System Specifications

Operating System: x86_64

CPU: Intel ® Xeon ® CPU E5-2620

GPU: Tesla K20c

Note: I used the given server for implementation and testing (especially for the parallel part of the code).

Implementation Details

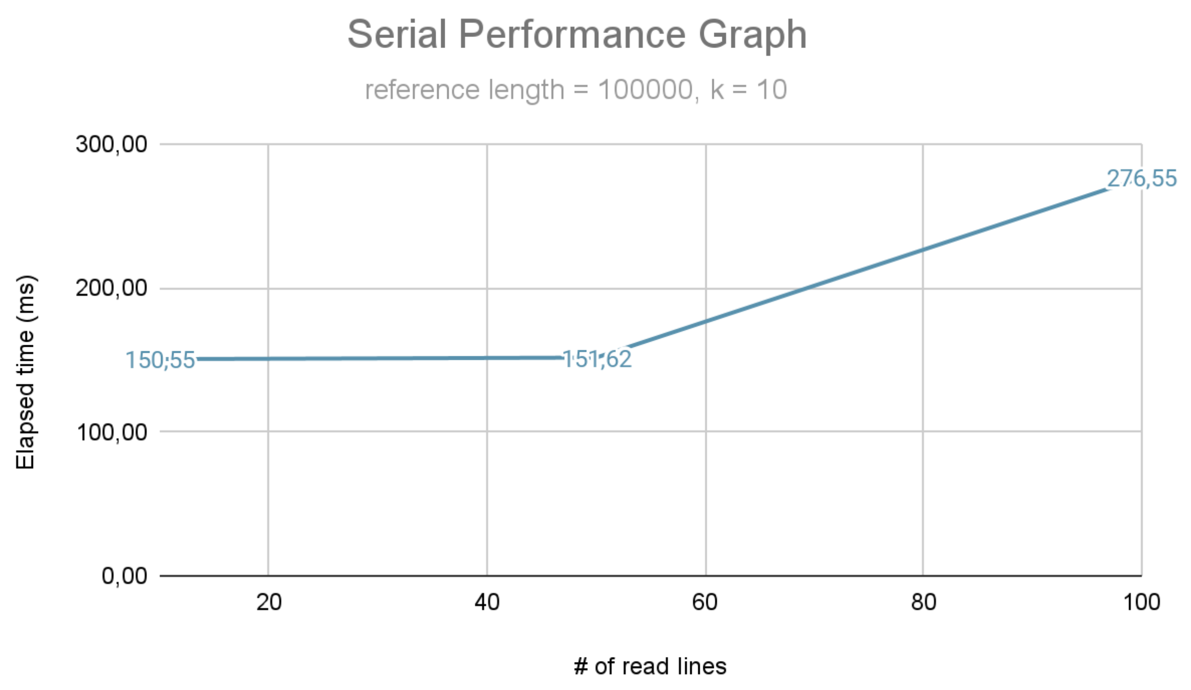
To implement the serial part of the project, I used the “unordered_map” library as a hash map. Firstly, I put each k-mer value from the reference string as a key to the hash map and for each key, there exists a hash object that holds both index values and hit counts so that I could hold more than one occurrence for each k-mer in the reference string. Then for each read line, I took their k-mer strings and search them in the map then I created a result array from them. Because I used the “unordered_map” library, the C++14 version has to be downloaded.

Note: In this implementation, I used my own device with C++14 downloaded. However, this serial code cannot be compiled in the server given.

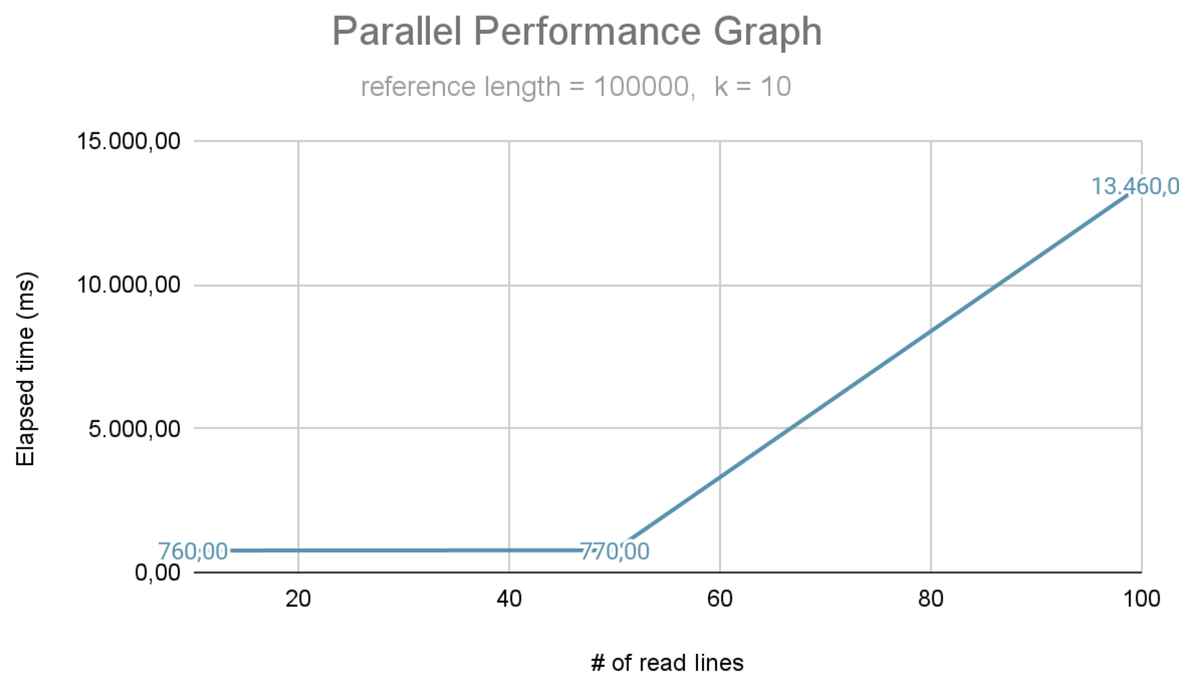
To implement the parallel part of the project, I copied the reference string from host to device. Also, I created one complete string for read lines by attaching them and I coped this string to the device memory, too. Lastly, I created a resulting array in the device memory. For each k-mer on the complete read line string, I created a thread in the CUDA device and by using their block and thread IDs, I calculated their first occurrence index and put the index values their respecting index in the result array. To handle the limitation that for one block there exists a maximum of 1024 threads, I calculated the number of blocks needed in the run time and create that number of blocks in the device with full thread capacity. Because I created unused threads to handle this problem, there may be some performance overheads while creating threads, however, because CUDA threads are very lightweight, I think that could be ignored. To compare the k-mer strings, I used the given __device__ function. Finally, I copied the resulting array from the device to the host.

