HYPER QUICK SORT

- Presenter: Mrunal Inge
- Instructor: Dr.Russ Miller

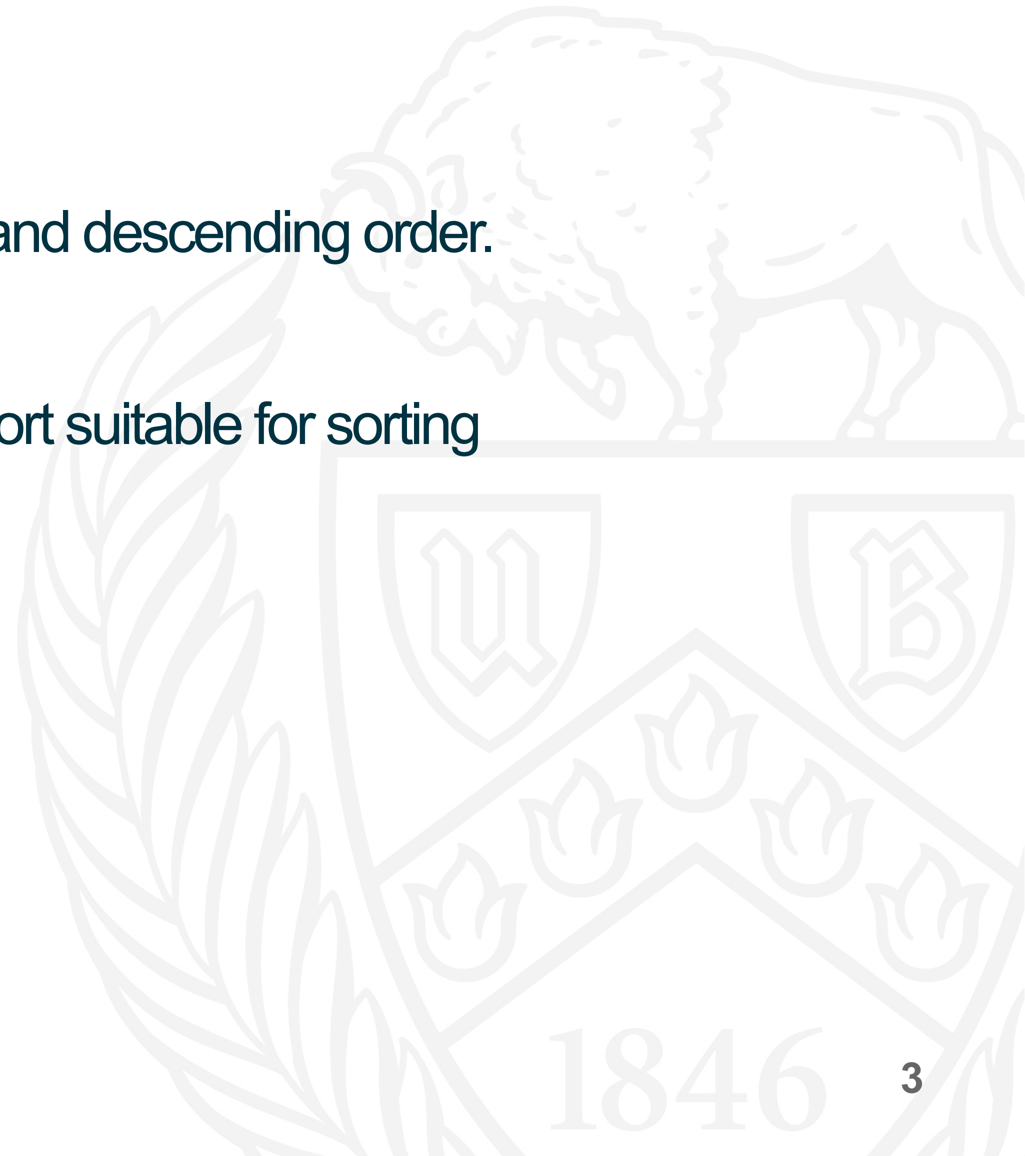
University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

CONTENTS:

- Introduction
- Sequential Quicksort
- Parallel Quicksort
- Example
- Hyperquicksort
- Readings
- Observations
- References

What is Quicksort?

- Quicksort is one of the algorithms used to sort in ascending and descending order.
- It is a divide and conquer algorithm.
- On an average it takes O(n log n) complexity, making quicksort suitable for sorting huge data volumes.



