GLOBAL ALIGNMENT OF DNA SEQUENCES IN PARALLEL
Distributed Computing

Project Repository
Adriano Ramón Herández
Computer And Systems Engineering
aramonh@unal.edu.co

Abstract

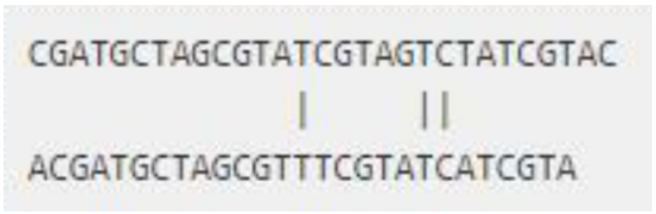
In this project, we seek to visualize in a practical way the implementation of parallelism for the Needleman-Wunsch algorithm, to seek to optimize processes of alignment of DNA sequences, in proteins and nucleotides.

Keywords: Alignment, Algorithm, DNA sequences, Loop Skewing, Parallelism

1 Introduction

In bioinformatics, the Needleman-Wunsch algorithm is commonly used to align nucleotide or protein sequences. It was proposed in 1970 by Saul Needleman and Christian Wunsch in their paper A general method applicable to the search for similarities in the amino acid sequence of two proteins.

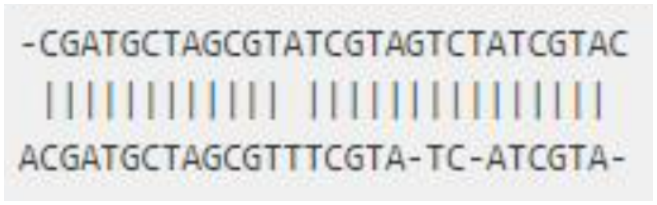
The Needleman–Wunsch algorithm is an example of dynamic programming, and it is guaranteed to find the alignment with the maximum score. Aligning sequences consists of searching for all areas of significant similarity between two or more sequences.



```
CGATGCTAGCGTATCGTAGTCTATCGTAC
           |     ||
ACGATGCTAGCGTTTCGTATCATCGTA
```

Figure 1: Strings to align.

Shifting one of the sequences the necessary positions to find the best possible alignment.



```
-CGATGCTAGCGTATCGTAGTCTATCGTAC
 |||||||||||| ||||||||||||||
ACGATGCTAGCGTTTCGTA-TC-ATCGTA-
```

Figure 2: Aligned sequences.

2 Methods

1. Initialization.
2. Matrix Filling (scoring).
3. Solution Recovery (Backtracking).
4. Alignment.

2.1 Initialization

To find if two biological sequences are homologous, the two sequences are compared in a substitution matrix, thus creating a matrix of M+1 columns and N+1. The first row and first column are zero padded.

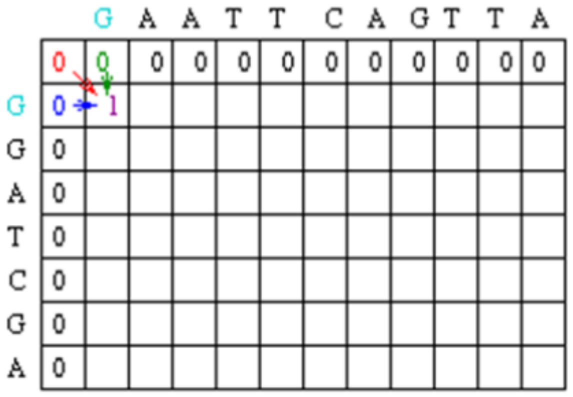


Figure 3: Initialization.

2.2 Scoring

The matrix is filled with a condition that takes into account three neighboring elements of the cell to be filled. The condition can be expressed as:

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + s(i,j) \\ F(i-1, j) - w \\ F(i, j-1) - w \end{cases}$$

Figure 4: Condition.

Where W is the penalty for finding a gap in the alignment and S(i,j) is the similarity or equality function. You can put the values that you think are convenient for W and S(i,j). The specific condition that for the filling of each cell, the largest of three elements found within the matrix will be taken into account, according to the values previously assigned to them.

