Comparison of Communication/Synchronization Models in Parallel Programming on Multi-Core Cluster

ENZO RUCCI, ARMANDO E. DE GIUSTI, FRANCO CHICHIZOLA, R. MARCELO NAIOUF AND LAURA C. DE GIUSTI

Instituto de Investigación en Informática LIDI (III-LIDI) – School of Computer Science – Universidad Nacional de La Plata, Argentina.
{erucci, degiusti, francoch, mnaiouf, ldgiusti}@lidi.info.unlp.edu.ar.

Abstract. *Taking into account the increase in use of the multi-core cluster architecture, in this paper we analyze the use of the various communication models (message passing, shared memory, their combination) to efficiently exploit the power of the architecture.*
Smith-Waterman algorithm, whose parallelization is based on a pipeline scheme due to problem data dependence, is used as test case to determine the similarity degree of two DNA sequences.
Finally, future research lines are mentioned, aimed at optimizing the use of memory levels in the architecture.

Keywords. *multi-core cluster, hybrid communication model, 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 multi-processor architectures configured as clusters, multi-clusters, and grids, supported by networks with different characteristics and topologies has become generalized, for the development of both parallel algorithms and distributed Web services [3][4][5]. Cloud computing developments follow the same line [6].
The technological change, mainly based on multi-core processors, has created the need for researching mixed or hybrid models where shared memory and message schemes are in coexistence [7][8].
In this context, it is important to study the modeling of the behavior of this type of parallel systems, as well as develop new paradigms and tools for efficient application programming [9][10][11].

1.1 Multi-core cluster

A multi-core processor is formed by the integration of two or more computational cores within the same chip [12]. The reasons for their development are based on the energy consumption and heat generation problems that appear when the speed of a processor is escalated.
A multi-core processor increases the yield of an application by dividing the computation work among the available cores [13].
A cluster is a parallel processing system formed by a set of computers interconnected over some kind of network and that cooperate as if they were an "only and integrated" resource, regardless of the physical distribution of its components. Each "processor" may have different hardware and operating system, and it can even be a "multiprocessor" [14].

1.2 Study application

One of the areas of greatest interest and growth in the last few years within the field of parallel processing is that of the treatment of large volumes of data such as DNA sequences. The extensive comparison processing required for the analysis of genetic patterns demands a significant effort in the development of efficient parallel algorithms [15].
The center for all bioinformatic 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 oftentimes 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 escalated as the square of sequence size [16].
Taking into account that sequences can have up to 10^9 nucleotides each, the time and memory required to solve this problem in a sequential manner is impracticable. This leads to the parallelization of the algorithm over powerful parallel architectures.

1.3 DNA Sequence Comparison on a Multi-core Cluster

Taking into account the increase in use of the multi-core cluster architecture, it is important to study new parallel algorithm programming techniques that efficiently exploit the power of the architecture by combining shared memory and message passing.
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 multi-core cluster support architecture.
The architecture used in this paper is a Blade with 8 blades. Each blade has 2 quad core Intel Xeon e5405 2.0 GHz processors; 2 Gb of RAM memory (shared between both processors); and 2 X 6Mb L2 cache for each pair of cores [17][18]. This architecture allows a comprehensive analysis of the three approaches (messages, shared memory, and hybrid).
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. 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 [14].
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:

a. Start row 0 and column 0 of H with 0, as indicated in Equation 1.

$$H_{i0} = H_{0j} = 0 \qquad \text{for } 0 \le i \le N \text{ and } 0 \le j \le M \tag{1}$$

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

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

- $V(a_i, b_j)$ 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 a 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 Equation 3.

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

- R_{ij} is the score in row i considering a gap, and is calculated with Equation 4.

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

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

$$g(x) = q + rx \qquad (q \ge 0; e \ge 0) \quad (5)$$

c. The similarity score is obtained as shown in Equation 6.

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

d. 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).

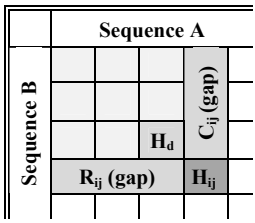

Fig. 1. Data dependency scheme

Figure 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 Figure 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}, …..).
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. Equation 7 shows the values for h corresponding to the example shown in Figure 1.

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

- An element e to temporarily store the last value calculated in the row that is being processed. In Figure 1, $e = H_{i,j-1}$.
- A vector c of length $M+1$ that at each position keeps the maximum score considering a gap in that column. Equation 8 shows the values for c corresponding to the example shown in Figure 1.

