Global Sequence Alignment

SPP Project

Team Members:
1. Shreyansh Agarwal (2020101013)
2. Ayush Agrawal (2020101025)

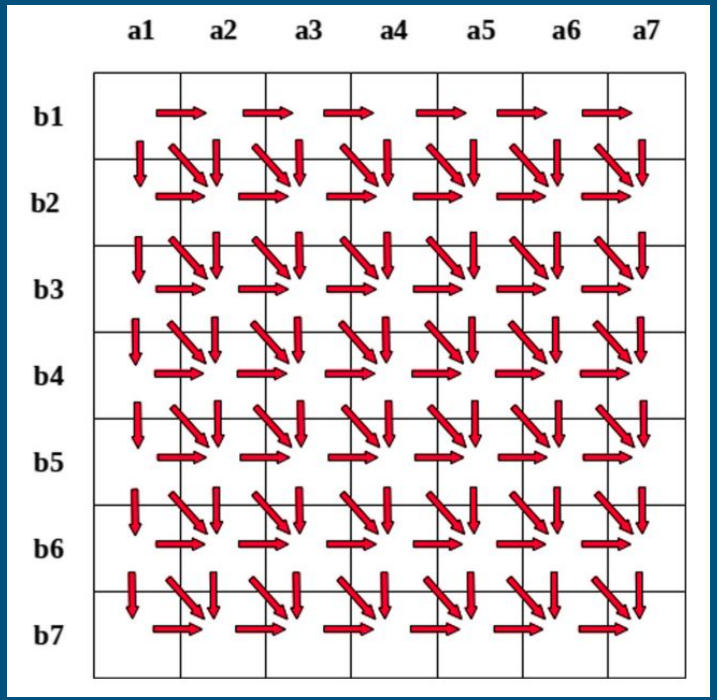
Sequence Alignment

- DNA consists of sequence of bases which determines its structure and functions.
- Aligning two sequences helps us to detect similarities and mutations, discover homology, to find variations responsible for diseases etc.
- Allows us for gene comparison within the population.
- Allows to to understand evolution by comparing genes within the species.

Global Alignment

- Aligns complete sequences
- Dynamic Programming Algorithm to compute the optimal score given a scoring scheme, proposed by Needleman and Wunsch
- Time Complexity: O(mn)
- Space Complexity: O(mn)
- Gives the optimal alignment
- Can be time-taking if sequences are long

