CSE305 Project : Parallel sequence alignment

Darya TODOSKOVA, Elsa BISMUTH, Jean-Sébastien GAULTIER

June 7, 2023

https://github.com/daryatodoskova/CSE305-Project-Parallel-sequence-alignment/tree/main

Abstract

Sequence alignment is a fundamental task in bioinformatics, playing a crucial role in comparing and analyzing genetic sequences. In this paper, we present our implementation and parallelization of two recent algorithms, Needleman-Wunsch and Smith-Waterman, for sequence alignment. We evaluate the performance of these algorithms on real-life data and discuss the results obtained. Our findings demonstrate the potential of parallelization in significantly improving the efficiency of sequence alignment.

1 Introduction

Sequence alignment helps in understanding the evolutionary relationships between organisms, identifying functional elements within genomes, and predicting protein structures. Two popular algorithms, with a similar core structure [SBM16], used for sequence alignment are the Needleman-Wunsch and Smith-Waterman algorithms. Both compute the distance between each nucleotide (or amino acid) and are both dynamic programming-based algorithms. These algorithms can be implemented with a time complexity of O(nm) [RA04], where n and m represent the lengths of the sequences being aligned. However, as this complexity becomes significant for lengthy sequences, a solution is to parallelize these algorithms, in order to enhance their performance on large-scale sequence alignment tasks. We will thus manage to reduce the execution time of these algorithms and make them more efficient in handling real-life genomic data.

2 Specs

All tests were run on a Microsoft Surface Book. The specifics of the computer are as such:

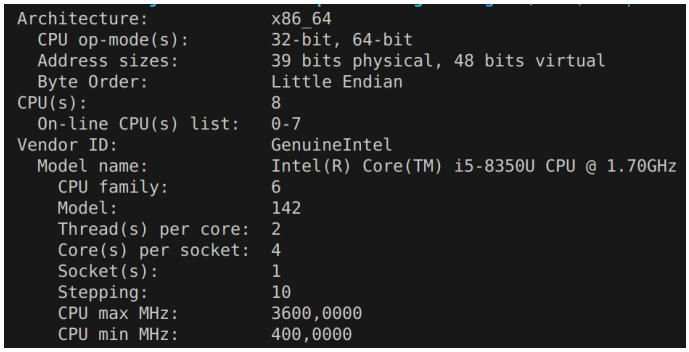

Figure 1: Intel Corporation UHD Graphics 620 (rev 07)

3 Methodology

The Needleman-Wunsch algorithm [NW70] performs global sequence alignment, i.e. we return the best alignment over the entire two sequences. On the other hand, the Smith-Waterman algorithm performs local sequence alignment, which identifies the best local alignments within two sequences.

Our objective is to analyze two sets of ordered nucleotide or amino acid sequences and determine the best alignment, emphasizing substitutions, insertions, and deletions.

3.1 Implementation of Needleman-Wunsch Algorithm : Global Alignment

The Needleman-Wunsch algorithm is used to align two sequences, such as DNA or protein sequences, in order to identify similarities and differences between them. It assigns scores to different types of matches, mismatches, and gaps, and finds the alignment that maximizes the total score.

Here's how the algorithm works:

- Initialize a matrix: Create a matrix with dimensions (length of sequence1 + 1) by (length of sequence2 + 1). This matrix will be used to store alignment scores.
- Fill the first row and column: Set the values of the first row and column of the matrix. Each cell represents the score of aligning a prefix of one sequence with gaps in the other sequence.
- Fill the rest of the matrix: Starting from the second row and second column, calculate the alignment scores for each cell based on three possible operations:
  - Match: If the corresponding characters in both sequences are the same, add the match score to the diagonal cell's score.
  - Mismatch: If the characters are different, add the mismatch penalty to the diagonal cell's score.
  - Gap: Add the gap penalty to the score of the cell above or the cell to the left.

  Choose the maximum score among these three options and store it in the current cell.
- Traceback: Start from the bottom-right cell of the matrix and move towards the top-left cell, following the path with the highest scores. At each step, determine the operation (match, mismatch, or gap) that led to the current score and update the alignment accordingly.
- Return the alignments: The final alignment will be the characters chosen along the traceback path. These aligned sequences highlight the similarities and differences between the original sequences.

The Needleman-Wunsch algorithm guarantees finding the optimal alignment with the highest possible score.

3.2 Implementation of Smith-Waterman Algorithm : Local Alignment

The Smith-Waterman algorithm [SW81] is similar to the Needleman-Wunsch algorithm, but it allows local alignments - we return the biggest region of similarity over two sequences. We implement the algorithm by constructing a scoring matrix and calculating the scores and traceback pointers in a similar manner as the Needleman-Wunsch algorithm. However, we introduce modifications to identify and report local alignments within the sequences.

Here's an overview of the Smith-Waterman:

- Initialize a matrix : same as for NW.
- Fill the first row and column: Set the values of the first row and column of the matrix to 0.
- Fill the rest of the matrix: same as for NW, by choosing the maximum score among these three options and storing it in the current cell. However, if the resulting score is negative, set it to 0 to allow for local alignments.

