Parallel Computing - Lab 1

Vaibhav A Gadodia - vag273

April 1, 2019

1 Table 1: Running Time

Unknowns \ Processes	1	2	10	20
100	0.2708 s.	0.2870 s.	0.3224 s.	0.4098 s.
1000	0.5668 s.	0.5288 s.	0.6016 s.	0.8878 s.
10000	35.0470 s.	27.0288 s.	27.5668 s.	40.8012 s.

2 Table 2: Speedup

Unknowns \ Processes	1	2	10	20
100	1.0000	0.9436	0.8399	0.6608
1000	1.0000	1.0719	0.9422	0.6384
10000	1.0000	1.2967	1.2713	0.8590

3 Analysis

As can be seen in the results tables above, there are certain combinations of the number of unknowns and the number of processes where parallel computing results in a faster program. However, such cases are in the minority and even in these cases, the speedup is not very large.

On performing a time analysis of the underlying code blocks in the program, it turned out that the program was losing a significant chunk of time in the MPI calls as follows:

- The initial MPI calls to initialize the processes resulted in a lot of time lost. As the number of processes increased to 20, this resulted in losing up to 1 whole second.

- The program was losing significant time in the initial MPI collective calls that were used to distribute the input data from process 0 to all other processes.
- A significant amount of time was also lost in the MPI_Allgather call that was being made every iteration to collect the newly calculated unknowns and that were necessary to compute the next iteration of the unknowns. In the case of 20 processes and 10000 unknowns, up to 1 second in these calls.

Furthermore, the time saved in computing the unknowns per process by distributing the work across processes wasn't large enough to offset the lost time due to the overhead of MPI calls.

The fact that the speedup was seen in the three combinations where it was stems from the reasons above.

For the case with 100 unknowns, the performance decreases as the number of processes is increased. This happens because while the time saved in computing the unknowns is there, it is much less than the time lost in the MPI overhead of creating new processes and in communicating via collective calls.

A similar pattern persists in the cases with 1000 and 10000 unknowns. However, in these two cases, there are 3 specs. where a minor amount of speedup is achieved.

The specs. where a speedup is achieved are the ones where the time saved in distributing the work of computing unknowns is big enough to offset the time lost in MPI calls. Thus, it is no surprise that these specs. are the ones where either there are a large number of unknowns (10000) but not too many processes (2 and 10) or there are a medium amount of unknowns (1000) and a small number of processes (2).

In conclusion, the results in the table above and the rough time analysis over the code shows that this program implementation has a significant amount of room to be improved. Some ways to improve the performance might include decreasing the number of collective MPI calls per iteration and improved load balancing.