Efficient mapping of genomic sequences to optimize multiple pairwise alignment in hybrid cluster platforms

Alberto Montañola[1,*], Concepció Roig[1] and Porfidio Hernández[2]

[1]Department of Computer Science and Industrial Engineering, Universitat de Lleida, Lleida, Spain.
alberto.montanola@udl.cat, roig@diei.udl.cat

[2]Computer Architecture and Operating Systems Department, Universitat Autònoma de Barcelona, Cerdanyola del Vallès, Spain.
Porfidio.Hernandez@uab.es

Summary

Multiple sequence alignment (MSA), used in biocomputing to study similarities between different genomic sequences, is known to require important memory and computation resources. Nowadays, researchers are aligning thousands of these sequences, creating new challenges in order to solve the problem using the available resources efficiently. Determining the efficient amount of resources to allocate is important to avoid waste of them, thus reducing the economical costs required in running for example a specific cloud instance. The pairwise alignment is the initial key step of the MSA problem, which will compute all pair alignments needed. We present a method to determine the optimal amount of memory and computation resources to allocate by the pairwise alignment, and we will validate it through a set of experimental results for different possible inputs. These allow us to determine the best parameters to configure the applications in order to use effectively the available resources of a given system.

1 Introduction

In biology, a genomic sequence is a character string representing the codification of different basic structural elements known as amino-acids that will compose a specific protein with specific properties. There is the need for several applications to study the differences between different genomic sequences, for example in the aid to study the evolution of species and other usages in genomic research. In order to be able to perform this sequence comparison, we need to compare these character strings in order to find matching patterns between them. This comparison as shown in Figure 1, when performed with two sequences, is known as pair sequence alignment. In some cases comparing two sequences is sufficient, but in other cases there is need of comparing thousands of sequences. Then this problem is known as the Multiple Sequence Alignment (MSA).

The MSA problem [1] aims to find out the best possible alignment of all sequences. The steps composing the MSA process are illustrated in Figure 2, and can be described as follows:

*To whom correspondence should be addressed.

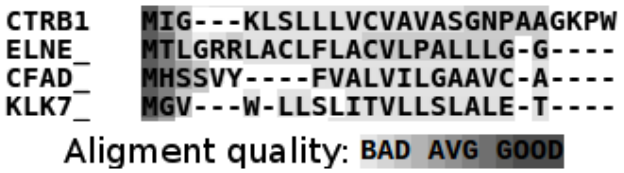

Figure 1: MSA: Multiple Sequence alignment example.

1. Read a list of genomic sequences to be aligned.
2. Pairwise alignment of all possible pairs of input sequences, calculating a similarity score for each pair.
3. Building a phylogenetic guide tree relating the sequences by similarity. The tree indicates the order for the subsequent multiple alignment.
4. Progressive alignment of the final tree to obtain the multiple alignment of all sequences.

The overall computation cost of MSA is exponential[2], being the pairwise alignment step the part which is determinant in this cost. Studying the scalability of MSA helps us to determine how much we can obtain from parallelizing MSA and to which limit we can take this parallelization into account. Thus, the generation of an efficient parallel implementation of the MSA process necessarily implies an improvement of the global pairwise alignment step.

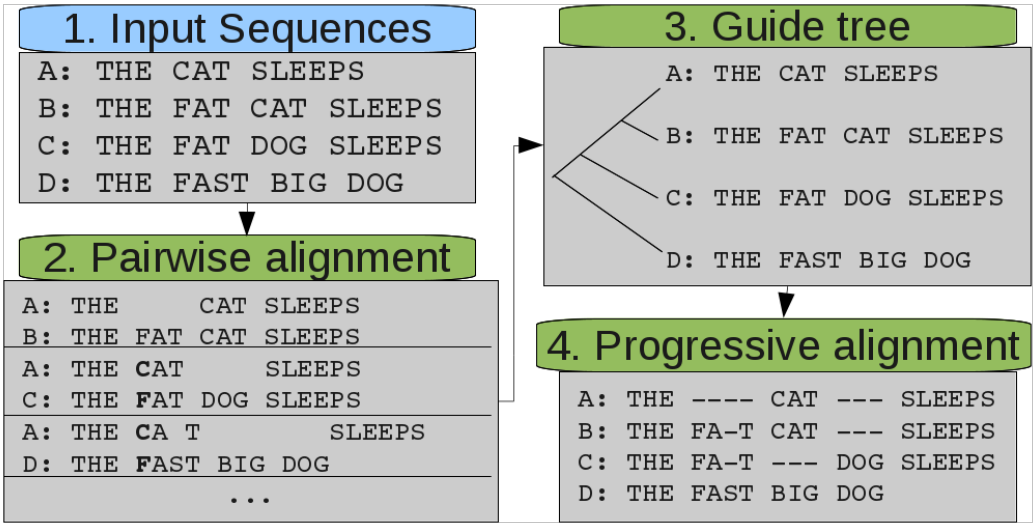

Figure 2: MSA: Multiple Sequence Alignment steps.

