PARALLEL PROCESSING DNA SEQUENCES ON MULTICLUSTER AND GRID ARCHITECTURES. SOFTWARE OVERHEAD*

Franco Chichizola[1], Marcelo Naiouf[2], Laura De Giusti[3], Ismael Rodríguez, Armando De Giusti[4]
{francoch, mnaiouf, ldgiusti, ismael, degiusti}@lidi.info.unlp.edu.ar

Instituto de Investigación en Informática (III-LIDI) – Facultad de Informática – UNLP

Summary

A DNA sequence analysis parallelization in large databases using cluster, multi-cluster, and GRID is presented.

Achievable speedup, scalability, and overhead introduced by communications are discussed, and the impact of the Grid middleware on the performance obtained with clusters is detailed.

The experimental work carried out with homogeneous and heterogeneous clusters is presented, along with a comparison of the results obtained when migrating the algorithms to a GRID.

Finally, current lines of work related to the study of models and paradigms for the resolution of parallel algorithms on GRID architectures are presented.

Key words: Parallel Algorithms, Sequence Alignment, Distributed Databases, Clusters, Multi-clusters, GRID.

[1] Full-Time Associate Professor
[2] Full-Time Head Professor
[3] Full-Time Associate Professor
[4] CONICET Principal Researcher, Full-Time Head Professor

* This paper is partly financed by CIC, the CyTED GRID Project, and YPF Foundation.

1. INTRODUCTION

The evolution of processing towards parallelism has been evident almost from the very appearance of digital computers. There are multiple factors that have driven software concurrence and hardware multi-processing, but there are two that should be highlighted:

- ✓ The need to reduce the processing time of large volumes of data (numerical problems, models, large databases, images, expert systems, biotechnology, etc.)
- ✓ Information processing (data, signals) in real time to make decisions both in industrial and administrative environments (robotics, military industry, multimedia systems in real time, geo-referencing, pattern recognition)

This evolution causes a great effort to be done to transform sequential processing into parallel processing in an attempt to reduce processes execution times and response times to real-world events [1]. Within this scope, there are also two main factors that guided technological evolution:

- ✓ The growth of computational power, given by the evolution of the technology used in components and processing architectures (supercomputers, hypercubes of homogeneous processors, large networks of non-homogeneous processors, processors specialized in images, processors for the treatment of signals)
- ✓ The transformation and creation of algorithms that exploit the implicit concurrence of the problem at hand to its maximum, so as to distribute processing tasks and minimize total response time. Naturally, this transformation should also be adapted to the physical support architecture.

One of the areas of greatest interest and growth in the last few years within the field of parallel processing applications is that of the treatment of large volumes of data such as DNA sequences, which are actually the driver for this application. The type of extensive comparison processing carried out to analyze genetic patterns (in a database that can even be physically distributed and partially replicated) requires a significant effort in the development of efficient parallel algorithms.

DNA is the biological element that differentiates species, or the so-called "types". Therefore, DNA sequences profiling is carried out as a worldwide effort. With the development of techniques that allow unraveling the information contained in DNA, conditions were favorable for the emergence of bioinformatics, which is a branch that seeks not only to acquire, store and organize the biological information contained in DNA molecules, but also to analyze and interpret these data. It involves the resolution of complex problems using tools provided by computational systems. The diagnosis and treatment of medical conditions, the production of genetically enhanced foods, or the identification of living beings focusing on traceability or paternity systems are, among others, major applications in the area. The more complete the reference population stored in the database is, the higher the certainty of the analysis will be.

In this paper, the parallelization of an algorithm to analyze similarities in DNA sequences in large databases using distributed architectures such as clusters and GRIDs is analyzed.

2. PARALLEL ARCHITECTURES

The use of architectures such as clusters, multi-clusters, and GRIDs, communicated through messages and supported by networks with different characteristics and topologies has become generalized, for the development of both parallel algorithms and distributed Web services [2][3].

