Nikhil Gosike – ng1449
Parallel Computing
Professor Mohamed Zahran
April 1st 2019

Lab 1 Report

Introduction

This is a program that uses MPI to parallelize and solve and system of equations. Using the given input and set of initial X's, this program attempts to estimate the "unknowns" within an absolute relative error.

Tables

Below are 4 tables showing the execution time and speedup of my program. The first three tables show the time for problem sizes of 100, 1000, and 10000 unknowns. Each is tested on 1, 2, 10, and 20 processes, and each undergoes 5 trials and the time is averaged. The fourth table calculates the speedup defined as S = T_serial/T_parallel. The time is taken as the "real" time outputted when running "time mpirun" in commandline. It should be noted that NYU CIMS account has a 2GB quota limit, and, so I was not able to run a problem size of 100000 for the program.

	100.txt					
	Trials					
# of processes	1	2	3	4	5	**Average**
1	0.386	0.387	0.396	0.403	0.567	**0.4278**
2	0.405	0.387	0.394	0.392	0.394	**0.3944**
10	0.499	0.492	0.503	0.492	0.498	**0.4968**
20	0.647	0.640	0.635	0.653	0.626	**0.6402**

	1000.txt					
	Trials					
# of processes	1	2	3	4	5	**Average**
1	0.831	0.836	0.828	0.830	0.839	**0.8328**
2	0.767	0.771	0.758	0.788	0.751	**0.767**
10	0.786	0.783	0.778	0.776	0.789	**0.7824**
20	0.877	0.899	0.891	0.900	0.893	**0.892**

	10000.txt Trials					
# of processes	1	2	3	4	5	**Average**
1	51.686	53.044	51.788	52.227	51.411	**52.0312**
2	39.902	39.605	40.777	40.571	40.794	**40.3298**
10	31.645	30.954	31.348	30.617	30.952	**31.1032**
20	29.497	30.155	30.633	29.992	30.789	**30.2132**

	Speedup = T_serial/T_parallel Problem size		
# of processes	100	1000	10000
1	1.000	1.000	1.000
2	1.085	1.086	1.290
10	0.861	1.064	1.673
20	0.668	0.934	1.722

Conclusions

Based on the tables above, a number of conclusions can be drawn upon. We can look at these metrics in two ways. First we can keep the problem size fixed and see how the number of processes effects the time and speedup. Looking at 100 and 1000 unknowns, it becomes clear that they are relatively small problem, as they cannot benefit from the parallelization. We see that as the number of processes speedup is insignificantly over one or actually goes below 1. This makes sense, however, as the overhead of process creation, synchronization, communication, and termination are expensive and are not outweighed by the problem size. More specifically the use of MPI_Barrier and MPI_Allgather create heavy overheads. MPI_Barrier essentially creates a synchronization point where all processes have to finish before continuing. MPI_Allgather requires communication among all processes to aggregate vector information.

As we go to 10000 unknowns, we start to see significant speedup well over 1 (however less than 2). The much larger problem size outweighs the overhead and can really take advantage of parallelization. This is further emphasized as we keep the number of processes fixed and increase the problem size. Significant speedup can be seen. With this 100000 unknowns seems feasible to achieve a 2x or higher speedup and should be tested in order to see good speedup.

Another thing to note is some inconsistency with the times. Due to OS stress, I/O variation, non-uniform memory access, etc. we see that sometimes the times significantly increase or decrease. For example, for 100 unknowns and 1 process trial 5 showed a relatively much longer length of time.