HYPER QUICKSORT

By Mohd Ehtesham Shareef

Sequential Quicksort

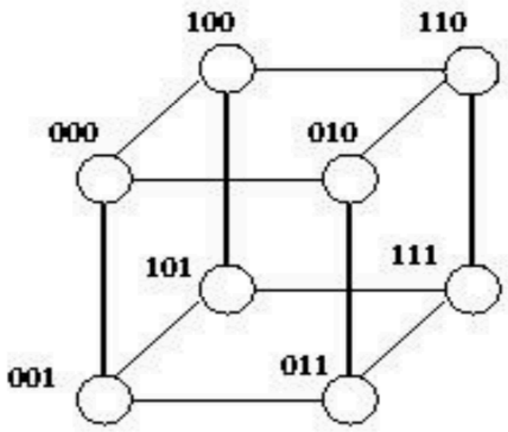
- Select median as pivot from the sample data set picked from the actual data set.
- Divide the list into two sub lists: a “low list” containing numbers smaller than the pivot, and a “high list” containing numbers larger than the pivot
- The low list and high list recursively repeat the procedure to sort themselves.
- The final sorted result is the concatenation of the sorted low list, the pivot, and the sorted high list.

Parallel Quicksort

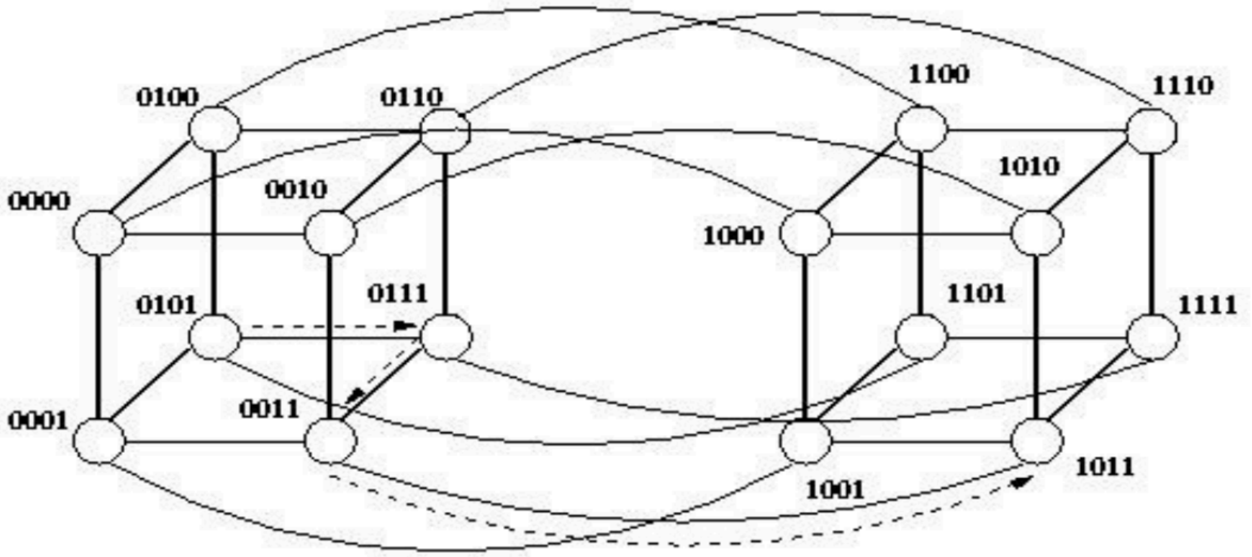
- We choose a pivot from one of the processes and broadcast it to every process.
- Each process divides its unsorted list into two lists: those smaller than (or equal) the pivot, those greater than the pivot. Each process in the upper half of the process list sends its “low list” to a partner process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

Hyper Quicksort

- Implementation of parallel quick sort on a hyper cube.
- N dimensional hypercube (number of processors is equal to 2^N).
- 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

Algorithm

- Each process 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 process 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 processes.
- The next step is different in hyper quick sort.
  - On each process, the remaining half of local list and the received half-list are merged into a sorted local list.
- Recursion within upper-half processes and lower-half processes.

