Massively Parallel Sequence Alignment

LINUS ZWAKA, Rensselaer Polytechnic Institute, USA
LOUIS PARRINELLO, Rensselaer Polytechnic Institute, USA
BRIAN HENRY, Rensselaer Polytechnic Institute, USA
LIAM GREEL, Rensselaer Polytechnic Institute, USA

Additional Key Words and Phrases: CUDA, MPI, Sequence Alignment, Nucleotide, Protein, Amino Acids, Parallel Performance, DNA, RNA, Biology

ACM Reference Format:
Linus Zwaka, Louis Parrinello, Brian Henry, and Liam Greel. 2024. Massively Parallel Sequence Alignment. 1, 1 (April 2024), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The field of biology has many areas of modern research within genomic data and despite this, many projects require sequencing and alignment of genome data such as DNA/RNA nucleotides or amino acids among proteins to determine similarities among them. Finding an accurate alignment can provide information about common ancestors, evolutionary conditions, and functional similarities [3]. Both building blocks, nucleotides and amino acids undergo random mutation most commonly through single substitution, insertion of a new section, or deletion of such sections [3].

For such a common overlap, even the most efficient algorithms remain computationally complex. Alignment by hand is possible for smaller sequences however with growing sequence length, the problem quickly grows beyond human effort.

In the modern era of supercomputers, many algorithms for aligning sequences have been developed however many were not designed with largely parallel machines of today including the world's top supercomputers like AiMOS at Rensselaer Polytechnic Institute. There are four formats for these alignment algorithms: global alignments, local alignments, pairwise alignments, and multiple alignment each having their own algorithms and use cases.

In order for computers to be able to tell apart a good from bad sequence, the algorithms give each tested alignment a score based on an algorithm dependent ruling based on the three states of a comparison: match, mismatch, or gap [1].

Aligning such data has proven a flexible task, solvable by various means but most notably by probability calculations through a Hidden Markov Model, heuristic functions, or dynamic programming. The scope of this paper will deal in calculating global alignments in order to produce output with certainty on an individual pair of alignments

Authors' Contact Information: Linus Zwaka, zwakal@rpi.edu, Rensselaer Polytechnic Institute, Troy, New York, USA; Louis Parrinello, parril2@rpi.edu, Rensselaer Polytechnic Institute, Troy, New York, USA; Brian Henry, henryb2@rpi.edu, Rensselaer Polytechnic Institute, Troy, New York, USA; Liam Greel, greell@rpi.edu, Rensselaer Polytechnic Institute, Troy, New York, USA.

ACM XXXX-XXXX/2024/4-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

while expanding to use smart heuristics for merging into a multi-alignment process.

2 NEEDLEMAN-WUNSCH ALGORITHM

The Needleman-Wunsch Algorithm is a dynamic programming solution to finding the highest scoring alignment across two sequences by breaking the problem of sequence alignment into the smaller problem of pair matching [1]. The algorithm achieves this by constructing a two-dimensional grid of dimensions *mxn* where m and n are the lengths of the data being sequenced. The steps in full for the algorithm are described as follows:

- Construct the grid
- Populate primary data
- Choose scoring method
- Populate dependent data
- Backtrace

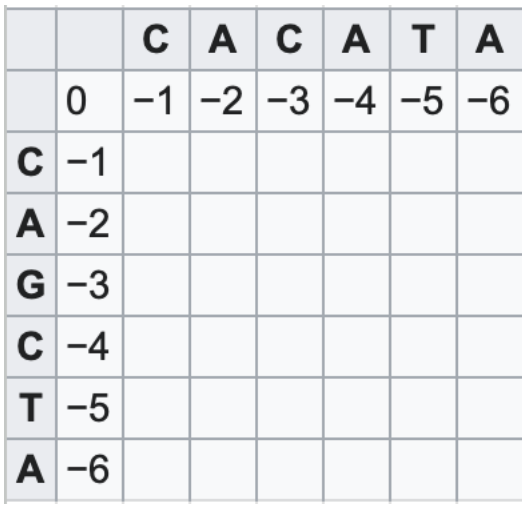
The proceeding subsections will cover the procedure for each execution step in greater detail.

