702 SEMINAR
PROGRAMMING MASSIVELY PARALLEL SYSTEMS

ODD-EVEN TRANSPOSITION SORT USING MPI

Instructor: Dr. Russ Miller

UB Distinguished Professor

By Hao Wang

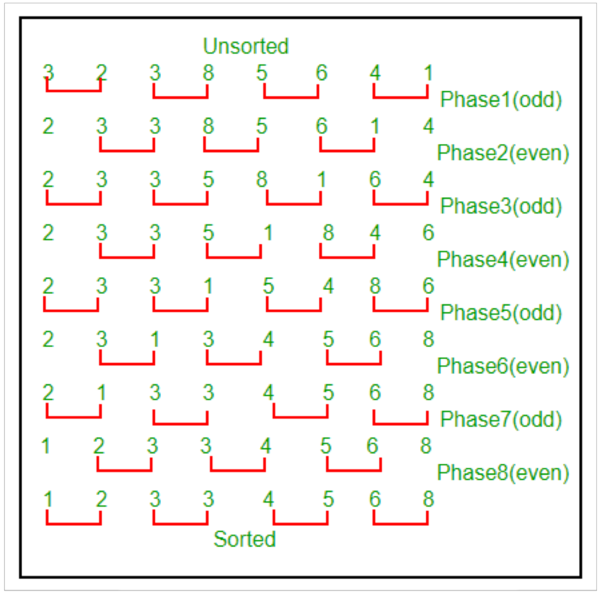
University at Buffalo The State University of New York

- Background
- Sequential algorithm
- Parallel algorithm
- Goal to achieve
- Experiment Results
- Observation
- Reference

• Background

Odd–even transposition sort is a relatively simple sorting algorithm, developed originally for use on parallel processors with local interconnections. It is a comparison sort related to bubble sort, with which it shares many characteristics.

- It functions by comparing all odd/even indexed pairs of adjacent elements in the list and, if a pair is in the wrong order (the first is larger than the second) the elements are switched.
- The next step repeats this for even/odd indexed pairs (of adjacent elements).
- Then it alternates between odd/even and even/odd steps until the list is sorted.
- The total steps of **odd–even transposition sort** will no more than the total number of elements.[1]



(1). https://en.wikipedia.org/wiki/Odd–even_sort

- Sequential algorithm

- Example: given a list of {5, 9, 4, 3}
- 1. odd phase (9, 4) → {5, 4, 9, 3}
- 2. even phase(5,4), (9, 3) → {4, 5, 3, 9}
- 3. odd phase (5, 3) → {4, 3, 5, 9}
- 4. even phase (4, 3), (5, 9) → {3, 4, 5, 9}

- Running time

- The running time is like bubblesort, is simple but not very efficient.
- $O(n^2)$

- Parallel Algorithm (recap)

Suppose we have n elements need to be sorted and p processors. Each processor should be responsible for n/p elements.

1. Sort the local elements in each processor, use a fast sequential sorting algorithm like quick sort.
2. Now each processor contains a local sorted elements
3. Swap processors' elements:
   1. Odd phase: swap (p[1], p[2]), (p[3], p[4]) …
   2. Even phase: swap (p[0], p[1]), (p[2], p[3]) …
4. Since each processor has stored more than one elements, we let the left side processor in the pair keep the smaller half of the elements, the right side processor in the pair keep the larger half of the elements..
5. Keep iterating odd and even phase until all elements are sorted

- SLURM script and config

- SLURM script to run the sorting algorithm on the CCR

```
#!/bin/sh
#SBATCH --partition=general-compute --qos=general-compute
#SBATCH --time=07:00:00
#SBATCH --nodes=16
#SBATCH --constraint=IB
#SBATCH --ntasks-per-node=1
#SBATCH --job-name="sortingTest"
#SBATCH --output=job.node_16_key2.out
#SBATCH --exclusive

echo "SLURM_JOB_ID"=$SLURM_JOB_ID
echo "SLURM_JOB_NODELIST"=$SLURM_JOB_NODELIST
echo "SLURM_NNODES"=$SLURM_NNODES
echo "SLURMTMPDIR"=$SLURMTMPDIR

#The initial srun will trigger the SLURM prologue on the compute  nodes.
module load intel/12.1
module load intel-mpi/5.1.1
#key1: 100,000
#/nodes: 2: 50000   4: 25000    8: 12500  16: 6250   32: 3215   64: 1563

#key2: 200,000
#/nodes: 2: 100000   4: 50000    8: 25000  16: 12500   32: 6250   64: 3215

#key3: 1,000,000
#/nodes: 2: 500000   4: 250000    8: 125000  16: 62500   32: 32150   64: 15625

#key4: 2,000,000
#/nodes: 2: 1000000   4: 500000    8: 250000  16: 125000   32: 62500   64: 32150

nodes=16
key=key2
sep="_"
mpicc -o node_$nodes$sep$key main.c
#arg1: key_size, arg2: nums_nodes
mpirun /user/hwang67/node_$nodes$sep$key 200000 $nodes
echo "sorted"
```