When one needs to solve a problem of a determined complexity, a set of several questions will arise on which amount of memory and computation resources should be given to the application. Determining the correct resources will ensure the efficient use of the available infrastructure resources assuring that more than necessary is not spent in the MSA process.

We are proposing a method that estimates the average resources needed to solve the problem, given a set of several varying input sequences, by changing different input parameters, we can

study the scalability and sensibility to computation time usage and how the overall resources are used. The implementation always reaches a scalability limit were by adding more resources to the system the overall computation time remains stable, thus we need to find the maximum amount of resources to give to the application in order to ensure an effective usage of them.

In this paper we study the impact of the initial pairwise alignment step when it is applied to a large number of sequences, and we focus on its improvement by studying its performance. The main goal of our work, consists in developing a method to determine the best configuration and tuning parameters that should be used to run the MSA on a specific hardware.

This article is an extended version of our previous contribution [3] and our other previous work [4], where we presented a hybrid programming solution, that combines the shared memory model with a distributed one, by expanding as many parallel threads as possible on a determined number of computing nodes.

In the present work, we carry out an experimental study to evaluate the performance and scalability of our mechanism according to the number and the length of sequences to be aligned. Additionally, we propose a method to estimate the number of computation resources needed to solve the problem, given a set of input sequences, in such a way that an efficient use of resources could be achieved. Finally, we compare the experimental data with the estimated prediction given by our proposed model.

The remainder of the paper is organized as follows: Section 2 reviews related work, Section 3 describes the used multiple pairwise implementation, Section 4 discusses the experimental results and finally, Section 5 presents our conclusions.

2 Related work

Sequence alignment is a widely used application in the biocomputing community, where several previous efforts have been done in order to solve the problem [5]. For the pairwise alignment of two single sequences, authors have proposed methods such as Smith-Waterman [6] and Needleman and Wunsch [7]. Although both methods build a comparison matrix to determine the best alignment, the first one prioritizes the alignment of local parts of the sequence, while the second one focuses on aligning the whole sequences.

When more than two sequences are involved, the problem complexity scales into MSA, where algorithms have to compute all possible combinations of pairs of sequences in order to achieve a final multiple alignment. Such implementations are for example T-Coffee [8], Dialign-TX [9], Muscle [10], ClustalW [11] and ClustalΩ [12] among several others.

In order to improve the performance of these MSA algorithms, some authors have proposed distributed memory solutions based on message passing (MPI), including Parallel-T-Coffee [13] and ClustalW-MPI [14] [15] that have considerably improved their speedup. They coincide in the need to align several pairs of sequences. Thus, they are focused on solving the problem using only a distributed memory paradigm and lack the benefits of a hybrid system (distributed/shared memory).

While the previous implementations will run several instances of the Smith-Waterman algorithm in parallel in order to align as many sequences as possible, there are also parallel implementations focused on the Smith-Waterman algorithm itself using shared memory such as [16], which describes an accelerated version using SIMD processor extensions, and [17] which describes an accelerated version running on GPU. These implementations are more specific in aligning a single pair of sequences as fast as possible rather than several pairs of them. Using both methods may benefit from a better hybrid implementation that would exploit and use the modern parallel systems, made up of nodes with multi-core CPUs and GPUs, more efficiently.

3 Multiple pairwise implementation

In this section we are going to briefly describe the algorithm used to carry out step 2 of pairwise alignment in the MSA process. The current version of the algorithm is available at `http://lescaffee.sevendeset.org`. We study its impact and efficiency. Furthermore, we are going to present a model used to determine the amount of resources required for its optimal use.

We assume a computing platform based on a hybrid architecture where cores in the same node have a shared memory access while different nodes act in a distributed memory fashion through message-passing. The proposed algorithm distributes the pairwise alignment work by mapping to each available core and node a partition of sequences from all possible combinations, by distributing the total computation time in the best uniform way across the system.

