Chris Jimenez
Dr. Mohamed Zahran
Parallel Computing
March 15, 2014

Programming Assignment 1 Report

Introduction

For this programming assignment, MPI was used to implement a method to solve a group of linear equations. That is, given a set of equations and corresponding unknowns, the implemented program will solve for those unknowns within a given relative error.

Input/Output

The program used the Jacobi method to solve for these unknowns, which is similar to the Gaussian-Seidel method. The input of the program are the given .txt files `small.txt`, `medium.txt` and `large.txt`. The output of the program is the calculated unknowns and the number of iterations it took to calculate them.

Data

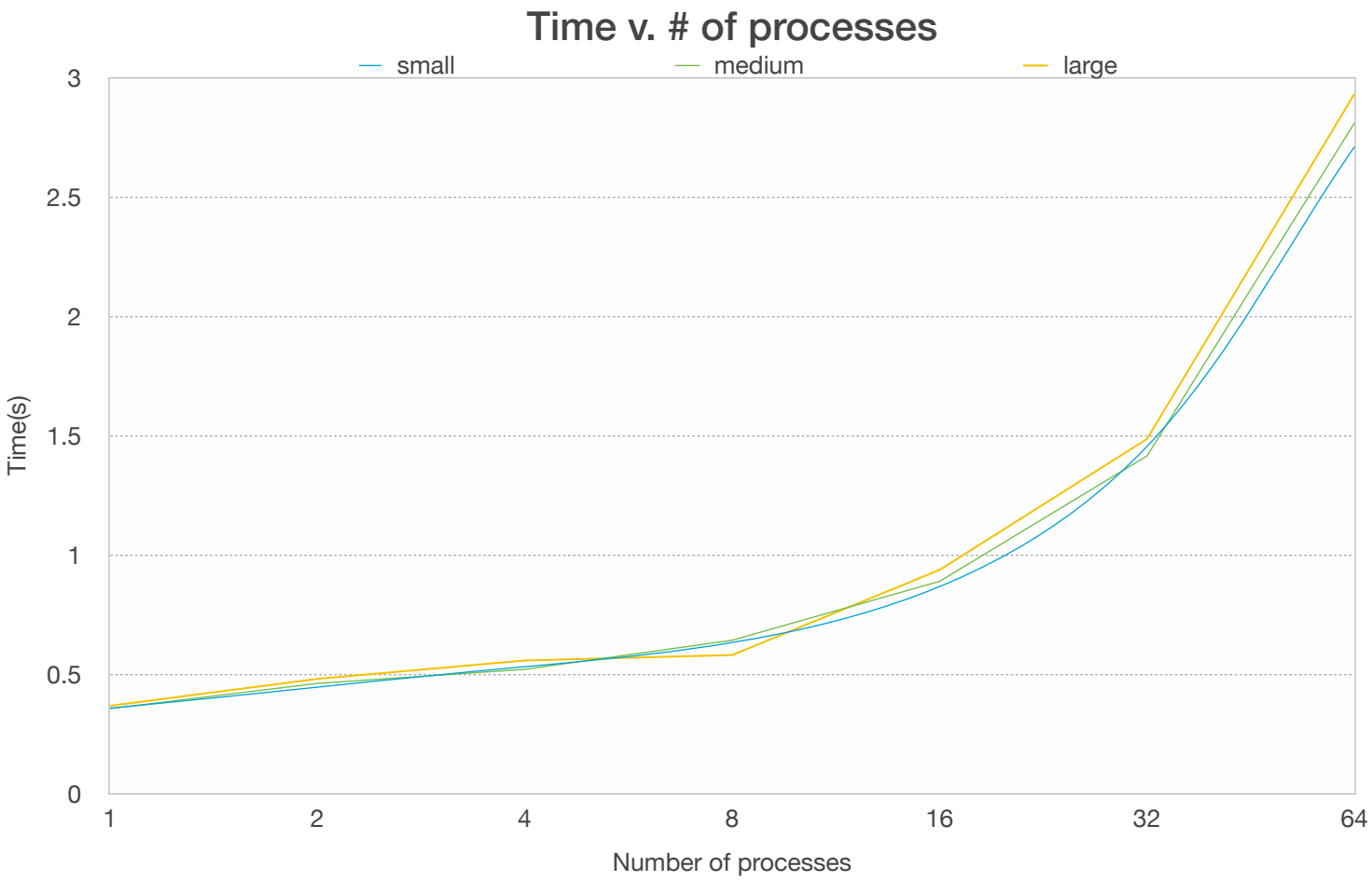
The average running time for each input file for different number of processes are shown in the tables below. A visual representation between the average times of the three input files can be seen in the first graph on page 3. The speedup graph can also be seen on page 3.

