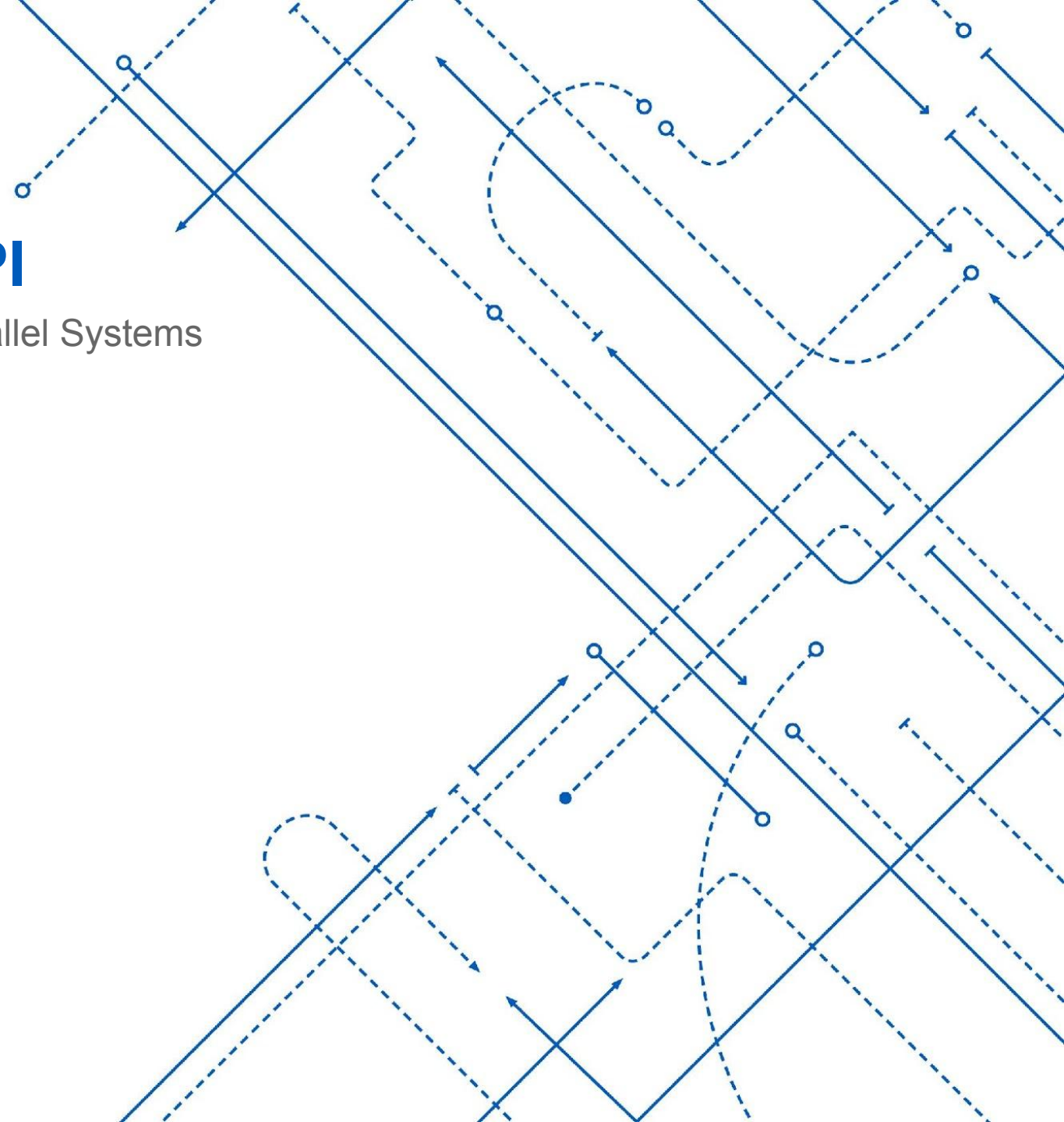


Parallel Merge Sort Using MPI

CSE 702: Seminar on Programming Massively Parallel Systems

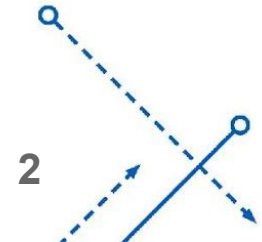
Course Instructor:
Dr. Russ Miller
UB Distinguished Professor
Department of Computer Science & Engineering
State University of New York at Buffalo

Prepared By:
Swati Nair
UB Person Number: 50246994

University at Buffalo The State University of New York

Agenda

- Sequential Merge Sort
- Sequential Algorithm Analysis
- Proposed parallel algorithm
- Experimentation in CCR
- Obtained results and analysis
- Challenges
- Learning from the course
- Conclusion
- References



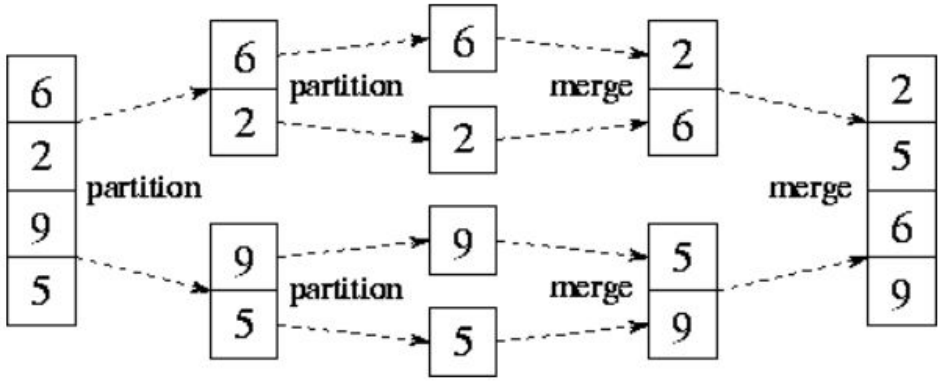
Sequential Merge Sort

mergesort(int[] a, int left, int right)

