Journal of Communication and Computer 9 (2012) 1364-1371

Parallel Pipelines for DNA Sequence Alignment on a Cluster of Multicores: A Comparison of Communication Models

Enzo Rucci, Franco Chichizola, Marcelo Naiouf, Laura De Giusti and Armando De Giusti.
Institute of Research in Computer Science LIDI (III-LIDI), School of Computer Science, National University of La Plata, 1900 La Plata (Buenos Aires), Argentina.

Received: May 7, 2012 / Accepted: June 6, 2012 / Published: December 31, 2012.

Abstract: HPC (high perfomance computing) based on clusters of multicores is one of the main research lines in parallel programming. It is important to study the impact of programming paradigms of shared memory, message passing or a combination of both on these architectures in order to efficiently exploit the power of these architectures. The Smith-Waterman algorithm is used as study case for the local alignment of DNA sequences, which allows establishing the similarity degree between two sequences. In this paper, the Smith-Waterman algorithm is parallelized by means of a pipeline scheme due to the data dependencies that are inherent to the problem, using the various communication/synchronization models mentioned above and then carrying out a comparative analysis. Finally, experimental results are presented, as well as future research lines.

Key words: Cluster of multicores, communication models, parallel programming, pipeline, Smith-Waterman.

1. Introduction

The study of distributed and parallel systems is one of the most active research lines in computer science nowadays [1, 2]. In particular, the use of multiprocessor architectures configured in clusters, multiclusters, grids and clouds, supported by networks with different characteristics and topologies, has become general, not only for the development of parallel algorithms but also for the execution of processes that require intensive computation and the provision of concurrent web services [3-6].

The term cluster is applied to "sets of computers built with standard hardware components that act as if they were an only computer"[4, 7]. Cluster technology has evolved to support activities that go from supercomputing applications and mission-critical software, to web services and high-performance databases.

Clusters' computing is the result of the convergence of several current trends, including the availability of cheap high-performance processors and high-speed networks, the development of software tools for high-performance distributed computation, and the growing need for computer power for the applications that require it [8].

The technological change, mainly with the development of multicore processors, has led to the development of hybrid parallel architectures that

Franco Chichizola, Ph.D. student, professor, research field: distributed and parallel systems. E-mail: francoch@lidi.info.unlp.edu.ar.
Marcelo Naiouf, Ph.D., chair professor, research field: distributed and parallel systems. E-mail: mnaiouf@lidi.info.unlp.edu.ar.
Laura De Giusti, Ph.D., professor, research field: distributed and parallel systems. E-mail: ldgiusti@lidi.info.unlp.edu.ar.
Armando De Giusti, chair professor, main researcher CONICET, research fields: distributed and parallel systems. E-mail: degiusti@lidi.info.unlp.edu.ar.
Corresponding author: Enzo Rucci, Ph.D. student, research field: distributed and parallel systems. E-mail: erucci@lidi.info.unlp.edu.ar.

combine shared and distributed memory.

In this context, it is important to study the modeling of the behavior of this type of mixed parallel systems, as well as develop new paradigms and tools for efficient application programming [9-11].

The technological change caused by energy consumption and heat generation problems that appear when scaling processor speed has caused the appearance of multicores. This type of processors is formed by the integration of two or more computer cores within the same chip, and increases application performance by dividing computing work among all available cores [12, 13].

The incorporation of this type of processors to conventional clusters gives birth to architecture that combines shared and distributed memory, known as cluster of multicores [14, 15].

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. The extensive comparison processing required to analyze genetic patterns demands a significant effort in the development of efficient parallel algorithms [16].

The center for all bioinformatics operations and analyses is partly held by sequence alignment, both for pattern searching among amino acid and nucleotide sequences, and for the search of phylogenetic relationships among organisms. 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 often used as basic pattern to compare different alignment techniques. If the length of the sequences involved are *N* and *M*, the complexity of the algorithm is *O(NxM)*. Thus, the problem is scaled as the square of sequence size [17].

Taking into account that sequences can have up to 10^9 nucleotides each, the time and memory required to solve this problem with a sequential algorithm is impracticable. This leads to the parallelization of the algorithm over powerful parallel architectures.

Taking into account the increase in use of the cluster of multicore architecture, it is important to study parallel algorithm programming techniques that efficiently exploit the power of the architecture.

