithm in muscle: In order to compare the algorithms, the same tests were applied to an implementation of Muscle using the multithreaded approach. The measured times are presented in Fig. 7. The execution times achieved with the proposed algorithm are clearly superior than those achieved by Luo's algorithm. This is evident in three aspects:

- The proposed algorithm is less dependent of the problem size, being able to execute larger problems with fewer nodes than Luo's algorithm
- The speedup is better than Luo's, being above 2 in almost all cases
- The absolute time needed for each execution is lower than the time needed by Luo's, sometimes using less than half of that time

These aspects can be exemplified comparing both algorithms in a given case. For example, Luo's algorithm demanded 443 sec to align 4000 sequences in a 9 nodes configuration, while the multithreaded algorithm performed the same alignment in 243 sec with only 5 nodes (and spent 168 sec using the same 9 nodes).

The chart in Fig. 8 shows the idle times (in percentage) achieved with the multithreaded algorithm for the case with 500 sequences running on four nodes. The time spent by the communication threads is considered as idle time too, since it does not correspond to data munching. From this figure it is easy to see that the nodes running the slave threads spend most of the time processing now (idle times lower than 40%), with only the master node remaining idle for more than 50% of the time.

The performance improvement is a direct consequence of the reduction in the communication costs and the better scheduling policy introduced by thread utilization. This also induced a better memory usage, which enabled to run larger problems with fewer nodes.

Speedup evaluation: As already stated, the speedup alone is not an important issue for this type of application. It is important to reduce the time spent processing, but it is more important to be able to solve larger problems, which are constrained by memory usage. Besides this consideration, the speedup provided by the multithreaded algorithm is much better than the speedup provided by Luo's algorithm. This section discusses the speedup for the alignment procedure.

Prior to this analysis it must be said that results published in Luo's study cannot be compared to these. The reasons for this are two. First, Luo executed its implementation in ClustalW and if one compares sequential results achieved there with those achieved here will notice that the ClustalW version is remarkably slower than the Muscle version (about ten times slower). Second, the tests in Luo's article were conducted over very small datasets, with the largest one aligning 80 sequences with less than 400 residues each, while the smallest test presented here aligned 500 sequences with about 1000 residues each.

The size of the dataset used here created some approximations on the evaluation of the speedup. The data to determine the speedup is the same that appears on Fig. 6 and 8. The speedups presented here were estimated using linear regression to determine the execution time for a single node when it was unavailable. Linear regression is a lower-bound approximation since it is known that the execution time curve follows in an exponential for the first few nodes in the system. Fig. 9 presents the speedups calculated this way for Luo's algorithm and Fig. 10 presents the speedups for the multithreaded algorithm.

Comparing both figures it is easy to see that the multithreaded approach shows speedups at least 50% higher than in Luo's algorithm. For larger sets of sequences the difference is even higher. This difference, as stated before, represents the ability to solve larger problems introduced by the multithreaded algorithm.