1. If the input sequence has fewer than two elements, return
2. Partition the input sequence into two halves: mid = (left + right)/ 2
3. Sort the two subsequences using the same algorithm:

   mergesort(a, left, mid-1)

   mergesort(a,mid,right)
4. Merge the two sorted subsequences to form the output sequence

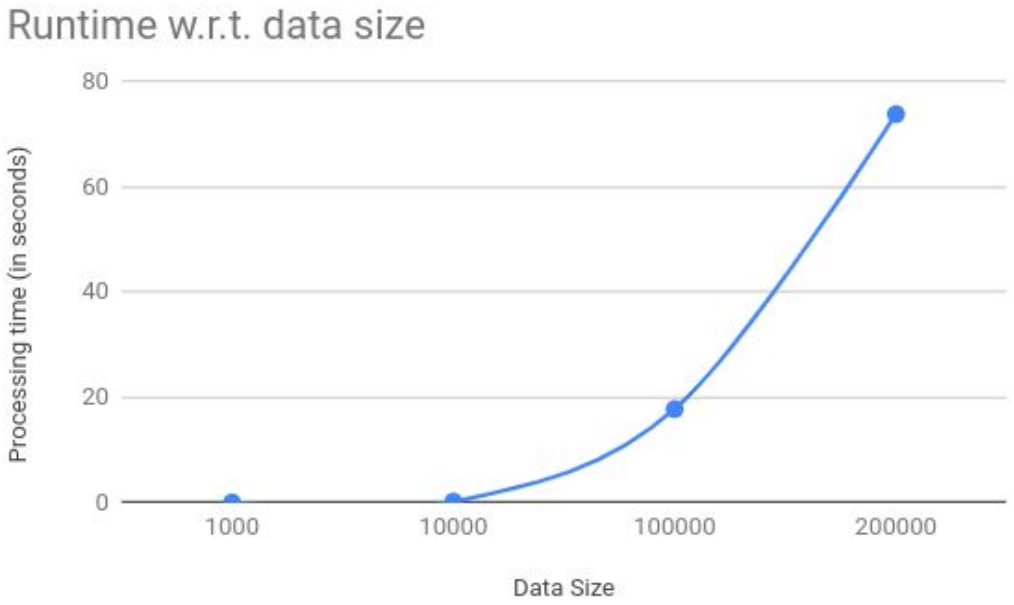


Runtime: O(Nlog N)

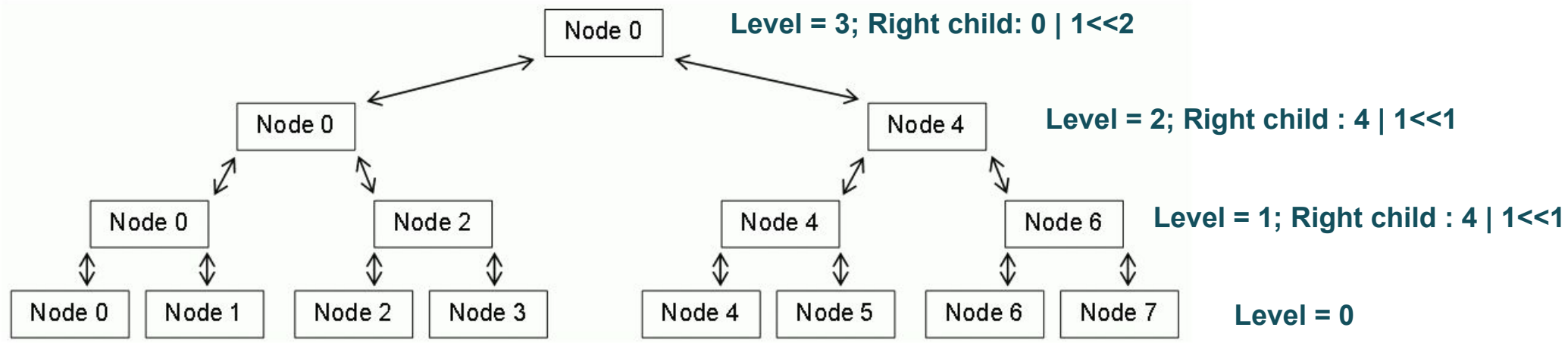
Sequential Algorithm Analysis:

Data Size	Time taken by recursive solution
1000	0.001869
10000	0.175596
100000	17.789284
200000	73.883391
1000000 (1M)	4963.379900

Time in seconds



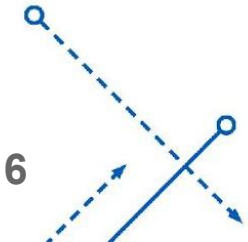
Proposed Parallel Algorithm



1. To compute rank of right child: **myRank | (1<<(myHeight-1))**
2. To compute rank of parent: **myRank & ~(1<<myHeight) =>** Needed to send the sorted data to parent node

Proposed Parallel Algorithm

1. Node with rank 0 is the host node. It computes the height of the node and get the entire dataset
2. Node 0 initiates the parallel merge operation
3. For internal nodes (height > 0) including node 0,
    a. Divide the data in half and send the right half to the right child as computed in previous slide
    b. Recursively call parallel merge operation for the left half on the same node
    c. Also, receive the sorted data from right child
    d. Merge the sorted left and right halves
4. If it is a leaf node, just do internal sorting
5. If parent's rank <> node's rank, send the sorted data to parent node
6. Finally, node 0 will have the entire sorted result