Time Complexity

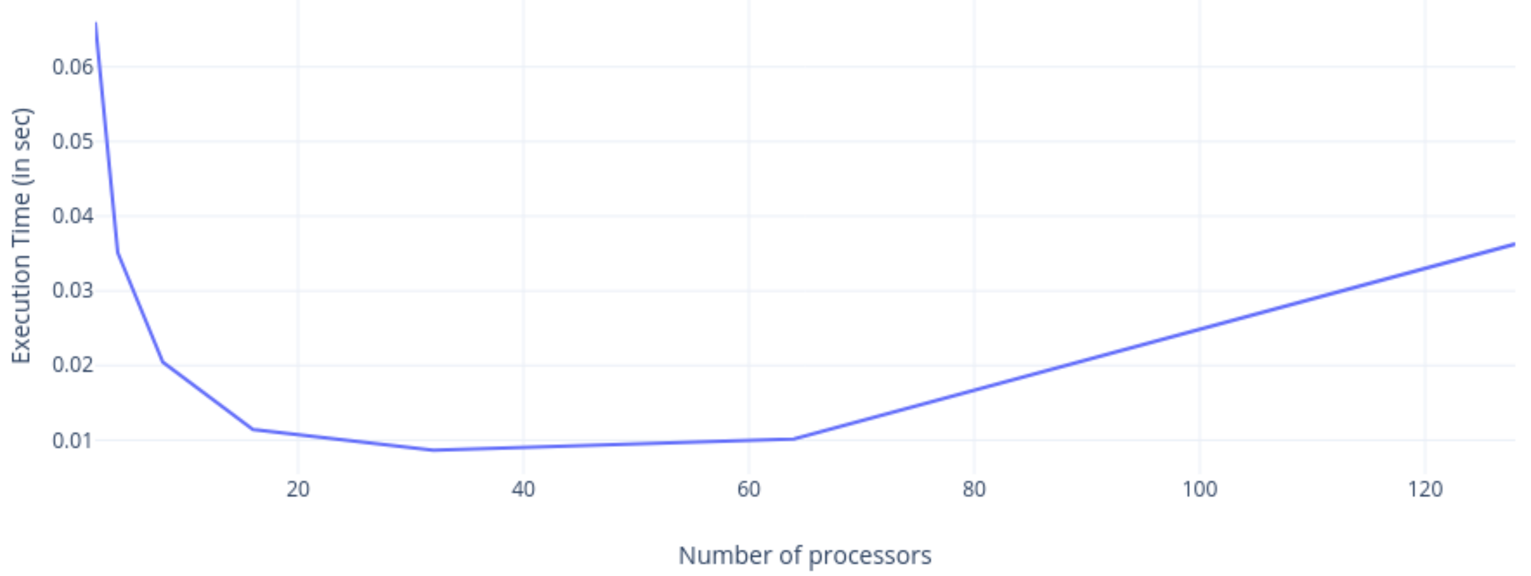
$$\Theta\left(N \log N + \frac{d(d+1)}{2} + dN\right).$$

- NlogN to sort the local list to find the median which will be the pivot.
- d(d+1)/2 for the broadcast step in step 4 of the previous slide.
- dN is the time required for exchanging and merging of the set of elements.

