PaMSA: A Parallel Algorithm for the Global Alignment of Multiple Protein Sequences

Irma R. Andalon-Garcia and Arturo Chavoya
Department of Information Systems
Guadalajara University – CUCEA
Guadalajara, Jalisco, Mexico

***Abstract*—Multiple sequence alignment (MSA) is a well-known problem in bioinformatics whose main goal is the identification of evolutionary, structural or functional similarities in a set of three or more related genes or proteins. We present a parallel approach for the global alignment of multiple protein sequences that combines dynamic programming, heuristics, and parallel programming techniques in an iterative process. In the proposed algorithm, the longest common subsequence technique is used to generate a first MSA by aligning identical residues. An iterative process improves the MSA by applying a number of operators that were defined in the present work, in order to produce more accurate alignments. The accuracy of the alignment was evaluated through the application of optimization functions. In the proposed algorithm, a number of processes work independently at the same time searching for the best MSA of a set of sequences. There exists a process that acts as a coordinator, whereas the rest of the processes are considered slave processes. The resulting algorithm was called PaMSA, which stands for Parallel MSA. The MSA accuracy and response time of PaMSA were compared against those of Clustal W, T-Coffee, MUSCLE, and Parallel T-Coffee on 40 datasets of protein sequences. When run as a sequential application, PaMSA turned out to be the second fastest when compared against the nonparallel MSA methods tested (Clustal W, T-Coffee, and MUSCLE). However, PaMSA was designed to be executed in parallel. When run as a parallel application, PaMSA presented better response times than Parallel T-Cofffee under the conditions tested. Furthermore, the sum-of-pairs scores achieved by PaMSA when aligning groups of sequences with an identity percentage score from approximately 70% to 100%, were the highest in all cases. PaMSA was implemented on a cluster platform using the C++ language through the application of the standard Message Passing Interface (MPI) library.**

Keywords*—*Multiple Sequence Alignment; parallel programming; Message Passing Interface

I. Introduction

A fundamental research subarea of bioinformatics is biological sequence alignment and analysis, which focuses on developing algorithms and tools for comparing and finding similarities in nucleic acid (DNA and RNA), and amino acid (protein) sequences [1]. The sequence similarities found are used for identifying evolutionary, structural or functional similarities among sequences in a set of related genes or proteins [2]. The set of sequences to be aligned are assumed to have an evolutionary relationship. Sequence alignment plays a central role in several areas of biology, such as phylogenetics, structural biology, and molecular biology.

Multiple sequence alignment (MSA) can be defined as the problem of comparing and finding which parts of the sequences are similar and which parts are different in a set of three or more biological sequences. The resulting alignment can be used to infer sequence homology. Homologous sequences are sequences that share a common ancestor and usually also share common functions.

Multiple sequence alignment is a well-known problem in computer science. A number of strategies have been applied to obtain MSAs, such as progressive alignment methods [3][4], iterative methods [5][6], dynamic programming [7], genetic algorithms [8], greedy algorithms [9], Markov chain processes [10], and even simulated annealing methods [11]. Currently, MSAs are obtained via two main approaches. The most popular alternative is the progressive multiple sequence alignment method. The main drawback with progressive alignments is that errors in the initial alignments of the most closely related sequences are propagated to the final multiple sequence alignment. The second most common approach to accomplish MSAs is the use of heuristic methods, which are more efficient than dynamic programming, but that do not guarantee finding an optimal alignment.

The main contribution of the present work is the development of a parallel algorithm—PaMSA, which stands for Parallel MSA—for the global alignment of multiple protein sequences. The strategies applied in PaMSA to obtain an MSA of a set of sequences differ from those of other currently used MSA algorithms in several ways. The PaMSA algorithm is not a progressive-alignment approach, as all sequences are aligned simultaneously. In contrast to existing heuristic alignment methods, which start from completely unaligned sequences, the PaMSA algorithm generates an initial MSA of the sequences based on a Longest Common Subsequence (LCS) of the set of sequences to be aligned. In addition, in the PaMSA algorithm several processes work independently at the same time searching for the best MSA of a set of sequences. Thus, the PaMSA algorithm combines a number of strategies to produce the sequence alignment.

The PaMSA algorithm was implemented as a parallel program that runs on a cluster platform; however it is not necessary to have a cluster environment to execute the application, as it can run even on a single processor. Currently, only protein sequences are aligned by PaMSA, but it is possible to adapt the implementation to align nucleic acid sequences as well.

Our implementation of PaMSA was compared against the currently used MSA algorithms Clustal W [3], T-Coffee [4], MUSCLE [5], and Parallel T-Coffee [12]. The comparison against the first three methods was done using a sequential version of PaMSA, as these methods are non-parallel implementations of the respective algorithms. The comparison in all cases was through the application of the sum-of-pairs function [13]. PaMSA was faster than Parallel T-Coffee, whereas the sequential version of PaMSA was the second fastest when compared against the nonparallel methods.

The remainder of this article is organized as follows. Section II describes the PaMSA algorithm and the metrics used to evaluate the alignments, whereas Section III specifies the protein sets and the conditions for the runs. Results are presented in Section IV with a discussion of their relevance. Finally, Section V presents the conclusions and future work.

II. The PaMSA Algorithm

Our parallel approach for the global alignment of multiple protein sequences, PaMSA [14], combines dynamic programming, heuristics, and parallel programming techniques in an iterative process. Dynamic programming techniques are applied for setting up an initial alignment. The algorithm improves the initial MSA in an iterative manner by applying a number of operators that move, delete or realign gaps. The algorithm ends when the termination criteria are reached.

The PaMSA algorithm was implemented on a cluster platform. Hence, in this approach a number of processes work in parallel for the search of the best MSA of a set of sequences. If np is the number of processes used by the algorithm, the number of possible different MSA solutions is equal to np. For example, if $np = 2$, there will be 2 independent processes searching for the best alignment. As the number of processes increases, the number of solutions increases as well. A consecutive integer $0, 1, 2, \ldots, np - 1$ is assigned to each process, which acts as an identification number (id) for the process. There exists a process that acts as a coordinator, whereas the rest of the processes are considered slave processes. The id for the coordinator process is always equal to zero. Slave processes have a consecutive integer id, which goes from 1 to the total number of processes minus one. The algorithm was implemented to be run on a cluster, however it works also on a nonparallel environment. In order to evaluate the quality of the MSA, a number of objective functions were implemented.

