A CGM-Based Parallel Algorithm Using the Four-Russians Speedup for the 1-D Sequence Alignment Problem

Jerry Lacmou Zeutouo* — Grace Colette Tessa Masse* — Franklin Ingrid Kamga Youmbi*

* Department of Mathematics and Computer Science
Faculty of Science, University of Dschang
PO Box 67, Dschang-Cameroon
jerrylacmou@gmail.com, gracetessa98@gmail.com, xingridkamga@gmail.com

ABSTRACT. The CGM model is one of the most widely used models for the design of parallel algorithms. It has shown its efficiency in solving several problems modeled by dynamic programming such as the longest common subsequence problem, which is a special case of the one-dimensional sequence alignment problem. This problem consists in aligning two strings of length n to measure their similarity. It is widely used in many fields, particularly in Bioinformatics where n is usually very large. Brubach and Ghurye proposed a sequential solution based on the Four-Russians speed-up that requires $\mathcal{O}(n^2/\log n)$ execution time. To the best of our knowledge, there are not yet parallel solutions on the CGM model to solve this problem. This paper is exclusively dedicated to this task. Our solution applies to both local and global similarity computations and is based on Brubach and Ghurye's sequential algorithm. It requires $\mathcal{O}(n^2/p \log n)$ execution time with $\mathcal{O}(p)$ communication rounds on p processors.

RÉSUMÉ. Le modèle CGM est l'un des modèles les plus utilisés pour la conception d'algorithmes parallèles. Il a montré son efficacité pour la résolution de plusieurs problèmes modélisés par la programmation dynamique comme le problème de la plus longue sous-séquence commune, qui est un cas particulier du problème d'alignement de séquence à une dimension. Ce problème consiste à aligner deux chaînes de longueur n afin de mesurer leur similarité. Il est largement utilisé dans de nombreux domaines, en particulier dans la Bio-informatique où n est généralement très grand. Brubach et Ghurye ont proposé une solution séquentielle basée sur l'accélération de Four-Russians qui nécessite un temps d'exécution de $\mathcal{O}(n^2/\log n)$. À notre connaissance, il n'existe pas encore de solutions parallèles sur le modèle CGM pour la résolution de ce problème. Ce papier est exclusivement dédié à cette tâche. Notre solution s'applique aux calculs de similarité locaux et globaux, et est basée sur l'algorithme séquentiel de Brubach et Ghurye. Elle nécessite un temps d'exécution de $\mathcal{O}(n^2/p \log n)$ avec $\mathcal{O}(p)$ rondes de communication sur p processeurs.

KEYWORDS : parallel algorithm, CGM model, dynamic programming, sequence alignment, Four-Russians

MOTS-CLÉS : algorithme parallèle, modèle CGM, programmation dynamique, alignement de séquences, Four-Russians

1. Introduction

Bioinformatics is a multi-disciplinary field of Biotechnology research that involves biologists, computer scientists, physicians, mathematicians, and bioinformaticians, intending to solve a scientific problem posed by Biology. It focuses on the development and application of computationally intensive methods to improve the understanding of biological processes. One of its fundamental problems is the alignment of sequences, crucial for molecular prediction, molecular interactions and phylogenetic analysis. It is generally used to extract functional and evolutionary information about genes and proteins.

Sequence alignment is the problem of comparing biological sequences by looking for a series of nucleotides or amino acids that appear in the same order in the input sequences, possibly introducing gaps [1]. It is a means of visualizing the similarity between sequences based on the notions of similarity or distance. Two types of alignments are considered: global alignments, which take into account the whole of two sequences to be compared, and local alignments, which make it possible to detect the segment of the first sequence which is most similar to a segment of the other. Due to dynamic programming, both types of alignments can be run in $\mathcal{O}(n^2)$ time where n is the length of two strings [2]. Based on the Four-Russians method, several sped-up sequential solutions have been proposed like the algorithm of Brubach and Ghurye running in $\mathcal{O}(\frac{n^2}{\log n})$ time [3].

