CS542200 Parallel Programming
Homework 1: Odd-Even Sort

Due: October 23, 2016

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.

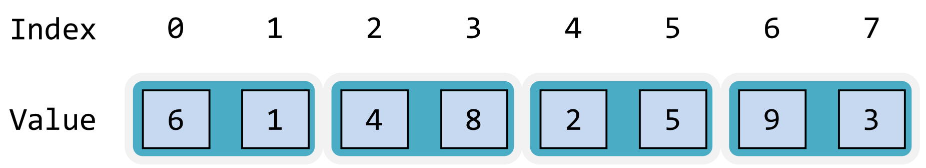
2 Problem Description

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*.

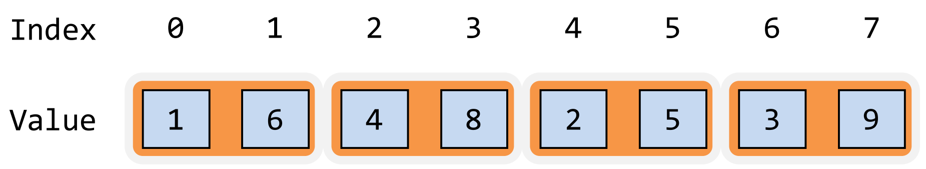
In even-phase, all even/odd indexed pairs of adjacent elements are compared. If a pair is in the wrong order, the elements are switched. Similarly, the same process repeats for odd/even indexed pairs in odd-phase. The odd-even sort algorithm works by alternating these two phases until the list is completely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

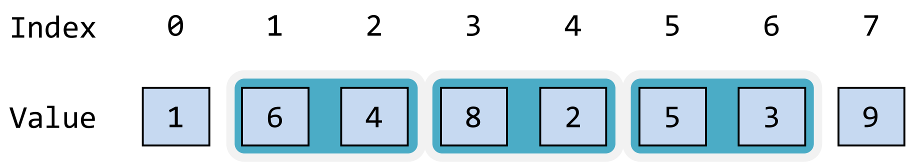
1. [Even-phase] even/odd indexed adjacent elements are grouped into pairs.



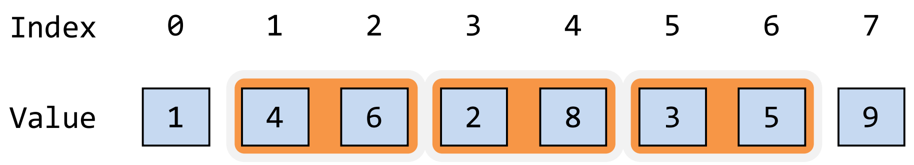
2. [Even-phase] elements in a pair are switched if they are in the wrong order.



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 programs are required to read an input file, and generate output in another file.
2. Your program accepts 3 input parameters, separated by space. 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 input file lists ***n*** 32-bit signed integers in binary format. Please refer to the sample input files.
4. The output file lists the ***n*** 32-bit signed integers from the input file in ascending order. Please refer to the sample output files.

4 WORKING ITEMS

1. **Problem assignments:** You are required to implement 2 versions of odd-even sort under the given restrictions.

   - Basic odd-even sort implementation
     - Your program is strictly limited to odd-even sort between elements. That means each element **can only be swapped with its adjacent elements** in each operation like showing by the step-by-step examples above.
     - Your goal is to make sure the correctness of your program, and the flexibility of your program, so that the program can handle arbitrary number of input values using arbitrary number of cores. **Details in grading section below.**
   - Advanced odd-even sort implementation
     - Only the communication pattern between processes is restricted, so that each MPI process can only send messages to its neighbor processes. However, there is no restriction on the content inside a message or the number of elements sent in the message. There is no restriction on what operations are performed by a process in an even-odd swapping phase.
     - For instance, MPI process with rank 6 can only send messages to process with rank 5 and process with rank 7. But the message can contain all the numbers from rank 6 MPI process at once.
     - You are also free to use any sorting algorithm within an MPI process.
     - Your goal is to optimize the performance, and reduce the total execution time.

2. **Requirements & Reminders**:

   - 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.
   - **Must use MPI-IO** to do file input and output.
   - **Must follow the format of input/output file and execution command line.**
   - If you are not sure whether your implementation follows the rules, please discuss with TA for approval.

5 REPORT

Report must contain the following contents, and you can add more as you like.

1. **Title, name, student ID**

2. **Implementation**
   Briefly describe your implementation in diagrams, figures, sentences, especially in the following aspects:

   ✓ How do you measure computing time, communication time and IO time? What functions you use? Why?

   ✓ How you deal with the condition of the number of input item and the number of process are arbitrary?