A. General structure

As mentioned above, the PaMSA algorithm is not a progressive-alignment approach, as all sequences are aligned simultaneously. In contrast to existing heuristic alignment methods, which start from completely unaligned sequences, PaMSA generates an initial MSA of the sequences based on a Longest Common Subsequence (LCS) of the set of sequences. The proposed algorithm follows a strategy analogous to a parallel genetic algorithm. The main steps in the general structure of a simple genetic algorithm (GA) are followed in the basic PaMSA algorithm procedure (Fig. 1). In PaMSA there is a population of initial MSAs, whereas in a GA there is a population of random initial solutions. In PaMSA, alignments are given a score, whereas in a GA, individual solutions are evaluated by an optimization function. An alignment is improved by applying operators in PaMSA, whereas individuals in a population evolve by applying operators in a GA. Finally,

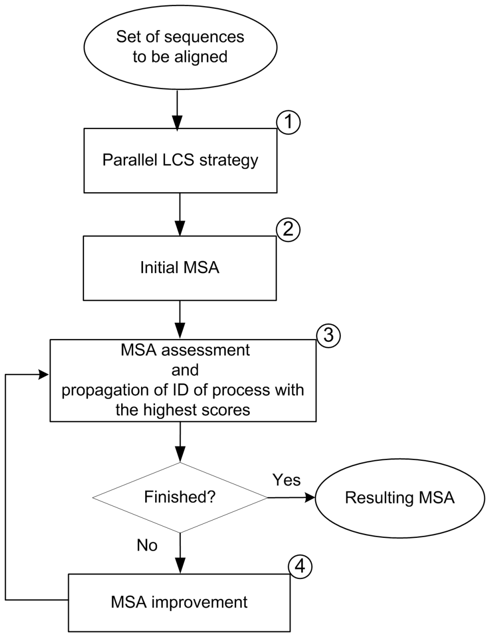
Set of sequences to be aligned
1
Parallel LCS strategy
2
Initial MSA
3
MSA assessment and propagation of ID of process with the highest scores
Finished?
Yes
Resulting MSA
No
4
MSA improvement

Fig. 1. The basic PaMSA algorithm procedure.

in both algorithms, operators are applied in an iterative process until a predefined condition is satisfied.

PaMSA combines a number of strategies to produce the sequence alignment, which are briefly described next and explained in more detail in the following sections. First, a well-known LCS technique for two sequences that uses dynamic programming was adapted and implemented to obtain an LCS of more than two sequences. In this approach, a number of processes work in parallel, so that each process calculates an LCS of the sequences. Even though all processes apply the same algorithm to the same set of sequences, the resulting LCSs are possibly different, because the calculations are based on a different order of sequences and there exists the possibility of having more than one LCS for the same sequences. Second, an algorithm is applied to the set of sequences in order to generate a first MSA by aligning identical residues, as well as similar residues, as much as possible. This algorithm uses the LCS generated at each process, which can be different from the LCSs in the other processes. This approach allows various potential solutions to be running in separate processes. Third, after the first MSA is generated in each process, the quality of the MSA is evaluated using a set of objective functions (OFs). Each process evaluates its MSA, and the slave processes send the scores of four of the OFs to the coordinator—the ID, the SY, the SP, and the PWS scores, described below—, which receives the scores and determines what process has the best MSA for all four OFs. The coordinator then propagates the id of the process with the best scores to all the slave processes. If the alignment has not shown improvement in all processes in two consecutive iterations, or if a predefined number of iterations is reached, the algorithm ends and the process with the best alignment of the sequences provides the resulting MSA. Otherwise the alignment is improved at each process by iteratively applying a number of operators that move, delete or realign gaps in the sequences following specific rules. These proposed operators perform a search along the length of the sequences with the aim of finding an opportunity to improve the alignment. The search is focused on the detection of gaps in order to minimize their number. The operators accept a certain number of parameters. Therefore, the operators can act differently in the sequences of the separate processes in order to have a variety of potential solutions. After each iteration, the resulting MSA is evaluated in all processes. This procedure is repeated until the termination criteria mentioned above are met.

B. The LCS technique

Given a sequence $S_i = s_{i1}s_{i2}\ldots s_{im}$, a subsequence of S_i is a sequence $S' = s'_1 s'_2 \ldots s'_p$, defined by $s'_k = s_{ir_k}$, where m is the length of sequence S_i, $r_1 < r_2 < r_p$, p is the number of selected items from sequence S_i, $1 \leq k \leq p$, and $p \leq m$; i.e. S' can be obtained by deleting $m - p$ (not necessarily contiguous) symbols from S_i without changing its order.

Let $S_1 = s_{11}s_{12}\ldots s_{1m}$ and $S_2 = s_{21}s_{22}\ldots s_{2n}$ be two sequences of length m and n, respectively. The sequence $S' = s'_1 s'_2 \ldots s'_p$ is a common subsequence of S_1 and S_2, if S' is a subsequence of both sequences. The LCS of S_1 and S_2 is the longest sequence S' that is a subsequence of both S_1 and S_2. In general, the LCS problem consists of finding the maximal-length subsequence—i.e. there exists no other subsequence that has greater length—that is a common subsequence of the sequences.

Let S_1 and S_2 be the above defined sequences of length m and n, respectively. The algorithm implemented to obtain the LCS of two sequences [15] uses dynamic programming and requires calculating the LCS table (LCST) as

$$LCST(i,j) = \begin{cases} 0 & \text{if } j = 0 \quad \text{or} \quad i = 0 \\ LCST(i-1,j-1)+1 & \text{if } i > 0, j > 0 \ \ and \ \ s_{1i} = s_{2j} \\ max(LCST(i,j-1), LCST(i-1,j)) & \text{if } i > 0, j > 0 \ \ and \ \ s_{1i} \neq s_{2j} \end{cases},$$

where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. The number of rows and columns in LCST are $m + 1$ and $n + 1$, respectively, whereas the cell $LCST(i, j)$ is the element in the LCS table at row i and column j. The LCS table stores numbers which correspond to the actual length of the LCS. After filling the LCS table, the lower right cell in the table contains the length of the LCS. The longest common subsequence can be found by tracing back from the cell at $LCST(m, n)$. Each time a match is found, it is appended to the longest common subsequence and a movement is made to cell $LCST(i - 1, j - 1)$. When the symbols do not match, a movement is made to the cell with $max(LCST(i - 1, j), LCST(i, j - 1))$ in order to find the next match. In general, there may be several such paths, because the LCS is not necessarily unique, i.e. it is possible to have more than one LCS. For example, let S_1="MFVFS" and S_2="MVFVS". After application of the previous rules, the subsequence "MFVS' is the LCS of the sequences. However, if we placed the sequences in the inverted order, the subsequence "MVFS" would be the LCS of the sequences.

C. Parallel LCS strategy

Let $S = \{S_1, S_2, \ldots, S_n\}$ be a set of n protein sequences, where $S_i = s_{i1}, s_{i2}, \ldots, s_{im_i}$, m_i is the length of S_i for $i = 1, 2, \ldots, n$, and s_{ik} is the k^{th} residue in the sequence S_i. Let np be the number of processes used by the algorithm. In this step, each process calculates an LCS of the set S. First, sequences are read and saved into an array. The procedure in this step is as follows: the i^{th} process applies the LCS algorithm to all possible pairs of sequences $LCS(S_i, S_j)$ in the set S that result from the combination of sequence S_i for $i = 1, 2, \ldots, n$ with the rest of the sequences S_j, for $j = 1, 2, \ldots, n$ and $i \neq j$. Even though all processes apply the same algorithm to the same set of sequences, the resulting LCSs are possibly different, because the calculations are based on a different order of sequences and there exists the possibility of having more than one LCS for the same sequences, as previously noted. For example, in the first iteration of this step, *Process 1* applies the algorithm to the following pairs of sequences:

$$LCS(S_1, S_2), LCS(S_1, S_3), \ldots, LCS(S_1, S_n).$$

In the same manner, *Process 2* will apply the algorithm to the following pairs of sequences:

$$LCS(S_2, S_1), LCS(S_2, S_3), \ldots, LCS(S_2, S_n),$$

and a similar strategy is applied for the rest of the np processes.

