Peri Hassanzadeh
ECE1570 High Performance Computing
Dr. Bigrigg
December 3, 2021

Tightly and Loosely Coupled Systems

Introduction

In this assignment, I was able to use threads to develop data parallel algorithms as well as clusters to gain a better understanding maximizing the usage of a system to achieve as much speedup as possible utilizing multiple configurations of processors. I identified the speedup I could achieve by using multiple processors on the CRC server.

The main objective of each program no matter it's algorithm was to use Dijkstra's algorithm to find the shortest path from the source node (0,0) to each other node in the 100 by 100 matrix. I was then able to split the matrix into four smaller matrices of 50 by 50. After calculating the shortest path with Dijkstra's algorithm, I output the results of each path into a text file for the data parallel and MPI versions.The timing of each version of the project was calculated and run five times each to get an average result which will be further analyzed.

In order to run each file, make sure that all .cpp files are in the same folder as the working directory.Windows refers to my local environment and Linux refers to the server environment and the output files will be in the same directory after the exe is run. This format will allow for successful compilation of each implementation resulting in an executable file with the following commands:

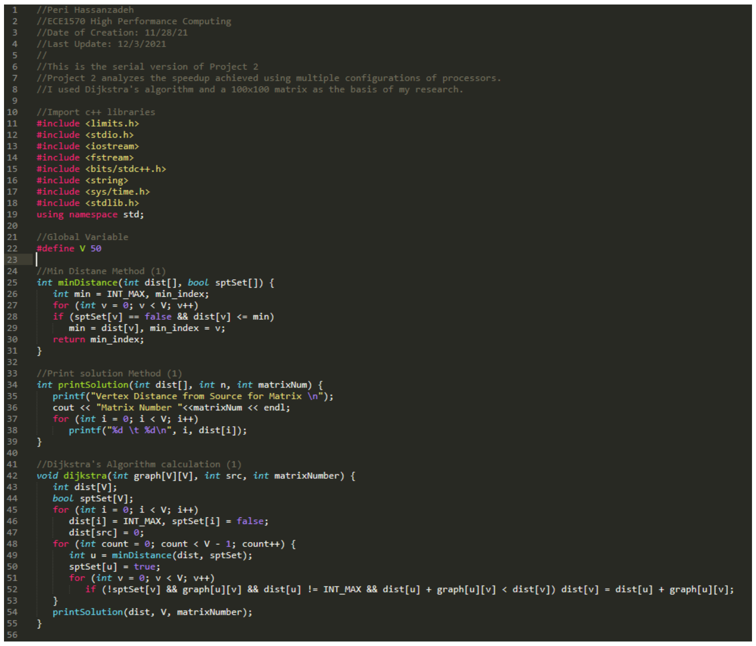
Serial command:
```
g++ P2_serial.cpp -o P2_serial
```
Data Parallel command:
```
Windows: g++ P2_thread.cpp -o P2_thread -lpthread
Linux: sbatch P2_thread.script
```
MPI command:
```
Linux: sbatch P2_mpi.script
```

Finally, note that each implementation could be optimized in various ways. I will go over a few points that I wish I would have been able to develop on given better time management.

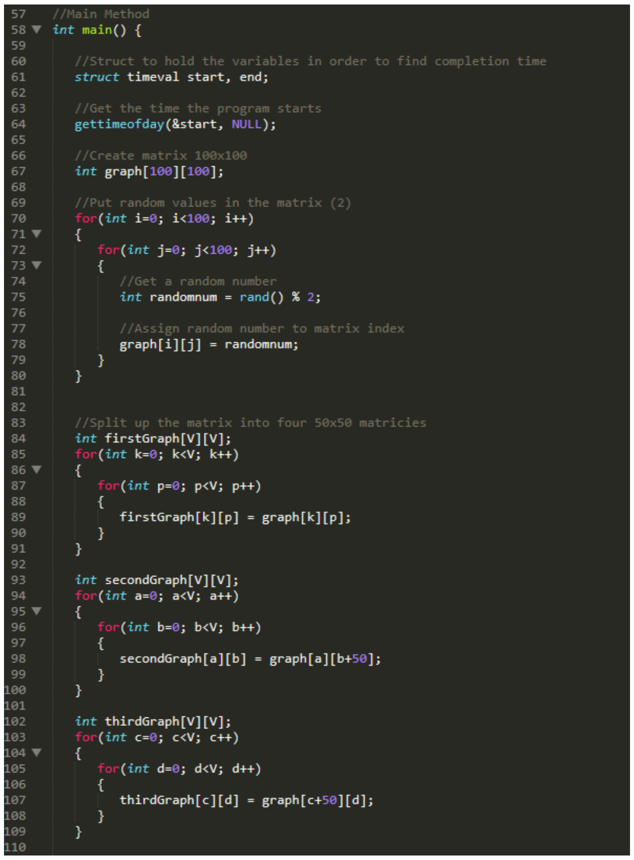
Serial Version

The files that correspond with the serial version are as follows:
P2_serial.cpp - implementation of serial version
P2_serial.exe - compiled program file
P2_serial.script - Used to run on CRC
Serial.out - Results from CRC script

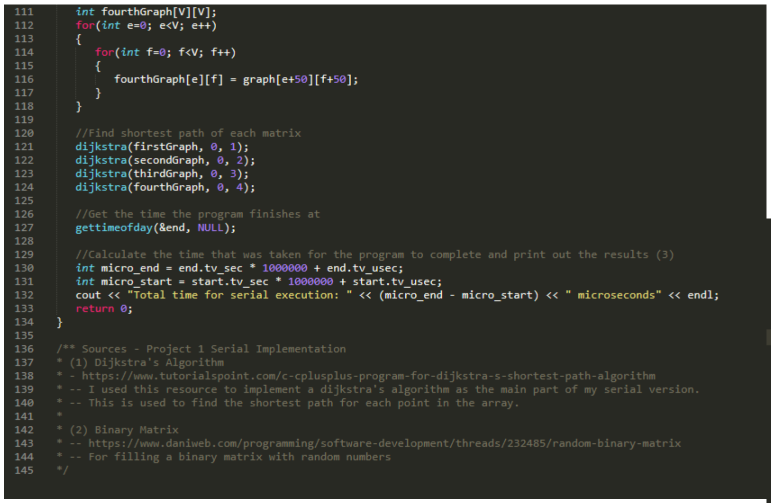
Screenshots of Code:



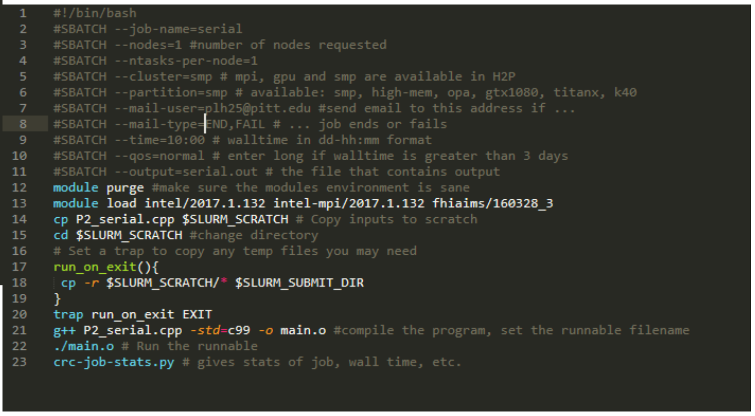
```cpp
//Peri Hassanzadeh
//ECE1570 High Performance Computing
//Date of Creation: 11/28/21
//Last Update: 12/3/2021
//
//This is the serial version of Project 2
//Project 2 analyzes the speedup achieved using multiple configurations of processors.
//I used Dijkstra's algorithm and a 100x100 matrix as the basis of my research.

//Import c++ libraries
#include <limits.h>
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <bits/stdc++.h>
#include <string>
#include <sys/time.h>
#include <stdlib.h>
using namespace std;

//Global Variable
#define V 50

//Min Distane Method (1)
int minDistance(int dist[], bool sptSet[]) {
   int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++)
   if (sptSet[v] == false && dist[v] <= min)
      min = dist[v], min_index = v;
   return min_index;
}

//Print solution Method (1)
int printSolution(int dist[], int n, int matrixNum) {
   printf("Vertex Distance from Source for Matrix \n");
   cout << "Matrix Number "<<matrixNum << endl;
   for (int i = 0; i < V; i++)
      printf("%d \t %d\n", i, dist[i]);
}

//Dijkstra's Algorithm calculation (1)
void dijkstra(int graph[V][V], int src, int matrixNumber) {
   int dist[V];
   bool sptSet[V];
   for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;
      dist[src] = 0;
   for (int count = 0; count < V - 1; count++) {
      int u = minDistance(dist, sptSet);
      sptSet[u] = true;
      for (int v = 0; v < V; v++)
         if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v]) dist[v] = dist[u] + graph[u][v];
   }
   printSolution(dist, V, matrixNumber);
}