Experimentation in CCR

1. **Allocation of nodes using salloc/sbatch script:**

   salloc --nodes=8 --ntasks-per-node=1 --time=00:30:00 --exclusive

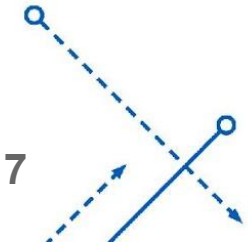
   Batch Script (in next slide)

2. **Monitoring of submitted jobs:**

   squeue -u swatishr

3. **To check node availability**

   sinfo



Experimentation in CCR: SBATCH script

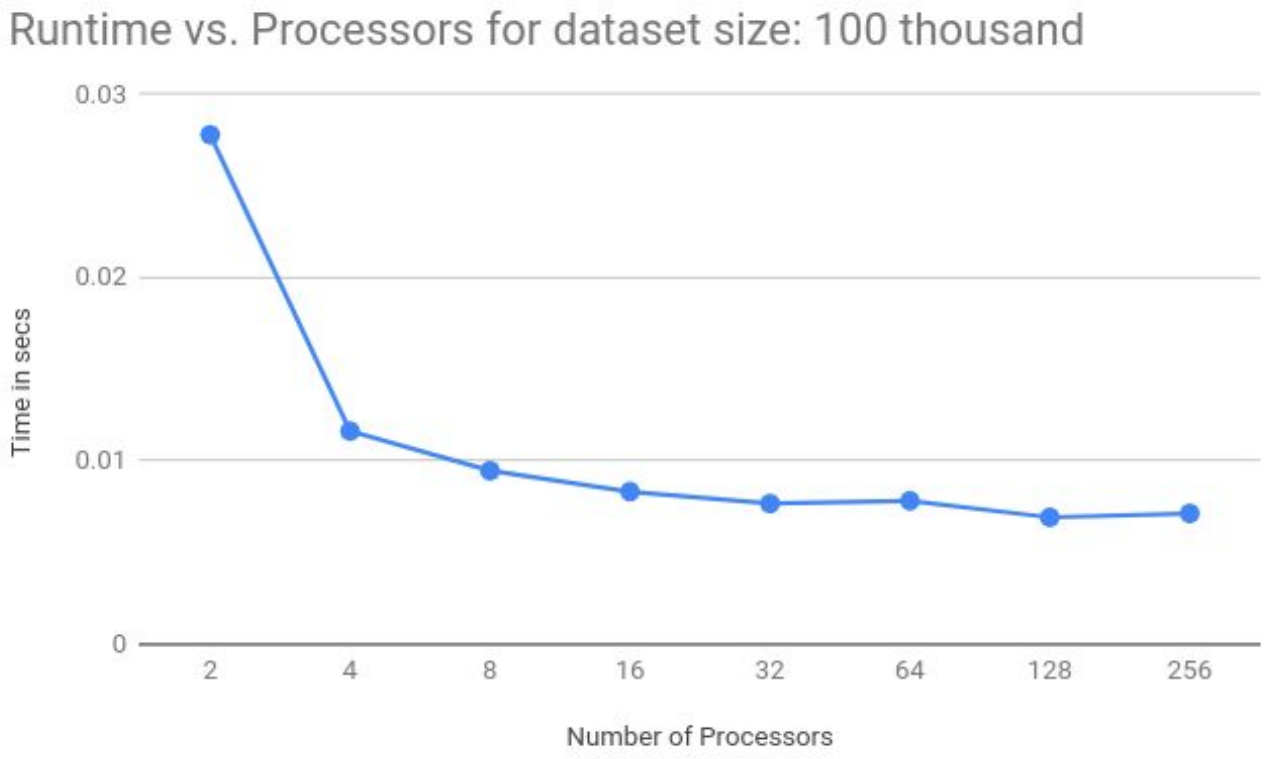
```
#!/bin/sh
#SBATCH --partition=general-compute
#SBATCH --qos=general-compute
#SBATCH --cluster=ub-hpc
#SBATCH --exclusive
#SBATCH --constraint=IB&CPU-E5645
#SBATCH --time=00:10:00
#SBATCH --nodes=64
#SBATCH --ntasks-per-node=1
#SBATCH --error=mergesort_err_par_64_corr.txt
#SBATCH --output=mergesort_out_par_64_corr.txt
#SBATCH --mail-user=swatishr@buffalo.edu
#SBATCH --mail-type=END
#SBATCH --job-name=par_merge_sort

module load intel
module load intel-mpi

export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so

srun parallelMS_final 100000
echo "Done 100000"
```

Results

1. For some data size, plot processing time vs number of nodes
   a. Tested on 9 different data sizes: 100000, 200000, 1M, 2M, 4M, 8M, 100M, 200M, 1 billion (showing results for few of them)
   b. Number of nodes: 2, 4, 8, 16, 32, 64, 128, 256
2. Plot speed-up that shows performance of parallel over sequential
3. Plot graphs that depict for a particular number of processor, how the runtime is affected with data size

