CSE 702: SEMINAR ON PROGRAMMING MASSIVELY PARALLEL SYSTEMS

Learning and Implementing Parallel Odd-even sort using MPI in C

PREPARED BY:
Charushi Nanwani (UB PERSON NUMBER: 50248736)

University at Buffalo
School of Engineering and Applied Sciences

1846

University at Buffalo
School of Engineering and Applied Sciences

Overview

- Bubble Sort
- Odd Even Transposition Sort
- Sequential and Parallel Algorithms
- Implementation
- Obtained Results
- Observations
- References

Bubble Sort

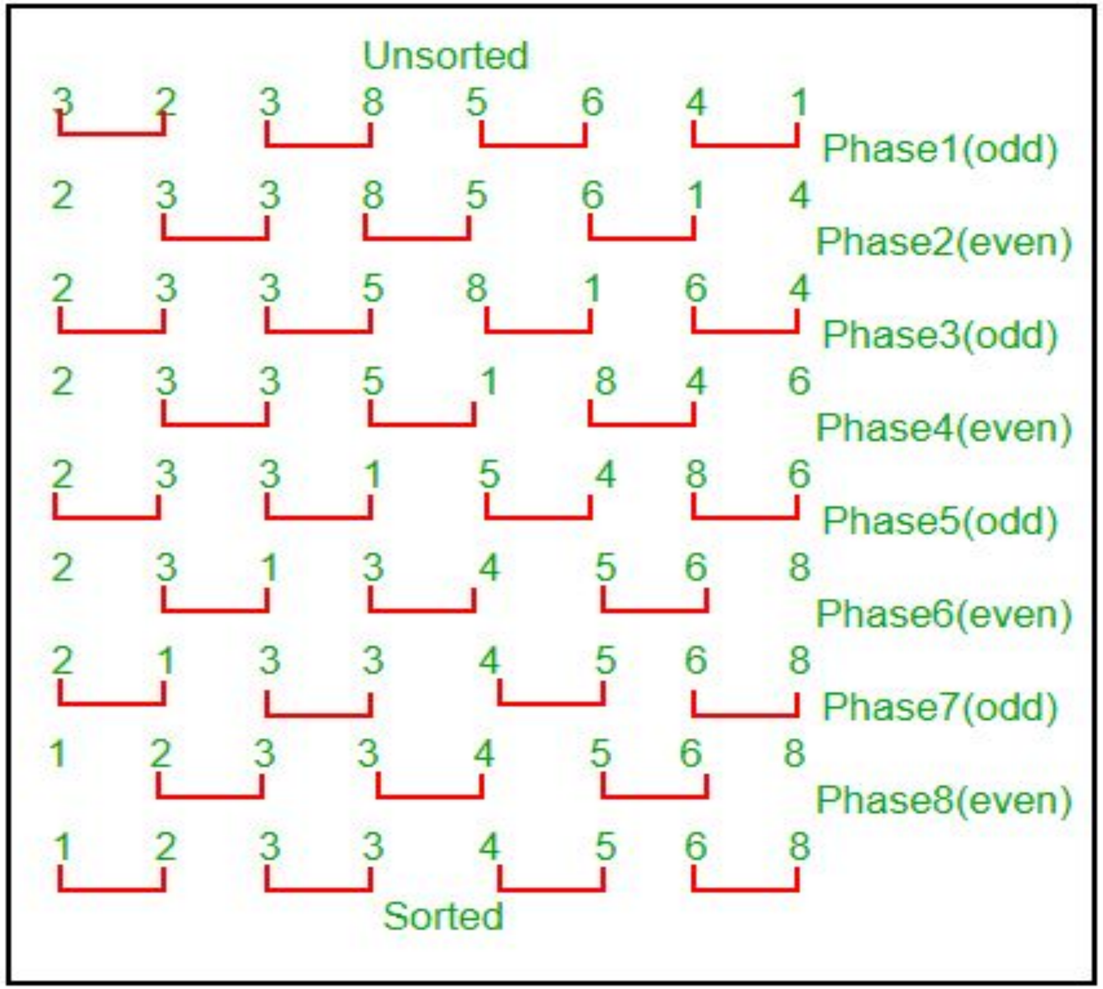
- Compares two consecutive values at a time and swaps them if they are out of order
- Number of comparisons and swaps:n(n-1)/2 which corresponds to a time complexity O(N^2)

```
void Bubble_sort(
        int  a[]  /* in/out */,
        int  n    /* in     */) {
    int list_length, i, temp;

    for (list_length = n; list_length >= 2; list_length--)
        for (i = 0; i < list_length-1; i++)
            if (a[i] > a[i+1]) {
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
            }

}  /* Bubble_sort */
```

Odd Even Transposition sort

- Variant of the Bubble Sort
- Operates in two alternate phases
- ***Phase Even***
  - even processes exchange values with right neighbors
- ***Phase Odd***
  - odd processes exchange values with right neighbors
- List will always be sorted after n phases

Sequential Odd Even Sort



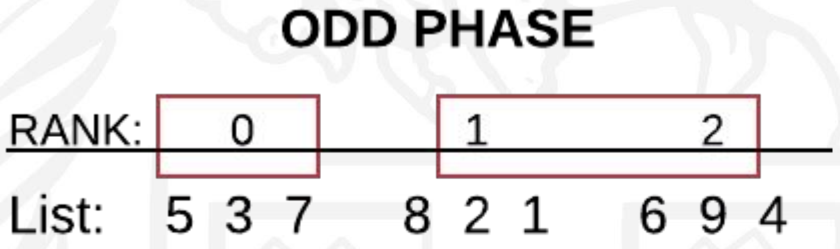

```
void Odd_even_sort(
      int  a[]  /* in/out */,
      int  n    /* in     */) {
   int phase, i, temp;

   for (phase = 0; phase < n; phase++)
      if (phase % 2 == 0) { /* Even phase */
         for (i = 1; i < n; i += 2)
            if (a[i-1] > a[i]) {
               temp = a[i];
               a[i] = a[i-1];
               a[i-1] = temp;
            }
      } else { /* Odd phase */
         for (i = 1; i < n-1; i += 2)
            if (a[i] > a[i+1]) {
               temp = a[i];
               a[i] = a[i+1];
               a[i+1] = temp;
            }
      }
}  /* Odd_even_sort */
```