A *cluster* is a type of distributed processing system formed by a set of interconnected *stand-alone* machines that work co-operatively as a sole and integrated computation resource. When connecting two or more clusters over a LAN or WAN, some form of *multi-cluster* is obtained.

Different multi-cluster variations are obtained if all clusters are on the same network or if they are connected over various networks; if the operating system support is common to all components; if each cluster is homogeneous or heterogeneous; if the communications network has the same bandwidth between all nodes or if bandwidth is variable (typical of a WAN over Internet); and if each cluster is dedicated to the application defined for the multi-cluster or shares it with other tasks.

A *GRID* is a type of parallel distributed system that allows sharing, selecting and adding geographically distributed autonomous resources, such as computers, software, data, databases, special devices, instruments, and people. This collaborative configuration depends on the availability, capacity, cost, and requirements of the user [4]. A GRID can also be defined as a virtual information processing environment where the user has the “illusion” of an only and powerful computational resource which is actually distributed [5].

Some of the characteristics of a GRID environment are [6]:

- ✓ The GRID integrates resources (processors, instruments, databases, etc.) that are heterogeneous, geographically distributed and in general connected over a WAN, and which can join or leave the GRID dynamically.
- ✓ Resources can be accessed *on-demand* by a set of users who are part of a virtual community.
- ✓ The GRID is configured with general-purpose protocols and infrastructure, not necessarily common to all its nodes.

If we consider the functionalities of a "layered" GRID structure, the following can be mentioned:

a. The lowest level, consisting on GRID services (“Factory layer”) to support the use of local resources (processors, data, network).
b. The layer that provides authentication and security services to allow the exchange of data between remote resources (“Connective Layer”).
c. The resource administration layer (“Resource Layer”), which allows sharing resources and establishing a logical connection between them.
d. The collective layer is the one that coordinates the interactions between multiple resources associated to physically distributed processes.
e. Finally, the application layer is the one in charge of the interaction with the user, so that users can “transparently” visualize the virtual configuration with which they will work.

Some authors consider that a GRID is a “Cluster of Clusters,” which results in a somewhat restrictive definition, but one that is useful for the evolution of parallel applications from clusters to GRIDs. [7]. Some similarities and differences can be mentioned:

- ✓ In the case of clusters, usually an only virtual parallel machine is configured, and it can run a dedicated application. A GRID allows configuring multiple virtual parallel machines for several simultaneous users/applications.

- ✓ Both clusters and GRIDs are based on heterogeneous processors. However, in the case of GRIDs, this heterogeneity is extended to the communications network and the type of components in each node, which can be processors, instruments, sensors, etc.
- ✓ The middleware required for GRIDs is more complex than that of clusters. Basically, in order to configure the virtual parallel machine, a stage to identify and locate physical resources is required. Also, GRIDs require monitoring the execution of tasks over multiple virtual machines with different levels of users who have different access rights to resources.
- ✓ In addition to this, the tools to develop applications require a greater level of abstraction in GRIDs, due to the complexity and diversity of the multiple users that can use the architecture.

It should be mentioned that a multi-cluster structure, visualized as a *limited number of dedicated clusters that co-operate in an only parallel application*, is an intermediate point between clusters and a GRID, and it will require some special services in its middleware (specially to authenticate user rights of users accessing remote resources) [8].

There are currently many application areas for these architectures, such as: scientific computation, simulation, industrial models, medicine, e-commerce, distributed database management, Internet (portals, Web services), E-Government, or critical applications such as nuclear reactors, banks, military weapons or industrial control in real time [9].

3. SEQUENCE SIMILARITY IN DATABASES

There are currently many databases with information related to DNA sequences. These databases (centralized or distributed) record specific properties of the sequences to be used in different applications. All of them share the characteristic of being composed by thousands of records of DNA sequences that grow exponentially year after year.

