HYPER QUICKSORT

Chandani Jaiswal

CSE 633 Spring 2020

Instructor: Prof. Dr. Russ Miller

May 07, 2020

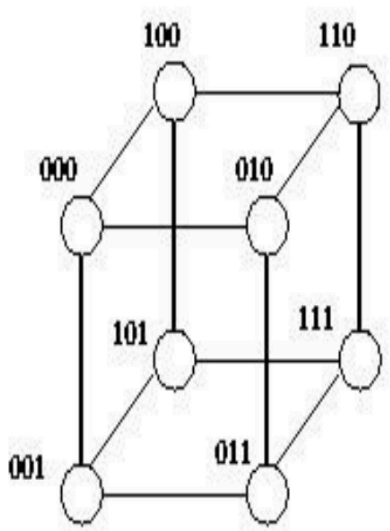
University at Buffalo The State University of New York

Agenda

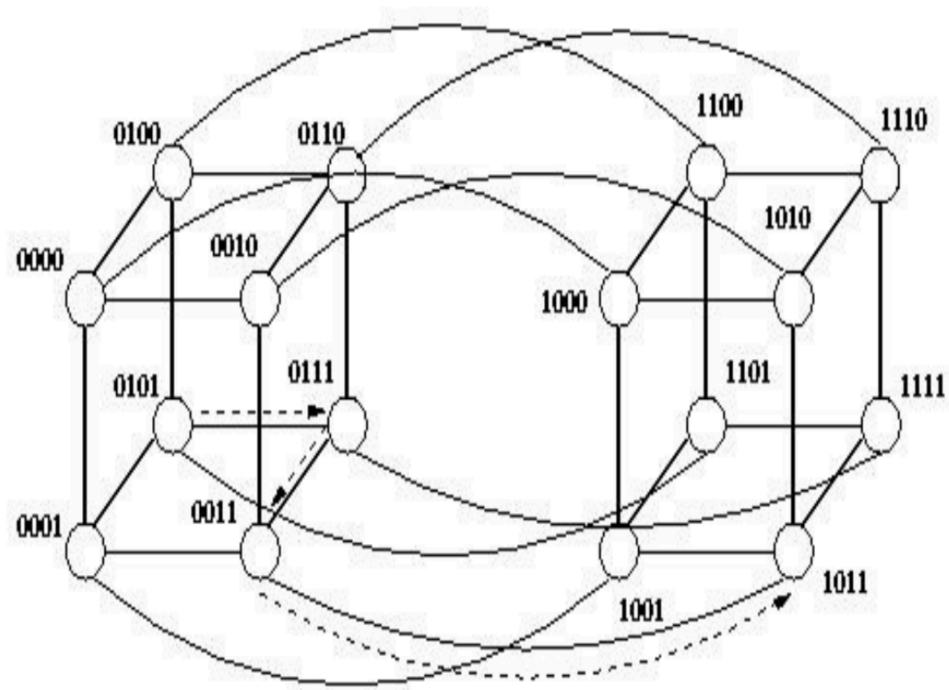
- Overview of Parallel Algorithm
- Modified Hyper Quick Sort Algorithm
- Working Example
- Complexity Analysis
- Observations
- Results on Small Data
- Results on Big Data
- Speedups for different data
- Limitations
- Learnings
- References

Hypercube

A hypercube of size n consists of n processors indexed by the integers {0,1, . . . , n − 1} where n > 0 is an integral power of 2. Processors A and B are connected if and only if their unique log2 n-bit strings differ in exactly one position.



3-D hypercube



4-D hypercube

Parallel Quick Sort Algorithm

- We randomly choose a pivot from one of the processers and broadcast it to every processor.
- Each processor divide its unsorted list into two lists: those smaller than (or equal) the pivot, those greater than the pivot.
- Each processor in the upper half of the processor list sends its "low list" to a partner processor in the lower half of the processor list and receives a "high list" in return.
- Now, the upper-half processors have only values greater than The pivot, and the lower-half processors have only values smaller than the pivot.
- Thereafter, the processors divide themselves into two groups and the algorithm continues recursively.
- After log(P) recursions, every processor has an unsorted list of values completely disjoint from the values held by the other processers.
- The largest value on processor i will be smaller than the smallest value held by processor i + 1
- Each processor can sort its list using sequential quicksort.