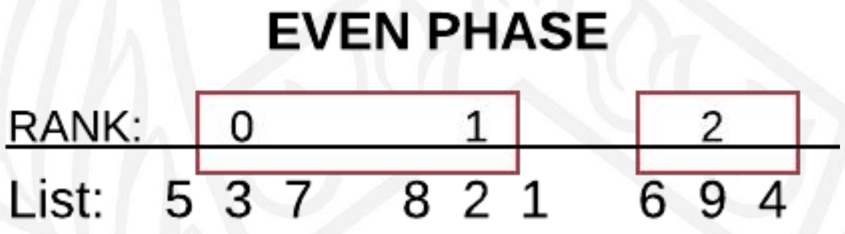
Parallel Algorithm

Time	Process			
	0	1	2	3
Start	15, 11, 9, 16	3, 14, 8, 7	4, 6, 12, 10	5, 2, 13, 1
After Local Sort	9, 11, 15, 16	3, 7, 8, 14	4, 6, 10, 12	1, 2, 5, 13
After Phase 0	3, 7, 8, 9	11, 14, 15, 16	1, 2, 4, 5	6, 10, 12, 13
After Phase 1	3, 7, 8, 9	1, 2, 4, 5	11, 14, 15, 16	6, 10, 12, 13
After Phase 2	1, 2, 3, 4	5, 7, 8, 9	6, 10, 11, 12	13, 14, 15, 16
After Phase 3	1, 2, 3, 4	5, 6, 7, 8	9, 10, 11, 12	13, 14, 15, 16

Implementation

- n elements and p processors
- each processor receives n/p elements
- sort local elements using quicksort (faster!)
- **Odd Phase**:
  - (p1,p2), (p3,p4), ….
  - the two processors exchange data
  - odd numbered processor keeps the lowest of n/p elements
  - even numbered processor keeps the highest of n/p elements
- **Even Phase**:
  - (p0,p1), (p2,p3), ….
  - even numbered processor keeps lowest of n/p elements
  - odd numbered processor keeps highest of n/p elements





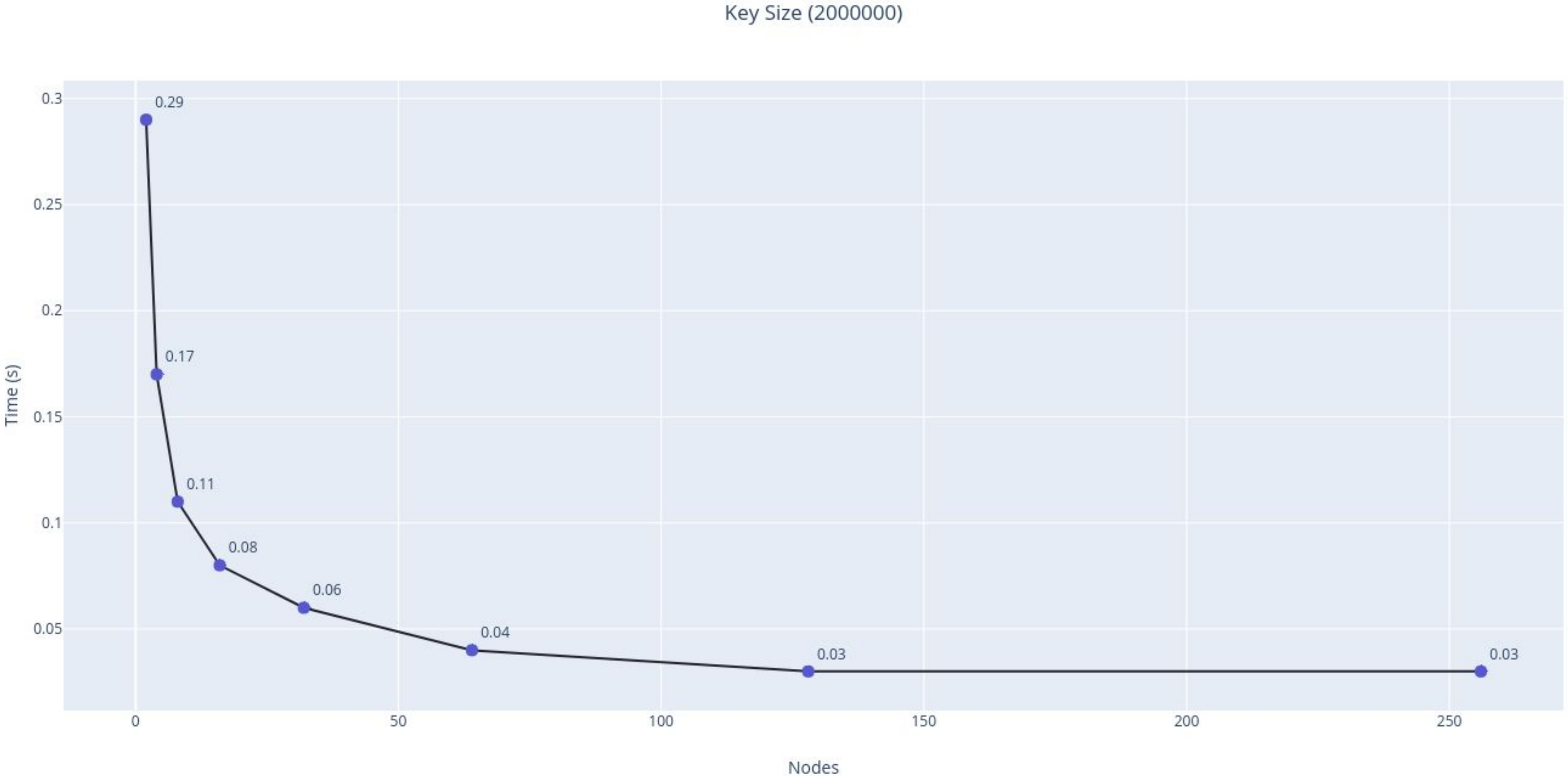
Script for running SLURM job

```
#!/bin/sh
#SBATCH --nodes=32
#SBATCH --ntasks-per-node=1
#SBATCH --constraint=IB
#SBATCH --partition=general-compute --qos=general-compute
#SBATCH --time=12:00:00
#SBATCH --mail-type=END
#SBATCH --mail-user=charushi@buffalo.edu
#SBATCH --output=odd_even_n1_32.out
#SBATCH --job-name=testing_mpi_odd_even
#SBATCH --requeue
echo "SLURM NODES"=$SLURM_NNODES
module load intel/14.0
module load intel-mpi/4.1.3
module list
#mpicc -lm -o odd_even mpi_odd_even.c
ulimit -s unlimited


export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so

mpicc -lm -o odd_even_n1_32 mpi_odd_even.c
time srun  ./odd_even_n1_32 g 1000000

#
echo "All done!"
```