With this information, the characteristics of a new sequence (*test*) can be predicted by comparing it to the sequences stored in the database. There is a set of methods that allow quantifying the similarity ratio between the *test* sequence and each stored sequence, in order to search in the database for all those sequences that fulfill a certain similarity criterion with *test* [10].

The Smith-Waterman algorithm for local alignment is one of these methods; it focuses on similar regions only in part of the sequences, which means that the purpose of the algorithm is finding small, locally similar regions. This method has been used as the basis for many subsequent algorithms and is oftentimes used as basic pattern to compare different alignment techniques.

The complexity of the Smith-Waterman algorithm to carry out the comparison between each pair of sequences increases with sequence size [11]. When considering that this comparison is performed for each of the sequences stored in the database, a sequential execution on a "standard" computer becomes impracticable as the size of the database increases.

3.1. Smith-Waterman Algorithm

This method allows aligning two DNA sequences by inserting *gaps* (if necessary) that are used to detect locally similar regions that may indicate the presence of a relation between both sequences, which is done by assigning a similarity score.

The algorithm calculates a similarity score between two sequences and then employs a backwards alignment for an optimal result. This last part is not needed in this case, since only a similarity score between *test* and each of the sequences in the *database* is required, and, based on these scores, determine which is the most similar sequence [10].

The following paragraphs explain the operation of the algorithm to find a similarity score between two DNA sequences.

Given two sequences, A and B:

$$A = a_1 a_2 a_3 \ldots . a_M \qquad B = b_1 b_2 b_3 \ldots . b_N$$

a matrix *H* is built with *(N+1)x(M+1)*, so that nucleotide bases that form sequence *A* label the rows (starting from 1), and the bases that form *B* label the columns (starting from 1). The following steps are applied to calculate the values of *H* that will yield the similarity score between *A* and *B*:

a. Start with 0 on row 0 and column 0 of *H*.

b. Calculate the value of H_{ij} for $\forall i \in [1,..,N]$ and $\forall j \in [1,..,M]$ by means of function 1. This value indicate the maximum similarity between two segments ending in a_i and b_j respectively.

$$H_{i\,j} = \max\begin{cases} 0 \\ H_{i-1,j-1} + V(a_i, b_j) \\ C_{i\,j} \\ F_{i\,j} \end{cases} \tag{1}$$

where

$$C_{i\,j} = \max_{0 \le k \le i}\{H_{i-k,j} - g(k)\} \tag{2}$$

$$F_{i\,j} = \max_{0 \le l \le j}\{H_{i,j-l} - g(l)\} \tag{3}$$

$$g(x) = q + rx \qquad (q \ge 0; r \ge 0) \tag{4}$$

At this point, $V(a_i, b_j)$ is the matches function that indicates the score given by matching a_i and b_j, which is based on a substitution matrix; C_{ij} is the score obtained considering a *gap* in *j* columns that is calculated with equation 2; F_{ij} is the score obtained considering a *gap* in row *i* and is calculated with equation 3; *g(x)* is the penalty function for a *gap* of length *x* and is obtained with equation 4; and *q* is the penalty applied for opening a *gap*, whereas *r* is the penalty corresponding to its prolongation.

c. The similarity score is obtained as shown in equation 5.

$$P = \max_{(0 \le i \le N)(0 \le j \le M)}\{H_{i\,j}\} \tag{5}$$

4. SEQUENTIAL AND PARALLEL ALGORITHM

For this experiment, the sequences considered are of different size and are represented by arrays of characters, where the alphabet is formed by the 4 nucleotide bases (A, C, G, T). The database is formed by *K* sequences that are stored in one file, and the *test* sequence is in a separate file.

