CS3210 Lab 4

Aaron Sng (A0242334N)

22 October 2021

Machines Used

- soctf-pdc-001 - Intel Xeon Silver 4114 with 10 physical cores and 20 hyperthreads
- soctf-pdc-014 - Intel i7-7700 with 4 physical cores and 8 hyperthreads

Exercise 6

To satisfy the requirements of the four MPI configurations, the following methods were used. They are in the following order:

1. A rank file mapping each process to one core was used. The host name is specified.
2. A machine file indicating that four slots are to be used on the i7 node.
3. A machine file indicating that ten slots are to be used on the Xeon node.
4. A machine file indicating that fourteen slots are to be used on both i7 and Xeon node. i7 has four slots, while the Xeon has ten slots.

The current implementation used follows a blockwise data distribution technique. Such an implementation is relatively straightforward to implement, as each processor has its own fixed task. Assuming that each task has an equal task size, and that each processor takes about the same time to complete each task, the blockwise data distribution technique generally helps in reducing starvation of other processors. However, that is generally not the case for most distributed systems. In the provided lab machines, each i7 core will be substantially more faster than one Xeon core. Therefore, the likelihood of processor starvation increases, especially for systems used in the lab. In table 1, the trend becomes apparent. Each i7 processor completes its tasks 64% faster than each Xeon processor. To prevent starvation, the block-cyclic distribution is favoured. In this data distribution technique, a block size of one row is defined. Subsequently, each row is distributed to the next available processor that is free to compute the resultant elements. In fact, the block-cyclic distribution reduce the starvation of different processors. This is illustrated in table 2, where all processors, regardless of processor type, takes about the same time to perform computation. The effect is also illustrated in the wall clock of both programs. On fourteen processors, the blockwise distribution wall clock timing takes 11.43 seconds, while the block-cyclic distribution takes 7.58 seconds.

Rank	0	1	2	3	4	5	6	7	8	9	10	11	12
Processor Type	Xeon	Xeon	Xeon	Xeon	Xeon	Xeon	Xeon	Xeon	Xeon	i7	i7	i7	i7
Execution Time	9.30	9.24	9.27	9.26	9.31	9.29	9.30	9.26	9.39	5.62	5.67	5.67	5.68

Table 1: 2048x2048 Matrix Multiplication using Blockwise Distribution

Rank	0	1	2	3	4	5	6	7	8	9	10	11	12
Processor Type	Xeon	Xeon	Xeon	Xeon	Xeon	Xeon	Xeon	Xeon	Xeon	i7	i7	i7	i7
Execution Time	5.46	5.47	5.47	5.60	5.59	5.60	5.61	5.48	5.58	5.62	5.59	5.62	5.58

Table 2: 2048x2048 Matrix Multiplication using Block-Cyclic Distribution

Exercise 9

Flipping `MPI_Send` and `MPI_Receive` results in a deadlock. The functions `MPI_Send` and `MPI_Receive` generally imply that the process-to-process communication between the master and worker is of blocking-asynchronous type. With both `MPI_Send` and `MPI_Receive` now flipped, both processes will now reach the receive stage. The child process will execute receive stage, but will not have its corresponding send stage from the master process. As a result, the child process is unable to return from the receive operation. Likewise, the master process will execute the receive stage, but will not have its corresponding send stage from the child process. The master process will not return from this and will continue running, resulting in an overall deadlock between the two processes.

Exercise 10

Flipping `MPI_ISend` and `MPI_IReceive`, unlike the example in exercise 9, did not result in a deadlock. The functions `MPI_ISend` and `MPI_IReceive` indicate that the program utilises a non-blocking-synchronous communication protocol. Having such a protocol allows both `MPI_IReceive` and `MPI_ISend` to be returned, and hence both functions to execute. As the communication protocol is synchronous, all processes will terminate upon receipt of its corresponding information. In other words, all processes spawned will eventually send information, and all processes will eventually receive the same information. As a result, the output for both flipping and without flipping will be the same, which was generally observed.