HPC Hw5 (Zhe Chen)

Machine configuration

Results are obtained by running on CIMS's HPC cluster Prince.

Modules environment is

```
3) cuda/8.0.44 7) gcc/6.3.0 8) intel/17.0.1 9) openmpi/intel/1.8.8
```

I am only able to request only 16 nodes and 1GB memory since it's too crowded on Prince.

```
srun --nodes=16 --ntasks-per-node=1 -t2:00:00 --mem=10000 --pty /bin/bash
```

P0: Final Project Update

Date	Members	Tasks
04/29 - 05/03	(Zhe & Guanchun)	Literature search, discuss algorithms and write pseudocode and think about implementation with CUDA
05/03 - 05/10	(Zhe & Guanchun)	Complete psedo-code and serial coding and is now trying switching it to parallel implementation

P1: MPI-Parallel 2D Jacobian smoother

I implemented a MPI version of Jacobian methon in Jacobi-omp.cpp. Number of processors should be $p=4^j$ and number of pts is $N=2^j$ NI.

One should notice that it shoudld be paied attention to communication order of OMP_Send and OMP_Recv to avoid deadlock. Moreover, it's better to gather a array for communication rather than communicate point-by-point between nodes.

Here we fix number of iteration to be 1000 for the convenience of comparison.

Here's makefile for compiling (-O3 flag) and one should run it as,

```
mpirun -n p ./jacobi-mpi N MaxIters
```

1. Weal scaling: First, we fix Nl=100 and increase j.

Results are showed as below. Increase of consuming time shows the cost of communication between nodes.

j	0	1	2
p	1	4	16
N	100	200	400
t/s	0.017996	0.029307	0.087737

2. Strong scaling: Then, We fix number of points to be N=400 and increase nodes to see its scaling with processors.

p	1	4	16
t/s	1.410616	0.152822	0.088982

We got a nealy half scaling as p increase from 4 to 16. However, consuming time for p=1 is much bigger than 4*t(p=4), which shows that it's memery bounded as p=1.

P2: Parallel sample sort.

I implemented sample sort using parallel algorithm in ssort.cpp. Suppose we have P processors, algorithm is as following.

We initialize N uniformly distributed random integers in each processors. A local naive sort is done at first. Then we gathers p-1 splitters from each nodes and select p-1 splitters uniformly from these p(p-1) number and broadcast them to each nodes. Each node using the splitters to get p bins. One should notice that we only need lower_bound function since it's already well-sorted locally. Finally we communicate each bins to its accordingly nodes and write output file locally with filename "output{p}.txt". The results are as following. The scaling of N is good but bad for number of processors p. This is because N is number for each processor. Thus, there're NP number to be sorted in total and cost is paid to communicate between processors.

N;p	4	8	16
10^4	0.022499	0.049597	0.191107
10^5	0.025785	0.060652	0.231230
10^6	0.129566	0.160658	0.319016