Parallel implementation of Sequence Alignment

Final project

Course 10324, Parallel and Distributed Computation

2020 Spring Semester

Developer: **Vladislav Chebanov**

ID: **323682294**

Project definition:

Sequence Alignment – a way to estimate a similarity of two strings of letters - is an important field in bioinformatics. Sequence is a string of capital letters including hyphen sign (-), for example

```
PSHLQYHERTHTGEKPYECHQCGQAFKKCSLLQRHKRTHTGEKPYE
```

Each letter in the sequence represents DNA, RNA, or protein. Identification of region of similarity of set of Sequences is extremely time consuming. This project deals with a simplified version of Sequence Alignment of two sequences. The purpose of the project is to parallelize the basic algorithm to produce an efficient computation within MPI, OpenMP and CUDA environment.

Environment work explanation:

- **MPI:**

I use MPI in this project to split the working load for two computers based on the same LAN network. The main computer reads data from the "input.txt" file and then sends the data to a second computer. After the send the main pc starts computing the first half of the data while the second computer splits the data and uses the other half for computation. After they are done computing, the slave pc sends the data to the master, which writes the results to the "output.txt" file ordered by the initial order of sequences.

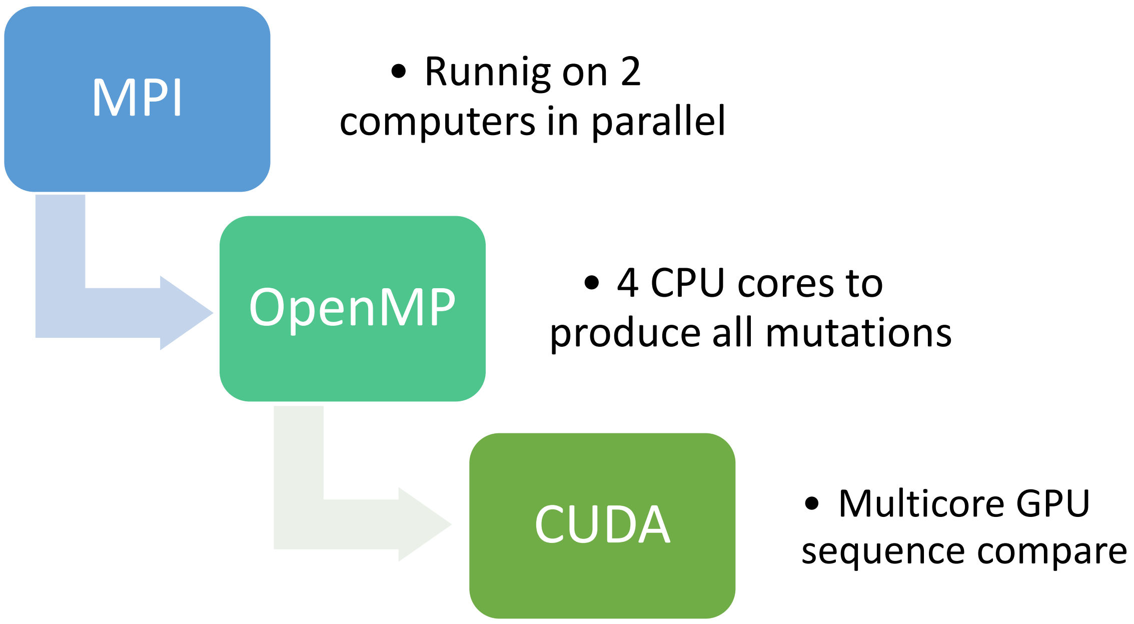
- **OpenMP:**

OpenMP is used for parallel preparation of mutation matrix. Every core receives a part of a loop and then appends the data into a 2d array of chars. After it is done, he sends the matrix to CUDA.

- **CUDA:**

The main activity happens in CUDA. CUDA receives an array of all the mutation possible for one string. Then the program uploads the data to the graphics card memory. With the power of CUDA cores it makes compares of all the mutations in parallel with blazingly fast speed and returns an array of chars for counting maximum score. Then we repeat the process with offset. When calculations are done, CUDA rolls back the results for file outputting.

Illustration of work.