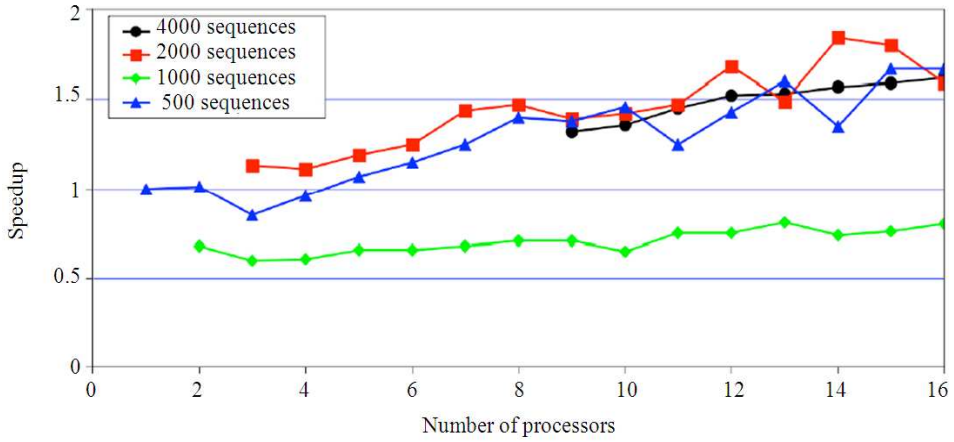

Fig. 9: Speedup curves achieved with Luo's algorithm

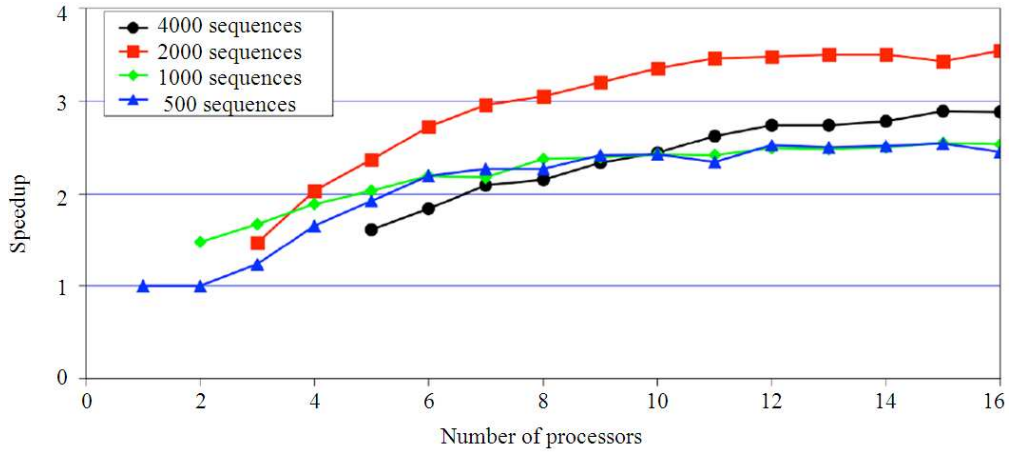

Fig. 10: Speedup curves achieved with multithreaded algorithm

DISCUSSION

Evaluating the variation in the sequence size: The previous tests were concerned with the evaluation of the alignment procedure for different number of sequences. Another important aspect in the biological problem is related to the size of each sequence, that is, the number of residues in the sequence. It is expected that longer sequences demand more time to be aligned, since there are more symbols to be evaluated. The following test shows how the multithreaded algorithm behaves when the number of residues in each sequence is modified.

The tests involved cases with 500 sequences with sizes of 1000, 2000, 3000, 4000 and 5000 residues, also retrieved from the NCBI database (Sherry *et al*., 2001). The measured speedup for these cases are presented in Fig. 11. It is evident that for longer sequences the speedup curves are more scalable than for shorter sequences, what is not surprising.

The behavior of the multithreaded algorithm shows that its efficiency scales with the growth in the problem size. This is true either when the number of sequences or their size increases. Therefore, the multithreaded approach is an interesting improvement for progressive alignments.

The phylogenetic tree and the algorithm's scheduling efficiency: It has been demonstrated that the multithreaded approach improves the alignment efficiency. It should be clear also that its major constraint is the data availability during the process of task assignment. Since each task represents one node in the phylogenetic tree and its input data comes from its child nodes, it is expected that the tree topology should influence the algorithm efficiency. During this section this impact is evaluated

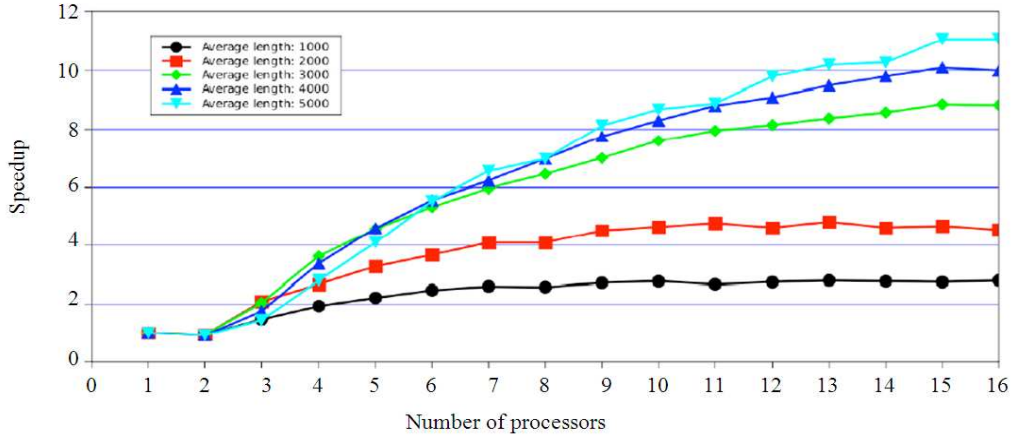

Fig. 11: Speedups achieved with sequence size variation (500 sequences)

