PARALLEL MERGE SORT USING MPI

Instructor: Dr. Russ Miller

Prepared by: Ashwin Panditrao Jadhav

University at Buffalo The State University of New York

Agenda

- Sequential Merge Sort Algorithm
- Sequential Merge Sort Analysis
- Parallel Merge Sort Algorithm
- Experimentation in CCR
- Parallel Algorithm Results and Analysis
- Challenges and Learnings
- Conclusion
- References

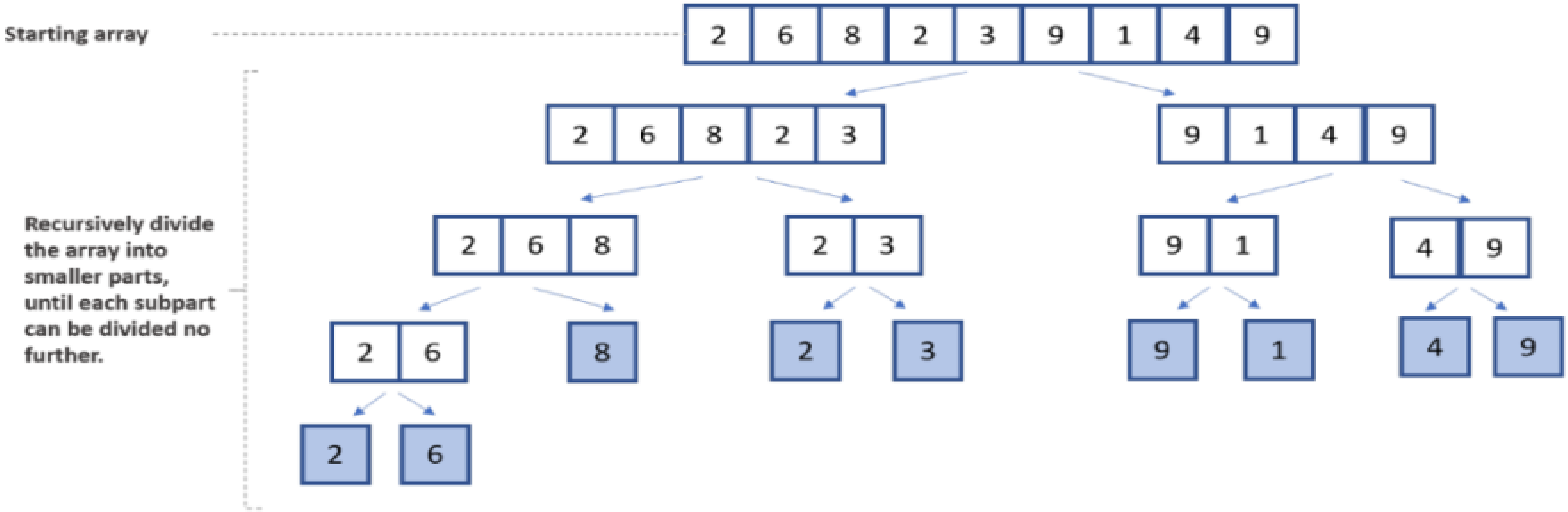
University at Buffalo The State University of New York

SEQUENTIAL MERGE SORT

Sequential Merge Sort Algorithm (Divide)

Divide Step

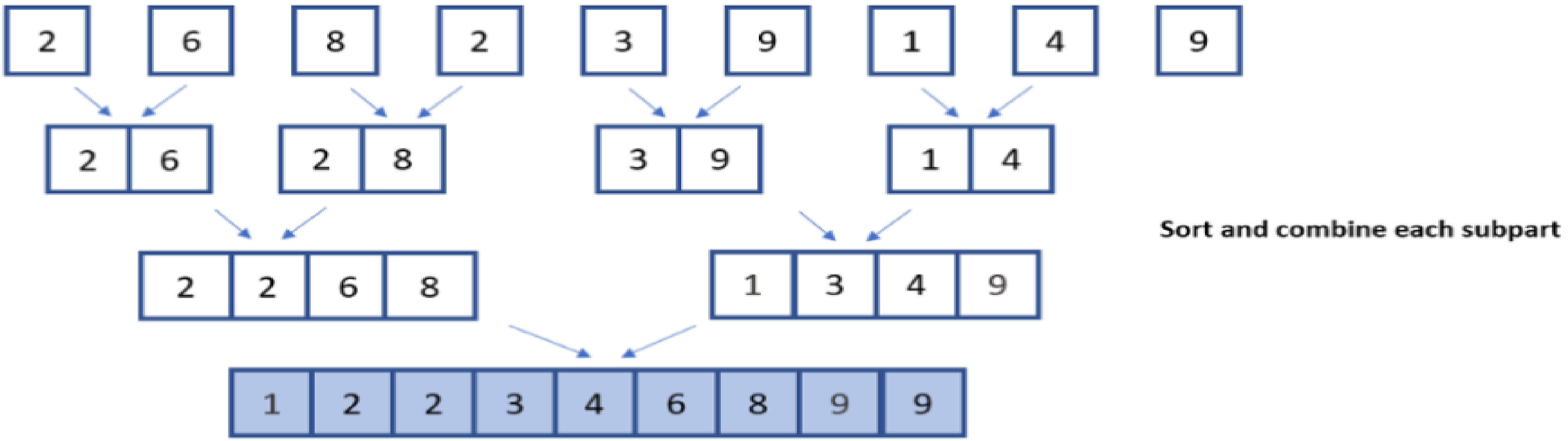
- We first divide the array in halves.
- Then each of these two lists are further broken down in the same manner.
- Until they can no longer be divided.
- Which leaves only one element at the end.



Sequential Merge Sort Algorithm (Conquer & Combine)