   ✓ How do you sort in the advanced version?

   ✓ Other efforts you've made in your program

3. **Experiment & Analysis**
   Explain how and why you do these experiments? Explain how you collect those measurements? Show the result of your experiments in plots, and **explain your observations**.

   You are recommended to generate your own test case. Make sure your experiment results are accurate and meaningful. (e.g. running time are long enough)

   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.

4. **Experience / Conclusion**
It could include these following aspects:

- ✓ Your conclusion of this assignment.
- ✓ What have you learned from this assignment?
- ✓ What difficulty did you encounter in this assignment?
- ✓ If you have any feedback, please write it here. Such as comments for improving the spec of this assignment, etc .

Please explain your graph. Do not just put the graphs without any explanations!

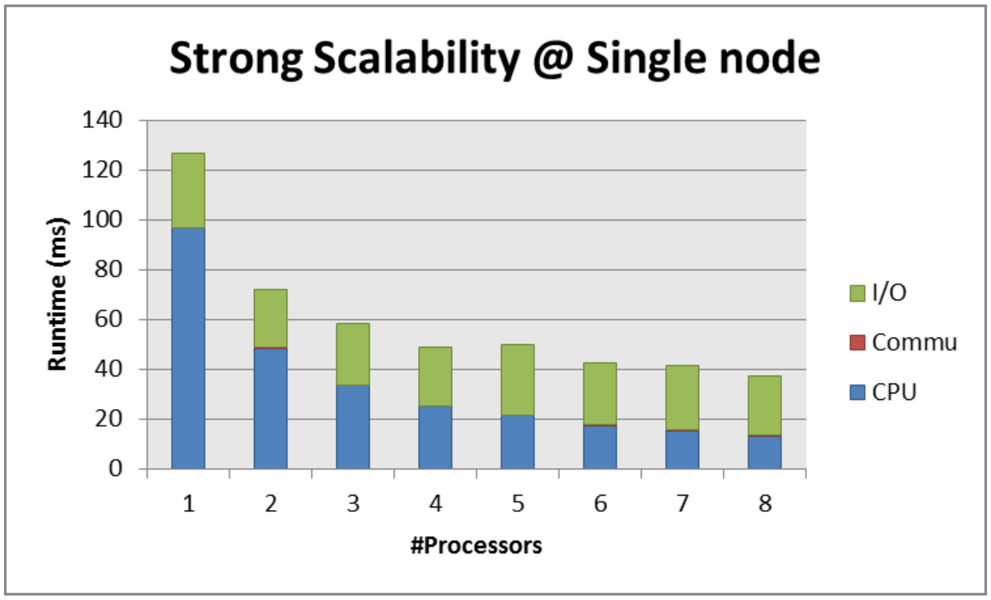


Figure 1: Strong scalability on single-node

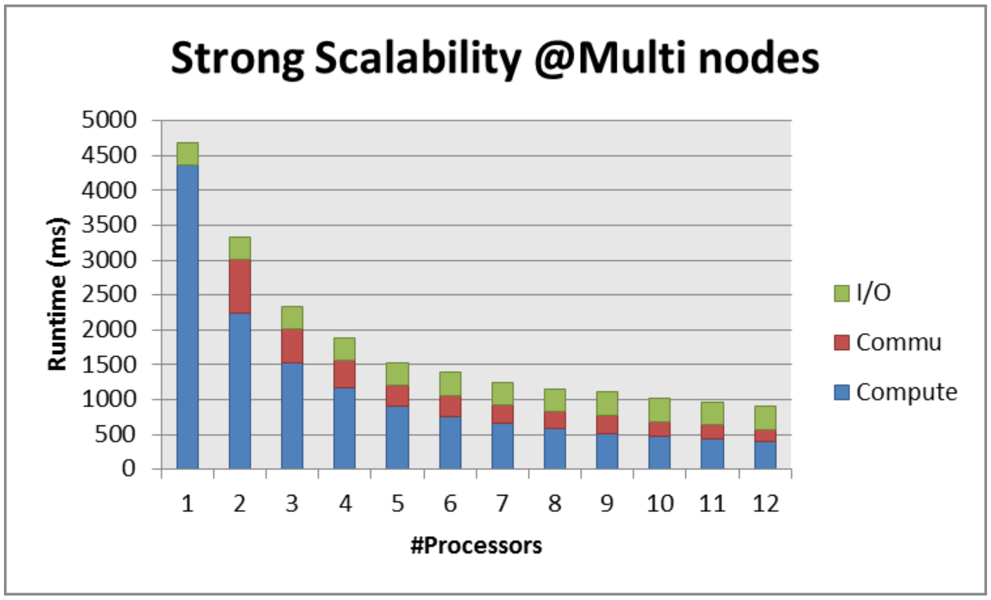


