# Assignment 2 Report

Subject: ECE 284 - Parallel Computing in Bioinformatics

Submitted by: Ashwin Agesh Somanathan (A59025915)

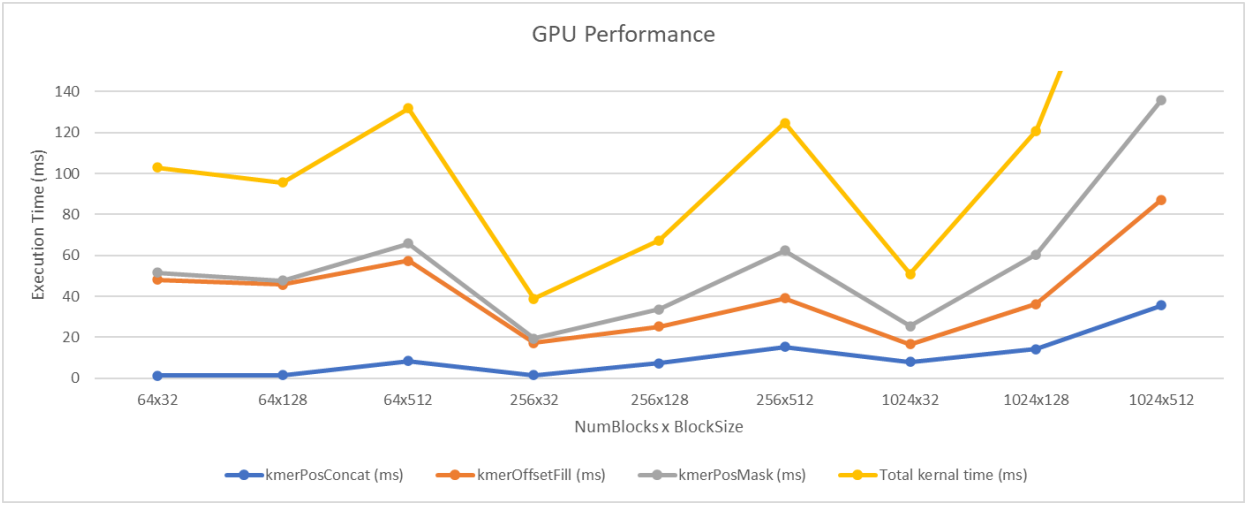
## Parallelizing seed table construction

Q 5)

### Part a)

KmerSize = 12

| NumBlocks | BlockSize | NumBlocks x BlockSize | kmerPosConcat (ms) | kmerOffsetFill (ms) | kmerPosMask (ms) | Total kernal time (ms) |
|---|---|---|---|---|---|---|
| 64 | 32 | 64x32 | 1.3501 | 46.714 | 3.4651 | 51.5292 |
| 64 | 128 | 64x128 | 1.4926 | 44.195 | 2.0878 | 47.7754 |
| 64 | 512 | 64x512 | 8.5314 | 48.834 | 8.5904 | 65.9558 |
| 256 | 32 | 256x32 | 1.4917 | 15.807 | 2.1129 | 19.4116 |
| 256 | 128 | 256x128 | 7.4324 | 17.952 | 8.2742 | 33.6586 |
| 256 | 512 | 256x512 | 15.284 | 23.861 | 23.256 | 62.401 |
| 1024 | 32 | 1024x32 | 7.933 | 8.727 | 8.7876 | 25.4476 |
| 1024 | 128 | 1024x128 | 14.318 | 22.071 | 24.05 | 60.439 |
| 1024 | 512 | 1024x512 | 35.563 | 51.592 | 48.796 | 135.951 |



**Part b)**

Best overall performance seen for the following configuration:

Numblocks (gridsize) = 256

Block Size = 32

Total time = 19.4116ms

**Part c)**

Relative speedup for the best performing configuration w.r.t. the original sequential case:

| Kernel | kmerPosConcat (us) | kmerOffsetFill (us) | kmerPosMask (us) | Total kernal time (us) |
|---|---|---|---|---|
| Original code (Sequential) | 768.57 | 1661.71 | 1007.75 | 3438.03 |
| Fastest parallel | 1.4917 | 15.807 | 2.1129 | 19.4116 |
| Speedup | 515.23 | 105.12 | 476.95 | 177.11 |