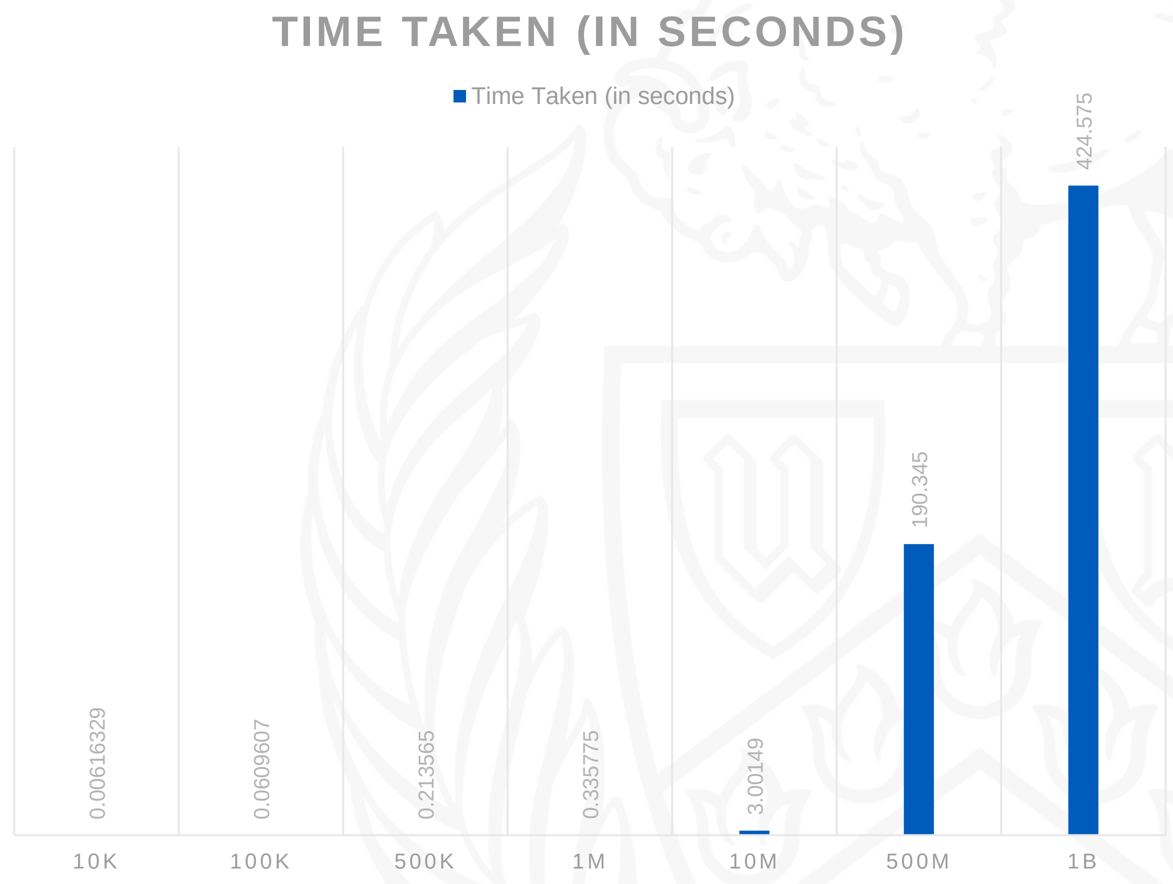
Conquer & Combine Step

- We one by one take each element from left and right like in a loop.
- We do a comparison between these two elements..
- The smaller element is appended to the list first.
- Then the pointer of the list whose element is appended is incremented..



Sequential Merge Sort Analysis

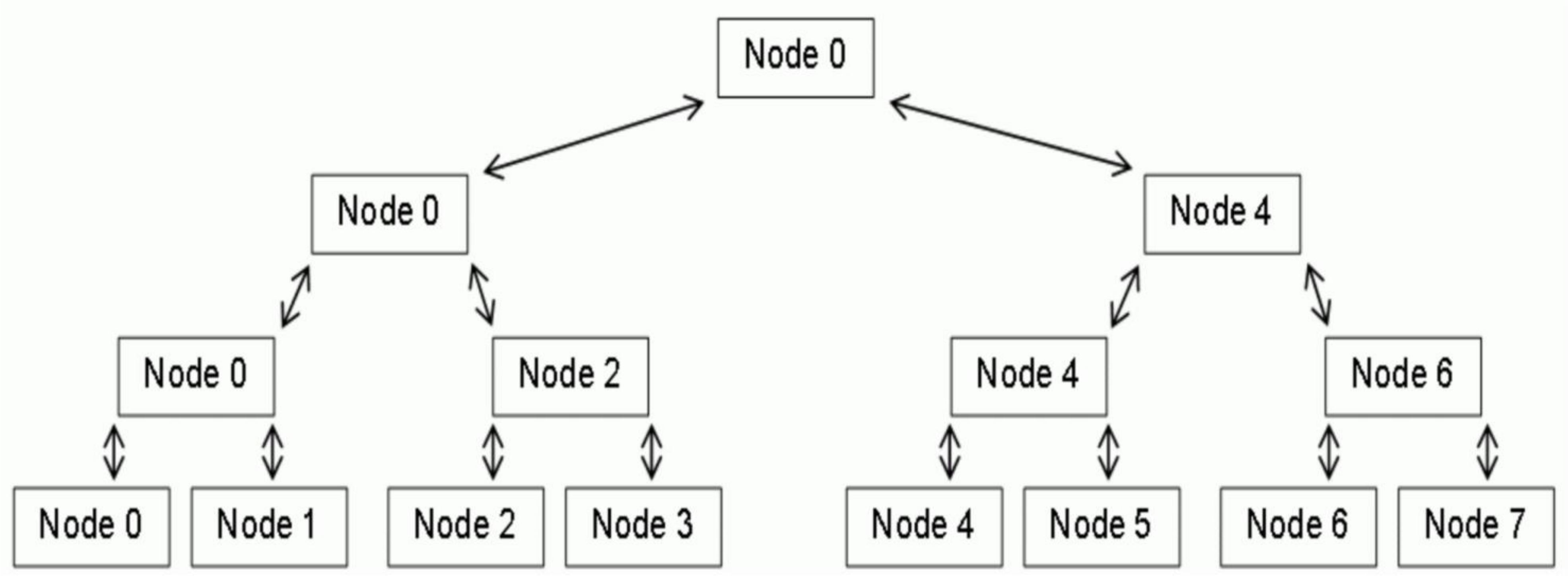
Data Size	Time Taken (in seconds)
10000(10K)	0.00616329
100000(100K)	0.0609607
500000(500K)	0.213565
1000000(1M)	0.335775
10000000(10M)	3.00149
500000000(500M)	190.345
1000000000(1B)	424.575



