Multicore Processors - Spring 2019
Programming Assignment 2

Daniel Rivera Ruiz
Department of Computer Science
New York University
drr342@nyu.edu

1 Results

Table 1 shows the average running time in seconds (measured over 5 repetitions using the `time` command) for the linear equation system solver implemented using MPI. The rows represent the number of unknowns in the system and the columns the number of processes (ranks) used in the MPI program. In a similar manner, table 2 shows the speedup for each experiment with respect to its single-process counterpart. All experiments were performed using a 0.001 error rate or 30 iterations (whichever occurred first).

Table 1: Average time in seconds

	Processes				
Unknowns	1	2	10	20	40
10	**0.558**	0.585	1.125	–	–
100	0.530	**0.524**	1.335	2.086	3.927
1,000	1.146	**1.083**	1.417	2.100	3.767
10,000	69.538	41.947	**13.425**	18.243	21.344
100,000	8,016.985	4,203.631	1,030.280	**717.815**	914.058

Table 2: Speedup

	Processes				
Unknowns	1	2	10	20	40
10	**1.00**	0.95	0.50	–	–
100	1.00	**1.01**	0.40	0.25	0.13
1,000	1.00	**1.06**	0.81	0.55	0.30
10,000	1.00	1.66	**5.18**	3.81	3.26
100,000	1.00	1.91	7.78	**11.17**	8.77

2 Discussion

Looking at table 2 we notice that there is barely any speedup for the first three problem sizes, and in most cases there's even a slowdown. The intuition behind this is that the overhead generated by 1) spawning new processes and 2) communication and coordination among them is much higher than the speedup gained by parallelizing the application.
If we take for instance the problem of size 10, using 1 or 2 processes yields very similar results. This means that the gains introduced by the parallelism more or less match the overhead of generating one additional process. However, when we move to 10 processes there is a considerable slowdown: generating 9 additional processes (which won't even have a lot of work to do because of the problem size) plus

coordinating communication among them clearly becomes more dominant in the execution time than the benefits of parallelization. For problem size 100 and 1,000 we observe similar results, with even worse slowdowns for 20 and 40 processes. Only at problem size 10,000 do we start to observe clear speedups for all number of processes.
Some additional observations worth mentioning are:

- Regardless of the number of processes, we observe that the speedup improves as the problem size increases. This means that if we were to run experiments with bigger problem sizes, we would obtain even better results when using more processes. In general if the problem size is big enough, we would expect the speedup to converge to a value upper-bounded by the number of processes (accounting for the overhead generated by communication and synchronization), i.e. 2x for 2 processes, 10x for 10 processes, etc. In fact, the experiment with 2 processes is prety much reaching this upper-bound at the problem size 100,000.
- Given the nature of the problem most of the execution time is spent reading the input file. Taking Amdahl's law into consideration, it was necessary to implement a way of reading the input in parallel in order to obtain considerable speedup. The first implementation attempts where only the computations were parallelized failed miserably, with execution times that were always worse as the number of processes increased. Effectively, there was only one process doing all the heavy work (reading the input file) and in addition we were introducing the overhead of process generation and communication without any considerable profit.