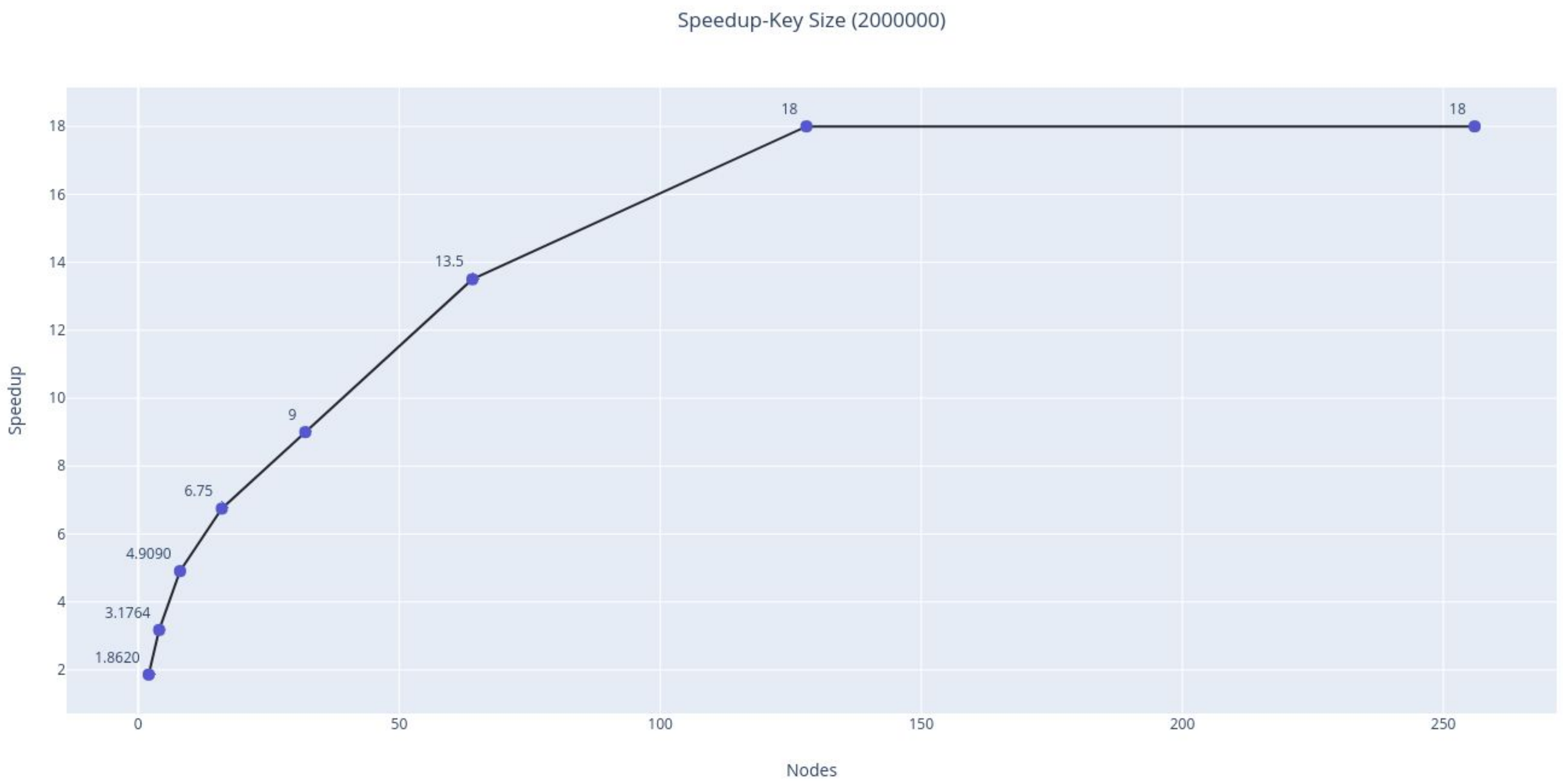
Parallel Running Time

of elements: 2 million

Nodes	Runtime
2	0.29
4	0.17
8	0.11
16	0.08
32	0.06
64	0.04
128	0.03
256	0.03



Parallel Speedup



Speedup: Ratio of serial runtime of sequential algorithm for solving a problem to the time taken by the parallel algorithm for solving the same problem