4.1. Sequential Solution

The sequential algorithm loads the *test* sequence in an array; then, for each sequence S_i in the database, it loads S_i in an array and obtains the *Similarity Score* (P_i) between *test* and S_i by means of the Smith-Waterman algorithm that was explained in Section 3.1, storing in P_{max} the maximum Similarity Score obtained so far, and in S_{max}, the sequence number that yielded this value. Figure 1 shows a pseudo-code of this solution.

```
int Look_for_Similarity (Base, ArchTest)
 { int $S_{max}$, $P_{max}$, $P$
   char *test, *S

   $P_{max} = -1;$
   test = Load_Sequence (ArchTest, 0)
   for (i=0; i<K; i++)
         S = Load_Sequence (Base, i)
         P = Smith_Waterman (test, S)
         if  ($P > P_{max}$)
               $P_{max} = P$
               $S_{max} = i$
    return $S_{max}$
 }
```

Figure 1: Pseudo-code of the sequential solution

4.2. Parallel Solution with Semi-Dynamic Distribution

For the parallel solution carried out, it is assumed that there are *B+1* processors (one acting as *master* and *B* as *workers*). The database has *K* sequences to be compared with the *test* sequence. This base is replicated in each of the *worker* processors, whereas the file with the *test* sequence is only stored at the *master* processor.

Taking into account the possible heterogeneity of processors and the various lengths of the different sequences in the database, workload is semi-dynamically distributed between processors. This technique allows distributing a percentage (*LI*) of the workload (sequences of the database that will be compared with *test*) statically at the beginning of the application, and then distributing workload on demand as the *workers* finish their job. The operation of this method is explained below:

✓ The *master* processor (b_0) distributes the initial sequences:

- b_0 calculates the number (*KI*) of sequences that should be processed statically when the operation starts based on the percentage (*LI*) of initial distribution, as shown by equation 6.

$$KI = \frac{K \times LI}{100} \tag{6}$$

- b_0 obtains the number of sequences (k_i) that each *worker* should initially process based on their computational power (w_i), as shown by equation 7.

$$powerTot = \sum_{i=1}^{B} w_i \qquad\qquad k_i = \frac{KI \times w_i}{powerTot} \tag{7}$$

- b_0 distributes the initial *KI* sequences by assigning k_i consecutive sequences to b_i, with $i = 1..B$. In order to assign a set of sequences to work with, it indicates the position of the first and the last sequence within the database.

✓ The master processor (b0) distributes the remaining KR sequences, where initially, KR = K – KI:

- b_i requests b_0 for more sequences to solve upon finishing its assignment. For any $i = 1..B$.
- If $KR > 0$, b_0 assigns *KB* (or *KR* if $KR<KB$) consecutive sequences to the b_i processor that sent the request, being *KB* the number of sequences sent for each assignment request.
- If $KR > 1$, b_0 updates *KB* as shown in equation 8.

$$KR = KR - KB \quad (8)$$

Figure 2 shows how the distribution is carried out assuming a system composed by four *workers* in heterogeneous processors whose relative computational powers are w_1=3, w_2=3, w_3=2, w_4=2. The value of *LI* = 50 %, and that of *KB* = 5.

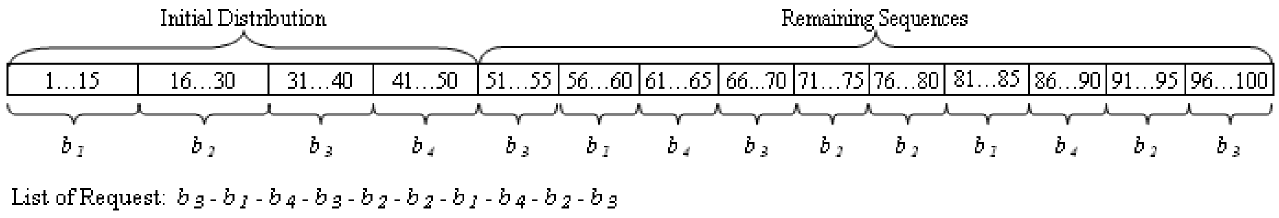

Figure 2: Semi-dynamic distribution

The following paragraphs describe how processes (master and workers) operate in the proposed parallel solution, taking this technique into account to distribute workload among processors. Processes act like this:

✓ *Master* process:

- It loads the *test* sequence in an array and sends it to all *workers*.
- It carries out the *Initial Distribution*, indicating each *worker* the block of sequences from the database that were assigned to it.
- It distributes the remaining sequences on demand.
- It informs all *workers* when there is no more work to be done so that they send back the partial results obtained (maximum similarity score and corresponding sequence).
- It calculates the maximum score from all partial results, thus obtaining the sequence that is "most similar" to *test*.