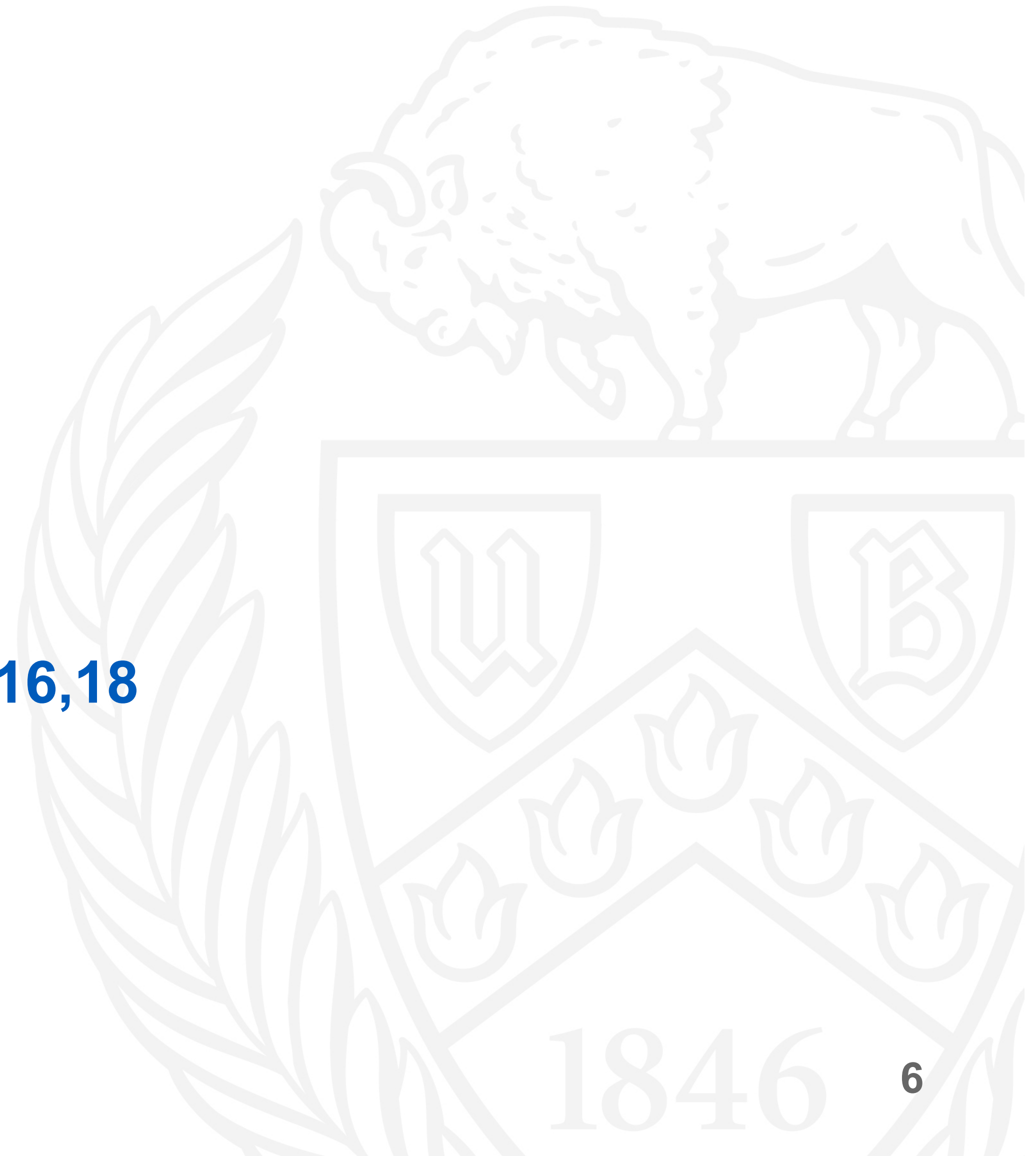
Sequential Quicksort

- We pick a pivot element and partition the array around the pivot element.
- The pivot element can be any random element or median of subset of elements from the array.
- Suppose the median is x, all the elements less than x will go to the left of x, we call it as low list and elements greater than x would go a to the right of x, we call it as high list.
- Then the low list and high list recursively sorts itself as mentioned above.
- The final sorted list will be the concatenation of the low list, high list and the median.

