High Performance Computing
Anthony Trubiano
Assignment 6 (due May 13, 2019)

Note: All local CPU computations were performed with an Intel Core i7-8750H Processor with base frequency 2.20 GHz. The maximum main memory bandwidth is 41.8 GB/s. At 16 double precision operations per cycle, the theoretical max flop rate would be about 35.2 GFlops/s. It has 6 cores and can reach 12 threads through hyper-threading.

1. **Project Update** Most of the immersed boundary method code is written. We have a Stokes fluid solver that uses the OpenMP implementation of the parallel FFTW3 library to perform the solves. We have initialization routines and a class for the boundary that uses OpenMP parallelization for the force spreading and velocity interpolation steps. We are still working on the MPI parallel FFTW3 library, then we will run scaling tests.

2. **MPI Parallel 2-D Jacobi Smoother** Using MPI, a 2-D Jacobi smoother was implemented. It is restricted to using $p = 4^j$ processors for integer j, and a total number of points that divides 2^j. This implementation is blocking; each process performs a Jacobi update, then communicates the proper ghost points.

   We run both a strong and weak scaling study on Prince in which we fix the number of iterations to be 500. For the weak scaling study, we increase the number of points and processes such that each process handles a fixed $N_l = 1000$ points. For the strong scaling study, we fix a total number of points $N = 10000$ and increase the number of processes. The timings for both of these studies is shown in Figure 1.

   For the strong scaling, we see close to optimal scaling, as multiplying the number of processes by 4 decreases the time by about a factor of 4 each time. There is slight deviation for 64 processes, probably from communication time. The weak scaling study shows roughly constant timings when each process has the same amount of work. The increase can also be attributed to communication costs.

3. **Parallel Sample Sort** We implement the parallel sample sort algorithm using MPI. We locally sort each bin, determine appropriate values to split the data at, then send each process all the value according to its splitter range. This data is then sorted locally again and output to files. To test this, using 64 cores on Prince, we sort lists of random integers of size $N = 10^4$, 10^5, and 10^6 per processor. The timings are 0.67s, 0.7s, and 1.05s, respectively. We see the performance is much better than the $O(N \log N)$ it would take in serial. For $N \leq 10^5$, the timings are all very similar, so all the effort is likely spent on communication until this point.

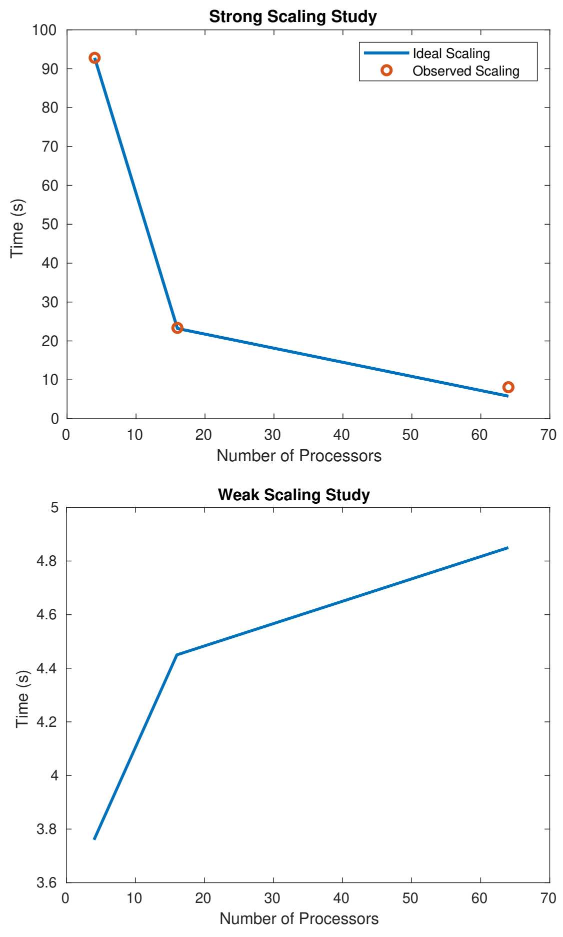

Figure 1: Timings for a strong and weak scaling study for the Jacobi smoother using 500 iterations.