The results obtained from this first iteration are saved in order to create a new set of sequences. Thus, this new set of sequences contains the LCSs of the pairs of sequences in the original set S and its size will be $n - 1$. Next, the process repeats this iterative procedure with the obtained LCSs until there remains only one LCS. When this happens, it means that the LCS of the sequences in the set S has been found.

D. Setting up an initial MSA

After the LCS is obtained, an algorithm is applied to the set of sequences in order to generate a first MSA. This algorithm aligns identical residues of the sequences by using the resulting LCS. In general, the algorithm aligns identical residues, as well as similar residues, as much as possible.

Let A be an array of strings, with the sequences to be aligned, and n the number of rows (sequences) in array A, with $A_i = a_{i1}, a_{i2}, \ldots, a_{im_i}$ the i^{th} row in array A, m_i the length of sequence in A_i, for $i = 1, 2, \ldots, n$, and a_{ij} the j^{th} element (residue) in row A_i (sequence i), for $j = 1, 2, \ldots, m_i$. Moreover, let $R = r_1 r_2 \ldots r_p$ be a string with the LCS of the sequences in array A, with p the length of string R. The algorithm initiates with $j = 1$ (pointing to the first column in array A) and $k = 1$ (pointing to the first element in string R). The element r_k in string R is compared with all elements a_{ij} in A for $i = 1, 2, \ldots, n$. The resulting comparison can fall into one of the following three cases:

Case A. All elements a_{ij} in A match the element r_k in R. No gap is inserted in the sequences of array A, so identical residues are aligned, and k is increased for $k = 1, 2, \ldots, p$, in order to point to the next element r_{k+1} in string R.

Case B. None of the elements a_{ij} in A match the element r_k in R. No gap is inserted in the sequences of array A, so residues at this position are aligned, and k is increased for

$k = 1, 2, \ldots p$ in order to point to the next element r_{k+1} in string R.

Case C. Only some elements a_{ij} in A match the element r_k in R. An iterative procedure introduces a gap in the sequences of array A at position a_{ij}, if $a_{ij} = r_k$ for $i = 1, 2, \ldots, n$.

In all the previous cases, j is increased in order to point to the next residue of sequences in array A, for $j = 1, 2, \ldots, m$. The iterative procedure is repeated until the last element r_p in string R is processed. The maximum length of sequences in array A is calculated, and this length is established as the length of the initial MSA. Finally, gaps are inserted if needed at the end of sequences having a smaller length than the maximum length calculated, so that all sequences have the same length. As a result of the previous calculations, a matrix is created containing an initial MSA.

The procedure described above uses the LCS generated at each process, which could be different from the LCSs in the other processes. This approach allows various potential solutions to be running in separate processes.

E. MSA assessment

The accuracy of PaMSA—i.e. the quality of the alignment—is evaluated using the following five optimization functions (OFs):

$$OFs = \begin{cases} ID(MSA) \\ SY(MSA) \\ SP(MSA) \\ PWS(MSA) \\ NG(MSA) \end{cases},$$

where ID measures the identity percentage score, SY evaluates the similarity percentage score, SP calculates the sum-of-pairs score, PWS obtains a pairwise score of the sequences compared with the first sequence in the alignment, and NG counts the number of gaps in the alignment.

1) Identity percentage (ID): In our implementation, the identity percentage score among the sequences being aligned is calculated as

$$ID(A) = \left(\left(\sum_{j=1}^{r} \sum_{i=1}^{n} a_{ij} \right) * 100 \right) / r \,, \tag{1}$$

where A is the array with the MSA as previously defined, r is the length of the aligned sequences in A, n is the number of sequences aligned, and $\sum_{i=1}^{n} a_{ij}$ is counted only if all the a_{ij} are identical for $i = 1, 2, \ldots, n$ in the j^{th} column of the MSA for $j = 1, 2, \ldots, r$. In general, the higher the identity percentage score, there better the alignment.

2) Similarity percentage (SY): The similarity percentage score is calculated using the same formula used to calculate the identity percentage score. However, the sum $\sum_{i=1}^{n} a_{ij}$ is counted only if all the a_{ij} in the j^{th} column of the MSA are similar—i.e. not necessary identical but imperatively different from a gap. In general, the higher the similarity percentage score, the better the alignment.

3) Sum of pairs (SP): The sum-of-pairs score is a metric for measuring MSA accuracy, based on the number of correctly aligned residue pairs, where the score of all pairs of sequences in the multiple alignment is added to the overall score. The SP score is calculated as

$$SP(A) = \sum_{i=1}^{r} sp(a_i) \,, \tag{2}$$

$$sp(a_i) = \sum_{1 \le k < l \le n} s(a_{ki}, a_{li}) \,, \tag{3}$$

where A is the array with the MSA as previously defined, r is the length of the aligned sequences in A, n is the number of rows (sequences) in array A, and $s(a_{ki}, a_{li})$ is the score obtained by comparing the k^{th} row in the i^{th} column of the MSA with the l^{th} row in the same i^{th} column of the MSA for $k = 1, 2, \ldots, n-1$ and for $l = 2, 3, \ldots, n$. This score is calculated using the following general formula:

$$s(a_{ki}, a_{li}) = \begin{cases} 1 & \text{if } a_{ki} = a_{li} \\ -1 & \text{if } a_{ki} \neq a_{li} \\ -2 & \text{if } a_{ki} = gap \ XOR \ a_{li} = gap \\ 0 & \text{if } a_{ki} = gap \ AND \ a_{li} = gap \end{cases}.$$

The value of the sum-of-pairs score depends on the number of sequences aligned, the length of the sequences aligned, and the similarity among the sequences aligned. Therefore, there is not a pre-established range of values for this score. The higher the sum-of-pairs score of a particular set of sequences, the better its alignment. It is possible to use a substitution matrix to compare the residues among sequences in order to obtain better alignments. The BLOSUM62 matrix is provided in our implementation as it is the *de facto* standard in protein database searches and sequence alignments [16].

4) Pairwise score (PWS): The pairwise score of sequences was included in the evaluation of our algorithm. In our implementation, this pairwise score obtained among all pairs of sequences is calculated as

$$PWS(A) = \sum_{i=1}^{r} \sum_{j=2}^{n} s(a_{1i}, a_{ji}) \,, \tag{4}$$

where A is the array with the MSA as previously defined, r is the length of the aligned sequences in A, n the number of rows (sequences) in array A, and $s(a_{1i}, a_{ji})$ is the score obtained by comparing the $first$ row in the i^{th} column of the MSA with the j^{th} row in the same i^{th} column of A. The same comparison evaluation criteria as in *SP* are used. The value of the pairwise score depends on the number of sequences aligned, the length of the sequences aligned, the similarity among the sequences aligned, and the first sequence in the alignment. Hence, there is not a pre-established range of values for this score. In general, the higher the sum-of-pairs score of a particular set of sequences, the better the alignment.

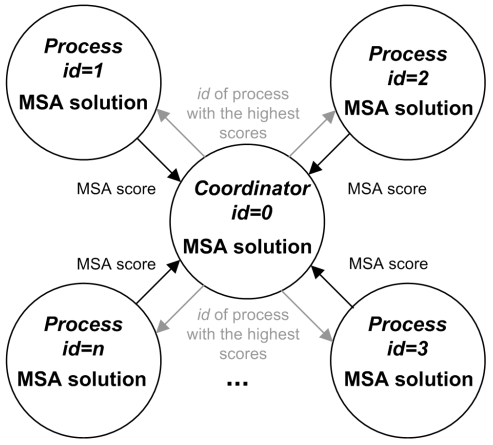

Fig. 2. Communication topology used among processes.

