Spring 2023:
Advanced Topics in Numerical Analysis:
High Performance Computing
Assignment 4
Keigo Ando (ka2705)

1. **Greene network test.** Use the pingpong example from class to test the latency and bandwidth on Greene.

   [solution] The nodes on the greene that I ran the program on are `cs482` and `cs483`. They have two Intel Xeon Platinum 8268 CPUs @ 2.90GHz each, which means that each node has 48 compute nodes. For implementation, see `pingpong.c`, which converts the original file to a c file. In my experiments, each node communicated with each other in a single thread, and the latency and bandwidth were as follows

   - `pingpong latency: 9.907620e-04 ms`
   - `pingpong bandwidth: 1.204634e+01 GB/s`

   You can check `pingpong.sbatch` to see how the actual job was done on greene.

2. **MPI ring communication.** Write a distributed memory program that sends an integer in a ring starting from process 0 to 1 to 2 (and so on). The last process sends the message back to process 0. Perform this loop N times, where N is set in the program or on the command line.$\cdots$

   [solution]

   - For the first part I ran the program on the greene with $N = 1000$, 4 nodes and 4 processes per node. You can find my implementation in the file named `int_ring.c`. The nodes on the greene that I ran the program on are `cs301`,`cs302`,`cs303`, and `cs304`. The total timing and estimated latency were as follows.
     - `Result after 1000 loops: 120000`
     - `Total time: 232.102925 s`
     - `Estimated latency: 0.014506 s`
   - For the second part, we communicate a large array of 2MByte in a ring with the same setting. You can find my implementation in the file named `int_ring_block.c`. The nodes on the greene that I ran the program on are the same as the previous experiment.The total timing and estimated bandwidth were as follows.
     - `Total time: 229.641449 s`
     - `Estimated bandwidth: 0.146117 GB/s`

     You can check `int_ring.sbatch` to see how the actual jobs were done on greene for both parts.

3. **Pick one out of the following two:**

   (a) Implement an MPI version of the scan function we implemented in OpenMP. $\cdots$

   (b) Extend the MPI-parallel implementation of the 1D Jacobi smoothing from class to 2D. $\cdots$

   [solution] I choose (b). You can find my implementation in `jacobi.c`.

4. **Pitch your final project.** Summarize your current plan for the final project.···

   [Solution]

   I will be working with Xuijin He on parallel-in-time integration. In general, most parallelization efforts for numerical solutions of partial differential equations have focused on spatial discretization, but in order to develop parallelization methods for faster computation, the parallel-in-time integration method has recently been developed as a parallelization method for temporal discretization. In our project, we deal with a typical parallel-in-time integration method called Parareal, introduced by Lions et al. [1].

   Specifically, we will implement a 1D heat equation based on the Parareal method/algorithm and compare its error and timing with non-parallel implementations. We are considering using MPI and Cuda for the implementation. If progress is made, we may proceed with implementing the 2D case as well.

References

[1] J.-L. Lions, Y. Maday, and G. Turinici. A "parareal" in time discretization of PDE's. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 332:661–668, 2001.