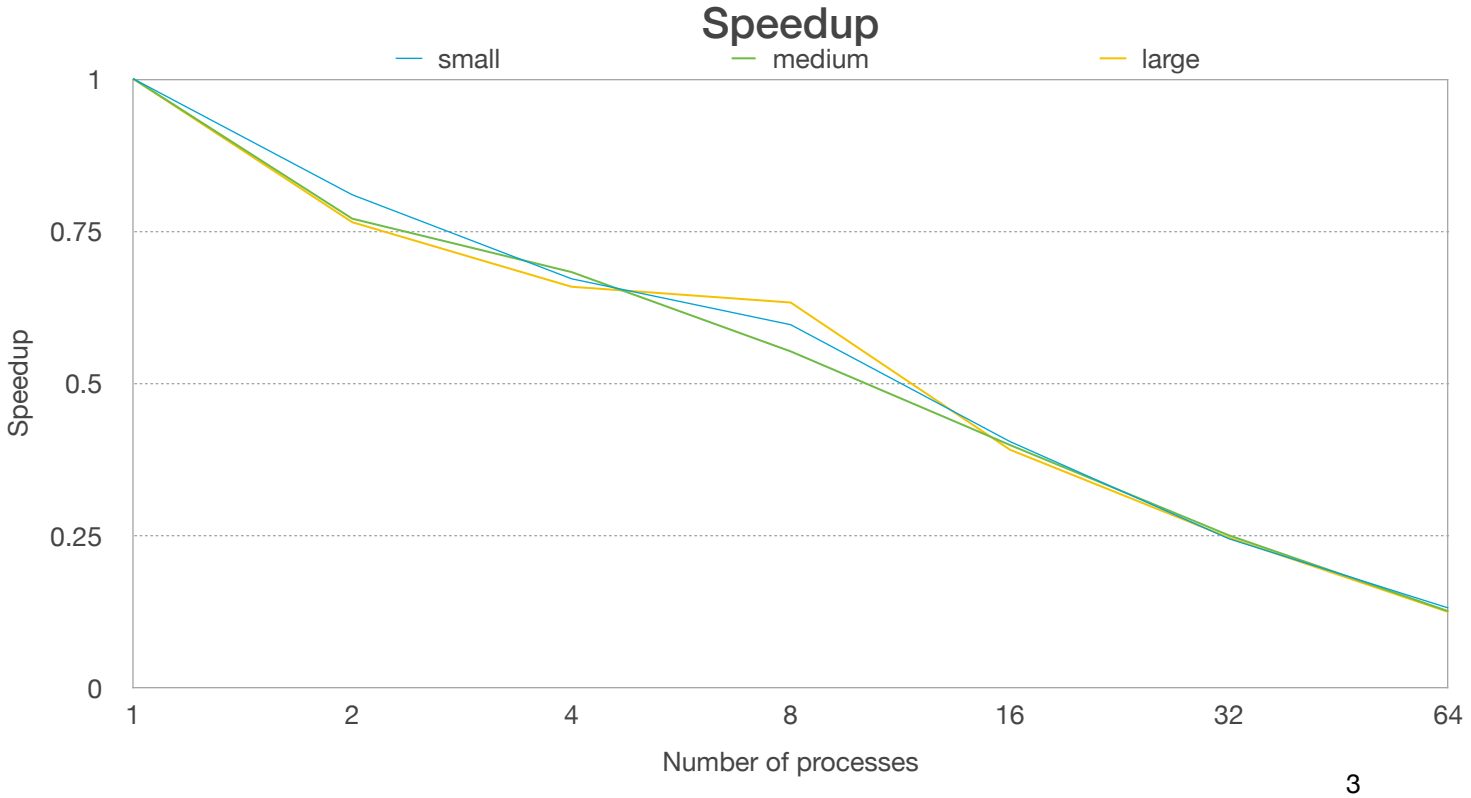
Program Time(s)-small						
# of processes	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
1	0.347	0.391	0.342	0.347	0.352	0.3558
2	0.437	0.440	0.435	0.453	0.433	0.4396
4	0.587	0.526	0.541	0.506	0.489	0.5298
8	0.607	0.591	0.604	0.578	0.604	0.5968
16	0.888	0.895	0.854	0.868	0.893	0.8796
32	1.388	1.571	1.364	1.545	1.400	1.4536
64	2.701	2.850	2.673	2.634	2.691	2.7098

