Programming Assignment Report 3

Tridiagonal Matrix Solver

Approach:

Both the PPT and PDD algorithms were detailed in the instructions so it was not difficult to check values on each step.

Setup:

1. Ask the user for matrix size to initialize arrays
2. Read data based on input
3. Check inputted data to avoid conflict
4. Solve the whole matrix on rank = 0 to find serial runtime.
5. Run the algorithm steps
6. Write results to a file

Insight Gained and Problems:

The biggest problem was understanding the Z matrix and h vector in Step 4 of the PPT. This was due to the construction of the matrix and understanding the values passed in Step 2 played a big part in dictating how the vector was formed.

After understanding what the v and w vectors signified and why values v(1 -> m-1) were dropped in the communication message, the vectors and matrix used in the calculation became more significant and gave insight to the creation of Z and h.

Optimizations:

1. Step 3 in PPT was initially attempted with a 6 MPI_Send() and 6 MPI_Recv(). This was changed to MPI_BCast(). Since this does the communication as the previous implementation but faster due to the communication being optimized, this option was chosen

2. During the implementation of the algorithms, many data structures were used. To reduce space complexity and access time, these structures were freed efficiently.

Attachment:

PPT.c	The c file with PPT Implementation (Directory with test cases)
PPT.c	The c file with PDD Implementation (Directory with test cases)
final_results.csv	The CSV files with results of the algorithm with mentioned parameters
data.csv	The excel file with the data used for the plots

Instructions:

For PPT & PDD directories

Compile: mpicc PPT.c -o ans

mpicc PPD.c -o ans

Run: mpiexec -n p ./ans (p = # of processors)

Test Case: **Provided Case**

mpicc PPT.c -o ans

mpiexec -n **2** ./ans

Do you want to use provided test case (Y/…)? Y

Enter the size of the matrix (1 value): 4096

Correctness_Test Case # 1 (Checking for correct values for 2 processors):

mpicc PPT.c -o ans

mpiexec -n **2** ./ans

Do you want to use provided test case (Y/…)? N

Do you want to use created test case 1 (Y/…)? Y

Enter the size of the matrix (1 value): 12

(This should print the values for x for serial and parallel)

Correctness_Test Case # 2 (Checking for correct values for n processors): (n >2)

mpicc PPT.c -o ans

mpiexec -n **4** ./ans

Do you want to use provided test case (Y/…)? N

Do you want to use created test case 1 (Y/…)? N

Do you want to use created test case 1 (Y/…)? Y

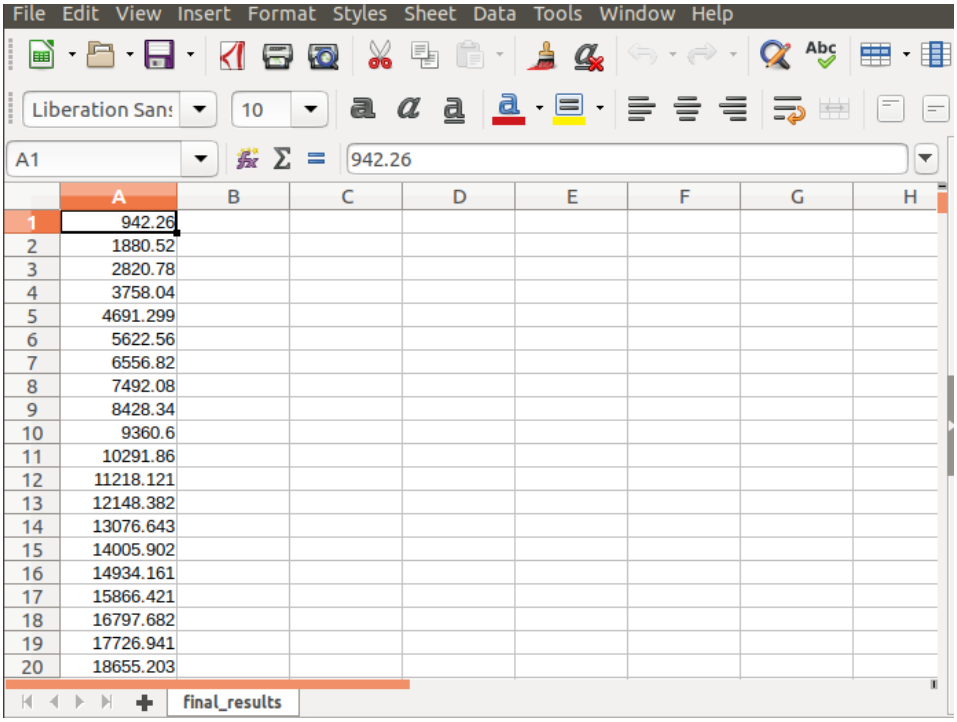
Enter the size of the matrix (1 value): 12