```



```cpp
//Main Method
int main() {

   //Struct to hold the variables in order to find completion time
   struct timeval start, end;

   //Get the time the program starts
   gettimeofday(&start, NULL);

   //Create matrix 100x100
   int graph[100][100];

   //Put random values in the matrix (2)
   for(int i=0; i<100; i++)
   {
      for(int j=0; j<100; j++)
      {
         //Get a random number
         int randomnum = rand() % 2;

         //Assign random number to matrix index
         graph[i][j] = randomnum;
      }
   }


   //Split up the matrix into four 50x50 matricies
   int firstGraph[V][V];
   for(int k=0; k<V; k++)
   {
      for(int p=0; p<V; p++)
      {
         firstGraph[k][p] = graph[k][p];
      }
   }

   int secondGraph[V][V];
   for(int a=0; a<V; a++)
   {
      for(int b=0; b<V; b++)
      {
         secondGraph[a][b] = graph[a][b+50];
      }
   }

   int thirdGraph[V][V];
   for(int c=0; c<V; c++)
   {
      for(int d=0; d<V; d++)
      {
         thirdGraph[c][d] = graph[c+50][d];
      }
   }

```



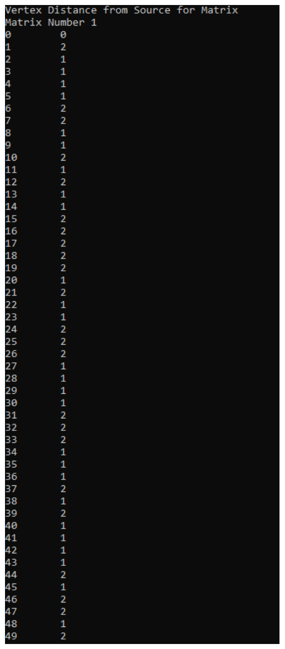
```cpp
   int fourthGraph[V][V];
   for(int e=0; e<V; e++)
   {
      for(int f=0; f<V; f++)
      {
         fourthGraph[e][f] = graph[e+50][f+50];
      }
   }

   //Find shortest path of each matrix
   dijkstra(firstGraph, 0, 1);
   dijkstra(secondGraph, 0, 2);
   dijkstra(thirdGraph, 0, 3);
   dijkstra(fourthGraph, 0, 4);

   //Get the time the program finishes at
   gettimeofday(&end, NULL);

   //Calculate the time that was taken for the program to complete and print out the results (3)
   int micro_end = end.tv_sec * 1000000 + end.tv_usec;
   int micro_start = start.tv_sec * 1000000 + start.tv_usec;
   cout << "Total time for serial execution: " << (micro_end - micro_start) << " microseconds" << endl;
   return 0;
}

/** Sources - Project 1 Serial Implementation
* (1) Dijkstra's Algorithm
* - https://www.tutorialspoint.com/c-cplusplus-program-for-dijkstra-s-shortest-path-algorithm
* -- I used this resource to implement a dijkstra's algorithm as the main part of my serial version.
* -- This is used to find the shortest path for each point in the array.
*
* (2) Binary Matrix
* -- https://www.daniweb.com/programming/software-development/threads/232485/random-binary-matrix
* -- For filling a binary matrix with random numbers
*/
```



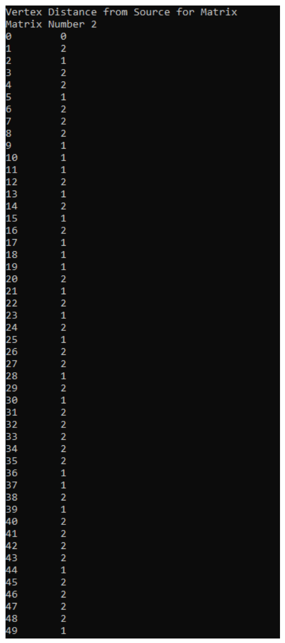
```bash
#!/bin/bash
#SBATCH --job-name=serial
#SBATCH --nodes=1 #number of nodes requested
#SBATCH --ntasks-per-node=1
#SBATCH --cluster=smp # mpi, gpu and smp are available in H2P
#SBATCH --partition=smp # available: smp, high-mem, opa, gtx1080, titanx, k40
#SBATCH --mail-user=plh25@pitt.edu #send email to this address if ...
#SBATCH --mail-type=END,FAIL # ... job ends or fails
#SBATCH --time=10:00 # walltime in dd-hh:mm format
#SBATCH --qos=normal # enter long if walltime is greater than 3 days
#SBATCH --output=serial.out # the file that contains output
module purge #make sure the modules environment is sane
module load intel/2017.1.132 intel-mpi/2017.1.132 fhiaims/160328_3
cp P2_serial.cpp $SLURM_SCRATCH # Copy inputs to scratch
cd $SLURM_SCRATCH #change directory
# Set a trap to copy any temp files you may need
run_on_exit(){
 cp -r $SLURM_SCRATCH/* $SLURM_SUBMIT_DIR
}
trap run_on_exit EXIT
g++ P2_serial.cpp -std=c99 -o main.o #compile the program, set the runnable filename
./main.o # Run the runnable
crc-job-stats.py # gives stats of job, wall time, etc.
```

The screenshots on the previous page are for the serial implementation of the project. The code may also be viewed in the submitted files as P2_serial.cpp and P2_serial.script. Zoom in for a better view of the code or open the specified file.

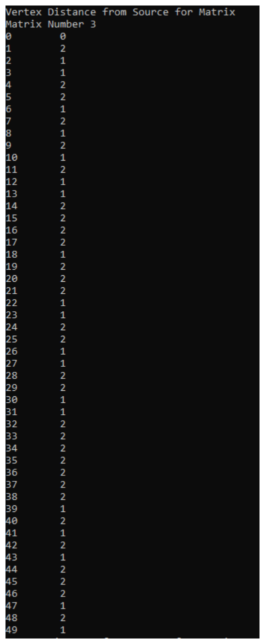
Output:
From CRC Server:



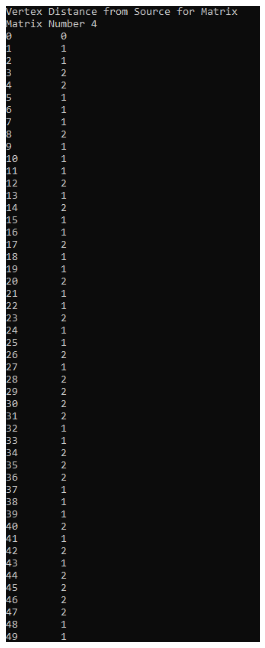
```
Vertex Distance from Source for Matrix
Matrix Number 1
0       0
1       2
2       1
3       1
4       1
5       1
6       2
7       2
8       1
9       1
10      2
11      1
12      2
13      1
14      1
15      2
16      2
17      2
18      2
19      2
20      1
21      2
22      1
23      1
24      2
25      2
26      2
27      1
28      1
29      1
30      1
31      2
32      2
33      2
34      1
35      1
36      1
37      2
38      1
39      2
40      1
41      1
42      1
43      1
44      2
45      1
46      2
47      2
48      1
49      2
```



```
Vertex Distance from Source for Matrix
Matrix Number 2
0       0
1       2
2       1
3       2
4       2
5       1
6       2
7       2
8       2
9       1
10      1
11      1
12      2
13      1
14      2
15      1
16      2
17      1
18      1
19      1
20      2
21      1
22      2
23      1
24      2
25      1
26      2
27      2
28      1
29      2
30      1
31      2
32      2
33      2
34      2
35      2
36      1
37      1
38      2
39      1
40      2
41      2
42      2
43      2
44      1
45      2
46      2
47      2
48      2
49      1
```

