CS542200 Parallel Programming
HW1: Odd-Even Sort

Due: October 26, 2014

1 Goal

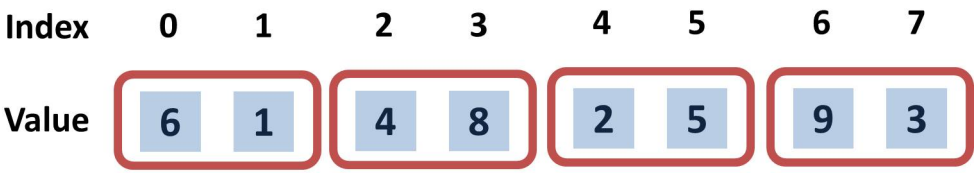
This assignment helps you to get familiar with MPI by implementing odd-even sort. Besides, in order to measure the performance and scalability of your program, experiments are required. Finally, we encourage you to optimize your program by exploring different parallel strategies for bonus points.

2 Problem Description

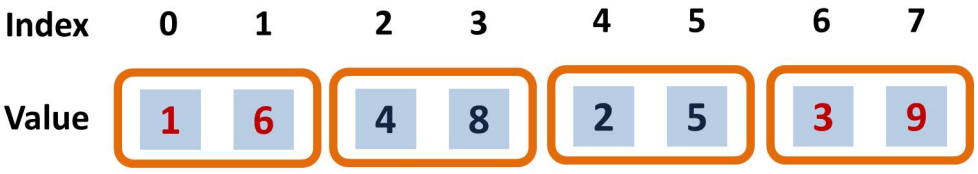
In this assignment, you will implement odd-even sort algorithm using MPI library. Odd-even sort contains two phases, odd-phase and even-phase. In even-phase, it makes adjacent elements be (even, odd)-indexed pairs and elements of a pair are switched if the first is greater than the second. Similarly, it repeats this for (odd, even)-indexed pairs in odd-phase. Odd-even sort runs these two phases alternatively until the list is sorted.

The execution process of odd-even sort is described step by step as below:

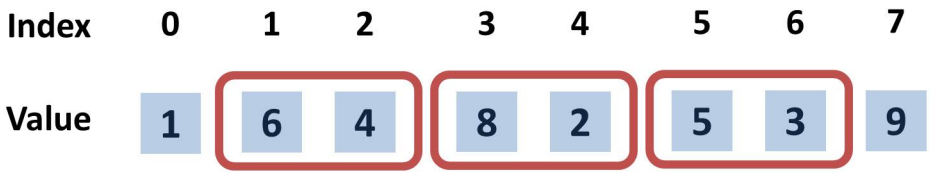
1. **[Even-phase]** Make (even, odd)-indexed pairs of adjacent elements.



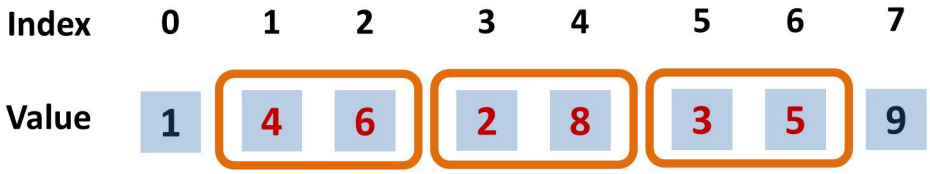
2. **[Even-phase]** Compare items in a pair each time, put smaller one to left side, and put bigger one to right side.



3. [**Odd-phase**] Make (odd, even)-indexed pairs of adjacent elements.



4. [**Odd-phase**] Do the same thing as step 2.



5. Run odd-phase and even-phase alternatively until **no swap-work happens** in both odd-phase and even-phase.

3 Input/Output Format

1. Your program has to be able to **read file**, and generate output in another file.
2. Your program accepts three input parameters. First one is a single integer $\boldsymbol{n}$, the size of list. The second one is the input file name and last is the output file name. Make sure users can **assign test cases through command line**, and check specific output file. For instance :

```
$ mpirun ./HW1_103065566 10 infile outfile
```

3. The test case contains $\boldsymbol{n}$ positive 32-bit integers. You have to read data by **MPI-IO API**.
4. For output file, list the sorted values from test cases, and output them by **MPI-IO API**, too.

4 Working items

You must implement two versions of odd-even sort with the given restriction. Although total execution steps will be bounded, your program should detect whether the list is sorted or not. Your program should stop if no swap-work happens in both two phases. If you are not sure whether your implementation follows the rules, please discuss with TA for approval.