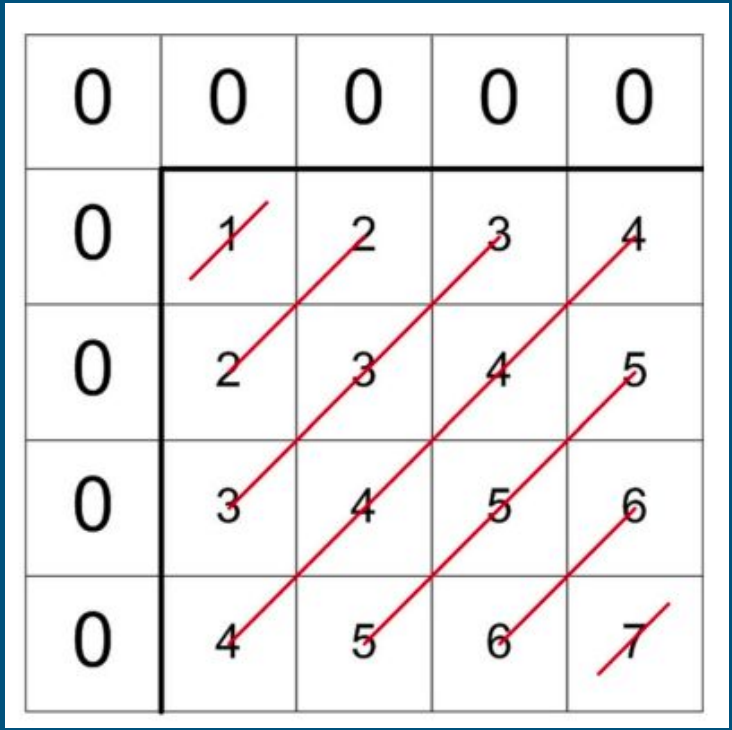
Dependency Analysis of the Algorithm



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

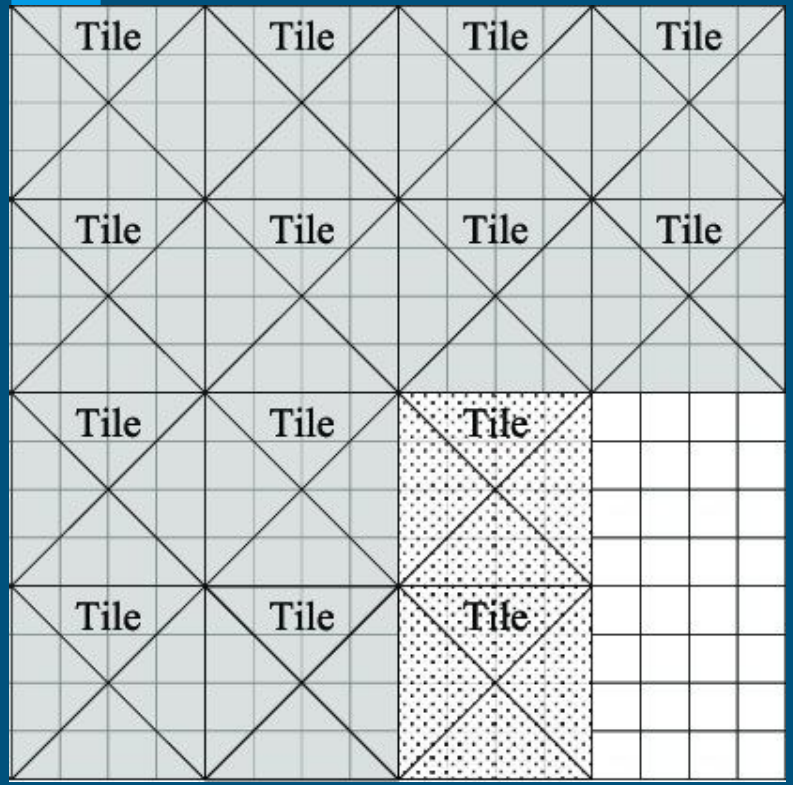
Not parallelizable along row, column or diagonal

Anti-Diagonal Parallelization



- We observe that there are no dependencies along the anti-diagonal !
- Requires the values from i-1 and i-2 diagonal for computing values in the ith diagonal
- We can parallelize along an anti diagonal.
- Parallelization at the cost of cache performance.

Tiling Approach



- Divide the whole DP matrix into tiles of size k.
- Each tile will have k rows and k columns.
- The tiles along a diagonal can be computed in a parallel fashion.
- The computation of cells within a tile can be done as per the original method.
- This combines the advantage of parallelization with cache coherency.

Correctness Evaluation Approach

- Executed the algorithm on two sample sequences.
- Compared the final DP Matrix with the result of online calculator
- Verified that the DP Matrix was correct.
- Compared the Traceback Matrix as well as final alignment with the results of online calculator.
- Repeated the process for few sequences to establish that our program is accurate

Compute Configuration

- Operating System: Ubuntu 21.04
- Processor: Ryzen 5-4600H
- Cores: 6 (12 threads)
- RAM: 16 GB
- CPU Frequency: 1400 MHz (Upto 4000 Mhz)
- L1, L2, L3 Cache sizes: 192KB, 3MB, 8MB

Performance Evaluation Framework

- Execution time is the main criteria for judging the performance of different implementations.
- Executed the different algorithms for varying length of DNA sequences.
- Calculated GINTOPS/s and memory bandwidth for different algorithms

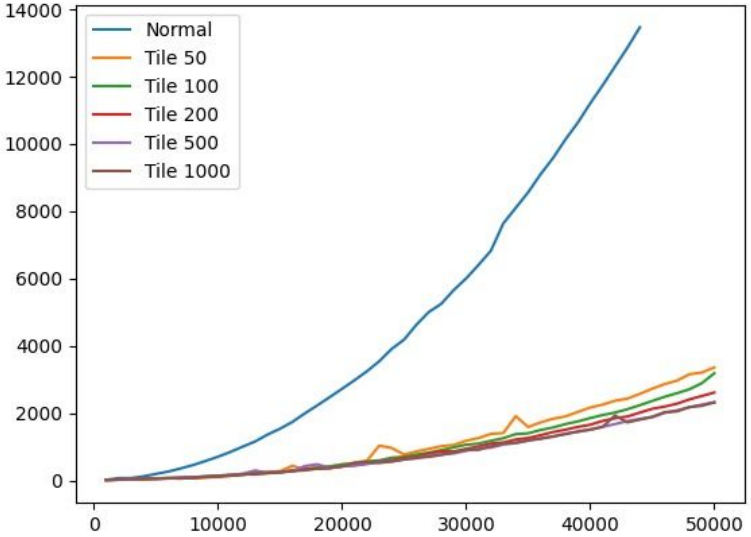
Datasets Used

- Random DNA sequences of length 1000 - 50000.