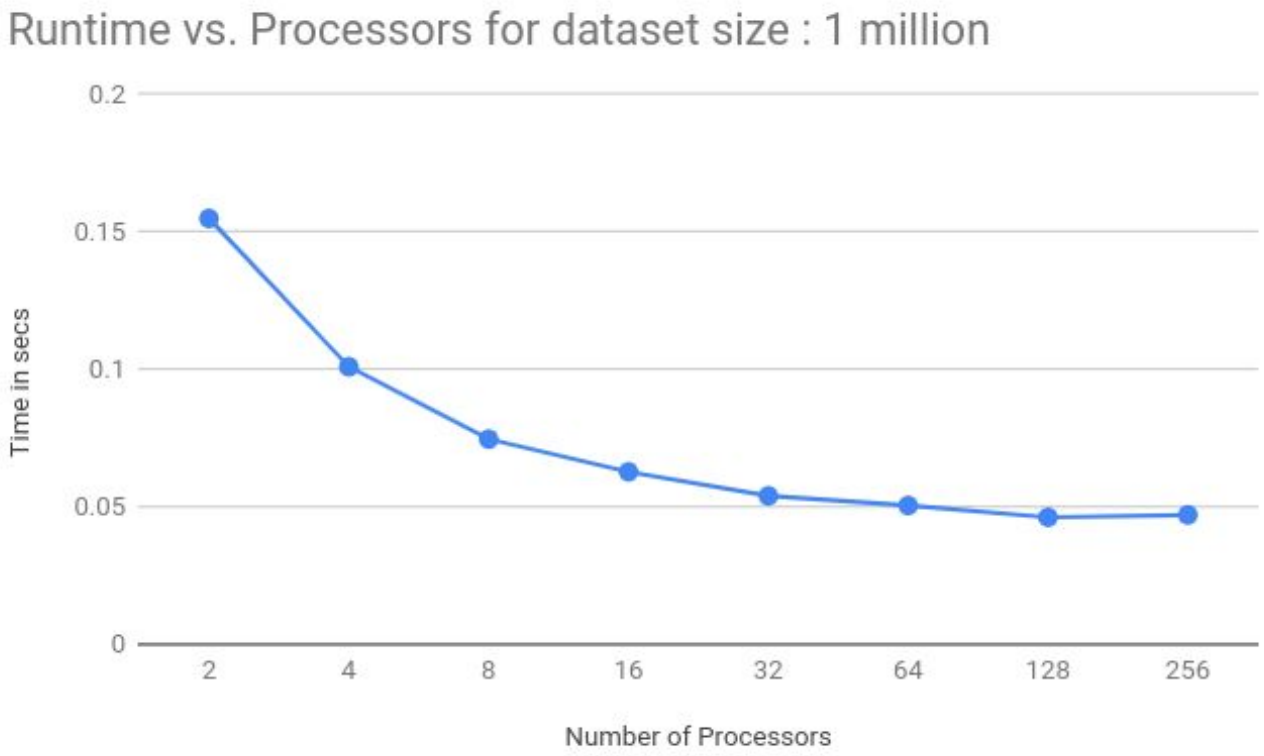
Runtime Vs Number of nodes for N = 100000

Data Size: 100000	
Processors	Time
2	0.027800
4	0.011632
8	0.009468
16	0.008330
32	0.007679
64	0.007830
128	0.006931
256	0.007143



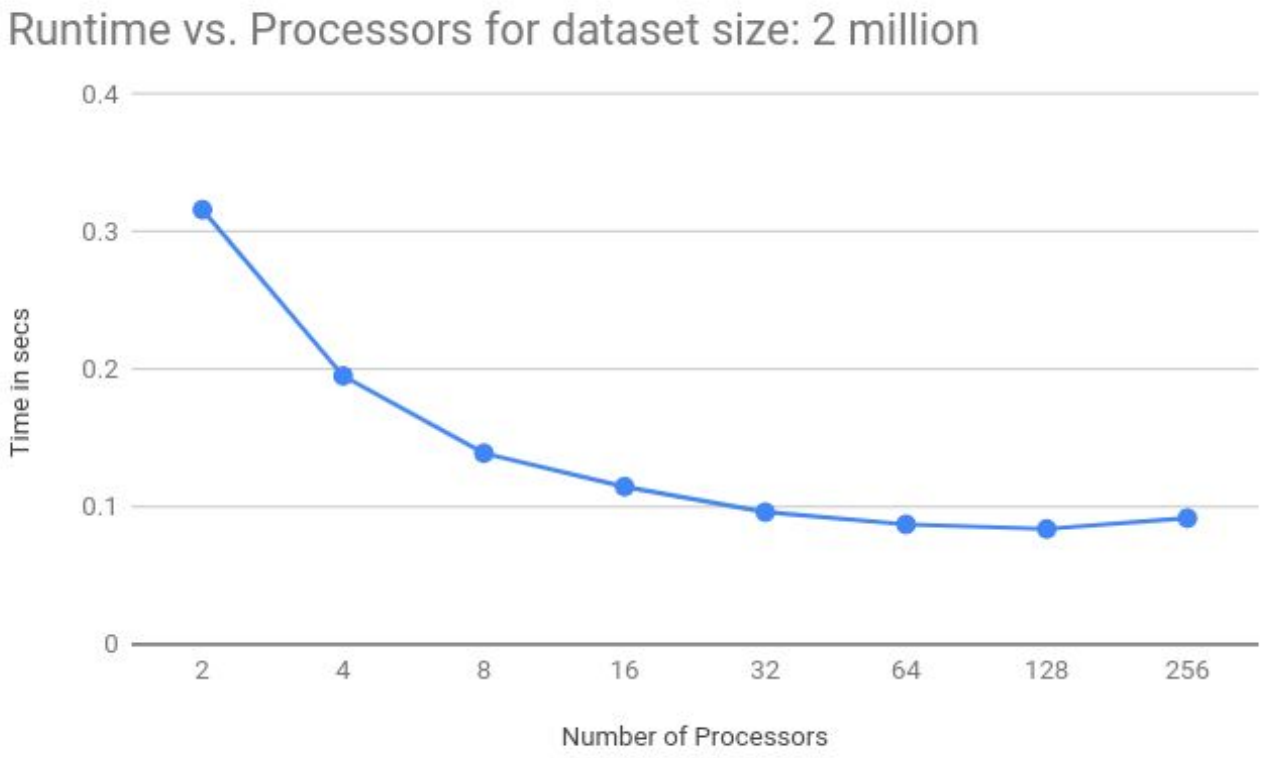
Runtime Vs Number of nodes for N = 1 million