The parallelization of the sequential algorithm on different parallel computing models was extensively treated by the community of parallel processing researchers. A PRAM algorithm running in $\mathcal{O}(\log n \log m)$ time with $\frac{nm}{\log m}$ CREW processors has been proposed by Apostolico et al. [4]. Alves et al. [5] proposed a parallel solution for a variant of the problem under the CGM model using weighted graphs that requires $\log p$ rounds and runs in $\mathcal{O}(n^2 \log m/p)$ time. Recently, Kim et al. [6] have proposed a space-efficient alphabet-independent Four-Russians' lookup table and a multithreaded Four-Russians' edit distance algorithm. The experiments they performed on CUDA-supported GPU show that the algorithm runs about 942 times faster than the sequential version of the original Four-Russians' algorithm for 100 pairs of random strings of length approximately $1,000$ when the alphabet size $|\Sigma| = 4$ and the Four-Russians tiling factor $t = 4$. To our knowledge, there is no solution on the CGM model for solving both the global and local sequence alignment problems.

In this paper, we tackle the problem of parallelizing the Brubach and Ghurye's sequential algorithm [3] for the one-dimensional sequence alignment problem on the CGM model (Coarse-Grained Multicomputer). Our solution requires $\mathcal{O}(n^2/p \log n)$ execution time with $\mathcal{O}(p)$ communication rounds on p processors. The choice of the model is due to the fact that it is more suitable to the current parallel architectures and it has already been used to efficiently solve several problems such as the optimal binary search tree [7].

The remainder of this paper is organized as follows: in section 2 Brubach and Ghurye's solution to the problem is recalled, then in section 3, we present our CGM-based solution and present the experimental results in section 4. The last section concludes our work.

2. Sequential algorithm for the 1-D sequence alignment problem

The sequence alignment method seeks to optimize the alignment score. This score is related to the similarity rate between the two compared sequences. It measures the

number of edit operations it takes to convert a sequence X into another Y. They include insertions, deletions, and substitutions of a single character. The weighted variant assigns a cost to each of the mentioned operations and through the use of a penalty matrix in case, costs appear not to be constant. The computation of the line-up score varies from global alignment to local alignment.

2.1. Global alignment

A global alignment of two sequences X and Y can be given by computing the distance between them [8]. The elementary editing operations are described as follows: substitution of an a symbol by a b symbol, deletion of an a symbol and insertion of a b symbol. There is also the ability to calculate global alignments using similarity scores instead of distance. A score is associated with each elementary editing operation. Given an alphabet Σ of symbols, for $a, b \in \Sigma$: $Sub(a, b)$ is the score for substituting symbol a with symbol b, $Del(a)$ is the score for deleting symbol a, $Ins(a)$ is the score for inserting symbol a.

To compute the score $d(X, Y)$ for the two sequences X and Y of length n, a dynamic programming table M with $n + 1$ rows and $n + 1$ columns is used such that $T[i, j] = d(X[0 \dots i], Y[0 \dots j])$ for $i = 0, \dots, n$ and $j = 0, \dots, n$. It follows that $d(X, Y) = T[n, n]$. The values of the table T is computed by the following recurrence formula (1) for $i = 0, 1, \dots, n$ and $j = 0, 1, \dots, n$:

$$T[i, j] = max \begin{cases} T[i-1, j-1] + Sub(X[i], Y[j]), \\ T[i-1, j] + Del(X[i]), \\ T[i, j-1] + Ins(Y[j]). \end{cases} \quad (1)$$

For any $0 \leq i \leq n$, $0 \leq j \leq n$, the boundary conditions of the recursive formular (1) are $T[0, 0] = 0$, $T[i, 0] = T[i-1, 0] + Del(X[i])$ and $T[0, j] = T[0, j-1] + Ins(Y[j])$.