(This should print the values for x for serial and parallel)

Results (PPT & PDD):

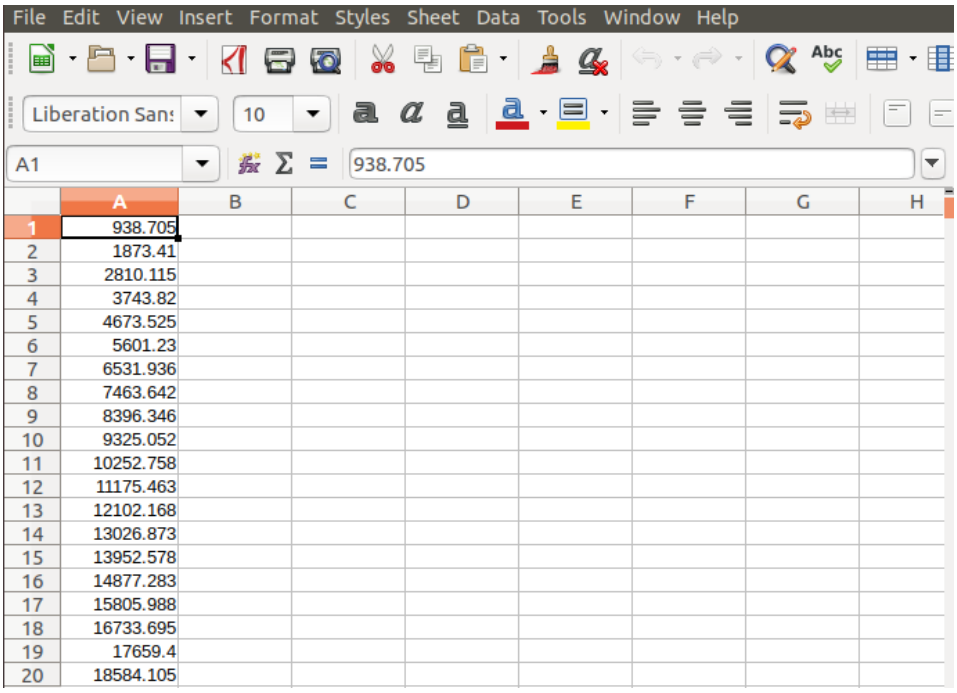
final_results.csv holds the values for the provided test case and there is some error due to rounding.

PPT (p = 32):



	A	B	C	D	E	F	G	H
1	942.26							
2	1880.52							
3	2820.78							
4	3758.04							
5	4691.299							
6	5622.56							
7	6556.82							
8	7492.08							
9	8428.34							
10	9360.6							
11	10291.86							
12	11218.121							
13	12148.382							
14	13076.643							
15	14005.902							
16	14934.161							
17	15866.421							
18	16797.682							
19	17726.941							
20	18655.203							

PDD (p = 2):



	A	B	C	D	E	F	G	H
1	938.705							
2	1873.41							
3	2810.115							
4	3743.82							
5	4673.525							
6	5601.23							
7	6531.936							
8	7463.642							
9	8396.346							
10	9325.052							
11	10252.758							
12	11175.463							
13	12102.168							
14	13026.873							
15	13952.578							
16	14877.283							
17	15805.988							
18	16733.695							
19	17659.4							
20	18584.105							

Correctness Test Case 1:

PPT:

```
jjoyson@jjoyson-VirtualBox:~/Desktop/CS 546/3/PPT$ mpicc PPT.c -o ans
jjoyson@jjoyson-VirtualBox:~/Desktop/CS 546/3/PPT$ mpiexec -n 2 ./ans
Do you want to use provided test case (Y/...)? N
Do you want to use created test case 1 (Y/...)? Y
Enter the size of the matrix (1 value): 12
Serial Results:
x[0] = -0.366
x[1] = -0.464
x[2] = -0.490
x[3] = -0.497
x[4] = -0.499
x[5] = -0.500
x[6] = -0.500
x[7] = -0.499
x[8] = -0.497
x[9] = -0.490
x[10] = -0.464
x[11] = -0.366
Serial Execution Time 0.00 seconds

Parallel Results:
x[0] = -0.366
x[1] = -0.464
x[2] = -0.490
x[3] = -0.497
x[4] = -0.499
x[5] = -0.500
x[6] = -0.500
x[7] = -0.499
x[8] = -0.497
x[9] = -0.490
x[10] = -0.464
x[11] = -0.366
Execution Time 0.08 seconds
```