M: Million
B: Billion

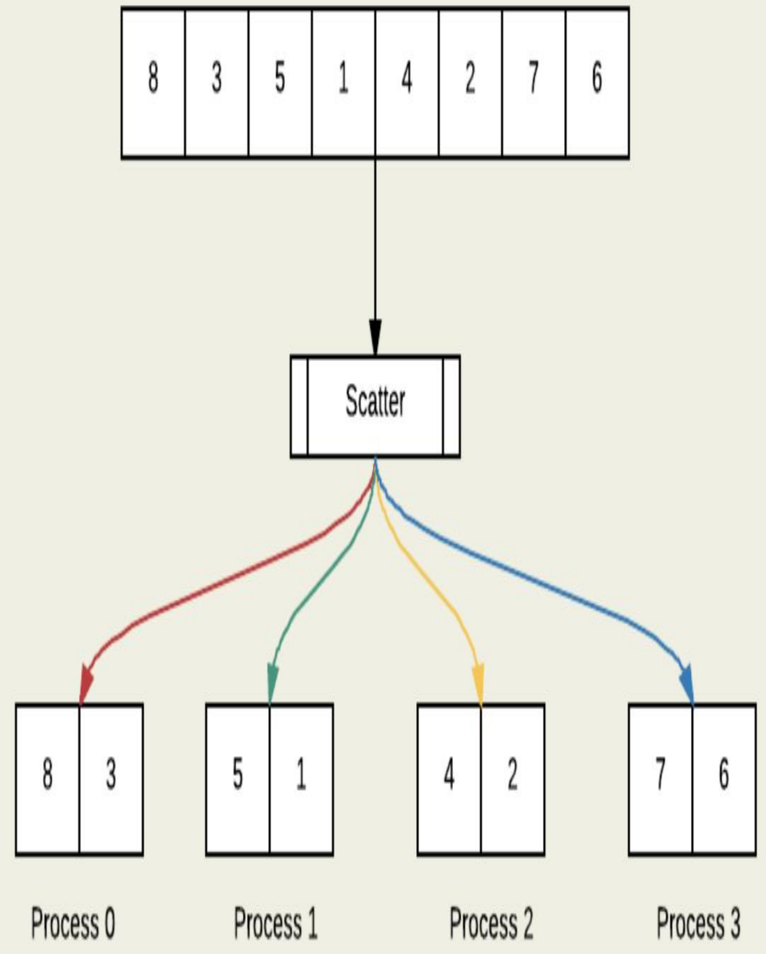
PARALLEL MERGE SORT ALGORITHM

Proposed Parallel Merge Sort Algorithm



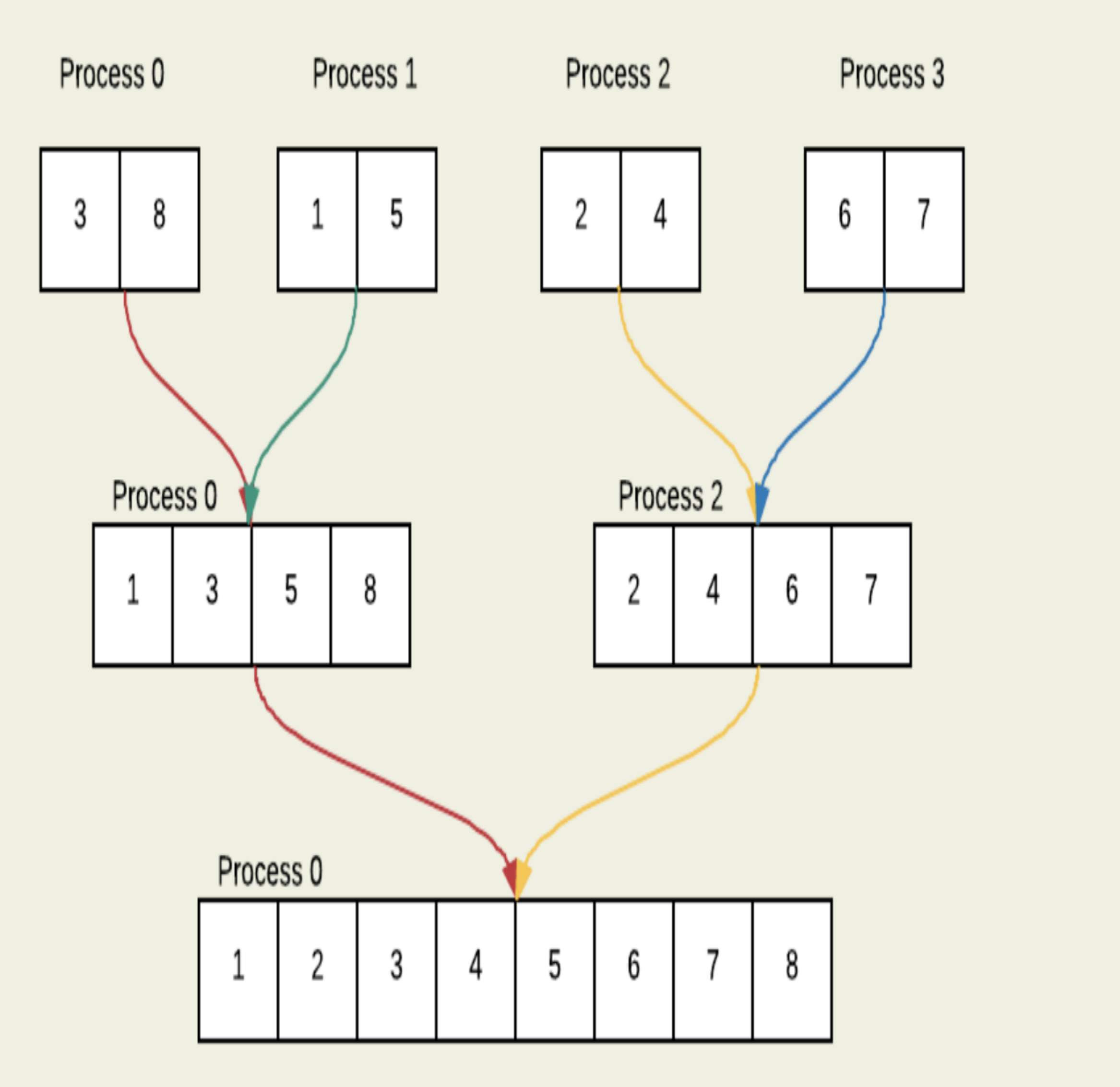
Proposed Parallel Merge Sort Algorithm(Divide)

1. Node having the rank 0 is the host node. It gets the entire dataset and computes the height of node.
2. Host node with rank 0, initiates the parallel merge operation.
3. For internal nodes (height > 0) and node 0:
   a) Divide the data in half and send the right half to the right child.
   b) Recursively call parallel merge operation for the left half on the same node.



Proposed Parallel Merge Sort Algorithm(Conquer)

1. Now, receive the sorted data from right child.
2. Merge the sorted left and right child halves.
3. If it is a leaf node, just do internal sorting.
4. Send the sorted data to parent node.
5. Finally, node 0 will have the entire sorted result.