Program Time(s)-medium						
# of processes	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
1	0.347	0.359	0.358	0.349	0.357	0.354
2	0.459	0.492	0.454	0.444	0.449	0.4596
4	0.524	0.483	0.554	0.505	0.527	0.5186
8	0.610	0.633	0.658	0.643	0.660	0.6408
16	0.876	0.891	0.892	0.881	0.898	0.8876
32	1.399	1.398	1.398	1.415	1.458	1.4136
64	2.574	2.951	2.879	2.984	2.660	2.8096

Program Time(s)-large						
# of processes	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
1	0.382	0.350	0.355	0.403	0.340	0.366
2	0.558	0.459	0.448	0.475	0.454	0.4788
4	0.552	0.543	0.584	0.555	0.545	0.5558
8	0.597	0.569	0.599	0.568	0.560	0.5786
16	0.996	0.843	1.076	0.882	0.882	0.9358
32	1.474	1.456	1.695	1.439	1.359	1.4846
64	2.849	3.044	2.946	3.036	2.787	2.9324

Speedup							
Data	T1	T2	T4	T8	T16	T32	T64
Small	1	0.8094	0.6716	0.5962	0.4045	0.2448	0.1313
Medium	1	0.7702	0.6826	0.5524	0.3988	0.2504	0.1260
Large	1	0.7644	0.6585	0.6326	0.3911	0.2465	0.1248


Time v. # of processes
small
medium
large
3
2.5
2
1.5
1
0.5
0
Time(s)
1
2
4
8
16
32
64
Number of processes


Speedup
small
medium
large
1
0.75
0.5
0.25
0
Speedup
1
2
4
8
16
32
64
Number of processes

Conclusion

The first graph showed that as the number of processes increased for all three curves, the running time increased as well. Which indicates that the number of processes didn't really increase the the speed of the program itself. Perhaps there were several obvious reasons why. In the program, matrix A, vector x and vector b were scattered to all processes by the master process(process 0), which could be an expensive operation if p is large. So of course, as p increases, the amount of communication between the master process and its “workers” increases as well, which increased the overall running time of the program.

In addition, in the do-while loop, a call to `MPI_Allgatherv` is made in each iteration, which collects all the `local_x`’s of each process and puts them in a vector called `curr_x` which holds all the current vales of the unknowns. This call works by having all p processes communicate with each other in order to collect a specific amount of information from each process. So this call could be expensive and will cause the running time to increase if the number of processes was large. This is the reason why in the second graph, as the number of processes increases, all three curves actually don't end up getting a speedup but a “slow down. ” In other words, as p increases, the speedup linearly decreases from 1 to roughly 0.125. That is, the running time for 64 processes is roughly 0.125 that of the running time for the program running with one process(T1).