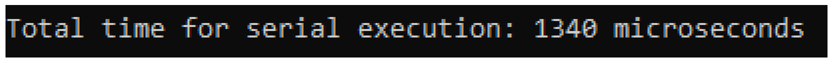


```
Vertex Distance from Source for Matrix
Matrix Number 3
0       0
1       2
2       1
3       1
4       2
5       2
6       1
7       2
8       1
9       2
10      1
11      2
12      1
13      1
14      2
15      2
16      2
17      2
18      1
19      2
20      2
21      2
22      1
23      1
24      2
25      2
26      1
27      1
28      2
29      2
30      1
31      1
32      2
33      2
34      2
35      2
36      2
37      2
38      2
39      1
40      2
41      1
42      2
43      1
44      2
45      2
46      2
47      1
48      2
49      1
```

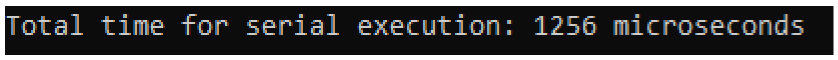


```
Vertex Distance from Source for Matrix
Matrix Number 4
0       0
1       1
2       1
3       2
4       2
5       1
6       1
7       1
8       2
9       1
10      1
11      1
12      2
13      1
14      2
15      1
16      1
17      2
18      1
19      1
20      2
21      1
22      1
23      2
24      1
25      1
26      2
27      1
28      2
29      2
30      2
31      2
32      1
33      1
34      2
35      2
36      2
37      1
38      1
39      1
40      2
41      1
42      2
43      1
44      2
45      2
46      2
47      2
48      1
49      1
```

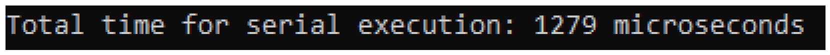
Execution time:



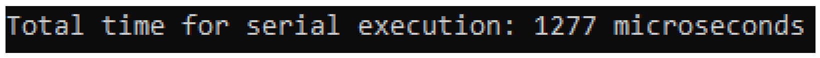
```
Total time for serial execution: 1340 microseconds
```



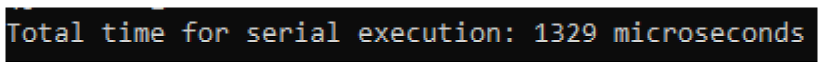
```
Total time for serial execution: 1256 microseconds
```



```
Total time for serial execution: 1279 microseconds
```



```
Total time for serial execution: 1277 microseconds
```



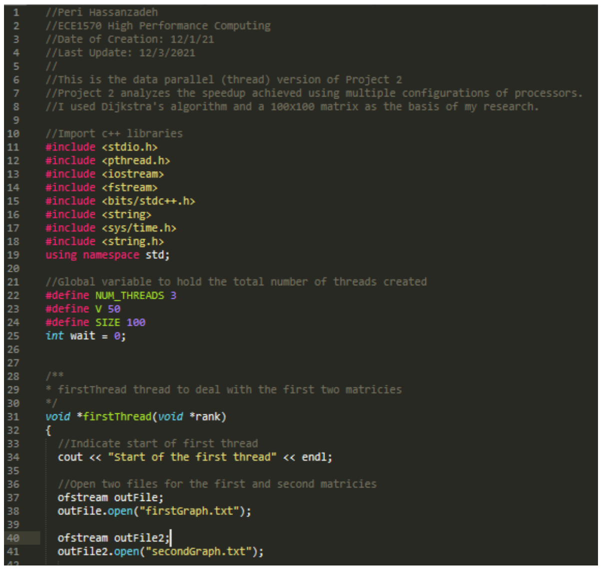
```
Total time for serial execution: 1329 microseconds
```

From the CRC, the average execution took 1296 microseconds which is significantly less than that of my local machine which had an average of 1600 microseconds. Additionally, I will make sure to analyze the rest of the execution times for data parallel and clusters from the server as well as my local machine. Comparing the two values will allow me to properly decide proper scaling and different implementation benefits and drawbacks.

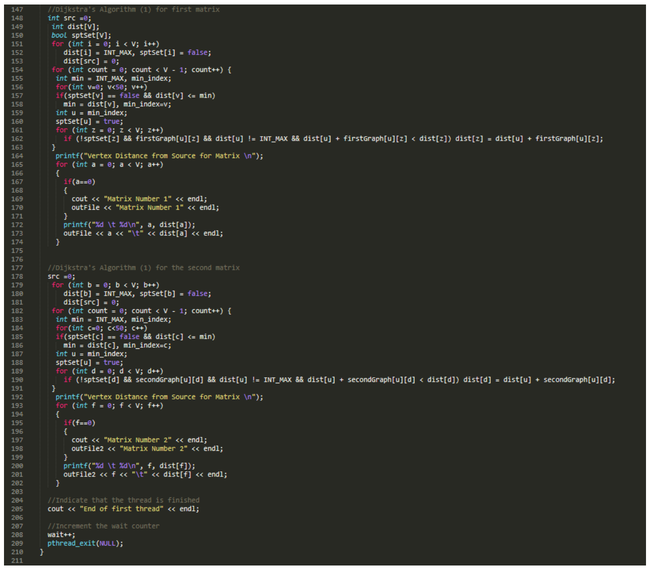
Data Parallel (Thread) Version

The files that correspond with the data parallel version are as follows:
P2_thread.cpp - implementation of data parallel version
P2_thread.exe - compiled program file
P2_thread.script - to compile on CRC
Thread.out - Result from running on CRC script
First through fourthGraph.txt - Text files containing results

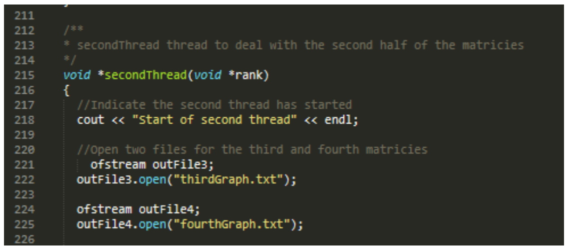
Screenshots of Code:



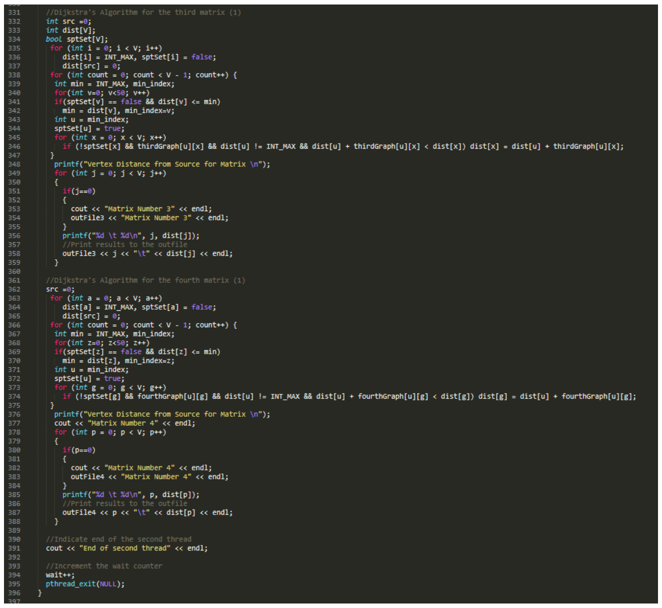
```cpp
//Peri Hassanzadeh
//ECE1570 High Performance Computing
//Date of Creation: 12/1/21
//Last Update: 12/3/2021
//
//This is the data parallel (thread) version of Project 2
//Project 2 analyzes the speedup achieved using multiple configurations of processors.
//I used Dijkstra's algorithm and a 100x100 matrix as the basis of my research.

//Import c++ libraries
#include <stdio.h>
#include <pthread.h>
#include <iostream>
#include <fstream>
#include <bits/stdc++.h>
#include <string>
#include <sys/time.h>
#include <string.h>
using namespace std;

//Global variable to hold the total number of threads created
#define NUM_THREADS 3
#define V 50
#define SIZE 100
int wait = 0;


/**
* firstThread thread to deal with the first two matricies
*/
void *firstThread(void *rank)
{
  //Indicate start of first thread
  cout << "Start of the first thread" << endl;

  //Open two files for the first and second matricies
  ofstream outFile;
  outFile.open("firstGraph.txt");

  ofstream outFile2;
  outFile2.open("secondGraph.txt");
```