Experimentation in CCR: SBATCH script

```
#!/bin/bash

#SBATCH --clusters=ub-hpc
#SBATCH --partition=general-compute
#SBATCH --qos=general-compute
#SBATCH --exclusive
#SBATCH --mem=64000
#SBATCH --output=%jp2.stdout
#SBATCH --error=%jp2.stderr

####### CUSTOMIZE THIS SECTION FOR YOUR JOB
#SBATCH --job-name="ajms"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --time=01:00:00

module load gcc
module load intel-mpi/2019.5
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so

srun --mpi=pmi2 a 10000
srun --mpi=pmi2 a 100000
srun --mpi=pmi2 a 500000
srun --mpi=pmi2 a 1000000
srun --mpi=pmi2 a 100000000
srun --mpi=pmi2 a 500000000
srun --mpi=pmi2 a 1000000000

```

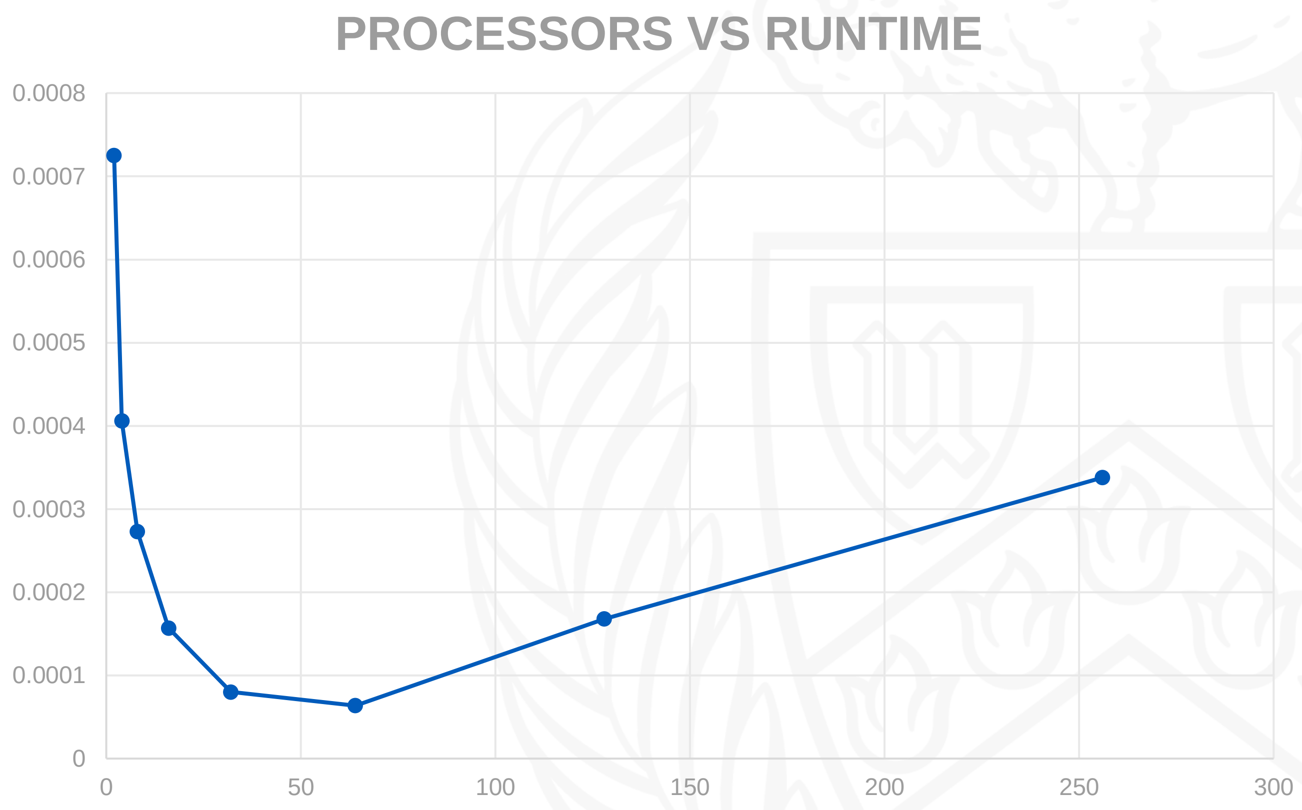
Experiments:

- For some constant data size, plotted sorting time vs number of processors.
    - Tested for 7 different data sizes: 10K, 100K, 500K, 1M, 10M, 500M, 1B
    - For number of processors: 2, 4, 8, 16, 32, 64, 128, 256
- Plotted speed-up of parallel approach vs the sequential approach.
- Also, shown and plotted sorting time for a particular data size running on different number of processors.

Runtime Vs Number of Processors (keeping data size constant)

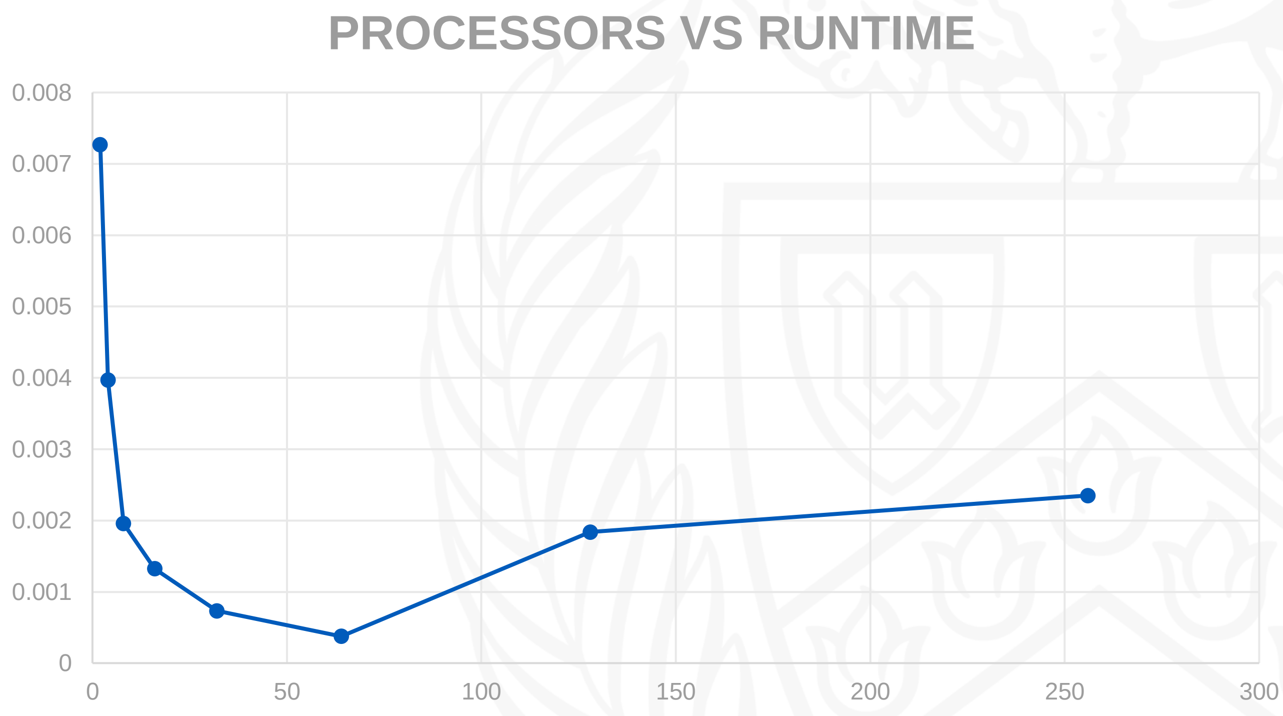
Runtime Vs Number of Processors for Data Size: 10000(10K)