Parallel Quicksort

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

Example



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

After scattering above List:

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

Example

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

Example

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 || 4,7,2,6,9 || 12,13,15,17,20 || 11,14,16,18,19

Example

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 || 4,7,2,6,9 || 12,13,15,17,20 || 11,14,16,18,19

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

Example

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

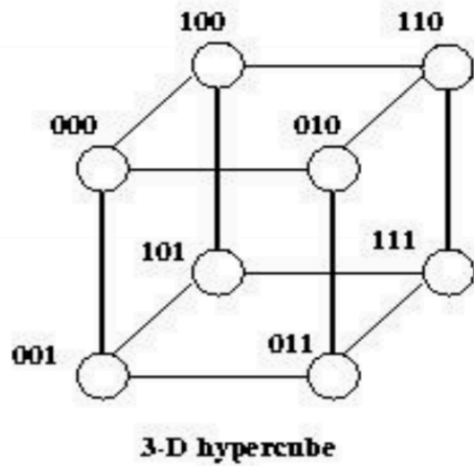
1,3,5,8,10 || 2,4,6,7,9 || 12,13,15,17,20 || 11,14,16,18,19

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

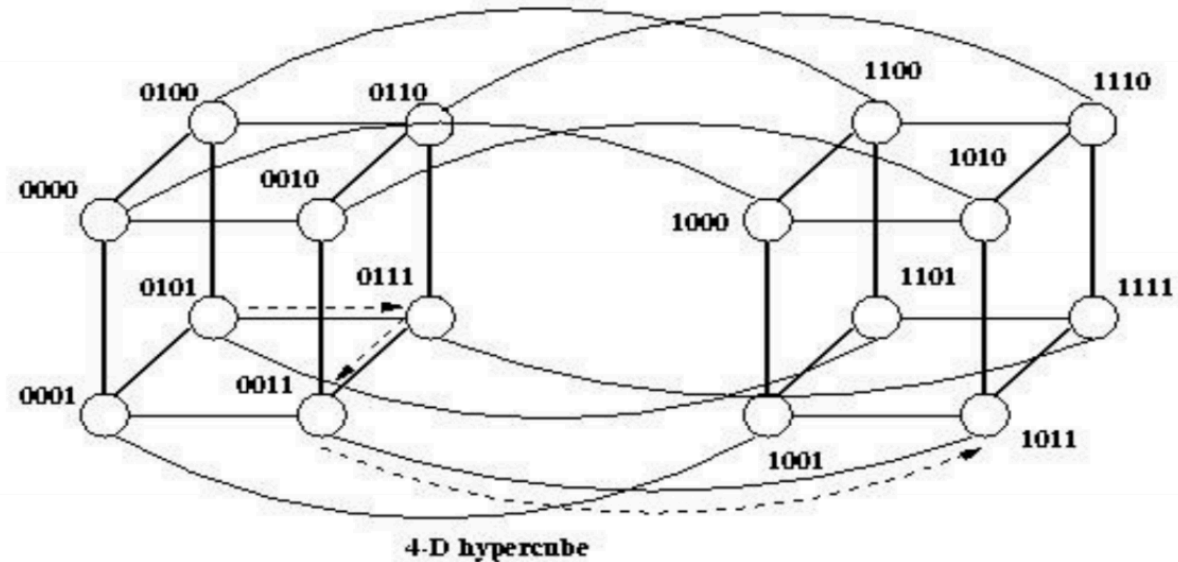
1,2,3,4,5 || 6,7,8,9,10 || 11,12,13,14,15 || 16,17,18,19,20

Hyper Quicksort

- Hyper quick sort is the implementation of quick sort on a hypercube.
- In an N dimensional hypercube with number of processors equal to 2^N, any two processors are connected if and only if their unique log2 n-bits differ exactly in one position.