Note: This code is implemented and tested in the server given.

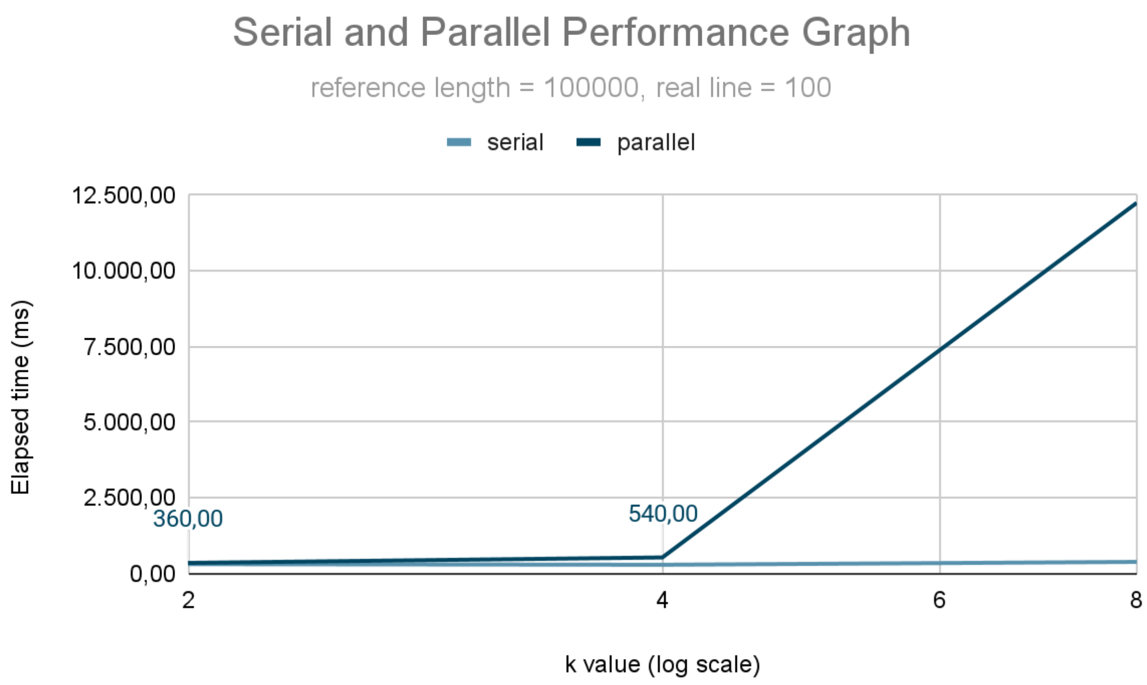
Analysis



Serial Performance Graph
reference length = 100000, k = 10
300,00
200,00
100,00
0,00
Elapsed time (ms)
150,55
151,62
276,55
20
40
60
80
100
# of read lines



Parallel Performance Graph
reference length = 100000, k = 10
15.000,00
10.000,00
5.000,00
0,00
Elapsed time (ms)
760,00
770,00
13.460,0
20
40
60
80
100
# of read lines



As it can be seen from the figure, because in the serial implementation hash map was used, the performance of the serial code always overtakes the parallel code. To create one complete read line array in the parallel implementation could be costly. Also, for each k-mer whole reference string is passed. Therefore, there exist some overheads because there are lots of threads and some complex implementations.

Note: Because I used the given server, I could not use the “nvprof”. Therefore, in this report, the nvprof output section does not exist.