CS542200 Parallel Programming
Homework 1: Odd-Even Sort

Revision 6

Due: Wed October 17, 2018 23:59

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 programs, experiments are required. Finally, we encourage you to optimize your programs by exploring different parallelizing strategies for bonus points.

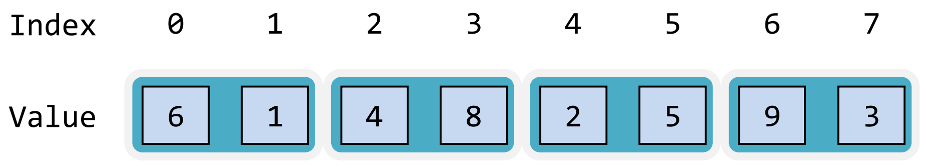
2 PROBLEM DESCRIPTION

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

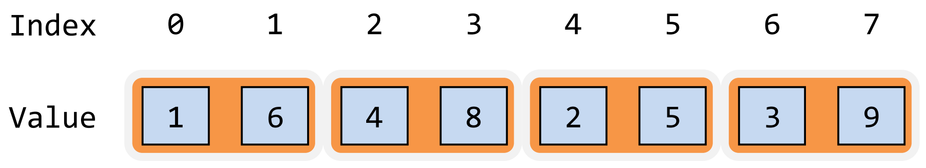
In even-phase, all even/odd indexed pairs of adjacent elements are compared. If a pair is in the wrong order, the elements are switched. Similarly, the same process repeats for odd/even indexed pairs in odd-phase. The odd-even sort algorithm works by alternating these two phases until the list is completely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

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)

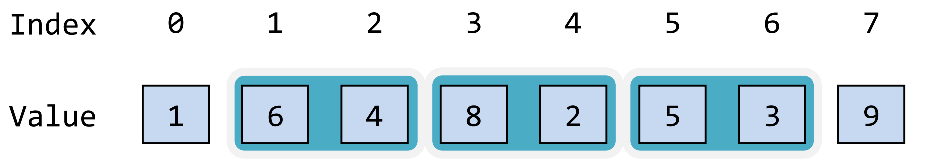
1. [Even-phase] even/odd indexed adjacent elements are grouped into pairs.



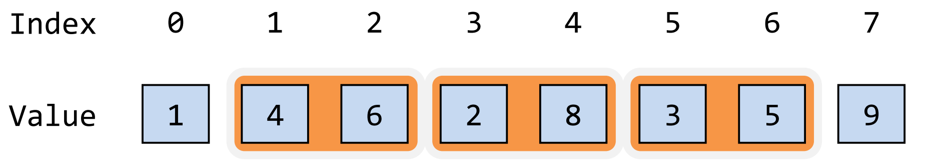
2. [Even-phase] elements in a pair are switched if they are in the wrong order.



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 programs accept 3 input parameters, separated by space. They are:

   i、 (Integer) the size of the list ***n*** ($1 \leq n \leq 536870911$)

   ii、 (String) the input file name

   iii、 (String) the output file name

   Make sure test cases can be assigned through the command line. For instance:

```
$   srun [options] ./basic n in_file out_file
```

3. The input file contains ***n*** 32-bit floats in binary format. The first 4 bytes represents the first floating point number, the fifth to eighth byte represents the second one, and so on. Please refer to the sample input files.
4. The output file should follow the same format of the input file. Please refer to the sample output files.

Note:

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

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

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

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

Any other valid float values are possible to show up in the input.

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
     - For the *elements* within a MPI process, you are allowed to use any method to sort them.
     - *Elements* can only be transferred between neighbor processes. For instance, MPI process rank 5 can only exchange *elements* with ranks 4 and 6, but not rank 3, 7, etc.
     - Transfer of other data (for example, the terminate condition) is not restricted to neighbor processes.
     - 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** (`MPI_File*` functions) to do file input and output.
   - **Must follow the format of input/output file and allow specifying command line arguments as above.**
   - 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

Your 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 handle arbitrary number of input items and processes?
   - ✓ 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 results of your experiments in plots, and explain your observations.