# of Processors	Time Taken (in seconds)
2	0.000725
4	0.000406
8	0.000273
16	0.000157
32	0.0000802
64	0.0000637
128	0.000168
256	0.000338



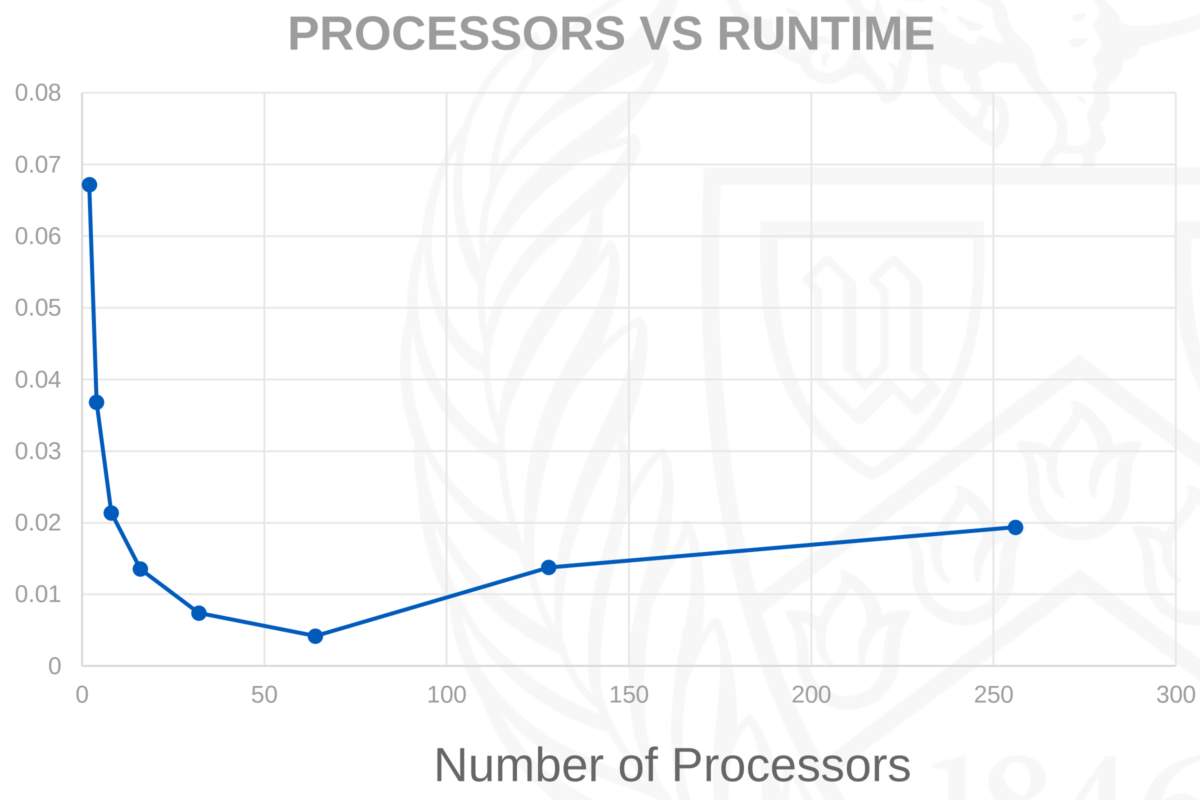
Runtime Vs Number of Processors for Data Size: 100000(100K)

# of Processors	Time Taken (in seconds)
2	0.00727
4	0.003968
8	0.001958
16	0.001325
32	0.000734
64	0.0003761
128	0.001838
256	0.002351



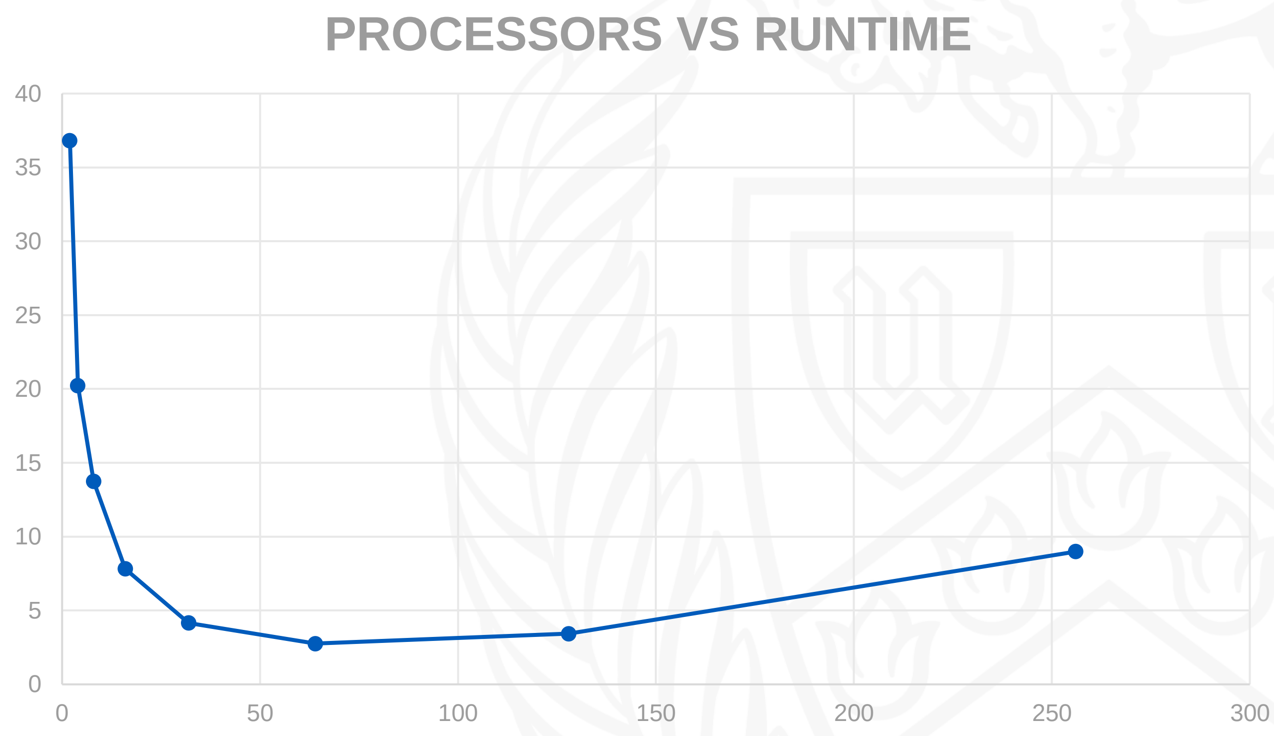
Runtime Vs Number of Processors for Data Size: 1000000 (1M)