2.1 Grid Construction

Accounting for the requirement that each pairwise comparison correlates to its own grid cell, the sequences are aligned in an *mxn* matrix such that the first two elements in the labeling row and column are empty (see Figure 1)

2.2 Populate Primary Data

Once the *mxn* grid has been made and the sequences are aligned along the edge, populate the first cell in the top left corner with 0 since this square does not represent any comparison. Since the scores in the first row and column have no previous iteration, they will be scored as if they have encountered a gap. The result is a decreasing sequence along the first row and column as seen in Figure 1. For consistency purposes, we have decided to always begin with the pictured configuration of decrementing by one along the first row and column for every grid.



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

Fig. 1. An example two-dimensional alignment grid with primary data as described by the Needleman-Wunsch algorithm

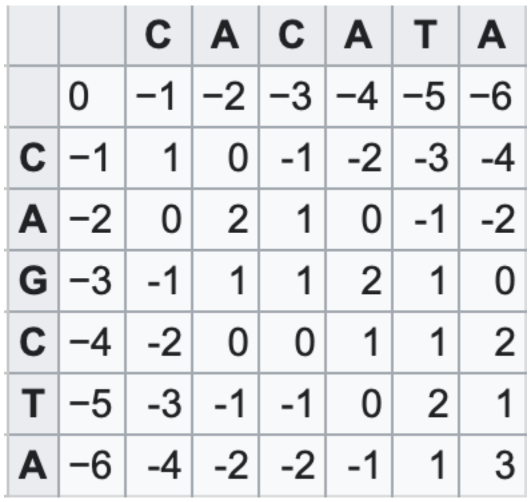
2.3 Scoring Method and Calculations

Scoring methods remain largely up to the discretion of each particular user however there are two main archetypes of scoring: matrix similarity and gap penalty. For reasons relating to our implementation, which will be described in the following section, we chose a gap penalty approach with the penalty being -1. Furthermore each cell with location (i, j) is scored in accordance with $self + max[(i-1, j), (i, j-1), (i-1, j-1)]$ where $self$ is determined from the scores for match or mismatch based on that specific cell's pairing. In our implementation, a match scores +1 and a mismatch scores -1. Figure 2 shows the result of this scoring algorithm applied to Figure 1.

2.4 Backtrace

Once the scoring matrix has been filled out, every pairing of the sequences has been calculated. In order to extract the highest scoring sequence; the best match, the algorithm begins in the bottom right cell and retraces the winners of the previous step's $max()$ argument.

- **Vertical $(\mathbf{i-1, j})$:** gap aligned with row argument



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

Fig. 2. A continuation of the alignment shown in Figure 1 after applying the scoring algorithm.

- **Horizontal** $(\mathbf{i}, \mathbf{j} - \mathbf{1})$**:** gap aligned with column argument
- **Diagonal** $(\mathbf{i} - \mathbf{1}, \mathbf{j} - \mathbf{1})$**:** no gap, match or mismatch

In the event of a tie between maximum scores, all of the alignments that tied are equally suitable matches and the possibilities branch at each backtraced value with multiple options.

3 IMPLEMENTATION

The Needleman-Wunsch algorithm proves itself able to be greatly parallelized due to its matrix-based calculations with minimal reliance [2][6]. Our improvements to the algorithm's traditional iterative implementation include using NVIDIA's CUDA library to enable highly parallel GPU calculation of grid cells through partially synchronized threads and reduced computation through our implementation of the backtrack step of the algorithm.

Furthermore, to expand beyond the original scope of the Needleman-Wunsch algorithm, we add an MPI layer to manage several alignments and perform the heuristic based star multiple alignment algorithm due to the continuation of dynamic programming to be an NP-complete problem with exponential complexity[10].

3.1 Single Alignment

The original implementation of the Needleman-Wunsch algorithm was designed for single alignment however it was developed and published before the popularity of parallel computing and thus implies an extreme iterative solution by calculating each cell one at a time, each time making three comparisons [1].

In more recent times, performance has increased by previously unimaginable margins. With the development of the CUDA suite, we have gained access to parallel code execution across many threads all running in massive parallel schemes due to the specialization of GPU cards.

Our kernel implementation is set apart from other approaches because instead of assigning one thread to an entire row, column, or diagonal strip of the grid, our implementation assigns one thread to one cell of the grid to perform calculations. Since the calculations for the current cell are only dependent on three cells of the last iteration, we can manually synchronize the threads on a delay meaning we no longer have to wait for each row to finish in its entirety and can start calculating as soon as those three dependencies are resolved.