Results

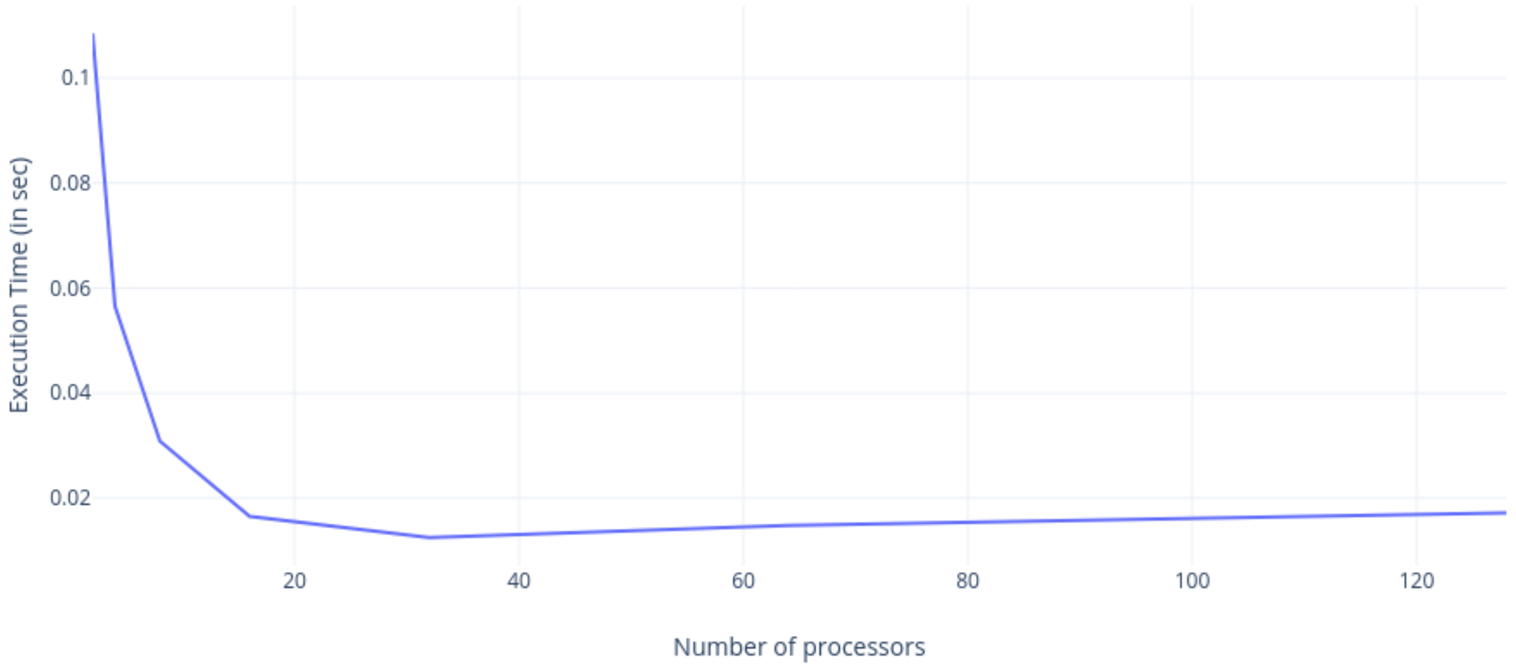
For 1/2 million values

Number of processors	Execution Time (in sec)
2	0.066013
4	0.035113
8	0.020448
16	0.011433
32	0.008648
64	0.010166
128	0.036267



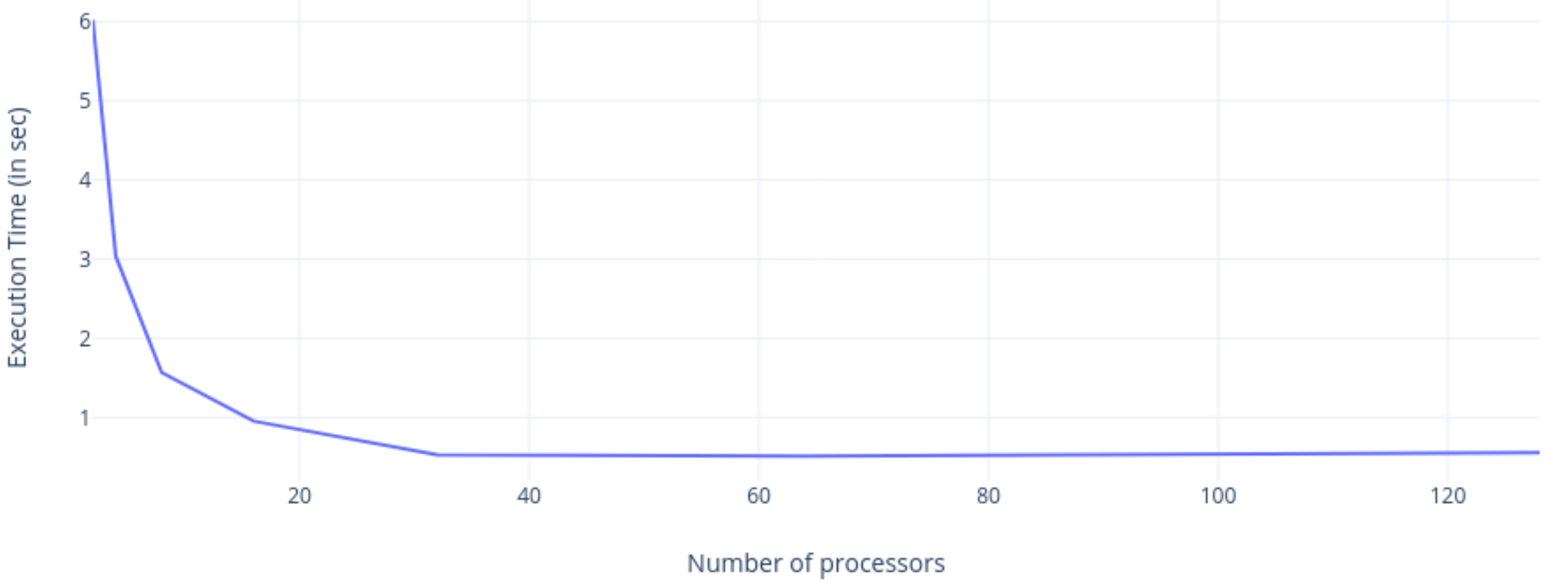
For 1 million values

Number of processors	Execution Time (in sec)
2	0.108501
4	0.056542
8	0.030841
16	0.016540
32	0.012461
64	0.014786
128	0.017203



