CS542200 Parallel Programming
Homework 1: Odd-Even Sort

Due: October 25, 2015

1 GOAL

This assignment helps you 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 parallelizing strategies for bonus points.

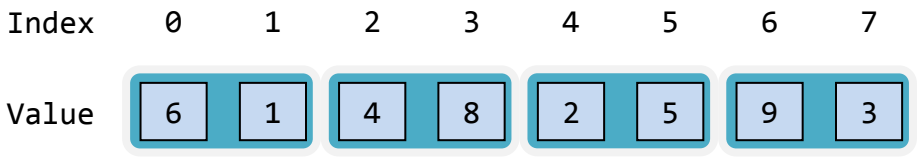
2 PROBLEM DESCRIPTION

In this assignment, you are required to implement odd-even sort algorithm using MPI Library. Odd-even sort is a comparison sort which consists of two main phases: *even-phase* and *odd-phase*.

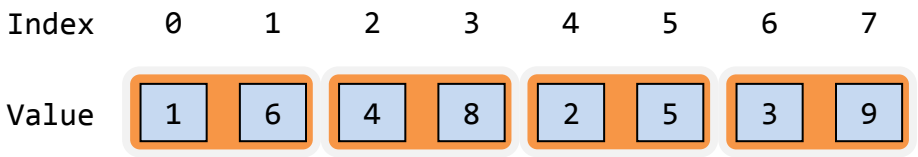
In even-phase, all even/odd indexed pairs of adjacent elements are compared. If a pair is in the wrong order, the elements are switched. Similarly, the same process repeats for odd/even indexed pairs in odd-phase. The odd-even sort algorithm works by alternating these two phases until the list is completely sorted.

In order for you to understand this algorithm better, the execution flow of odd-even sort is illustrated step by step as below: (We are sorting the list into ascending order in this case)

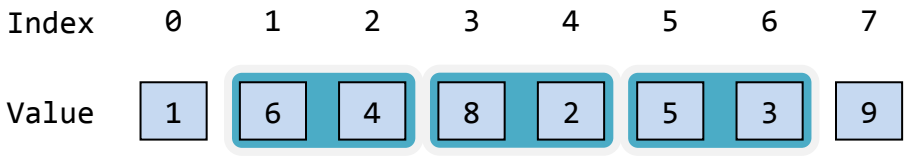
1. [Even-phase] even/odd indexed adjacent elements are grouped into pairs.



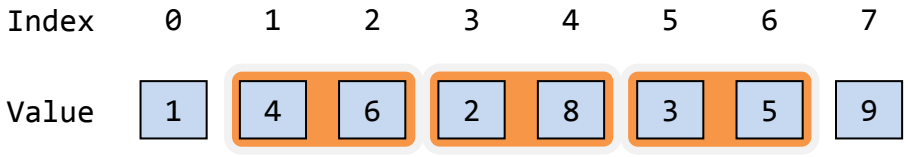
2. [Even-phase] elements in a pair are switched if they are in the wrong order.



3. [Odd-phase] odd/even indexed adjacent elements are grouped into pairs.



4. [Odd-phase] elements in a pair are switched if they are in the wrong order.



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

3 INPUT / OUTPUT FORMAT

1. Your program are required to read an input file, and generate output in another file.
2. Your program accepts 3 input parameters. They are:

   i、 (Integer) the size of the list ***n*** ($0 \leq n \leq 2147483647$)

   ii、 (String) the input file name

   iii、 (String) the output file name

   Make sure users can assign test cases through command line. For instance:

```
[s104012345@pp01 ~]# mpirun ./HW1_104012345 1000 in_file out_file
```

3. The test case consists of ***n*** 32-bit signed integers in binary format. You have to read it using **MPI-IO API**.
4. The output file should list the ***n*** 32-bit signed integers from the input file in ascending order, and output them by utilizing **MPI-IO API**, too.

4 WORKING ITEMS

Stopping Criteria: Although the number of iterations is bounded by the length of the list, your program should be able to detect whether the list is sorted or not and *terminate* after the condition is detected. In other words, your program should stop after **no swap-work happens** in both odd-phase and even-phase.

You are required to implement 2 versions of odd-even sort under the given restriction

If you are not sure whether your implementation follows the rules, please discuss with TA for approval.

1. Basic odd-even sort implementation
   - Your program is strictly limited to odd-even sort. Each element can only be swapped with its adjacent elements in each operation.
   - In other words, take the entire list of numbers and apply classic odd-even sort directly in parallel.
2. Advanced odd-even sort implementation
   - The only restriction is that each MPI process can only send messages to its neighbor processes. The number of elements sent in each message can also be arbitrary.
   - For instance, MPI process with rank 6 can only send messages to process with rank 4 and process with rank 5.
   - You are free to use any sorting algorithm within an MPI process.
   - Advanced version should achieve better performance than basic version.
3. Report
   - **Title, name, student ID**
   - **Implementation**
     Briefly describe your implementation in diagrams, figures, sentences.
   - **Experiment & Analysis**
     How and why you do these experiments? The result of your experiments, and try to explain them. (put some figures and charts)
     - **i、 System Spec**
       If you run your experiments on your own machine, please attach CPU, RAM, disk and network (Ethernet / Infiniband) information of the system.

ii、 Strong Scalability & Time Distribution

Observe strong scalability of the two implementations. Also, you should run them in single-node and multi-node MPI process layout to see the overhead of network communication.
Therefore, you must plot at least 4 figures:
{multi-core, single-core} × {basic, advanced}

Moreover, analyze the time spent in computing, communication, I/O of your program. You should explain how you measure these time in your program, and compare the time distribution under different MPI process layout.