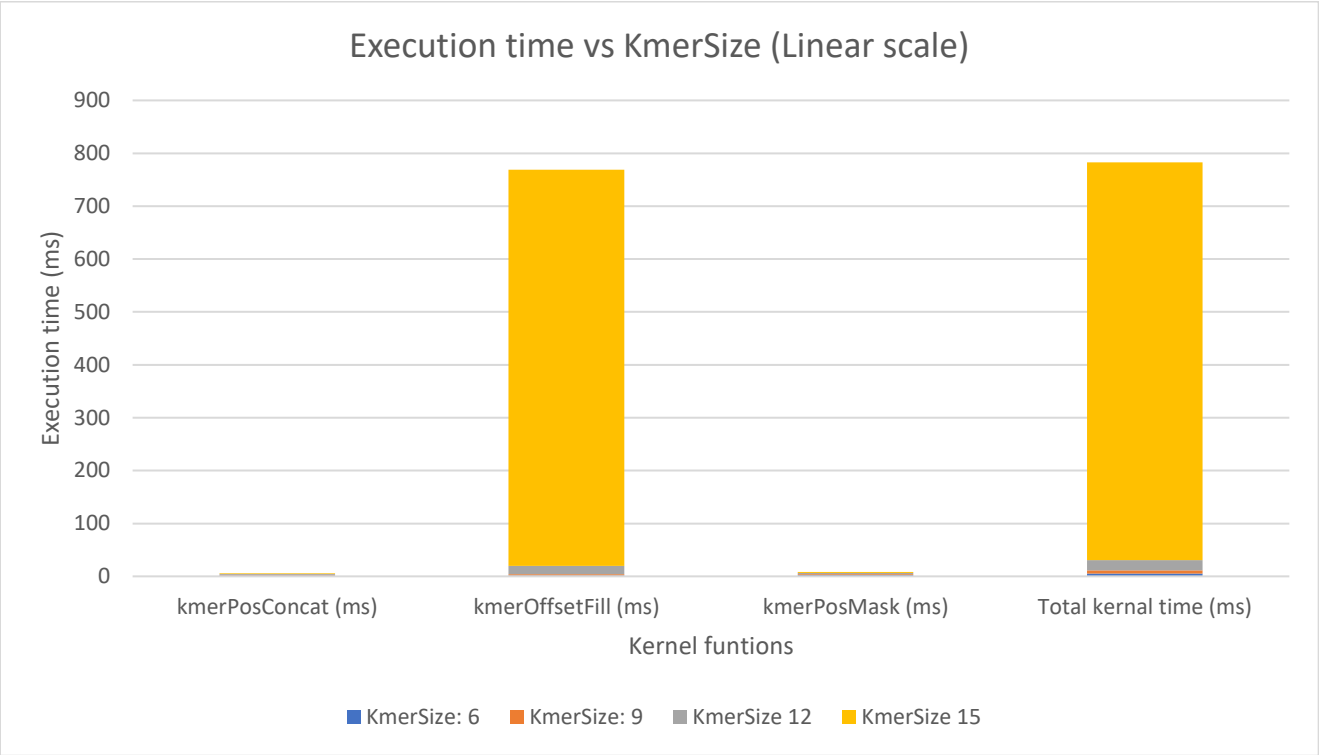
**Part d)**

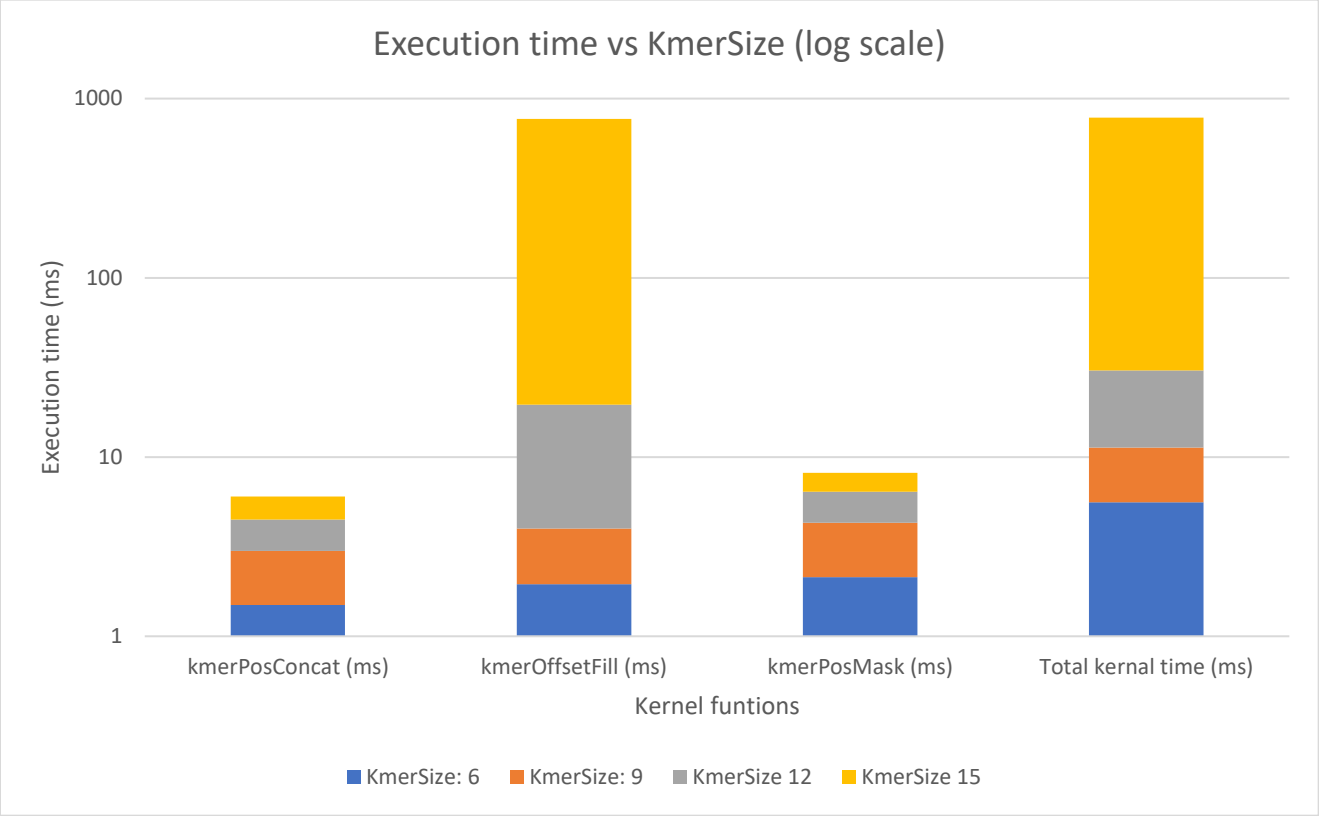
For the configuration 256 x 32, GPU overheads:

| Function | Execution time (Avg) (ms) |
|---|---|
| cudaMalloc | 29.822 |
| [CUDA memcpy HtoD] | 0.62290 |
| [CUDA memcpy DtoH] | 0.0017920 |
| cudaMemcpy | 0. 290 |
| cudaFree | 2.2469 |

The average cudaMalloc time is of the same order as the total execution time of the 3 kernels in the fastest parallel implementation. The minimum (0.2ms), however, is small. Since memory allocations are done before calling seedTableOnGpu, the runtime of this function is not impacted (26ms timed by timer object).

**Part e)**


Execution time vs KmerSize (Linear scale)
900
800
700
600
500
400
300
200
100
0
Execution time (ms)
kmerPosConcat (ms)
kmerOffsetFill (ms)
kmerPosMask (ms)
Total kernal time (ms)
Kernel funtions
KmerSize: 6
KmerSize: 9
KmerSize 12
KmerSize 15


Execution time vs KmerSize (log scale)
1000
100
10
1
Execution time (ms)
kmerPosConcat (ms)
kmerOffsetFill (ms)
kmerPosMask (ms)
Total kernal time (ms)
Kernel funtions
KmerSize: 6
KmerSize: 9
KmerSize 12
KmerSize 15

**Bonus Questions**

A)

The number of entries in the kmerOffset table is a function of kmerSize related as:

$$Number\ of\ entries = 4^{kmerSize}$$

The base 4 is a consequence of there being 4 bases: A, C, G, T. These can appear in any permutation.

It is possible to find occurrences larger than kmerSize for a table.

This can be done by breaking the larger kmer into smaller parts and finding the kmer entry that starts or ends with the sub-part. Then for the two kmer entries, find the entries in the position table where they sit in consecutive positions, signifying they are place one after the other.

Of course, the order of the sub-parts needs to be taken into account.

For example:

If we need to find a kmer AGTC in a table of kmerSize 2, we can break the search item into 2:

AG and TC

Then look in the position table where all they occur. If there are cases where TC is right after AG, then AGTC has been found.

B)

For large kmer sizes, the method mentioned in part (A) can be employed. Build a table for a smaller kmerSize, say 10 and break the actual kmer into parts.