Sorting

Jing Liu
TDB & LMB, Uppsala University

Programming of Parallel Computers, Feb 2016

Sorting

- most commonly used, and well studied.
- compare based
  - compare-exchange.
- non-compare based
- lower bound of any any comparison-based sort of n numbers is $\Theta(n \log n)$.

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Bubble/sinking sort

```
for i = n-1 down to 1
        for j = 1 to increase to i
                compare_exchange(a_j ,a_{j+1})
        end for
end for
```

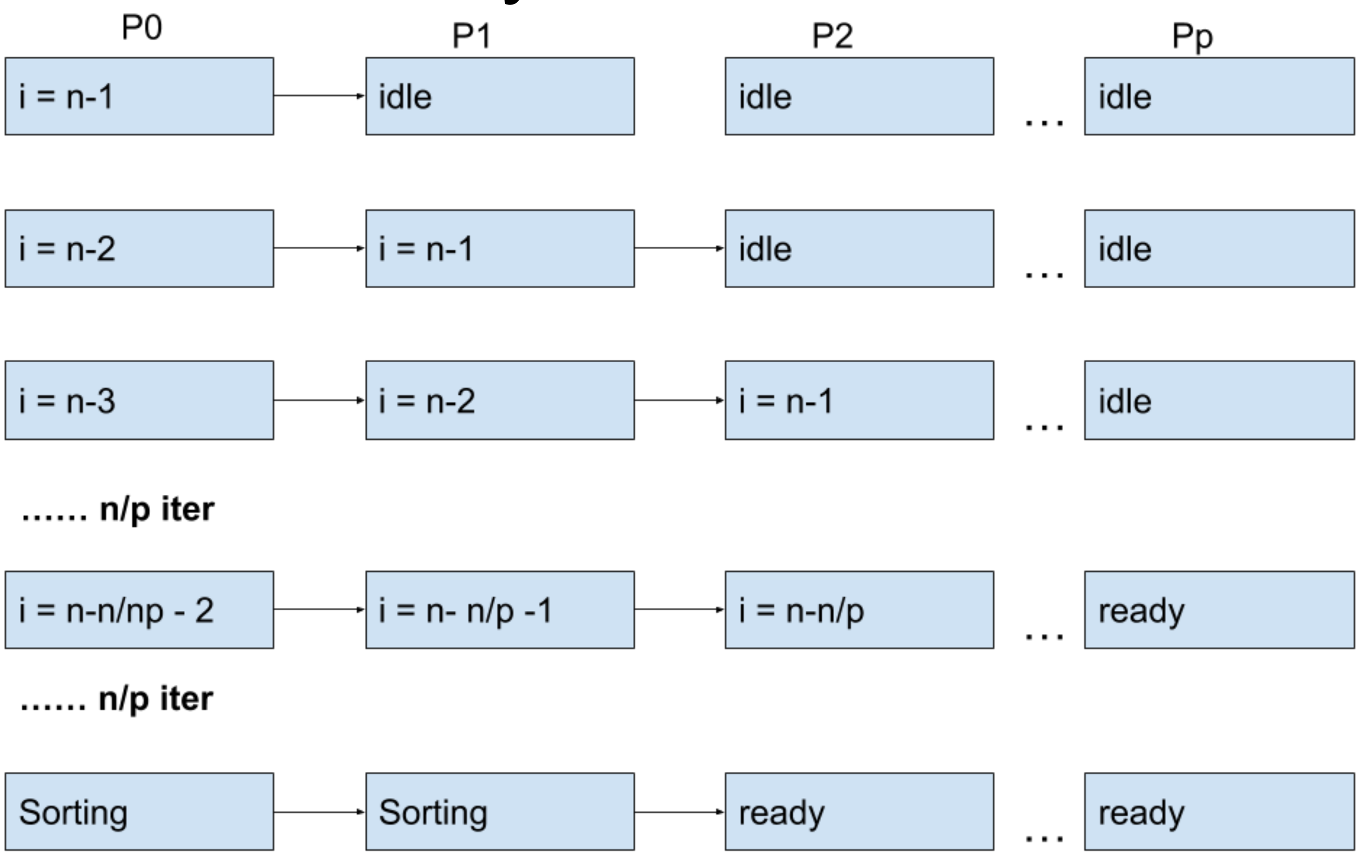
Worst case performance	$O(n^2)$
Best case performance	$O(n)$
Average case performance	$O(n^2)$

jing.liu@it.uu.se / jing.liu@icm.uu.se

Dept. of IT

Bubble/sinking sort

- In case of array is distributed



jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Bubble/sinking sort

- Bubble /sinking sort pipeline
  - Load balance?
    - idling?
    - startup time?
    - last processor get ready first?
  - communication?
    - In each iteration, n times

