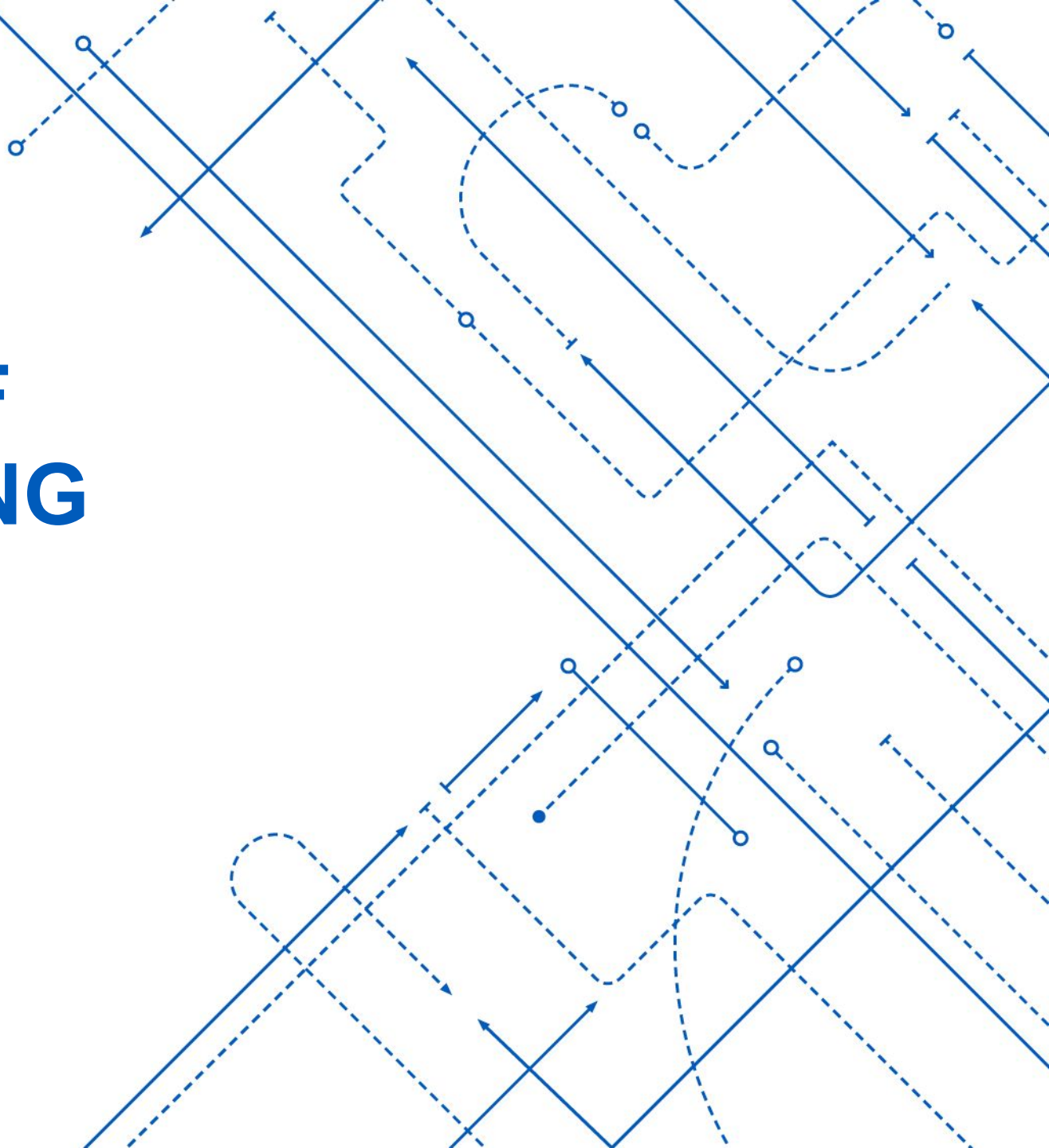
PARALLEL IMPLEMENTATION OF BITONIC SORT – USING MPI

-Kadambare Jayandran

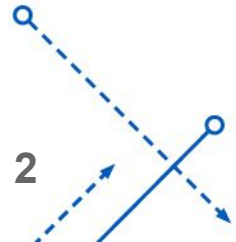
(kjayandr)

University at Buffalo The State University of New York



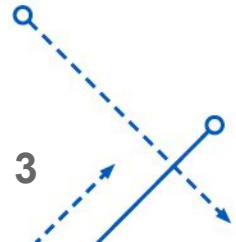
Sorting

- Arrange an unordered collection of items into a meaningful order.
- Efficient sorting is important for optimizing the efficiency of other algorithms (such as search) that require input data to be in sorted lists. Sorting is also often useful for canonicalizing data and for producing human-readable output.
- Sorting can be comparison-based or non comparison-based.
- The fundamental operation of comparison-based sorting is compare-exchange.

