DD2356 Assignment 3
MPI Programming

Yannis Paetzelt - paetzelt@kth.se

May 16, 2021

All code for this assignment can be found here.
(`https://github.com/K2017/DD2356/tree/master/Assignment-III`)

Exercise 1

1. Code on GitHub.
2. Compiled with Cray and no additional compiler flags.
3. MPI code is run with `srun` on a compute node.
4. The number of processes can be changed using the `-n` flag with `srun`. E.g. `srun -n 4` will use 4 processes to run the MPI code.
5. `MPI_Comm_size(MPI_COMM_WORLD, &size);` and `MPI_Comm_rank(MPI_COMM_WORLD, &rank);` are used to retrieve the number of processes and the rank of the calling process respectively.
6. The two most used MPI implementations are MPICH and OpenMPI.

Exercise 2

`MPI_Send` and `MPI_Recv` are blocking. This means they do not return to the caller until certain conditions are met. `MPI_Send` blocks the caller until the send buffer is safe to be (re)used. `MPI_Recv` blocks until the receive buffer contains the intended message (based on tag and source parameters).

Exercise 3

1.

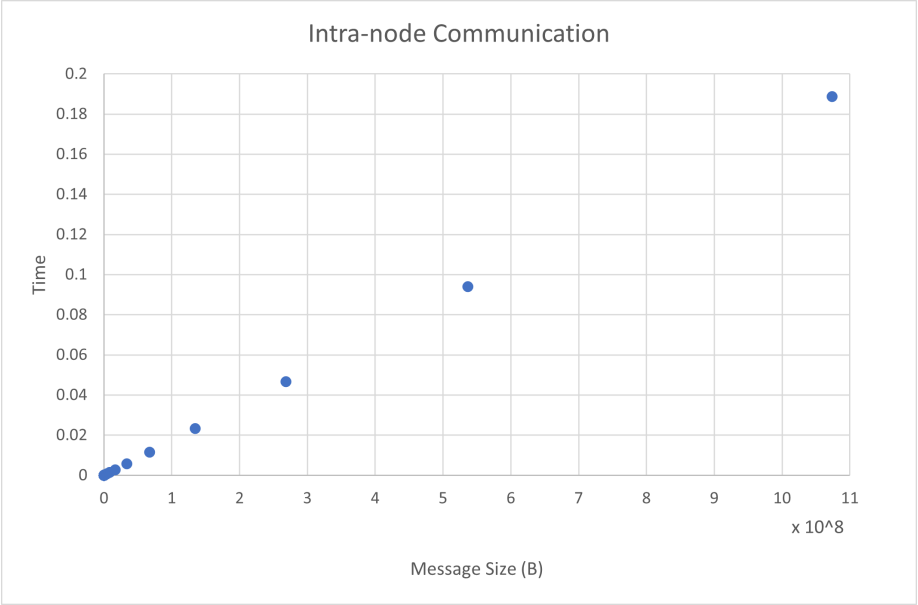

Figure 1: Ping-Pong intra-node results

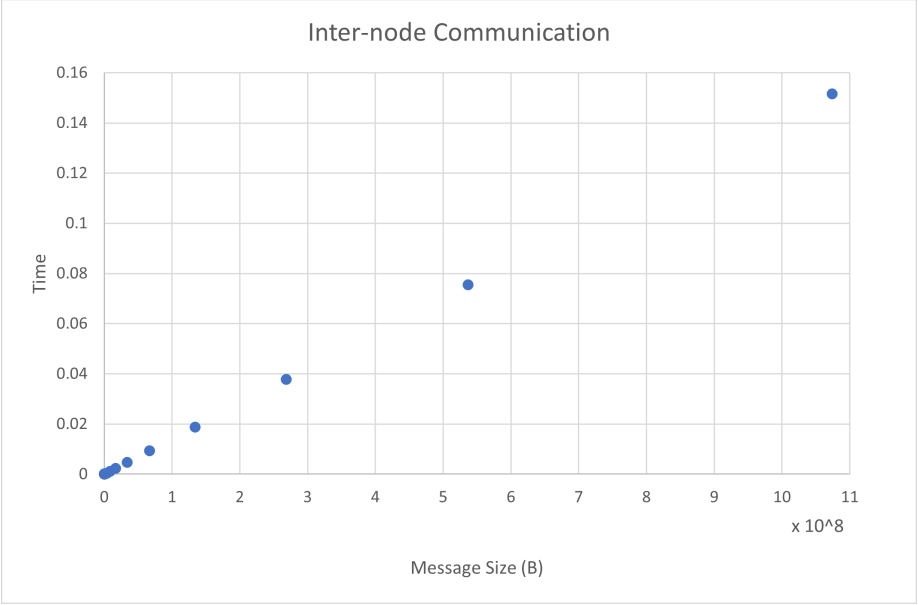

Figure 2: Ping-Pong inter-node results

2. After obtaining negative latencies when computing a line of best fit, even when excluding noisy results, the latencies were set to 1.6μs and 0.7μs for inter and intra-node respectively.

The fit was adjusted with these values in mind, and the bandwidth was calculated to be 7.09 GB/s and 5.7 GB/s for inter and intra-node communication respectively.

3. The modified code creates an `MPI_Win` window for one-sided communication. Performance was measured using `MPI_Get`.

4. Point-to-point communications transfers data directly between two processes' memory address spaces, whereas one-sided communications specifies a section of memory that is shared between processes.

5. Performance was measured in the same way as for the point-to-point version and a model was derived in the same way, with latency fixed for the iter-node measurement due to the appearance of negative latency.

