Spring 2019: Advanced Topics in Numerical Analysis: High Performance Computing Assignment 6

Kaizhe Wang (kw2223)

0. **Final project update.**

Project: Parallel computing of Galaxy Correlation Functions		
Week	Work	Who
04/15-04/21	Read papers underlying ideas, learn about the mesh lattice algorithm used here. Done.	Yucheng, Kaizhe
04/22-04/28	Write the serial code. Done.	Yucheng, Kaizhe
04/29-05/05	Parallelize the code with OpenMP. Verify the implementation. Done.	Yucheng, Kaizhe
05/06-05/12	We've successfully ran the codes with MPI.	Kaizhe, Yucheng
05/13-05/19	Test the performance of MPI, write report and prepare for presentation.	Kaizhe, Yucheng

1. **MPI-parallel two-dimensional Jacobi smoother.**
For the weak scaling study, I fixed the $N_l = 100$ and number of iterations to be 10000, increase the number of points as well as the MPI tasks. Here I plot the timing versus number of processors we use, in both regular scale and log-log scale.

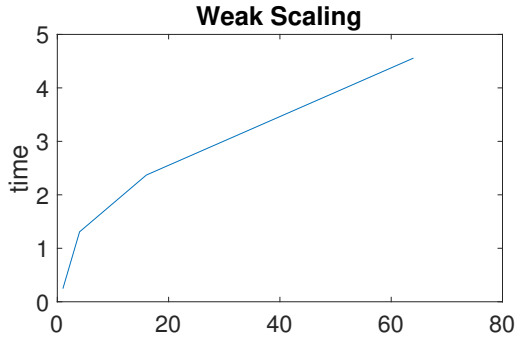

Figure 1: Weak Scaling plot of the timings and number of processors.

For the strong scaling study, choose N_l as large as possible to fit on one processor and keep the problem size unchanged. Here I choose $N = 40000$, number of iterations to be 10, increase the number of MPI task. I still plot the result in regular scale and log-log scale.

2. **Parallel sample sort.** Include the MPI rank in the filename (see the example `file-io.cpp` example file). Run your implementation of the sorting algorithm on at least 64 cores of Prince, and present timings (not including the time to initialize the input array or the time

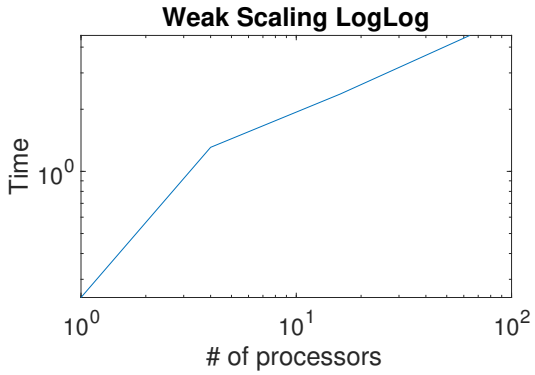

Figure 2: Weak Scaling plot of the timings and number of processors, log-log plot

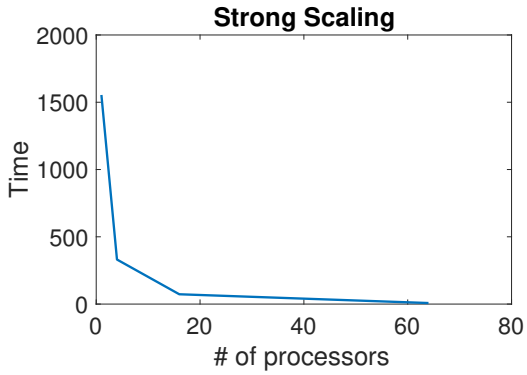

Figure 3: Strong Scaling plot of the timings and number of processors.

to write output to file) depending on the number of elements N to be sorted per processor (report results for $N = 10^4, 10^5$, and 10^6).

I ran the implementation of the sorting algorithm on 64 cores of Prince, and the timings versus the number of elements N to be sorted per processor is:

N_l	Time
10^4	0.971192
10^5	0.974211
10^6	1.28274
10^7	2.64913

Except $N = 10^7$, the timings are almost the same. The number of elements N can be assigned in the command line, say, `mpirun -np 64 ./ssort 100000`.

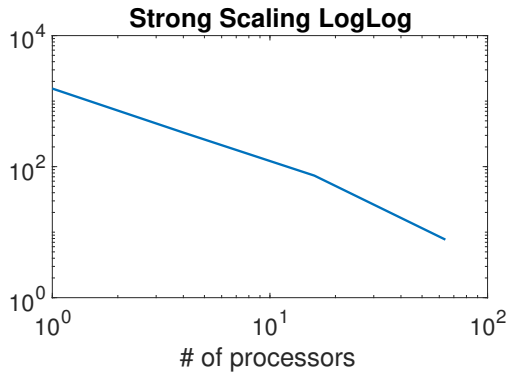

Figure 4: Strong Scaling plot of the timings and number of processors, log-log plot