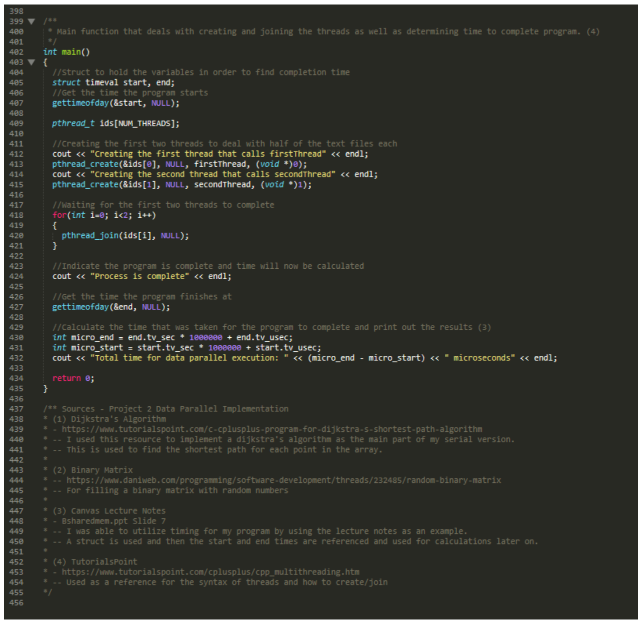
```cpp
  //Dijkstra's Algorithm (1) for first matrix
  int src =0;
   int dist[V];
   bool sptSet[V];
   for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;
      dist[src] = 0;
   for (int count = 0; count < V - 1; count++) {
    int min = INT_MAX, min_index;
    for(int v=0; v<50; v++)
    if(sptSet[v] == false && dist[v] <= min)
      min = dist[v], min_index=v;
    int u = min_index;
    sptSet[u] = true;
    for (int z = 0; z < V; z++)
      if (!sptSet[z] && firstGraph[u][z] && dist[u] != INT_MAX && dist[u] + firstGraph[u][z] < dist[z]) dist[z] = dist[u] + firstGraph[u][z];
   }
    printf("Vertex Distance from Source for Matrix \n");
    for (int a = 0; a < V; a++)
    {
      if(a==0)
      {
        cout << "Matrix Number 1" << endl;
        outFile << "Matrix Number 1" << endl;
      }
      printf("%d \t %d\n", a, dist[a]);
      outFile << a << "\t" << dist[a] << endl;
    }


  //Dijkstra's Algorithm (1) for the second matrix
  src =0;
   for (int b = 0; b < V; b++)
      dist[b] = INT_MAX, sptSet[b] = false;
      dist[src] = 0;
   for (int count = 0; count < V - 1; count++) {
    int min = INT_MAX, min_index;
    for(int c=0; c<50; c++)
    if(sptSet[c] == false && dist[c] <= min)
      min = dist[c], min_index=c;
    int u = min_index;
    sptSet[u] = true;
    for (int d = 0; d < V; d++)
      if (!sptSet[d] && secondGraph[u][d] && dist[u] != INT_MAX && dist[u] + secondGraph[u][d] < dist[d]) dist[d] = dist[u] + secondGraph[u][d];
   }
    printf("Vertex Distance from Source for Matrix \n");
    for (int f = 0; f < V; f++)
    {
      if(f==0)
      {
        cout << "Matrix Number 2" << endl;
        outFile2 << "Matrix Number 2" << endl;
      }
      printf("%d \t %d\n", f, dist[f]);
      outFile2 << f << "\t" << dist[f] << endl;
    }

  //Indicate that the thread is finished
  cout << "End of first thread" << endl;

  //Increment the wait counter
  wait++;
  pthread_exit(NULL);
}
```



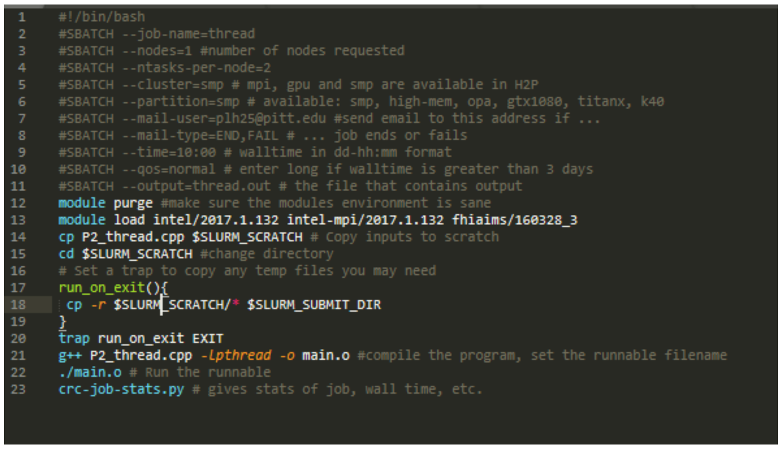
```cpp
/**
* secondThread thread to deal with the second half of the matricies
*/
void *secondThread(void *rank)
{
  //Indicate the second thread has started
  cout << "Start of second thread" << endl;

  //Open two files for the third and fourth matricies
    ofstream outFile3;
  outFile3.open("thirdGraph.txt");

  ofstream outFile4;
  outFile4.open("fourthGraph.txt");
```



```cpp
  //Dijkstra's Algorithm for the third matrix (1)
  int src =0;
  int dist[V];
  bool sptSet[V];
   for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;
      dist[src] = 0;
   for (int count = 0; count < V - 1; count++) {
    int min = INT_MAX, min_index;
    for(int v=0; v<50; v++)
    if(sptSet[v] == false && dist[v] <= min)
      min = dist[v], min_index=v;
    int u = min_index;
    sptSet[u] = true;
    for (int x = 0; x < V; x++)
      if (!sptSet[x] && thirdGraph[u][x] && dist[u] != INT_MAX && dist[u] + thirdGraph[u][x] < dist[x]) dist[x] = dist[u] + thirdGraph[u][x];
   }
    printf("Vertex Distance from Source for Matrix \n");
    for (int j = 0; j < V; j++)
    {
      if(j==0)
      {
        cout << "Matrix Number 3" << endl;
        outFile3 << "Matrix Number 3" << endl;
      }
      printf("%d \t %d\n", j, dist[j]);
      //Print results to the outfile
      outFile3 << j << "\t" << dist[j] << endl;
    }

  //Dijkstra's Algorithm for the fourth matrix (1)
  src =0;
   for (int a = 0; a < V; a++)
      dist[a] = INT_MAX, sptSet[a] = false;
      dist[src] = 0;
   for (int count = 0; count < V - 1; count++) {
    int min = INT_MAX, min_index;
    for(int z=0; z<50; z++)
    if(sptSet[z] == false && dist[z] <= min)
      min = dist[z], min_index=z;
    int u = min_index;
    sptSet[u] = true;
    for (int g = 0; g < V; g++)
      if (!sptSet[g] && fourthGraph[u][g] && dist[u] != INT_MAX && dist[u] + fourthGraph[u][g] < dist[g]) dist[g] = dist[u] + fourthGraph[u][g];
   }
    printf("Vertex Distance from Source for Matrix \n");
    cout << "Matrix Number 4" << endl;
    for (int p = 0; p < V; p++)
    {
      if(p==0)
      {
        cout << "Matrix Number 4" << endl;
        outFile4 << "Matrix Number 4" << endl;
      }
      printf("%d \t %d\n", p, dist[p]);
      //Print results to the outfile
      outFile4 << p << "\t" << dist[p] << endl;
    }

  //Indicate end of the second thread
  cout << "End of second thread" << endl;

  //Increment the wait counter
  wait++;
  pthread_exit(NULL);
}
```