MPI
• Runnig on 2 computers in parallel
OpenMP
• 4 CPU cores to produce all mutations
CUDA
• Multicore GPU sequence compare

The rational of choosing the specific architecture:

- **MPI –** we want our solution to be run at as many free computers as possible to receive the fastest solution possible (limits of project definition are 2 computers).

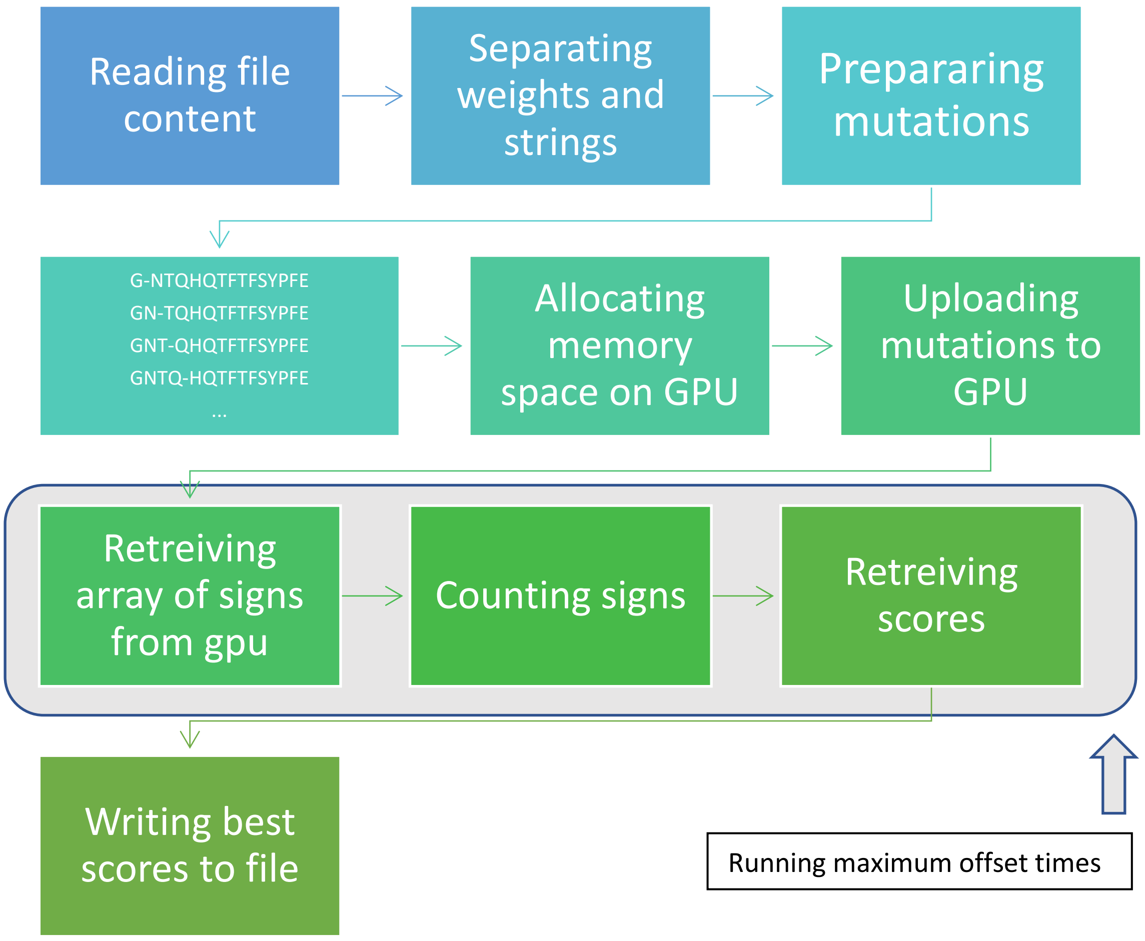
- **OpenMP –** is a great architecture for parallelizing loops for example. Therefore, I have chosen it to produce mutations. However, I have not chosen it for score calculation because score calculation is a reduction problem. If we use parallel computation for reduction, we need to use locks and synchronizations of threads which makes our solution sequential.

- **CUDA –** with GPU multicores we can easily compare two sequences in parallel. This makes the comparation fast and very efficient. However, CUDA memory is limited. In this case we want our mutations to be divided into chunks.