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

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

2.2 General Parallel Solution of Smith-Waterman Algorithm

Due to the dependency of data mentioned in the previous section, the problem needs to be solved by following a pipeline scheme, where stages S perform the same work over various consecutive nucleotide subsets of the first sequence (A in Figure 1). In each cycle, stage s_i (for $i \in [1, S\text{-}1]$) 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.

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 Figure 1) is distributed by p_0 among the *S* processes that form the pipeline.

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 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.

2.3 Hybrid Parallelization of the Algorithm by Integrating Message Passing and Shared Memory

When using a hybrid architecture, the different memory levels (among cores in a same blade) and the interconnecting network (among cores in different blades) should be considered to determine the optimal size *BS*. This leads to a solution that combines the use of message passing with shared memory.
This hybrid solution is based on the use of a pipeline of *P* stages as the one described in Section 2.2.1, each of these stages using a pipeline of *T* phases as the one detailed in Section 2.2.2.
When each process p_i begins (for $i \in [0, P\text{-}1]$), it generates *T-1* threads to jointly solve the data blocks corresponding to the different cycles. Thus, there are *P*x*T* threads (all *P* processes plus all *T-1* threads generated by each of them), which means that the set of nucleotides from the first sequence (*A* in Figure 1) is equally distributed among *P*x*T* threads.
When process p_i (for $i \in [0, P\text{-}1]$) needs to solve a data block (with BS_{mp} elements), it divides it in sub-blocks of BS_{sm} nucleotides each to be solved by the pipeline corresponding to that process. To take advantage of the features of the architecture, the optimal BS_{mp} and BS_{sm} values have to be determined for each case.

3. Experimental Work

In this paper, language C is used with OpenMPI and/or Pthreads libraries to handle message passing and threads, respectively.
As mentioned in the Section 1, a Blade with 8 blades, each with two 2.0 GHz quad core Intel Xeon e5405 processors, was used. Each blade has 2 Gb RAM memory (shared between both processors) and 2 x 6Mb L2 cache for each pair of cores.
Two types of tests were carried out:

- Using one blade of the Blade. Testing in this case purely parallel algorithms to determine suitable data block sizes.
- Using the entire architecture. Testing in this case the hybrid algorithm and the one that uses only message passing to compare both behaviors.

3.1 Tests with a single blade of the Blade

Tests were carried out on a single blade of the Blade (using the 8 cores) to analyze the behavior of the purely parallel algorithms described in Sections 2.2.1 and 2.2.2.
The tests carried out vary in sequence length (*N = 65536, 131072, 262144, 524288, 1048576*) and block size (*BS = 8, 16, 32, 64, 128, 256, 512, 1024, 2048*).
As a result of these tests, it was observed that the efficiency achieved by the algorithm that uses shared memory is slightly higher than the efficiency of the solution that uses message passing. This leads to the proposal of the hybrid algorithm described in Section 2.3 to fully exploit the architecture.
Also, the results obtained allowed determining the ideal values for BS_{sm} = 16 and BS_{mp} =128. Detailed results can be found in [19].

3.2 Tests with the entire architecture

In order to analyze the behavior of the hybrid algorithm, it is compared with the algorithm that uses message passing only, using the 8 cores with different numbers of blades (4 or 8). The same as in Section 3.1, sequence length varies (*N = 65536, 131072, 262144, 524288*). In the following paragraphs, the tests carried out are described.

- *MP:* the algorithm that uses only message passing is used with various block sizes (*BS = 128, 256, 512, 1024, 2048*).
- *HY*: the hybrid algorithm is used with a process p_i and 3 threads for each processor in each blade. That is, each shared memory pipeline uses an entire quad core processor. The value of BS_{sm} remains unchanged during the tests, and the size of the blocks in the message passing pipeline varies (BS_{mp} = 128*, 256, 512, 1024, 2048*).

4. Results

To assess the behavior of the algorithms developed when escalating the problem and/or the architecture, *efficiency* is analyzed (in this case, on homogeneous architectures, since all cores are equal) [1][2][20]. Equation 9 indicates how to calculate this metric, where p is the total number of cores used.

$$Efficiency = \frac{Speedup}{p} \quad (9)$$

Figure 2 shows the efficiency achieved by the algorithms *MP* and *HY* detailed in Section 3.2 for the most significant block sizes (BS_{mp} = 128, 512 and 2048). For readability, only the results obtained when using all 8 blades of the architecture are shown, since when using only 4, a similar behavior is observed, with a slight increase in efficiency.