Hyper Quicksort Algorithm

- Each processor starts with a sequential quicksort on its local list
- Now we have a better chance to choose a pivot that is close to the true median.
  - The processor that is responsible for choosing the pivot can pick the median of its local list.
- The three next steps of hyper quick sort are the same as in parallel algorithm 1.
  - Broadcast
  - Division of "low list" and high list"
  - Swap between partner processors
- The next step is different in hyper quick sort.
  - On each processor, the remaining half of local list and the received half-list are merged into a sorted local list.
- Recursion within upper-half processors and lower-half processors.

Example

P0 P1 P2 P3

1,5,10,12,17 || 2,6,9,14,19 || 3,8,13,15,20 ||4,7,11,16,18

Each process sorts values it controls

Example

P0 P1 P2 P3

1,5,10,12,17 || 2,6,9,14,19 || 3,8,13,15,20 ||4,7,11,16,18

1,5,10,12,17 || 2,6,9,14,19 || 3,8,13,15,20 ||4,7,11,16,18

Process P0 broadcasts its median value

Example

P0 P1 P2 P3

1,5,10,12,17 || 2,6,9,14,19 || 3,8,13,15,20 ||4,7,11,16,18

1,5,10,12,17 || 2,6,9,14,19 || 3,8,13,15,20 ||4,7,11,16,18

1,3,5,8,10 || 2,4,6,7,9 || 12,13,15,17,20 ||11,14,16,18,19

Processes will exchange "low", "high" lists
P0-P2 and P1-P3

Example

P0 P1 P2 P3

1,5,10,12,17 || 2,6,9,14,19 || 3,8,13,15,20 ||4,7,11,16,18

1,5,10,12,17 || 2,6,9,14,19 || 3,8,13,15,20 ||4,7,11,16,18

1,3,5,8,10 || 2,4,6,7,9 || 12,13,15,17,20 ||11,14,16,18,19

1,3,5,8,10 || 2,4,6,7,9 ||| 12,13,15,17,20 ||11,14,16,18,19

Processes merge kept and received values.
Processes P0 and P2 broadcast median values.

Example

P0 P1 P2 P3

1,5,10,12,17 || 2,6,9,14,19 || 3,8,13,15,20 ||4,7,11,16,18

1,5,10,12,17 || 2,6,9,14,19 || 3,8,13,15,20 ||4,7,11,16,18

1,3,5,8,10 || 2,4,6,7,9 || 12,13,15,17,20 || 11,14,16,18,19

1,3,5,8,10 || 2,4,6,7,9 ||| 12,13,15,17,20 || 11,14,16,18,19

1,2,3,4,5 || 6,7,8,9,10 ||| 11,12,13,14,15 || 16,17,18,19,20

Communication pattern for second exchange

Complexity Analysis

- The N log N term represents the sequential running time from Step 2. The d(d + 1)/2 term represents the broadcast step used in Step 4.
- The dN term represents the time required for the exchanging and merging of the sets of elements.

$$\Theta\left(N \log N + \frac{d(d+1)}{2} + dN\right).$$

Observations

1. Log P steps are needed in the recursion.
   - The expected number of times a value is passed from one process to another is log P / 2 , that is quite some communication overhead!
   - The median value chosen from a local segment may still be quite different from the true median of the entire list.