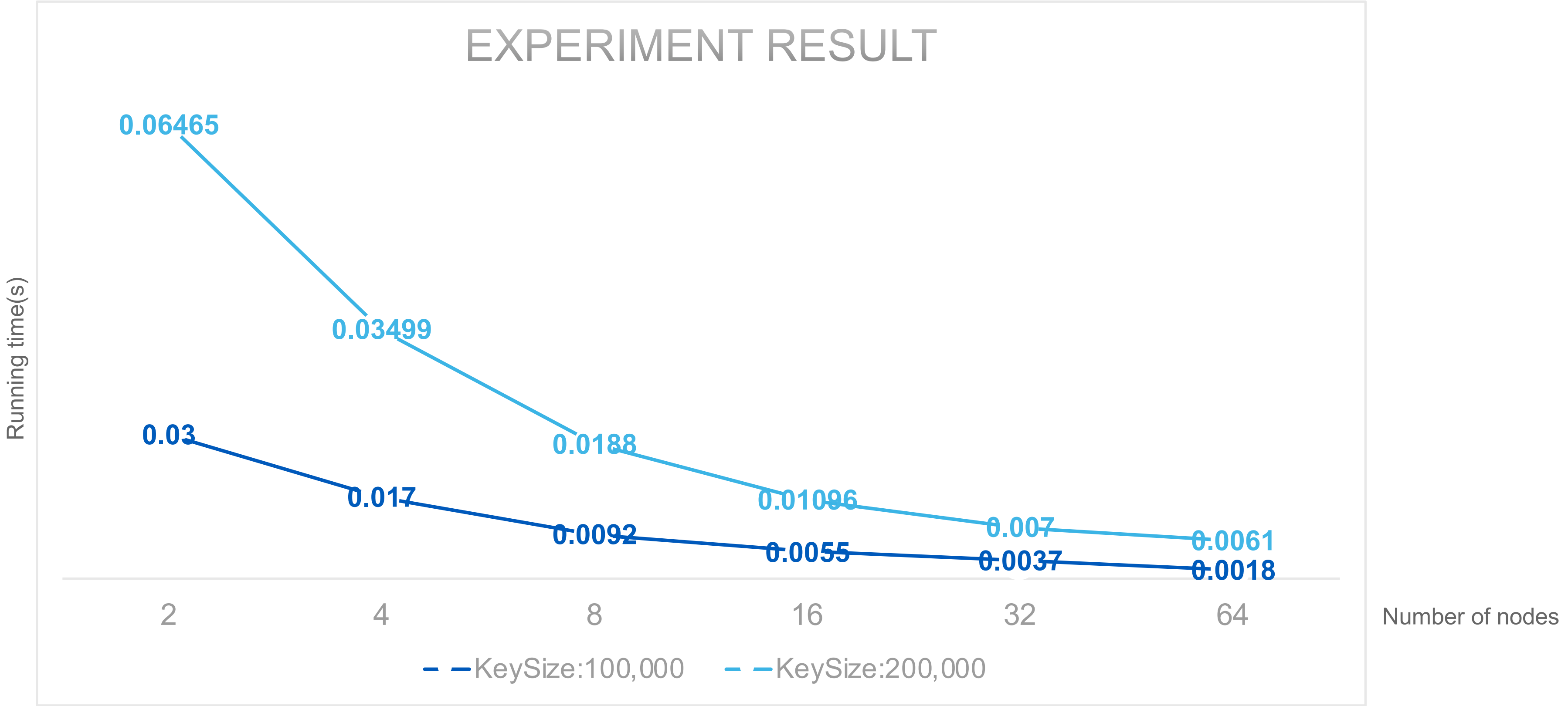
- ## Goal to achieve
  - Use different size of array and different number of processors to implement parallel odd-even transposition sort to get the run time of the algorithm.
  - Make graph of the results achieved and find the relationship between task size, number of processors and the run time.

