CS 378: Programming For Performance Project 3

Sai Avala
Vaibhav Gupta
Sudheesh Katkam

November 10, 2014

1 Introduction

For this project we parallelized Dijkstra's Single Source Shortest Path (SSSP) algorithm and tested this function against the full US Road Map Travel Graph. We implemented four separate schedulers to show the variations in performance when applied to the parallelized Dijkstra's algorithm. These four schedulers are as follows:

- Global LIFO
- Global FIFO
- Global Priority Queue
- Priority queue per thread, work is pushed locally, threads steal when out of work

2 Dijkstras Algorithm

Dijkstras algorithm is a graph search algorithm solves for the shortest path from a single source given non-negative edge weights. Essentially what this algorithm does is that given a starting node, it will go ahead, pick the rest of the neighbors, calculates the difference between the current node and the neighboring nodes and updates the distances for each neighboring node if it is smaller.

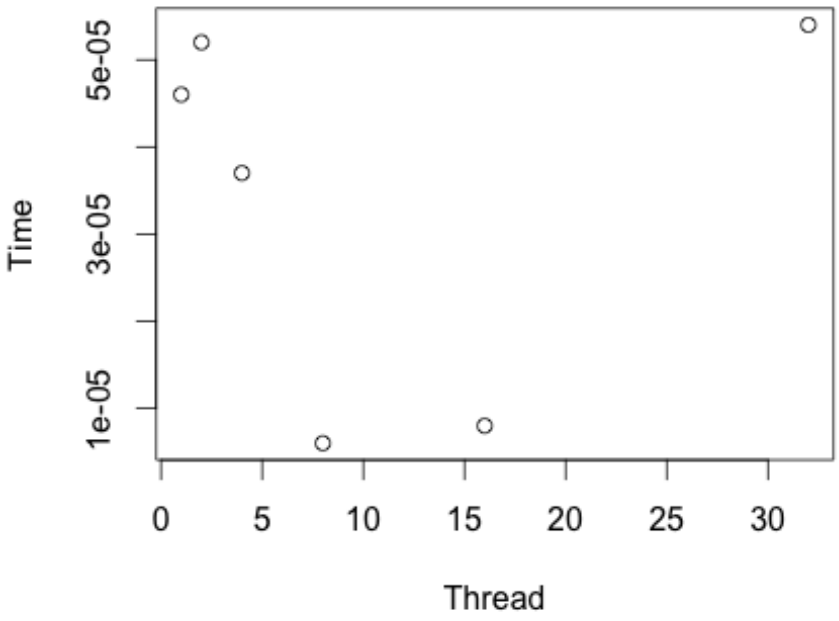
3 Results

This section here displays both the results and plots for the different schedulers. By looking at the graphs on the subsequen pages, we notice that our FIFO Queue scheudler implementation works the best. There are points in the graph where the data might seem "random". However, this could possibly be explained by the fact that the total run time for the whole data set is not that long. Then, when generating threads to work with the small runtime, we would run into seemingly random results because the generation of threads and joining of the threads could change the actual time. What we noticed here is that unless the working set size is somewhat large, generating threads would seem to provide soemwhat unusual results. Also, please note that we ran this project on the NYC Travel Time graph, not the Full USA graph. This is because runtimes for LIFO would have been over 7+ hours, and there wasn't enough Service Units allocation available on Stampede to actually run the program.
The raw data results are below:

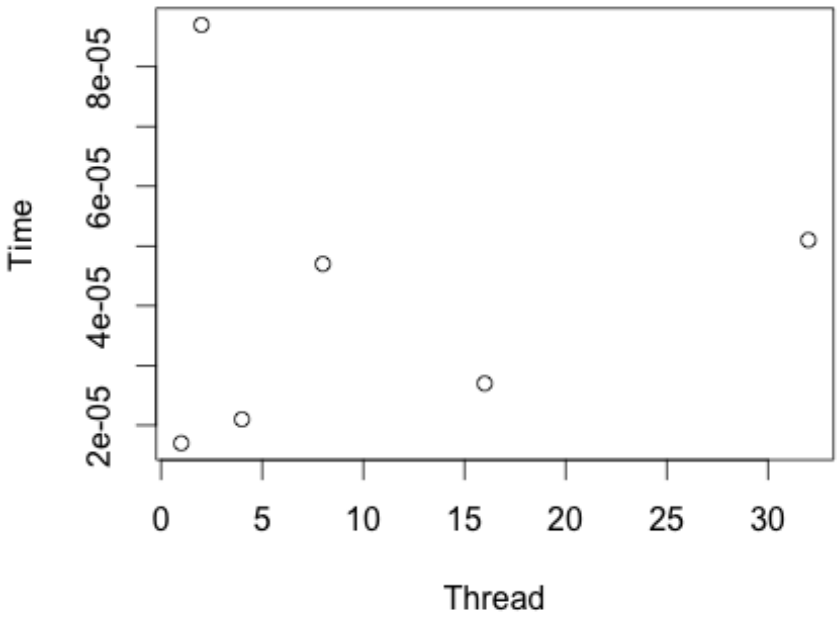
```
    32 Threads
./project3 32 nyc-raod.gr 733846
----Queue----
Time:   0.000054
----Stack----
Time:   0.000051
----Priority Queue----
Time:   0.000056
----Shared Priority Queue----
Time:   0.000050

    16 Threads
./project3 16 nyc-raod.gr 366923
----Queue----
Time:   0.000008
----Stack----
Time:   0.000027
----Priority Queue----
Time:   0.000012
----Shared Priority Queue----
Time:   0.000016

    8 Threads
./project3 8 nyc-raod.gr 183462
----Queue----
Time:   0.000006
----Stack----
Time:   0.000047
----Priority Queue----
```

```
Time:   0.000062
----Shared Priority Queue----
Time:   0.000064

    4 Threads
./project3 4 nyc-raod.gr 91731
----Queue----
Time:   0.000037
----Stack----
Time:   0.000021
----Priority Queue----
Time:   0.000052
----Shared Priority Queue----
Time:   0.000112

    2 Threads
./project3 2 nyc-raod.gr 45866
----Queue----
Time:   0.000052
----Stack----
Time:   0.000087
----Priority Queue----
Time:   0.000041
----Shared Priority Queue----
Time:   0.000050

    1 Thread
./project3 1 nyc-raod.gr 22933
----Queue----
Time:   0.000046
----Stack----
Time:   0.000017
----Priority Queue----
Time:   0.000056
----Shared Priority Queue----
Time:   0.000050
```