In particular, the approach of the application to study is attractive due to its complexity and the possibility of breaking down parallel algorithm concurrency into "blocks" of different dimensions, which allows an optimal adaptation of the application to the support architecture.

In this paper, various parallelizations of the Smith-Waterman algorithm are presented, which use a pipeline scheme due to the data dependencies associated to the problem, and using different communication/synchronization models. Also, a comparative analysis of the yields obtained with each model is carried out.

In Section 2, the Smith-Waterman algorithm is explained, together with the sequential and the parallel solutions used in this paper. In Section 3, the experimental work carried out is described, whereas in Section 4, the results obtained are presented and analyzed. Finally, Section 5 presents the conclusions and future lines of work in relation to this paper.

2. Smith-Waterman Algorithm Definition

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. If gaps are inserted, that is, certain elements of the sequences are not aligned to achieve a better overall alignment, a penalization is applied.

The algorithm calculates a similarity score between two sequences and then, if necessary, employs a backwards alignment process for an optimal result [16].

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

Given two sequences: $A = a_1a_2a_3...a_M$ and $B = b_1b_2b_3...b_N$, a matrix H of $(N+1)x(M+1)$ is built, in such a way that the nucleotide bases that form sequence A label the rows (starting with 1), and those from sequence B label the columns (starting with 1). The following steps are applied to calculate the values of H that will yield the similarity score between A and B:

Start row 0 and column 0 of H with 0, as indicated in Eq. (1).

$$H_{i0} = H_{0j} = 0 \qquad \text{for } 0 \leq i \leq N \text{ and } 0 \leq j \leq M \quad (1)$$

Calculate the value of H_{ij}, $\forall i \in [1, .., N]$ and $\forall j \in [1, .., M]$ by means of Eq. (2). This value indicates the maximum similarity between two segments ending in a_i and b_j, respectively.

$$H_{ij} = \max\begin{cases} 0 \\ H_{i-1,j-1} + V(a_i, b_j) \\ C_{ij} \\ F_{ij} \end{cases} \quad (2)$$

$V(a_i, b_j)$ is the matching function that indicates the score obtained for matching a_i with b_j. It is based on a table of values called substitution matrix that describes the probability of a nucleotide base from sequence A at position i to occur in sequence B at position j. The most common matrix is the one that rewards with positive value when a_i and b_j are identical, and punishes with a negative value otherwise.

C_{ij} is the score in column j considering a gap, and is calculated with Eq. (3).

$$C_{ij} = \max_{1 \leq k \leq i}\{H_{i-k,j} - g(k)\} \quad (3)$$

F_{ij} is the score in row i considering a gap, and is calculated with Eq. (4).

$$F_{ij} = \max_{1 \leq l \leq j}\{H_{i,j-l} - g(l)\} \quad (4)$$

$g(x)$ is the penalization function for a gap of length x, and is obtained with Eq. (5), q being the penalization applied for opening a gap and r the penalization for prolonging it.

$$g(x) = q + rx \qquad (q \geq 0; r \geq 0) \quad (5)$$

The similarity score is obtained as shown in Eq. (6).

$$G = \max_{(0 \leq i \leq N)(0 \leq j \leq M)}\{H_{ij}\} \quad (6)$$

Based on the position in matrix H where the value G was found (representing the end of the highest-scoring alignment between both sequences), a backwards process is performed to obtain the pair of segments with maximum similarity, until a position whose value is 0 is reached, this being the starting point of the segment.

2.1 Sequential Solution of Smith-Waterman Algorithm

In this section, the sequential solution of Smith-Waterman algorithm is analyzed with the purpose of determining the similarity score between two DNA sequences. This means that the backwards process is not taken into account when obtaining the segment that represents the optimal alignment (step d of the algorithm explained in the previous section is not performed).

Fig. 1 shows the data dependency that exists for calculating matrix values. To obtain $H_{i,j}$, the result of $H_{i-1,j-1}$ (H_d in Fig. 1) is required, and the score must be known when considering a gap in row i and another one in column j. This restriction allows calculating H values from top to bottom and left to right (H_{11}, H_{12}, H_{13}, …, H_{21}, H_{22}, H_{23}, etc.).