$$S = T_s/T_p$$

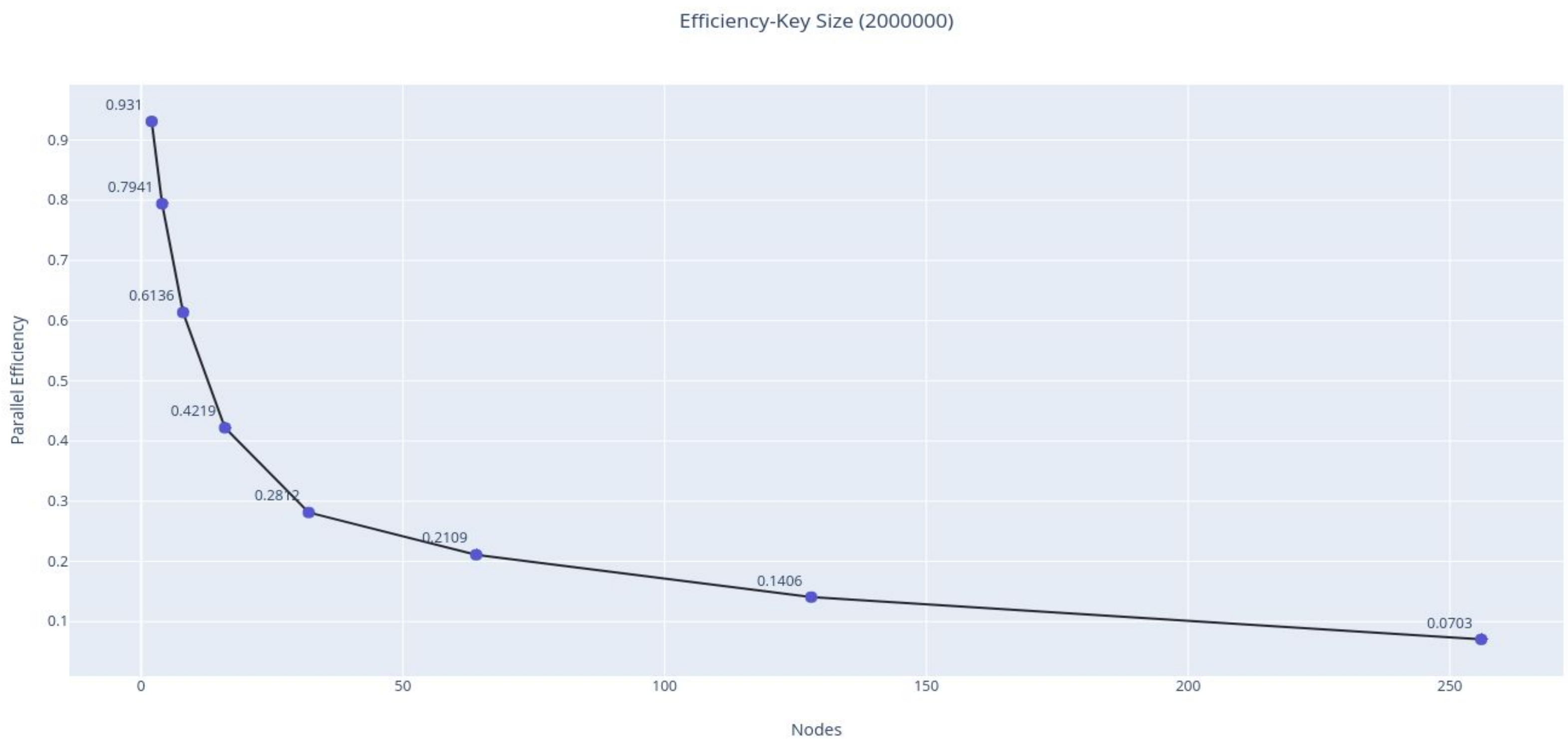
Nodes	Speedup
2	1.862
4	3.1764
8	4.909
16	6.75
32	9
64	13.5
128	18
256	18

Parallel Efficiency

Efficiency: Measures the fraction of time, for which a processor is usefully utilized.