# of Processors	Time Taken (in seconds)
2	0.067182
4	0.036808
8	0.02138
16	0.01354
32	0.007387
64	0.004175
128	0.013749
256	0.019365



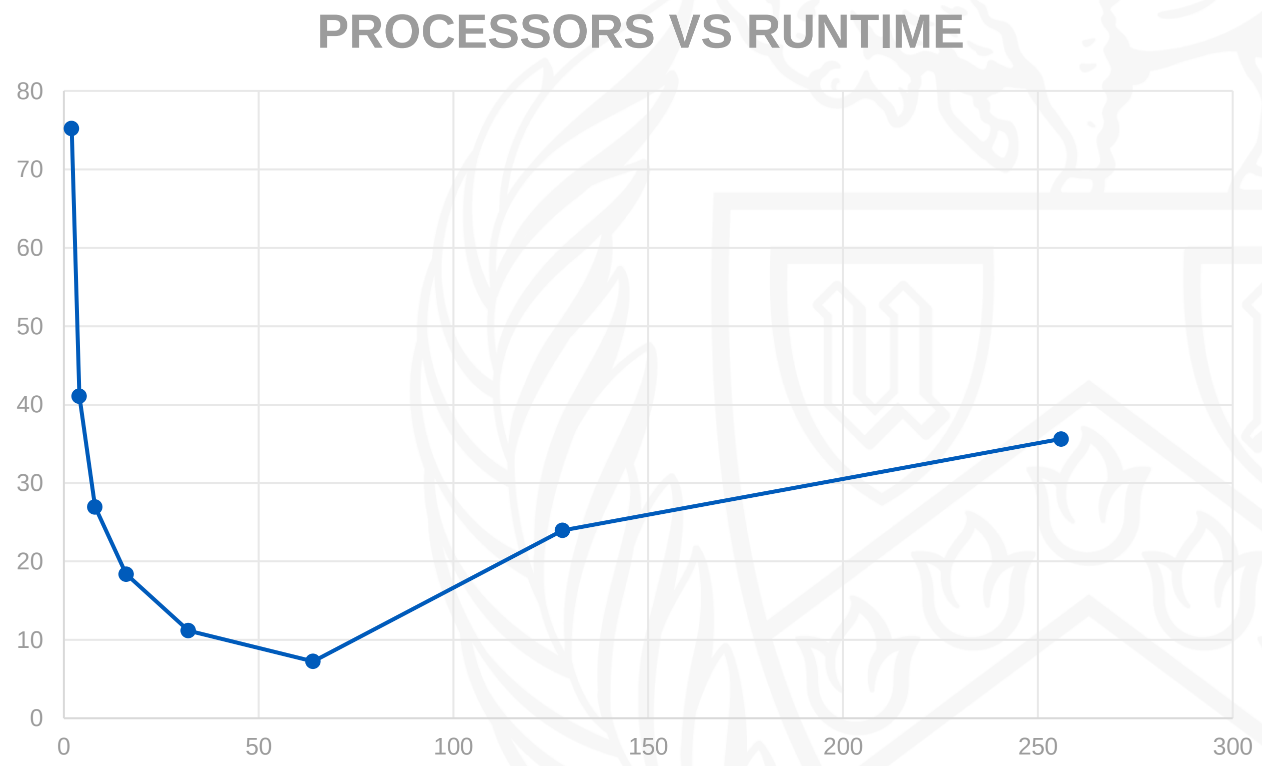
Runtime Vs Number of Processors for Data Size: 500000000 (500M)

# of Processors	Time Taken (in seconds)
2	36.810129
4	20.218637
8	13.750573
16	7.810129
32	4.1484
64	2.752187
128	3.427604
256	8.982689



Runtime Vs Number of Processors for Data Size: 1000000000 (1B)

# of Processors	Time Taken (in seconds)
2	75.223702
4	41.084074
8	26.964285
16	18.385541
32	11.179744
64	7.262068
128	23.951129
256	35.620181