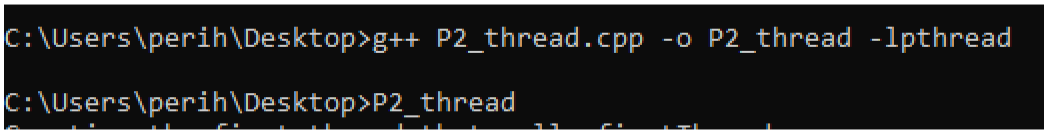
```cpp
/**
 * Main function that deals with creating and joining the threads as well as determining time to complete program. (4)
 */
int main()
{
  //Struct to hold the variables in order to find completion time
  struct timeval start, end;
  //Get the time the program starts
  gettimeofday(&start, NULL);

  pthread_t ids[NUM_THREADS];

  //Creating the first two threads to deal with half of the text files each
  cout << "Creating the first thread that calls firstThread" << endl;
  pthread_create(&ids[0], NULL, firstThread, (void *)0);
  cout << "Creating the second thread that calls secondThread" << endl;
  pthread_create(&ids[1], NULL, secondThread, (void *)1);

  //Waiting for the first two threads to complete
  for(int i=0; i<2; i++)
  {
    pthread_join(ids[i], NULL);
  }

  //Indicate the program is complete and time will now be calculated
  cout << "Process is complete" << endl;

  //Get the time the program finishes at
  gettimeofday(&end, NULL);

  //Calculate the time that was taken for the program to complete and print out the results (3)
  int micro_end = end.tv_sec * 1000000 + end.tv_usec;
  int micro_start = start.tv_sec * 1000000 + start.tv_usec;
  cout << "Total time for data parallel execution: " << (micro_end - micro_start) << " microseconds" << endl;

  return 0;
}

/** Sources - Project 2 Data Parallel Implementation
* (1) Dijkstra's Algorithm
* - https://www.tutorialspoint.com/c-cplusplus-program-for-dijkstra-s-shortest-path-algorithm
* -- I used this resource to implement a dijkstra's algorithm as the main part of my serial version.
* -- This is used to find the shortest path for each point in the array.
*
* (2) Binary Matrix
* -- https://www.daniweb.com/programming/software-development/threads/232485/random-binary-matrix
* -- For filling a binary matrix with random numbers
*
* (3) Canvas Lecture Notes
* - Bsharedmem.ppt Slide 7
* -- I was able to utilize timing for my program by using the lecture notes as an example.
* -- A struct is used and then the start and end times are referenced and used for calculations later on.
*
* (4) TutorialsPoint
* - https://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm
* -- Used as a reference for the syntax of threads and how to create/join
*/
```



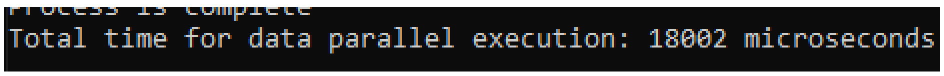
```
#!/bin/bash
#SBATCH --job-name=thread
#SBATCH --nodes=1 #number of nodes requested
#SBATCH --ntasks-per-node=2
#SBATCH --cluster=smp # mpi, gpu and smp are available in H2P
#SBATCH --partition=smp # available: smp, high-mem, opa, gtx1080, titanx, k40
#SBATCH --mail-user=plh25@pitt.edu #send email to this address if ...
#SBATCH --mail-type=END,FAIL # ... job ends or fails
#SBATCH --time=10:00 # walltime in dd-hh:mm format
#SBATCH --qos=normal # enter long if walltime is greater than 3 days
#SBATCH --output=thread.out # the file that contains output
module purge #make sure the modules environment is sane
module load intel/2017.1.132 intel-mpi/2017.1.132 fhiaims/160328_3
cp P2_thread.cpp $SLURM_SCRATCH # Copy inputs to scratch
cd $SLURM_SCRATCH #change directory
# Set a trap to copy any temp files you may need
run_on_exit(){
 cp -r $SLURM_SCRATCH/* $SLURM_SUBMIT_DIR
}
trap run_on_exit EXIT
g++ P2_thread.cpp -lpthread -o main.o #compile the program, set the runnable filename
./main.o # Run the runnable
crc-job-stats.py # gives stats of job, wall time, etc.
```

The screenshots on the page above are for the data parallel implementation of the project. I did not include the hardcoded matrices that were based on the serial version of the code. The code may also be viewed in the submitted files as P2_thread.cpp and P2_thread.script. Zoom in for a better view of the code or open the specified file.

Output:
From my local machine:



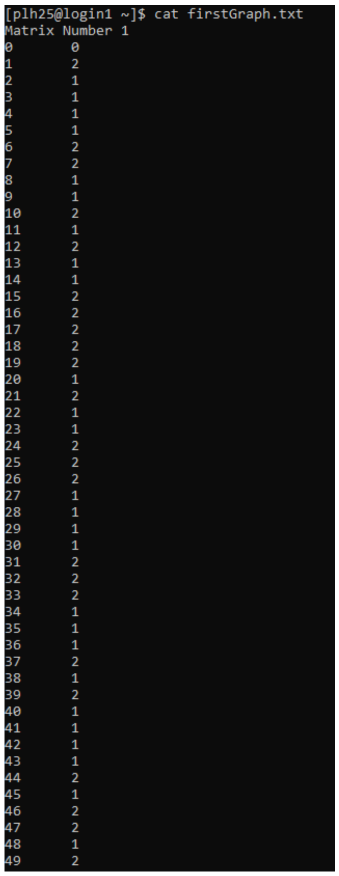
```
C:\Users\perih\Desktop>g++ P2_thread.cpp -o P2_thread -lpthread

C:\Users\perih\Desktop>P2_thread
[illegible]
```



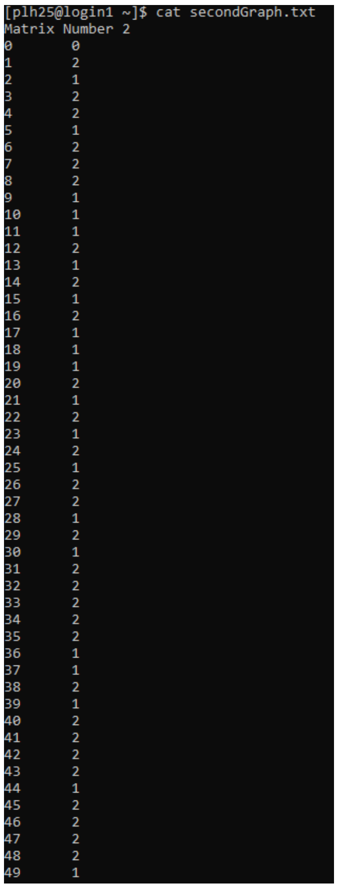
```
Process is complete
Total time for data parallel execution: 18002 microseconds
```

The execution of the program took 18002 microseconds on my local machine.