E = S/p

$E = T_s / p\ T_p$

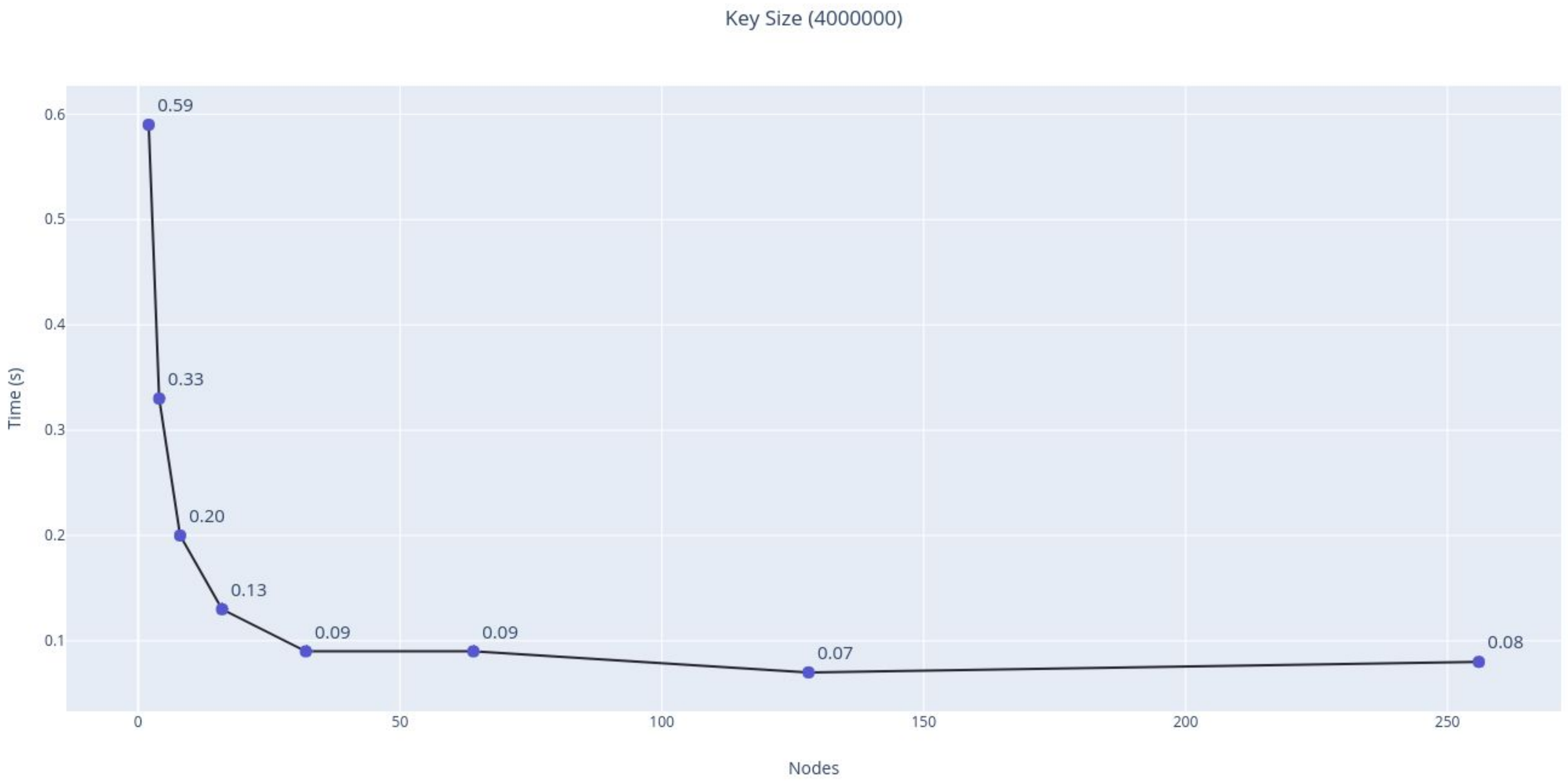


Nodes	Efficiency
2	0.931
4	0.7941
8	0.6136
16	0.4219
32	0.2812
64	0.2109
128	0.1406
256	0.0703

