Parallelization of Programs

Assignment 2 - Report
Suraj Gaikwad
12241830

March 24, 2025

1 Explaination of Logic Used

1.1 Sequential

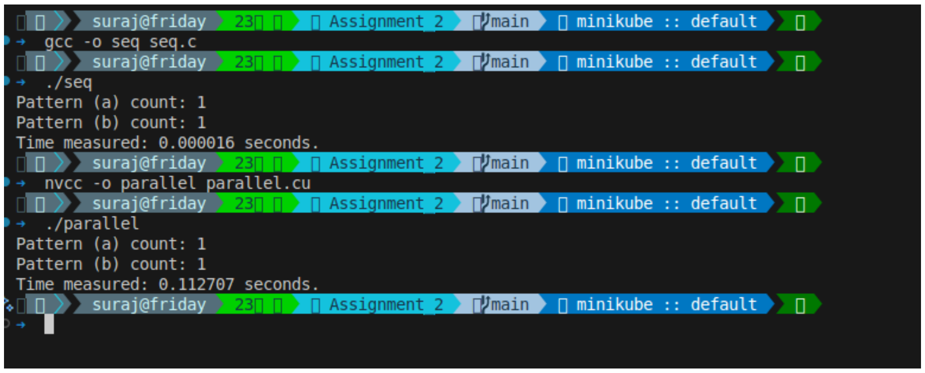
I am traversing each node (node1 in code). Checking its every adjacent node(node2). Now by observing every adjacent node of node2 (say node3), if node1 is equal to node3 we should skip this iterration. Else I check for whether node3 is directly connected to to node1. If it is then node1, node2, node3 forms a triangle(pattern b), else it forms a line(pattern a) Then i Update the shared variables accordingly.

1.2 Parallel

Instead of traversing all nodes for node1 we can use cuda for parallel processing. The logic is same as sequential but we are parallezing the first loop.

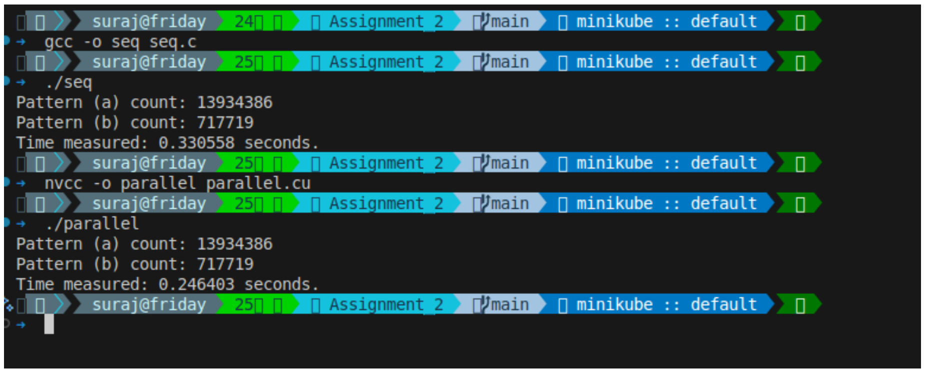
2 Speedup Comparision

We can see in the following screenshots that the parrallel version of the code are much faster on the bigger graph(data.tsv) as compared to smaller graph(small_graph.tsv). The speedup has been increased from 0.00014 to 1.375.



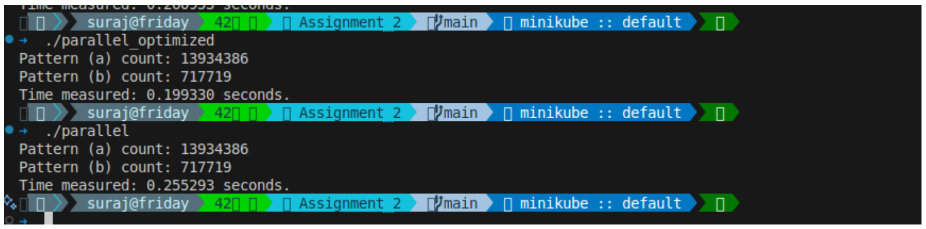
```
suraj@friday  23   Assignment 2  main   minikube :: default
→ gcc -o seq seq.c
suraj@friday  23   Assignment 2  main   minikube :: default
→ ./seq
Pattern (a) count: 1
Pattern (b) count: 1
Time measured: 0.000016 seconds.
suraj@friday  23   Assignment 2  main   minikube :: default
→ nvcc -o parallel parallel.cu
suraj@friday  23   Assignment 2  main   minikube :: default
→ ./parallel
Pattern (a) count: 1
Pattern (b) count: 1
Time measured: 0.112707 seconds.
suraj@friday  23   Assignment 2  main   minikube :: default
→
```

Figure 1: Result on Small Graphs



```
suraj@friday  24   Assignment 2  main   minikube :: default
→ gcc -o seq seq.c
suraj@friday  25   Assignment 2  main   minikube :: default
→ ./seq
Pattern (a) count: 13934386
Pattern (b) count: 717719
Time measured: 0.330558 seconds.
suraj@friday  25   Assignment 2  main   minikube :: default
→ nvcc -o parallel parallel.cu
suraj@friday  25   Assignment 2  main   minikube :: default
→ ./parallel
Pattern (a) count: 13934386
Pattern (b) count: 717719
Time measured: 0.246403 seconds.
suraj@friday  25   Assignment 2  main   minikube :: default
→
```

Figure 2: Result on Bigger Graphs



```
  suraj@friday  42  Assignment 2  main  minikube :: default
→  ./parallel_optimized
Pattern (a) count: 13934386
Pattern (b) count: 717719
Time measured: 0.199330 seconds.
  suraj@friday  42  Assignment 2  main  minikube :: default
→  ./parallel
Pattern (a) count: 13934386
Pattern (b) count: 717719
Time measured: 0.255293 seconds.
  suraj@friday  42  Assignment 2  main  minikube :: default
→
```

Figure 3: Results when Optimization Applied

3 Optimizations used

I have used two optimizations technique to speed up the performance.

1. Optimizing Memory Transfer – Pinned Memory: As discussed in the class I have used the Pinned memory for faster data transfer/

2. I have Also used Overlapping Data Transfer and Execution in Streams for faster executions. First i have divided whole data into streams. I have tried with multiple streams and got best results when i used 3 streams.

You can refer to the above screenshot.

These both optimization combined reduced the speed of execution from 0.255 seconds to 0.199 seconds which makes our **parallel code 1.28 times faster**.