Taking into account that step d of the algorithm is not carried out, matrix H does not have to be stored in full, all that is needed is:

A vector h of length $M+1$ that at each position keeps the value obtained in the last processed row over that column. Eq. (7) shows the values for h corresponding to the example shown in Fig. 1.

$$h_k = \begin{cases} H_{i,k} & k < j-1 \\ H_{i-1,k} & k \geq j-1 \end{cases} \quad (7)$$

An element e to temporarily store the last value calculated in the row that is being processed. In Fig. 1, $e = H_{i,j-1}$.

A vector c of length $M+1$ that is at each position keeps the maximum score considering a gap in that

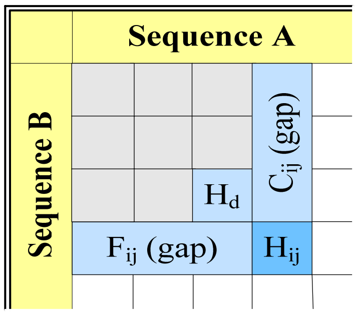

Fig. 1 Data dependency scheme.

column. Eq. (8) shows the values for c corresponding to the example shown in Fig. 1.

$$c_k = \begin{cases} C_{ik} & k < j \\ C_{i-1,k} & k \geq j \end{cases} \qquad (8)$$

An element f that keeps the maximum score considering a gap in the row that is being processed. In the example shown in Fig. 1, $f = F_{i,j-1}$.

2.2General Parallel Solution of Smith-Waterman Algorithm

The data dependency mentioned in the previous section causes the problem to be solved following a pipeline scheme where S stages perform the same work over various consecutive nucleotide subsets of the first sequence (A in Fig. 1). In each cycle, stage s_i (for $i \in [1, S-1]$) receives a data block from s_{i-1}, solves part of its work, and then sends these results to s_{i+1} (except for the last stage which does not need to send its results to any other stage). The first stage (s_0) only performs its work by sending partial results (corresponding to a block) to its successor.

An important aspect of this solution is selecting the number of elements (BS) from sequence B that form the data blocks that are sent from one process to another, taking into account that:

Pipeline parallelism is exploited to its maximum capacity only after S-1 cycles have been processed. That is, when all stages have received work to do. The larger the BS, the longer the time required to fill the pipe, and therefore, the lower its exploitation. From this point of view, BS should tend to 1.

If the size of BS is very small, the stages spend more time communicating partial results than actually processing information. From this point of view, BS should tend to N.

A suitable block size should be found, so that data communication and data processing can be done simultaneously. The optimal size does not only depend on the architecture used, but also on the communication model implemented.

In previous works, a procedure for calculating the optimal value of BS based on architecture characteristics and sequence size has been established [18].

2.2.1 Message Passing as Communication Model

In this case, each pipeline stage is carried out by a different process p_i (for $i \in [0, S\text{-}1]$), and partial results are communicated by sending messages between consecutive processes. The first sequence (A in Fig. 1) is distributed by p_0 among the S processes that form the pipeline.

Fig. 2 shows a schem2e of the parallel solution of Smith-Waterman algorithm using message passing as communication model.

2.2.2 Shared Memory as Communication Model

In this case, each pipeline stage is carried out by a different thread t_i (for $i \in [0, S\text{-}1]$). Instead of

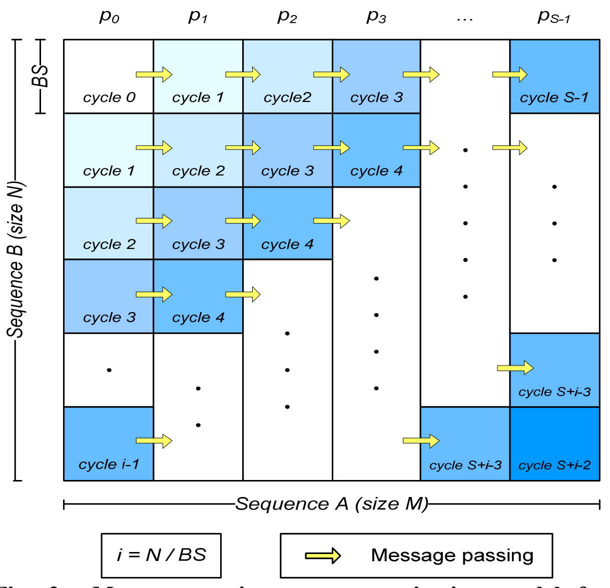

Fig. 2 Message passing as communication model for parallel solution of Smith-Waterman algorithm.

communicating partial results through message passing, these are kept in the shared memory as a single structure (as in the sequential algorithm). Consecutive threads are synchronized to indicate that work with a new data block can begin.