✓ *Worker* processes:

- They receive the *test* sequence from the *master* process.
- They receive the position on the database of the first and the last sequences that they should process corresponding to the *Initial Distribution*.
- For each sequence S_i that they should process, they load S_i in an array and obtain the *Similarity Score* (P_i) between *test* and S_i by applying the Smith-Waterman algorithm explained in Section 3.1, storing in P_{max} the maximum Similarity Score so far, and in S_{max} the number corresponding to the sequence that yielded that value (from among all sequences assigned to them).
- While the *master* process does not inform them that there is no more work to be done, they receive the position of another block of sequences and analyze them as described above.
- When receiving an end-of-work indication, they send the *master* process the maximum Similarity Score and the position in the database corresponding to the sequence that yielded that value.

5. EXPERIENCE CARRIED OUT

The experience carried out is divided in two parts. The first part consists in assessing the performance of the parallel algorithm over a multi-cluster architecture. The second part is focused on studying the overhead generated when migrating the application to a GRID architecture.

5.1. Performance Assessment on a Multi-Cluster Architecture

The heterogeneous multi-cluster architecture used in this experience is formed by two clusters whose processors have different characteristics:

- *Cluster1*: 16 processors (2.4 GHz Pentium 4 and 1GB RAM memory each one).
- *Cluster2*: 4 processors (2 GHz Celeron and 128 MB RAM memory each one).

Communication within each cluster is carried out by means of an Ethernet network, and a switch is used for inter-cluster communication.

The language used for implementations is C with the MPI library to manage communications between processes [12].

For this experiment, four databases with 10,000 sequences each are generated. The difference between these databases is the maximum size of their elements (1000, 2500, 5000, and 10,000); a corresponding *Test* sequence is generated for each size. Within each database, sequence length varies between 75% and 100% of the maximum size. Different tests were performed varying the following:

- The subset of processors in each cluster
- The size of the sequences in the database and the *Test* sequence
- The number (*K*) of sequences from the database that will be taken into account (1000, 2500, 5000 and 10,000)
- The percentage (*LI*) of initial distribution for the semi-dynamic distribution (0, 25, 50, 75, 100)
- The number of sequences (*KB*) that are distributed on each workload request (1, 5, 10, 20, y 50).

In order to simplify result visualization in this paper, the value of *K* (10,000), *LI* (50) and *KB* (10) were kept constant. Tests are described taking these values into account; the characteristics corresponding to each of the tests are detailed in Table 1. The results of all tested combinations are reported in [13].

Test	Number of Workers		Sequence Size
	Cluster 1	Cluster 2	
1	3	1	750 – 1000
2	6	2	750 – 1000
3	12	4	750 – 1000
4	3	1	1750 – 2500
5	6	2	1750 – 2500
6	12	4	1750 – 2500
7	3	1	3750 – 5000
8	6	2	3750 – 5000
9	12	4	3750 – 5000
10	3	1	7500 – 10000
11	6	2	7500 – 10000
12	12	4	7500 – 10000

Table 1. Details of the tests performed.

5.2. Assessment of the Overhead Generated by a GRID Architecture

To analyze the behavior of the application upon migration to a GRID environment, the following architectures were used:

- *Cluster*: subset of 6 processors of *cluster* 1 mentioned before.

- *Grid*: 6 processors (2.66 GHz Pentium 4 and 512 MB RAM memory each one). All of them configured as GRID nodes using Globus Toolkit 4.0.7 as GRID Middleware.