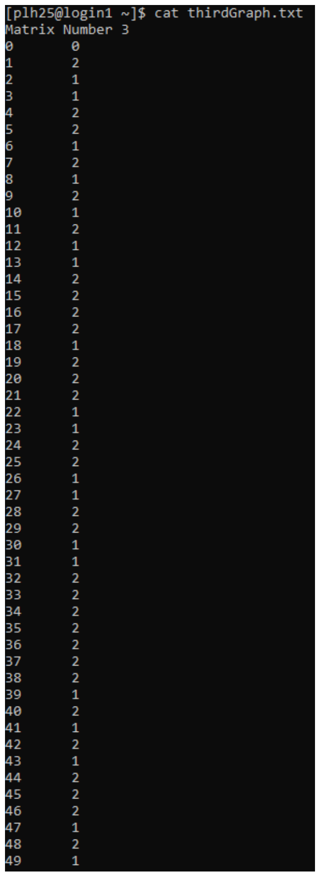
CRC Server:



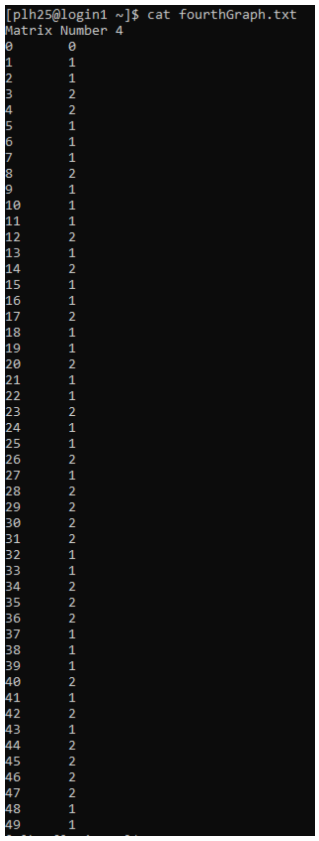
```
[plh25@login1 ~]$ cat firstGraph.txt
Matrix Number 1
0       0
1       2
2       1
3       1
4       1
5       1
6       2
7       2
8       1
9       1
10      2
11      1
12      2
13      1
14      1
15      2
16      2
17      2
18      2
19      2
20      1
21      2
22      1
23      1
24      2
25      2
26      2
27      1
28      1
29      1
30      1
31      2
32      2
33      2
34      1
35      1
36      1
37      2
38      1
39      2
40      1
41      1
42      1
43      1
44      2
45      1
46      2
47      2
48      1
49      2
```



```
[plh25@login1 ~]$ cat secondGraph.txt
Matrix Number 2
0       0
1       2
2       1
3       2
4       2
5       1
6       2
7       2
8       2
9       1
10      1
11      1
12      2
13      1
14      2
15      1
16      2
17      1
18      1
19      1
20      2
21      1
22      2
23      1
24      2
25      1
26      2
27      2
28      1
29      2
30      1
31      2
32      2
33      2
34      2
35      2
36      1
37      1
38      2
39      1
40      2
41      2
42      2
43      2
44      1
45      2
46      2
47      2
48      2
49      1
```



```
[plh25@login1 ~]$ cat thirdGraph.txt
Matrix Number 3
0       0
1       2
2       1
3       1
4       2
5       2
6       1
7       2
8       1
9       2
10      1
11      2
12      1
13      1
14      2
15      2
16      2
17      2
18      1
19      2
20      2
21      2
22      1
23      1
24      2
25      2
26      1
27      1
28      2
29      2
30      1
31      1
32      2
33      2
34      2
35      2
36      2
37      2
38      2
39      1
40      2
41      1
42      2
43      1
44      2
45      2
46      2
47      1
48      2
49      1
```



```
[plh25@login1 ~]$ cat fourthGraph.txt
Matrix Number 4
0       0
1       1
2       1
3       2
4       2
5       1
6       1
7       1
8       2
9       1
10      1
11      1
12      2
13      1
14      2
15      1
16      1
17      2
18      1
19      1
20      2
21      1
22      1
23      2
24      1
25      1
26      2
27      1
28      2
29      2
30      2
31      2
32      1
33      1
34      2
35      2
36      2
37      1
38      1
39      1
40      2
41      1
42      2
43      1
44      2
45      2
46      2
47      2
48      1
49      1
```

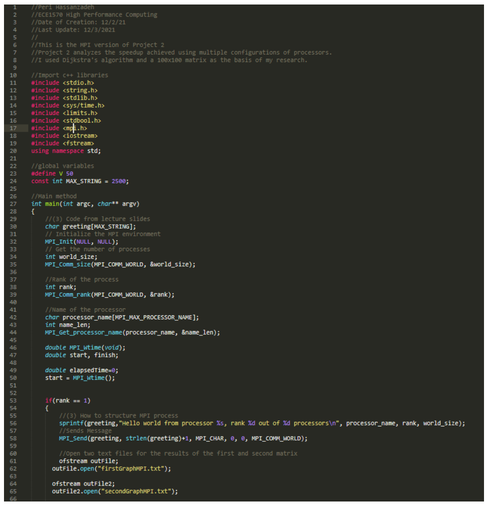
From the CRC, the execution took an average of 649 microseconds which is also significantly less than that of my local machine similar to what happened in the serial implementation.

The screenshots above represent the output of Dijkstra's algorithm within different text files corresponding to different matrices. It was easier to view the results within a text file since the output on the command prompt was skewed due to the threads running at the same time. Analysis became much more simple once the data was pushed to a text file.

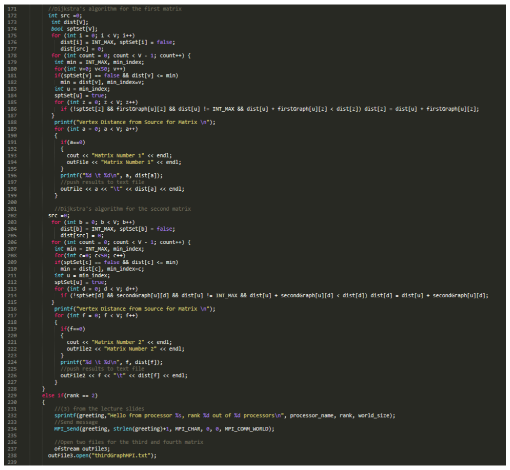
Loosely Coupled (Cluster) Version

The files that correspond with the cluster version are as follows:
P2_mpi.cpp - implementation of clusters
P2_mpi.script - to compile on CRC
mpi.out - Result from running on CRC script
First through fourthGraph.txt - Text files containing results

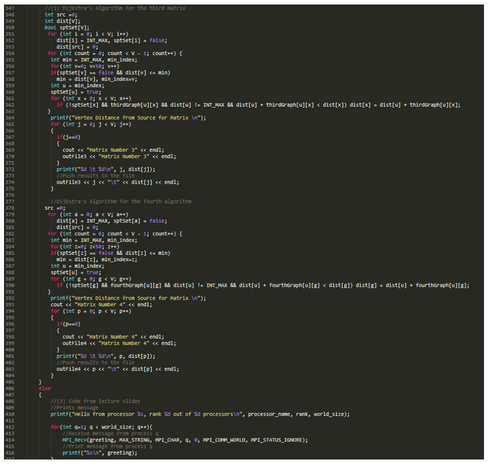
Screenshots of Code:



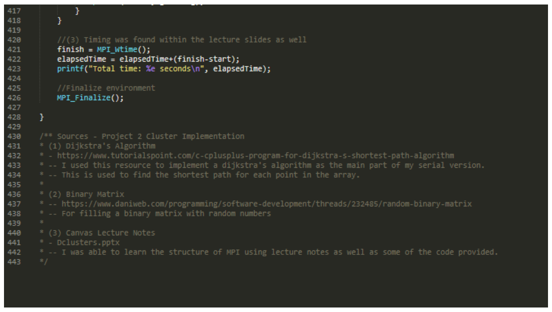
```cpp
//Peri Hassanzadeh
//ECE1570 High Performance Computing
//Date of Creation: 12/2/21
//Last Update: 12/3/2021
//
//This is the MPI version of Project 2
//Project 2 analyzes the speedup achieved using multiple configurations of processors.
//I used Dijkstra's algorithm and a 100x100 matrix as the basis of my research.

//Import c++ libraries
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/time.h>
#include <limits.h>
#include <stdbool.h>
#include <mpi.h>
#include <iostream>
#include <fstream>
using namespace std;

//global variables
#define V 50
const int MAX_STRING = 2500;

//Main method
int main(int argc, char** argv)
{
    //(3) Code from lecture slides
    char greeting[MAX_STRING];
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);
    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    //Rank of the process
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    //Name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    double MPI_Wtime(void);
    double start, finish;

    double elapsedTime=0;
    start = MPI_Wtime();


    if(rank == 1)
    {
        //(3) How to structure MPI process
        sprintf(greeting,"Hello world from processor %s, rank %d out of %d processors\n", processor_name, rank, world_size);
        //Sends Message
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);

        //Open two text files for the results of the first and second matrix
        ofstream outFile;
      outFile.open("firstGraphMPI.txt");

      ofstream outFile2;
      outFile2.open("secondGraphMPI.txt");