2.2. Local alignment

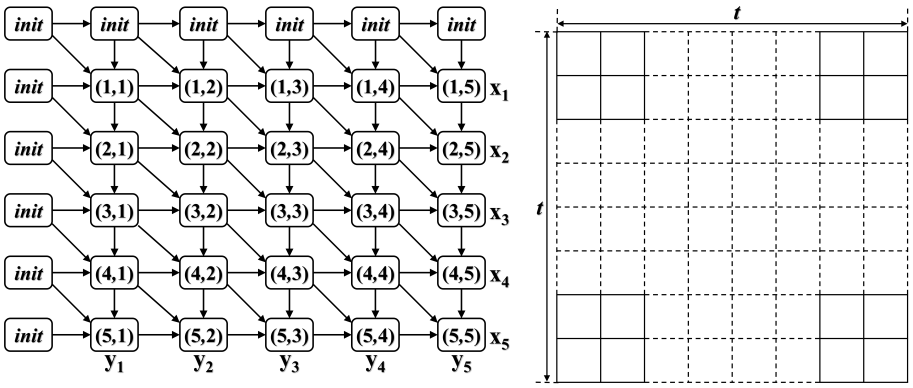
A local alignment of two sequences X and Y consists in finding the X segment that is most similar to a Y segment [8]. The local edit score of two sequences X and Y is defined by $s(X, Y)$, the maximum similarity between an X segment and a Y segment. To compute $s(X, Y)$ for the two sequences X and Y of length n, we use a dynamic programming table T_s with $n+1$ rows and $n+1$ columns such that for $i = 0, \dots, m$ and $j = 0, \dots, n$, $T_s[i, j] = max\{s(X[l \dots i], Y[k \dots j]) | 0 \leq l \leq i \text{ and } 0 \leq k \leq j\} \cup \{0\}$.

For $i = 0, 1, \dots, n$ and $j = 0, 1, \dots, n$, the values in the table T_s can be calculated with the recurrence formula (2) :

$$T_s[i, j] = max \begin{cases} T_s[i-1, j-1] + Sub(X[i], Y[j]), \\ T_s[i-1, j] + Del(X[i]), \\ T_s[i, j-1] + Ins(Y[j]), \\ 0. \end{cases} \quad (2)$$

The boundary conditions of the recursive formular (2) are $T_s[0, 0] = T_s[i, 0] = T_s[0, j] = 0$ for any $0 \leq i \leq n$, $0 \leq j \leq n$. Fig. 1a shows the form of the task graph or dependency graph [1] for both global and local sequence alignment problems when $n = 5$. The value at position (i, j) of the table T_s depends only on the values at the three neighbouring positions $(i-1, j-1), (i-1, j)$ and $(i, j-1)$.

1. The task graph is equivalent for dynamic programming tables T and T_s.



(a) Task graph for a problem of size 5×5.

(b) A single t-block.

Figure 1 – Task graph and a single t-block.

2.3. Speedups and the Four-Russians'

Dynamic programming algorithms can sometimes be made even faster by applying speedups such as the Knuth-Yao quadrangle-inequality speedup or the Four-Russians speedup.

Significantly applied to numerous problems, the Four-Russians technique arose from works [9] related to boolean matrix multiplication conducted by the four authors (only one is Russian though): Arlazarov, Dinic, Kronrod, and Faradzev. The idea behind the speedup is to tile the dynamic programming table into smaller blocks whose solutions are foreseen, precomputed and stored in a lookup table. The goal sought is spending less time on those by merely searching for them.

Unsurprisingly, an improvement (and sole) to this quadratic time bound has been first proposed using the Four-Russians technique by Masek and Paterson [2], achieving $\mathcal{O}(n^2/\log n)$. Crochemore et al. [10] later achieved the same time bound for unrestricted scoring matrices. Consequent to the conditional hardness results [11], this is unlikely to be improved further for arbitrary strings even on small alphabets, unless the Strong Exponential Time Hypothesis (SETH) does not hold.

2.4. Brubach and Ghurye's efficient lookup table

The bottleneck in the aforementioned speedup lies in the space demanded by the table, and the time spent computing it. In 2018, Brubach and Ghurye [3] introduced a process in which the Four-Russians lookup table can be built in $\mathcal{O}(t^2 \lg t)$, queried in $\mathcal{O}(t)$ and stored in $\mathcal{O}(t^2)$ space. t represents the size of the smaller blocks used for the ti phase. This globally outweighs all other algorithms known to date.

All credit to Masek and Paterson [2], the *t*-blocks within the dynamic programming table are tiled in a way they overlap by one column/row on each side as shown in Fig. 1b. Given a *t*-block first row and column, the block function of the lookup process has to provide its last row and column. Running the block function time and again in a row-wise (or column-wise) manner will eventually yield the global edit distance score in the bottom-right cell.

Aside from further efficiency, there is a huge improvement in the approach proposed by Brubach and Ghurye: the lookup table construction is only concerned about the strings provided as input. Let's delve into this new way of storing and querying lookup entries.