Fig. 3 shows a scheme of the parallel solution of Smith-Waterman algorithm using shared memory as communication model.

2.2.3 Hybrid Communication model: Combination of Message passing and Shared Memory

When each process p_i begins (for $i \in [0, P\text{-}1]$) it generates $T\text{-}1$ threads ($S = P\times T$) to jointly solve the data blocks corresponding to the different cycles. Thus, there are $P\times T$ threads (all P processes plus all $T\text{-}1$ threads generated by each of them), which means that the set of nucleotides from the first sequence is equally distributed among $P\times T$ threads.

The threads that are in the same node synchronize and communicate through shared memory, whereas those that are in different nodes do so through message passing.

Fig. 4 shows a scheme of the parallel solution of Smith-Waterman algorithm using a hybrid communication model.

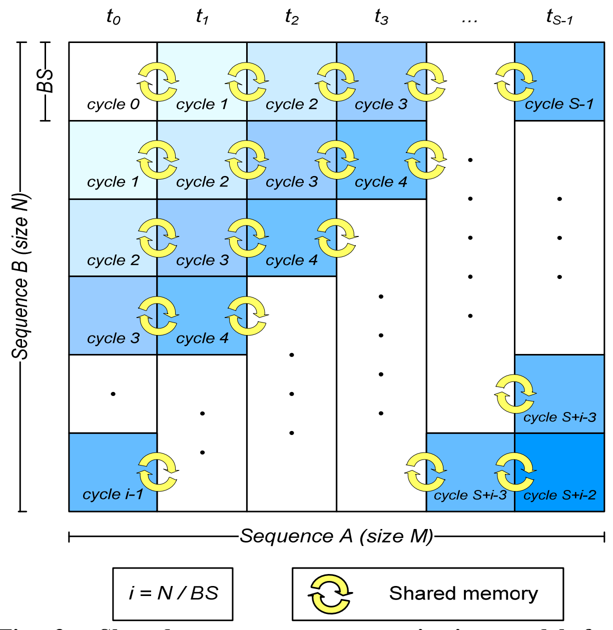

Fig. 3 Shared memory as communication model for parallel solution of Smith-Waterman algorithm.

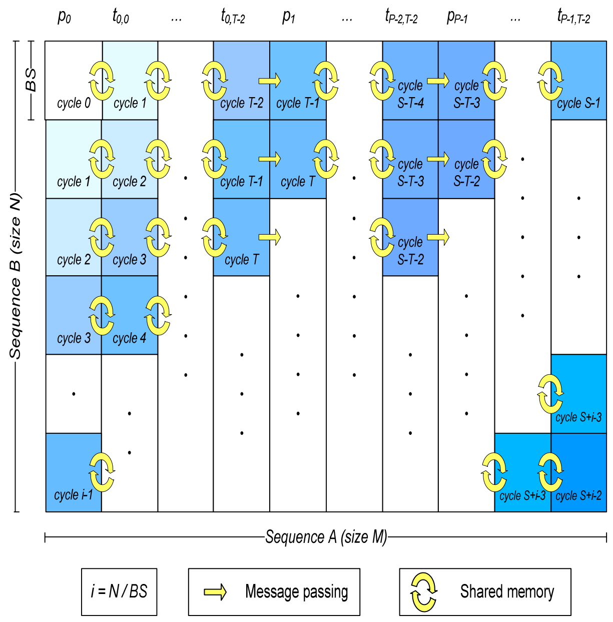

Fig. 4 Hybrid communication model for parallel solution of Smith-Waterman algorithm.

3. Experimental Work

In this paper, language C is used with OpenMPI and Pthreads libraries for message passing and the handling of threads, respectively.

3.1 Architecture Used

To analyze the behavior of the algorithms, tests were carried out on a cluster of blade multicores with eight blades and two quad core Intel Xeon e5405 2.0 GHz processors each. Each blade has 2 GB RAM memory (shared between both processors) and 2 x 6Mb L2 cache for each pair of cores [19, 20].

3.2 Algorithms Used

The algorithms used in this experiment are described below:

MP: this solution is based on using a pipeline of $S = P$ stages as the one described in Section 2.2.1, where P is the number of cores used;

HY: this hybrid solution is based on using a pipeline of $S = P\times T$ stages as the one described in Section 2.2.3, where P is the number of blades used and T is the number of cores in each blade.