2. Although better than parallel quicksort algorithm 1, load imbalance may still arise.

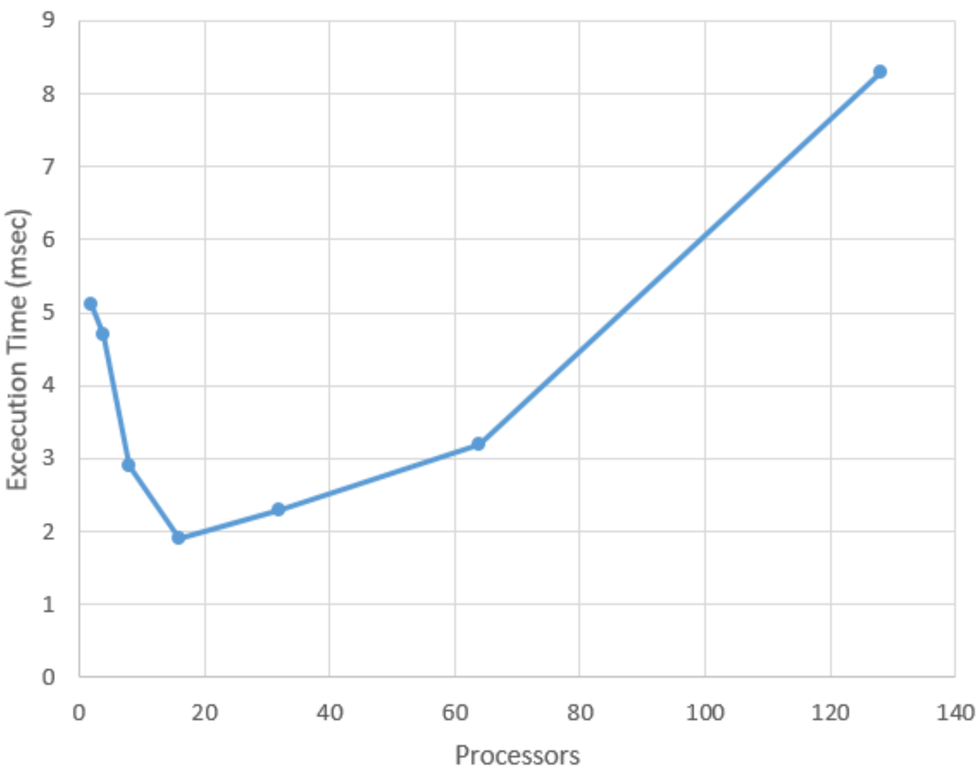
Solution:

- Algorithm 3 – parallel sorting by regular sampling

OBSERVATIONS

Small Data (1 Million)

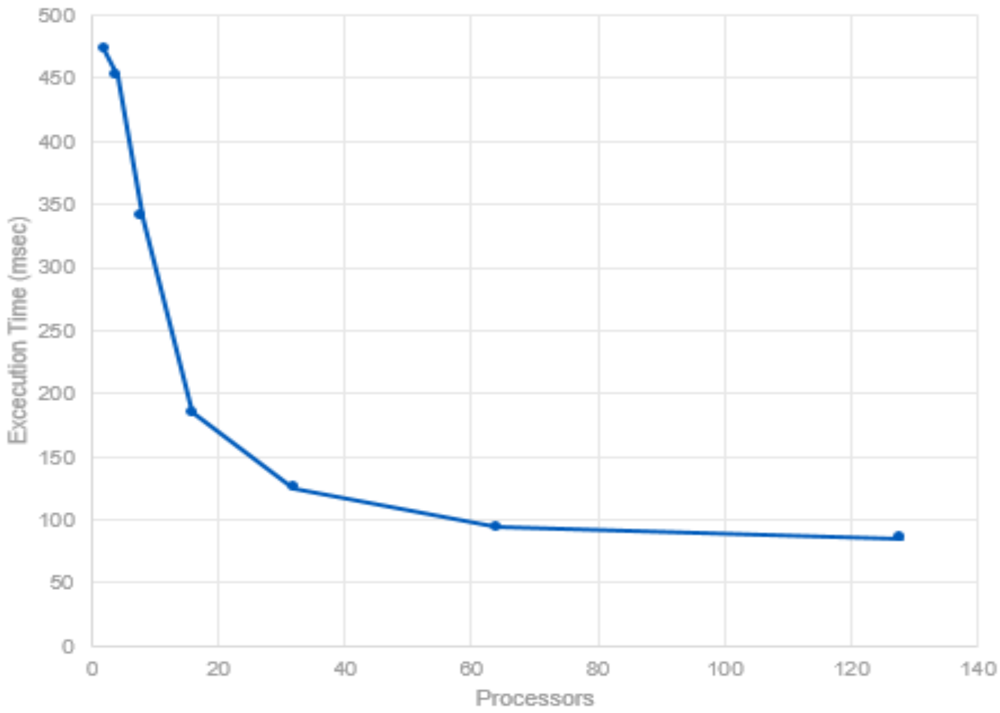
Number of Processors	Execution Time (msec)
2	5.2
4	4.6
8	2.8
16	1.8
32	2.4
64	3.3
128	8.4