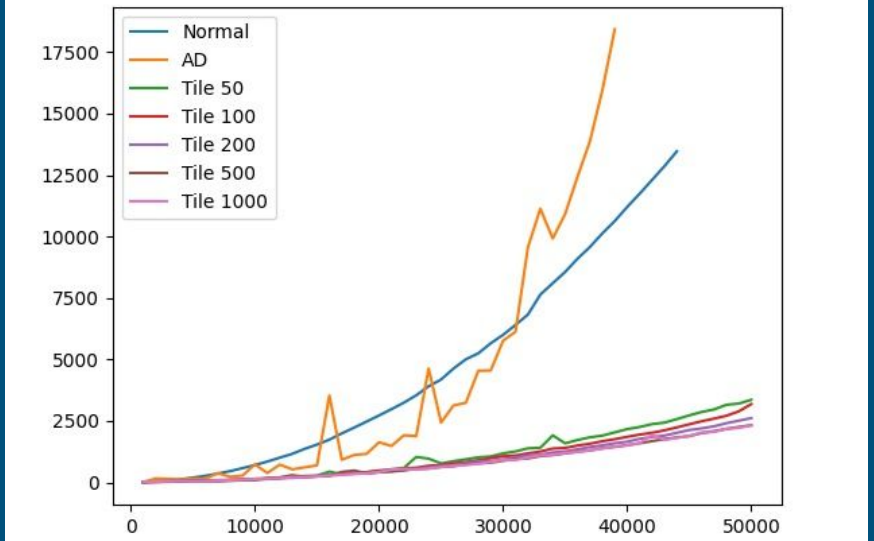
Performance

- Baseline Performance: 48812 ms, for a sequence size of 42000 (using naive row wise sequential, icc compiler and without any optimization flags)
- Best Performance: 1671 ms, for the same sequence size (using tile based method, parallelization of tiles along a diagonal, -O3 flag, icc compiler)
- Speed Up: 48812/1671 = 29.21 times
- Speed Up: 12860/1671 = 7.69 times (using -O3 flag in rowwise)
- GINTOPS/s : 6.13 GINTOPS/s (Best Implementation), 0.215 GINTOPS/s (Baseline Implementation), 0.841 (Baseline with -O3).
- Memory Bandwidth: 14.31 GB/s (Best Implementation), 0.50 GB/s (Baseline Implementation), 1.96 GB/s (Baseline with -O3).