It should be noted that an algorithm that uses only shared memory can not be used as communications model because there is no memory level available that is shared among the various nodes of the support architecture.

3.3 Tests Carried Out

The algorithms were tested using all the cores with different numbers of blades: two, four and eight, which means that 16, 32 and 64 cores were used, respectively. Sequence sizes of various lengths (*65,536; 131,072; 262,144; 524,288; 1,048,576*) were also taken. Table 1 shows the optimal block size (*BS*) used in each test in accordance with the function described in previous works [18].

4. Results

To assess the behavior of the algorithms developed when escalating the problem and/or the architecture, the speedup and efficiency of the tests carried out are analyzed [1, 4, 8, 21].

The speedup metrics is used to analyze the algorithm performance in the parallel architecture as indicated in Eq. (9).

$$Speedup = \frac{SequentialTime}{ParallelTime} \tag{9}$$

To assess how good the speedup obtained is, efficiency is calculated. Eq. (10) indicates how to calculate this metric, where p is the total number of used cores.

$$Efficiency = \frac{Speedup}{p} \tag{10}$$

Fig. 5 shows the efficiency achieved by the algorithms *MP* and *HY* when using two, four and eight blades of the architecture for different problem sizes (*N*). The sizes of the data blocks used were detailed in Table 1.

This chart shows that both algorithms obtain good efficiency levels taking into account the interaction pattern between them. It can also be seen that both algorithms increase their efficiency as the length of the sequences increases (size of the problem) and, on the other hand, as it is to be expected in most parallel systems, the efficiency decreases when the total number of nodes used increases.

Fig. 6 shows the percentage of the relative difference between the efficiencies of both algorithms (*prd*), calculated by means of Eq. (11).

$$prd = \frac{efficiency(MP) - efficiency(HY)}{efficiency(HY)} \times 100 \tag{11}$$

The chart shows that the performance of both algorithms is similar, although MP has a slight advantage over HY. The sizes of the data blocks used were detailed in Table 1.

The difference between the efficiencies achieved by MP and HY is not significant, being less than 2% in all cases. It can be observed that the difference

Table 1 Block size *BS* used in each test run on the cluster of multicores.

Cores	Sequence size				
	65,536	131,072	262,144	524,288	1,048,576
16	48	48	48	48	48
32	47	47	47	47	47
64	47	47	47	47	47

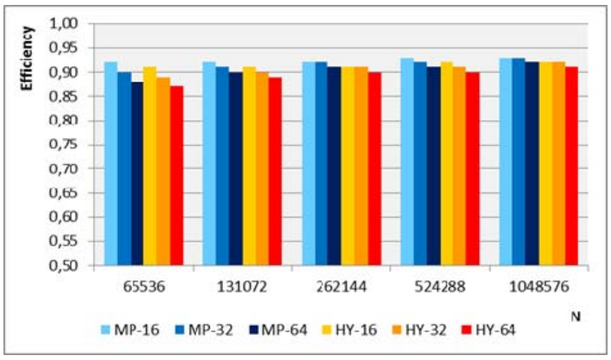

Fig. 5 Efficiency achieved by algorithms MP and HY when using 16, 32 and 64 cores of the architecture for different problem sizes (*N*).

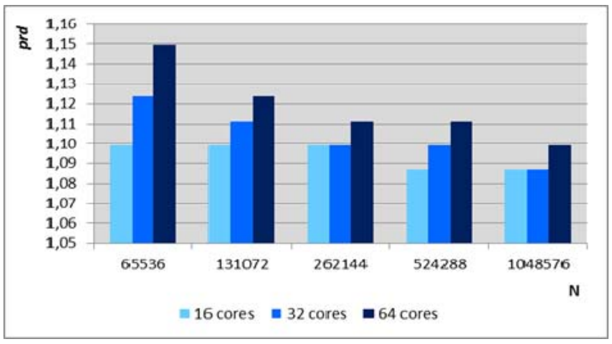

Fig. 6 *prd* with 16, 32 and 64 cores for different sequence lengths *(N)*.

decreases as the size of the problem increases, and inversely, it increases as the total number of cores increases. The similarity of the results is favored by the use of the same resolution scheme by both solutions. The minimum difference between the efficiencies is due to two factors. First, both algorithms have reduced memory requirements, which does not allow exploiting the benefits of using shared memory. The second factor is the optimization of current message passing libraries to work in shared memory environments. The combination of these two factors improves the performance of MP versus HY.