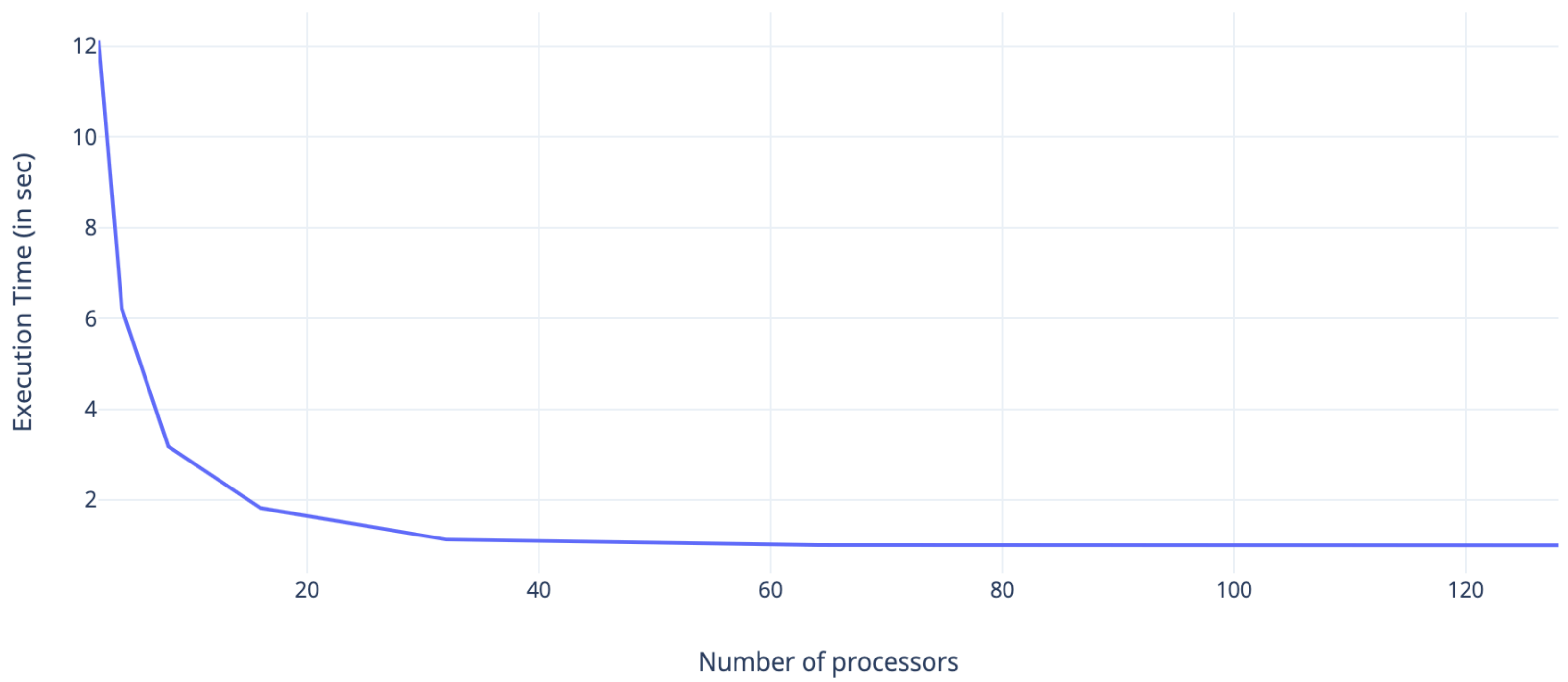
For 50 million values

Number of processors	Execution Time (in sec)
2	6.024641
4	3.035160
8	1.568181
16	0.953733
32	0.526780
64	0.511470
128	0.555545



For 100 million values

Number of processors	Execution Time (in sec)
2	12.129532
4	6.211917
8	3.180447
16	1.818400
32	1.127691
64	1.003973
128	1.001266



Observations

- Computations become faster as a result of parallelization for large amounts of data.
- Very high communication overhead as the number of processors increase after a certain point.
- In order to achieve better performance its important to identify the optimal number of processors that would be required for any given 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
- https://pdfs.semanticscholar.org/16f2/590017d1cf27f60d869366ce281eb5e00802.pdf
- MPI C Documentation

Thank You