5) Number of gaps (NG): The number of gaps is an additional score, which is calculated using the formula

$$NG(A) = \sum_{j=1}^{r} \sum_{i=1}^{n} a_{ij} \,, \tag{5}$$

where A is the array with the MSA as previously defined, r is the length of the aligned sequences in A, n is the number of rows (sequences) in array A, and a_{ij} is counted only if it is a gap. The value of the number of gaps depends on the number of sequences aligned, the length of the sequences aligned, and mainly on the similarity among the sequences aligned. Therefore, there is not a pre-established range of values for this score. The fewer the number of gaps of a particular set of sequences, the better the alignment.

After the first MSA is generated in each process, the alignment is evaluated using the implemented OFs. Each slave process evaluates its MSA and sends the scores of four of the OFs to the coordinator—the ID, the SY, the SP and the PWS scores—, which receives the scores and determines which process has the best MSA for all four OFs (Fig. 2). The Number of Gaps (NP) score is calculated and displayed in the screen output, but it was left out of the selection criterion, as preliminary results suggested that the other four OFs were sufficient to evaluate the alignment.

After the coordinator process receives the OFs scores from the slave processes, it propagates the id of the process with the best scores to all the slave processes. If the scores of all four OFs of the MSA have not shown improvement in two consecutive iterations or if a predefined number of iterations is reached, the algorithm ends and the process with the best alignment of the sequences provides the resulting MSA. Otherwise the alignment is improved by iteratively applying a number of operators that move, delete or realign gaps in the sequences following specific rules.

F. Improvement of the MSA

In order to improve the MSA, sixteen operators were defined in the present work. In the current version of PaMSA, there exist two main groups of operators: the basic operators, and the refinement operators, both shown in Table I. The main differences between the two groups of operators are that refinement operators can be applied even when only one sequence of the two sequences has the gaps, and that some of them are applied only in the last iteration of the algorithm—i.e. when the number of generations was reached or if there was no improvement in the alignment after two consecutive iterations—, in contrast to basic operators which are applied only when both sequences have gaps. The proposed operators perform an exhaustive search along the total length of all sequences with the aim of finding an opportunity to improve the alignment. The search is focused on the detection of gaps and identical or similar residues that are not totally aligned.

TABLE I. ALIGNMENT IMPROVEMENT OPERATORS

Operator	Function	Type
mGapRF_3	Moves three gaps in 1st sequence to the right	BS
mGapRS_3	Moves three gaps in 2nd sequence to the right	BS
mGapRF_2	Moves two gaps in 1st sequence to the right	BS
mGapRS_2	Moves two gaps in 2nd sequence to the right	BS
mGapRF_1	Moves a gap in 1st sequence to the right	BS
mGapRS_1	Moves a gap in 2nd sequence to the right	BS
mGapRF_G	Moves a gap in 1st sequence to the right	BS
mGapRS_G	Moves a gap in 2nd sequence to the right	BS
rGaps	Removes an MSA column if all elements are gaps	BS
mGapRF_3S	Realigns three gaps in 1st sequence to the right	RF
mGapRS_3S	Realigns three gaps in 2nd sequence to the right	RF
mGapRF_2S	Realigns two gaps in 1st sequence to the right	RF
mGapRS_2S	Realigns two gaps in 2nd sequence to the right	RF
mGapLF_1S	Realigns a gap in 1st sequence to the right	RF
mGapLS_1S	Realigns a gap in 2nd sequence to the right	RF
mGapn	Moves a residue in 2nd sequence *n* columns	RF

Type: BS = Basic, RF = Refinement.

The operators are always applied to pairs of sequences. At every iteration, operators are applied—when necessary—to each of the potential solutions running in the independent processes. An assessment method marks columns of sequences in the MSA when their elements are totally aligned, so that the algorithm will not apply the operators to those columns in future iterations. This strategy improves the performance of the algorithm.

1) Basic operators: There are nine basic operators (Table I), which mainly move gaps trying to minimize their number by eliminating columns that only contain gaps. The $mGapRF\_3$ operator moves three gaps to the right in the first sequence of a pair of sequences being compared in the alignment. This operator is applied in order to align identical residues. The $mGapRS\_3$ operator acts in a similar way as the $mGapRF\_3$ operator, but in this case the operator is applied to the second sequence of the pair of sequences being compared. In the same manner, the $mGapRF\_2$ and the $mGapRS\_2$ operators move two gaps to the right in the first and second sequence, respectively. These operators are also applied in order to align identical residues. Similarly, the $mGapRF\_1$ operator moves a gap to the right in the first sequence of a pair of sequences with the aim of aligning identical residues. The $mGapRS\_1$ operator moves a gap to the right in the second sequence of the pair of sequences being compared in order to align identical residues.

The $mGapRF\_G$ and the $mGapRS\_G$ operators move a gap to the right in the first or second sequence, respectively, of a pair of sequences. These operators are applied in order to reduce the number of gaps by aligning similar—i.e. non-identical—residues.

The *rGaps* operator is used to remove a column from the alignment when all the residues in the column are gaps. This operator is applied after the application of any of the other operators. Once the *rGaps* operator has been applied, a new assessment of the MSA is made in order to update the MSA scores.

2) *Refinement operators:* As can be seen in Table I, there are seven refinement operators. The $mGapRF\_3S$ and $mGapRS\_3S$ operators move three gaps to the right in the first and the second sequence, respectively, of a pair of sequences being compared. These operators are applied in order to align identical residues. Unlike the $mGapRF\_3$ and $mGapRS\_3$ operators, these refinement operators are applied even when only one of the sequences has the gaps. The $mGapRF\_2S$ and $mGapRS\_2S$ operators act similarly as the $mGapRF\_3S$ and the $mGapRS\_3S$ operators, but in this case the refinement operators realign only two gaps to the right.

The three remaining refinement operators, $mGapLF\_1$, $mGapLS\_1$, and $mGapn$, move a gap to the left in order to align identical residues in the alignment. Because these operators are the only ones that move gaps to the left, they are applied at the last iteration of the algorithm.

III. Method

The implementation of the PaMSA algorithm was developed on a computer cluster provided by Intel Corporation, which contained 10 nodes, each node with two Intel Xeon 5670 6-core 2.93 GHz CPUs, 24 GB of 1066 MHz DR3 RAM, and two 274 GB 15K RPM hard drives. The operating system used was Red Hat Enterprise Linux 5 Update 4 with Perceus 1.5 Clustering Software and Server 5.3 running Intel MPI 3.2. An implementation of PaMSA can be downloaded from http://www.bioinformatics.org/pamsa.

The Message Passing Interface (MPI) library was used in our implementation of the algorithm. MPI defines the syntax and semantics of a set of functions in a library designed to exploit the existence of multiple processors, and it provides the synchronization and communication needed among processes. Synchronous communication operations were used in this work to handle communication and synchronization among tasks. When a synchronous operation is invoked, a process sends a message and then waits for a response before proceeding with the process flow. Object-oriented and structured programming paradigms were applied using C++ as the programming language. The PaMSA algorithm was implemented on a cluster platform using the Linux operating system; however, PaMSA can be run on a nonparallel environment.