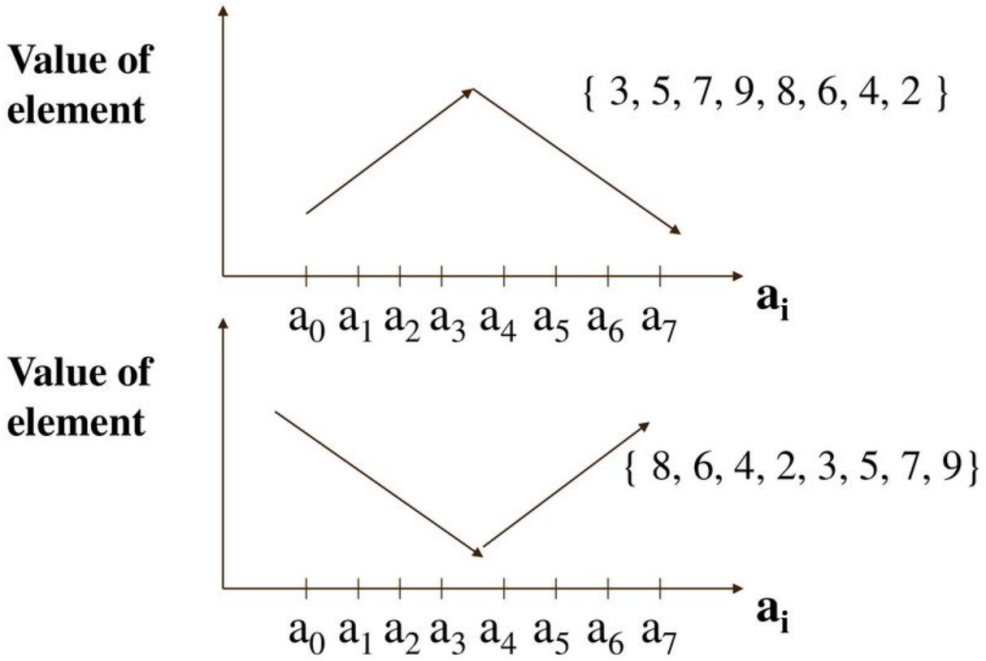


Why Parallelize?

- Sequential algo better time complexity eg:MergeSort is O(n*Log n), Bitonic sort O(n Log ^2n)
- Parallel computing saves time, allowing the execution of applications in a shorter wall-clock time.
- Solve Larger Problems in a short point of time. Compared to serial computing, parallel computing is much better suited for modeling, simulating and understanding complex, real-world phenomena.