Key Size: 1000000 (1 million)	
Processors	Time
2	0.154900
4	0.100955
8	0.074639
16	0.062747
32	0.053992
64	0.050502
128	0.046206
256	0.047069



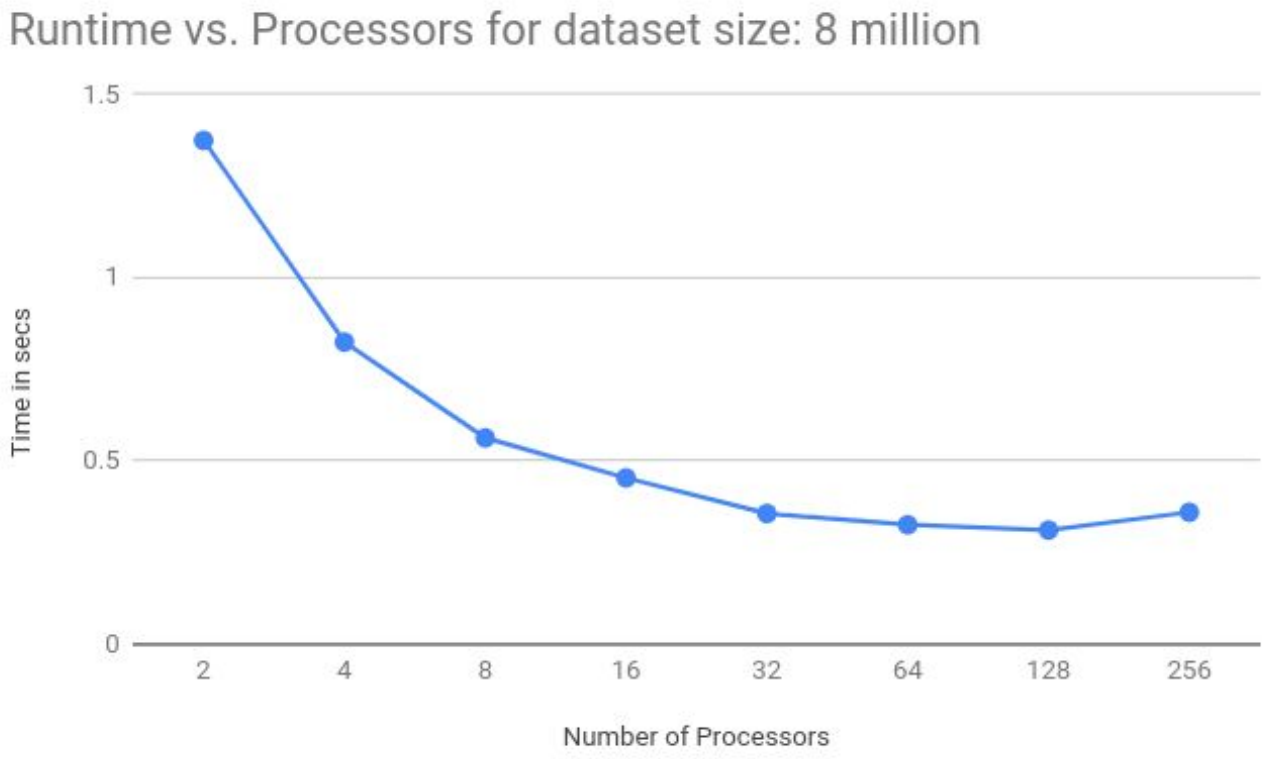
Runtime Vs Number of nodes for N = 2 million

Key Size: 2000000 (2 million)	
Processors	Time
2	0.316157
4	0.195291
8	0.139053
16	0.114699
32	0.096136
64	0.087273
128	0.083914
256	0.091789



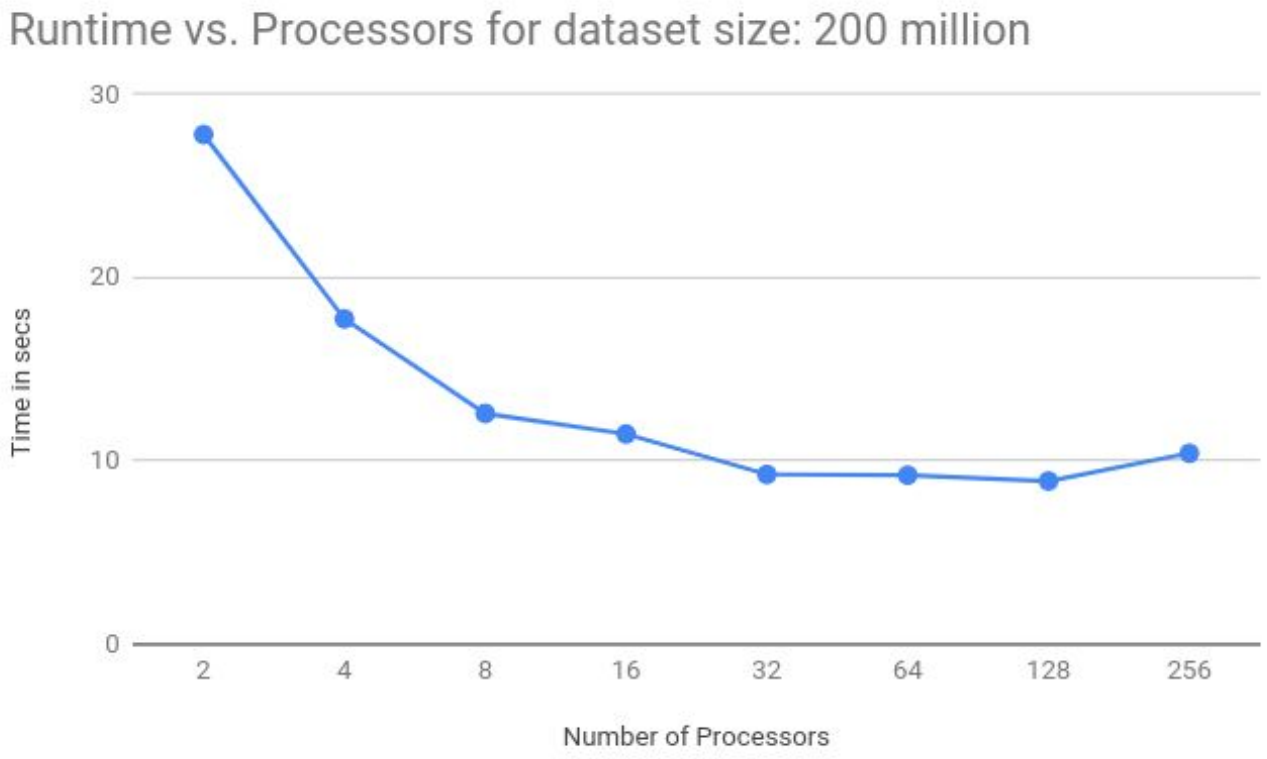
Runtime Vs Number of nodes for N = 8 million