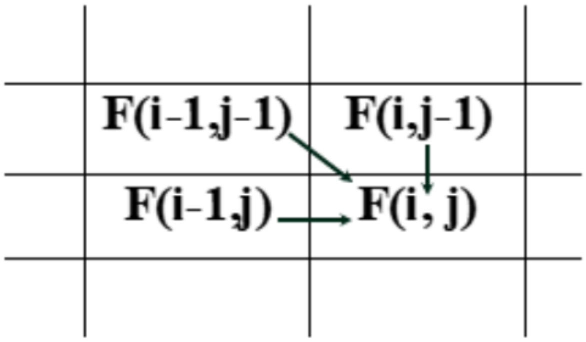
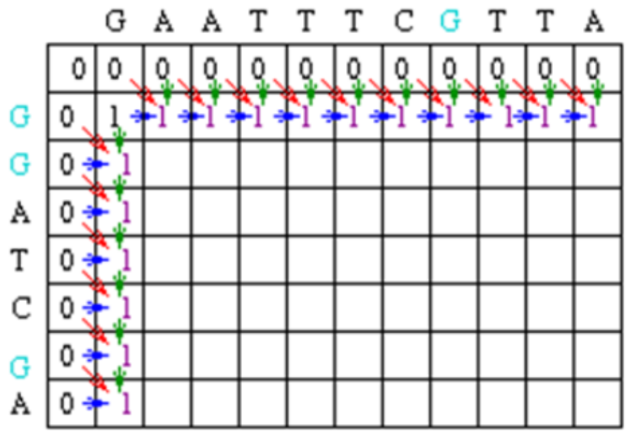


Figure 5: Neighbor behavior.

The filling of the matrix corresponds to giving a value to the intersection of the rows and columns, according to the scoring scheme of the given condition.

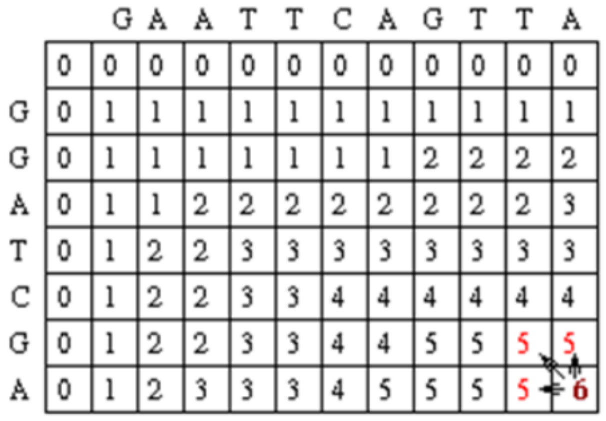


		G	A	A	T	T	T	C	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1										
A	0	1										
T	0	1										
C	0	1										
G	0	1										
A	0	1										

Figure 6: Initialization and filling.

2.3 Backtracking

It consists of taking the last coincidence of the alignment and starting to look for the path that maximizes the function.



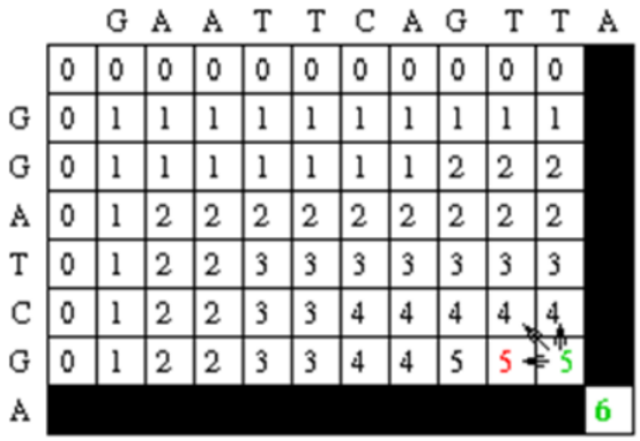
		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1	2	2	2	2
A	0	1	1	2	2	2	2	2	2	2	2	3
T	0	1	2	2	3	3	3	3	3	3	3	3
C	0	1	2	2	3	3	4	4	4	4	4	4
G	0	1	2	2	3	3	4	4	5	5	5	5
A	0	1	2	3	3	3	4	5	5	5	5	6

Figure 7: Backtracking.

In the previous matrix the maximum alignment is 6.

The backtracking starts at position M,J of the matrix at the position where the maximum alignment score occurs. The algorithm walks through the neighbors of the current cell to identify its predecessors. That is, look at the neighbors to the left, the diagonal neighbor, and the neighbor above. Potential neighbors are marked in red. In the example above they are all equal to 5.

