Parallel quicksort

Mahmud Allahverdiyev
Skolkovo Institute of Science and Technology

Sorting

- In practice
  - Faster search & look-up routines
  - More efficient data processing
  - Doing stream operations, merging over sequence
- Available algorithms[1]
  - Slow $O(n^2)$ or worse: Bubble, Insertion, Selection, …
  - Fast $O(nlog(n))$ Quicksort*, Merge sort, Heapsort, …
  - Fast*/non-comparison sorts: Counting, Radix, Bucket, …

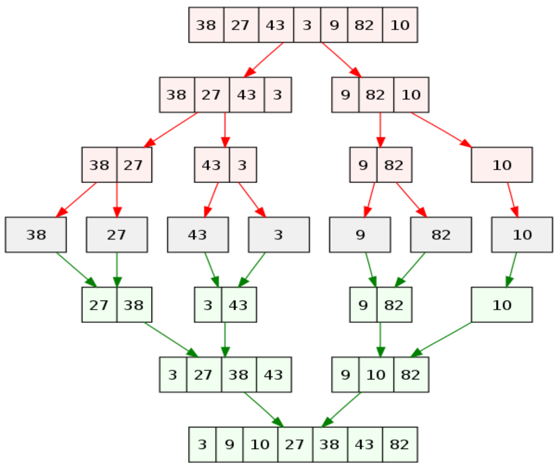
img ref: https://en.wikipedia.org/wiki/Sorting_algorithm

HPC & Sorting

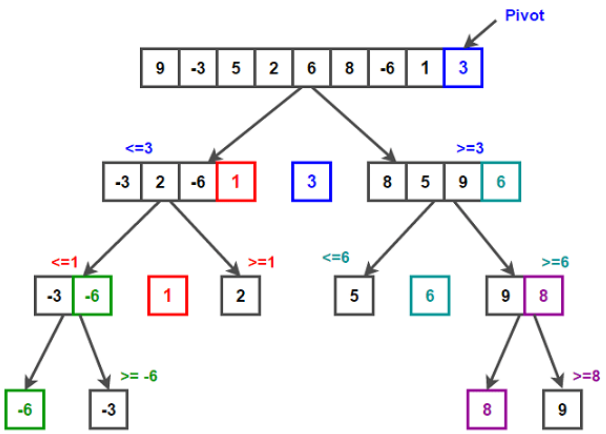
- In practice, Quicksort and Merge sort are most popular sorting algorithms
- Many programming languages and libraries provide [*hybrid*] built-in sorting routines. (C++ STL, Java Collections, etc.)
- Several of the sorting algorithms can be employed in parallel environments and DFS

Merge sort & Quicksort recap

- Merge sort
  - Recursively split the problem into smaller instances and solve them separately
  - Merge sorted lists back



- Quicksort
  - Recursively choose pivots and split data around the pivot
  - Sort split subsequences separately



img refs:
https://en.wikipedia.org/wiki/Merge_sort
https://www.techiedelight.com/quicksort/

Parallel quicksort

- There have been several methods proposed to parallelize sorting algorithms.

- Making use of parallelism in the following methods:
  - Sorting local lists recursively and exchanging data according to pivot element → load balancing issues
  - Hyperquicksort - similar to the previous idea, approximate median selection done by manager process
  - <u>Regular Sampling method</u>- sample sorted local lists before exchange between different processes

Regular Sampling method

- Each process (machine) sorts have its own initial local list (N = 27, p = 3, n = $\frac{N}{p}$ = 9)

3	14	15	92	65	35	89	79	52	Process 1
38	46	26	43	42	33	79	50	28	Process 2
84	19	71	69	40	93	75	10	58	Process 3

- Each process sorts its own local list in $O(nlog(n))$

3	14	15	35	52	65	79	89	92	Process 1
26	28	33	38	42	43	46	50	79	Process 2
10	19	40	58	69	71	75	84	93	Process 3

Regular Sampling method

- Each process samples local data at *indices $\frac{n}{p} - 1, 2\frac{n}{p} - 1, 3\frac{n}{p} - 1, 4\frac{n}{p} - 1, \ldots$

3	14	15	35	52	65	79	89	92	Process 1
26	28	33	38	42	43	46	50	79	Process 2
10	19	40	58	69	71	75	84	93	Process 3

- Manager process (i.e. process 1) gathers sampled p^2 points from all processes and sorts them in $O(p^2 \log(p))$

15	33	40	43	65	71	79	92	93	Manager

Idea: If we select $p - 1$ pivots from the above samples, they will be sufficient to route each data element to its correct destination!

$\frac{n}{p}, 2\frac{n}{p}, 3\frac{n}{p}, 4\frac{n}{p}, \ldots$ also work in similar fashion.

Regular Sampling method

- Manager selects $p - 1$ pivots and broadcasts them to all the processes

15	33	40	43	65	71	79	92	93	Manager

40	71	Pivots

3	14	15	35	52	65	79	89	92	Process 1
26	28	33	38	42	43	46	50	79	Process 2
10	19	40	58	69	71	75	84	93	Process 3

less than 40
less than 71
greater or equal to 71

- Each process sends chunks of its data based on pivots to other processes

3	14	15	35	26	28	33	38	10	19	Process 1
52	65	42	43	46	50	40	58	69		Process 2
79	89	92	79	71	75	84	93			Process 3

Regular Sampling method

- At the last step, each process sorts its final data

3	14	15	35	26	28	33	38	10	19	Process 1
52	65	42	43	46	50	40	58	69		Process 2
79	89	92	79	71	75	84	93			Process 3

Now, The whole data is sorted and sequentially stored in distributed system.

3	10	14	15	19	26	28	33	35	38	Process 1
40	42	43	46	50	52	58	65	69		Process 2
71	75	79	79	84	89	92	93			Process 3

Notes:

- Except pivots, every data element is sent/received at most once.
- The lengths of final local lists differ at most by $O(p)$.