Worst case performance $O(n^2)$

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Odd-even sort

- Two phases
  - Odd: C/E elements only with odd indices with their right neighbor
  - Even: C/E elements only with even indices with their right neighbor

  - Work within each phase is perfectly parallel!
  - n-times

Worst case performance	$O(n^2)$
Best case performance	$O(n)$

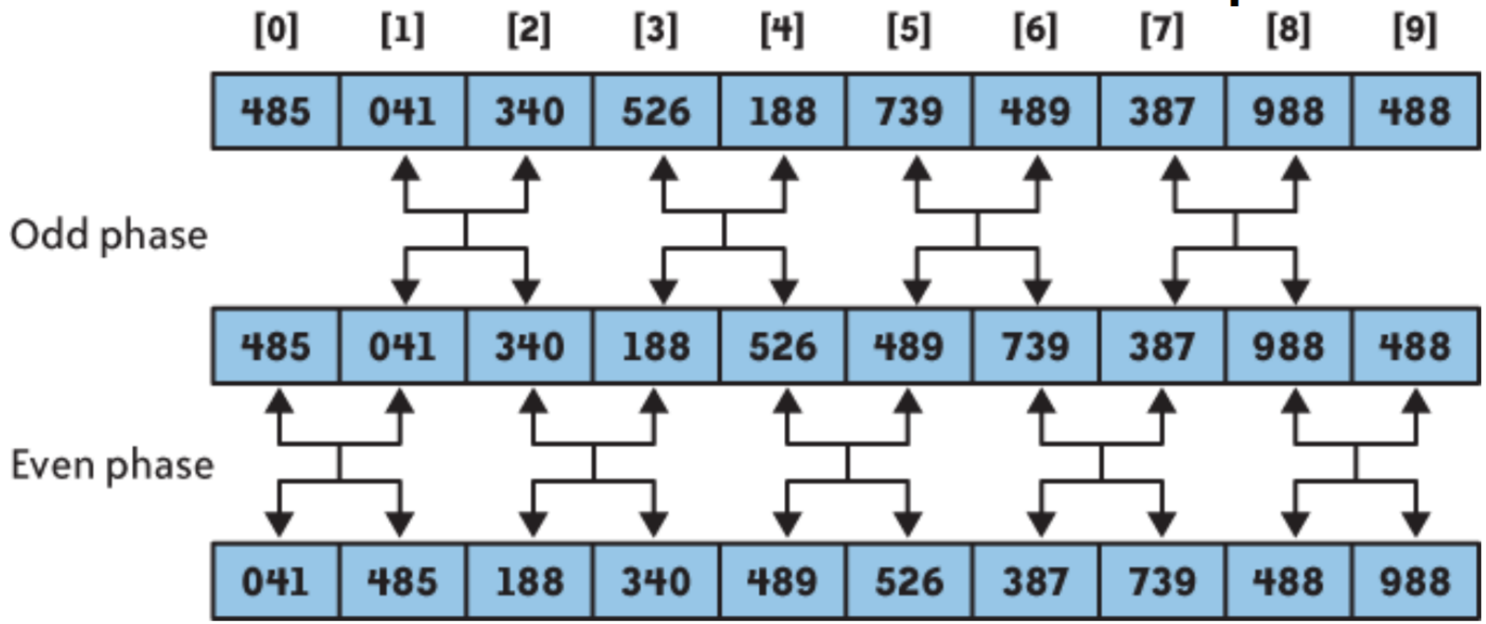
jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Odd-even sort

Alternate between odd and even phases



UPPSALA UNIVERSITET

Dept. of IT

Odd-even sort

- n-times
- perfect parallel loops

```
template <class T>
void OddEvenSort (T a[], int n)
{
    for (int i = 0 ; i < n ; i++)
    {
        if (i & 1) // 'i' is odd
        {
            for (int j = 2 ; j < n ; j += 2)
            {
                if (a[j] < a[j-1])
                    swap (a[j-1], a[j]) ;
            }
        }
        else
        {
            for (int j = 1 ; j < n ; j += 2)
            {
                if (a[j] < a[j-1])
                    swap (a[j-1], a[j]) ;
            }
        }
    }
}
```