Key Size: 8000000 (8 million)	
Processors	Time
2	1.374457
4	0.824450
8	0.563136
16	0.454040
32	0.356974
64	0.326099
128	0.311515
256	0.360204



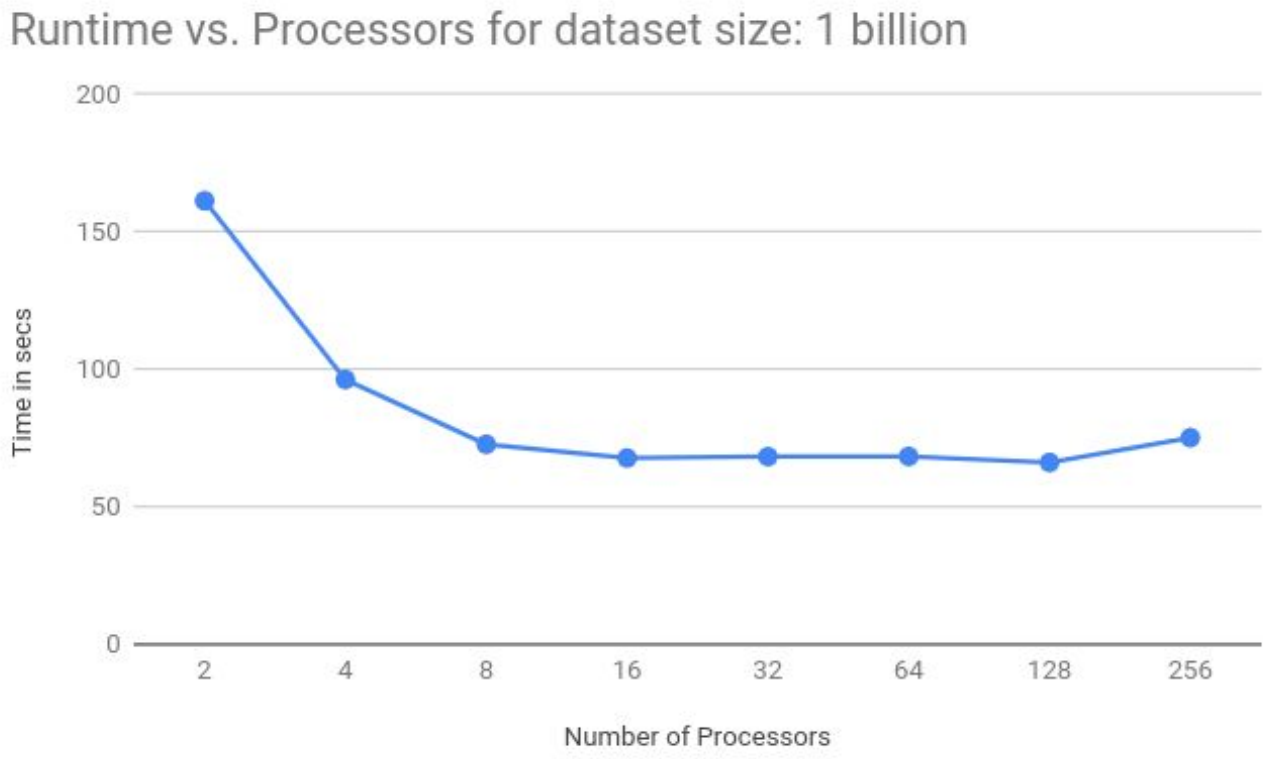
Runtime Vs Number of nodes for N = 200 million

Key Size: 200000000 (200 million)	
Processors	Time
2	27.812386
4	17.753978
8	12.595132
16	11.473544
32	9.268599
64	9.226259
128	8.909778
256	10.424679

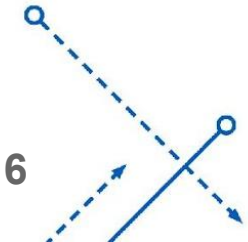


Runtime Vs Number of nodes for N = 1 billion

Key Size: 1000000000 (1 billion)	
Processors	Time
2	161.343157
4	96.306835
8	72.752683
16	67.765338
32	68.262727
64	68.336235
128	66.149395
256	75.171655