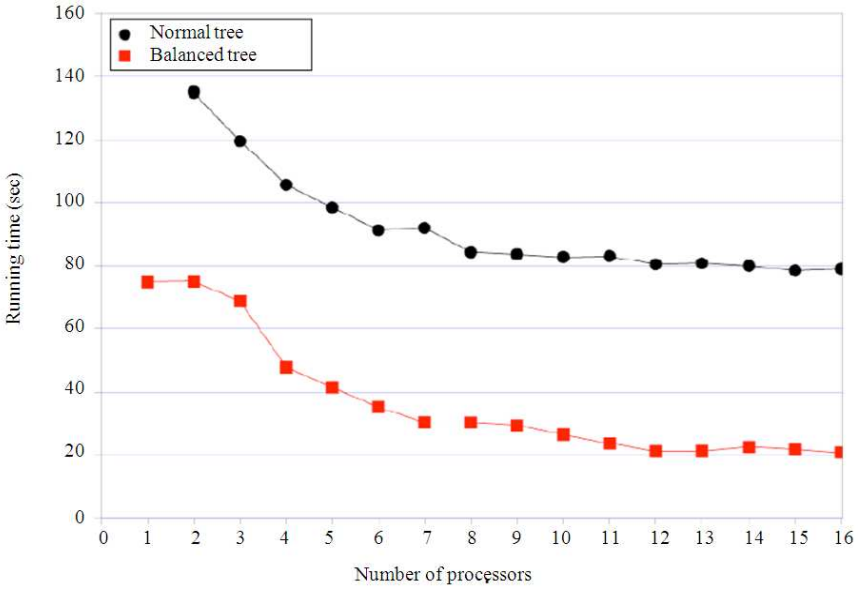

Fig. 12: Running times for the multithreaded algorithm using a balanced tree and an unbalanced tree (1000 sequences)

The phylogenetic tree is produced during the initial phases of the progressive alignment. This encompass a pairwise alignment and the use of the score matrix to generate a binary tree, which is usually incomplete. One common approach to determine the phylogenetic tree is the UPGMA method (Larkin *et al*., 2007), which was modified to create a balanced binary tree from the ordinary one. The evaluation was restricted to the comparison of the ordinary and the balanced trees since the balanced tree represents the optimal case for data dependency (Wallace *et al*., 2004).

It must be noted here that this study cannot be applied in a straightforward manner. The problem here is that the phylogenetic tree has a biological meaning and modifications on its topology could produce results that may not be reliable. In other words, if the arrangement of the tree is modified, the sequence similarities will not imply on nodes vicinity. This may lead to alignments that are only locally optimal.

Figure 12 shows the execution times for both, balanced and unbalanced, trees for the case with 1000 sequences. The performance of the multithreaded algorithm is always better than the unbalanced tree execution. This result was expected since balanced trees imply in less time spent waiting for synchronization and data availability.

Table 1: Parallelism degree using balanced and unbalanced trees, for the input with 1000 sequences

Number of slaves	Parallelism Balanced	degree-pd (N) Not balanced
1	1.00	1.00
2	2.00	1.75
3	2.99	2.35
4	3.98	2.78
5	4.95	3.15
6	5.91	3.42
7	6.89	3.61
8	7.87	3.78
9	8.76	3.89
10	9.70	4.00
11	10.63	4.08
12	11.62	4.13
13	12.49	4.18
14	13.32	4.23
15	14.27	4.27

The reduction in the synchronization may be numerically determined by a metric equivalent to the system usage. This metric is defined here as the parallelism degree of the tree. The parallelism degree for *N* processors (pd (N)) is determined as follows Eq. 2:

$$pd(N) = \frac{total - number\_of\_alignment\_nodes}{independent\_blocks} \quad (2)$$

where, *independent_blocks* is the number of blocks containing at most *N* node alignments that may be executed in parallel. The number of alignments is known from the phylogenetic tree. The number of independent blocks is empirically determined during the progressive alignment, using the list of ready nodes as reference.