UPPSALA UNIVERSITET

Dept. of IT

Parallel Odd-even sort

- Divide data equally
- All processors sort locally.
- Two phases
  - Even: Processor with EVEN id merge data with next processor (to the right)
  - Odd: Processor with Odd id merge data with next processor (to the right)
  - P steps, or abort when no changes

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Parallel Odd-even sort

- Problems:
  - load imbalance: only half processors are working in each step
  - Unnecessary communication: Data moved back and forth
  - slow

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Quick sort

Worst case performance	$O(n^2)$
Best case performance	$O(n \log n)$ (simple partition) or $O(n)$ (three-way partition and equal keys)
Average case performance	$O(n \log n)$

- In sequential (Recursive algorithm)
  - Select a pivot (which one?)
  - Divide data into two lists according to the pivot element (smaller/ larger)
  - Sort the lists independently with Qucicksort (call the quick sort function again)
  - Quicksort (pivot , list)

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Quick sort in parallel

- Naïve method
  - Start with one processor and all data
  - In each split employ a new processor for the other part
  - After $\log_2 P$ steps sort locally with each processors

  - Different pivot makes different, try different pivot selection strategy on LAB.

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Quick sort in parallel

- An improved parallel quick sort
  - Divide data equally and sort locally
  - Select pivot (the median) and broadcast within processor set
  - In each processor divide data according to pivot
  - Divide the processors into 2 sets, and exchange data pairwise between processors in the two sets such that the processors in one set gets data smaller that pivot and the other get larger ones
  - Merge data and keep data sorted
  - repeat 2~5 $\log_2 P$ steps

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Quick sort in parallel

- Problems:
  - Complex algorithm
  - Selection of pivot is important, a bad selection can lead to load imbalance.

➔ How to choose the pivot?

jing.liu@it.uu.se / jing.liu@icm.uu.se

Quick sort in parallel

- Selecting Pivot -- medians

8 processor

step1: select median in P0

step2: select median in P0, P4

step3: select median in P0,P2,P4,P6

- How about if the data is almost sorted?

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Quick sort in parallel

- Selecting Pivot – median of medians with respect to each processor set

  - Select medians from P0 ~ P7
  - Select the median of these medians for each processor set

  - What if all medians are bad? either too high or to low?

jing.liu@it.uu.se / jing.liu@icm.uu.se

Quick sort in parallel

- Selecting Pivot -- means

  8 processor

  step1: select mean in P0

  step2: select mean in P0, P4

  step3: select mean in P0,P2,P4,P6

  - How about if the data is not uniform?

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Quick sort in parallel

- Selecting Pivot -- means

  8 processor

  step1: select mean in P0

  step2: select mean in P0, P4

  step3: select mean in P0,P2,P4,P6

  - How about if the data is not uniform?

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Quick sort in parallel

- Selecting Pivot – Select medians at once
- For all steps in Quick sort, we need P-1 pivots, and we will select them at once
  - Each process select *L* evenly distributed elements within its data.
  - Sort all selected elements (*L*P*) globally
  - Choose *P-1* evenly distributed elements as pivots and broadcast

Good if *P or L* is big enough, but expensive.

jing.liu@it.uu.se / jing.liu@icm.uu.se

Quick sort in parallel

- Selecting Pivot – Statistical expectation values for the medians.
  - If the distribution is known, eg: normal, uniform.

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Bucket sort

Worst case performance	$O(n^2)$
Best case performance	$\Omega(n+k)$
Average case performance	$\Theta(n+k)$

- Algorithm:
  - Define k-buckets in the interval [min, max], and filter the elements into the buckets
  - Assign the buckets into processors
  - Sort the buckets locally in each processor (parallel)
  - Problem: 1) Large serial section in filtering.

2) Load imbalance, difficult to create buckets with equal number of elements

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

Bucket sort

- How to do filtering?
  - Assume equal sized buckets in the interval [min, max], and the unsorted list is ***a***

$$b = \left\lceil \frac{a[i] - min}{max - min} * nbuckets \right\rceil - 1$$

  - Linear time, independent of nbuckets

jing.liu@it.uu.se / jing.liu@icm.uu.se

UPPSALA UNIVERSITET

Dept. of IT

read more

- sorting https://en.wikipedia.org/wiki/Sorting_algorithm
- $O(n^2)$, $O(\log n)$

https://en.wikipedia.org/wiki/Big_O_notation

jing.liu@it.uu.se / jing.liu@icm.uu.se