You can refer to Figure(1) and Figure(2) as examples.

iii、 Speedup Factor

You can refer to Figure(3) as an example.

iv、 Performance of different I/O ways

You can use different ways to perform IO, such as sequential I/O and MPI-IO, with different input sizes.

You can refer to Figure(4) as an example.

v、 Compare two implementations

Compare the performance your basic and advanced implementations. Try to use some plots to explain why the advanced version can achieve better performance.

vi、 Others

Additional plots (with explanation) and studies. The more, the better.

- **Experience / Conclusion**

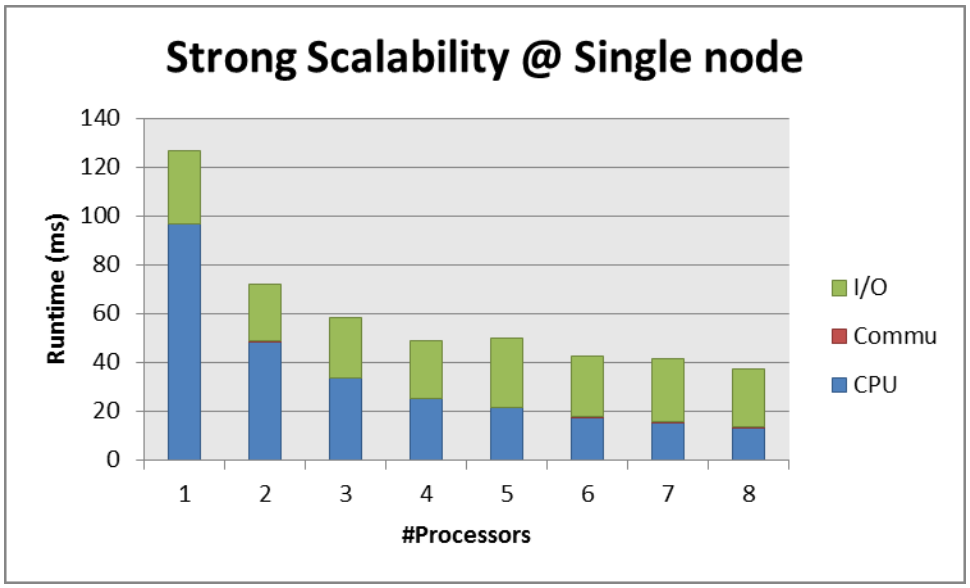


Figure 1: Strong scalability on multi-nodes

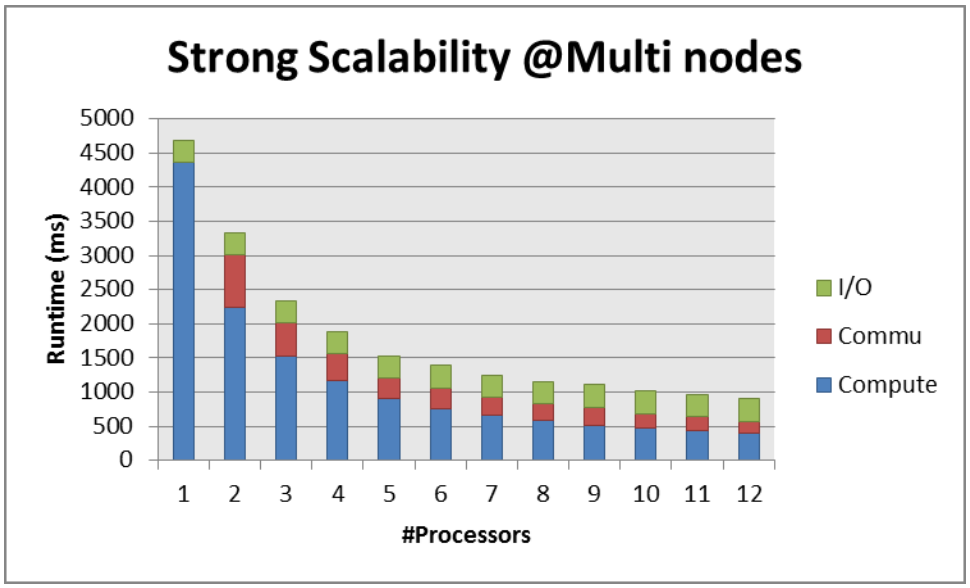


Figure 2: Strong Scalability on single node

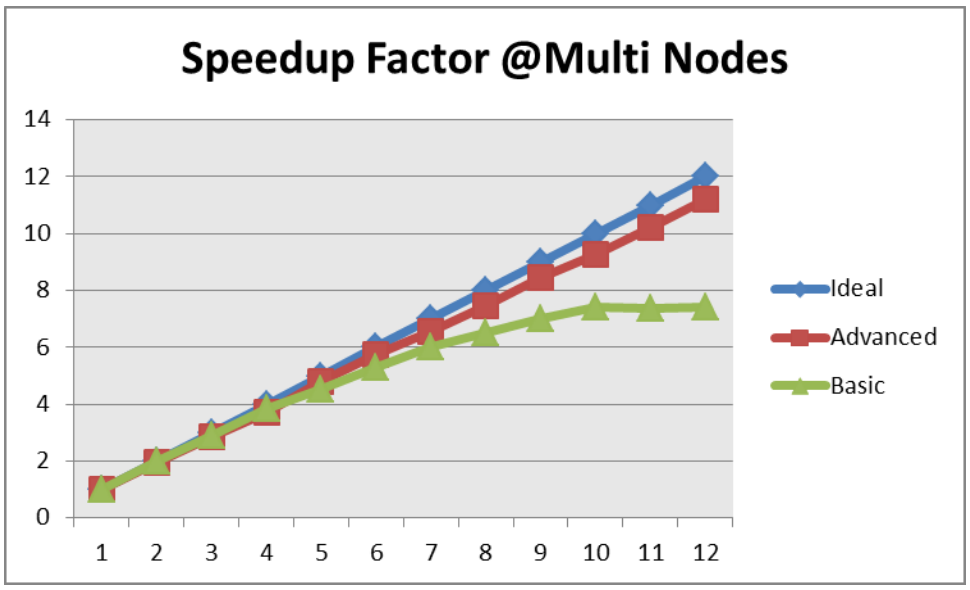


Figure 3: Speedup factor

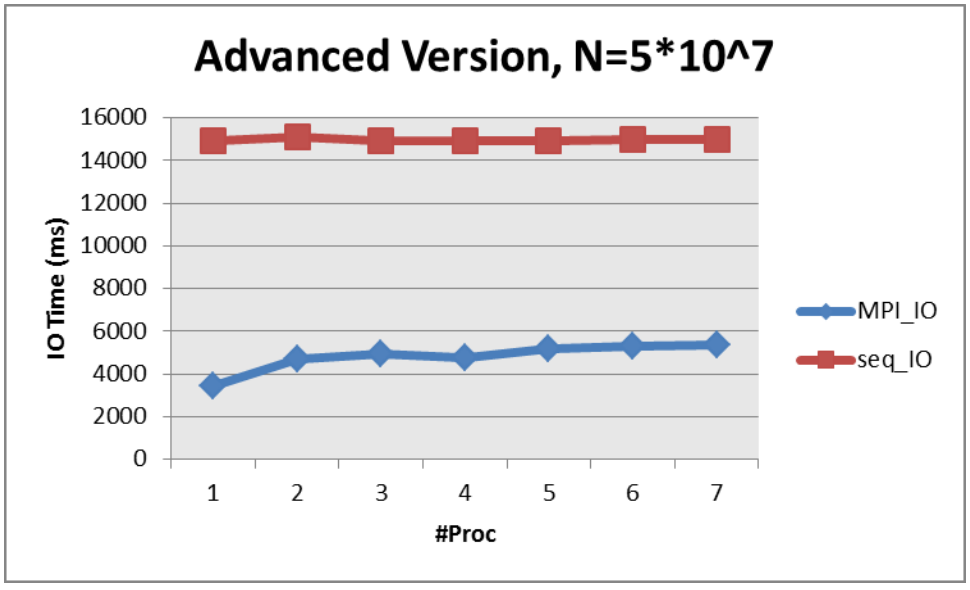


Figure 4: Performance of different I/O ways

5 GRADING

1. **Correctness** (50%)

   i、 Basic version is correct when the number of input items is the same as the number of MPI processes. [5%]

   ii、 Basic version is correct when the number of input items can be divided by the number of MPI processes. [10%]

   iii、 Basic version is correct when the number of input items can be arbitrary without any restriction, which can even be less than the number of processes. [15%]

   iv、 Advanced version is correct with arbitrary input problem size, without any restriction. [20%]

2. **Report** (30%)

   Grading 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, you can implement the static version (with fixed number of phases) and compare the performance between static and dynamic version.

3. **Demo** (20%)

6 REMINDER

1. Please package your codes and report in a file named **HW1_{student-ID}.zip** which contains:

   i、 HW1_{student-ID}_basic.c (or .cpp)

   ii、 HW1_{student-ID}_advanced.c (or .cpp)

   iii、 HW1_{student-ID}_report.pdf

   And upload to iLMS before **10/25(Sun) 23:59**

2. Since we have limited resources for you guys to use, please start your work ASAP. Do not leave it until the last day!

3. Late submission penalty policy please refer to the course syllabus.

4. **Do NOT try to abuse the computing nodes by ssh to them directly**. If we ever find you doing that, you will get 0 point for the homework!

5. Asking questions through iLMS or email are welcomed!