1. Basic odd-even sort implementation
   a) Your algorithm is strictly limited to odd-even sort. Therefore, even when you want to sort multiple elements within the memory of a single MPI task, you still have to follow the odd-even sort algorithm. In other words, a value can only be swapped between its adjacent items in each operation.
2. Advanced odd-even sort implementation
   a) The only restriction is that each MPI task can only send messages to its neighbors during the odd-even sort process. The number of elements sent in each message can also be arbitrary.
   b) The advanced version should achieve better performance than basic version.
3. Report
   a) **Title, name, student ID**
   b) **Implementation**
   Briefly describe your implementation in diagrams, figures or sentences.
   c) **Experiment & analysis**
   What and why you do this experiment? the result of your experiments, and **try to explain them**. (put some figures and charts)
      i. **System Spec**
      If you run experiment on your own machines, please attaches CPU, RAM, disk and network (ethernet/infiniband) information of your system.
      ii. **Strong scalability & Time distribution**
      Observing strong scalability of two implementations. Also, you must run them in single-core and multi-core distribution to see the overhead of communication.
      Therefore, you must plot at least 4 figures **{multi-nodes, single-node} x {basic, advanced}**.
      Moreover, analyzing the time for computing, communication and I/O of your program. You should explain how you measure these time intervals in your program, and compare the time distribution under different distribution.
      You can refer Figure(1) and Figure(2) as examples.
      iii. **Speedup factor**
      You can refer Figure(3) as example.
      iv. **Performance of different I/O ways**
      You can use different data size to observe the trend of performance.
      You can refer Figure(4) as example.
      v. **Compare two implementation**
      Compare the performance of your basic and advanced implementation, and use some plots to explain why the advanced version can achieve better performance.
      vi. **Others**
      Additional performance analysis plots and studies. The more, the better.
   d) **Experience/Conclusion**