This metric shows how scalable the alignment procedure is. It is easy to notice from Table 1 that when the tree is balanced the problem presents a better scalability. The obvious conclusion is that one should build balanced phylogenetic trees. This is not the usual procedure since the biological meaning must be preserved, leading to unbalanced trees in most situations.

An indirect consequence of balanced phylogenetic trees is that intermediate nodes in the alignment demand less data than nodes in the top levels of the tree. Since balanced trees have more nodes in the bottom levels, these nodes demand less data and can be finished faster than upper nodes. This speeds up the execution and reduces the demand for memory, allowing its use in smaller clusters. Indeed, it was possible to perform the alignment for 1000 sequences with only one host using a balanced tree against two nodes for the unbalanced one.

CONCLUSION

In this study it was presented and evaluated a multithreaded algorithm to perform multiple sequence alignment in a progressive approach. The proposed algorithm uses threads to eliminate the latency introduced by synchronization and communication constraints to process single tasks. It is aimed for distributed memory systems, such as Beowulf clusters, which have an intensive use in genomic research nowadays.

Another important contribution is the scheduling policy. The scheduling and task allocation follows a procedure that prioritizes pairings task-host where the need of data movimentation is minimal. This improvesthe performance since if the task allocated to a given host demands for data already in that host, then no time is spent on data migration (or interprocess communication).

The use of separate threads for communication and processing, besides the task allocation procedure based on data locality, allowed for performance improvements in two ways:

- Achieving faster execution through the reduction in the time needed for processes synchronization due to data dependencies
- Reducing the need for data migration through an efficient policy for tasks scheduling, based on the amount of data that is ready for consumption

These characteristics enabled a better use of CPU cycles and also a better memory usage. Using less CPU cycles and less memory enabled, at the end, a reduction in the amount of resources needed to run larger problems.

Another relevant result presented here is the algorithm scalability. It was shown that the multithreaded algorithm is scalable to the number of sequences as well as to the size of the sequences. This means that it may be applied for problems and systems of several sizes.

One important remark here is these results were achieved using larger datasets than those presented in several proposed algorithms. Despite that, the execution times were comparable to their execution times. This indicates that the algorithm is actually very fast.

In a different direction, it was also evaluated the impact of the phylogenetic tree topology over the algorithm performance. The algorithm has a remarkable improvement if the tree is balanced, which is not simple to achieve since the nodes in the tree have biological meanings, restricting their movimentation to reach balance.

From the results achieved during this work it is possible to devise some future steps towards a very efficient parallel multiple sequence alignment tool. These steps include:

- Investigating the use of B-trees to store the phylogenetic tree, since B-trees are implicitly balanced, aiming to evaluate the preservation of its biological meaning
- Investigating strategies to improve speedup, even for a larger number of hosts
- Implementing a ClustalW-Multithreaded version, by inserting the multithreaded approach into ClustalW
- Investigating the possibility of using a finer granularity (parallelize the alignment of single nodes) whenever the number of ready nodes is lower than the number of idle hosts

These aspects should improve even further the performance achieved with the multithreaded algorithm. As presented here, using threads is viable and helpful. It enables solving multiple alignments in small and much less expensive, clusters, being, therefore, an interesting contribution to this field.

ACKNOWLEDGEMENT

This study was partially supported by the Sao Paulo State Research Foundation-FAPESP (Brazil) under Grant No. 06/59592-0.

REFERENCES

Rashid, N.A.A., R. Abdullah and R.E.M. Al-Khatib, 2009. A survey of compute intensive algorithms for ribo nucleic acids structural detection. J. Comput. Sci., 5: 680-689. DOI: 10.3844/jcssp.2009.680.689

Arcuri, H.A., G.F.D. Zafalon, E.A. Marucci, C.E. Bonalumi and N.J.F.D. Silveria *et al*., 2010. SKPDB: A structural database of shikimate pathway enzymes. BMC Bioinform., 11: 12-12. DOI: 10.1186/1471-2105-11-12

Bhandarkar, S.M., S.A. Machaka, S.S. Chirravuri and J. Arnold, 1998. Parallel computing for chromosome reconstruction via ordering of DNA sequences (1998). Parallel Comput., 24: 1177-1204. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.84.8250

