# CS542200 Parallel Programming HW1: Odd-Even Sort

Josh Kao

NTHU LSA Lab

2014/09/30

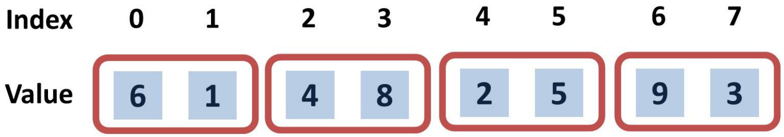
## Outline I

## What's odd-even sort

- It contains 2 phases: Even-phase and Odd-phase
- Runs these two phase alternatively until it converged.

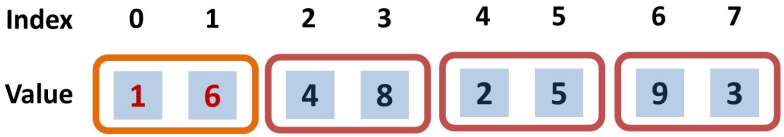
## Even-phase

- Partition the list into (even, odd)-indexed pairs



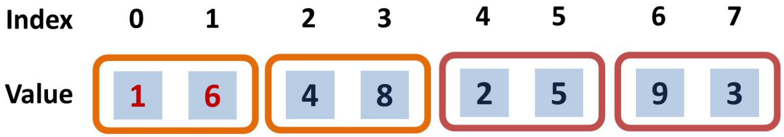
## Even-phase

- Compare (6,1) pair, and swap



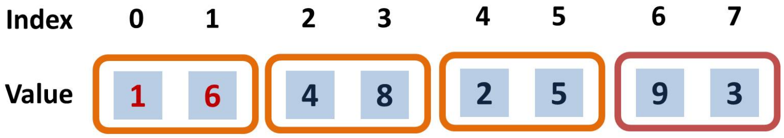
## Even-phase

- Compare (4,8) pair, and do nothing



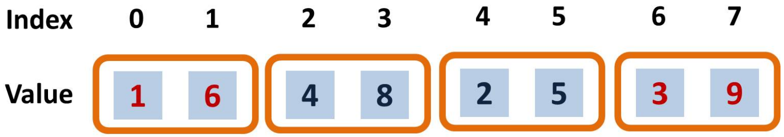
## Even-phase

- Compare (2,5) pair, and do nothing



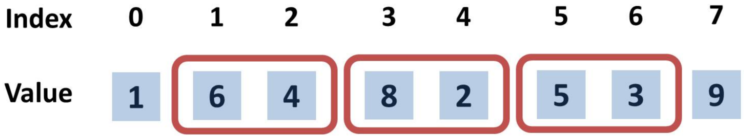
## Even-phase

- Compare (9,3) pair, and swap



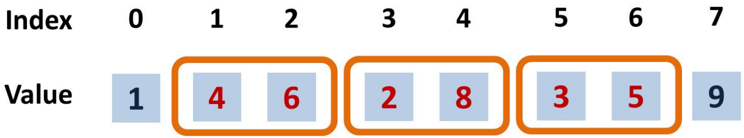
## Odd-phase

- Partition the list into (odd, even)-indexed pairs



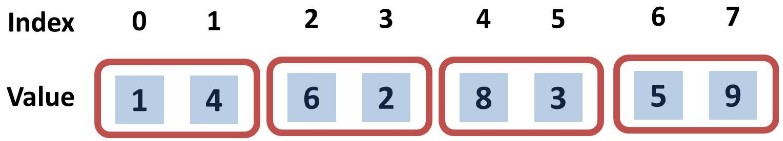
## Odd-phase

- Repeat the same actions as what even-phase do



## Even-phase

- Back to even-phase, and repeat
  [even-phase] → [odd-phase] → [even-phase] until it converged



## Outline I

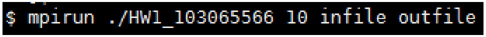
## Execution Format

- Your program has to be able to **reading file**, and generate output in another file.
- Your program accepts two input parameters.