Quick run example with one computer:

- "input.txt"

```
2  1.5  1.1  1.3
MNMLWVVSGQTYQQLPVDFKTFRQATVGNTQHQTFTFSYPFE
1
GNTQHQTFTFSYPFE
```


Reading file content
Separating weights and strings
Prepararing mutations
G-NTQHQTFTFSYPFE
GN-TQHQTFTFSYPFE
GNT-QHQTFTFSYPFE
GNTQ-HQTFTFSYPFE
...
Allocating memory space on GPU
Uploading mutations to GPU
Retreiving array of signs from gpu
Counting signs
Retreiving scores
Writing best scores to file
Running maximum offset times

- "output.txt"

```
n = 27 k = 15
```

Limits of the project:

Project defines limits of 3000 signs for sequence 1 and 2000 signs for sequence 2. However, this program is limited only by GPU memory space. GPU memory maximum size is 2,048 MBytes or 2,147,483,648 bytes. Therefore, we divide mutations in chunks of 3 in case of its size is over the limit.

Project defines use of up to two computers via MPI. This project can be run on one or two machines.

Fail safe:

All the functions (memory allocations, compare functions…) in this project are wrapped by macro used to check the result correctness. In case of bad answer the program exits and massage is written.

Time estimation:

After computation is done massage is printed showing how much time took for computing one sequence. Maximum time for longest compare is ~40 seconds.

Complexity evaluation:

- MPI – (sequences number * sequences length) / number of computers
- OpenMP – sequence length / number of cores
- CUDA – (sequence length * mutated sequence length) / (CUDA blocks / CUDA threads)

README.txt file content:

```
================================================================================
 Name          : DNA.c
 Author        : Vlad Chebanov
 Version       : FINAL
 Description   : Parallel implementation of Sequence Alignment
================================================================================

Running requirements:
        1) GPU with CUDA API
        2) "inc" folder with CUDA libraries
        3) linux operating system
        4) nvcc CUDA compiler
        5) mpicc MPI compiler

Configurating project properties:
        1) Open "defines.h".
        2) Edit whatever you need.

Input File format:
        First line - weight coefficients W1, W2, W3, W4.
        Next line – Seq1 (not more than 3000 chars in line).
        Next line – the number NS2 of Sequences Seq2 to check against Seq1.
        Next NS2 lines - Seq2 in each line (not more than 2000 chars in each line).
        Example:
                2  1.5  1.1  1.3
                MNMLWVVSGQTYQQLPVDFKTFRQATVGNTQHQTFTFSYPFE
                3
                GNTQHQTFTFSYPFE
                EGQATVGNT
                MNMLWVVSGQTY

Output File format:
        This file contains NS2 lines with n and k found for each Sequence Seq2 from the input file, in order
        Sequences Seq2 appear in the input file, where n is an Offset and k is a hyphen location of the Mutant
        Sequence MS(k) with the best Alignment Score.
        Example for input from above:
                n = 26 k = 1
                n = 21 k = 9
                n = 0 k = 12

Building the project:
        1) Open terminal in current folder.
        2) Check that CUDA version in makefile is right.
        3) Run "make".

Cleaning project folder:
        1) Open terminal in current folder.
        2) Run "make clean" ("make clean" will delete all the .o, the main executable and output.txt files).

Running the project:
        1) Put inside current folder file with data named "input.txt"
        2) Open terminal in current folder.
        3) For one thread run "make run1".
        4) For two threads run "make run2".
        5) For two threads on cluster run "make runOn2" after you create "mf" file with the computers ip's.
```

Makefile content:

```
build:
	mpicxx -fopenmp -c DNA.c -o DNA.o
	mpicxx -fopenmp -c SequencesComparator.c -o SequencesComparator.o
	mpicxx -fopenmp -c IO.c -o IO.o
	nvcc -I./inc -c cudaFunctions.cu -o cudaFunctions.o
	mpicxx -fopenmp -o DNA DNA.o SequencesComparator.o IO.o cudaFunctions.o
		/usr/local/cuda-9.1/lib64/libcudart_static.a -ldl -lrt

clean:
	rm -f *.o ./DNA output.txt

run1:
	mpiexec -np 1 ./DNA

run2:
	mpiexec -np 2 ./DNA

runOn2:
	mpiexec -np 2 -machinefile  mf  -map-by  node  ./DNA
```