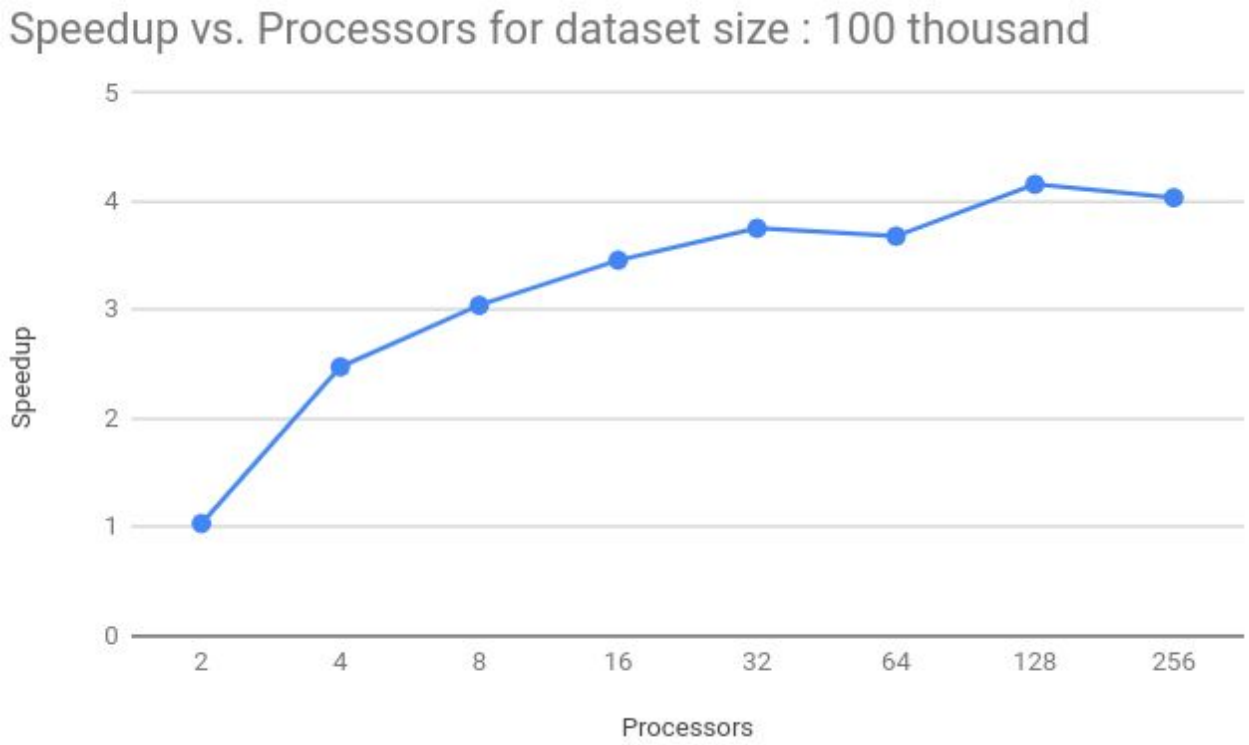
Speedup: Ratio of sequential to parallel execution time



Speedup for N = 100000

Baseline:
Runtime on 1 node (Sequential): 0.028825 sec

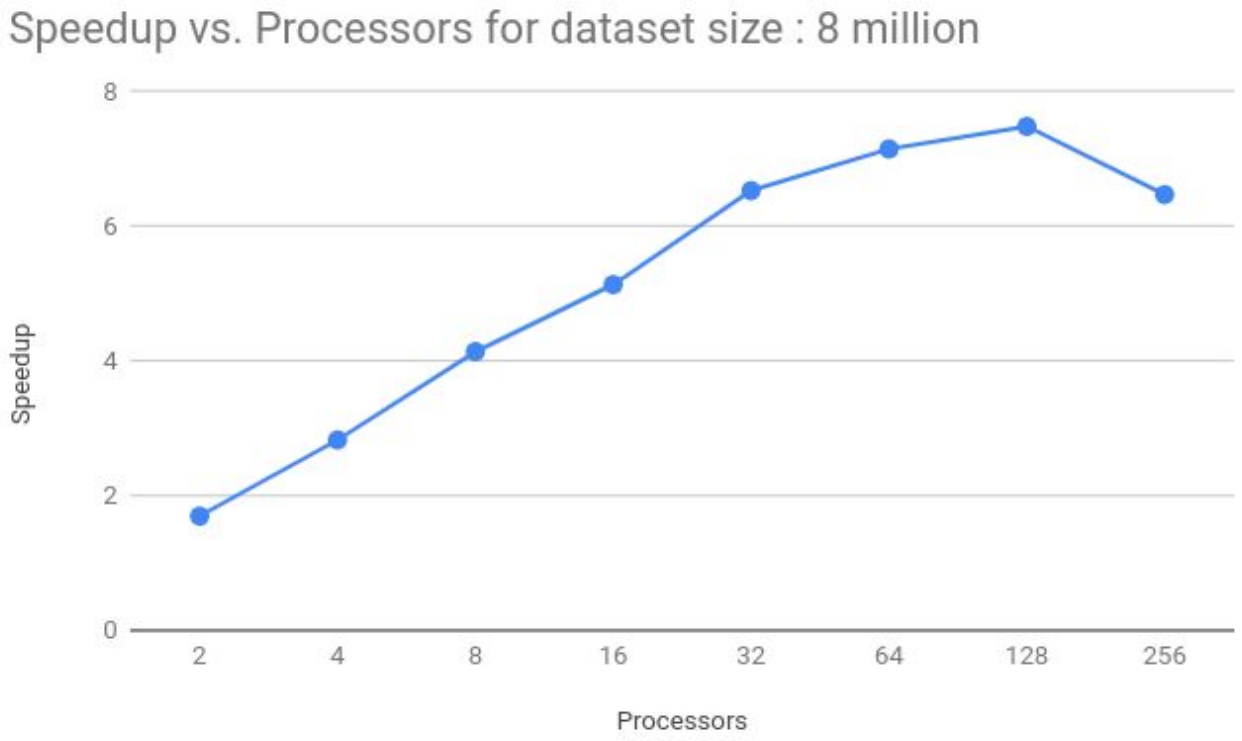
Key Size: 100000	
Processors	Speedup
2	1.0368
4	2.478
8	3.0444
16	3.4603
32	3.7537
64	3.6813
128	4.1588
256	4.0354