EXPERIMENT RESULT

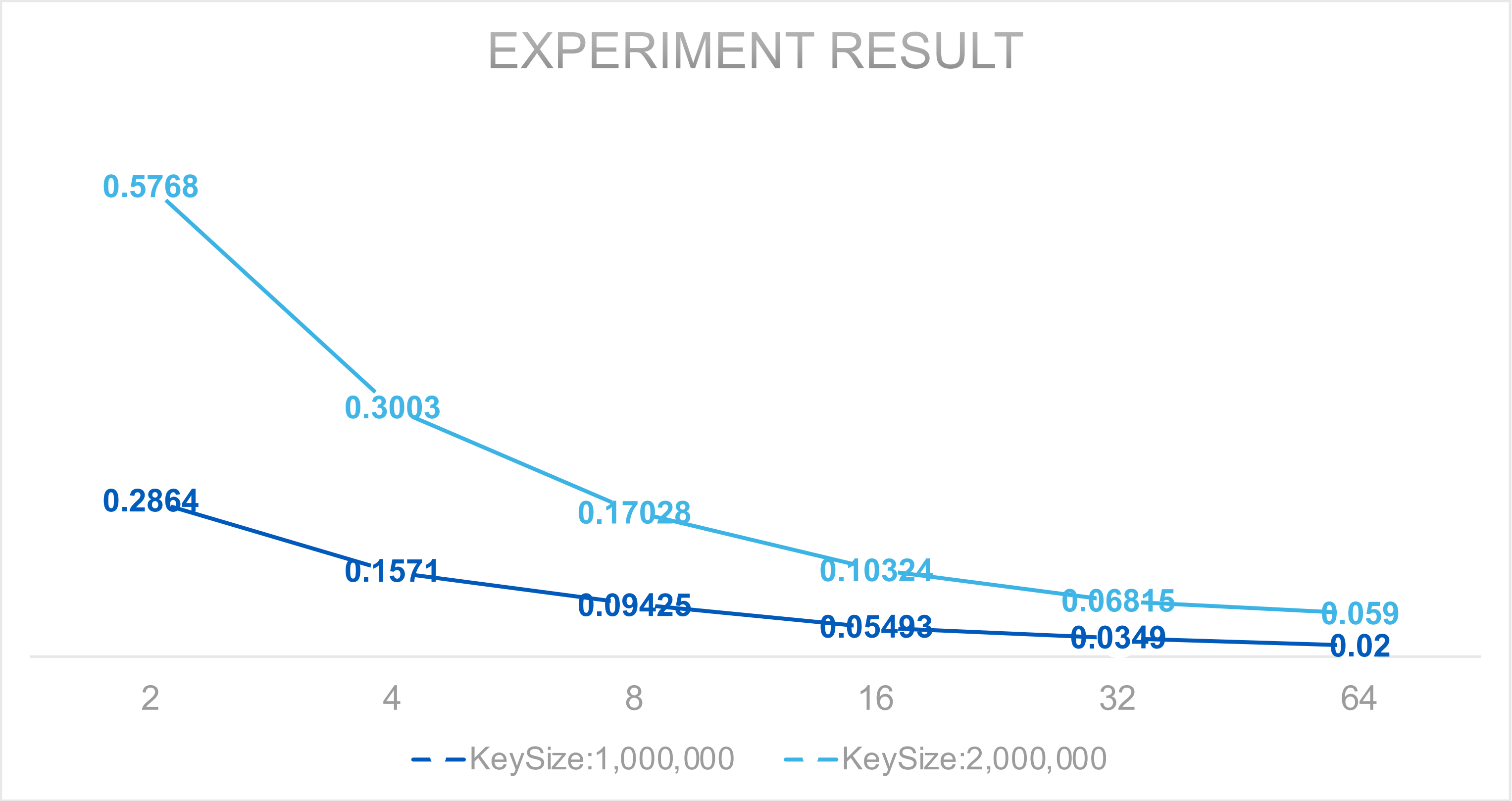
Different key size

Different number of processors

Number of processors :	100,000 (s)	200,000 (s)
2	0.030	0.06465
4	0.0170	0.03499
8	0.0092	0.0188
16	0.0055	0.01096
32	0.0037	0.0070
64	0.0023	0.0059