3-D hypercube



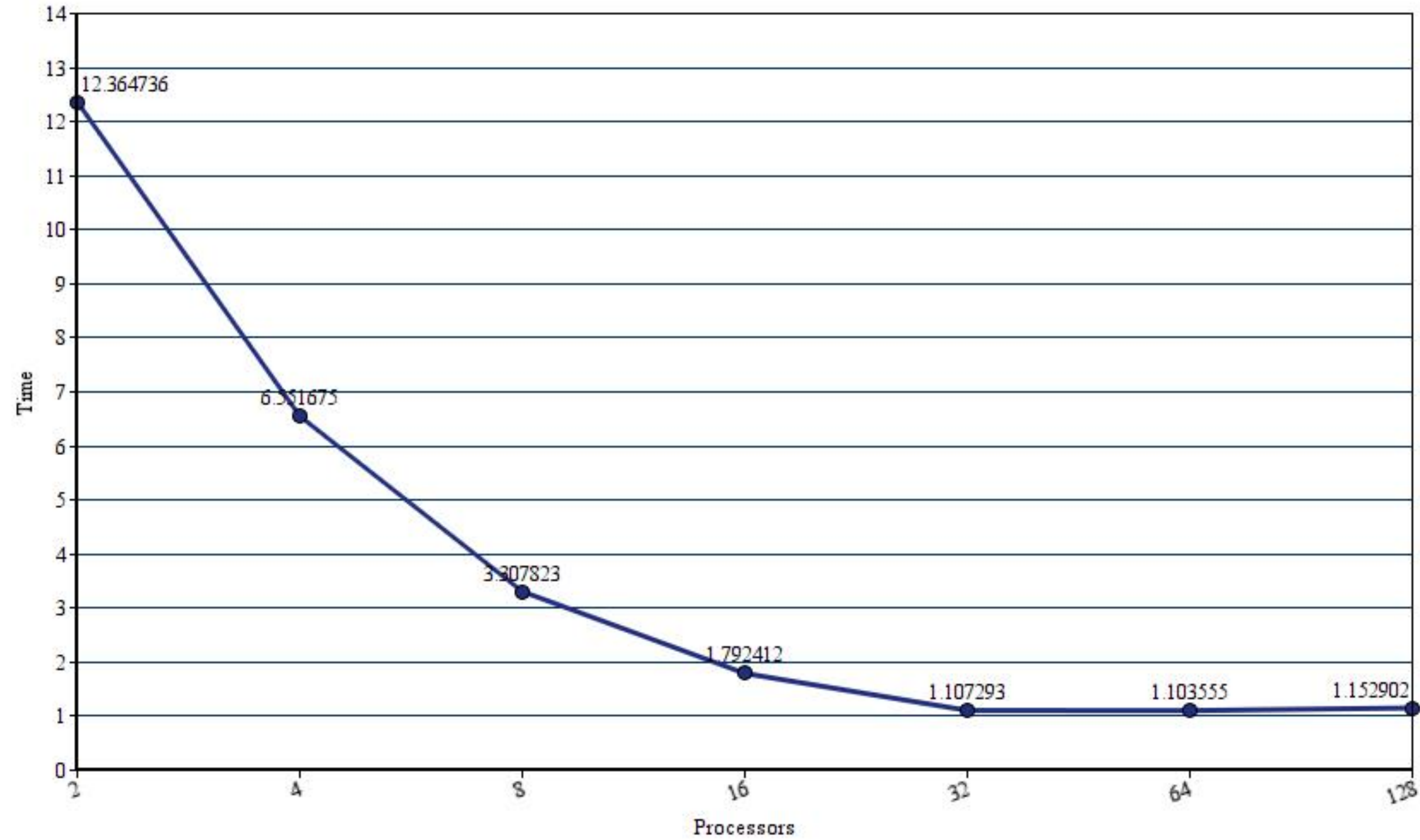
4-D hypercube

Hyper quick sort Algorithm

- Basically, the algorithm is similar to the parallel algorithm.
- Each process starts with the sequential quicksort on its local list.
- Now we can have a better a chance of choosing a pivot which is closer to the median.
- The processor responsible for choosing the median will pick a median from its local list and broadcast it to all processors.
- Then we divide into low list and high list and swap between the partner processors.
- This step is the only different step in hyper quick sort, on each processor the remaining half of the local list and received half list are merged into a sorted local list.
- Finally, we recurse between the upper half and lower half processors.

Sample Readings:

Data Points	2	4	8	16	32	64	128
64000	0.01151883	0.0072725	0.004509833	0.00328183	0.005636	0.00631	0.00806483
128000	0.021566167	0.01311067	0.0079413	0.0047925	0.0046935	0.0041425	0.018263106
1024000	0.1160203	0.0626567	0.031275167	0.016847167	0.0112013	0.0181873	0.02082483
10000000	1.072375	0.551030833	0.294687833	0.164162667	0.1584645	0.100521	0.14607433
50000000	5.949621	3.034593167	1.5774985	0.864040167	0.611053333	0.6066963	0.6183323
100000000	12.36473	6.55167533	3.307823	1.7924115	1.10729333	1.1035555	1.1529018333


Amdhal's Plot (10000000 data points)
Time
Processors
12.364736
6.551675
3.307823
1.792412
1.107293
1.103555
1.152902
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
2
4
8
16
32
64
128

Sample Readings Gustufson's:

Number of Processors	Data Points	Time (in seconds)
2	10000000	1.072375
4	20000000	1.17264933
8	40000000	1.249788833
16	80000000	1.427517167

