Documentation

Daniel Ivkovich - 316421262

**Problem Definition**

Let Seq1 and Seq2 be a given Sequences of letters. For all Mutant Sequences of Seq2 find an offset n which produce a maximum Alignment Score against Seq1. Among all results, choose the Mutant Sequence MS(k) with the best Alignment Score.

**Solution**

I have chosen to solve this problem by combining the capabilities of MPI, CUDA, and OMP interfaces in a specific way the computation of the algorithm shall be more efficient in terms of time complexity.

**Usage of MPI in the project**

In this solution, the outer wrapping of the algorithm is done via the MP interface.

In the first part, MPI splits the job evenly among the (**size**) processes that were assigned to the program at execution. Each process will read the entire information from the input file and convert the information into appropriate data structures that I have created beforehand:

struct MainSequence – will hold the main Sequence, its length **L1**, and the scoring weights.

struct Sequence – will hold a single minor Sequence and its length **L2**.

Once this has been accomplished, each process will perform the same job, with the same amount of load in the same sequences, balanced between them, but on different offsets:

There are **size** processes, each one will take care of 1/size of the possible offsets to the current Sequence. Summing their work will end up with the entire combinations of offsets.

Each process will get to compute on each sequence making it more balanced if the sequences vary in length.

Given a current minor sequence with length **L1**, for any or its **offsets,** there exist L mutations, indexed from 1 to **L1**. The processes will send each of such combination to compute an alignment string inside CUDA.

**Usage of CUDA in the project**

The CUDA memory is allocated early in the algorithm because we only need offsets and **mutation index** in order to explain to CUDA what to compute for every **minor sequence** in the input. Therefore, before calculating combinations, each process will allocate **seq1 pointer**, **seq2** **pointer** and a soon to be filled empty **signs** String **pointer** – and that is all the structures that CUDA will need.

Given such a combination was sent by an MPI process, and CUDA knows the main sequence and the minor sequence, it can now start its kernel on the relevant mutation and offset of the minor sequence and compute the alignment string into **signs**.

Each thread in CUDA will compute on a single pair of characters: one from **seq1** and one from **seq2** + mutation + **offset**.

To make things simple, using all threads, each CUDA calculation is made to create a alignment string between one combination of **offset** and mutation and the **main sequence** **seq1**.

**Usage of OMP in the project**

1. Before the offset possibilities are distributed among processes mutation indexes are distributed among the threads using omp for each process.

2. After CUDA has finished creating a **alignment** **string** on a given pair of **seq1** and **seq2**, OMP is then used to compute the **alignment** **score** of that string. Many for loops are used in these calculations and therefore, OMP is a good tool to load the balance of the loops using the available threads.

After Computing each **alignment** **score,** we compare the **optimal score** with the **current score** and find the **maximal score** with its corresponding **mutation index** and **offset**.

The fact that each process holds only a part of the **offsets**, they will all send their best combination and corresponding score to process number 0 and it will receive the information and decide who between the **size** results is the **optimal one** between **all offsets**.

**Complexity Analysis**

Some definitions:

sequenceCount = C

length of the main Sequence = L1

length of any minor Sequence = L2

possible offsets = L1 – L2

possible mutations = L2

number of processes = S

number of CUDA threads = T1

number of OMP threads = T2

signs – calculating the alignment string is O(L2)

scoring – calculating the score on a alignment string is also O(L2)

My serial solution time complexity (big 'O' annotation):

O (sequenceCount \* offsets \* mutations \* (signs + scoring) = O ( C \* (L1 – L2) \* L2 \* 2L2)

MPI – deals with offset distribution evenly among processes.

CUDA – deals with creating each alignment string.

OMP – deals with calculating the score and distributing mutations to inside processes.

My parallel solution time complexity (big 'O' annotation):

O (sequenceCount \* (offsets/number of processes) \* mutations/OMP threads \*

\* (L2/CUDA threads + L2/OMP threads)) =

= O ((C \* (L2 / T2) \* (L1 – L2)/(S) \* [(L1)/(T1) + L1/T2])