Results presented in this work were obtained from alignments performed on the Hybrid Cluster Supercomputer *Xiuhcoatl* of the General Coordination of Information and Communications Technologies (CGSTIC) at CINVESTAV, in Mexico City. This cluster contained 88 nodes, each node with 1056 Intel X5675 CPUs, 2112 GB of RAM, and 22000 GB in local hard disk drives.

Table II presents the 40 datasets of protein sequences that were used in the present work in order to analyze the performance of PaMSA and the other MSA methods tested. Each protein dataset was chosen according to its number of sequences, identity percentage, and length average. Datasets were organized in eight groups of five sequence clusters each, named from A to H (i.e. Group A, Group B, and so on). The groups of sequences were obtained from the UniProt Reference Clusters (UniRef) contained in the UniProKB protein database. At this site, sequences are classified in groups—called clusters—according to their identity percentage; thus, similar sequences can be obtained through a database query. Sequences that belong to a specific cluster are called cluster members.

TABLE II. Cluster Groups used for the Alignments

Group	Dataset	Cluster name	Number of sequences	Average length	Minimum identity (%)
A	1	C1CIT9	10	100	100.0
	2	P15020	52	100	100.0
	3	E5G6P9	98	100	100.0
	4	Q8XA92	129	100	100.0
	5	A7X428	177	100	100.0
B	6	B5Y0W9	40	80	100.0
	7	B5XNF9	40	127	100.0
	8	Q5HNW5	40	210	100.0
	9	B5XZH9	40	309	100.0
	10	Q5HPT8	40	467	100.0
C	11	Q5HP42	37	468	100.0
	12	C1HFG3	45	597	100.0
	13	A0A1X4	50	757	100.0
	14	B3BQU1	83	843	100.0
	15	Q5HG69	114	914	100.0
D	16	O53454	48	369	98.9
	17	Q833H5	66	228	98.2
	18	Q6GIG7	37	354	97.4
	19	A7WZ82	128	100	91.0
	20	P10321	18	355	90.7
E	21	Q2YIT5	55	231	87.3
	22	A0A084	14	326	85.8
	23	A0B0S7	13	407	83.5
	24	Q9KTA3	72	303	81.8
	25	K0HNA3	77	193	79.8
F	26	A0AZ41	25	169	76.6
	27	A0A2S1	20	129	78.2
	28	A0A125	14	47	75.5
	29	A0A0A7	10	147	75.9
	30	A0A1U6	9	124	70.0
G	31	A0A0X0	7	374	69.9
	32	A0B092	21	264	68.2
	33	Q3SZ22	5	279	66.3
	34	A0AZ19	23	116	65.7
	35	A0B0W2	13	174	61.6
H	36	A0A092	19	294	60.6
	37	A0A0T0	25	228	53.7
	38	A0A9I3	31	308	52.1
	39	A0A132	20	50	52.3
	40	A0A194	48	119	50.4

The Identity score shown is the minimum of all methods tested for each cluster. These protein datasets can be found at http://www.uniprot.org/uniref/.

The number of iterations of the PaMSA algorithm can be modified by the user as a parameter, the default value being five. A file with the resulting MSA was created with sequences in *clustal* format. The MSA output file has the same name as the input file but with the *pamsa* extension. Basic validations are implemented, such as verification of the existence of the input file with the sequences to be aligned, the creation of the output file, the correct introduction of the parameters given, and the verification of the FASTA format of the sequences to be aligned. PaMSA was compared against the following versions of the MSA programs: MUSCLE v3.7, Clustal W v2.0.10, T-Coffee v9.03r1318, and Parallel T-Coffee v1.913, all of them running in Linux.

IV. Results and Discussion

In this section we present results obtained from alignments using PaMSA, as well as comparisons made against several methods commonly used for MSA, namely MUSCLE, Clustal

W, T-Coffee, and Parallel T-Coffee—a parallel implementation of T-Coffee. Of particular note is the Parallel T-Coffee method, which runs on a cluster platform and uses the MPI library, just as our implementation of the PaMSA algorithm. The variables used for evaluating the performance of the methods tested were the MSA accuracy (quality of the alignment), and the response time.

A. *MSA accuracy results*

The sum-of-pairs score was used in the present work for evaluating the quality of the alignments, as it is a simple and sensitive measure for assessing the accuracy of alignments and has been widely used [17]. The greater the sum-of-pairs score, the better the alignment obtained; thus, the alignment with the highest sum-of-pairs score is considered the most accurate (the best) MSA of all the alignments obtained. The sum-of-pairs scores for all dataset groups and MSA methods are presented in Table III.

TABLE III. SUM-OF-PAIRS SCORES OF RESULTING MSAs

Group	Dataset	MUSCLE	Clustal W	T-Coffee	PaMSA	Parallel T-Coffee
A	1	4500*	4500*	4500*	4500*	4500*
	2	132600*	132600*	132600*	132600*	132600*
	3	475300*	475300*	475300*	475300*	475300*
	4	825600*	825600*	825600*	825600*	825600*
	5	1557600*	1557600*	1557600*	1557600*	1557600*
B	6	62400*	62400*	62400*	62400*	62400*
	7	99060*	99060*	99060*	99060*	99060*
	8	163800*	163800*	163800*	163800*	163800*
	9	241020*	241020*	241020*	241020*	241020*
	10	364260*	364260*	364260*	364260*	364260*
C	11	311688*	311688*	311688*	311688*	311688*
	12	591030*	591030*	591030*	591030*	591030*
	13	927325*	927325*	927325*	927325*	927325*
	14	2868730*	2868730*	2868730*	2868730*	2868730*
	15	5803340*	5803340*	5803340*	5803340*	5803340*
D	16	413408*	413408*	413408*	413408*	413408*
	17	481125*	481125*	481125*	481125*	481125*
	18	233649*	233649*	233649*	233649*	233649*
	19	803395*	803395*	803395*	803395*	803395*
	20	50301*	50301*	44567	50301*	50301*
E	21	305824	305824	305894*	305894*	305894*
	22	25766*	25766*	22003	25766*	25766*
	23	27094	27094	23018	27100*	22765
	24	721444*	721444*	721444*	721444*	721444*
	25	493975*	493975*	493975*	493975*	493975*
F	26	43936	42690	44148*	44148*	40559
	27	21960*	21960*	21960*	21960*	21960*
	28	3291*	3291*	2758	3291*	2980
	29	4637*	4637*	3769	4637*	4637*
	30	3467*	3467*	3467*	3467*	3467*
G	31	5951*	5951*	4093	5951*	5951*
	32	47610*	47434	43219	47144	45592
	33	1742*	1734	1062	1727	1046
	34	23312	23312	23312	23314*	23314*
	35	9934*	9934*	8804	8610	8659
H	36	30864*	30864*	27440	30639	27168
	37	50765*	50765*	46414	45218	46442
	38	137083*	137083*	127785	136413	127467
	39	5598*	5598*	4856	5598*	4975
	40	119868*	119868*	119868*	119868*	119868*

* = Best alignment obtained.

As can be seen, all algorithms achieved the optimal MSA—the alignment with the highest sum-of-pairs score—and 100% of identity percentage, when datasets of protein sequences from Groups A, B, and C were used (Table III). The datasets from these groups had originally 100% of identity percentage among them; thus, the LCS found by the PaMSA algorithm corresponded exactly to the sequences to be aligned, making it simple in this case to find the MSA with a perfect identity percentage.