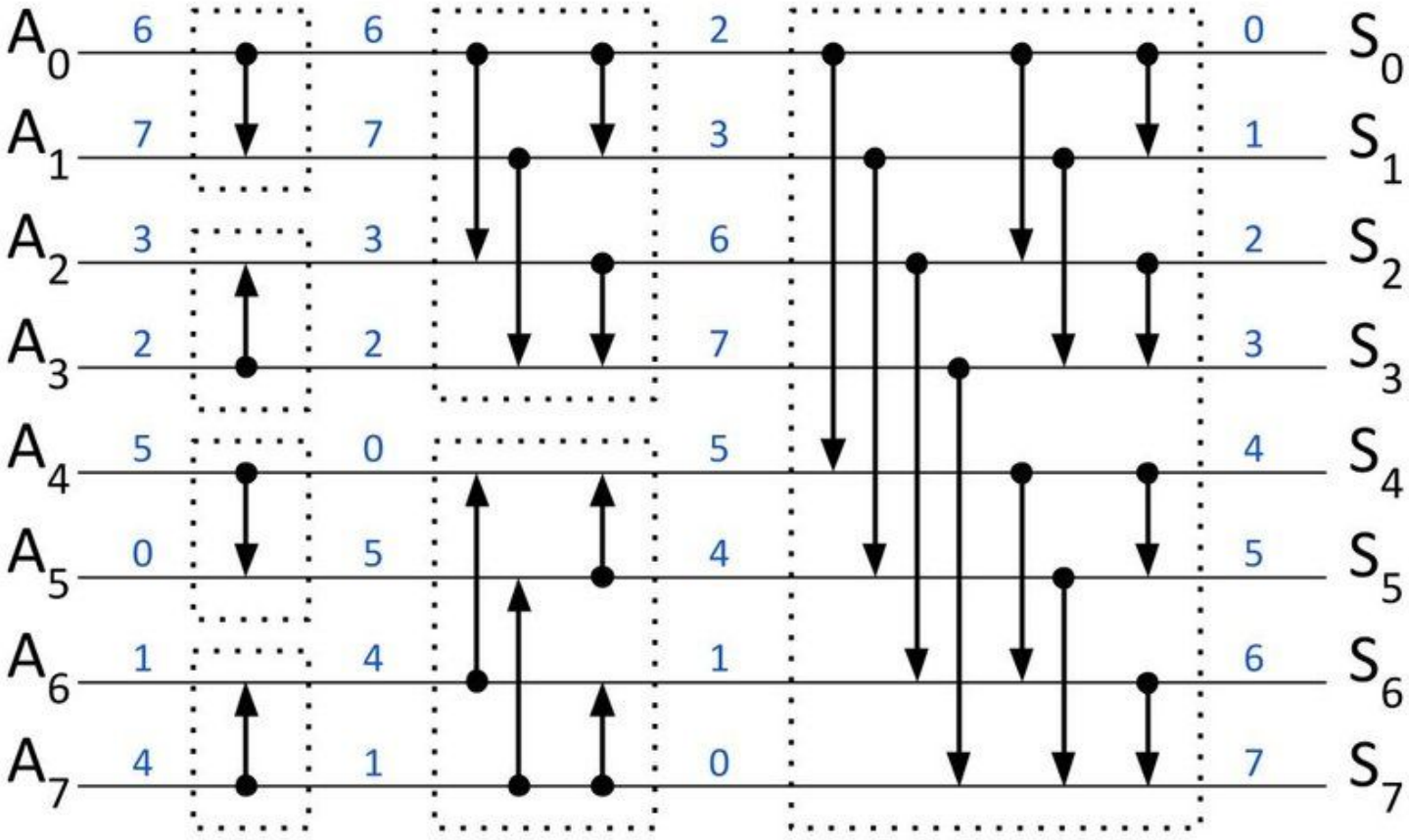


Bitonic sort

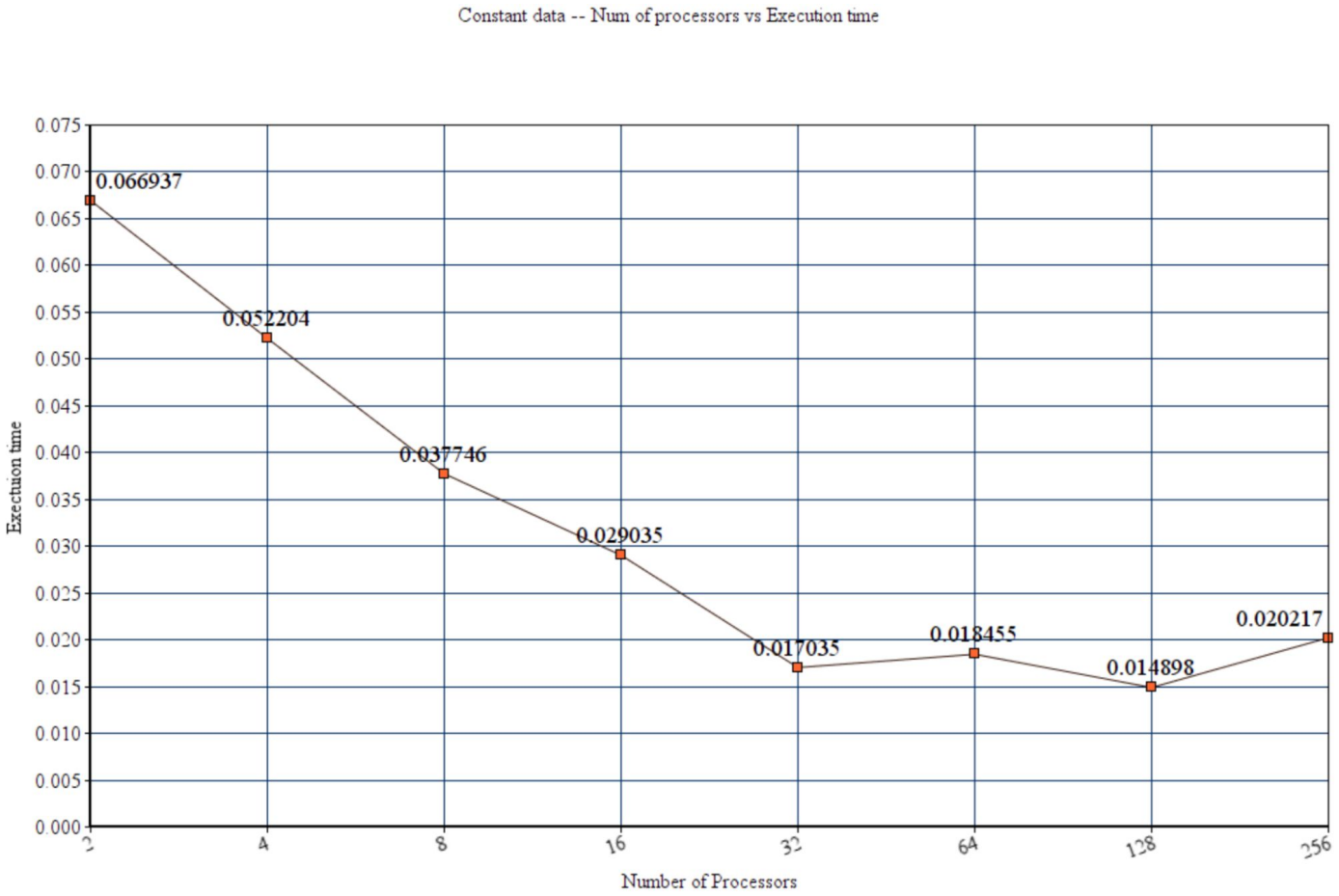


- A sequence a = (a1, a2, . . ., ap) of p numbers is said to be bitonic if and only if • a1 ≤ a2 ≤ . . . ≤ ak≥ . . . ≥ ap, for some k, 1 < k < p, or
- • a1 ≥ a2 ≥ . . . ≥ ak≤ . . . ≤ ap, for some k, 1 < k < p, or
- ‘a’ can be split into two parts that can be interchanged to give either of the cases.
- A sequence is bitonic if it monotonically increases and then monotonically decreases, or if it can be circularly shifted to monotonically increase and then monotonically decrease.