Gustafson’s Plot

Gustafson’s Plot

Time: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6

Processors: 2, 4, 8, 16

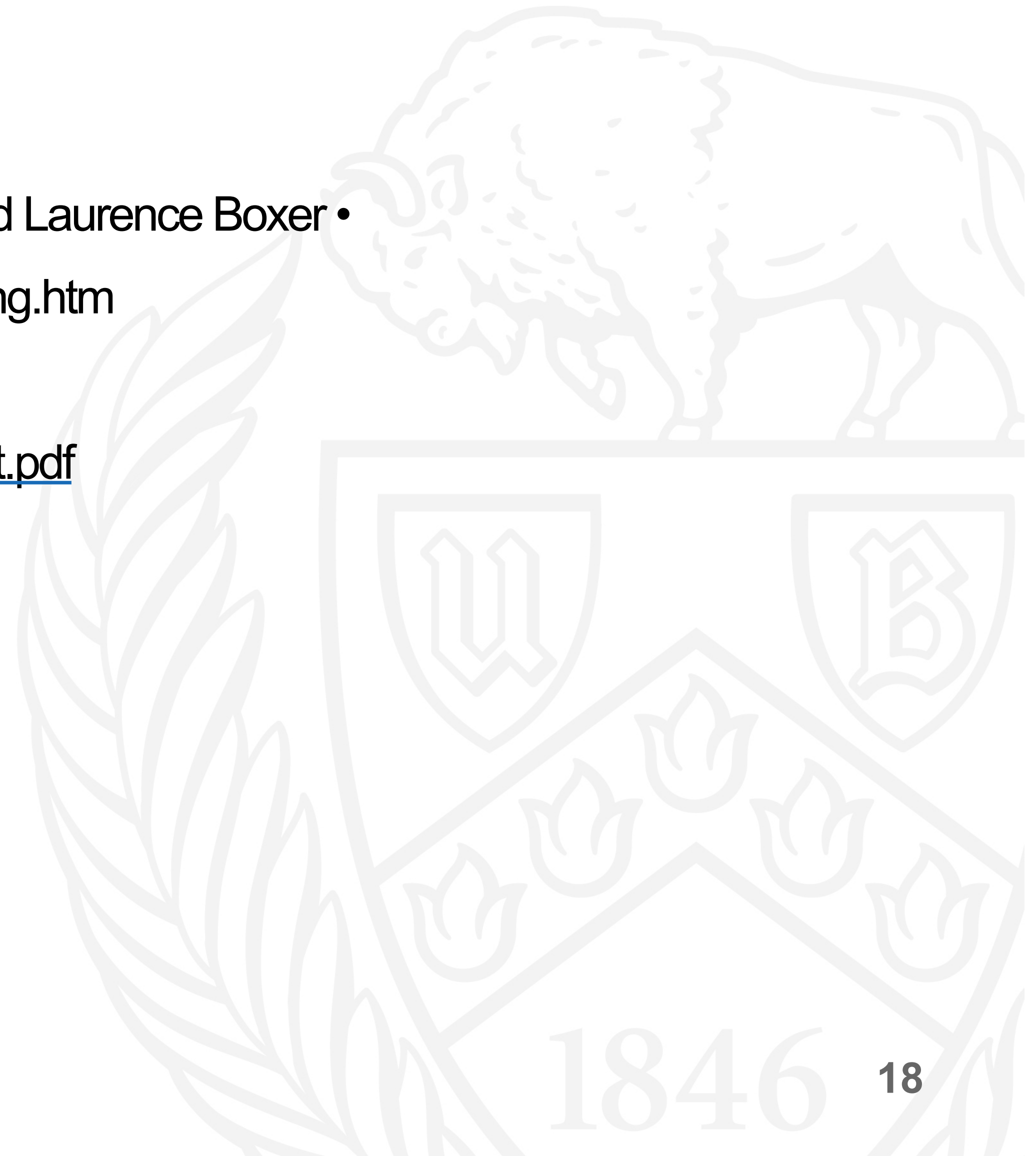
1.072375
1.172649
1.249789
1.427517

Observation:

- Computations become faster as a result of parallelization for large amount of data.
- When we keep increasing the number of processors, after certain point of time communication overhead between the processors increases and the time required for computations also increases. This can be seen from the Amdahl's plot.
- Thus, to achieve a better speed up, it is important to select optimum number of processors for any computation.

References:

- Algorithms Sequential and Parallel: A Unified Approach by Russ Miller and Laurence Boxer •
- https://www.tutorialspoint.com/parallel_algorithm/parallel_algorithm_sorting.htm
- MPI C Documentation
- http://www.cas.mcmaster.ca/~nedialk/COURSES/4f03/Lectures/quicksort.pdf



Questions?

Thank You!