In the MSA accuracy results obtained from alignments using datasets of protein sequences from Group D (clusters with an identity percentage score within the range from 90% to 99%), the T-Coffee method achieved the best MSA in 4 out of 5 cases tested (Table III). The PaMSA, MUSCLE, Clustal W and Parallel T-Coffee methods obtained the best alignment in all datasets of this group, based on the sum-of-pairs scores.

Clusters of sequences with an identity percentage score approximately within the range from 80% to 89% were used in alignments with sequences from Group E. Datasets from this group have slightly dissimilar sequences. The sum-of-pairs scores obtained using the PaMSA algorithm were the highest in all cases tested (Table III), i.e. the PaMSA algorithm obtained the best alignment in this group of alignments. The MUSCLE, Clustal W and T-Coffee methods obtained the best MSA in 3 out of 5 cases tested, whereas Parallel T-Coffee achieved the best MSA in 4 out of 5 cases.

The MSA accuracy results obtained in alignments using Group F (with an identity percentage score approximately within the range from 70% to 79%) were similar to the results obtained with Group E—the PaMSA algorithm also obtained the best alignment in all the cases tested. Datasets from this group have more variable sequences than the previous groups. The T-Coffee and Parallel T-Coffee methods achieved the best MSA in 3 out of 5 cases. The MUSCLE and Clustal W methods obtained the best MSA in 4 out of 5 cases tested.

The PaMSA algorithm obtained less accurate alignments, according to the sum-of-pairs score, than the MUSCLE and Clustal W methods in at least four cases tested from Group G (clusters with an identity percentage approximately within the range from 60% to 69%) and Group H (clusters with an identity percentage approximately within the range from 50% to 59%). However, the MSA accuracy results obtained by the PaMSA algorithm were equal or better than those obtained by the T-Coffee and Parallel T-Coffee methods using these groups of sequences.

In general, results show that the MUSCLE method had the best MSA accuracy of the methods tested, as it obtained the best alignments (according to the sum-of-pairs score) in all but four of the 40 cases tested. The Clustal W method and the PaMSA algorithm were a close second place in accuracy, achieving the best alignment in 34 out of 40 cases tested. The Parallel T-Coffee method obtained the best alignment in 30 of the cases tested, against the 26 achieved by the T-Coffee method.

With the exception of MUSCLE, PaMSA and the other tested MSA methods had trouble finding accurate alignments when using datasets with an identity percentage lower than 70%. Nevertheless, even in this case PaMSA was able to find the best alignment in 4 out of 10 datasets.

B. *Response time results of nonparallel methods*

This section presents the execution time results obtained from alignments using PaMSA and three common nonparallel methods for MSA: MUSCLE, Clustal W and T-Coffee. For these alignments, PaMSA and the other three methods were executed in a nonparallel environment. It should be mentioned that the results shown are the best execution times achieved from a set of five runs. Alignments were made under the same conditions for all the methods compared—i.e. computer, environment, operating system, and timer.

Table IV presents the execution time results in seconds obtained from alignments for all dataset groups. As can be seen, in Group A the MUSCLE method achieved the best response times, whereas the PaMSA algorithm had better response times than the Clustal W and T-Coffee applications for all datasets in this group. The performance results obtained using Group B were similar to the results from alignments using Group A, i.e. the MUSCLE method achieved the best response times and the PaMSA algorithm was the second best. The performance results obtained using Group C were similar to the results from the previous group, with the MUSCLE and the PaMSA methods in first and second place, respectively.

TABLE IV. SINGLE-PROCESSOR EXECUTION TIME RESULTS IN SECONDS

Group	Dataset	MUSCLE	PaMSA	Clustal W	T-Coffee
A	1	0.033*	0.034	0.053	0.398
	2	0.184*	0.445	0.532	2.208
	3	0.446*	1.521	1.709	7.094
	4	0.686*	2.614	2.903	12.615
	5	1.139*	4.934	5.446	25.833
B	6	0.107*	0.188	0.235	1.280
	7	0.173*	0.389	0.502	1.691
	8	0.311*	0.908	1.249	2.637
	9	0.533*	1.813	2.565	4.009
	10	0.968*	3.920	5.675	6.583
C	11	0.891*	3.358	5.089	5.531
	12	1.633*	7.886	11.427	11.365
	13	2.715*	15.422	22.097	22.451
	14	5.944*	52.515	71.605	65.168
	15	9.954*	113.462	149.945	154.006
D	16	0.859*	3.628	5.050	7.196
	17	0.655*	2.856	3.638	7.389
	18	0.595*	2.024	2.968	4.084
	19	0.679*	2.604	2.843	12.483
	20	0.411*	0.477	0.817	1.282
E	21	0.626*	1.958	2.574	5.403
	22	0.295	0.282*	0.468	0.884
	23	0.227*	0.320	0.630	0.976
	24	1.089*	4.869	7.433	12.050
	25	0.769*	3.015	3.397	7.984
F	26	0.176*	0.252	0.366	1.038
	27	0.095*	0.102	0.164	0.646
	28	0.026*	0.026*	0.042	0.448
	29	0.049*	0.049*	0.077	0.465
	30	0.041	0.032*	0.063	0.401
G	31	0.104	0.094*	0.211	0.512
	32	0.268*	0.388	0.612	1.132
	33	0.110*	0.134	0.168	0.800
	34	0.058	0.036*	0.086	0.372
	35	0.097	0.094*	0.145	0.514
H	36	0.374	0.342*	0.631	1.151
	37	0.288*	0.359	0.619	1.336
	38	0.811*	0.945	2.363	3.021
	39	0.035*	0.039	0.048	0.533
	40	0.214*	0.500	0.611	2.200

* = Best execution time.

From the execution time results obtained using Group D, there are no differences with previous results regarding the order of the best two methods, i.e. the MUSCLE method also achieved the best response times, whereas the PaMSA algorithm had better response times than the Clustal W and T-Coffee applications. The performance results obtained using Group E were similar to those of the previous alignments, with the exception of Dataset 22, with which PaMSA achieved the best response time. In the rest of the datasets from this group, the MUSCLE method achieved the best response times. The PaMSA algorithm showed once again with this group better response times than the Clustal W and T-Coffee methods. The performance results obtained using Group F were different from those of the previous alignments; in alignments using this group, PaMSA achieved the best response time when using Dataset 30, whereas this algorithm and MUSCLE method reached a tie in best execution time in two instances. The PaMSA algorithm was again superior to the Clustal W and T-Coffee methods when testing this group of sequences. The response times obtained by the PaMSA algorithm using Group G were superior to those of the other methods tested in three out of five alignments, whereas the MUSCLE algorithm achieved the best response time in the other two cases. Finally, using Group H, the MUSCLE method obtained the best response time in all but one of the cases tested in this group of alignments, whereas the PaMSA algorithm was again superior to the Clustal W and T-Coffee programs.

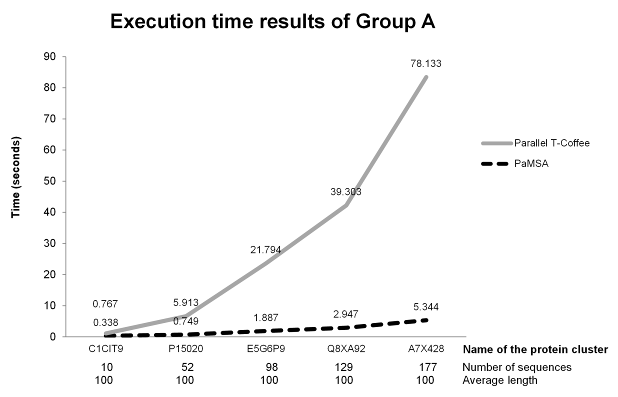

Fig. 3. Parallel execution times of PaMSA and Parallel T-Coffee using datasets from Group A. The times shown are the best of five runs for each dataset.