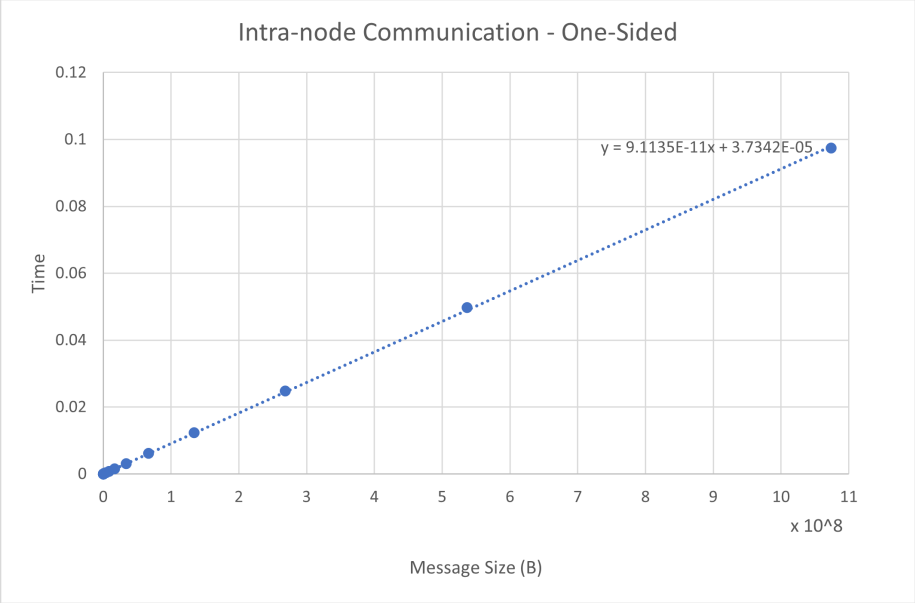

Figure 3: One-sided ping-pong intra-node results

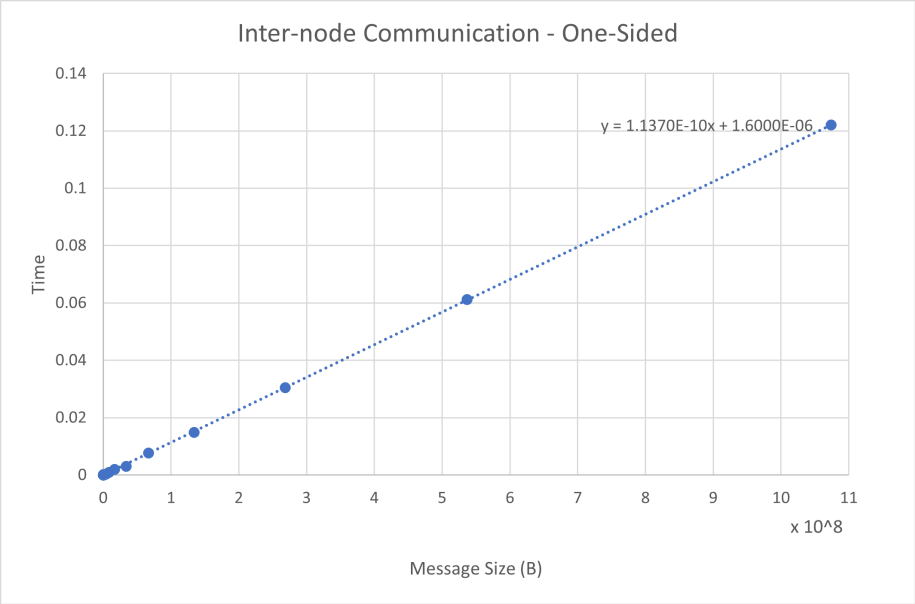

Figure 4: One-sided ping-pong inter-node results

The bandwidths according to each model were 10.97GB/s and 8.8GB/s for one-sided intra and inter-node communication respectively. The modeled latency for intra-node communications was 37.3ms.

6. The postal model is not the best performance model in parallel computing as it does not take into account multicore architectures and the fact that these kinds of systems rarely provide full bandwidth to all processes on a node. Additionally, it does not account for the interface between nodes.

   The authors of the paper propose some modifications to the postal model that takes into account eager/rendezvous thresholding and bandwidth limits into and out of a node:

   $$T = \alpha + kn/\min(R_N, kR_C)$$

   where k is the number of processes, R_N is the injection bandwidth, and R_C is the communication rate an individual process can achieve.

   The latency α is the same as in the previous model.