```



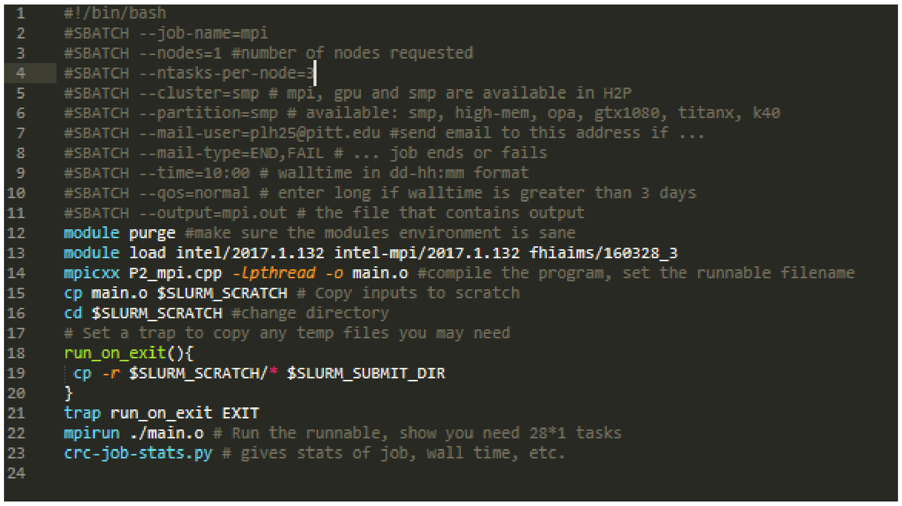
```cpp
      //Dijkstra's algorithm for the first matrix
      int src =0;
       int dist[V];
       bool sptSet[V];
       for (int i = 0; i < V; i++)
          dist[i] = INT_MAX, sptSet[i] = false;
          dist[src] = 0;
       for (int count = 0; count < V - 1; count++) {
        int min = INT_MAX, min_index;
        for(int v=0; v<50; v++)
        if(sptSet[v] == false && dist[v] <= min)
          min = dist[v], min_index=v;
        int u = min_index;
        sptSet[u] = true;
        for (int z = 0; z < V; z++)
          if (!sptSet[z] && firstGraph[u][z] && dist[u] != INT_MAX && dist[u] + firstGraph[u][z] < dist[z]) dist[z] = dist[u] + firstGraph[u][z];
       }
        printf("Vertex Distance from Source for Matrix \n");
        for (int a = 0; a < V; a++)
        {
          if(a==0)
          {
            cout << "Matrix Number 1" << endl;
            outFile << "Matrix Number 1" << endl;
          }
          printf("%d \t %d\n", a, dist[a]);
          //push results to text file
          outFile << a << "\t" << dist[a] << endl;
        }

        //Dijkstra's algorithm for the second matrix
      src =0;
       for (int b = 0; b < V; b++)
          dist[b] = INT_MAX, sptSet[b] = false;
          dist[src] = 0;
       for (int count = 0; count < V - 1; count++) {
        int min = INT_MAX, min_index;
        for(int c=0; c<50; c++)
        if(sptSet[c] == false && dist[c] <= min)
          min = dist[c], min_index=c;
        int u = min_index;
        sptSet[u] = true;
        for (int d = 0; d < V; d++)
          if (!sptSet[d] && secondGraph[u][d] && dist[u] != INT_MAX && dist[u] + secondGraph[u][d] < dist[d]) dist[d] = dist[u] + secondGraph[u][d];
       }
        printf("Vertex Distance from Source for Matrix \n");
        for (int f = 0; f < V; f++)
        {
          if(f==0)
          {
            cout << "Matrix Number 2" << endl;
            outFile2 << "Matrix Number 2" << endl;
          }
          printf("%d \t %d\n", f, dist[f]);
          //push results to text file
          outFile2 << f << "\t" << dist[f] << endl;
        }
    }
    else if(rank == 2)
    {
        //(3) from the lecture slides
        sprintf(greeting,"Hello from processor %s, rank %d out of %d processors\n", processor_name, rank, world_size);
        //Send message
        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);

        //Open two files for the third and fourth matrix
        ofstream outFile3;
      outFile3.open("thirdGraphMPI.txt");

```



```cpp
      //(1) Dijkstra's Algorithm for the third matrix
      int src =0;
      int dist[V];
      bool sptSet[V];
       for (int i = 0; i < V; i++)
          dist[i] = INT_MAX, sptSet[i] = false;
          dist[src] = 0;
       for (int count = 0; count < V - 1; count++) {
        int min = INT_MAX, min_index;
        for(int v=0; v<50; v++)
        if(sptSet[v] == false && dist[v] <= min)
          min = dist[v], min_index=v;
        int u = min_index;
        sptSet[u] = true;
        for (int x = 0; x < V; x++)
          if (!sptSet[x] && thirdGraph[u][x] && dist[u] != INT_MAX && dist[u] + thirdGraph[u][x] < dist[x]) dist[x] = dist[u] + thirdGraph[u][x];
       }
        printf("Vertex Distance from Source for Matrix \n");
        for (int j = 0; j < V; j++)
        {
          if(j==0)
          {
            cout << "Matrix Number 3" << endl;
            outFile3 << "Matrix Number 3" << endl;
          }
          printf("%d \t %d\n", j, dist[j]);
          //Push results to the file
          outFile3 << j << "\t" << dist[j] << endl;
        }

        //Dijkstra's Algorithm for the fourth algorithm
      src =0;
       for (int a = 0; a < V; a++)
          dist[a] = INT_MAX, sptSet[a] = false;
          dist[src] = 0;
       for (int count = 0; count < V - 1; count++) {
        int min = INT_MAX, min_index;
        for(int z=0; z<50; z++)
        if(sptSet[z] == false && dist[z] <= min)
          min = dist[z], min_index=z;
        int u = min_index;
        sptSet[u] = true;
        for (int g = 0; g < V; g++)
          if (!sptSet[g] && fourthGraph[u][g] && dist[u] != INT_MAX && dist[u] + fourthGraph[u][g] < dist[g]) dist[g] = dist[u] + fourthGraph[u][g];
       }
        printf("Vertex Distance from Source for Matrix \n");
        cout << "Matrix Number 4" << endl;
        for (int p = 0; p < V; p++)
        {
          if(p==0)
          {
            cout << "Matrix Number 4" << endl;
            outFile4 << "Matrix Number 4" << endl;
          }
          printf("%d \t %d\n", p, dist[p]);
          //Push results to the file
          outFile4 << p << "\t" << dist[p] << endl;
        }
    }
    else
    {
        //(3) Code from lecture slides
        //Prints message
        printf("Hello from processor %s, rank %d out of %d processors\n", processor_name, rank, world_size);

        for(int q=1; q < world_size; q++){
            //Receive message from process q
            MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            //Print message from process q
            printf("%s\n", greeting);
```



```cpp
            }
        }

        //(3) Timing was found within the lecture slides as well
        finish = MPI_Wtime();
        elapsedTime = elapsedTime+(finish-start);
        printf("Total time: %e seconds\n", elapsedTime);

        //Finalize environment
        MPI_Finalize();

    }

    /** Sources - Project 2 Cluster Implementation
    * (1) Dijkstra's Algorithm
    * - https://www.tutorialspoint.com/c-cplusplus-program-for-dijkstra-s-shortest-path-algorithm
    * -- I used this resource to implement a dijkstra's algorithm as the main part of my serial version.
    * -- This is used to find the shortest path for each point in the array.
    *
    * (2) Binary Matrix
    * -- https://www.daniweb.com/programming/software-development/threads/232485/random-binary-matrix
    * -- For filling a binary matrix with random numbers
    *
    * (3) Canvas Lecture Notes
    * - Dclusters.pptx
    * -- I was able to learn the structure of MPI using lecture notes as well as some of the code provided.
    */
```