Bitonic sort



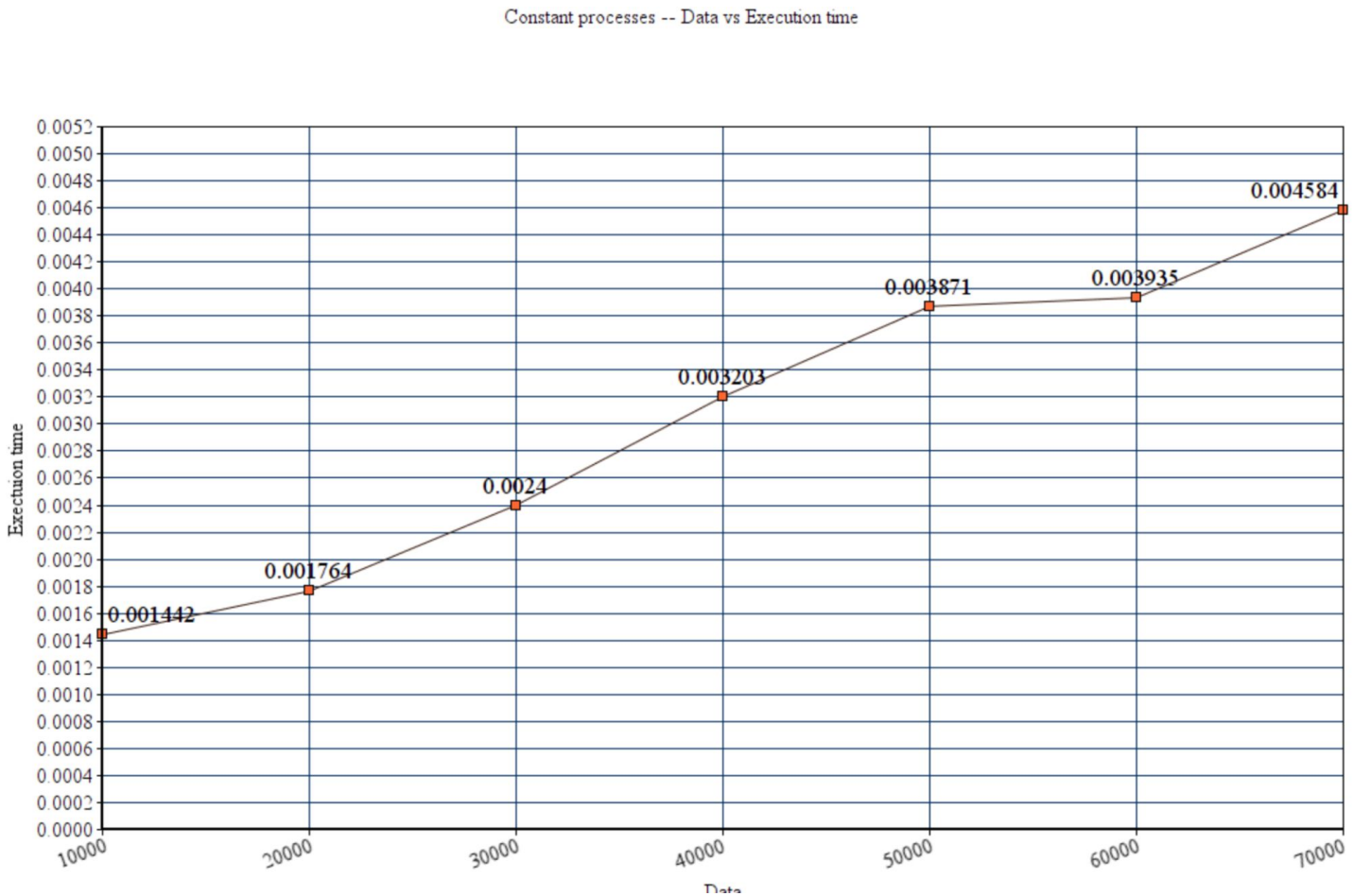
Constant Data



CONSTANT DATA - 320000

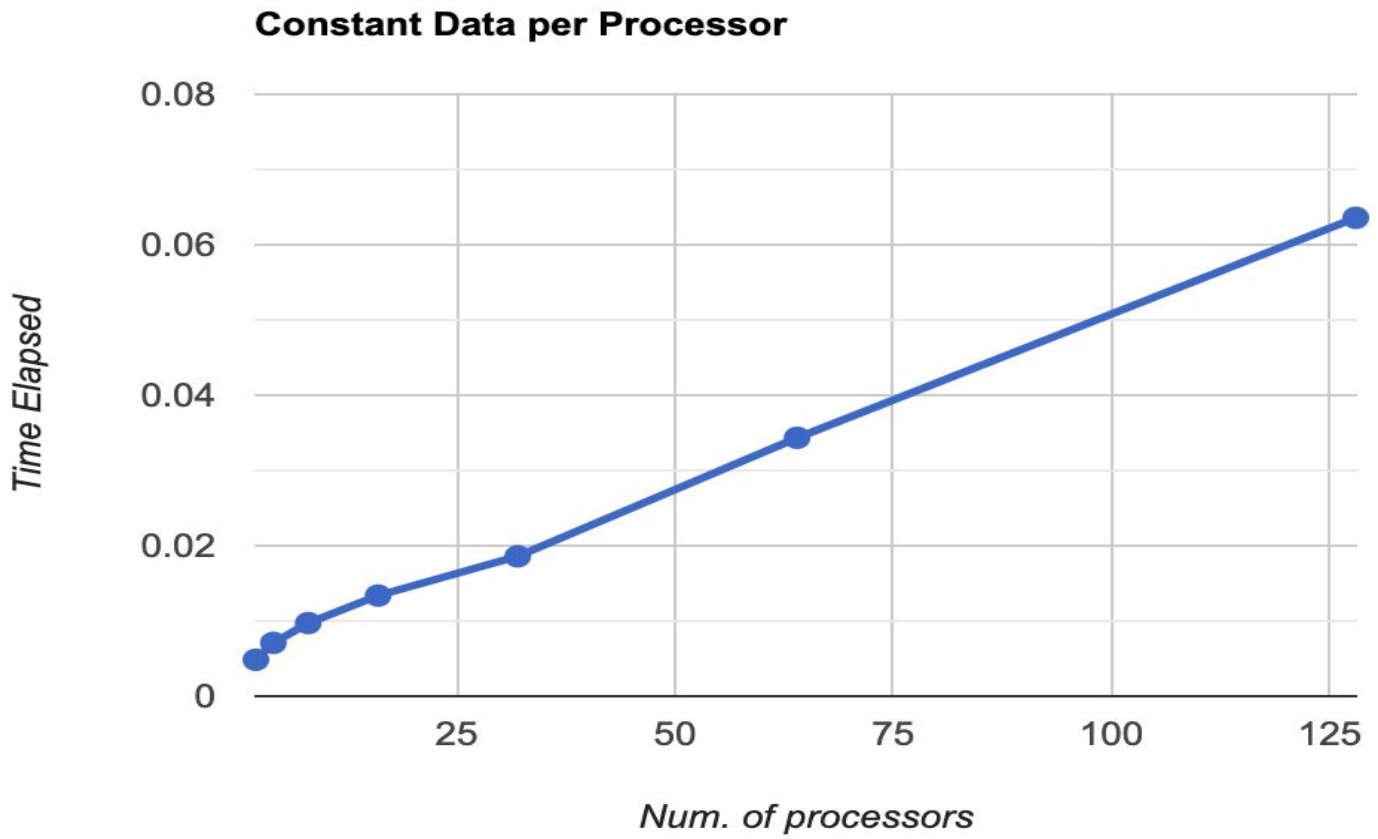
Processes	Time(secs)
2	0.066937
4	0.052204
8	0.037746
16	0.029035
32	0.017035
64	0.018455
128	0.014898
256	0.020217