We consider a number of n sequences of similar length to be aligned. Then, the number of pairwise alignments to carry out will be $\frac{n^2-n}{2}$, which corresponds to the total number of pairs. Assuming a balanced distribution of all pairwise alignments into the cores of the system and considering that a single pair alignment is entirely done in a core, the total pairwise computation (t_{pwc}) can be calculated with expression (1), where pwt is the time needed to carry out a single pairwise operation and f is the total number of cores.

$$t_{pwc} = pwt \times \frac{n^2 - n}{2f} \tag{1}$$

The expected communications overhead (t_{comm}) of the message passing stack, for the distribution of their sequences to all nodes and the gathering of all the aligned pairs, can be calculated with expression (2), where ℓ is the average sequence length in bytes, bw the transmission speed in bytes per second and k is the number of nodes of the system.

$$t_{comm} = \frac{\ell \times n \times k}{bw} + \frac{\ell \times (n^2 - n)}{bw} \tag{2}$$

The total computation time t, given in (3), corresponds, at maximum, to the sum of total pairwise computation and the communication time. It has also to be added a value k_o that corresponds to the remaining application overhead used to compute the pairwise tasks and reading

sequences from disk. The communication time between the cores in the same node is considered to be negligible.

$$t = t_{pwc} + t_{comm} + k_o \tag{3}$$

Finally, the expected total amount of memory (mem) that will be used by our implementation can be defined by (4), were k_{sm} is the average size of required data structures for one pair of sequences and k_{th} the overhead penalty for each additional thread. In average, we consider the size in memory of two aligned sequences as 3ℓ, taking into account that the aligned sequences can add gaps inside. Additionally it has to be added the storage of the n original sequences for each of the k nodes.

$$mem = \frac{n^2 - n}{2} \times (3 \times \ell + k_{sm}) + f \times k_{th} + n \times \ell \times k \tag{4}$$

From equation (3), by making $k = f = 1$, we can calculate the speedup as the sequential computation time ($pwt \times \frac{n^2-n}{2}$) divided by the parallel computation time t. According to this, the efficiency is calculated with expression (5) as the obtained speedup with respect to the optimum one, that is of number of cores. This permits us to determine the expected number of resources needed for a specific input problem.

$$\textit{Efficiency} = \frac{pwt \times bw + 2 \times \ell}{pwt \times bw + 2 \times \ell \times f} \tag{5}$$

This means that the efficiency directly depends on the length of the input sequences and the computing time required to align one pair of sequences. From this expression, it can be calculated the recommended number of cores to achieve the desired efficiency.

The accuracy of the performance parameters previously defined, is proved for the implementation of the multiple pairwise alignment. Which will carry out the steps illustrated in Figure 3, by expanding the appropriate number of MPI processes and threads that yield an efficiency over 0.5. The functionality of these steps is the following:

1. Input sequences are parsed and loaded into the memory by a master task, and the system is queried to determine the total number of nodes, maximum memory available and number of available cores per node. Then we determine some system information, by testing the data transfer speed t_k and calculating a set of benchmarking pairs of sequences in order to calculate t_{pwt}. Then, using equation (5), we calculate the ideal f value to run the application and the required application memory using (4).
2. All possible pairs of sequences are generated and distributed.
3. Each process spawns a worker pool of c threads, where c is the total number of cores available in a given node.
4. Each thread carries out the Smith-Waterman algorithm for each pair of sequences that has been assigned to.
5. All threads and processes are joined and all aligned pairs are merged into a final list.

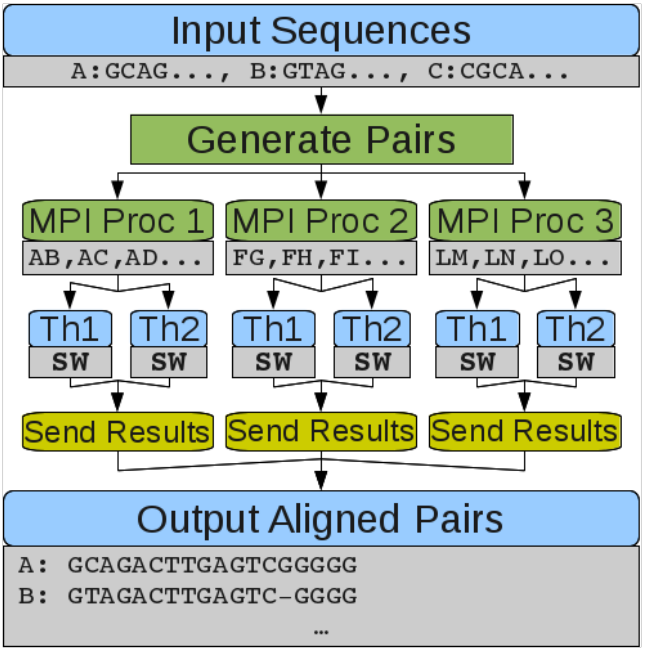

Figure 3: Parallel multiple pairwise alignment steps.