PDD:

```
jjoyson@jjoyson-VirtualBox:~/Desktop/CS 546/3/PDD$ mpiexec -n 2 ./ans
Do you want to use provided test case (Y/...)? N
Do you want to use created test case 1 (Y/...)? Y
Enter the size of the matrix (1 value): 12
Serial Results:
x[0] = -0.366
x[1] = -0.464
x[2] = -0.490
x[3] = -0.497
x[4] = -0.499
x[5] = -0.500
x[6] = -0.500
x[7] = -0.499
x[8] = -0.497
x[9] = -0.490
x[10] = -0.464
x[11] = -0.366
Serial Execution Time 0.00 seconds

Parallel Results:
x[0] = -0.366
x[1] = -0.464
x[2] = -0.490
x[3] = -0.497
x[4] = -0.499
x[5] = -0.500
x[6] = -0.500
x[7] = -0.499
x[8] = -0.497
x[9] = -0.490
x[10] = -0.464
x[11] = -0.366
Execution Time 0.02 seconds
```

Correctness Test Case 2:

PPT:

```
jjoyson@jjoyson-VirtualBox:~/Desktop/CS 546/3/PPT$ mpicc PPT.c -o ans
jjoyson@jjoyson-VirtualBox:~/Desktop/CS 546/3/PPT$ mpiexec -n 4 ./ans
Do you want to use provided test case (Y/...)? N
Do you want to use created test case 1 (Y/...)? N
Do you want to use created test case 2 (Y/...)? Y
Enter the size of the matrix (1 value): 12
Serial Results:
x[0] = -0.366
x[1] = -0.464
x[2] = -0.490
x[3] = -0.497
x[4] = -0.499
x[5] = -0.500
x[6] = -0.500
x[7] = -0.499
x[8] = -0.497
x[9] = -0.490
x[10] = -0.464
x[11] = -0.366
Serial Execution Time 0.00 seconds

Parallel Results:
x[0] = -0.366
x[1] = -0.464
x[2] = -0.490
x[3] = -0.497
x[4] = -0.499
x[5] = -0.500
x[6] = -0.500
x[7] = -0.499
x[8] = -0.497
x[9] = -0.490
x[10] = -0.464
x[11] = -0.366
Execution Time 0.37 seconds
```

PDD:

```
jjoyson@jjoyson-VirtualBox:~/Desktop/CS 546/3/PDD$ mpicc PDD.c -o ans
jjoyson@jjoyson-VirtualBox:~/Desktop/CS 546/3/PDD$ mpiexec -n 4 ./ans
Do you want to use provided test case (Y/...)? N
Do you want to use created test case 1 (Y/...)? N
Do you want to use created test case 2 (Y/...)? Y
Enter the size of the matrix (1 value): 12
Serial Results:
x[0] = -0.366
x[1] = -0.464
x[2] = -0.490
x[3] = -0.497
x[4] = -0.499
x[5] = -0.500
x[6] = -0.500
x[7] = -0.499
x[8] = -0.497
x[9] = -0.490
x[10] = -0.464
x[11] = -0.366
Serial Execution Time 0.00 seconds

Parallel Results:
x[0] = -0.366
x[1] = -0.463
x[2] = -0.488
x[3] = -0.497
x[4] = -0.498
x[5] = -0.497
x[6] = -0.497
x[7] = -0.498
x[8] = -0.497
x[9] = -0.488
x[10] = -0.463
x[11] = -0.366
Execution Time 0.07 seconds
```

Performance analysis:

Measuring the times for serial versus parallel was difficult in this program since the serial time was always 0.00 seconds. Since the problem size was not big enough and both PPT & PDD was implemented, the performance of both the algorithms was measured against each other; the times for these algorithms were > 0.00 seconds.

There were two variables that could be varied in the algorithm is the size of the matrix and the number of processors used. The following values were selected, and the performance metric used was execution time. The provided matrix was used for the test by varying the size. Ex: if 20 was used as the size, only the first 20 values for d.c would be used.