Constant Number of Processes



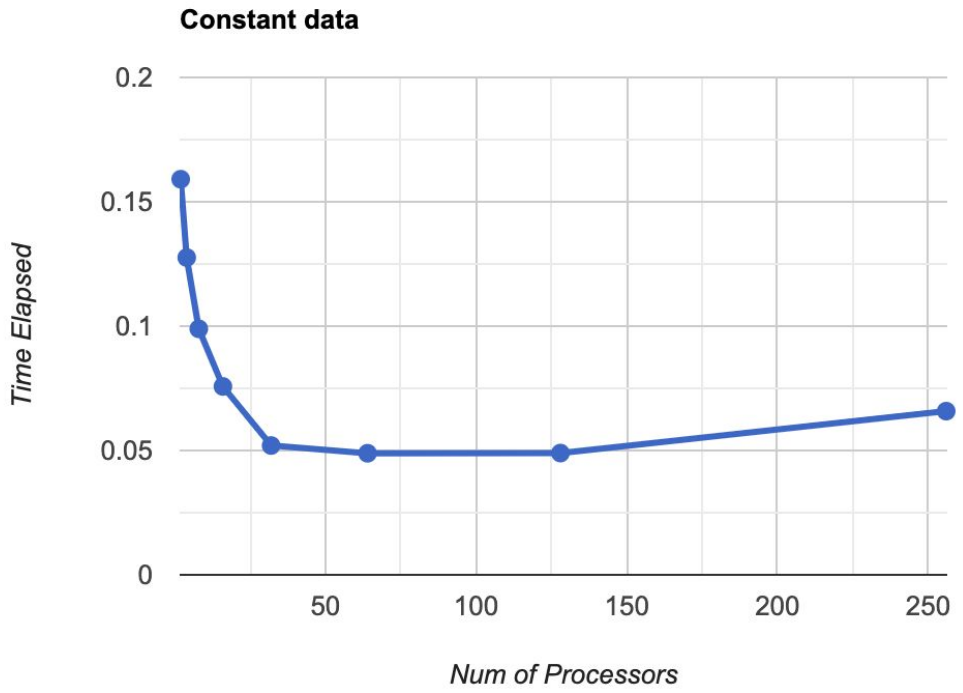
Data	Time(secs)
10000	0.001442
20000	0.001764
30000	0.0024
40000	0.003203
50000	0.003871
60000	0.003935
70000	0.004584