4 Experimental results

Different tests were performed in order to analyse the behaviour of our presented implementation, thus proving the correctness of the prediction model used to evaluate and determine the best parameters for a specific input. In these experiments we are measuring the total computation time and memory usage by the application and comparing it to the model that defines the expected behaviour of the experiments. Different sequence sets from 400 to 1400 sequences of lengths varying from 94 to 229 residues were used.

The first step is obtaining a set of input sequences of determined lengths used for the evaluation of the algorithm. The method used to obtain the sequences is to extract them from sets of sequences available from different public databases. The PFAM [18] database was used as the main input of sequences. The following sets were randomly obtained from PFAM database: PF07687, PF12704, PF00015, PF00149 and PF08242. On these sets, sequence lengths varies from 94 to 229 residues. Furthermore, using these sets as our source for sequences, we obtain smaller sets from 400 to 1400 sequences for being used as the input sets of our implementation.

The platform used to perform executions consists of a cluster composed by 24 nodes of Intel Quad Core processors running at 2.4 GHz with 8 Gigabytes of RAM each one. Moreover, the configuration used for each execution consists in running a fixed number of MPI processes, one per node, expanding as many threads as available cores on the node. For example, one execution may consist on expanding 5 MPI processes in 5 nodes with 4 threads each one, thus achieving a total number of 20 threads.

In all samples, the average processing time is calculated by measuring the total time required by the application since it loads the sequences from disk until all the results are written back to

disk by the master process. The total memory usage is measured on each node by measuring all memory allocations by providing replacements to the standard allocation library. The displayed results show the total system memory by adding the memory measures of each node.

Figure 4 shows the memory usage for different input sets of sequences, from 400 to 1400 of size 229 residues varying the number of threads/nodes used in the execution. It can be observed that in all the cases, incrementing the number of threads have an impact of increasing the memory about 50MB from the sequential execution to the 96 threads one.

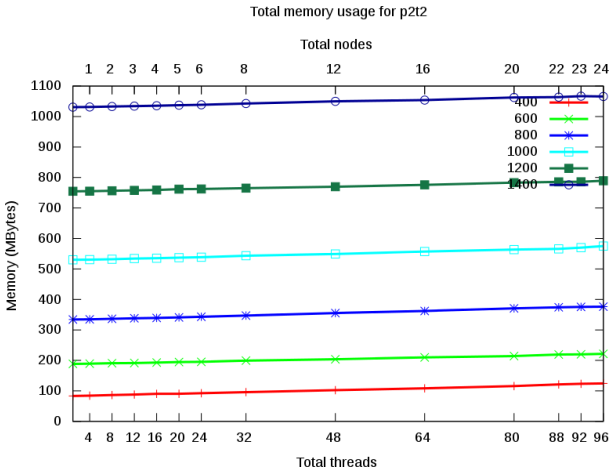

Figure 4: Memory usage.

By applying the previously seen expression (4), considering $k_{sm} = 400\ bytes$ and ignoring k_{th}, we determine the expected amount of memory consumption for 400 to 1400 sequences of 229 residues. These results are compared with experimental ones on Table 1. As can be seen, the values that we calculated can successfully predict the behaviour of the algorithm in the experimentation.

Table 1: Comparison of experimental memory usage versus the predicted one.