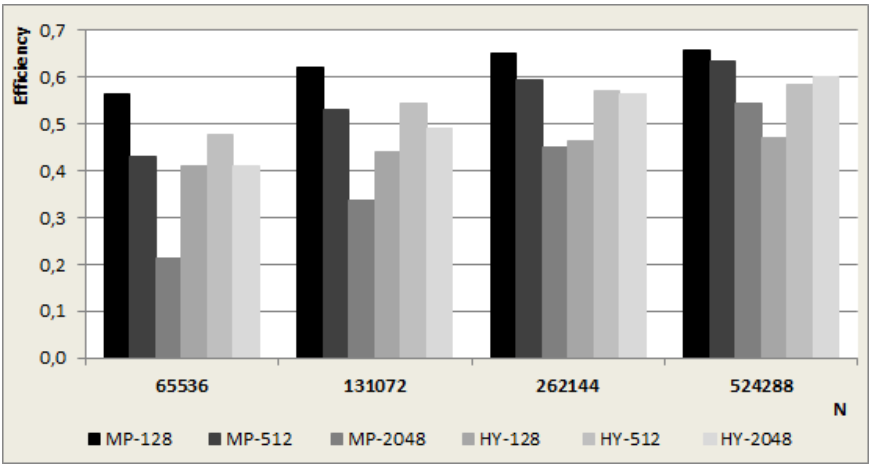

Fig. 2. Efficiency achieved by the algorithms *MP* and *HY* for different BS_{mp} and N values, using 64 cores (8 cores in 8 blades)

This chart shows that both algorithms increase their efficiency when the size of the problem increases (sequence length). On the other hand, these results confirm that, for algorithm *MP*, the ideal block size is 128. The hybrid algorithm (*HY*), however, tends to improve its efficiency when the size of the blocks increases, so that when comparing both algorithms using a block size of 2048, *HY* achieves a better efficiency.

Figure 3 presents a summary of the best efficiency achieved by each of the algorithms (*MP* and *HY*) when using 4 and 8 blades of the architecture. In the case of algorithm *MP (MP-4* and *MP-8* for 4 and 8 blades, respectively), it is achieved with a block size BS = 128. For *HY*, it is achieved when using BS_{mp} = 2048.

This chart shows that algorithm *MP* achieves a greater efficiency than the hybrid algorithm, and that the difference decreases as the size of the problem increases. On the other hand, as it is to be expected in most parallel systems, efficiency decreases when the number of cores used increases.
This figure also shows that when the total number of cores used increases, so does the difference between the efficiency achieved by *MP* and *HY*. Inversely, when the size of the problem increases, the gap decreases.

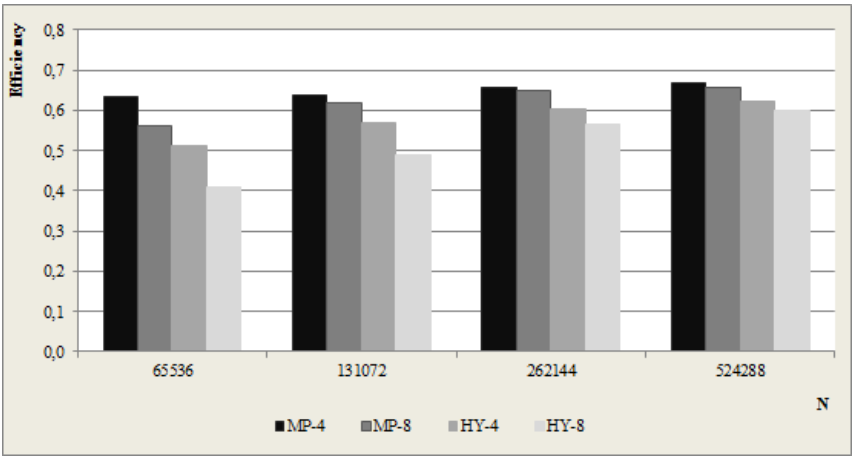

Fig. 3. Summary of the best efficiency achieved by each of the algorithms with 4 and 8 blades of the architecture

5. Conclusions and Future Works

In this paper, the Smith-Waterman algorithm is parallelized for the alignment of DNA sequences by means of a pipeline scheme due to the dependency of data that is inherent to the problem. The architecture used for the experiments is a multi-core cluster (8 blades with 8 cores each).
Given the characteristics of the architecture, the pipeline was initially implemented with two different communication models: message passing (*MP*) and shared memory (*SM*). The efficiency of both algorithms was compared when using a single blade of the architecture and a slight advantage of *SM* in relation to *MP* was observed.
Since the *SM* algorithm could not be used in the entire architecture (because there was no memory shared among the various blades), a third option was implemented (*HY*) using a hybrid communication model that combines message passing and shared memory. This version has a pipeline scheme among processes that communicate thorough message passing, and within each stage there is a shared memory pipeline to solve each data block.