- Traceback: Start from the cell with the maximum score in the matrix and follow the path of cells with increasing scores until a cell with a score of 0 is reached. At each step, determine the operation (match, mismatch, or gap) that led to the current score and update the alignment accordingly.
- Return the alignments: same as for NW. These aligned sequences represent the local alignment within the original sequences.

The Smith-Waterman algorithm allows for identifying local similarities within sequences, making it useful for detecting subsequences that share common patterns.

4 Parallelization Strategy

In order to improve the performance of the Needleman-Wunsch and Smith-Waterman algorithms, we employed the parallelization technique of diagonal wavefront [RACDS03]. This approach divides the alignment matrix into multiple diagonal wavefronts and assigns each wavefront to a separate thread, enabling parallel computation of both alignment scores and traceback.

It thus allows for concurrent computation of alignment scores within each wavefront, significantly reducing the overall execution time of the algorithms.

4.1 Technique Overview

- Divide the alignment matrix into multiple diagonal wavefronts.
- Assign each wavefront to a separate thread.
- Parallelize the computation of alignment scores within each wavefront, considering match scores, mismatch penalties, and gap penalties.
- Synchronize the threads after each wavefront computation step to maintain dependencies.
- Perform parallel traceback to determine the optimal alignment for each wavefront.

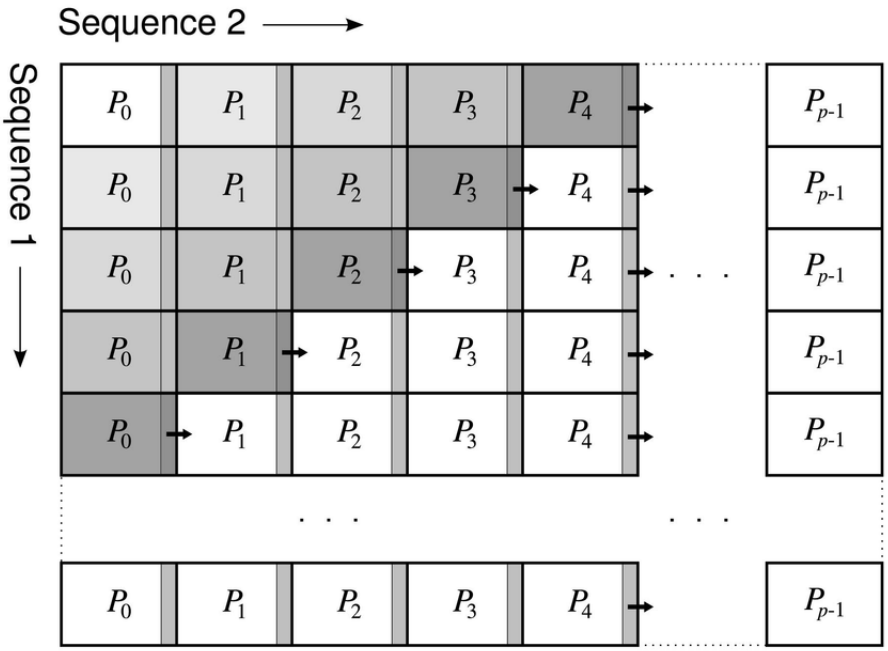

Figure 2: Block division in Diagonal Wavefront : the blocks in the same shade of gray are computed simultaneously in parallel [SA09]

As you can see above, the computation of blocks follows a diagonal wavefront pattern, where each thread is assigned a column of blocks. The shaded rightmost column of each computation block of the table has to be sent to the next thread in order to compute its assigned block in the next iteration.

4.2 Motive

The decision to choose the diagonal wavefront technique, instead of others, was based on several factors. Firstly, due to the simplicity and ease of its implementation (relatively straightforward approach to parallelization). Indeed, the wavefronts can be easily mapped to threads, without using complex data partitioning or load balancing.

5 Experimental Setup

5.1 Dataset

To evaluate the performance of our parallelized algorithms, we use real-life genomic sequences obtained from UniProtKB [1], in .fasta format . The dataset consists of a diverse set of sequences with varying lengths and complexities, representing typical scenarios encountered in bioinformatics research.

5.2 Performance Metrics and Results

We measure the performance of the algorithms based on the inputs sizes, the number of cores, and execution time.

We present the performance results obtained from our experiments on the real-life genomic dataset.

In order to evaluate the performance of the algorithms depending on the inputs' size, we construct the following graph. We can observe the time in milliseconds taken for the alignment of two sequences of same size using between 1 and 8 threads.