Speedup for N = 8 million

Baseline:
Runtime on 1 node (Sequential): 2.331913 sec

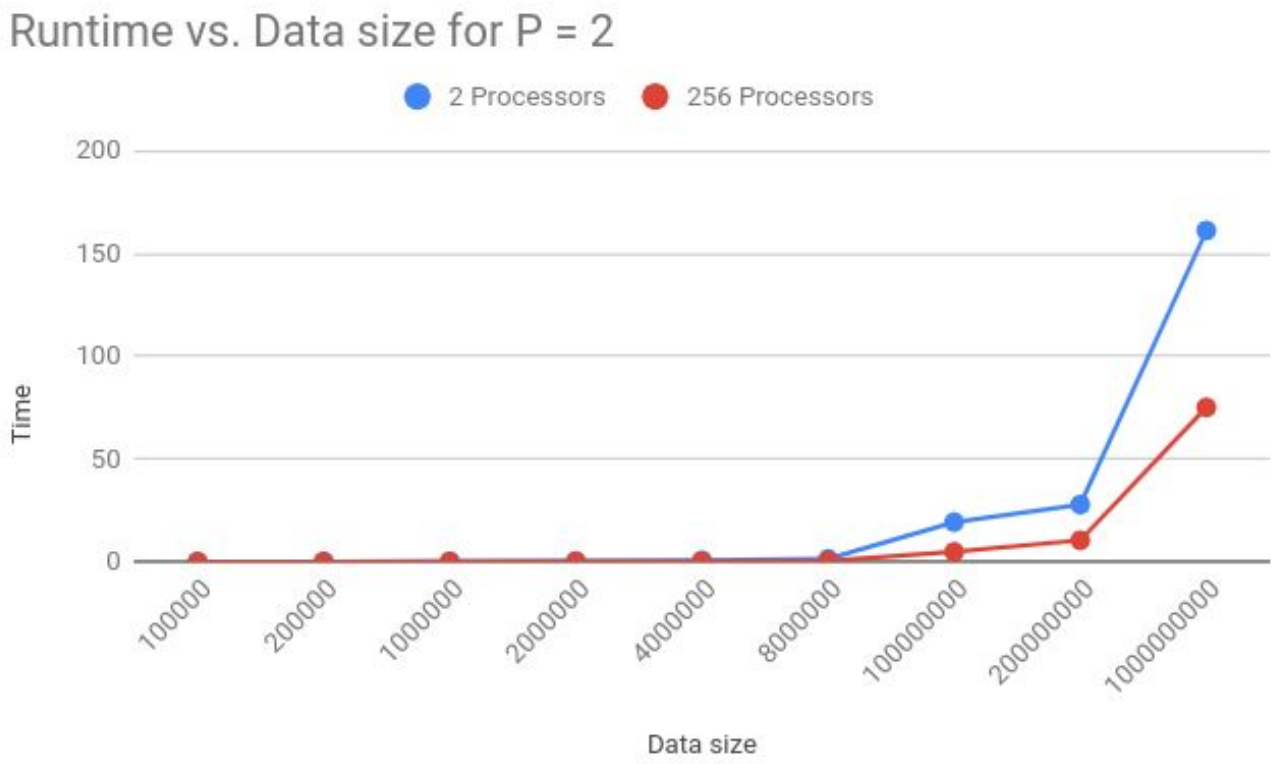
Key Size: 8 million	
Processors	Speedup
2	1.6966
4	2.8284
8	4.1409
16	5.1359
32	6.5324
64	7.1509
128	7.4857
256	6.4738



Runtime Vs Data size
(keeping number of processors constant)

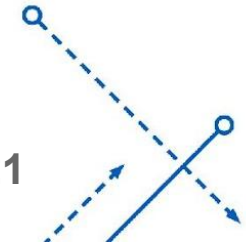
Runtime Vs Data size for P = 2 and 256

Data size	Time for 2 Processors	Time for 256 Processors
100000	0.0278	0.007143
200000	0.0291	0.01118
1000000	0.1549	0.047069
2000000	0.316157	0.091789
4000000	0.660208	0.181232
8000000	1.374457	0.360204
100000000	19.288978	4.784959
200000000	27.812386	10.424679
1000000000	161.343157	75.171655



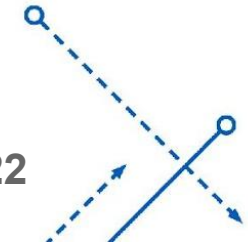
Challenges & Learnings

- Long time to provision 128 and 256 nodes
- Analyzed the difference in runtimes as the number of nodes increased or as the data size increases
- Understood where parallelization should be used and how it can speed up the performance of sequential algorithms.
- Knowledge on MPI, CCR and Slurm jobs
- Use of different SLURM jobs such as ***sinfo, squeue, srun and salloc*** while troubleshooting node allocation.



Conclusion

- Only one task was associated with each node, thus, every physical server initiated one process only.
- According to the results, parallelism can be efficient only upto a particular number of processors/nodes.
- For further addition of nodes, network latency and/or size of smallest chunk of data hampers the performance!



References

- Dr. Russ Miller's webpage: https://cse.buffalo.edu/faculty/miller/teaching.shtml
- Parallel Merge sort: https://www.mcs.anl.gov/~itf/dbpp/text/node127.html
- http://penguin.ewu.edu/~trolfe/ParallelMerge/ParallelMerge.html

Thank you