OBSERVATIONS

Large Data (10 Million)

Number of Processors	Execution Time (msec)
2	474
4	451
8	340
16	186
32	126
64	93
128	86



OBSERVATIONS SPEED UP

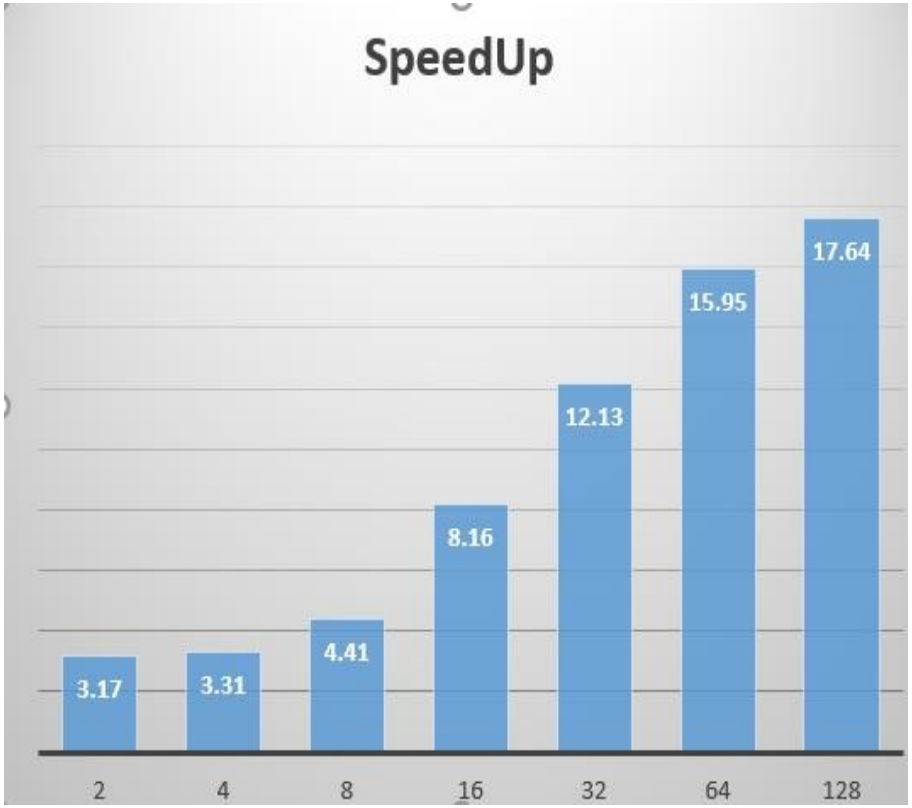
Small Data (1 Million)

Number of Processors	Speedup
2	4.70
4	4.89
8	8.27
16	12.63
32	10.43
64	7.5
128	2.89

OBSERVATIONS

Large Data (10 Million)

Number of Processors	SpeedUp
2	3.17
4	3.31
8	4.41
16	8.16
32	12.13
64	15.95
128	17.64



Limitations

The number of processors has to a be a power of 2. Very High communication overhead.

Learnings

- Observed the difference in runtimes for different number of processors so as the no of processors increase runtime decrease up to certain level and then its increases.
- In order to achieve better performance its critical to identify the optimal number of processors that would be required for any given computation.
- Its always better to limit the number of processors to get maximum speedup.

References

- **Algorithms, Sequential and Parallel: A Unified Approach – Russ Miller and Laurence Boxer. 3rd Edition.**
- **https://www.uio.no/studier/emner/matnat/ifi/INF3380/v10/undervisningsmateriale/inf3380-week12.pdf**

Questions??

THANK YOU

University at Buffalo The State University of New York