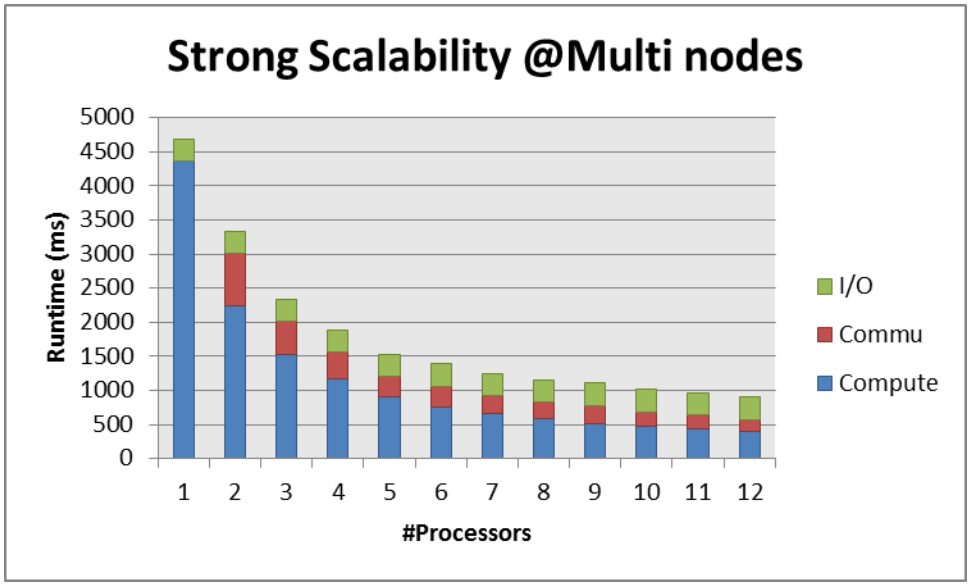


Figure 1: Advanced version

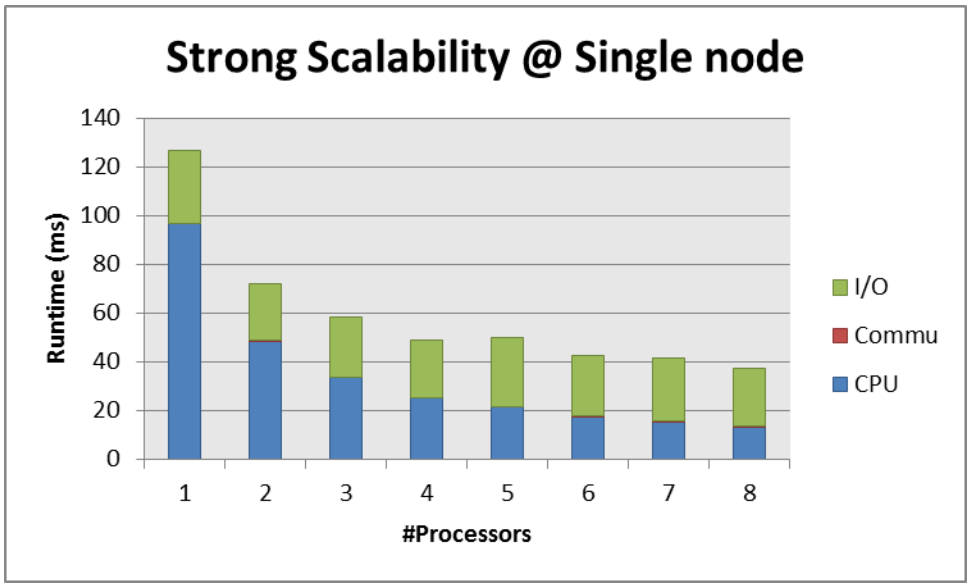


Figure 2: Basic Version

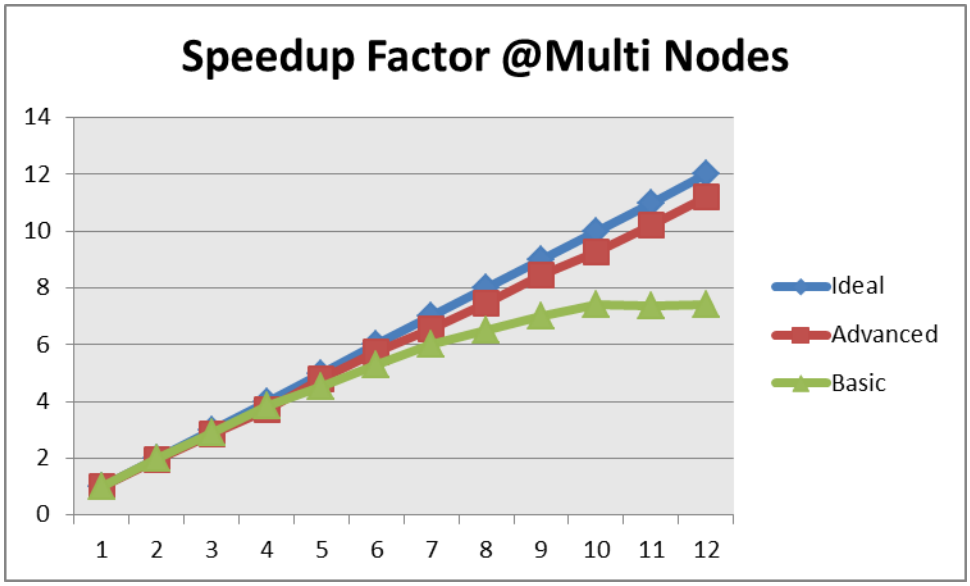


Figure 3: Speedup factor @Multi-Nodes

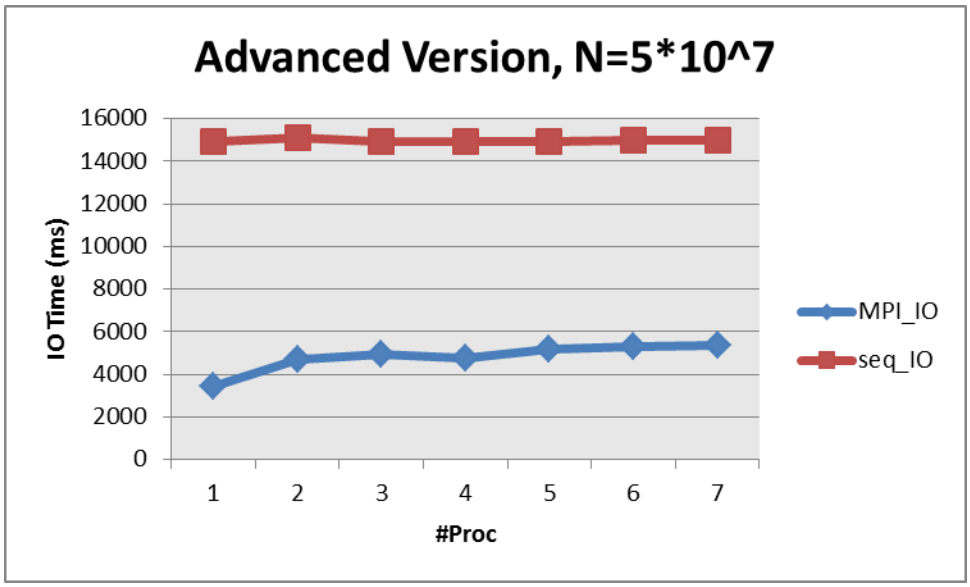


Figure 4: Data size N = 5 * 10^7

5 Grading

1. **Correctness** (50%)

   a) **[5%]** Basic version can solve the problem where the number of input items is the same as the number of MPI tasks.

   b) **[10%]** Basic version can solve the problem where the number of input items is multiplication of the number of MPI tasks.

   c) **[15%]** Basic version can solve the problem where the number of input items can be arbitrary without any restriction, and can even be smaller than the number of MPI tasks.

   d) **[20%]** Advanced version can also solve arbitrary input problem size.

   It will be judged by released testcases in ***shared/HW1*** directory.

2. **Report** (30%)
   It is based on your evaluation results, discussion and writing.
   If you want to get more points, design as more experiments as you can. For instance, implement the static version (fixed steps) and compare the performance between static and dynamic version.

3. **Demo** (20%)

6 Reminder

1. Please compress your codes and report in the file named **HW1_{Student-ID}.zip** which contains:

   a) HW1_{Student-ID}_basic.c (or *.cpp)

   b) HW1_{Student-ID}_advanced.c (or *.cpp)

   c) HW1_{Student-ID}_report.pdf

   And upload it to iLMS before 10/26(sun) 23:59:59

2. Because we have limited machines for you guys for tuning. Please **start your work ASAP** and do not leave it until the last day!

3. Late submission penalty policy please refer to syllabus.

4. Asking questions on iLMS or through e-mail is also welcome!