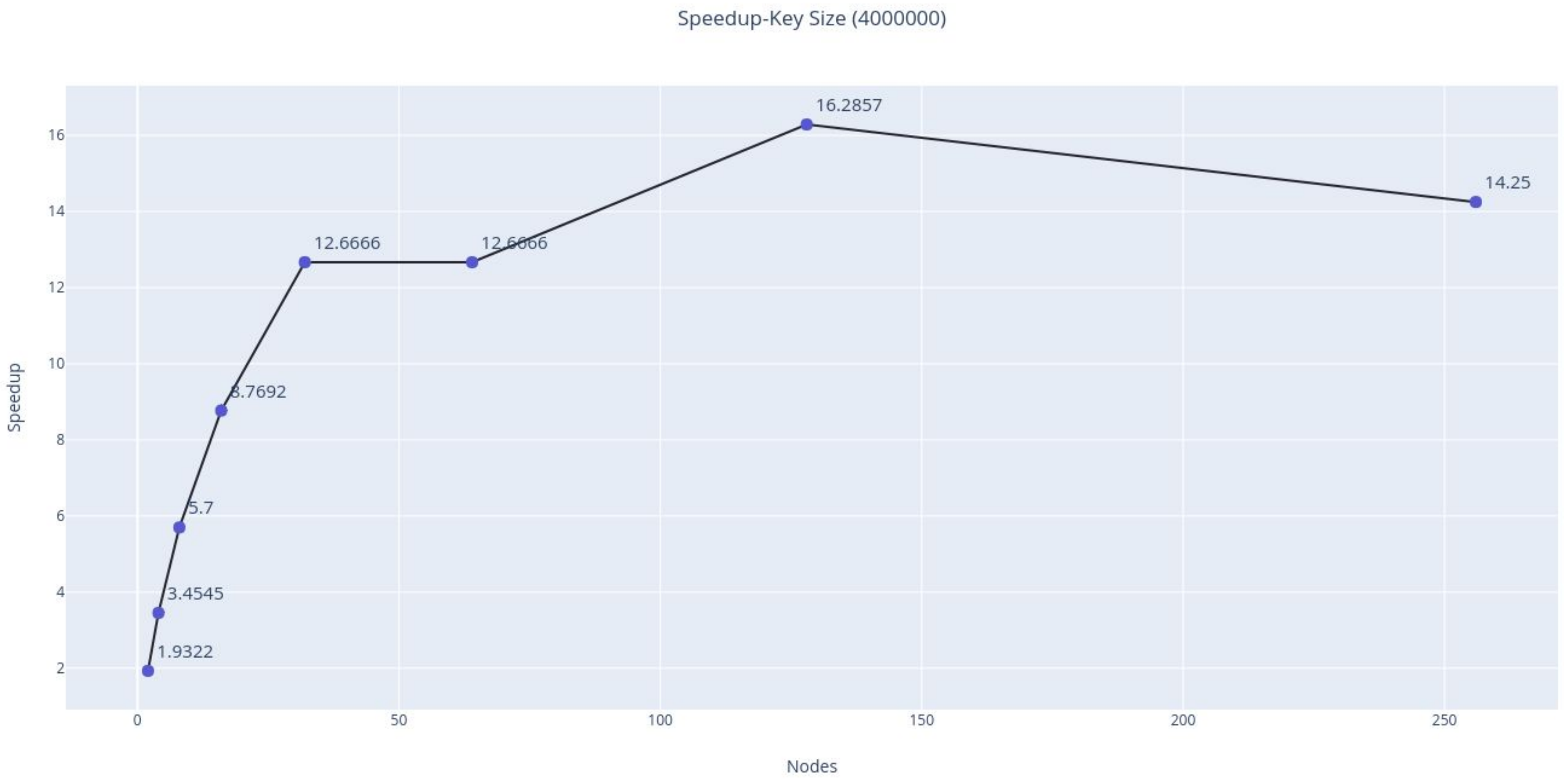
Parallel Runtime

of elements: 4 million

Nodes	Runtime
2	0.59
4	0.33
8	0.2
16	0.13
32	0.09
64	0.09
128	0.07
256	0.08

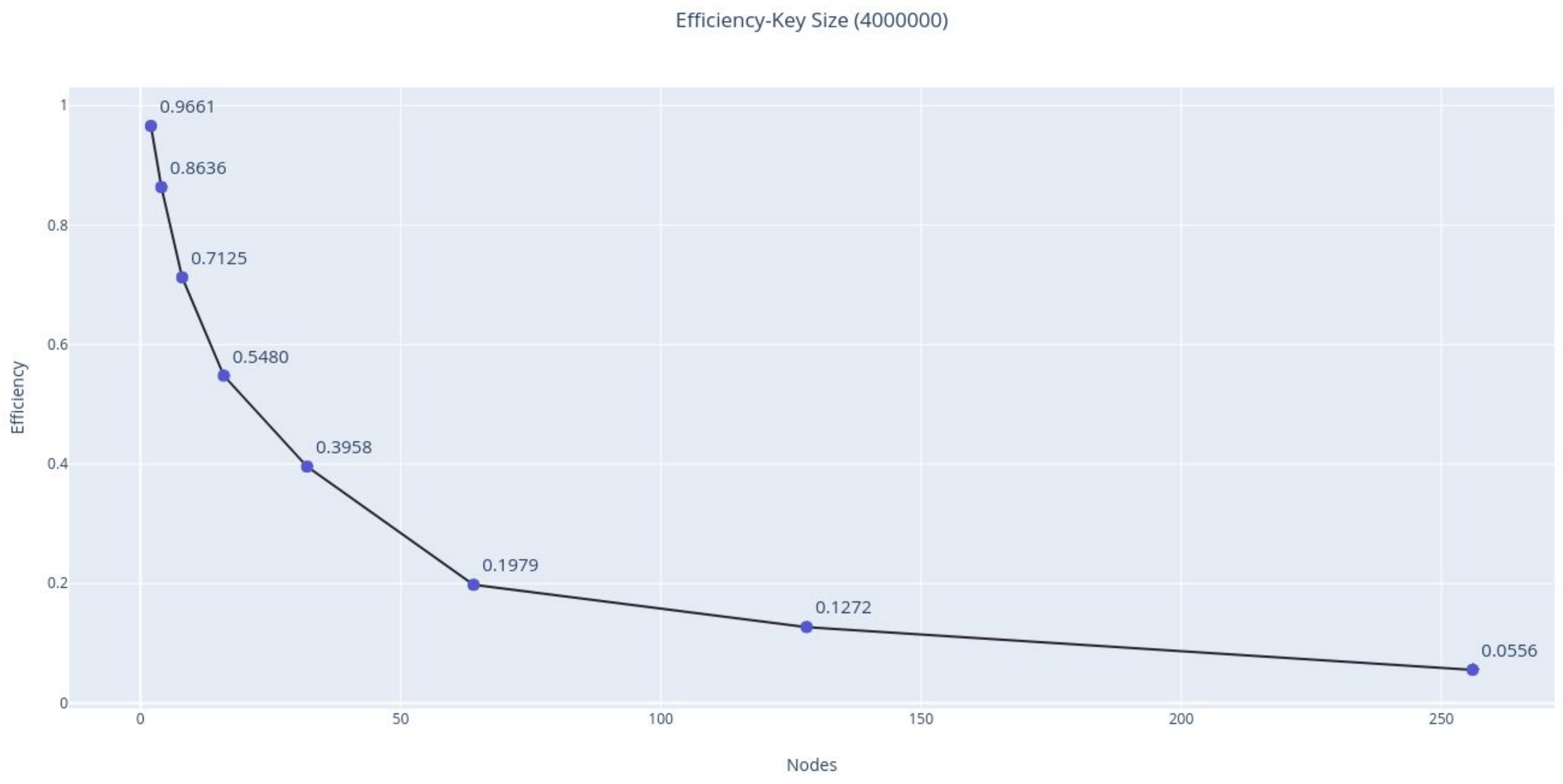


Speedup



Nodes	Speedup
2	1.9322
4	3.4545
8	5.7
16	8.7692
32	12.6666
64	12.6666
128	16.2857
256	14.25

