CS426
Project 2
Muhammad Arham Khan
21701848

Implementation details:

Part A:

I used the provided algorithm to calculate the shortest distances between the graph nodes. In the first part, I parsed the file for graph data. Then I utilized the provided vertices and edges list to traverse through multiple frontiers of the graph sections and do a breadth-first search. Since this algorithm relied heavily on repetitive loops through the graph, the runtime complexity was O(n^2logn) and the runtimes for multiple graphs were as follows:

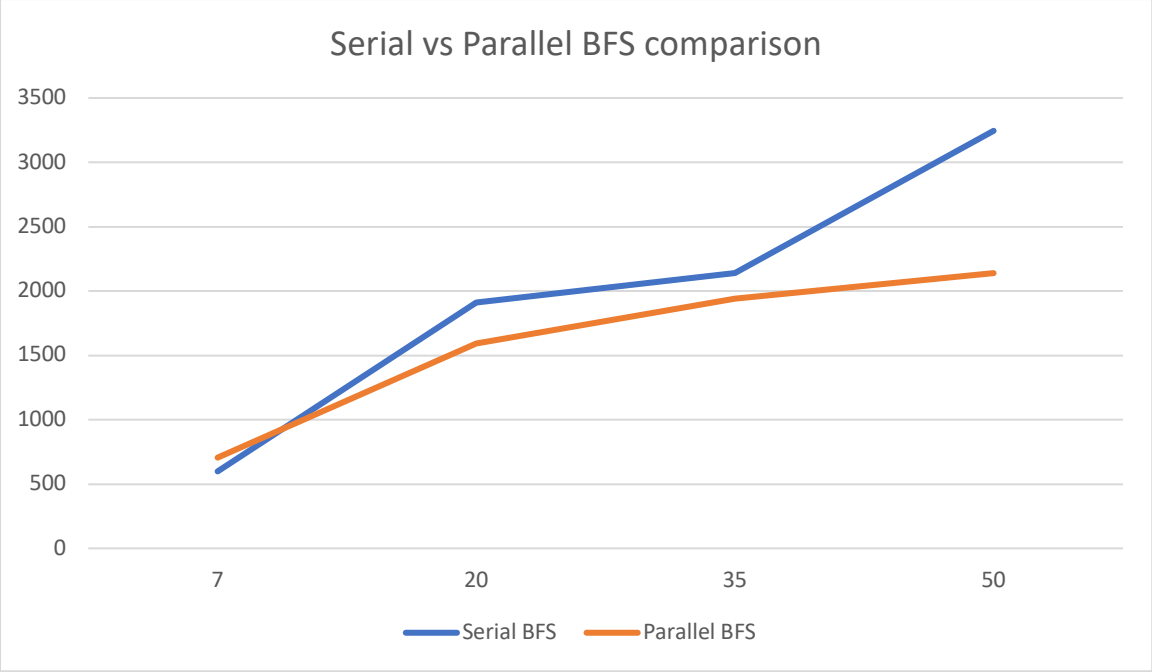
Vertices	**Time**
7	598 ms
20	1911 ms
35	2140 ms
50	3245 ms

Part B:

Since this implementation was meant to be parallel, I read the file in the master process (rank = 0) and used MPI_Barrier to hold all other processes till the files were read. I stored the main frontier and nextfrontier globally so that it was accessible across processes. So were the values of m, n and rows and vertice values. In the master process, I divided the rows and columns into subsections based on equal 1-D partition and then sent it to respective processes using MPI_SEND and MPI_RECEIVE. Once it had been received, each process processed their own set values and stored the distance values using an algorithm same as the one used in Part A and a global level. This utilized MPI_AlltoAll to process their out-neighbours from other partitions in a loop. The level-concurrence here was also achieved by running MPI_ Barrier to hold all processes on the same level avoid any race conditions. I tested this algorithm with 4 processes and different number of vertice graphs. The outputs are as follows

Vertices	**Time**
7	705 ms
20	1591 ms
35	1938 ms
50	2139 ms

Plot comparison:



Conclusion:

In my opinion the performance of the BFS algorithm is better in serial for smaller number of vertices as when processed in parallel, the bottleneck of level-concurrency arises and since each process is handling a lower number of vertices, there is an added delay of out-neighbour all-to-all communication. But for the higher number of vertices, the performance certainly increases as the processing load for each level is divided equally. Hence, I assume the performance of this algorithm is better suited for more vertices in a graph. To further improve the performance of this parallel algorithm, we could utilize one-to-one communication like MPI_SEND and MPI_RECEIVE to selective processes at the beginning and reduce the all-to-all communication delay.