Input n	Memory MB	
	Experimental	Predicted
400	94	92
600	196	196
800	343	341
1000	539	527
1200	762	755
1400	1038	1025

Figure 5 shows in logarithmic scale the execution time, in seconds, with the following graphs: t_{pwc}: the execution of Smith-Watermanm, t_{comm}: the communications time, and $total$: the total time. The simulated graphic is generated from data gathered from the prediction algorithm

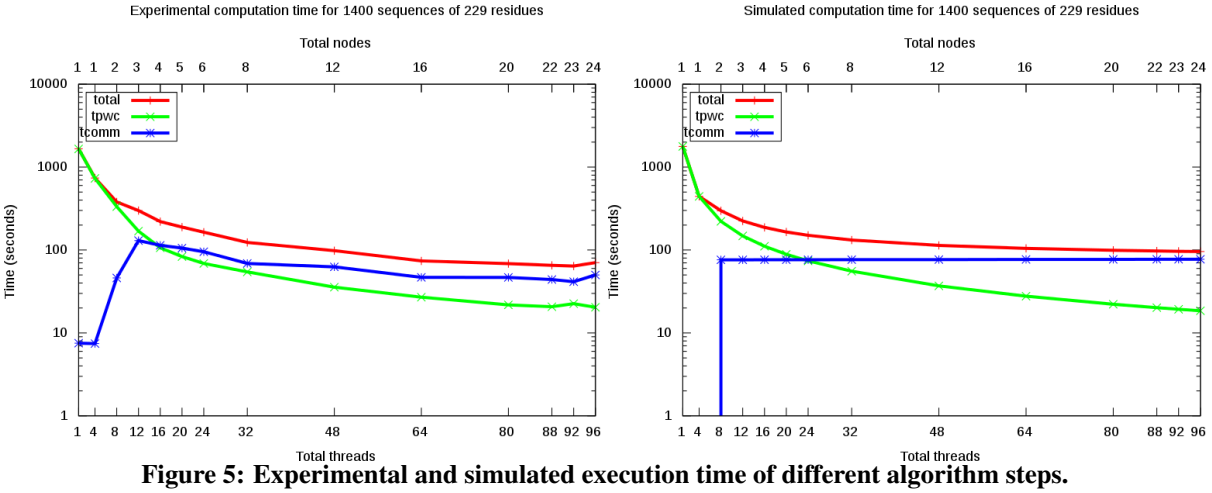

Figure 5: Experimental and simulated execution time of different algorithm steps.

using the aforementioned equation models. As it can be observed, the simulated graphs are a good prediction of the behaviour of the algorithm. Communications overhead stays stable on both graphs, being constant in the simulation because the time to transfer $\frac{n^2-n}{2}$ pairs of sequences is fixed indifferently on the number of nodes.

Figure 6 shows in logarithmic scale the computation time, in seconds of step 4, corresponding to the execution of Smith-Waterman in each core. It is also shown the corresponding SpeedUp. On the first row of graphs, we varied the number of sequences and fixed the length of them to 229 residues, and on the second one, we varied the length of them from 94 to 229 with a fixed number of 1400 sequences. As it can be observed the computation time increases in proportion of the number of sequences to be aligned and the amount of residues. In all the cases, increasing the number of threads entails a reduction of computation time. As we can see in the SpeedUp graph, it shows how the algorithm correctly scales, meaning the computation balance is correctly distributed among the nodes.

In the same way, Figure 7 shows the computation time and the corresponding SpeedUp for a fixed number of sequences ($n = 1400$), varying the length of them from 94 to 229 residues. As can be observed, the length has a similar impact, in computation time, since time increases with increasing length, and it decreases when the number of threads and nodes increase. The SpeedUp, shows how the sequence length also scales for longer sequences.

Figure 8 shows the average time invested in the communications layer of MPI, for different numbers of sequences, varying the number of nodes. It can be seen that the time invested in communications is proportional to the number of sequences as it is also established in equation (2). However, it can be observed in the experiment that this time is stabilized from 8 nodes to 24. Indifferently of the number of nodes, this time tends to remain stable and depends on the problem size, as it will determine the amount of data to transfer. This is displayed as constant line on the right graphs.

Finally, Figure 9 shows the sum of total computation time required by the expression (3). We can see that the time stabilizes to a certain point were adding more nodes will not improve the overall time, thus we can see that the impact of the communications are affecting negatively to the Efficiency endangering the scalability of the algorithm. From the prediction expression (5),

