**Parallel implementation of Sequence Alignment**

**Final project**

**Documentation file**

**Intro:** This program performs Parallel implementation of Sequence Alignment.

Given an input file containing Weight Coefficients, Sequence Seq1, and few Sequences Seq2, the program finds the max score of each seq2's and prints the offset index, mutant index, and max score of each one.

**Input:**

* Weight coefficients W1, W2, W3, W4
* Seq1
* The number NS2 of Sequences Seq2 to check against Seq1
* Seq2 in each line

**Output:**

This file contains NS2 lines with Offset index, Mutant index, and a max score of each seq2.

**Parallelizing the algorithm:**

1. MPI -

The MPI is performed using in a static method (I preferred to use this method because we know the number of seq2 from the input file)

* + The program ran with 2 processes, Master (root) and Slave.
  + In the first step, the master reads the input file and sends the data to the slave via broadcast.
  + Next, half of the strings (the seq2's) the master takes to calculate, and the remaining half the slave takes.
  + When the slave finishes calculating the results, he sends them to the master.
  + The master received the results from the slave (and in addition, he also has the results he calculated himself)
  + now, the master can print the results (to the output file and the console)

I chose to parallelize the strings of seq2 with MPI because each seq has independent calculations. In addition, each process has a lot of calculations to do and it's fit to use process or a different computer for this heavy task and not threads.

1. OMP - parallel offsets calculation
   * Using the Omp library I parallelized the task of calculating all the offsets.

It's a big array size that should be parallel. It's a loop that calculates the bast offset and the best result. Therefore, the omp fits perfectly for this task.

1. CUDA -
   * Using Cuda I parallelized the task of calculating the score of each mutant. There are a lot of mutants, and each mutant requires a simple calculation – and Cuda fit for this task. As we learned in class, with Cuda we can use a lot of threads and give each thread a simple tiny task.
   * What GPU returns to CPU is an array of all scores that calculated in parallel.
   * In addition, at class we also learned that copying the data from a host (CPU) to the device (GPU) is a time-consuming task, therefore, in order to use Cuda efficiently, I initialized Cuda data as the first task of the slave and master with a constant data (the matrix of weighs and the seq1) – that’s lead to copy the data from the host to the device only once.