CS542200 Parallel Programming
Homework 1: Odd-Even Sort

Due: October 15, 2017

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.

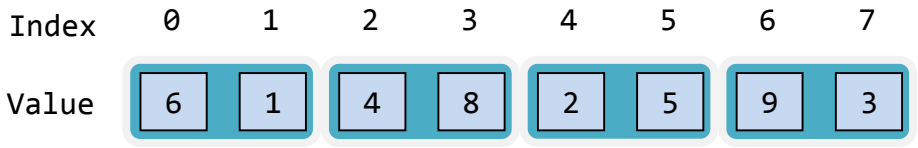
2 PROBLEM DESCRIPTION

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*.

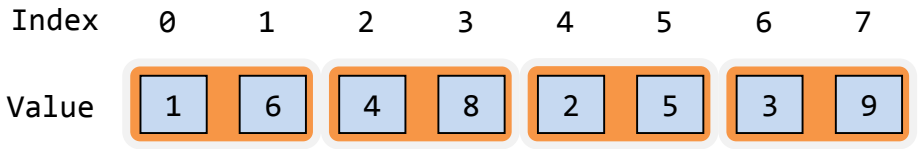
In even-phase, all even/odd indexed pairs of adjacent elements are compared. If a pair is in the wrong order, the elements are switched. Similarly, the same process repeats for odd/even indexed pairs in odd-phase. The odd-even sort algorithm works by alternating these two phases until the list is completely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

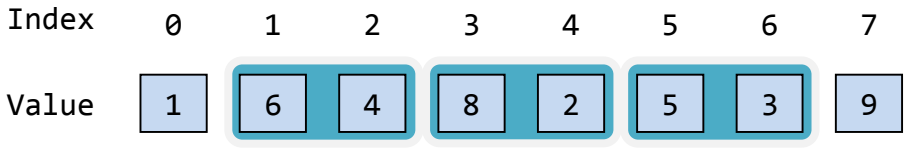
1. [Even-phase]
   even/odd indexed adjacent elements are grouped into pairs.



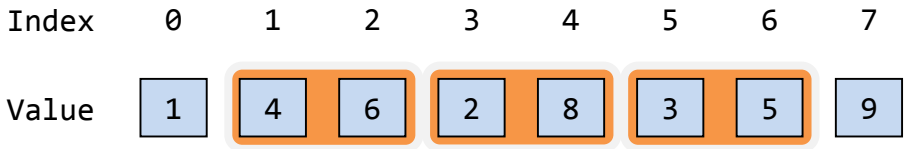
2. [Even-phase]
   elements in a pair are switched if they are in the wrong order.



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*** ($0 \leq n \leq 536870911$)

   ii、 (String) the input file name

   iii、 (String) the output file name

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

```
$  mpirun ./HW1_102030405_basic 1000 in_file out_file
```

3. The input file lists ***n*** 32-bit (4 Byte) floats in binary format. Please refer to the sample input files.
4. The output file lists the ***n*** 32-bit (4 Byte) floats from the input file in ascending order. Please refer to the sample output files.

Note:

The **float** here refers to **IEEE754 binary32**, or single-precision floating point.

You can use the **`float`** keyword in C/C++.

The input is guaranteed contain **none** of the following:

- `-INF`
- `+INF`
- `NAN`

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 be able to handle the arbitrary number of input values and the number of cores given in our test inputs. **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.**

- The implemented algorithm **must follow the odd-even sort principal**. 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 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.

   i、 Methodology

   (a). **System Spec (**If you run your experiments on your own machine**)** Please specify the system spec by providing the CPU, RAM, disk and network (Ethernet / Infiniband) information of the system.

   (b). **Performance Metrics**: How do you measure computing time, communication time and IO time? How do you compute the values in the plots?

   ii、 Plots: Speedup Factor & Time Profile

   - Conduct strong scalability experiments, and plot the speedup and time profile results as shown in figure1 and figure2.
   - Your plot must contain at least 4 settings (e.g., scales), and include both single node and multi-node environment.
   - Make sure your plot is properly labeled and formatted.

- You can generate your own test case with proper problem size to ensure the experimental results are accurate and meaningful. (e.g. running time are long enough)

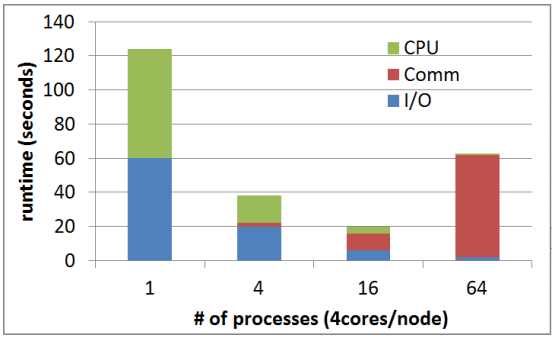


Figure 1: Time profile

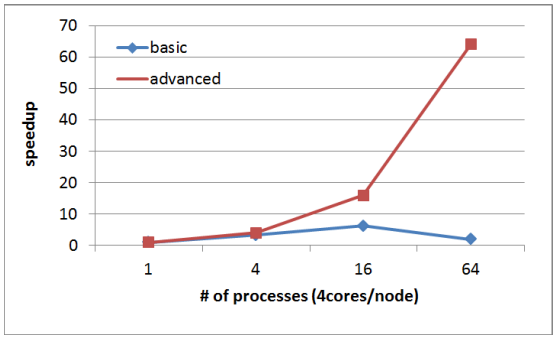


Figure 2: Speedup

iii、Discussion (Must base on the results in the plots)

- Compare the performance your basic and advanced implementations. How much faster is your advanced implementation? Why it can be faster?
- Compare I/O, CPU, Network performance. Which is the bottleneck? Why is it? How could it be improved? You may discuss this question for the two implementations separately or together.
- Compare scalability. Does your program scale well? Why or why not? How can we achieve better scalability? You may discuss this question for the two implementations separately or together.

iv、Others

You are strongly encouraged to conduct more experiments and analysis of your implementations.

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.

6 GRADING

1. **Correctness** (50%)

   i、 Basic version is correct when the number of input items is the same as the number of MPI processes. [10%]

   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** (15%)

   Based on how fast your advanced version can run.

3. **Report** (25%)

   Grading is based on your evaluation, discussion and writing. If you want to get more points, design or conduct more experiments to analyze your implementation.

4. **Demo** (10%)

   Demo will mainly focus on the following aspect:

   i、 Explain your implementation.

   ii、 Explain the key results and findings from your report.

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

7 REMINDER

1. Please upload the following files to **homework/HW1** directory on **apollo31** under your home directory before **10/15(Sun) 23:59**:

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

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

   iii、HW1_{student-ID}_report.pdf

   iv、 Makefile

   And make sure you have run the script `hw1-check` to check the format.

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** (including copying code from the Internet), but discussion on code is encouraged.

6. Asking questions through iLMS is welcomed!