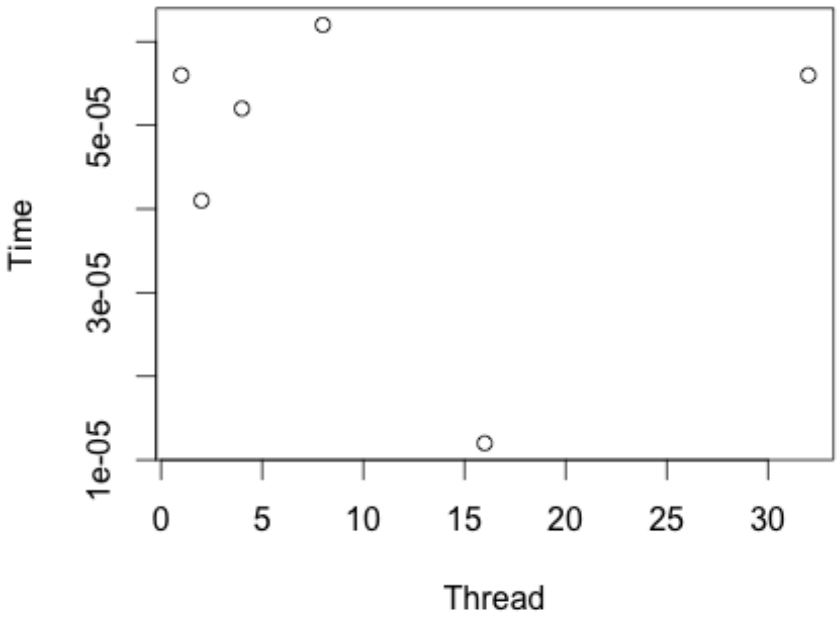
3.1 Queue Scheduler Results



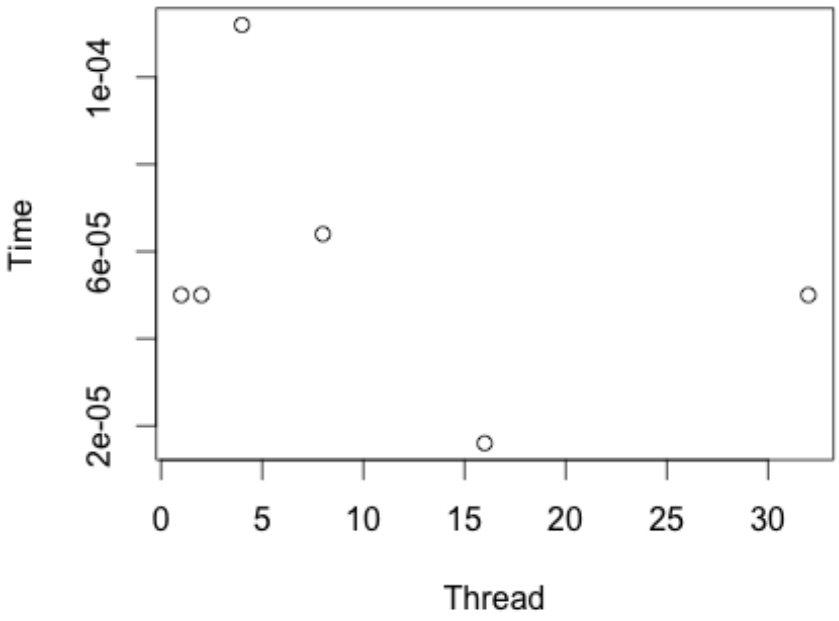
3.2 Stack Scheduler Results



3.3 Global Priority Queue Scheduler Results



3.4 Shared Priority Queue Scheduler Results



4 Stuff

haha