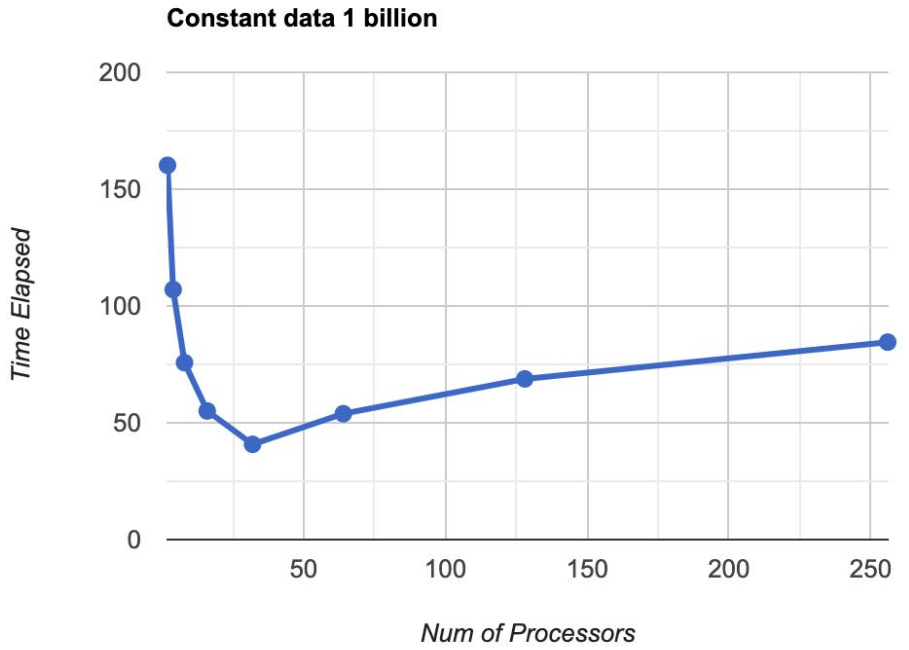
Constant Number of Processes



Data	Processor	Time(secs)
20000	2	0.004842
40000	4	0.007089
80000	8	0.009718
160000	16	0.013374
320000	32	0.01859
640000	64	0.034345
1280000	128	0.063585

Constant data for 1 million - 1 billion



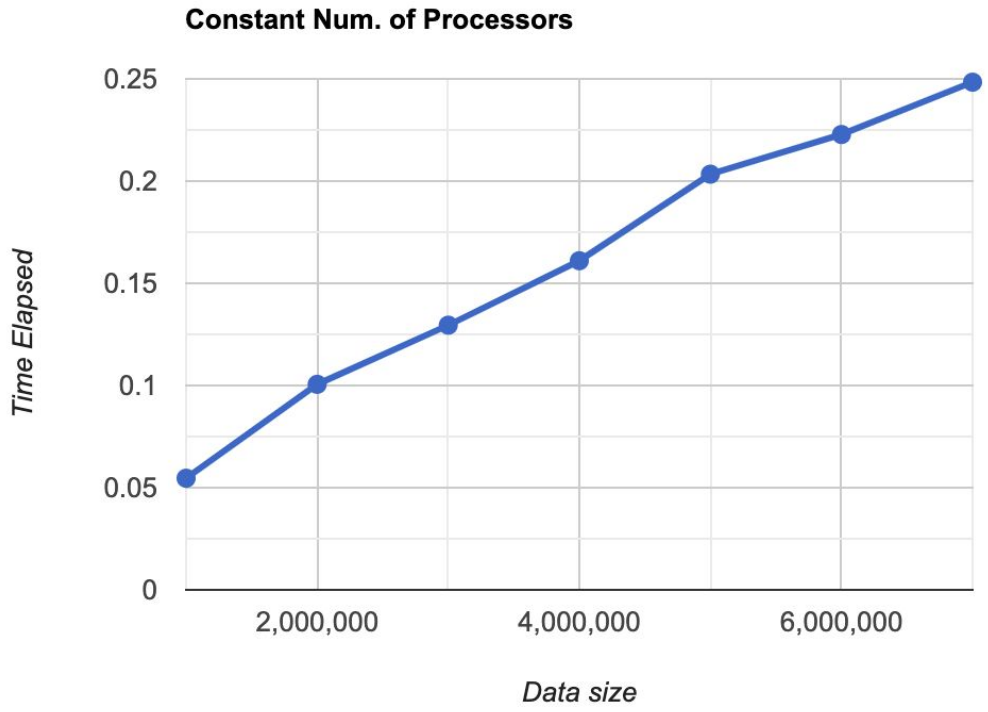


Constant data - 1 million & 1 billion

No of Processors	Time Elapsed (secs)
2	0.159068
4	0.127574
8	0.098923
16	0.075753
32	0.051997
64	0.048886
128	0.048964
256	0.065821

