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

Yongyan Rao, yr780@nyu.edu
(Dated: May 13, 2019)

I. 0. FINAL PROJECT

Project: Implementing FFT		
Week	Work	Who
04/15-04/21	Literature research on FFT and its algorithms	Yongyan Rao
04/22-04/28	Further literature research, implemented a sequential version of FFT, checked the correctness by comparing with GSL library	Yongyan Rao
04/29-05/05	Implemented OpenMP version and naive cuda version of FFT	Yongyan Rao
05/06-05/12	Ran tests on the implementations, and worked on report	Yongyan Rao
05/13-05/19		Yongyan Rao

II. 1. MPI-PARALLEL TWO-DIMENSIONAL JACOBI SMOOTHER

A. Blocking version weak scaling study

The following study was conducted with the following parameters, $lN = 100$, number of iterations 10.

Number of process(es)	1	4	16	64	256
Time (second)	0.000687	0.000895	0.099191	0.064470	3.127207

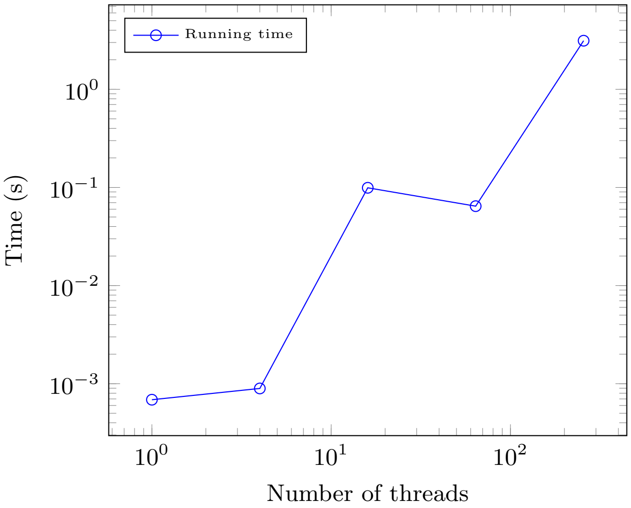

FIG. 1: Weak scaling test

A program is weakly scalable, if when the number of processes/threads increases x times, the program completes an x times larger problem within the same amount of time. Therefore, from the plot above, we can conclude that the Jacobi solver is not weakly scalable.

B. Blocking version strong scaling study

The following study was conducted with the following parameters, $N = \sqrt{10240000} = 3200$, number of iterations 10.

Number of process(es)	1	4	16	64	256
Time (second)	0.673925	0.171769	0.085054	0.117849	3.683604
Speedup	1	3.923437873	7.923495662	5.718546615	0.182952619

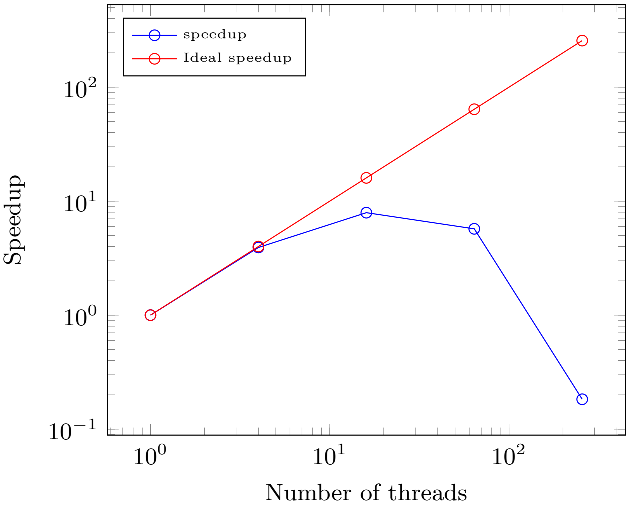

FIG. 2: Strong scaling test

A program is strongly scalable, if when the number of processes/threads increases x times, the program completes a same-size problem x faster, which means the speedup of the program is proportional to the number of processes/threads. Therefore, from the plot above, we can conclude that the Jacobi solver is not strongly scalable. Actually the linearity relation only holds up to four (4) processes.

C. Non-blocking version comparison

The same experiments were conducted using the non-blocking implementation.

Number of process(es)	1	4	16	64	256
Blocking time (second)	0.000687	0.000895	0.099191	0.064470	3.127207
Non-blocking time (second)	0.000673	0.000764	0.020252	0.042427	0.717789

Number of process(es)	1	4	16	64	256
Blocking time (second)	0.673925	0.171769	0.085054	0.117849	3.683604
Non-blocking time (second)	0.624653	0.160637	0.066820	0.057231	0.705940

From the comparison above, we can conclude that the non-blocking version of the program always has better performance than the blocking version. The advance is more significant when the problem size is large.

III. 2. PARALLEL SAMPLE SORT

With setups, `--nodes=8` and `--ntasks-per-node=8`, i.e., totally 64 processes, the running time versus the number of random numbers generated in a process is presented as below.

N	10^4	10^5	10^6
Time (second)	0.802799	0.851473	1.159685