This synchronization is implemented in our code by maintaining a separate zero-initialized grid containing the backtrace location for where the current cell must go to described by either 1 for diagonal, 2 for vertical, and 3 for horizontal. Each thread is placed in a waiting loop which periodically checks for all three dependency cells to have a non-zero value meaning they have been calculated in the score grid. Once the thread has approval from its dependent cells, it will calculate the score, backtrace code, and populate the appropriate cells in the score and backtrace grids.

The key feature in the synchronization method in our code is the "spinning" loop which waits for previous threads to finish computation. By implementing this loop instead of using a CUDA system call, we avoid forcing every thread to wait on a synchronization call and instead each thread starts immediately and remains unhindered for as long as the previous iteration remains calculated.

3.2 Multiple Alignment

A more advanced and useful approach is the concept of multiple sequence alignment (MSA). Utilizing CUDA and parallel computation, we can align tens or even hundreds of sequences simultaneously, greatly improving efficiency. MPI can facilitate distributing the workload across multiple nodes in a cluster, enhancing scalability.

While a brute-force dynamic programming approach exists for MSA, its runtime is often impractical. Parallelizing the computation with CUDA and MPI enables us to handle larger datasets and achieve faster alignment times. This approach leverages the computational power of GPUs and the scalability of distributed computing, making it suitable for analyzing complex biological sequences on modern high-performance computing systems.

There are several multiple sequence alignment algorithms, we stick with a simple star alignment algorithm implementation. This heuristic finds the sequence that is most similar to the rest of the sequences (center sequence) through a scoring matrix. We use a simple scoring method which applies a positive reward for matches and negative reward for non-matching elements.

After choosing a center sequence, we align all pairwise sequences with the center. Next, we iteratively merge the alignments using the aligned center sequence as a reference. For the purposes of our results, we only compute the scoring matrix and alignment stages as these are most computationally intensive and can utilize both CUDA and MPI for efficient parallel processing.

In MPI, the total number of alignments is divided by the number of ranks to assign each rank a fair share of the computation. After, we send the

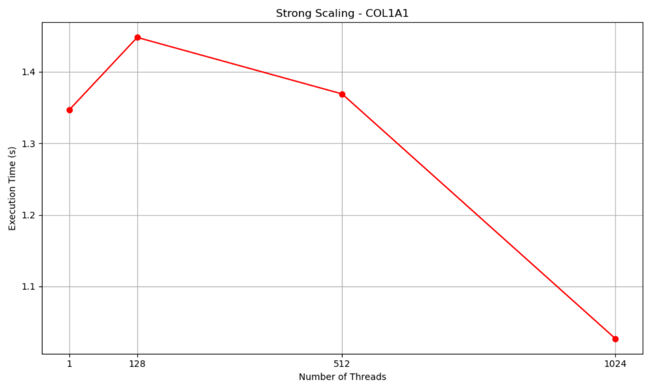


Fig. 3. Strong scaling of the Colla1 sequence

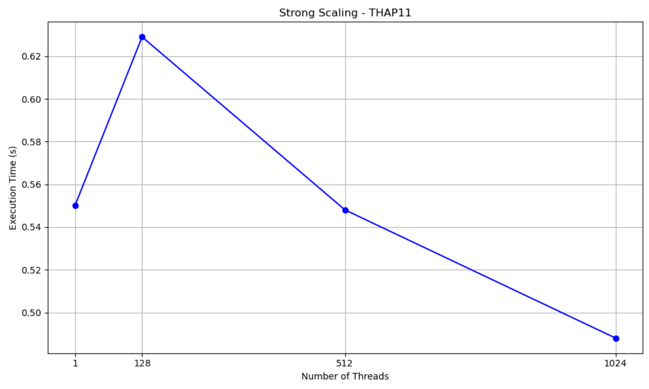


Fig. 4. Strong scaling of the Thap11 sequence

data to each rank and launch a CUDA kernel to align the sequences using our single sequence alignment implementation. After completion, the data is gathered back in main for further processing and merging. If we have n sequences of length k then we must compute n(n-1)/2 alignments where each alignment is $O(n^2)$.

Our total running time is thus $O(n^2k^2)$**. While this doesn't seem ideal, we found significant speedups by utilizing MPI and CUDA together to run the alignment process across several GPU nodes in a distributed system environment.