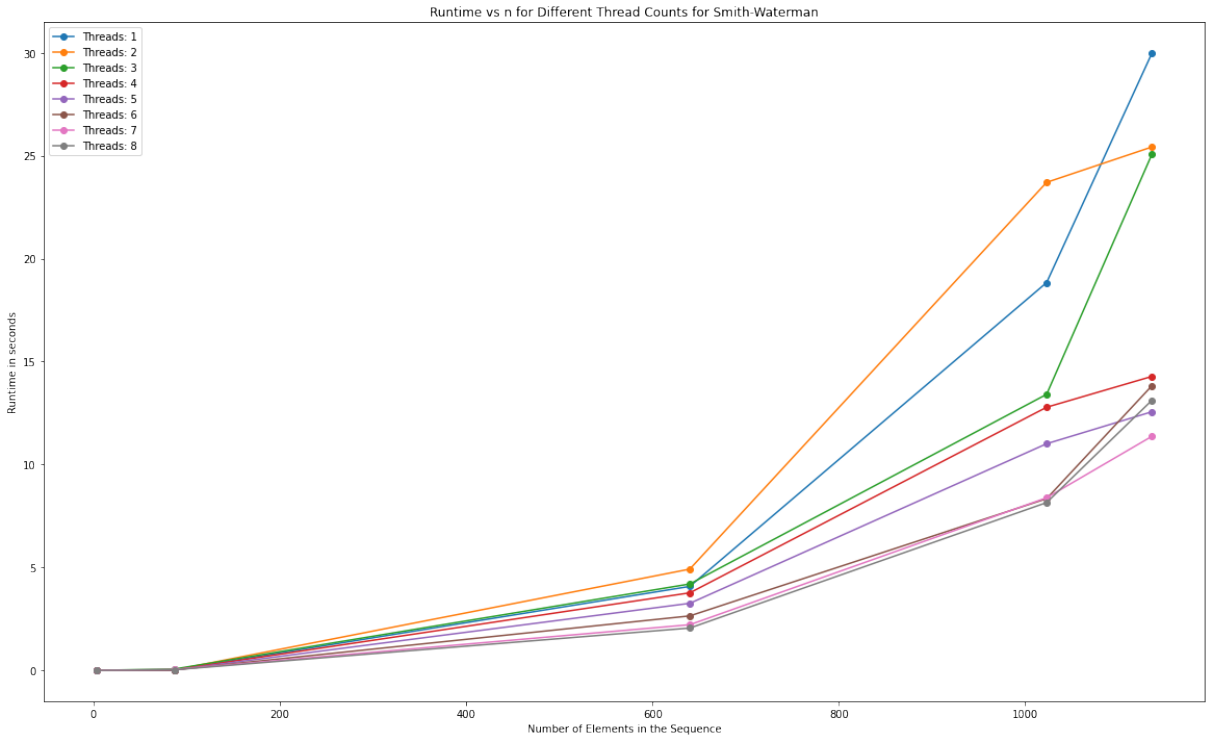

Figure 3: Graph of number of threads vs length of one sequence

[1] https://www.uniprot.org/

The time increases with the length of input as expected. We also notice that the higher the amount of threads, the faster the algorithm. The quickest is with 8 threads, which is logic since it is 'more parallelized'. For a number of threads higher than 8, the performance decreases again as the machine we are using has 8 cores (9 and 10 threads not shown in the graph).

The results demonstrate significant improvements in execution time for both algorithms when parallelized. Both the parallelized NW and SW algorithms shows a speedup compared to their sequential versions.

6 Further Discussion

As for further discussions, we can potentially further analyze certain key factors (the dataset size, algorithmic complexity, and hardware architecture) to better understand the impact of parallelization and provide insights into the scalability and limitations of our implementations.

We could further graph the number of thread in terms of the size of the matrix. We can imagine that the time would increase linearly in terms of the size of the matrix, thus implying that the time complexity of the parallelized algorithm only depends on the size of the matrix.

7 Conclusion

In conclusion, we have successfully implemented and parallelized the Needleman-Wunsch and Smith-Waterman algorithms for sequence alignment.The parallelized algorithms clearly outperform the sequential ones. Our experiments on real-life genomic data demonstrate the potential of parallelization in significantly improving the efficiency of sequence alignment tasks.

References

[NW70] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[RA04] S. Rajko and S. Aluru. Space and time optimal parallel sequence alignments. *IEEE Transactions on Parallel and Distributed Systems*, 15(12):1070–1081, 2004.

[RACDS03] Carlos Eduardo Rodrigues Alves, Edson Cáceres, Frank Dehne, and Siang Song. A parallel wavefront algorithm for efficient biological sequence comparison. pages 249–258, 05 2003.

[SA09] Abhinav Sarje and Srinivas Aluru. Parallel genomic alignments on the cell broadband engine. *Parallel and Distributed Systems, IEEE Transactions on*, 20:1600 – 1610, 12 2009.

[SBM16] Edans Flavius De Oliveira Sandes, Azzedine Boukerche, and Alba Cristina Magalhaes Alves De Melo. Parallel optimal pairwise biological sequence comparison: Algorithms, platforms, and classification. *ACM Comput. Surv.*, 48(4), mar 2016.

[SW81] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.