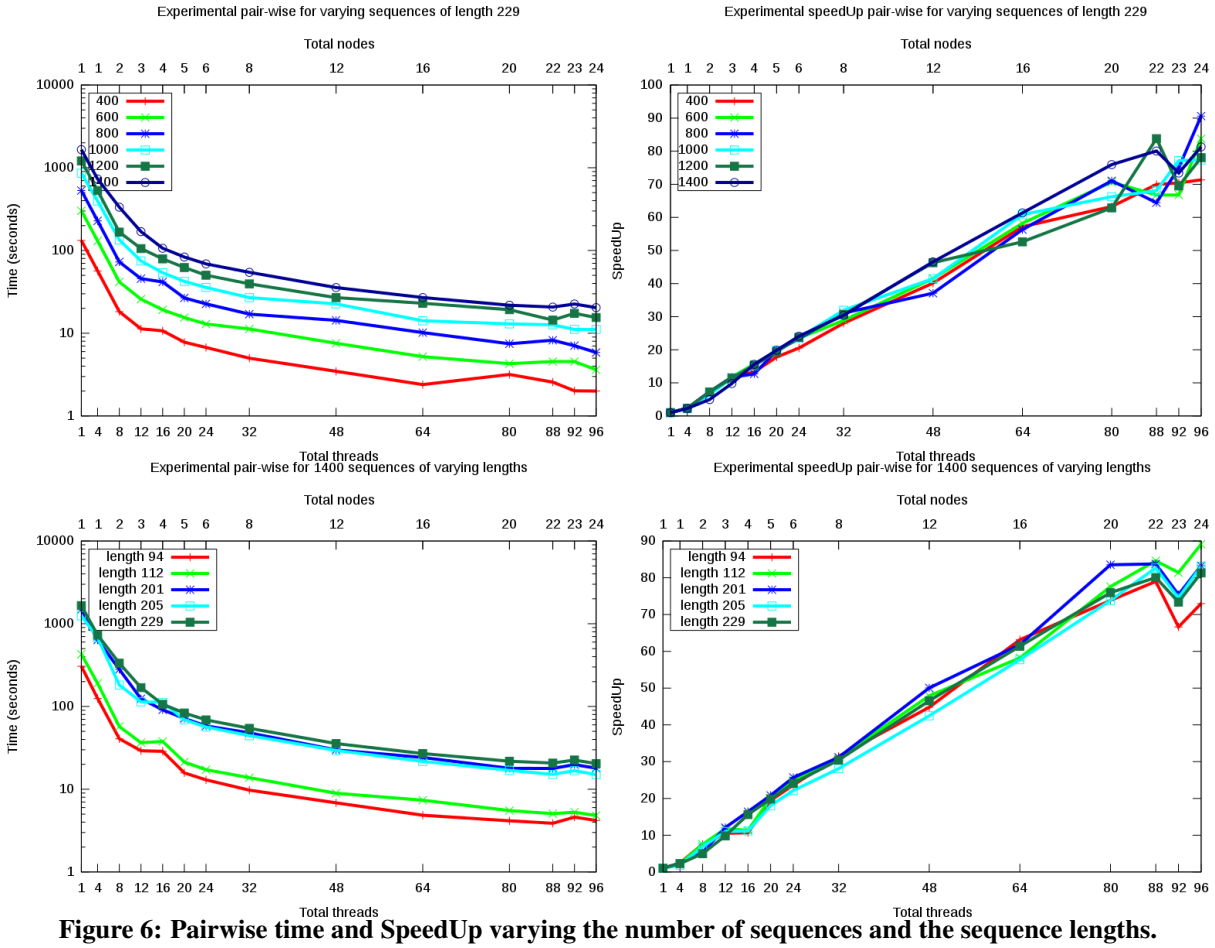

Figure 6: Pairwise time and SpeedUp varying the number of sequences and the sequence lengths.