**Format**

$ mpirun {YOUR PROGRAM} {DATA SIZE} {INPUT FILE} {OUTPUT FILE}

For instance,



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

## Testcase Format

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

## Outline I

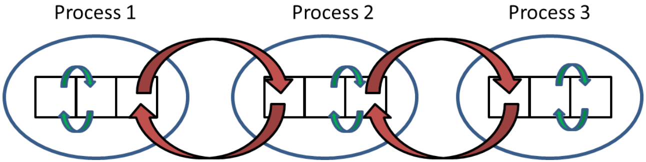
## Implementation

- Basic odd-even sort
  - Strictly limited to odd-even sort
- Advanced odd-even sort
  - MPI tasks can only send messages to its neighbors
  - Better performance

Your program should detect whether the list is sorted or not.

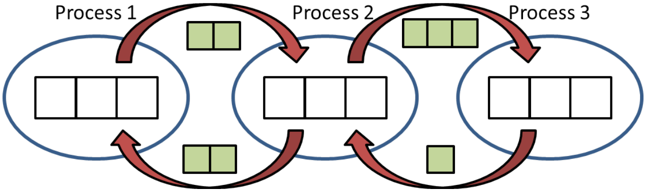
## Basic Version

- Basic odd-even sort
    - Strictly limited to odd-even sort



## Advanced Version

- Advanced odd-even sort
  - MPI tasks can only send messages to its neighbors
  - The number of elements sent in each message can be arbitrary



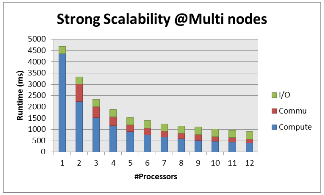
## Report

- **Implementation**
- **Experiments**
  1. Strong Scalability & Time Distribution
  2. Speedup Factor
  3. I/O
  4. Other

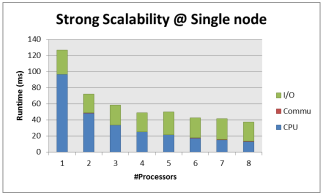
  Think of the trend of the figure and explain.
- **Experience/Conclusion**

## Strong Scalability & Time Distribution

- Plot at least 4 figures
  {basic, advanced} x {single-node, multi-nodes}
- Measure time interval of these metrics in different distribution:
  1. Computing
  2. Communication
  3. I/O



(a) Multi-Ndoes



(b) Single-Node

Figure: Strong Scalability

## Speedup Factor

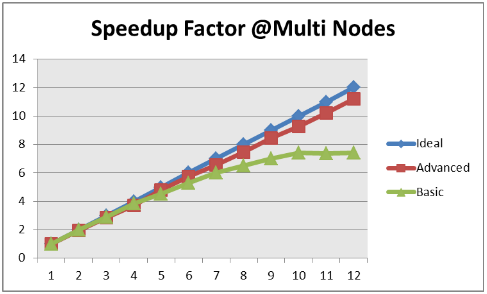


Figure: Speedup Facotr @MultiCore

## I/O

- Test different size of data to observe the trend of performance.

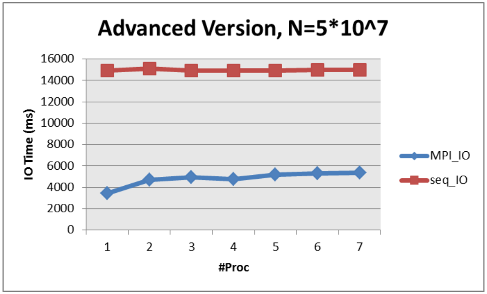


Figure: Data Size n = 5 * $10^7$

## Outline I

## Grading

1. **[50%]** Correctness
2. **[30%]** Report
3. **[20%]** Demo

For each detail, check out the document on iLMS.

## Outline I

## Reminder

- Upload **HW1_{Student-ID}.zip** to iLMS before 10/26(sun) 23:59:59
  1. HW1_{Student-ID}_basic.c (or *.cpp)
  2. HW1_{Student-ID}_advanced.c (or *.cpp)
  3. HW1_{Student-ID}_report.pdf
- Please **start your work ASAP** and do not leave it until the last day!
- Late submission penalty policy please refer to syllabus.
- Please do experiments by submitting jobs without running on headnode.
- Asking questions on iLMS or through e-mail is also welcome!