The communication between the processors forming the *GRID* is carried out by means of a Fast Ethernet (100 Mbps) network.

The language used for implementations is C with the MPI library (LAMMPI for *Cluster* and MPICH-G2 for *Grid*) to manage communications between processes [12].

To perform this analysis, three of the databases described in the previous section (5.1) with their corresponding *Test* sequences are used, and a set of tests varying the same parameters are defined. Each of these tests is carried out in both architectures (*Cluster* and *GRID*) in order to be able to compare their behaviors. As in the previous section (5.1), to simplify result visualization, the subset detailed in Table 2 is shown, where the values of *K* (10,000), *LI* (50) and *KB* (10) are kept constant. The results of all tested combinations are reported in [13].

Test	Number of Workers	Sequence Size
1	2	750 – 1000
2	3	750 – 1000
3	4	750 – 1000
4	5	750 – 1000
5	2	1750 – 2500
6	3	1750 – 2500
7	4	1750 – 2500
8	5	1750 – 2500
9	2	3750 – 5000
10	3	3750 – 5000
11	4	3750 – 5000
12	5	3750 – 5000

Table 2. Details of the tests performed.

6. METRICS AND RESULTS OBTAINED

This section presents the metrics used to measure the performance of the parallel system with the different architectures, together with the results obtained with the tests described in the previous section.

6.1. Speedup and Theoretical Speedup Analysis

In order to analyze the performance of the algorithm in the parallel architecture, speedup is used, whose value is obtained by means of equation 9.

$$Speedup = \frac{SequentialTime}{ParallelTime} \qquad (9)$$

In the case of heterogeneous architectures, the "*Sequential Time*" is given by the time of the best sequential algorithm run at the machine with greatest computational power [14][15][16]. Figure 3 shows a summary of the speedup obtained in the tests carried out.

In order to assess the level of speedup obtained, it is compared with the theoretical speedup of the architecture on which the test was run. This theoretical speedup considers the relative computational power of each machine with the power of the most powerful machine [17][18]. Theoretical speedup is calculated by means of equation 10.

$$TheoreticalSpeedup = \sum_{i=1}^{B} w_i \tag{10}$$

where B is the number of machines that form the architecture used.
w_i is the relative computational power of machine i as compared with the power of the most powerful machine. This relation is expressed by equation 11.

$$w_i = \frac{SequentialTime(MostPowerfulMachine)}{SequentialTime(m_i)} \tag{11}$$

Figure 4 shows the relation between real and theoretical speedup for the tests described in Section 5.1. This relation is equivalent to efficiency in homogeneous architectures, and is obtained by means of equation 12.

$$\mathrm{Re}\,lación = \frac{Speedup}{TheoreticalSpeedup} \tag{12}$$

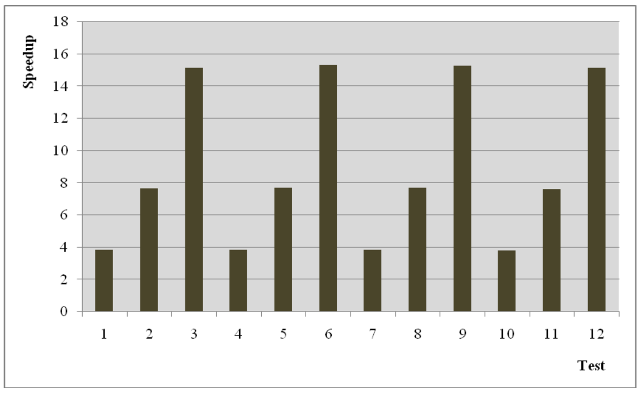

Figure 3: Speedup of the tests performed.