2.4.1. Notation

Consider a single *t*-block. Let $U = \{u_1, u_2, \ldots, u_{2t-1}\}$ denote the ordered set of cell labels ranging from the top-right cell to the bottom-left cell of the first row and column of the block. Likewise, let $V = \{v_1, v_2, \ldots, v_{2t-1}\}$ denote the ordered set of cell labels ranging from the top-right cell to the bottom-left cell of the last row and column of the block. Notice that u_1 and v_1 label the same top-right cell, as well as u_{2t-1} and v_{2t-1} label the bottom-left cell.

For a cell labeled u, the value in is denoted c_u. Formally, the block function is to return $c_v (v \in V)$ values given $c_u (u \in U)$ values. This is solved efficiently by considering least costs $c_{u,v}$ of paths from u to v through the block grid. We have that $c_v = \min_{u \in U}(c_u + c_{u,v})$.

2.4.2. Storing entries

The tiled *t*-blocks define sub-problems which pertain to corresponding sub-strings of the input. That said, for every sub-problem, we compute and store costs $c_{u,v}$ as defined above, in a $|V| \times |U|$ matrix M with a row for each $v \in V$ and a column for each $u \in U$. Entry M_{ji} will store the cost c_{u_i,v_j}. This matrix obviously consumes $\mathcal{O}(t^2)$ in memory space.

As to computing the cost $c_{u,v}$, this is instantly equivalent to finding the weight of a shortest path from cell u to cell v through a $t \times t$ grid. Jeanette Schmidt [12] came up in 1995 with a solution running in $\mathcal{O}(t^2 \lg t)$ time for every pair (u, v), by taking the problem to collections of binary trees.

2.4.3. Querying entries

A set of values c_u are provided to retrieve another set of c_v, both of size $(2t - 1)$. Due to the peculiarity of the lookup table, it is possible to complete this in $\mathcal{O}(t)$ time.

Matrices M used to store weights of shortest paths are proven to be Monge arrays [12]. And so will be the matrix M' derived from M, such that $M'_{ji} = c_{u_i} + M_{ji}$. The classic SMAWK algorithm [13] can therefore apply to obtain the row-minima of M' in $\mathcal{O}(t)$ time. Since not all M' entries are required, they are generated on the fly. Do note that these row-minima correspond to the aspired results $c_{v_j} = \min_{u \in U}(c_u + c_{u,v_j})$.

Algorithm 1 draws the big picture.

Algorithm 1: The general structure of sequential algorithm

Input : Two strings X and Y of the same length n
Output: The edit distance score between X and Y

```
1 Initialize a full (n+1) × (n+1) dynamic programming table D, by filling up its
    first row and column with accurate offset values;
2 Tile the whole table into smaller blocks of size t as described above;
3 for every block do
4 |   Compute and store the corresponding lookup matrix;
5 for every block in a row-wise manner do
6 |   Set upper-left corner to 0;
7 |   Lookup block's last row and column using the Four-Russians block function;
8 return n + sum(D[n + 1]);
```

3. CGM algorithm for the 1-D sequence alignment problem

This section describes our CGM solution. As the sequential algorithm, our solution is divided into two parts. Firstly, each processor computes the lookup table through the Brubach and Ghurye's sequential algorithm in $\mathcal{O}(t^2 \lg t)$. Secondly, we partition the task graph into sub-graphs of the same size, and we distribute them fairly onto the processors.

3.1. Partitioning strategy

Our technique consists in partitioning the task graph in two steps:

1) firstly, we partition the task graph into small blocks of size t referred to as *t-blocks* such that any two adjacent t-blocks overlap by either a row or column. After this first partitioning, the task graph is divided into k lines and k columns where $k = \left\lceil \frac{n}{t-1} \right\rceil$;

2) next, we subdivide the $\frac{n^2}{t^2}$ t-blocks into p lines and p columns of blocks referred to as *macro-blocks* and denoted by $MB(i, j)$. $MB(i, j)$ is a matrix of size $\left\lceil \frac{k}{p} \right\rceil \times \left\lceil \frac{k}{p} \right\rceil$ and is identified by the node on the lower right corner. Its evaluation computes the distance between $X[1 \ldots i]$ and $Y[1 \ldots j]$. Thus, $MB(n, n)$ will contain the final solution to the problem.

In the remainder of this paper, $z = \left\lceil \frac{k}{p} \right\rceil$. Figure 2 shows a scenario of this partitioning for $k = 8$ and $p = 4$. The number in each macro-block represents the diagonal to which it belongs.