As can be seen, in most of the MSAs with the datasets presented in Table II, the MUSCLE method achieved the best execution time results. However, the PaMSA algorithm was superior or equal to the MUSCLE method in some cases. On the other hand, the execution times achieved by the PaMSA algorithm were better (i.e. lower) than the results obtained using the Clustal W and T-Coffee programs in all the cases tested.

C. Response time results of parallel methods

In this section we present the execution time results achieved by comparing the PaMSA algorithm against Parallel T-Coffee—a parallel version of the T-Coffee method. Alignments using these two methods were executed in a cluster environment under the same conditions (cluster type, number of processes, MPI library, operating system, and timer). Table V shows the execution times in seconds of Parallel T-Coffee and PaMSA. The times shown are the best of five runs for each dataset.

Fig. 3 graphically shows the execution time results achieved by the PaMSA algorithm and the Parallel T-Coffee method when using datasets of protein sequences from Group A; similar results were found for the rest of the groups. In all the cases tested, execution times achieved by PaMSA were superior to the results obtained with the parallel version of T-Coffee.

In order to confirm the superiority in performance of the PaMSA algorithm over the Parallel T-Coffee method, the

TABLE V. EXECUTION TIME RESULTS OF PARALLEL T-COFFEE AND PAMSA IN SECONDS

Group	Dataset	Parallel T-Coffee time (s)	PaMSA time (s)	Speedup
A	1	0.767	0.338	2.3
	2	5.913	0.749	7.9
	3	21.794	1.887	11.5
	4	39.303	2.947	13.3
	5	78.133	5.344	14.6
B	6	2.658	0.515	5.2
	7	5.555	0.715	7.8
	8	14.774	1.247	11.8
	9	30.149	2.152	14.0
	10	72.149	4.253	17.0
C	11	60.319	3.745	16.1
	12	148.757	8.304	17.9
	13	304.629	16.065	19.0
	14	1113.234	53.668	20.7
	15	2510.508	116.098	21.6
D	16	53.413	4.397	12.1
	17	57.237	3.577	16.0
	18	40.496	2.493	16.2
	19	38.118	3.316	11.5
	20	10.586	0.848	12.5
E	21	37.36	2.429	15.4
	22	4.452	0.585	7.6
	23	4.713	0.671	7.0
	24	151.67	5.627	27.0
	25	44.113	3.785	11.7
F	26	3.144	0.617	5.1
	27	1.628	0.413	3.9
	28	0.643	0.332	1.9
	29	0.919	0.367	2.5
	30	0.748	0.359	2.1
G	31	1.481	0.413	3.6
	32	4.643	0.75	6.2
	33	1.11	0.353	3.1
	34	1.516	0.466	3.3
	35	1.371	0.419	3.3
H	36	6.641	0.671	9.9
	37	4.346	0.721	6.0
	38	17.966	1.484	12.1
	39	0.733	0.359	2.0
	40	6.833	0.886	7.7

speedup for all datasets in all groups was computed by dividing the execution time of the Parallel T-Coffee method by the execution time of the PaMSA algorithm. Results showed that the PaMSA algorithm had better response time than Parallel T-Coffee in all the cases tested, as seen in Table V. The PaMSA algorithm was at least 1.9 and up to 27 times faster than Parallel T-Coffee, depending on the number and length of the sequences to be aligned.

A multi-factor ANOVA was done by group in order to statistically compare the execution times of the PaMSA and Parallel T-Coffee algorithms. Two factors were considered for the eight groups. Seven of the eight groups considered the algorithm and the number of the sequences as factors, whereas in Group B the algorithm and the average length of sequences were considered as factors. Table VI shows the p-valued obtained for each group.

TABLE VI. ANOVA p-VALUES BY CLUSTER GROUP

Group	p-value
A	0.998
B	0.0933
C	0.1473
D	0.0084
E	0.1547
F	0.0733
G	0.0589
H	0.0745

Based on their p-values, Table VI shows that in five of the eight groups (A, B, F, G, and H) there was a statistically

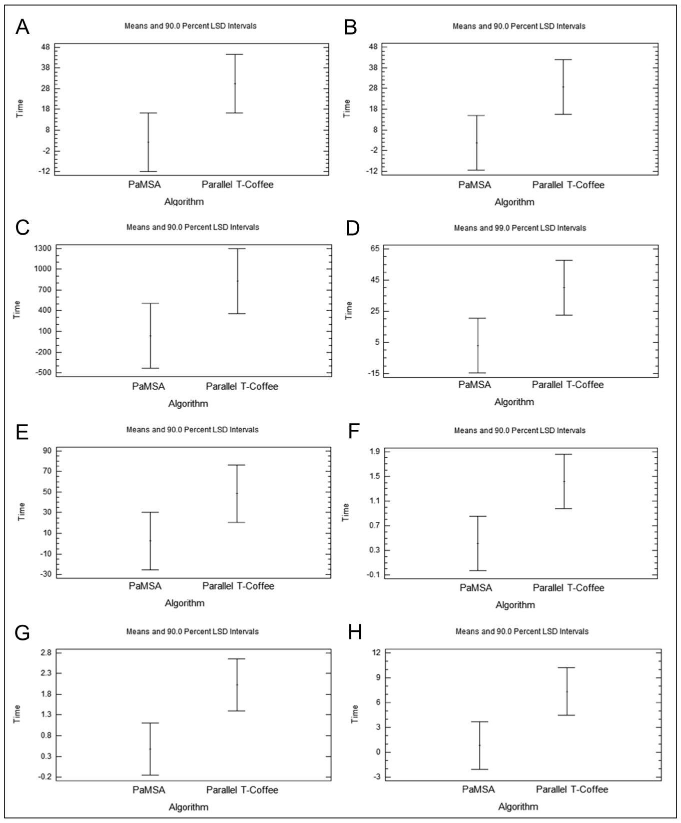

Fig. 4. Multi-factor ANOVA plot of means by group. The panel letters correspond to the protein sequence group being analyzed. Group B used the algorithm and the average length of sequences as factors, whereas the rest of groups considered the algorithm and the number of the sequences as factors.

significant difference between the execution time for the two algorithms at 90% of confidence, and in one of them (Group D) at 99% of confidence. In order to analytically discern which algorithm had better response time, a plot of means by group was performed. Fig. 4 graphically shows the plot of means by group. Insets 4A, 4B, 4D, 4F, 4G, and 4H show that PaMSA was statistically better than the Parallel T-Coffee algorithm in the corresponding groups, whereas Insets 4C and 4E show that PaMSA was statistically equal to Parallel T-Coffee in Groups C and E. Thus, statistically PaMSA was as fast or faster than the Parallel T-Coffee algorithm for all eight groups tested when the mentioned factors were considered.

