Final Project Parallel And Distributed Computer

**Parallel implementation of Sequence Alignment**

Irit Ashuri 204605109

MPI + OpenMP + CUDA

**Problem Definition:**

First, let’s define 2 groups of characters groups.

The first group contains groups who call Conservative Groups, the second group contains groups who call Semi Conservative Groups.

* Conservative Groups: NDEQ, NEQK, STA, MILV, QHRK, NHQK, FYW, HY, MILF.
* Semi Conservative Groups: SAG, ATV, CSA, SGND, STPA, STNK, NEQHRK, NDEQHK, SNDEQK, HFY, FVLIM.

By given a group of letters called Seq1 and a group of letters called Seq2 (when Seq2 must have at least 1 letter less the Seq1), we create a new string called Similarity String which contains the characters \*,:, .,' ', in the following way:

* Seq2 places under Seq1 with offset n from the start of Seq1 and a hyphen inserting after MS (Mutant Sequences) letters in Seq2. Seq2 does not allow to pass behind the end of Seq1.
* The letters from Seq1 that do not have a corresponding letter from Seq2 are ignored.
* If these letters are identical the pair is marked with Star sign (\*).
* Otherwise, the letters are checked if they both present at least in one of the 9 Conservative Groups. In case that the pair is found in one of the Conservative Group it is marked with Colon sign (:).
* If no Conservative Group is found, the pair is checked against 15 Semi-Conservative Groups. If the pair do present in one of the Semi-Conservative Groups, it is marked with Point sign (.).
* If the letters in the pair are not equal, do not present both not in Conservative nor in Semi Conservative groups – the pair is marked with Space sign (‘ ‘).
* MS location must be up or equal to 1, offset location must be up or equal to 0.

At the end of the check process, the whole Sequence of Signs is obtained. This Sequence is used to estimate the similarity of two sequences – Seq1 and Seq2.

For this project the following formula is used to estimate the Alignment Score:

**S = W1\*NumberOfStars - W2\*NumberOfColons – W3\*NumberOfPoints - W4\*NumberOfSpaces**

where Wi is the given weight coefficients.

For all Mutant Sequences and offsets n of Seq2, we need to choose the Mutant Sequence MS(k) and the offset n with the max Alignment Score.

**Project instruction:**

In this project, we get an input text file with Weight Coefficients, Sequence Seq1, and a few Sequences Seq2 (NS2).

For each Seq2 we return the MS and offsets which provide the max Alignment Score according to the definition above.

The result will be written to an output text file.

**Division of roles:**

**MPI:**using 2 processes – master and slave.

The master read all the data from the input file, store it, and send half of the Seq2 array to the slave (if he got an uneven amount of Seq2 he will send to the slave one more Seq2) and all the rest of the values he read.

The salve gets the information from the master and each one of them starts to work on his part of the Seq2 array.

The slave sends the results to the master, the master gets it, and write all the results to an output file.

**OMP:** each process uses the OMP library to separate the seq2 array he got in a way that each thread is calculating the results of one Seq2 at a time when all the thread working simultaneously.

Each thread will get an array of all optional Alignment Scores from the GPU, sort it and return the offset and the MS with the best result.

**Cuda:** each thread is sending the seq2 he is currently working on and the rest of the necessary data to the GPU, the GPU is calculating for each MS and offset option his Alignment Score simultaneously and send all of the results to the thread who called it.

The number of threads we use is according to the following calculation –

- For each block, the maximum thread allowed is 512.

- The number of threads that are going to work is the number of options we have to put the hyphen in Seq2 (which is equal to seq2 length) multiple with the number of options we have for offsets (which is equal to Seq1 length – Seq2 length).

- The number of blocks is num\_of\_threads / 512. If there is redundancy to this calculation, we add one more block.

**The Alignment Score calculation implementation**

Each thread inside of the GPU gets w array which contains the given weight coefficients, Seq1 Seq2.

According to the thread id and block id it calculates what MS (hyphen location) and what offset it calculate Alignment Score for.

**Ms = threadIdx.x + (blockDim.x \* blockIdx.x) % max\_ms + 1**

**Offset = threadIdx.x + (blockDim.x \* blockIdx.x) / max\_ms**

**Max\_ms – the number of options we have to put the hyphen**

 Its compare each latter according to the description above when instead of creating a new string it uses a similarity array In size 4, when similarity[0] represents the number of ‘\*’, similarity[1] represents the number of ‘:’, similarity[2] represents the number of ‘.’, and similarity[3] represent the number of ‘ ’.