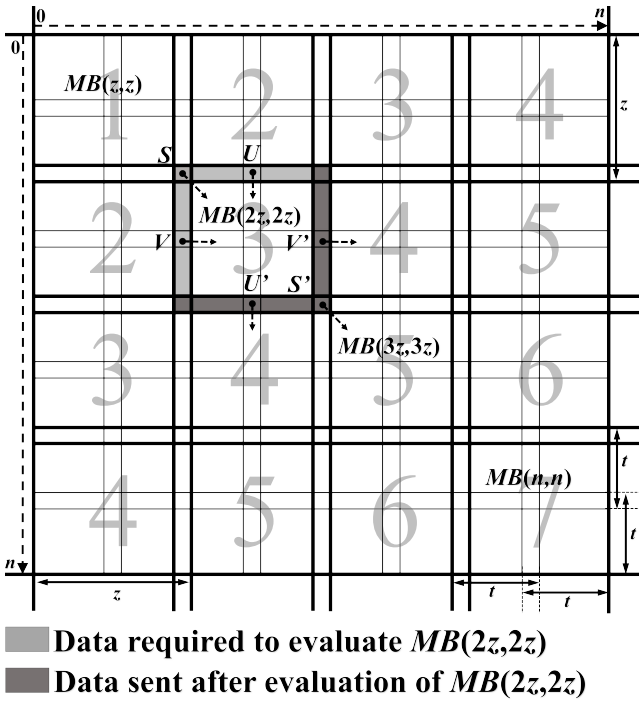


Figure 2 – Task graph partitioning into 8×8 t-blocks and 4×4 macro-blocks for $k = 8$ and $p = 4$.

REMARK. —
After partitioning,

1) all macro-blocks of the task graph are of the same size $z \times z$ when $n = pk(t-1)$;

2) any two adjacent macro-blocks overlap by either a row or column;

3) two blocks $MB(i,j)$ and $MB(t,u)$ belong to the same level if $|t-i| = |u-j|$;

4) the first diagonal of macro-blocks of the task graph (level 1) contains only the macro-block located in the top left corner of the graph, and the last one (level $2p-1$) consists only of the macro-block $MB(n,n)$;

5) from one diagonal to another, the number of macro-blocks increases by one unit starting from diagonal 1 to diagonal p, and decreases by one unit from the diagonal p to the diagonal $(2p-1)$.

Lemma 1 (Dependencies between macro-blocks)**.** *With $z = \left\lceil \frac{k}{p} \right\rceil$, let's consider a macro-block $MB(i,j)$ of the task graph:*

1) the evaluation of this block depends only on the sets of nodes S, U and V of blocks $MB(i-z, j-z)$, $MB(i-z, j)$ and $MB(i, j-z)$ respectively where S, U and V are defined by:

- $S = \{(i-z, j-z)\}$;
- $U = \{(i-z, j-z+1), (i-z, j-z+2), \ldots, (i-z, j)\}$;
- $V = \{(i-z+1, j-z), (i-z+2, j-z), \ldots, (i, j-z)\}$;

2) after the evaluation of this block, the new sets S', U', V' will be communicated respectively to t processors that will evaluate the blocks $MB(i+z, j+z)$, $MB(i+z, j)$ and $MB(i, j+z)$. These sets are defined by:

- $S' = \{(i,j)\}$;
- $U' = \{(i, j-1), (i, j-2), \ldots, (i, j-z+1)\}$;
- $V' = \{(i-1, j), (i-2, j), \ldots, (i-z+1, j)\}$.

Proof. These dependencies between blocks come from the dependencies between nodes of the task graph. □

3.2. Mapping macro-blocks onto processors

We use a snake-like mapping [7] which allows some processors to evaluate at most one block more than the others. This distribution consists in assigning all macro-blocks of a diagonal from the top left corner to the bottom right corner. This process is renewed until all processors have been used, starting with processor 0 and traveling through the blocks with a "snake-like" path. Figure 3 illustrates such a distribution for 4 processors.