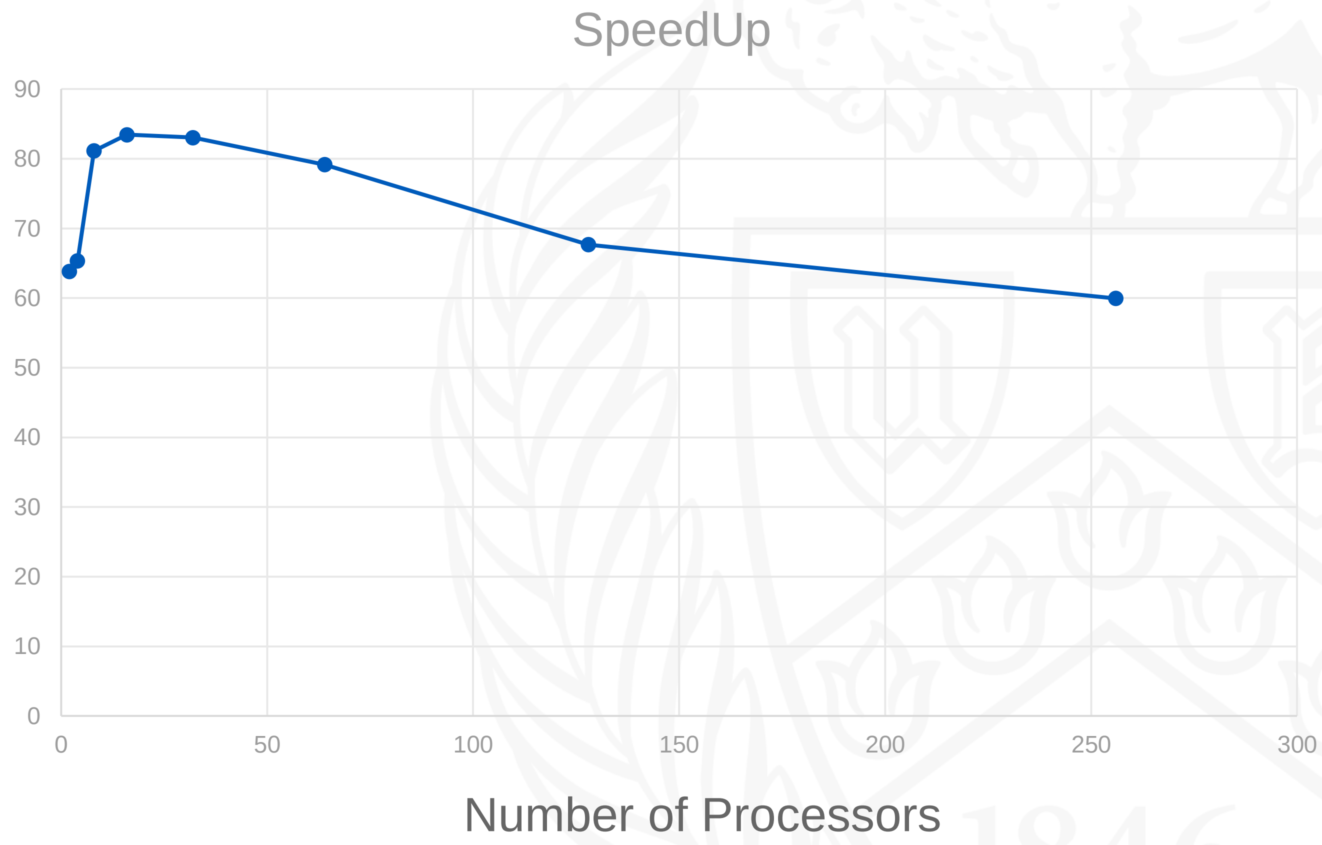


Speedup: Compared to sequential time

University at Buffalo The State University of New York

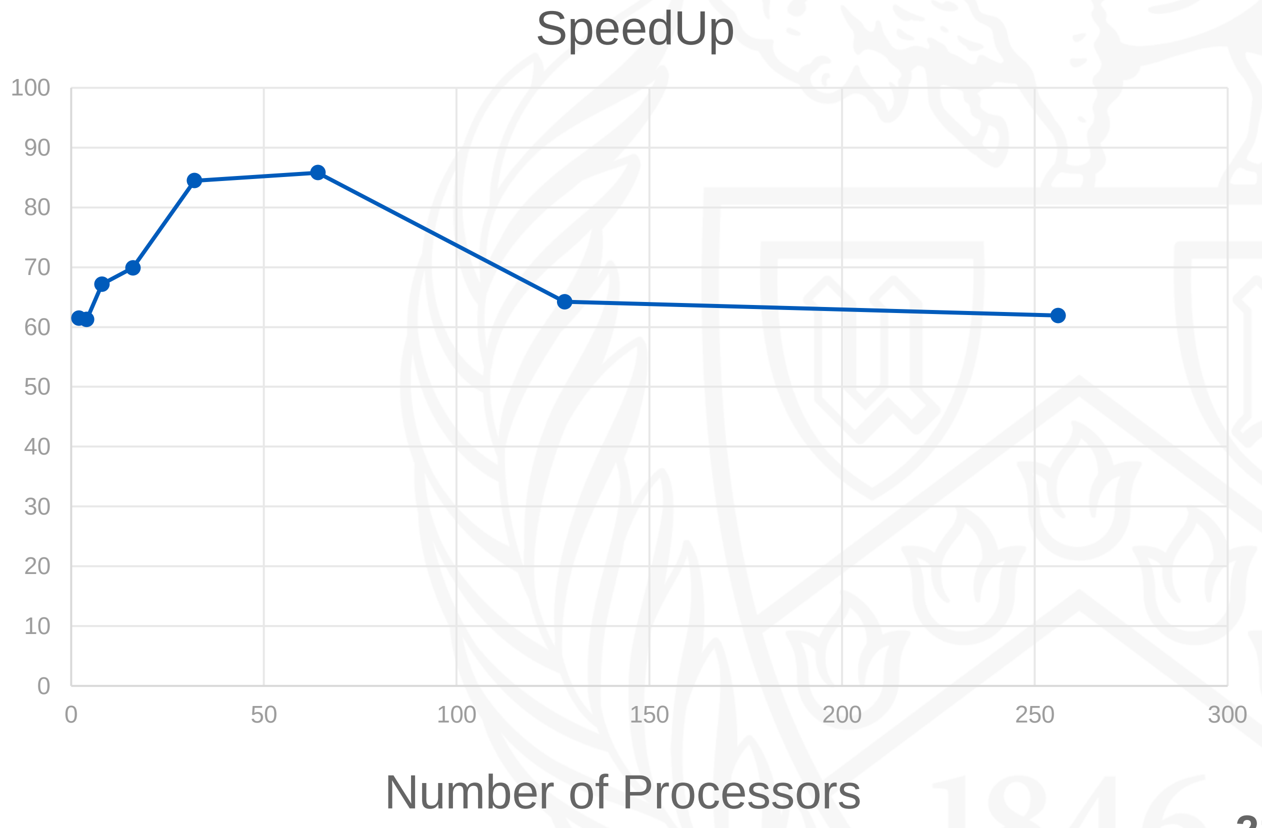
Speedup for Data Size: 100000 (100K)

# of Processors	Speedup
2	63.8524
4	65.36307
8	81.13416
16	83.46807
32	83.05272
64	79.16208
128	67.68661
256	59.968949



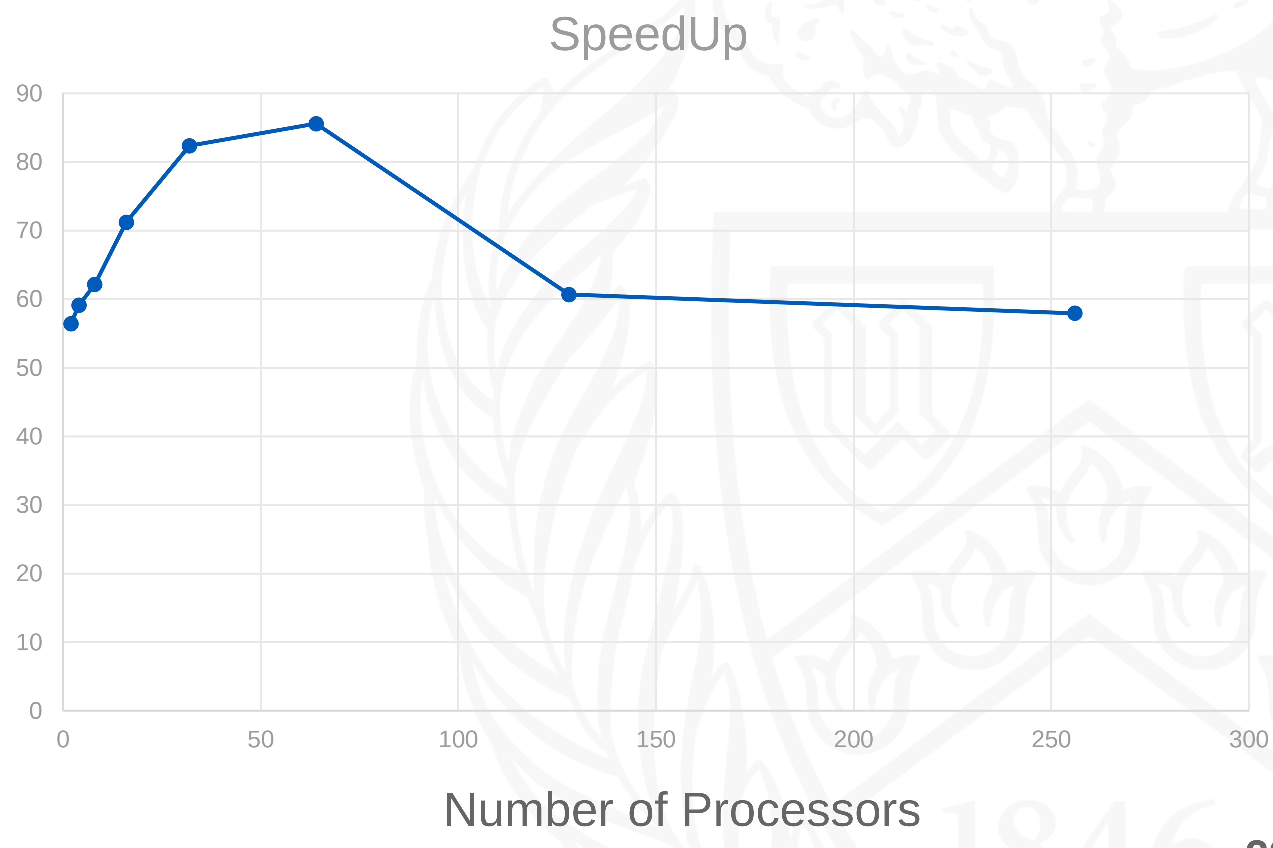
Speedup for Data Size: 1000000 (1M)