Exercise 4

4.1 Blocking Linear Reduction

1. See GitHub.

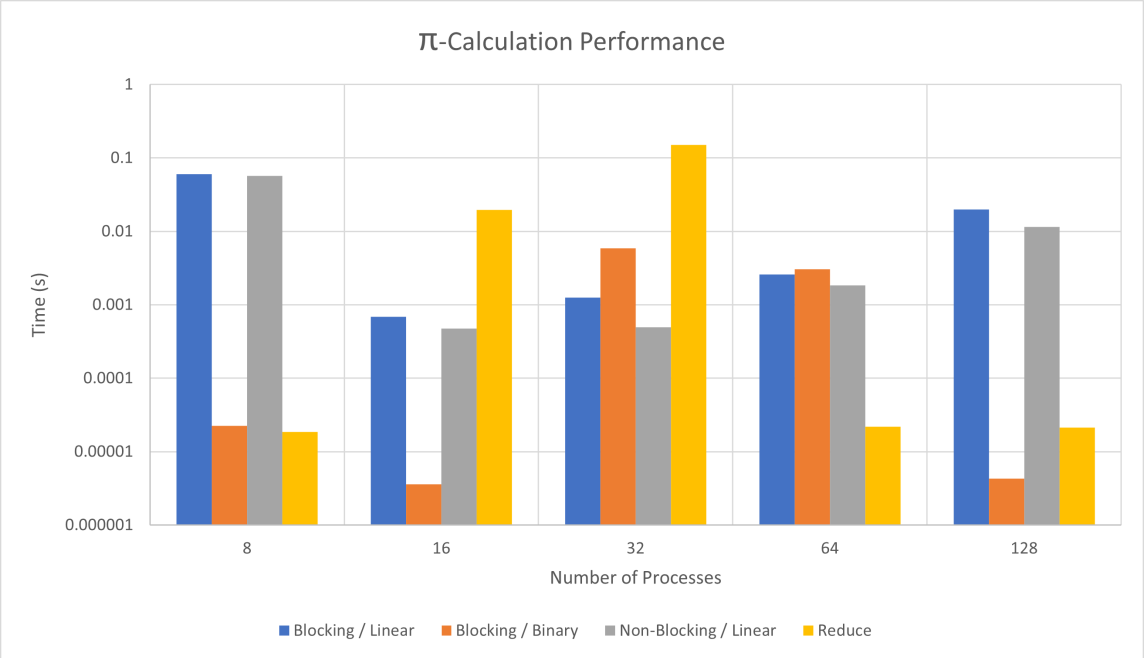

Figure 5: Combined performance measurements for exercises 4.1 to 4.4

2. See Figure 5. The plot is logarithmic on the vertical axis, so as the number of processes increases the execution time increases approximately by a factor of ten. The execution time was measured using the MPI function `MPI_Wtime()`.

3. The model used to compare communication performance is of the form (using intra-node latency and bandwidth):

$$T(n) = 1.6 * 10^{-6} + n * 1.41 * 10^{-10}$$

Plugging in the number of processes for n gives the following table of times:

n	T(n)
8	0.000001607
16	0.000001615
32	0.000001629
64	0.000001658
128	0.000001716

Table 1: Performance model for blocking linear reduction

Compared to the measured results, the model predicts much lower times than measured.

4.2 Blocking Binary Tree Reduction

1. See GitHub.

2. See Figure 5.
3. Using the same model as before, the measured results match the predictions more closely for small number of processes $n = 16$, and many processes $n = 128$, but not very well for other values.
4. For small number of processes, as well as very large number of processes, binary tree reduction performs better than linear reduction. Otherwise, they are comparable.