Lemma 2. *After the partitioning of the task graph and snake-like distribution scheme, each processor evaluates exactly p macro-blocks.*

Proof. There are p^2 macro-blocks after the partitioning of the task graph. Therefore, it is clear that each processor evaluates exactly p blocks. □

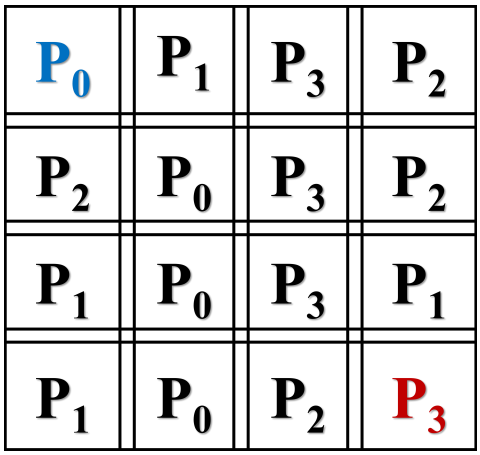


Figure 3 – Snake-like distribution onto processors for $p = 4$.

3.3. Overview of the algorithm

We present our CGM solution for the sequence alignment problem. Brubach and Ghurye's sequential algorithm is used for local computations. Based on the previous partitioning strategy and the mapping of blocks onto processors, the corresponding CGM algorithm is presented in algorithm 2.

Algorithm 2: The general structure of our CGM algorithm

Data: Two strings X and Y of the same length n
Result: The edit distance score between X and Y

1 Compute and store the lookup table for the $\frac{n^2}{t^2}$ t-blocks using the algorithm 1;
2 **for** $d = 1$ **to** $2p - 1$ **do**
3 Evaluation of the macro-blocks of diagonal d, using the algorithm 1;
4 Communication of the sets S', U' and V' to the processors of the blocks of diagonal $d + 1$;

Lemma 3. *The evaluation of a macro-block of size* $\left\lceil \frac{n}{p(t-1)} \right\rceil \times \left\lceil \frac{n}{p(t-1)} \right\rceil$ *requires* $\mathcal{O}\left(\frac{n^2}{p^2 t}\right)$ *computation time.*

Proof. The evaluation of a t-block needs $\mathcal{O}(t)$ time, and a macro-block contains at most $\mathcal{O}\left\lceil \frac{n^2}{p^2 t^2} \right\rceil$ t-blocks. □

Lemma 4. *Our CGM algorithm requires* $\mathcal{O}\left(\frac{n^2}{pt}\right)$ *time steps per processor.*

Proof. According to lemmas 2 and 3, it is clear that the computation time on each processor is $\mathcal{O}\left(\frac{n^2}{p^2 t}\right) \times p = \mathcal{O}\left(\frac{n^2}{pt}\right)$. □

Theorem 1. *By subdividing the task graph into macro-blocks of size* $\left\lceil \frac{n}{p(t-1)} \right\rceil \times \left\lceil \frac{n}{p(t-1)} \right\rceil$ *and using the snake-like distribution scheme, our CGM algorithm requires* $\mathcal{O}\left(\frac{n^2}{p \log n}\right)$ *execution time with* $\mathcal{O}(p)$ *communication rounds when* $t = \log n$.

Proof. Our CGM algorithm evaluates the task graph diagonal after diagonal starting from diagonal 1. Each processor evaluates at most one block in a diagonal. The evaluation of a macro-block requires $\mathcal{O}\left(\frac{n^2}{p^2t}\right)$ time steps according to lemma 3. Then, the evaluation of a diagonal of blocks needs $\mathcal{O}\left(\frac{n^2}{p^2t}\right)$ time steps. Since there are $2p-1$ diagonals of blocks in the task graph, the execution time is $\mathcal{O}\left(\frac{n^2}{p^2t}\right) \times (2p-1) = \mathcal{O}\left(\frac{n^2}{pt}\right)$ and the number of communication rounds is $\mathcal{O}(p)$. Therefore, when $t = \log n$, our solution runs in $\mathcal{O}\left(\frac{n^2}{p \log n}\right)$. □

4. Experimental results

In this section, we benchmark our CGM solution with Brubach and Ghurye's sequential algorithm. We implemented this algorithm on the cluster dolphin of the MATRICS platform of the University of Picardie Jules Verne [2] using 60 computation nodes (48 nodes called thin nodes with 48 × 128GB of RAM, and 12 named thick nodes with 12 × 512GB of RAM). Each node is made of two Intel Xeon Processor E5-2680 V4 (35M Cache, 2.40 GHz) and each of them consists of 14 cores. All nodes are interconnected with Omni-Path links providing 100Gbps throughput. The C++ programming language is used, on the operating system CentOS Linux release 7.6.1810. The inter-processor communication is implemented with the MPI library (OpenMPI version 1.10.4).