Instead of writing the similarity string characters, every time a character needs to write the corresponding location in the similarity array increase by 1.

At the end the Alignments Score calculation is performed as following:

Alignments Score = w[0]\* similarity[0] – w[1]\* similarity[1] – w[2]\* similarity[2] – w[3]\* similarity[3]

The result is comping back to the CPU (the host) and the CPU store it in an array.

After he got all the results, he sorts it and chooses the max Alignments Score.

**The rationale of choosing this architecture:**

I wanted the code to be able to run on two different computers so I used at first MPI library that allowed me to separate the problem to two and each computer will have to dill with half of the sequences.

When using OPM, there is an option to use only as many as cores the computer have simultaneously so I wanted each thread will work on one Seq each time and be helped by the CPU.

The CPU is working fast with no limitation of thread numbers, so I wanted it to get the heaviest calculation.

**Execution Time:**

**Definition:**

* Seq1 length = n1
* Each Seq2 length = n2
* Number of Seq2 = NS2
* Length of total characters in Seq2 array = = total\_n2
* Size of w = 4
* Size of conservative group = C
* Size of semi-conservative group = semiC

\*Execution time is depending on the number of cores each computer has and on how many computers are working (1 or 2).

**Running time for two computers:**

Master – computer1:

1. read from file

2. send data to the slave computer

3. for each core in the cp:

             \*\* number of offset Options =

length of Seq1 – length of Seq2 – 1 = n1 – n2 – 1

             \*\* number of Ms options = length of Seq2 = n2

3.1 GPU calculation for each optional offset and MS:

             3.3.1. compute seq2 length – o(n2)

             3.3.2. compute MS and offset to test - O (1)

3.3.3. calc similarity for each character –

go over each character **ch** in Seq1, start from the character in the offset location (o(n2)):

* for the corresponding in Seq2, check if its equal O (1).
* if not, go over each character in conservative groups and check if **ch** is in there and if yes check if the corresponding character from Seq2 is in the same group – O(C).
* if not found a match on conservative groups, do the same test on semi-conservative groups O(sC).

Total execution time = O(n2\*(C+sC)) = O(n2)

**Total running time for each thread in the GPU = O(n2)**

**Because all threads are working simultaneously the total running time for the core to calculate all optional results is equal to O(n2).**

     3.2 Find Best result -> go over each result and check if it’s the best –

O ((n1 – n2 – 1) \* n2) = O(n1n2)

4. get results from slave.

5. write all results to a file.

**Total core running time = O(n2) + O(n1n2) = O(n1n2)**

**Because all the cores are working simultaneously the total running time is O(n1\*max(n2)) \* (NS/2) if NS/2%number\_of\_cores == 0,**

**else O(n1\*max(n2)) \* (NS/2) + 1 =**

**O(n1\*max(n2))**

Slave – computer2:

1. Get data from the master.
2. Call al cores and do the same calculation as Master does -

O ((n1 – n2 – 1) \* n2) = O(n1n2)

1. Send results to the master.

The total running time is similar to the Master running time

**O(n1\*max(n2)) \* ((NS/2)/ number\_of\_cores) if NS/2%number\_of\_cores == 0, else O(n1\*max(n2)) \*( (NS/2)/ number\_of\_cores ) + 1**

**= O(n1\*max(n2))**

Eventually because the 2 off the computers are working simultaneously the running time of the program is O(n1\*max(n2)) \* ((NS/2)/ number\_of\_cores) if NS/2%number\_of\_cores == 0 , else O(n1\*max(n2)) \*( (NS/2)/ number\_of\_cores ) + 1

🡺 O(n1\*max(n2))

**for one computer:**

The running time is like above, but now we are using in each process in number\_of\_cores/2 cores.

So the total running time is **O(n1\*max(n2)) \* ((NS/2)/ (number\_of\_cores/2)) if NS/2%number\_of\_cores == 0 , else O(n1\*max(n2)) \*( (NS/2)/ (number\_of\_cores/2) ) + 1**

🡺 **O(n1\*max(n2))**

**For sequential solution:**

\*\* number of offset Options = length of Seq1 – length of Seq2 – 1

= n1 – n2 – 1

\*\* number of Ms options = length of Seq2 = n2

1.for each Seq2

- For each optional offset and optional MS, we do similarity test which takes O(n2), that means we do it (n1 – n2 – 1\*n2) times 🡺 O(n2\*n1(n2))

- Find the best solution O(n2).

2. we do NS2 times.

So the total execution time is 🡺 O(max(n2) \* n1(max(n2)))\*NS2