4 ANALYSIS

4.1 Strong Scaling

Comparing the problem scaling between our implementation and traditional serial implementation,

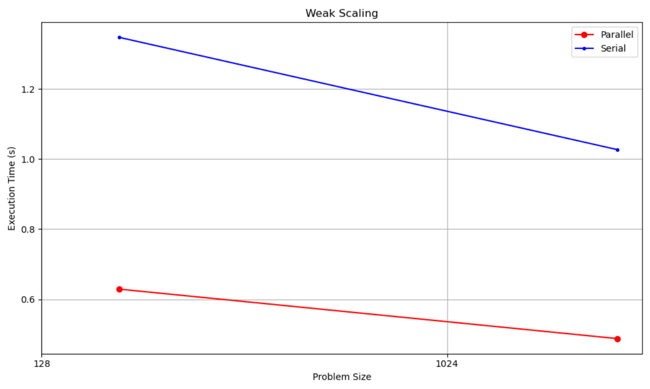


Fig. 5. Weak scaling of Thap11 and Colla1

we remained with the same problem but for each run increased the number of threads given to the program. As a result, we predictably saw the runtime decrease dramatically compared to the serial code (see Figure 3). This is due to the serial code only being able to calculate one cell at a time, finishing calculations of the entire row before being able to begin the cells with freed dependencies.

Observing a continual time improvement in our strong scaling experiments is likely due to the design of our kernel since threads are individually responsible for less work. Since each thread handles one cell instead of a whole row, they are able to be freed and reallocated to the next layer of oncoming cells much earlier than other implementations resulting in a cell with satisfied conditions to never wait for a thread to do calculations. This virtually zero wait time ensures maximum efficiency on thread counts of all sizes whereas traditional implementations have cells waiting for threads to finish several calculations before getting reallocated.

4.2 Weak Scaling

Comparing the serial code to our implementation, we see the data has decreased time due to the no waiting time explained in the strong scaling section

5 CONCLUSION

In this paper, we presented and implemented a stable, parallelized version of the Needleman-Wunsch algorithm engineered to run on CUDA enabled GPUs. Our version of the parallelized algorithm showed substantial improvements in deterministic global sequence alignments allowing us to further implement and combine with a star alignment heuristic method for multiple sequence alignment.

Our hope is that given the dynamic programming approach for multiple alignment is an NP-complete problem with exponential time, future writers will improve and expand on our multiple alignment implementation as to decrease the likelihood of a missing optimal alignment or improved CUDA-MPI integration.

6 REFERENCES

(1) Needleman, S. B., & Wunsch, C. D. (1970). 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. https://doi.org/10.1016/0022-2836(70)90057-4
(2) A GPU based implementation of Needleman-Wunsch algorithm using skewing transformation | IEEE Conference Publication | IEEE Xplore. (n.d.). Retrieved April 23, 2024, from https://ieeexplore.iee
(3) Altschul, S. F., & Pop, M. (2017). Sequence Alignment. In K. H. Rosen, D. R. Shier, & W. Goddard (Eds.), Handbook of Discrete and Combinatorial Mathematics (2nd ed.). CRC Press/Taylor & Francis. http://www.ncbi.nlm.nih.gov/books/NBK464187/
(4) Chakraborty, A., & Bandyopadhyay, S. (2013). FOGSAA: Fast Optimal Global Sequence Alignment Algorithm. Scientific Reports, 3, 1746. https://doi.org/10.1038/srep01746
(5) Chao, J., Tang, F., & Xu, L. (2022). Developments in Algorithms for Sequence Alignment: A Review. Biomolecules, 12(4), 546. https://doi.org/10.3390

(6) Gancheva, V., & Georgiev, I. (2019). Multithreaded Parallel Sequence Alignment Based on Needleman-Wunsch Algorithm. 2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE), 165–169. https://doi.org/10.1109/BIBE.2019.00037.

(7) Li, D., & Becchi, M. (n.d.). Multiple Pairwise Sequences Alignments with Needleman-Wunsch Algorithm on GPU.

(8) Lin, Y.-S., Lin, C.-Y., Li, S.-T., Lee, J.-Y., & Tang, C. Y. (n.d.). GPU-REMuSiC: the implementation of Constrain Multiple Sequence Alignment on Graphics Processing Units.

(9) Thompson, J. D., Plewniak, F., & Poch, O. (1999). A comprehensive comparison of multiple sequence alignment programs. Nucleic Acids Research, 27(13), 2682–2690.

(10) Zou, Q., Shan, X., & Jiang, Y. (2012). A Novel Center Star Multiple Sequence Alignment Algorithm Based on Affine Gap Penalty and K-Band. Physics Procedia, 33, 322–327. https://doi.org/10.101

Received 23 April 2024