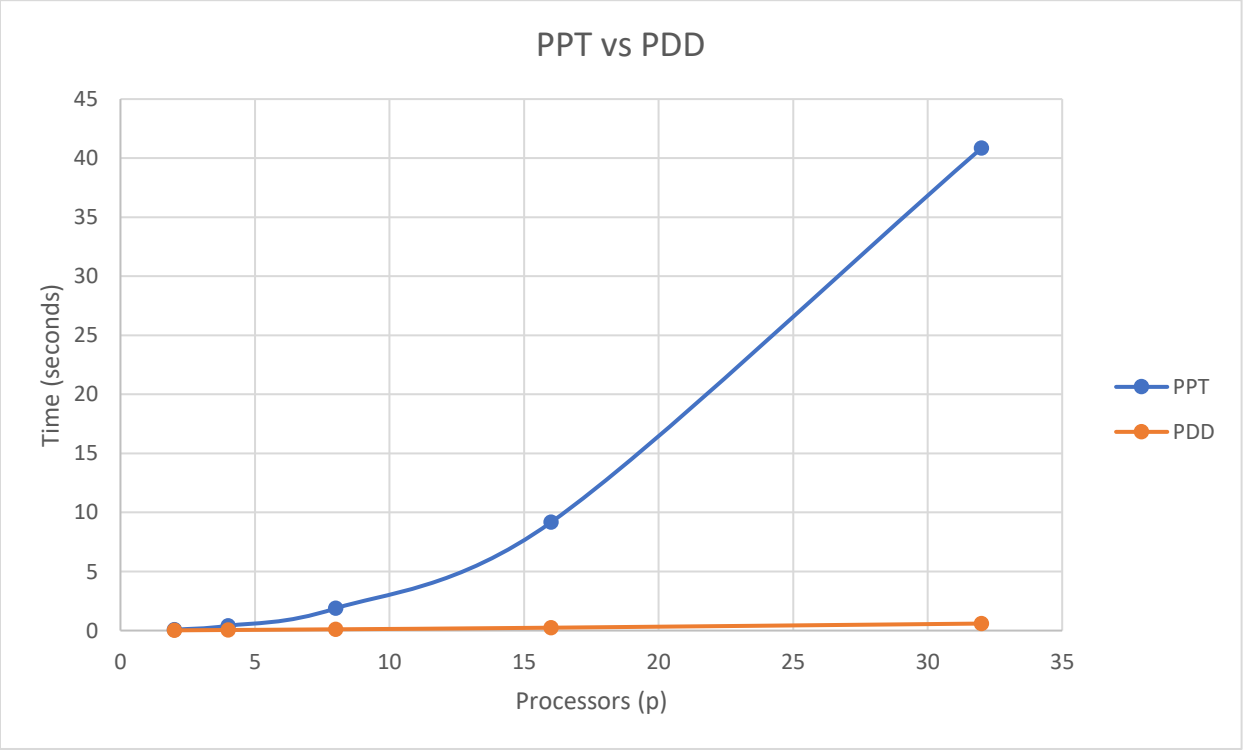
Processors: 2,4,8,16,32

Size: 128, 256, 512, 1024, 4096

PPT	2	4	8	16	32
N = 128	0.07	0.47	1.99	9.44	41.31
256	0.06	0.44	1.87	9.12	39.76
512	0.06	0.39	1.8	9.02	40.06
1024	0.06	0.35	1.84	8.81	38.85
2048	0.09	0.34	1.83	9.19	42.4
4096	0.07	0.44	1.95	9.39	42.68
AVE	0.068333	0.405	1.88	9.161667	40.84333

PDD	P = 2	4	8	16	32
N = 128	0.02	0.05	0.1	0.29	0.61
256	0.02	0.05	0.12	0.22	0.52
512	0.02	0.05	0.09	0.21	0.65
1024	0.02	0.04	0.09	0.22	0.57
2048	0.02	0.06	0.11	0.25	0.67
4096	0.02	0.04	0.14	0.26	0.53
AVE	0.02	0.048333	0.108333	0.241667	0.591667

After a few trials, it was evident that the matrix size had no impact on the time. Rather, only the number of processors did. So, the times were averaged and used for the PPT vs PDD graph.



Looking at the graph above, it is evident that the PDD algorithm is much faster than the PPT algorithm. This was due to the n-1 less communication made during the sharing of data and not needing to solve a pent-diagonal matric (Zy = h). As the processors increase, the execution time increases for both algorithms. The serial algorithm (ran the whole matrix on rank = 0) outputted a 0.00 sec execution time so it was not added to this graph.