We use a real biological DNA sequences [3] with $|\Sigma| = 4$. The results presented here are derived from its execution for different values of the triplet (m, n, p), where m and n are the size of sequences, with values ranging from 10^5 to 10^6, and p is the number of processors, with values in the set $\{1, 2, 4, 8, 16, 32, 48\}$.

Figures 4 and 5 show that our solution keeps good performance when the size of sequences and the numbers of processors increase. For $m = 10^6$, our algorithm runs about 11.43 times faster than the Brubach and Ghurye's sequential algorithm on 16 processors. The speed-up increases up to 20.40 on 32 processors and 30.05 on 48 processors. From all this, we can conclude that our algorithm is scalable to the increase of the size of sequences and the numbers of processors.

5. Conclusion and future research directions

By allowing a faster lookup table construction, Brubach and Ghurye have enhanced Masek and Paterson's Four-Russians-based solution to the edit distance problem [3]. We attempted in this paper to parallelize their results in application to the one-dimensional sequence alignment problem. Based on the CGM model, our solution clusters t-blocks into macro-blocks and follows the snake-like distribution mapping pattern onto p processors to achieve for $\mathcal{O}(p)$ rounds of communication, an execution time in $\mathcal{O}(n^2/p \log n)$. Experimental results show good agreement with theoretical forecasts. Noticing the fact that single processors uselessly compute large sets of entries in their respective lookup tables, one might be interested in cutting off the waste. Or even extend our solution to address the two-dimensional sequence alignment problem.

2. https://www.u-picardie.fr/recherche/presentation/plateformes/plateforme-matrics-382844.kjsp
3. https://www.ncbi.nlm.nih.gov/nuccore/110645304

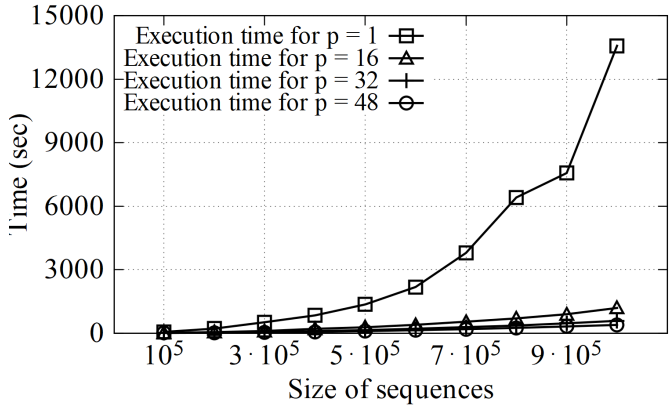


Figure 4 – Execution time for $m \in \left[10^5, \ldots, 10^6\right]$, $p \in \{1, 16, 32, 48\}$.

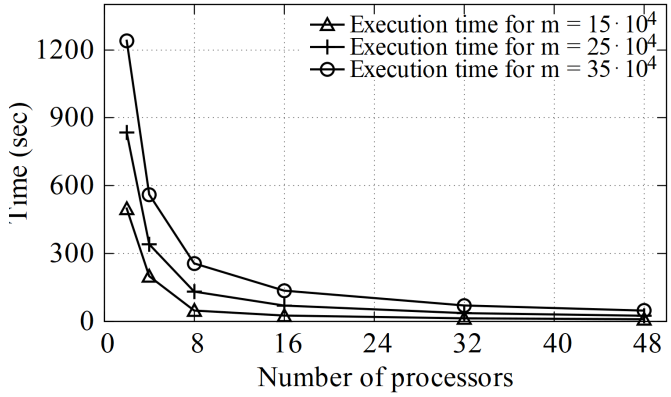


Figure 5 – Execution time for $m \in \{15{\times}10^4, 25{\times}10^4, 35{\times}10^4\}$, $p \in \{2, 4, 8, 16, 32, 48\}$.

Acknowledgements

The authors wish to express their gratitude to the computer Lab-MIS of the University of Picardie Jules Verne which made it possible to carry out the experimentations of this work. The authors also thank Dr. KENGNE TCHENDJI Vianney whose valuable comments and suggestions have significantly improved the readability of this work.