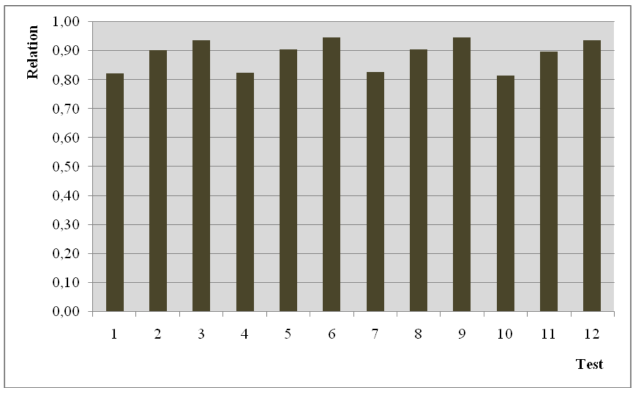

Figure 4: Relation between real and theoretical speedup for the tests performed.

6.2. Analysis of the Overhead Generated by the GRID Architecture

Since hardware characteristics of both architectures (*Cluster* and *GRID*) used for this analysis are different, the overhead generated by them cannot be measured by comparing their corresponding parallel execution times. For this reason, the metrics of *Efficiency* is used to compare both architectures.

As previously mentioned, *Efficiency* in homogeneous architectures is equivalent to the relation defined by equation 12 for heterogeneous systems. This value is obtained by means of equation 13.

$$Efficiency = \frac{Speedup}{NumberOfProcessors} \tag{13}$$

Figure 5 shows the comparison of speedup values achieved in both architectures, whereas Figure 6 compares the efficiency obtained. From this figures, it can be seen that the *GRID* architecture has a slightly lower efficiency, which indicates the overhead produced when running the application on this architecture.

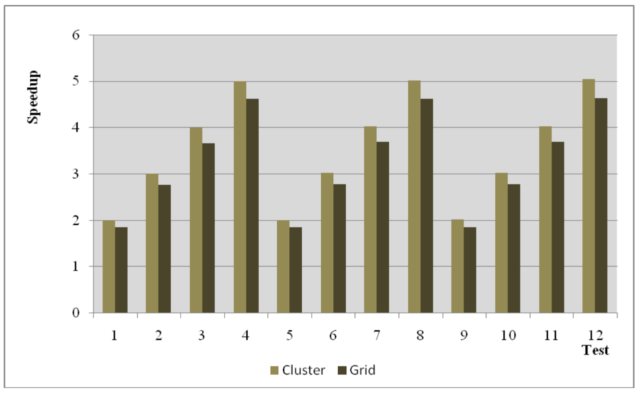

Figure 5: Speedup in *Cluster* and *GRID*.

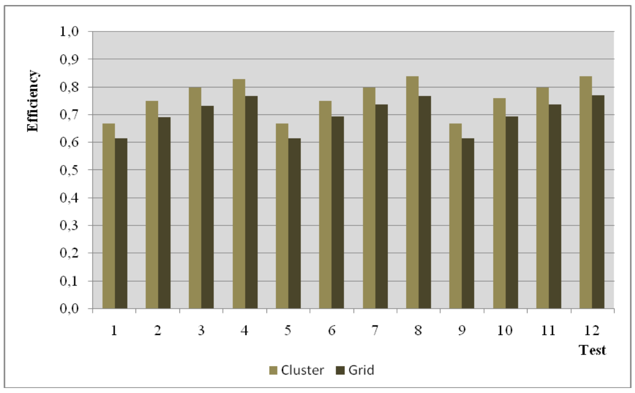

Figure 6: Efficiency in *Cluster* and *GRID*.

7. CONCLUSIONS AND FUTURE WORK

A parallel algorithm was developed to look for similarities in DNA sequences in large databases using the Smith-Waterman technique to analyze the similarity between two sequences. The algorithm was tested in a heterogeneous multi-cluster architecture with varying numbers of processors.

Then, different metrics were used to assess the behavior of this algorithm. The first of these was speedup, which, as Figure 3 shows, increases as the number of processors increases, and is constant for tests with equal number of processors even if sequence size is increases.