If the initial position does not coincide, any of the neighbors are valid to start performing the alignment They all generate a different alignment, therefore it is important to analyze the best path from the point of view of weigHs and take it.



		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	
G	0	1	1	1	1	1	1	1	1	1	1	
G	0	1	1	1	1	1	1	1	2	2	2	
A	0	1	2	2	2	2	2	2	2	2	2	
T	0	1	2	2	3	3	3	3	3	3	3	
C	0	1	2	2	3	3	4	4	4	4	4	
G	0	1	2	2	3	3	4	4	5	5	5	
A												6

Figure 8: Backtracking and most value.

When verifying the neighbors, the possible values are 4 and 5. The value that maximizes the function is MAX(4,4,5) = 5 The path to take is 5, for which a column must be moved to the left of the value being maximized.

In this way, the matrix is traversed, always keeping in mind that when all the scores are the same and the penalty is the same, any path can be taken, generating multiple solutions.

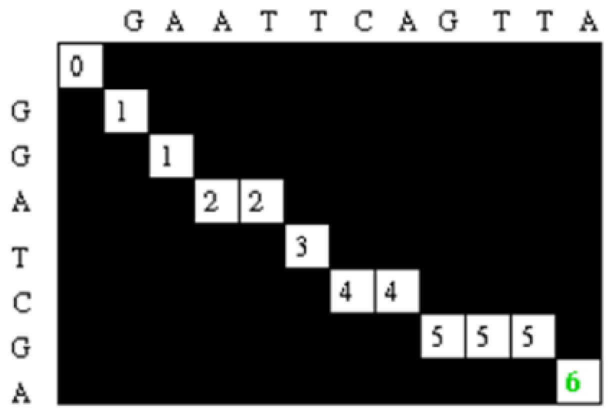
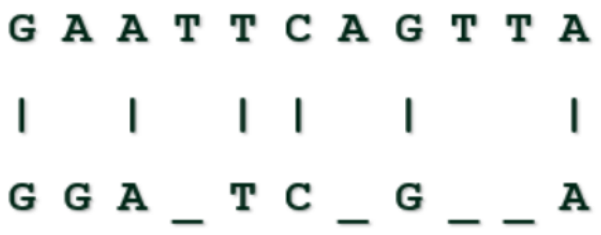


Figure 9: Generated way.

2.4 Alignment

Finally, we walk the path, from the upper left cell to the lower rigH cell, making the corresponding matching, verifying that each symbol is matched only once, if the symbol is required two or more times, in that case a space is generated bearing the hole symbol (-):



```
G A A T T C A G T T A
|   |   | |   |     |
G G A _ T C _ G _ _ A
```

Figure 10: Alignment.

Of the different alignments that can result from the substitution matrix, the alignment with the fewest gaps will be the most optimal alignment for the alignment solution.

3 Parallelization

The parallelization of the algorithm lies mainly in the filling of the scoring matrix, where thanks to the dependency of the data seen in figure 4, we see that for each cell (i,j) it is required to know the values (i-1 ,j),(i,j-1) and (i-1,j-1), that is, the left, upper and upper left diagonal cell, thanks to this data dependency we can apply what is known as Wavefront parallelism , where the wavefront is each antidiagonal of the scoring matrix and we can fill in the values of the antidiagonal in parallel.

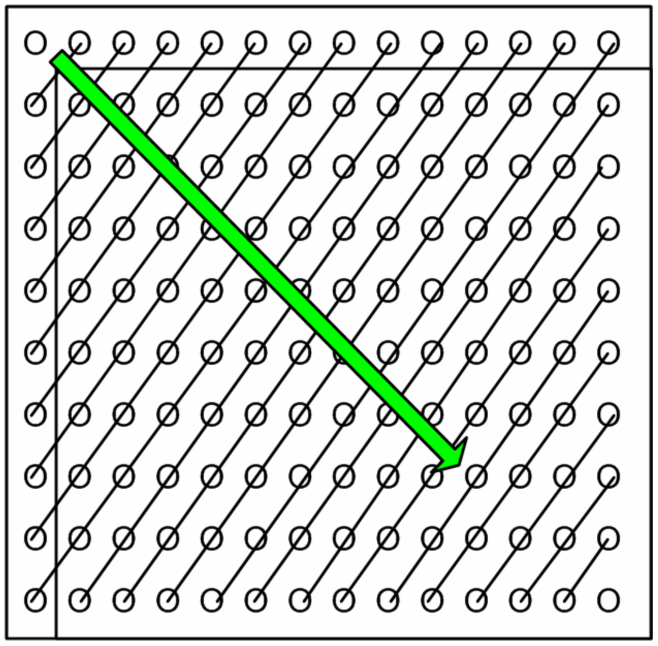


Figure 11: Parallelization Wavefront Antidiagonal.