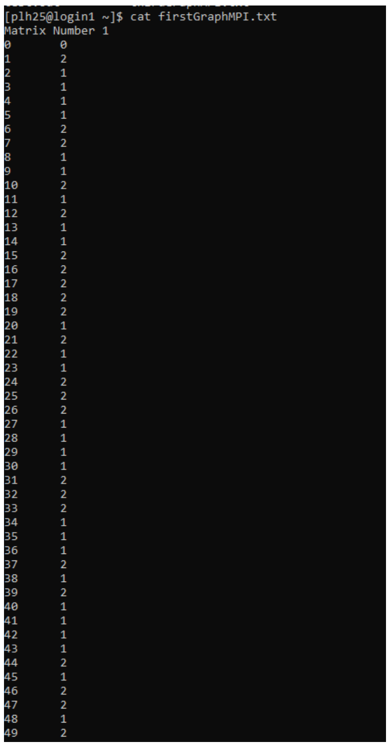
```bash
#!/bin/bash
#SBATCH --job-name=mpi
#SBATCH --nodes=1 #number of nodes requested
#SBATCH --ntasks-per-node=3
#SBATCH --cluster=smp # mpi, gpu and smp are available in H2P
#SBATCH --partition=smp # available: smp, high-mem, opa, gtx1080, titanx, k40
#SBATCH --mail-user=plh25@pitt.edu #send email to this address if ...
#SBATCH --mail-type=END,FAIL # ... job ends or fails
#SBATCH --time=10:00 # walltime in dd-hh:mm format
#SBATCH --qos=normal # enter long if walltime is greater than 3 days
#SBATCH --output=mpi.out # the file that contains output
module purge #make sure the modules environment is sane
module load intel/2017.1.132 intel-mpi/2017.1.132 fhiaims/160328_3
mpicxx P2_mpi.cpp -lpthread -o main.o #compile the program, set the runnable filename
cp main.o $SLURM_SCRATCH # Copy inputs to scratch
cd $SLURM_SCRATCH #change directory
# Set a trap to copy any temp files you may need
run_on_exit(){
 cp -r $SLURM_SCRATCH/* $SLURM_SUBMIT_DIR
}
trap run_on_exit EXIT
mpirun ./main.o # Run the runnable, show you need 28*1 tasks
crc-job-stats.py # gives stats of job, wall time, etc.

```

The screenshots above are for the task cluster implementation of the project. The code may also be viewed in the submitted files as P2_mpi.cpp and P2_mpi.script. Zoom in for a better view of the code or open the specified file.

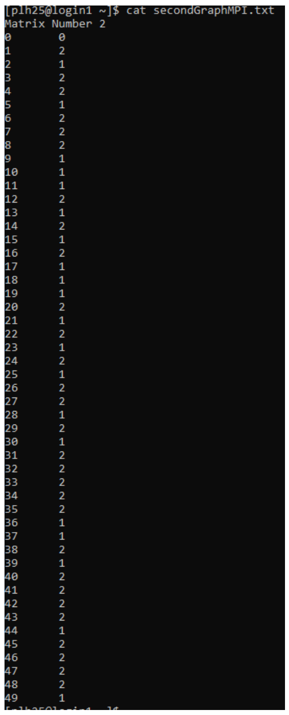
Output:

From my local machine, the program is unable to be executed.

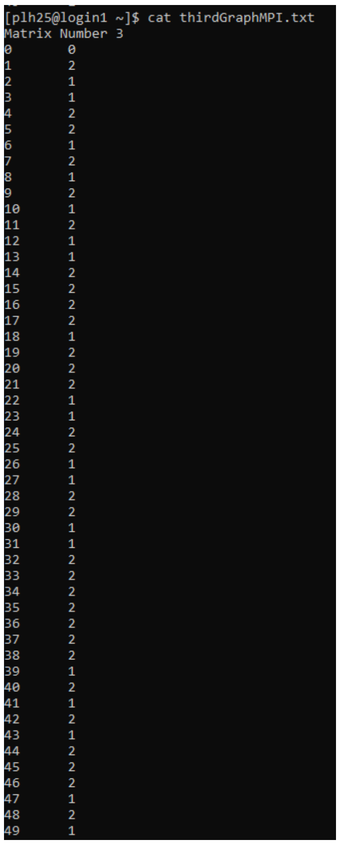
Bigrigg Server:



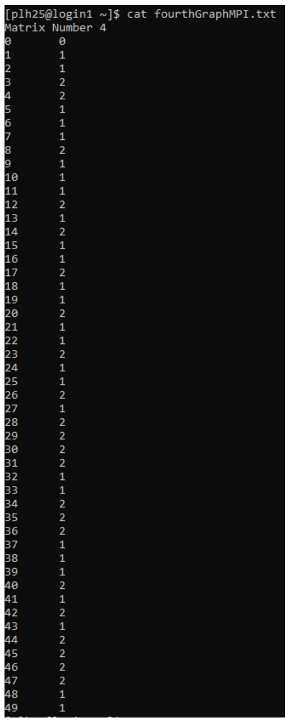
```
[plh25@login1 ~]$ cat firstGraphMPI.txt
Matrix Number 1
0       0
1       2
2       1
3       1
4       1
5       1
6       2
7       2
8       1
9       1
10      2
11      1
12      2
13      1
14      1
15      2
16      2
17      2
18      2
19      2
20      1
21      2
22      1
23      1
24      2
25      2
26      2
27      1
28      1
29      1
30      1
31      2
32      2
33      2
34      1
35      1
36      1
37      2
38      1
39      2
40      1
41      1
42      1
43      1
44      2
45      1
46      2
47      2
48      1
49      2
```



```
[plh25@login1 ~]$ cat secondGraphMPI.txt
Matrix Number 2
0       0
1       2
2       1
3       2
4       2
5       1
6       2
7       2
8       2
9       1
10      1
11      1
12      2
13      1
14      2
15      1
16      2
17      1
18      1
19      1
20      2
21      1
22      2
23      1
24      2
25      1
26      2
27      2
28      1
29      2
30      1
31      2
32      2
33      2
34      2
35      2
36      1
37      1
38      2
39      1
40      2
41      2
42      2
43      2
44      1
45      2
46      2
47      2
48      2
49      1
```



```
[plh25@login1 ~]$ cat thirdGraphMPI.txt
Matrix Number 3
0       0
1       2
2       1
3       1
4       2
5       2
6       1
7       2
8       1
9       2
10      1
11      2
12      1
13      1
14      2
15      2
16      2
17      2
18      1
19      2
20      2
21      2
22      1
23      1
24      2
25      2
26      1
27      1
28      2
29      2
30      1
31      1
32      2
33      2
34      2
35      2
36      2
37      2
38      2
39      1
40      2
41      1
42      2
43      1
44      2
45      2
46      2
47      1
48      2
49      1
```



```
[plh25@login1 ~]$ cat fourthGraphMPI.txt
Matrix Number 4
0       0
1       1
2       1
3       2
4       2
5       1
6       1
7       1
8       2
9       1
10      1
11      1
12      2
13      1
14      2
15      1
16      1
17      2
18      1
19      1
20      2
21      1
22      1
23      2
24      1
25      1
26      2
27      1
28      2
29      2
30      2
31      2
32      1
33      1
34      2
35      2
36      2
37      1
38      1
39      1
40      2
41      1
42      2
43      1
44      2
45      2
46      2
47      2
48      1
49      1
```

From the ssh server bigrigg.com, the execution took an average of 6.1979284e-04 seconds.

Optimizations

There are multiple optimizations that could be made to each implementation, but I would like to address some of the more obvious optimizations that would make a significant difference in the implementation and possibly efficiency of the program.

Starting off specifically with the serial implementation, the algorithm should have been as stated on the Canvas webpage, but I decided to use Dijkstra’s algorithm instead as discussed on the discussion board. I would have liked to have been able to find a way to split the matrices so that the source node was the same for each matrix and then the end result could be compiled together that way the results would be all for one matrix instead of four different ones. This could be applied to the thread and cluster versions as well.

Another large optimization that could be made is to create more threads in the data parallel version to be more of a streamlined process being able to break up the data even more. Specifically putting one matrix per thread.

Finally, the last optimization I wanted to mention was to be able to use more processes in the cluster version. With better time management I would have been able to better understand the purpose of clusters and be able to compare them to the threads with deeper analysis. Instead I spent time troubleshooting the CRC and using scripts.