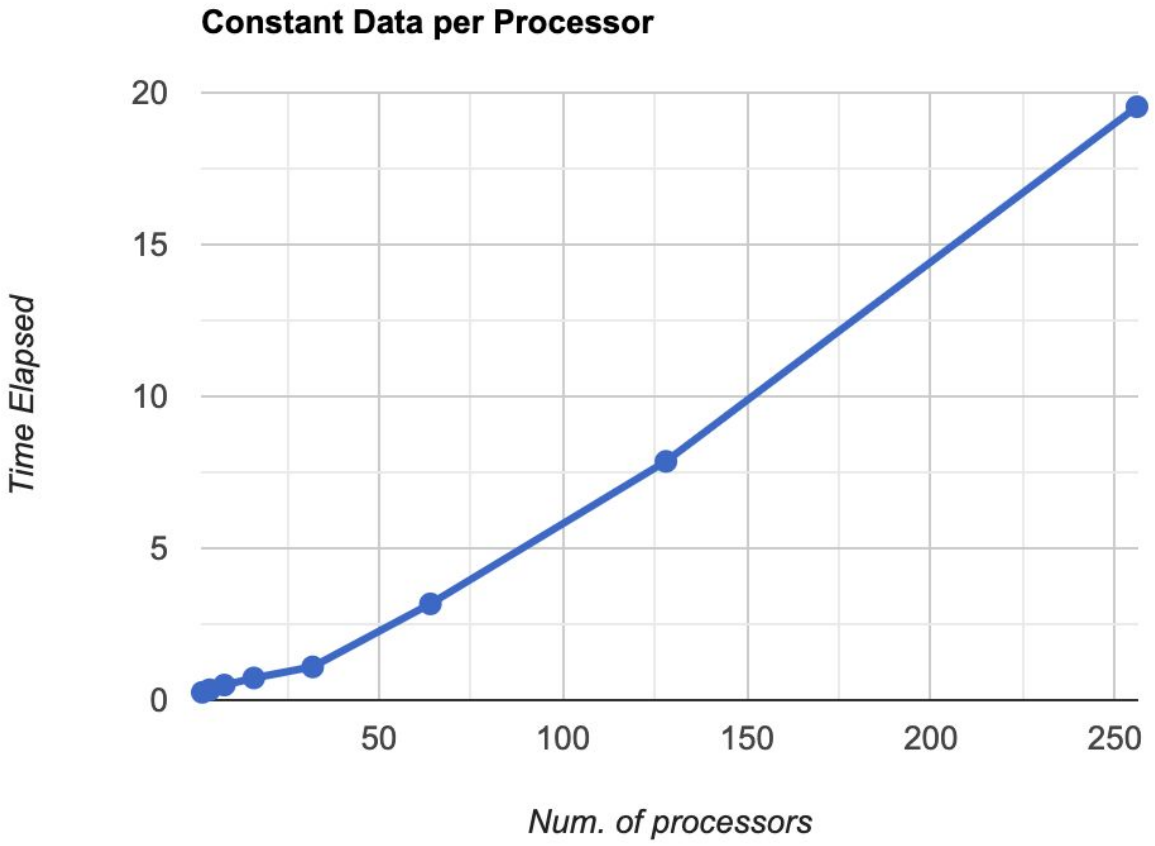
No of Processors	Time Elapsed (secs)
2	160.213859
4	107.03612
8	75.726802
16	55.047718
32	40.765924
64	53.894162
128	68.790941
256	84.524898

Constant Num. of Processors -32



Datasize	Time Elapsed(secs)
1000000	0.054629
2000000	0.100569
3000000	0.129495
4000000	0.161037
5000000	0.203386
6000000	0.222796
7000000	0.248393

Constant Data Per Processor



Number of Processors	Time Elapsed(secs)
2	0.257144
4	0.341612
8	0.496994
16	0.730705
32	1.094706
64	3.169772
128	7.86138
256	19.538398

1 million data per processor

Conclusion

- There is a steady growth rate in the amount of time taken for execution when the data per processor is constant.
- A similar trend is observed when the amount of data is increased but the number of processor is constant.
- In the case of constant data and increased number of processors we see that the least time is taken when we use 32 processors. This implies that though we have more computational power, when the number of processors is too much there is also communication overhead.

References

https://ubccr.freshdesk.com/support/solutions/folders/13000001591

https://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiintro/ppframe.htm

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

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

http://condor.cc.ku.edu/~grobe/docs/intro-MPI-C.shtml#:~:text=MPI%20is%20a%20library%20of,programs%20in%20C%20or%20Fortran77.&text=It%20is%20a%20library%20that,exchange%20information%20among%20these%20processes