6. References

[1] C. SHARMA , A. .K VYAS, "Parallel Approaches in Multiple Sequence Alignments", *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, num. 2, 2014.

[2] W. MASEK, M. PATERSON, "A faster algorithm computing string edit distances", *Journal of Computer and System sciences*, vol. 20, num. 1, pp. 18–31, 1980. doi:10.1016/0022-0000(80)90002-1.

[3] B. BRUBACH, J. GHURYE, "A Succinct Four Russians Speedup for Edit Distance Computation and One-against-many Banded Alignment", *Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*, pp. 1–12, 2018. doi:10.4230/LIPIcs.CPM.2018.13

[4] A. APOSTOLICO, M. J. ATALLAH, L. L. LARMORE, S. MACFADDIN, "Efficient parallel algorithms for string editing and related problems", *SIAM Journal on Computing*, vol. 19, num. 5, pp. 968–988, 1990.

[5] C. E. R. ALVES, E. N. CÁCERES, F. DEHNE, P. SHOR, "Parallel Dynamic Programming for Solving the String Editing Problem on a CGM/BSP", *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pp. 275–281, 2002. doi:10.1145/564870.564916

[6] Y. KIM, J.C. NA, H. PARK, J.S. SIM, "A space-efficient alphabet-independent Four-Russians' lookup table and a multithreaded Four-Russians' edit distance algorithm", *Theoretical Computer Science*, vol. 656, num. pp 173–179, 2016. doi:10.1109/ISTCS.1995.377044

[7] V. K. TCHENDJI, "Solutions parallèles efficaces sur le modèle CGM d'une classe de problèmes issus de la programmation dynamique", *Université de Picardie Jules Verne, Amiens, France, Ph.D. Thesis* , 2014.

[8] E. CHANONI , T. LECROQ, A. PAUCHET, P. SHOR, "Une nouvelle heuristique pour l'alignement de motifs 2D par programmation dynamique", *Journées Francophones de Planification, Décision et Apprentissage pour la conduite de systèmess, Jun 2008, Metz, France*, pp. 83–92, 2008.

[9] V. ARLAZAROV, Y. DINITZ, M. KRONROD, I. FARADZHEV, "On economical construction of the transitive closure of an oriented graph", *Doklady Akademii Nauk*, vol. 194, num. 3, pp. 487–488, 1970.

[10] M. CROCHEMORE, G. LANDAU, M. ZIV-UKELSON, "A subquadratic sequence alignment algorithm for unrestricted scoring matrices", *SIAM journal on computing*, vol. 32, num. 6, pp. 1654–1673, 2003. doi:10.1137/s0097539702402007

[11] A. BACKURS, P. INDYK, "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)", *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 51–58, 2015. doi:10.1145/2746539.2746612

[12] J. P. SCHMIDT, "All shortest paths in weighted grid graphs and its application to finding all approximate repeats in strings", *Proceedings Third Israel Symposium on the Theory of Computing and Systems*, pp 67–77, 1995. doi:10.1109/ISTCS.1995.377044

[13] A. AGGARWAL, M. KLAWE, S. MORAN, P. SHOR, R. WILBER, "Geometric applications of a matrix-searching algorithm", *Algorithmica*, vol. 2, num. 1-4, pp. 195–208, 1987. doi:10.1145/564870.564916

A. Description of the Coarse-Grained Multicomputer model

The BSP/CGM model (*Bulk Synchronous Parallel/Coarse-Grained Multicomputer*) seems to be the best for the design of algorithms that are not too dependent on a particular architecture [7]. A CGM machine is a set of p processors, each having its local memory of size s (with $\mathcal{O}(s) \gg \mathcal{O}(1)$) and connected through a router able to deliver messages in a point-to-point manner. Each CGM parallel algorithm is an alternation of local computations and global communication rounds. Each communication round consists in routing a single h-relation with $h = \mathcal{O}(s)$. Each CGM computation or communication round corresponds to a BSP super-step having a communication cost $g \times s$ [7]. Here, g is the cost of communication of a word in the BSP model. To produce an efficient BSP/CGM parallel algorithm, the effort of the designers must be to maximize speed-up and minimize the number of communication rounds (ideally, it must be independent of the problem size, and constant in the optimum).