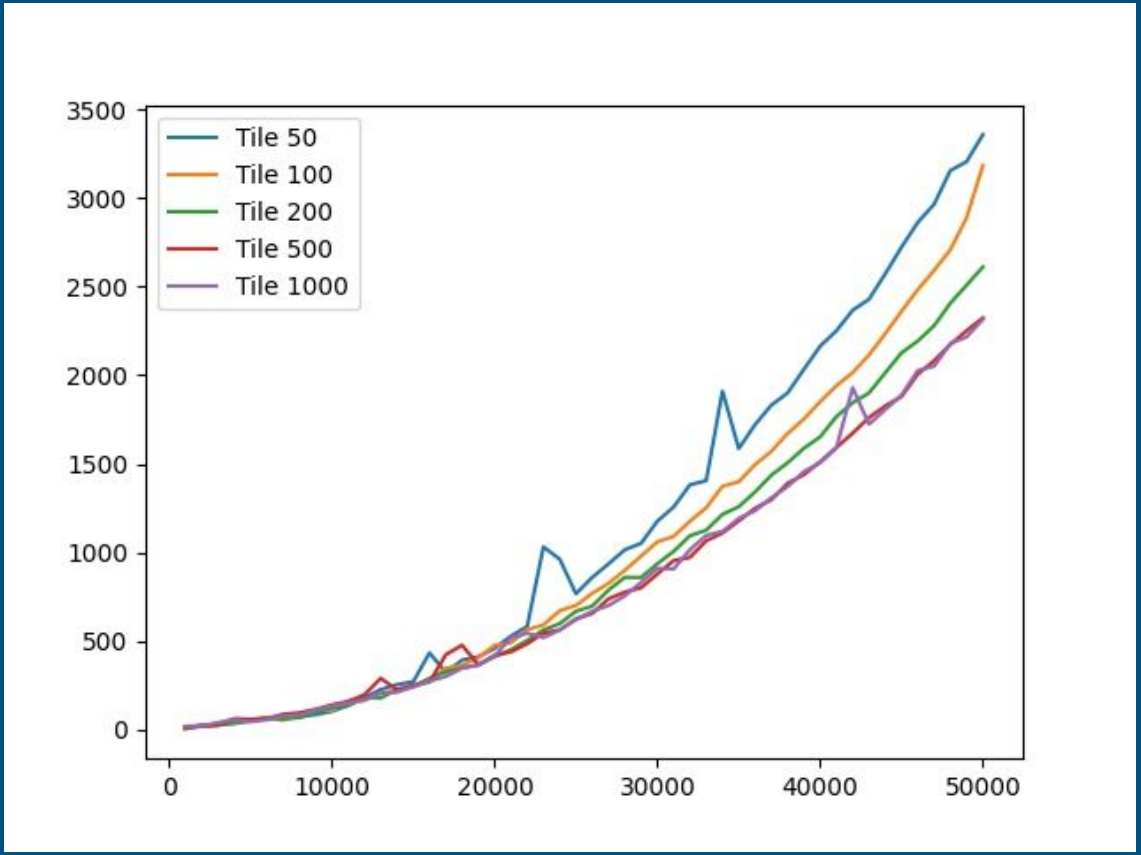
Graphs



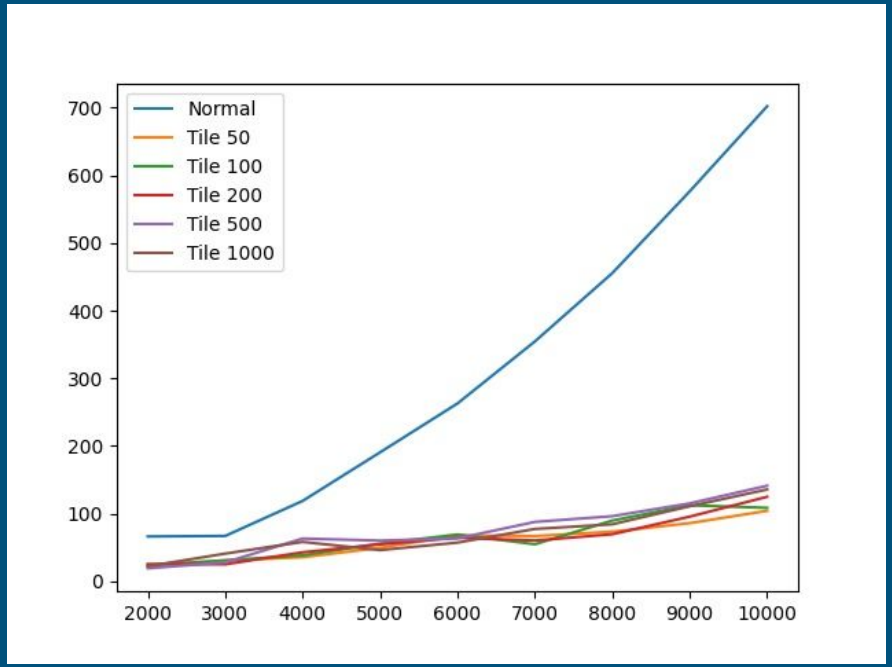
Time(in ms) vs Length of Sequence Graph for basic and tiled implementations



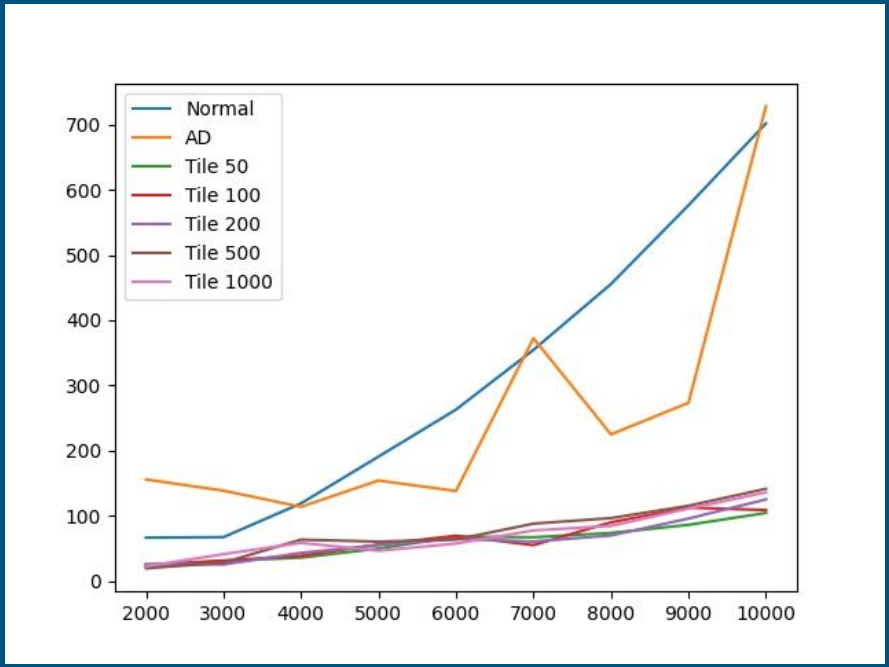
Time(in ms) vs Length of Sequence Graph for basic, anti diagonal and tiled implementation