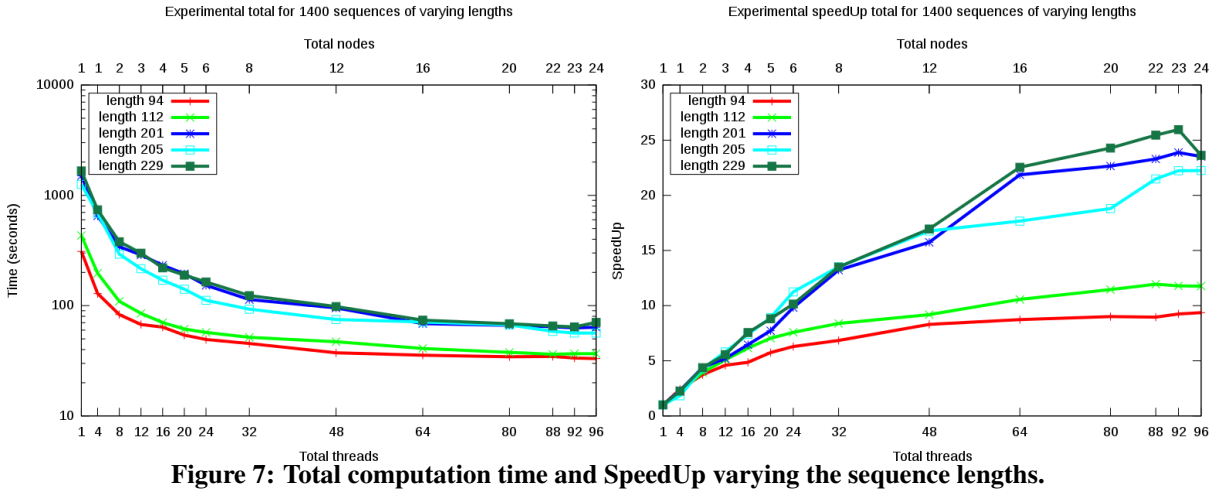

Figure 7: Total computation time and SpeedUp varying the sequence lengths.

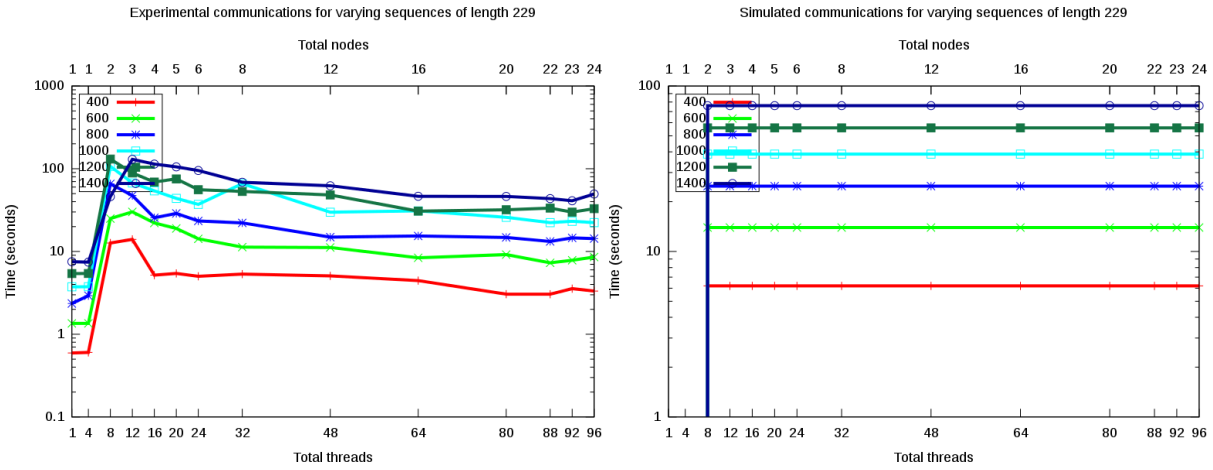

Figure 8: Communications overhead varying the number of sequences.

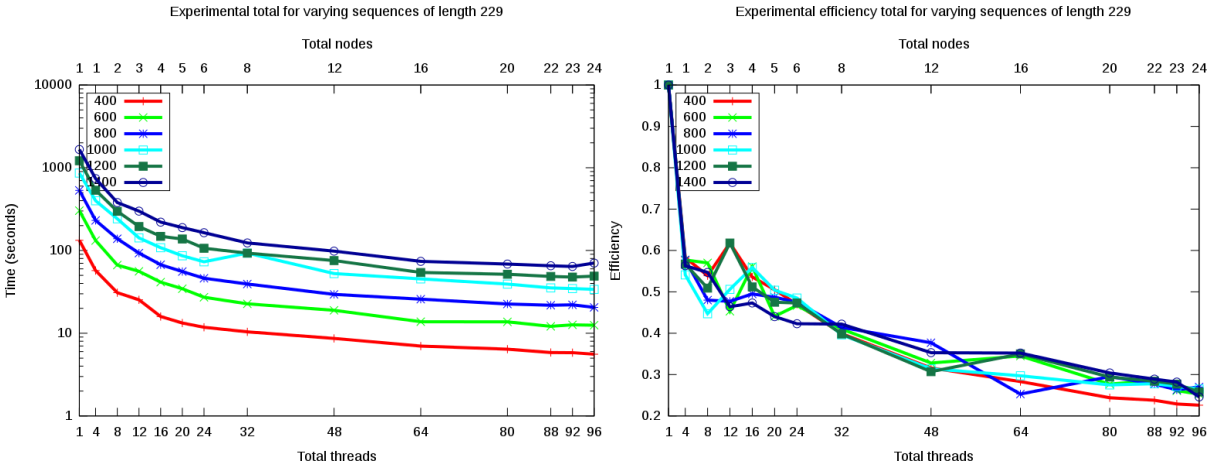

Figure 9: Total pairwise time and Efficiency varying the number of sequences.