5. Conclusions

In this paper, a comparison is made of various communication models implemented over a multicore cluster using the Smith-Waterman algorithm for DNA sequence alignment. Parallelization is carried out by means of a pipeline scheme because of the data dependency that is inherent to the problem.

The solutions presented, MP and HY, use the same resolution scheme but different communication models (message passing and a hybrid combining message passing and shared memory, respectively).

The algorithms were tested using various work and architecture sizes. The results show that performance is better when message passing is used as communication model rather than a hybrid combining message passing and shared memory. This is because of the low memory requirements of these algorithms, together with the optimization offered by current message passing libraries to work in shared memory environments.

Future lines of work focus on two aspects:

- analysis and optimization of hybrid solutions for certain types of problems, especially for those that support a composite parallel solution (combining more than one paradigm);
- analysis of the presented algorithms from the point of view of energy efficiency on different architectures [22, 23].

References

[1] A. Grama, A. Gupta, G. Karypis, V. Kumar, An Introduction to Parallel Computing, Design and Analysis of Algorithms, Pearson Addison Wesley, 2003.

[2] M.B. Ari, Principles of Concurrent and Distributed Programming, Addison-Wesley, 2006.

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

[4] J. Zoltan, P. Kacsuk, D. Kranzlmuller, Distributed and Parallel Systems: Cluster and Grid Computing, Springer, 2004.

[5] M.D. Stefano, Distributed Data Management for Grid Computing, John Wiley & Sons Inc, 2005.

[6] M. Miller, Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online, QUE Publishing, 2008.

[7] http://www.cs.mu.oz.au/678/.

[8] B. Wilkinson, M. Allen, Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers, Pearson Prentice Hall, 2005.

[9] S. Siddha, V. Pallipadi, A. Mallick, Process scheduling challenges in the era of multicore processors, Intel Technology Journal 11 (2007) 4.

[10] M. Olszewski, J. Ansel, S. Amarasinghe, Kendo: Efficient determistic multithreading in software, in: Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems, New York, USA, 2009.

[11] M. Bertogna, E. Grosclaude, M. Naiouf, A.D. Giusti, E. Luque, Dynamic on demand virtual clusters in grids, in: 3rd Workshop on Virtualization High-Performance Cluster and Grid Computing, Las Palmas de Gran Canaria, Spain, 2008.

[12] Advanced Micro Devices Inc., AMD Multi-core White Paper, 2005.

[13] T.W. Burger, Intel Multi-Core Processors: Quick Reference Guide, 2005.

[14] M.M. Cool, Scalable programming models for massively multicore processors, in: Proceedings of the IEEE 96, 2007.

[15] L. Chai, Q. Gao, D.K. Panda, Understanding the impact of multi-core architecture in cluster computing: A case study with Intel Dual-Core System, in: IEEE International Symposium on Cluster Computing and the Grid, Rio de Janeiro, Brazil, 2007.

[16] K. Attwood, D.J. Parry-Smith, Introduction to Bioinformatics, Pearson Educacion S.A , 2002.

[17] F. Zhang, X. Qiao, Z. Liu, A parallel Smith-Waterman

algorithm based on divide and conquer, in: Proceeding of the Fifth International Conference on Algorithms and Architecture for Parallel Processing, Fukuoka, Japan, 2002.

[18] F. Chichizola, Analytical Study of Optimal Block Size Based on Cluster Characteristics, Technical Report, III-LIDI, 2011.

[19] http://h18004.www1.hp.com/products/blades/components/c-class.html.

[20] http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00810839/c00810839.pdf.

[21] C. Leopold, Parallel and Distributed Computing, A Survey of Models Paradigms and Approaches, Wiley, New York, 2001.

[22] W.C. Feng, The importance of being low power in high-performance computing, Cyberinfrastructure Technology Watch Quarterly 1 (2005) 12-20.

[23] J. Balladini, E. Grosclaude, M. Hanzich, R. Suppi, D. Rexachs, E. Luque, Impact of parallel programming models and CPUs clock frequency on energy consumption of HPC systems, in: Proceedings of the XVI Argentine Congress on Computer Sciences, 2010.