V. CONCLUSION

The focus of this research project was to propose a parallel solution for the global multiple alignment problem of protein sequences by combining dynamic programming, heuristics, and parallel programming techniques in an iterative process. The resulting algorithm was named PaMSA, which stands for Parallel MSA. Execution time results obtained using the PaMSA algorithm compared against those of the Parallel T-Coffee method indicated that the PaMSA algorithm had equal or better performance in all the cases tested. Accordingly, a multi-factor ANOVA analysis was performed in order to confirm this tendency. Results of the statistical analysis indicate that PaMSA is as fast or faster than Parallel T-Coffee when considering the algorithm, and either the number of sequences or the average length of sequences (in Group B) as factors. It can be concluded that the PaMSA algorithm achieved in general better execution time results than the Parallel T-Coffee method under the conditions tested.

As for the comparison of execution times of the PaMSA algorithm against the sequential MSA methods tested, PaMSA was run as a one-processor application in a nonparallel environment and the results were compared against those of MUSCLE, T-Coffee and Clustal W. In 80% of the tested cases the MUSCLE method achieved shorter response times. However, the PaMSA algorithm was faster than the MUSCLE method in 15% of the cases. On the other hand, the execution times achieved by the PaMSA algorithm were better than the results obtained by Clustal W and T-Coffee in all the cases tested. It can be concluded that the PaMSA algorithm was the second faster of the methods under the nonparallel conditions tested.

As for the accuracy of the alignments, results achieved with the PaMSA algorithm in clusters of very similar protein sequences (within a range from 90% to 100% of identity percentage score, approximately) were at least as accurate as the alignments obtained with the other methods tested. It can be concluded that the PaMSA algorithm, along with the MUSCLE, Clustal W, and Parallel T-Coffee methods, achieved the best overall MSA accuracy results when using these groups of sequences. In general, when aligning closely related sequences, all the tested methods obtained the best—or close to the best—alignment. When using clusters of sequences with an identity percentage score of approximately 70% to 89%, PaMSA found the best alignment in all cases—according to the sum-of-pairs score—, whereas MUSCLE, Clustal W, T-Coffee, and Parallel T-Coffee could not find the best MSA in at least three cases. It can be concluded that the results achieved by the PaMSA algorithm were better than the other methods tested with these groups of sequences. It is possible to assume that when aligning more dissimilar sequences, not all methods can obtain the best alignment. Finally, when using clusters with approximately 50% to 69% of identity percentage score, the PaMSA algorithm achieved less accurate alignments than the MUSCLE and Clustal W methods in 6 out of 10 datasets in both cases. However, the alignments obtained by the PaMSA algorithm were equal or even better than the alignments obtained by the T-Coffee and Parallel T-Coffee methods in 8 out of 10 cases tested in these groups of sequences. According to our results, no single MSA method can always obtain the best alignment for all sets of sequences.

Future work will focus on further improvement of accuracy of the alignments obtained by PaMSA using benchmark protein databases, such as BAliBASE, PREFAB and SABmark. Additional MSA methods, such as MaFFT, will also be considered for comparison. As for improvement in performance, more work remains to be done by studying and applying other parallel optimization techniques in order to obtain better response times. One of the main problems in the evaluation of MSA methods is that it is possible to obtain different MSAs having the same assessment score, making it difficult to discern which of them is the best, especially when aligning very dissimilar sequences. In this case, it is necessary to conduct a thorough analysis to achieve the best results in terms of accuracy. The long-term goal of the present work is to provide researchers with state-of-the-art algorithms and software tools that can help them advance in their field in a more efficient manner.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Cuauhtémoc López-Martín for his help in the statistical analysis, and Dr. Luis Delaye for his useful comments on the design of our algorithm.

REFERENCES

[1] D. Mount, *Bioinformatics: sequence and genome analysis*. New York: Cold Spring Harbor Laboratory Press, 2004.

[2] T. L. Bailey, "Discovering sequence motifs," in *Bioinformatics: Data, Sequence Analysis and Evolution*, ser. Methods in Molecular Biology, J. M. Keith, Ed. Totowa, New Jersey: Humana Press, 2008, vol. 452, pp. 231–251.

[3] J. D. Thompson, D. G. Higgins, and T. J. Gibson, "Clustal W: improving the sensitivy of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Research*, vol. 22, no. 22, pp. 4673–4680, 1994.

[4] C. Notredame, D. G. Higgins, and J. Heringa, "T-Coffee: a novel method for fast and accurate multiple sequence alignment," *J. Mol. Biol.*, vol. 302, no. 1, pp. 205–217, 2000.

[5] R. C. Edgar, "MUSCLE: multiple sequence alignment with high accuracy and high throughput," *Nucleic Acids Research*, vol. 32, no. 5, pp. 1792–1797, 2004.

[6] B. Morgenstern, "DIALIGN: multiple DNA and protein sequence alignment at BiBiServ," *Nucleic Acids Research*, vol. 32, pp. Web Server Issue W33–W36, 2004.

[7] Y. Bilu, P. K. Agarwal, and R. Kilodny, "Faster algorithms for optimal multiple sequence alignment based on pairwise comparisons," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3, no. 4, pp. 408–422, 2006.

[8] C. Notredame and D. Higgins, "SAGA: sequence alignment by genetic algorithm," *Nucleic Acids Research*, vol. 24, no. 8, pp. 1515–1524, 1996.

[9] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller, "A greedy algorithm for aligning DNA sequences," *Journal of Computational Biology*, vol. 7, no. 1/2, pp. 203–214, 2000.

[10] J. Pei and N. Grishin, "MUMMALS: multiple sequence alignment improved by using hidden markov models with local structural information," *Nucleic Acids Research*, vol. 34, no. 16, pp. 4364–4374, 2006.

[11] J. Zola, D. Trystram, A. Tchernykh, and C. Brizuela, "Parallel multiple sequence alignment with local phylogeny search by simulated annealing," in *Proc. IEEE Int. Workshop on High Performance Computational Biology (HiCOMB)*, 2006.

[12] J. Zola, X. Yang, S. Rospondek, and S. Aluru, "Parallel T-Coffee: A parallel multiple sequence aligner," in *Proc. of ISCA PDCS-2007*, 2007, pp. 248–253.

[13] C. B. Do and K. Katoh, "Protein multiple sequence alignment," in *Functional Proteomics: Methods and Protocols*, ser. Methods in Molecular Biology, J. D. Thompson, C. Schaeffer-Reiss, and M. Ueffing, Eds. Totowa, New Jersey: Humana Press, 2008, vol. 484, pp. 379–413.

[14] I. R. Andalon-Garcia, A. Chavoya, and M. E. Meda-Campaña, "A parallel algorithm for multiple biological sequence alignment," in *Information Processign in Cells and Tissues*, ser. Lecture Notes in Computer Science, M. Lones, S. Smith, S. Teichmann, F. Naef, J. Walker, and M. Trefzer, Eds. Cambridge, U.K.: Springer-Verlag, 2012, vol. 7223, pp. 264–276.

[15] R. Wagner and M. Fischer, "The string-to-string correction problem," *ACM*, vol. 21, no. 1, pp. 168–173, 1974.

[16] M. P. Styczynski, K. L. Jensen, I. Rigoutsos, and G. Stephanopoulos, "BLOSUM62 miscalculations improve search performance," *Nature Biotechnology*, vol. 26, no. 3, pp. 274–275, 2008.

[17] O. Gotoh, "Significant improvement in accuracy of multiple protein-sequence alignments by iterative refinement as assessed by reference to structural alignments," *J. Mol. Biol*, vol. 264, no. 4, pp. 823–838, 1996.