The behavior of this algorithm was compared with that of *MP* using 4 and 8 full blades of the architecture. As a result, it was observed that *MP* achieves a greater efficiency than *HY*. This is because the optimal block size for *MP* (*BS* = 128) cannot be used in *HY* (BS_{mp} >> 128) pipeline because it would not be possible to generate enough sub-blocks to run the internal shared memory pipeline efficiently with its optimal block size (BS_{sm} = 16).
A future line of R&D is the 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). On the other hand, the scalability of the problem discussed, while ensuring a certain efficiency level, is also of interest.

References

1. Grama, A., Gupta, A., Karypis, G., Kumar, V. (2003). "An Introduction to Parallel Computing. Design and Analysis of Algorithms. 2nd Edition". Pearson Addison Wesley.
2. Jordan, H, Alaghband, G. (2002). "Fundamentals of parallel computing". Prentice Hall.
3. Dongarra, J., Foster, I., Fox, G., Gropp, W., Kennedy, K., Torczon, L., White, A. (2003). "The Sourcebook of Parallel Computing". Morgan Kauffman Publishers. Elsevier Science.
4. Juhasz Z., Kacsuk P., Kranzlmuller D. (editors) (2004). "Distributed and Parallel Systems: Cluster and Grid Computing". Springer; First Edition.
5. Di Stefano, M. (2005). "Distributed data management for Grid Computing". John Wiley & Sons Inc.
6. Miller, M. (2008). "Cloud Computing: Web-Based applications that change the way you work and collaborate online". QUE Publishing.
7. Mc Cool, M. (2007). "Programming models for scalable multicore programming". http://www.hpcwire.com/features/17902939.html
8. Chai, L., Gao, Q., Panda, D. K. (2007). "Understanding the impact of multi-core architecture in cluster computing: A case study with Intel Dual-Core System". IEEE International Symposium on Cluster Computing and the Grid 2007 (CCGRID 2007), 471-478.
9. De Giusti, L., Chichizola, F., Naiouf, M., De Giusti, A., Luque, E. (2010). "Automatic Mapping Tasks to Cores - Evaluating AMTHA Algorithm in Multicore Architectures". IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 2, No 1. 2010.
10. Olszewski, M., Ansel, J., Amarasinghe, S. (2009). "Kendo: Efficient Determistic Multithreading in Software". Architectural Support for Programming Languages and Operating Systems (ASPLOS '09).
11. Bertogna, M., Grosclaude, E., Naiouf, M., De Giusti, A., Luque, E. (2008). "Dynamic on Demand Virtual Clusters in Grids". 3rd Workshop on Virtualization in High-Performance Cluster and Grid Computing (VHPC 08). Spain.

12. AMD, “Evolución de la tecnología de múltiple núcleo”. 2009. http://multicore.amd.com/es-ES/AMD-Multi-Core/resources/Technology-Evolution.
13. Burger T. W., “Intel Multi-Core Processors: Quick Reference Guide”. http://cachewww.intel.com/cd/00/00/23/19/231912_231912.pdf
14. Grid Computing and Distributed Systems (GRIDS) Laboratory-Department of Computer Science and Software Engineering (University of Melbourne), “Cluster and Grid Computing”. 2007. http://www.cs.mu.oz.au/678/.
15. Attwood, T. K., Parry-Smith, D. J. (2002). “Introducción a la Bioinformática”. Pearson Educación S.A.
16. Zhang, F., Qiao, X., Liu, Z. (2002). “A Parallel Smith-Waterman Algorithm Based on Divide and Conquer”. Proceeding of the Fifth International Conference on Algorithms and Architecture for Parallel Processing. HP, “HP BladeSystem”. http://h18004.www1.hp.com/products/blades/components/c-class.html.
17. HP, “HP BladeSystem c-Class architecture”. http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00810839/c00810839.pdf.
18. Rucci, Enzo (2010). “Modelos de Comunicación en BLADE”. III-LIDI Technical Report.
19. Leopold, C. (2001). “Parallel and Distributed Computing. A survey of Models, Paradigms, and Approaches”. Wiley, New York.