with a target efficiency of 0.5, we determined that the best number of nodes for 400 sequences is 20, between 600 and 1000 is 24, for 1200 is 28, and lastly for 1400 is 32 cores. Thus, we can see that experimental results are not so far from the expected ones.

5 Conclusions

The pairwise sequence alignment, as a key step in the Multiple Sequence Alignment problem, will influence in the overall performance of the problem, being the step with more computing resources required. As an essential step towards MSA, we should find new methodologies and techniques to improve and to parallellize it. Furthermore, we need a way to predict which configuration is the best one for a specific input, in order to be able to run the given algorithm with the most efficient parameters, to use the resources in the best possible way.

This paper presents a new implementation based on the Smith-Waterman pairwise algorithm, aims to study its scalability and behaviour, and finds a way to predict the best parameters that

should be used. Thus, we present a hybrid implementation using a threading library combined with a message passing one to exploit the available resources as best as possible. Finally, we obtain a set of results that prove that the algorithm behaviour can be predicted by the defined equations.

Acknowledgements

This work was supported by the MEyC-Spain under contract TIN 2011-28689-C02-02 and Consolider CSD2007-0050. The CUR of DIUE of GENCAT and the European Social Fund.

References

[1] R. C. Edgar and S. Batzoglou. Multiple sequence alignment. *Current opinion in structural biology*, 16(3):368–373, 2006.

[2] L. Wang and J. Tao. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(3):337–348, 1994.

[3] A. Montañola, C. Roig and P. Hernández. Optimizing multiple pairwise alignment of genomic sequences in multicore clusters. In *PACBB*, pages 121–128. 2014.

[4] A. Montañola, C. Roig and P. Hernández. Pairwise sequence alignment method for distributed shared memory systems. *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 0:432–436, 2013.

[5] J. Daugelaite, A. O' Driscoll and R. D. Sleator. An overview of multiple sequence alignments and cloud computing in bioinformatics. *ISRN Biomathematics*, 2013(615630):14, 2013.

[6] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[7] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 3(48):443–53, 1970.

[8] C. Notredame, D. Higgins and J. Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–17, 2000.

[9] A. R. Subramanian, M. Kaufmann and B. Morgenstern. DIALIGN-TX: greedy and progressive approaches for segment-based multiple sequence alignment. *Algorithms for Molecular Biology*, pages 3–6, 2008.

[10] R. Edgar. MUSCLE: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 5(32):1792–1797, 2004.

[11] M. Larkin, G. Blackshields, N. Brown et al. ClustalW and ClustalX version 2. *Bioinformatics*, 21(23):2947–2948, 2007.

[12] F. Sievers, A. Wilm, D. Dineen et al. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology*, 7(1):539, 2011.

[13] J. Zola, X. Yang, S. Rospondek and S. Aluru. Parallel T-Coffee: A parallel multiple sequence aligner. *In Proc. of ISCA*, pages 248–253, 2007.

[14] K.-B. Li. ClustalW-MPI: ClustalW analysis using distributed and parallel computing. *Bioinformatics*, 19(12):1585–1586, 2003.

[15] J. Cheetham, F. Dehne, S. Pitre, A. Rau-Chaplin and P. Taillon. Parallel CLUSTAL W for PC Clusters. In V. Kumar, M. Gavrilova, C. Tan and P. LEcuyer (editors), *Computational Science and Its Applications ICCSA 2003*, volume 2668 of *Lecture Notes in Computer Science*, pages 300–309. Springer Berlin Heidelberg, 2003.

[16] M. Farrar. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, 2(23):156–161, 2006.

[17] Y. Liu, D. Maskell and B. Schmidt. CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. *BMC Research Notes*, page 2:73, 2009.

[18] M. Punta, P. Coggill, R. Eberhardt et al. The Pfam protein families database. *Nucleic Acids Research*, 40:D290–D301, 2012.