# of Processors	Speedup
2	61.49799
4	61.233758
8	67.150509
16	69.87446
32	84.454548
64	85.8042514
128	64.21776
256	61.92718



Speedup for Data Size: 1000000000 (1B)

# of Processors	Speedup
2	56.441651
4	59.103342
8	62.15745
16	71.230928
32	82.379771
64	85.584647
128	60.67217
256	57.950709

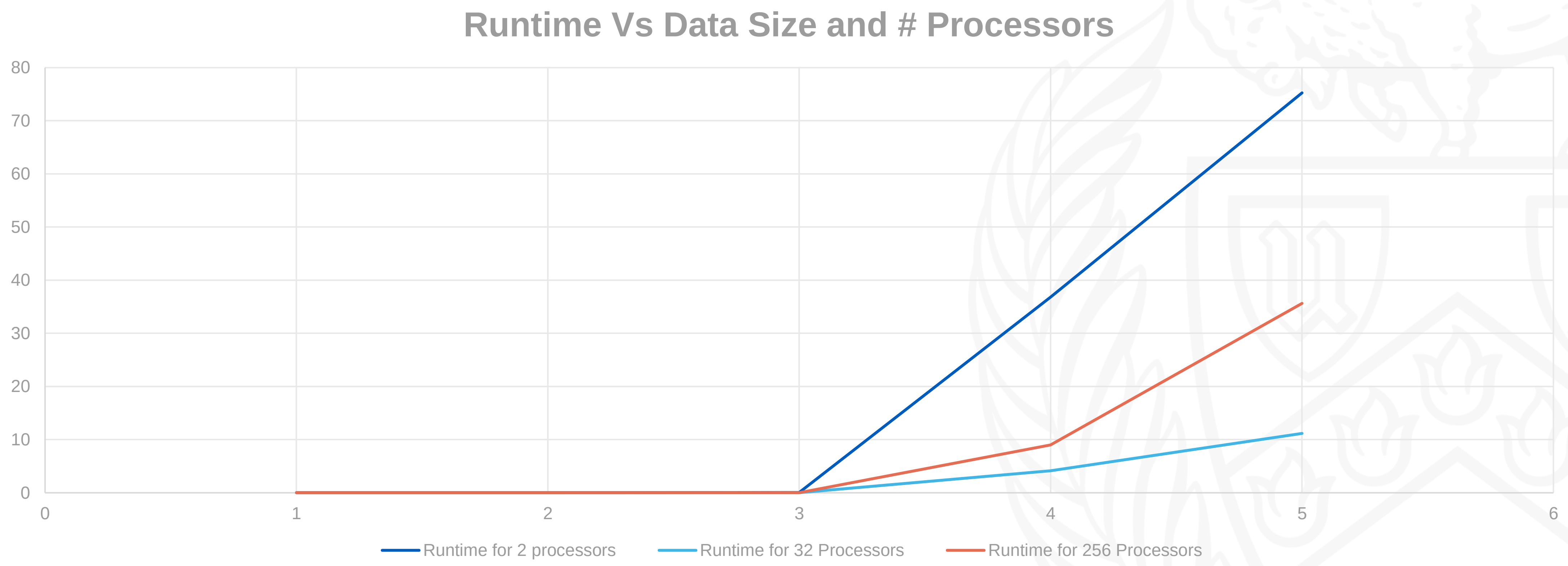


Runtime Vs Data Size (keeping number of processors constant)

Runtime Vs Data Size for Processors 2, 32 and 256

Data Size	Runtime for 2 Processors	Runtime for 32 Processors	Runtime for 256 Processors
10000(10K)	0.000725	0.0000802	0.000338
100000(100K)	0.00727	0.000734	0.002351
1000000(1M)	0.067182	0.007387	0.019365
500000000(500M)	36.810129	4.1484	8.982689
1000000000(1B)	75.223702	11.179744	35.620181

Runtime Vs Data Size for Processors 2, 32 and 256 Line Chart



Challenges & Learnings

- Generating and Collecting huge data at processor with rank 0.
- Long running time for 128 and 256 number of processors.
- Analyzed, how the runtime increases as the number of nodes increases against the data size.
- Understood where parallelization should be used to speed up the performance of sequential algorithm.
- Learned about MPI, CCR and Slurm Jobs.
- Learned about different SLURM commands like squeue, srun, sbatch,etc

Conclusion

- As per the results and graphs, we can see that the parallelism can be efficient only up-to a particular number of processor.
- And if we want to add further processors by adding the nodes, it also adds the network latency which again add up to the communication overhead.
- Further, I have used MPI_Scatter and MPI_Gather which are very costly in terms of communication time and eats up lot of bandwidth.

References

- Dr. Russ Miller’s webpage: https://cse.buffalo.edu/faculty/miller/teaching.shtml
- https://www.programiz.com/dsa/merge-sort
- https://www.mcs.anl.gov/~itf/dbpp/text/node127.html
- https://developer.nvidia.com/blog/merge-sort-explained-a-data-scientists-algorithm-guide/
- http://selkie-macalester.org/csinparallel/modules/MPIProgramming/build/html/mergeSort/mergeSort.html
- https://studylib.net/doc/5894233/parallel-merge-sort-implementation
- https://developer.nvidia.com/blog/merge-sort-explained-a-data-scientists-algorithm-guide/
- https://ubccr.freshdesk.com/support/solutions/articles/13000026245-tutorials-workshops-and-training-documents

Thank You!