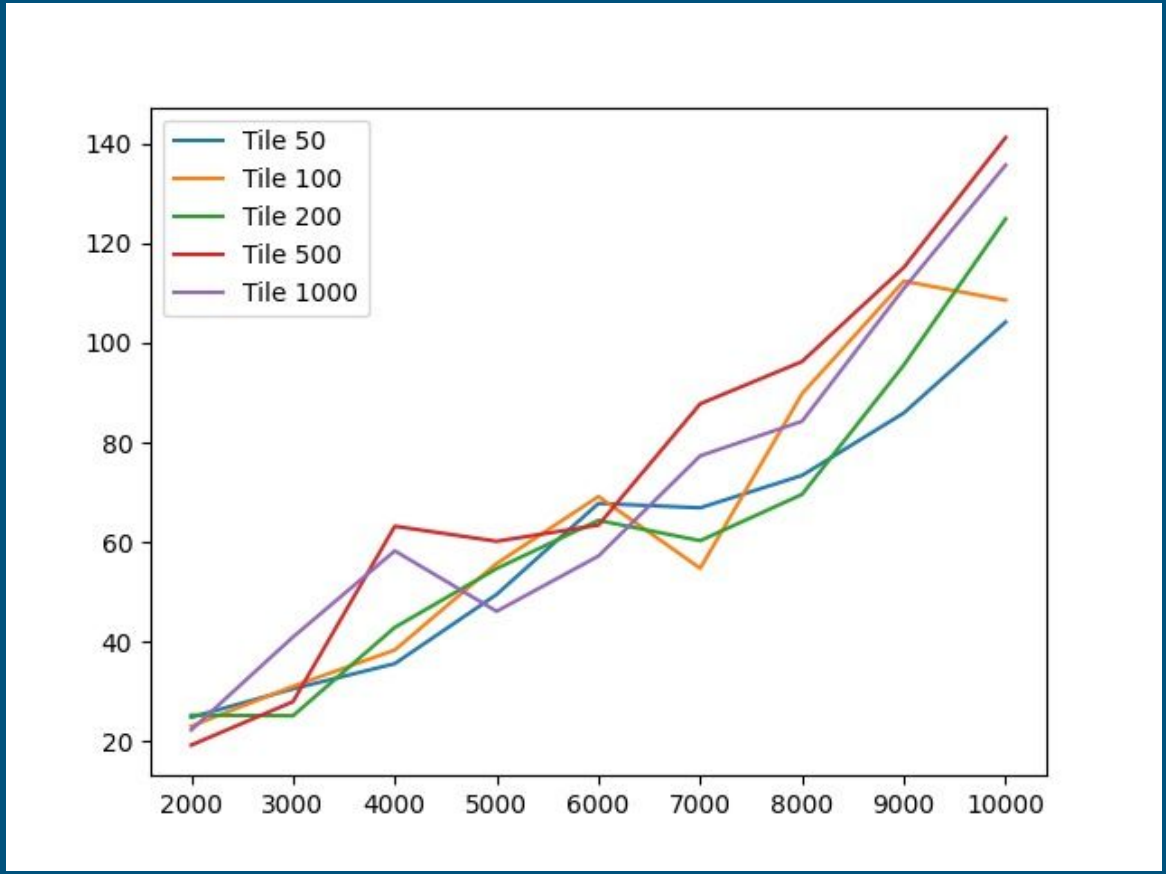
Time(in ms) vs Length of Sequence Graph for tiled implementation with various tile sizes



Time(in ms) vs Length of Sequence Graph for basic and tiled implementations (N = 1000 to 10000)



Time(in ms) vs Length of Sequence Graph for basic, anti diagonal and tiled implementation (N = 1000 to 10000)



Time(in ms) vs Length of Sequence Graph for tiled implementation with various tile sizes (N = 1000 to 10000)

Insights and Future Work

- Careful analysis has to be done while parallelizing the Needleman Wunsch Algorithm
- Comparison of 3 methods: Basic, Anti Diagonal and Tile based
- Can use the GPU using OpenCL or CUDA framework to speedup calculations
- Using heuristic based methods