5. At the limit, binary tree reduction should perform better than linear reduction. Due to limited send and receive buffer space, it is beneficial to limit the number of processes communicating at the same time. With binary reduction, at most half of the processes are communicating at any one time, whereas in linear reduction there is likely a bottleneck due to all the processes sending data at once.

4.3 Non-Blocking Linear Reduction

1. See GitHub.
2. See Figure 5.
3. `MPI_Isend` and `MPI_Irecv` are functions for non-blocking communication. The "I" stands for "Immediate", as in "Immediate return" [to the caller].
4. Non-blocking communication scales similarly to blocking communication, but is faster overall.

4.4 Collective Communication

1. See GitHub.
2. See Figure 5.

Exercise 5

1.

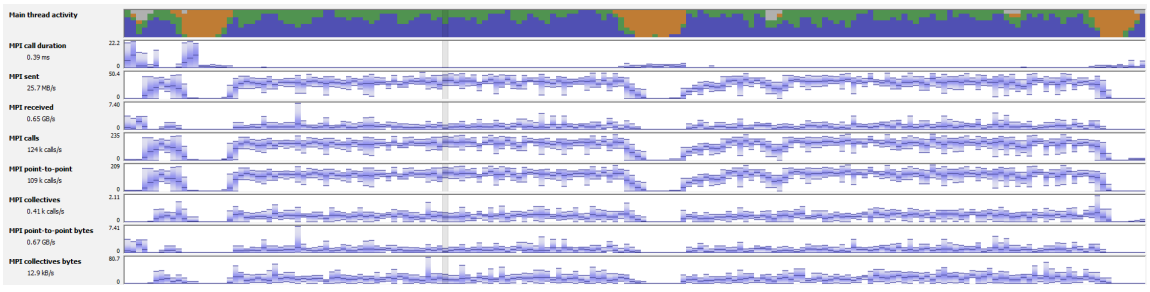

Figure 6: MAP results

2. From the analysis, it seems the `CalculateMoments()` function takes the most amount of time at 33.1% of the total core time. This makes sense as a significant fraction of MPI communication is taking place within, 26.7% of MPI calls.

3. MPI Sent bandwidth was measured to be 25.7 MB/s, and MPI Recieved 0.65 GB/s.