A technique used to implement this type of parallelism is a loop transformation called loop skewing, which consists of transforming the space of iterations to another where the access to the antidiagonals is aligned to one of the dimensions of the new transformation, facilitating parallel access from that new iteration space, is a loop transformation technique used in other types of problems such as the longest common subsequence problem, the Gauss Seidel numerical method and others in which it is possible to access parallel to the data of the antidiagonals of their matrices.

Iteration spaces are represented as vector spaces studied in linear algebra and can be transformed with linear transformations.

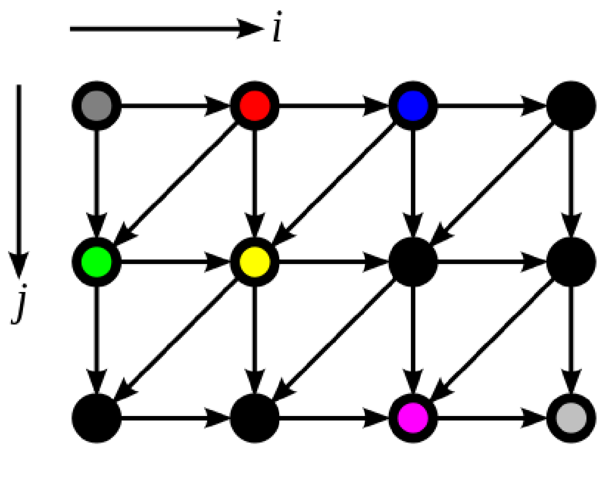


Figure 12: Loop Skewing and Parallelism Visualization before change.

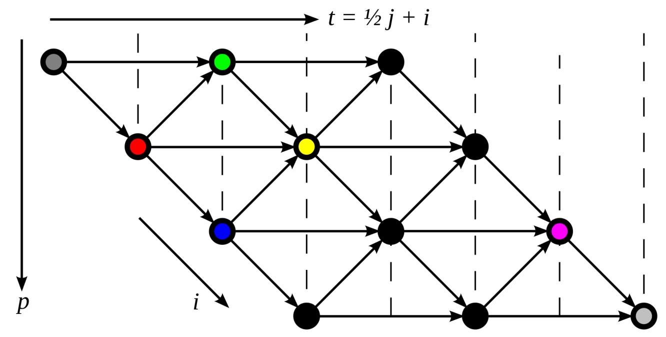


Figure 13: Loop Skewing and Parallelism Visualization after change.

4 Results

4.1 OpenMP

Once parallelization with OpenMP has been implemented, it is tested with 1,2,4,6,8 and 16 threads, where the following results were obtained:

threads	SpeedUP
1	1
2	1,7574
4	3,2073
8	3,6318
16	3,8926

Table 1: OpenMP SpeedUP comparison.

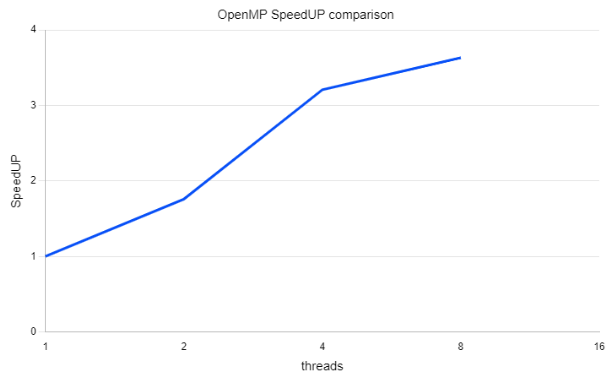


Figure 14: OpenMP SpeedUP comparison.

4.2 CUDA

Then CUDA parallelization was implemented, obtaining the following results.:

threads	SpeedUP
64	1
256	1,02355
1024	1,03452
4096	1,06178
9216	1,15637

Table 2: CUDA SpeedUP comparison.