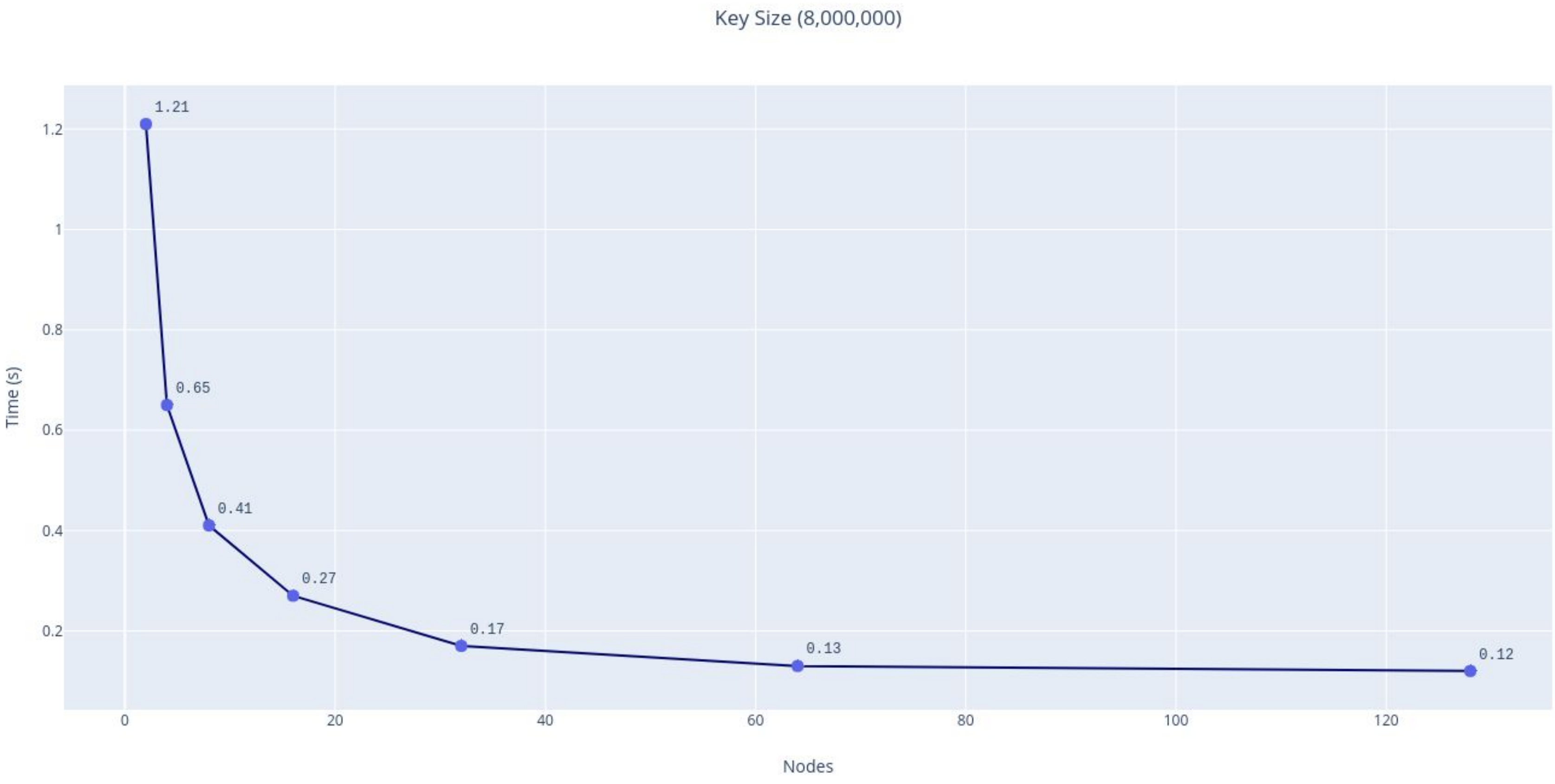
Efficiency



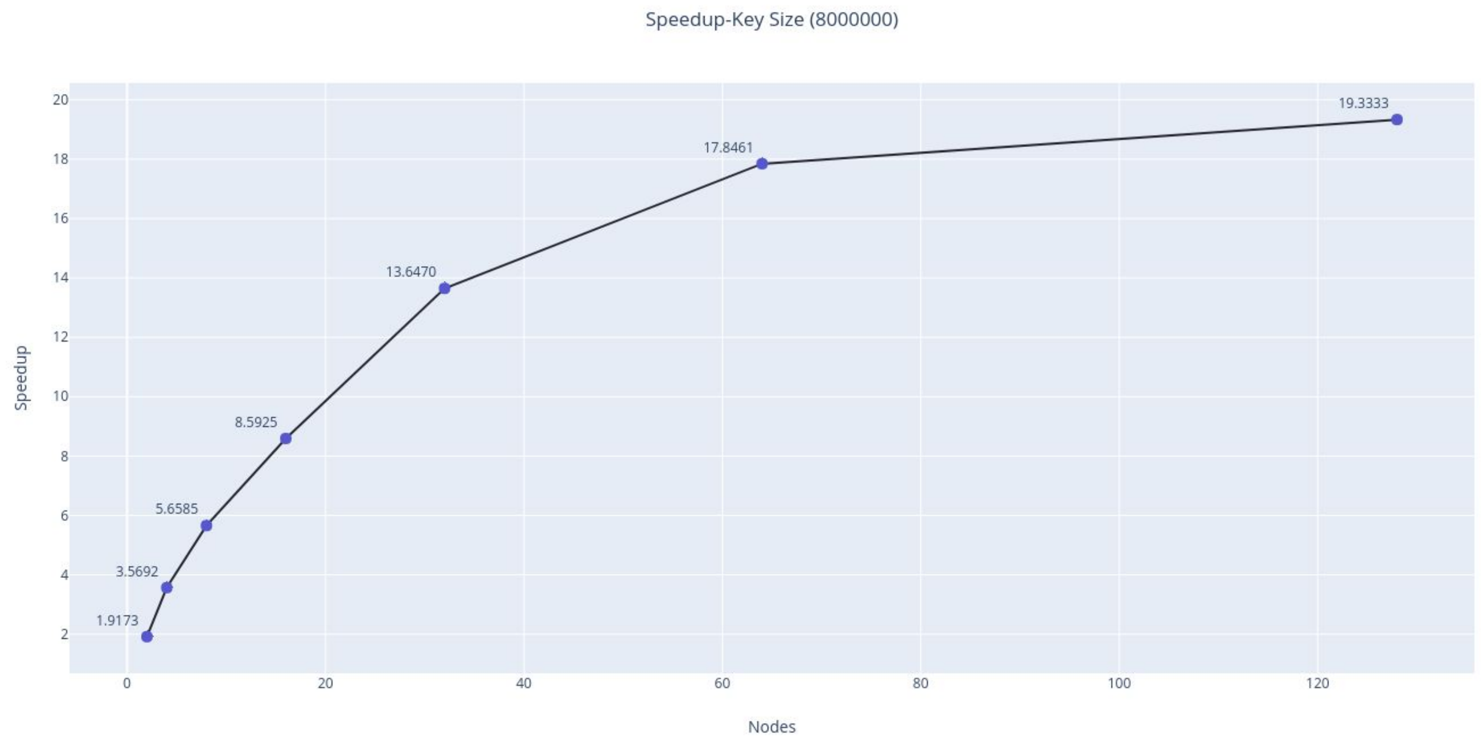
Nodes	Efficiency
2	0.9661
4	0.8636
8	0.7125
16	0.548
32	0.3958
64	0.1979
128	0.1272
256	0.0556