Figure 2: Strong Scalability on multi-node

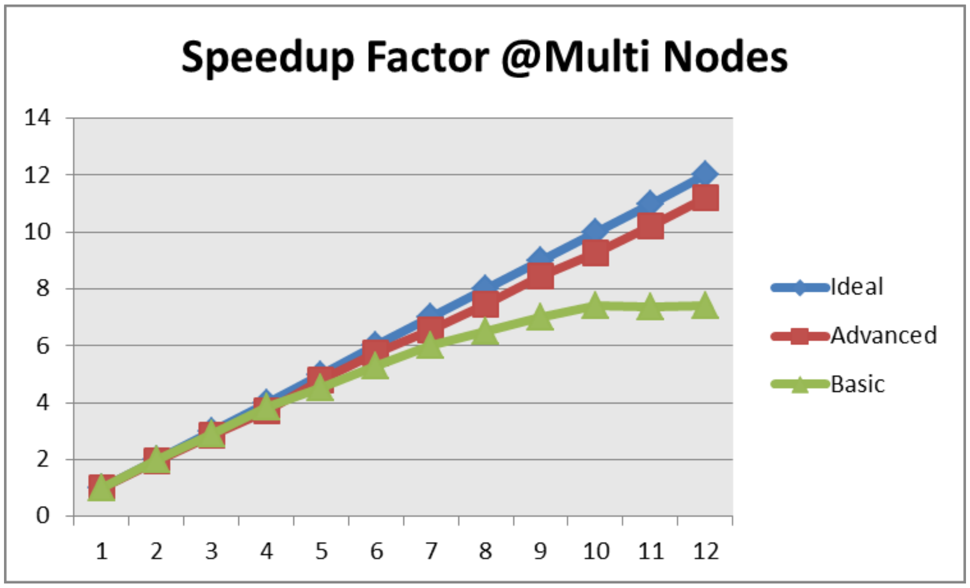


Figure 3: Speedup factor

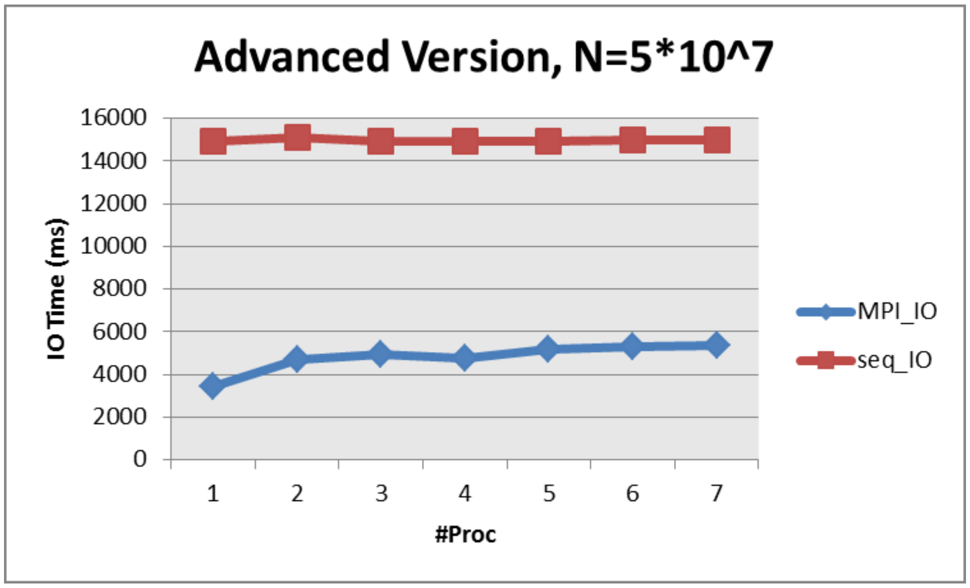


Figure 4: Performance of different I/O ways

6 GRADING

1. **Correctness** (45%)

    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. [15%]

2. **Performance** (10%)

    i、 Based on how fast your advanced version can run.

    ii、 Based on how good the scalability you can achieve.

3. **Report** (25%)

    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.

4. **Demo** (20%)

    Demo will mainly focus on the following aspect, make sure you are familiar with these:

    i、 Explain your figures in the report.

    ii、 Explain your implementation.

    iii、 Why and how you use MPI-IO.

    iv、 Your extra efforts. (why do you deserve more bonus points?)

7 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 **10/23(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. **0 will be given to cheater** (copying code), but discussion on code is encouraged.

6. Asking questions through iLMS is welcomed!