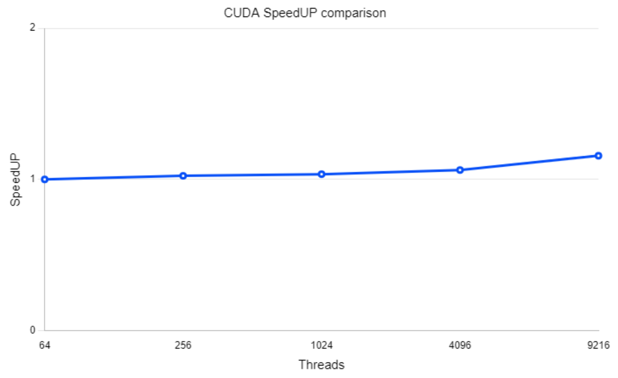


Figure 15: CUDA SpeedUP comparison.

4.3 MPI

For the implementation of MPI, a cluster was deployed in google cloud services, using 8 nodes including the master. It is tested where 1,2,4,6 and 8 processes in use of the nodes were tested. Obtaining the following results.:

process	SpeedUP
1	1
2	1,2131
4	0.10487
6	0.0817
8	0.0768

Table 3: MPI SpeedUP comparison.

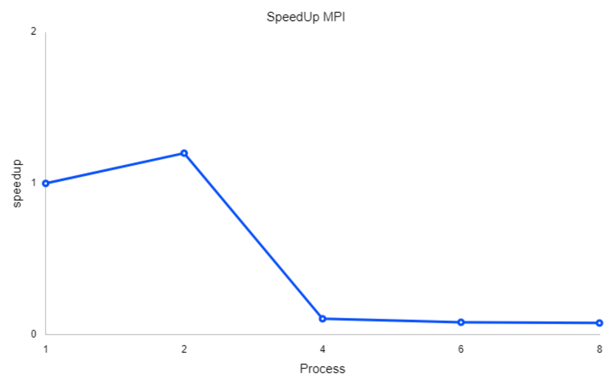


Figure 16: MPI SpeedUP comparison.

4.4 OPENCL

For the implementation of OpenCL colab was used, where the processes were divided according to the size of the sequences. Obtaining the following results.:

process	SpeedUP
1	1
3	1,05791
9	1,06992
27	1,11045
29	1,08296

Table 4: OpenCL SpeedUP comparison.

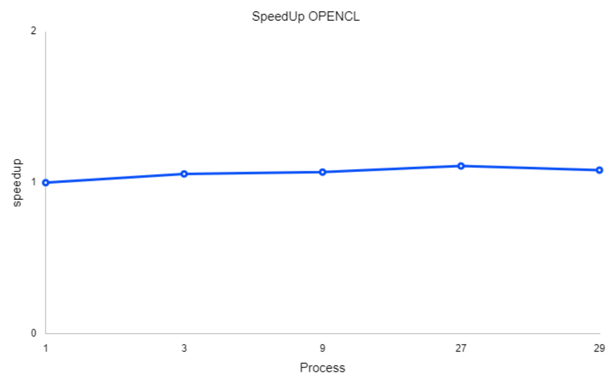


Figure 17: OpenCL SpeedUP comparison.

5 Conclusions

Being a matrix path processing that can become very long, the implementation of parallelism manages to significantly reduce the times with respect to its sequential execution, being logical that dividing the task makes it faster to complete its objective. However, the most notable thing is that the time is reduced especially when implementing a parallelization of 9216 threads in the case of cuda and 2 processes in the case of mpi, after that it tends to increase its time; On the other hand, in the cases of openmp, the execution times were not so low, but high time differences were noticed when increasing the number of threads. With openCL there was also performance, but the position of the tool was not won more efficient to parallelize in this case. That is to say, it is recommended to use mainly openmp and later cuda, to parallelize DNA sequence alignments.

6 References

http://www.bioinformaticos.com.ar/algoritmo-needleman-wunsch/
https://www.youtube.com/watch?v=5iXp2gh1S$_Ut = 1201s$
https://www.youtube.com/watch?v=KhuAlEbdJFct=162s
https://www.cri.ensmp.fr/ tadonki/PaperForWeb/tadonki$_loop.pdf$