of elements: 8 million

Nodes	Runtime
2	1.21
4	0.65
8	0.41
16	0.27
32	0.17
64	0.13
128	0.12

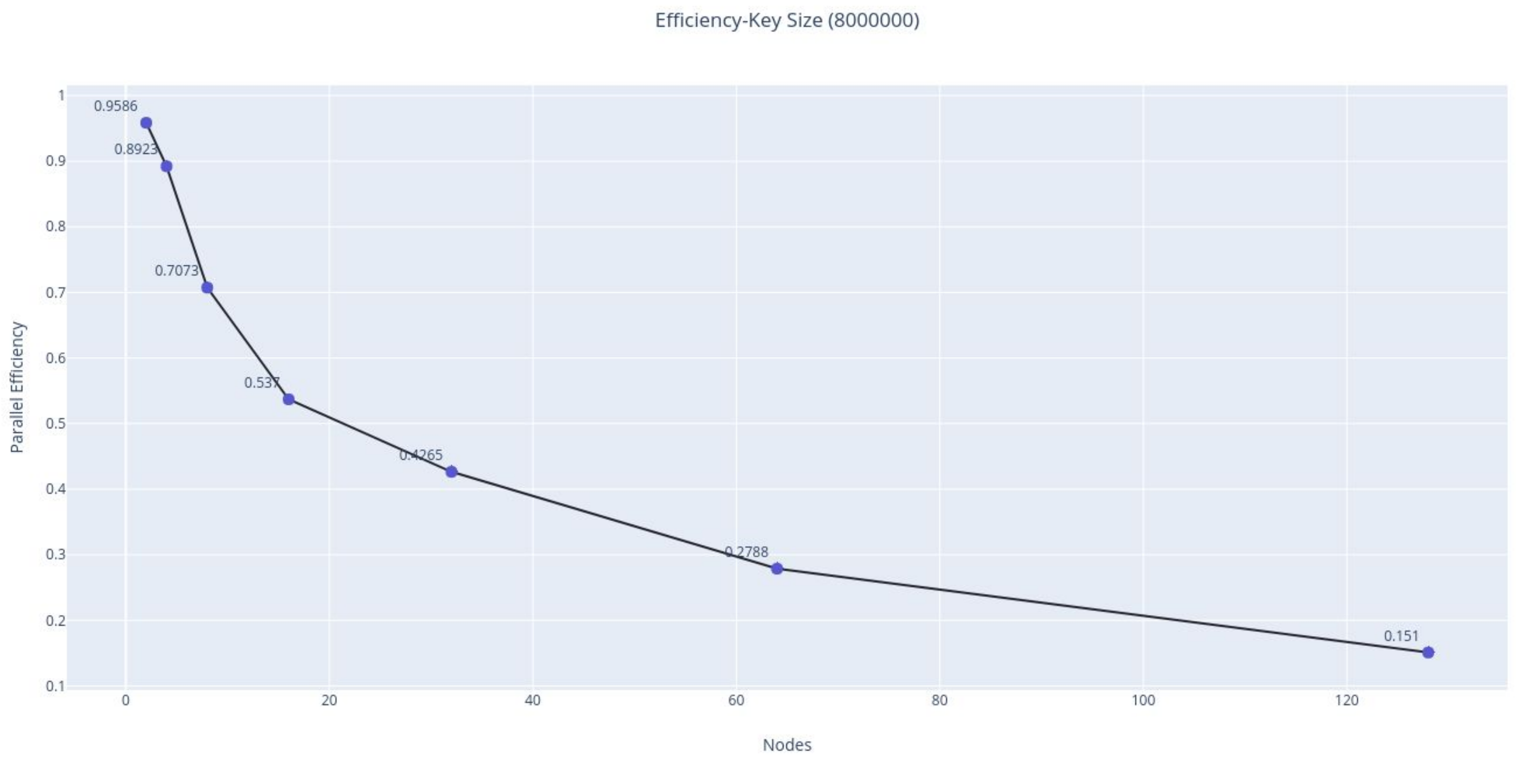


Speedup



Nodes	Speedup
2	1.9173
4	3.5692
8	5.6585
16	8.5925
32	13.647
64	17.8461
128	19.3333

Efficiency



Nodes	Efficiency
2	0.9586
4	0.8923
8	0.7073
16	0.537
32	0.4265
64	0.2788
128	0.151

Observations

- The runtime decreases on increasing the processors, but after a certain extent, becomes constant or increases again.
- Jobs with larger numbers as input are bound by sequential computation time for a small number of processors, but eventually adding processors causes communication time to take over.
- Communication overhead decreases speedup for a large number of processors (128, 256)
- Parallel computing is useful when the number of processors are small, or when the problem is perfectly parallel, and has a large amount of data which requires computation.

Things I learned

- Writing MPI programs in C
- How jobs are submitted and scheduled on CCR
- Basic slurm commands, as well as monitoring jobs
- Tradeoffs associated with using different number of processors
- Different factors that affect whether or not a job will parallelize well
    - Sequential Runtime and Communication Time

References

- Dr. Russ Miller's webpage: https://cse.buffalo.edu/faculty/miller/teaching.shtml
- Parallel Computing Sorting https://cs.nyu.edu/courses/spring14/CSCI-UA.0480-003/lecture11.pdf
- https://ubccr.freshdesk.com/support/solutions/articles/13000026245-tutorials-and-training-documents
- http://www.dcc.fc.up.pt/~fds/aulas/PPD/1112/sorting.pdf
- https://www.cs.uky.edu/~jzhang/CS621/chapter7.pdf

Thank you!