The second analysis allows comparing the speedup obtained with the theoretical speedup, which is an indication of how much the architecture is being exploited. Figure 4 shows that, in all cases, exploitation is above 82%, reaching a peak of 95% when 17 processors are used. It can also be seen that this relation increases as the number of processors increases.

Efficiency comparison shows a lower efficiency for Grid architecture due to middleware overhead. This overhead is 5 to 7 % . The overhead takes into account software layers and the increased complexity in message management for Grid.

Actual experimental work is focused in increasing de number of processors (in cluster and Grid) and extending Grid experiences using processors geographically distributed over different networks (located in Argentine, Brazil and Spain).

REFERENCES

[1] Jordan H, Alaghband G. “Fundamentals of parallel computing”. Prentice Hall, 2002.

[2] Juhasz Z.(Editor), Kacsuk P.(Editor), Kranzlmuller D.(Editor). “Distributed and Parallel Systems: Cluster and Grid Computing”. The International Series in Engineering and Computer Science. Springer, 1 edition. 2004.

[3] Di Stefano M. “Distributed data management for Grid Computing”. John Wiley & Sons Inc. 2005.

[4] Joseph J., Fellenstein C. ”Grid Computing”. On Demand Series. IBM Press. 2003.

[5] Foster I., Kesselman C., Kaufmann M. “The Grid 2: Blueprint for a New Computing Infrastructure”. The Morgan Kaufmann Series in Computer Architecture and Design. 2003.

[6] Abbas A. “Grid Computing: A Practical Guide to Technology and Applications”. Charles River Media. 2004.

[7] Berman F., Fox G., Hey A. “Grid Computing: Making The Global Infrastructure a Reality”. John Wiley & Sons. 2003.

[8] De Giusti A. et al. “Parallel algorithms on Multi-Cluster Architectures using GRID Middleware. Experiences in Argentine Universities”. Proceedings of the I Iberian Grid Infrastructure Conference. Spain. 2007. Pp. 322-332.

[9] Javadi B., Akbari M. A., Abawajy J. H. “Performance Analysis of Heterogeneous Multi-Cluster Systems”. Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPPW’05). 2005.

[10] Attwood T. K., Parry-Smith D. J. “Introducción a la Bioinformática”. Pearson Educación S.A. 2002.

[11] Zhang F., Qiao X., Liu Z. “A Parallel Smith-Waterman Algorithm Based on Divide and Conquer”. Proceeding of the Fifth International Conference on Algorithms and Architecture for Parallel Processing. 2002.

[12] Snir M., Otto S., Huss-Lederman S., Walker D., Dongarra J., “MPI: The Complete Reference”. The MIT Press, Cambridge, Massachusetts. 1996.

[13] De Giusti L. Chichizola F. “Búsqeda de Similitud en Secuencias de ADN: Análisis de Performance”. Technical Report. 2008.

[14] Grama A, Gupta A, Karypis G, Kumar V. “Introduction to parallel computing”. Second Edition. Pearson Addison Wesley, 2003.

[15] Dongarra J, Foster I, Fox G, Gropp W, Kennedy K, Torczon L, White A. “The Sourcebook of Parallel Computing”. Morgan Kauffman Publishers. Elsevier Science, 2003.

[16] Leopold C. "Parallel and distributed computing. A survey of models, paradigms, and approaches". Wiley Series on Parallel and Distributed Computing. Albert Zomaya Series Editor, 2001.

[17] Tinetti F. “Cómputo paralelo en redes locales de computadoras”. Tesis Doctoral. Universidad Autónoma de Barcelona, 2004. https://lidi.info.unlp.edu.ar/~fernando/publis/publi.html.

[18] Donaldson V, Berman F, Paturi R. “Program Speedup in a Heterogeneous Computing Network”. Journal of Parallel and Distributed Computing 21(3):316-322.1994.