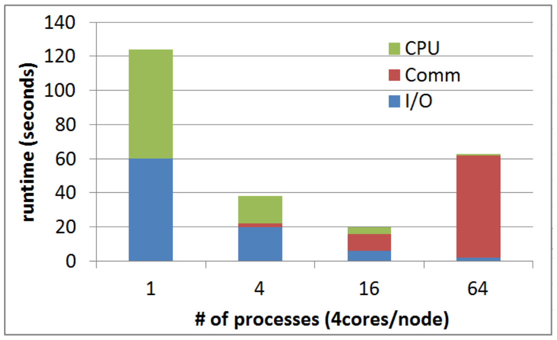
   i、 **Methodology**

   (a). **System Spec** (If you run your experiments on your own cluster) 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 the 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. (Please note that these plots are just examples of how you can layout your figures, and you are unlikely to get the same results)
   - Your plots must contain at least 4 settings (e.g., scales) for both single node and multi-node environment.
   - **Make sure your plots are properly labeled and formatted.**
   - You are encouraged generate your own test case with proper problem size to ensure the experimental results are accurate and meaningful. (e.g. Make sure the execution time is long enough to have meaningful difference for comparison)



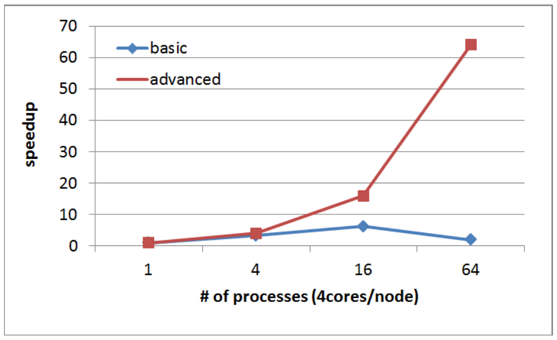


Figure 1: Time profile **Figure 2: Speedup**

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

- Compare the performance of your basic and advanced implementations. How much faster is your advanced implementation? Why can it be faster?
- Compare I/O, CPU, Network performance. Which is/are the bottleneck(s)? Why? How could it be improved? You may discuss for the two implementations separately or together.
- Compare scalability. Do your programs scale well? Why or why not? How can you achieve better scalability? You may discuss for the two implementations separately or together.

iv、 Others

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

4. Experiences / Conclusion

It can include these following aspects:

✓ Your conclusion of this assignment.

✓ What have you learned from this assignment?

✓ What difficulties 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. **[50%] Correctness**

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

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

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

   [15%] Advanced version produces the correct results with arbitrary input problem size, without any restriction.

2. **[15%] Performance**

   Based on how fast your advanced version can run.

   Visit http://140.114.91.183/scoreboard/hw1/ for **preliminary** results.

3. **[25%] Report**

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

4. **[10%] Demo**

   Demo will mainly focus on the following aspect:

   ✓ Explain your implementation.

   ✓ Explain the key results and findings from your report.

   ✓ **Your extra efforts. (Why you deserve more bonus points?)**

7 REMINDER

1. Please upload the following files to **~/homework/hw1** directory on **apollo31** under your home directory before **10/17 (Wed) 23:59**:

   i、 basic.c (or basic.cc)

   ii、 advanced.c (or advanced.cc)

   iii、Makefile

   Run `hw1-judge` to check your code.

   Please upload the following files to iLMS before 10/17 (Wed) 23:59

   iv、 report.pdf

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

3. For late submission penalty policy:

Multiplier	Submission time
*1	Before 10/17 (Wed) 23:59
*0.95	After 10/17 (Wed) 23:59, Before 10/20 (Sat) 23:59
*0.9	After 10/20 (Sat) 23:56, Before 10/24 (Wed) 23:59
*0	After 10/24 (Wed) 23:59

   For example, submission at 10/17 23:59:01 will receive the *0.95 multiplier penalty.

   The submission time is based on the clock on apollo31. You can use the **`date`** command to get the current time.

4. **0 will be given to cheater** (even copying code from the Internet), but discussion on code is encouraged.

5. You are welcome to ask questions through iLMS!