HPC23
Homework 3
Letao Chen
lc5187

git repo: https://github.com/LetaoC/HPC23.git

1. **Greene Network Test**
   After run pingpong.cpp on Greene, the latency is 1.651e-06 ms and bandwidth is 6.282591e+05 GB/s

2. **MPI ring communication**

   (a) After 10 iterations using 4 processors, the output is shown below:

```
Process 0 on cs334.hpc.nyu.edu out of 4
Process 2 on cs490.hpc.nyu.edu out of 4
Process 3 on cs491.hpc.nyu.edu out of 4
Process 1 on cs489.hpc.nyu.edu out of 4
Iteration 0: sum = 6
Iteration 1: sum = 12
Iteration 2: sum = 18
Iteration 3: sum = 24
Iteration 4: sum = 30
Iteration 5: sum = 36
Iteration 6: sum = 42
Iteration 7: sum = 48
Iteration 8: sum = 54
Iteration 9: sum = 60
```

   We can tell that each iteration these 4 processors add their rank to the sum, which is $0+1+2+3 = 6$ each iteration.

   (b) After 100000 iterations using 4 processors on Greene, the approximation latency and total time used are:

```
Process 0 on cs476.hpc.nyu.edu out of 4
Process 2 on cs478.hpc.nyu.edu out of 4
Process 1 on cs477.hpc.nyu.edu out of 4
Process 3 on cs479.hpc.nyu.edu out of 4
latency: 5.424488e-03 ms
Total time: 0.542449 s
```

   (c) Passing large array is coded in `array_ring.cpp`
   After 100000 iterations using 4 processors on Greene, the approximate bandwidth and the total time used are:

```
Process 1 on cs340.hpc.nyu.edu out of 4
Process 0 on cs339.hpc.nyu.edu out of 4
Process 3 on cs342.hpc.nyu.edu out of 4
Process 2 on cs341.hpc.nyu.edu out of 4
Bandwidth: 3.183351e+00 GB/s
Total time: 0.483644 s
```

3. **MPI Scan**
   In order to visualize more clearly, here is the result when size of array = 10:

```
ind 0 is 358
ind 1 is 1368
ind 2 is 7974
ind 3 is 366
ind 4 is 6338
ind 5 is 989
ind 6 is 8206
ind 7 is 1080
ind 8 is 1567
ind 9 is 6774
scan result 0 is 358
scan result 1 is 1726
scan result 2 is 9700
scan result 3 is 10066
scan result 4 is 16404
scan result 5 is 17393
scan result 6 is 25599
scan result 7 is 26679
scan result 8 is 28246
scan result 9 is 35020
mpi scan 0 is 358
mpi scan 1 is 1726
mpi scan 2 is 9700
mpi scan 3 is 10066
mpi scan 4 is 16404
mpi scan 5 is 17393
mpi scan 6 is 25599
mpi scan 7 is 26679
mpi scan 8 is 28246
mpi scan 9 is 35020
error = 0
```

We can see that the mpi scan result is the same as traditional scan result.
As I increase the length of the array, for example, N = 100000, the error is still 0.
The time needed to run on Greene when we use 16 processors and N = 100000:

```
Process 0 on cs479.hpc.nyu.edu out of 16
Process 1 on cs479.hpc.nyu.edu out of 16
Process 2 on cs479.hpc.nyu.edu out of 16
Process 3 on cs479.hpc.nyu.edu out of 16
Process 10 on cs490.hpc.nyu.edu out of 16
Process 5 on cs480.hpc.nyu.edu out of 16
Process 9 on cs490.hpc.nyu.edu out of 16
Process 6 on cs480.hpc.nyu.edu out of 16
Process 13 on cs491.hpc.nyu.edu out of 16
Process 8 on cs490.hpc.nyu.edu out of 16
Process 4 on cs480.hpc.nyu.edu out of 16
Process 12 on cs491.hpc.nyu.edu out of 16
Process 7 on cs480.hpc.nyu.edu out of 16
Process 11 on cs490.hpc.nyu.edu out of 16
Process 15 on cs491.hpc.nyu.edu out of 16
Process 14 on cs491.hpc.nyu.edu out of 16
Total time = 1
```

4. **Final Project with Weijian Feng on Parallel Sudoku**
   We plan to use OpenMP and CUDA to solve high-level Sudoku problems. First, we will write a correct sequential version, and then use it as a baseline to compare the performance improvements brought by OpenMP and CUDA parallelization, and explore both strong scalability and weak scalability.