Deng, X., E. Li, J. Shan and W. Chen, 2006. Parallel implementation and performance characterization of muscle. Proceedings of the 20th International Parallel and Distributed Processing Symposium Apr. 25-29, IEEE Xplore Press, pp: 1-7. DOI: 10.1109/IPDPS.2006.1639616

Du, Z., F. Lin and U.W. Roshan, 2005. Reconstruction of large phylogenetic trees: A parallel approach. Computational Biol. Chem., 29: 273-280. DOI: 10.1016/j.compbiolchem.2005.06.003

Ebedes, J. and A. Datta, 2004. Multiple sequence alignment in parallel on a workstation cluster. Bioinformatics, 20: 1193-1195. DOI: 10.1093/bioinformatics/bth055

Ishikawa, M., T. Toya, M. Hoshida, K. Nitta and A. Ogiwara *et al*., 1993. Multiple sequence alignment by parallel simulated annealing. Comput. Applied Biosci., 9: 267-273. DOI: 10.1093/bioinformatics/9.3.267

Keibler, E., M. Arumugam and M.R. Brent, 2007. The treeterbi and parallel treeterbi algorithms: Efficient, optimal decoding for ordinary, generalized and pair hmms. Bioinformatics, 23: 545-554. DOI: 10.1093/bioinformatics/btl659

Kleinjung, J., N. Douglas and J. Heringa, 2002. Parallelized multiple alignment. Bioinformatics, 18: 1270-1271. DOI: 10.1093/bioinformatics/18.9.1270

Larkin, M.A., G. Blackshields, N.P. Brown, R. Chenna and P.A. McGettigan *et al.*, 2007. Clustal w and clustal x version 2.0. Bioinformatics, 23: 2947-2948. DOI: 10.1093/bioinformatics/btm404

Li, K.B., 2003. ClustalW-MPI: ClustalW analysis using distributed and parallel computing. Bioinformatics, 19: 1585-1586. DOI: 10.1093/bioinformatics/btg192

Lopes, H.L. and G.L. Moritz, 2005. A distributed approach for a multiple sequence alignment algorithm using a parallel virtual machine. Proceedings of the 27th Annual International Conference of the Engineering in Medicine and Biology Society, Jan. 17-18, IEEE Xplore Press, Shanghai, pp: 2843-2846. DOI: 10.1109/IEMBS.2005.1617066

Luo, J., I. Ahmad, M. Ahmed and R. Paul, 2005. Parallel multiple sequence alignment with dynamic scheduling. Proceedings of the International Conference on Information Technology: Coding and Computing, Apr. 4-6, IEEE Xplor Press, pp: 8-13. DOI: 10.1109/ITCC.2005.223

Palu, A.D., A. Dovier and E. Pontelli, 2007. A constraint solver for discrete lattices, its parallelization and application to protein structure prediction. Software: Practice Exp., 37: 1405-1449. DOI: 10.1002/spe.810

Sherry, S.T., M.H. Ward, M. Kholodov, J. Baker and L. Phan *et al*., 2001. dbSNP: The NCBI database of genetic variation. Nucl. Acid Res., 29: 308-311. DOI: 10.1093/nar/29.1.308

Strumpen, V., 1995. Coupling hundreds of workstations for parallel molecular sequence analysis. Software: Practice Exp., 25: 291-304. DOI: 10.1002/spe.4380250305

Trystram, D. and J. Zola, 2005. Parallel multiple sequence alignment with decentralized cache support. Euro-Par 2005 Parallel Proces. DOI: 10.1007/11549468_133

Wallace, I.M., O. Orla and D.G. Higgins, 2004. Evaluation of iterative alignment algorithms for multiple alignment. Bioinformatics, 21: 1408-1414. DOI: 10.1093/bioinformatics/bti159

Zola, J., D. Trystram, A. Tchernykh and C. Brizuela, 2006. Parallel multiple sequence alignment with local phylogeny search by simulated annealing. Proceedings of the 20th International Parallel and Distributed Processing Symposium, Apr. 25-29, IEEE Xplore Press, Rhodes Island, pp: 1-8. DOI: 10.1109/IPDPS.2006.1639536