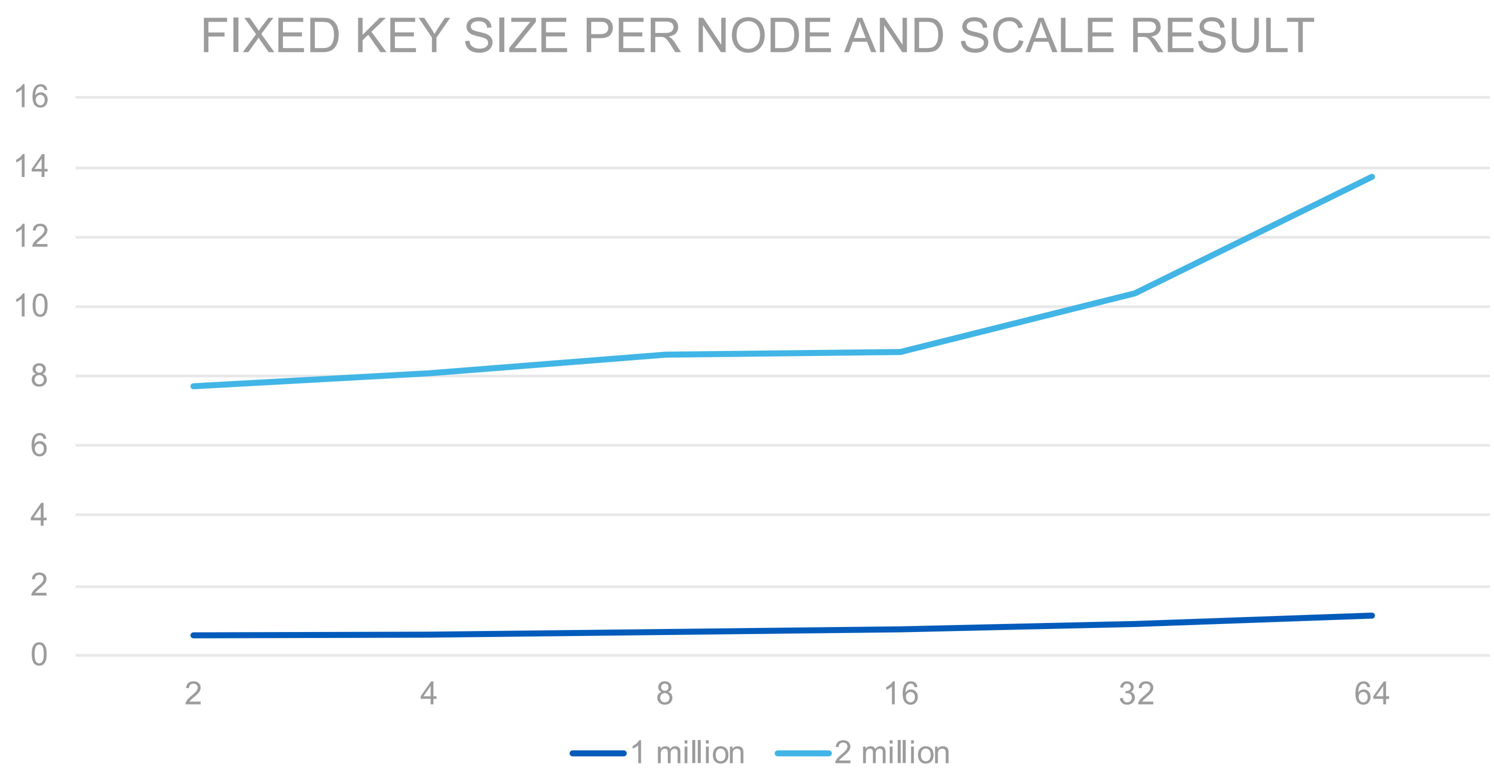

EXPERIMENT RESULT
Running time(s)
0.06465
0.03499
0.0188
0.01096
0.007
0.0061
0.03
0.017
0.0092
0.0055
0.0037
0.0018
2
4
8
16
32
64
Number of nodes
KeySize:100,000
KeySize:200,000

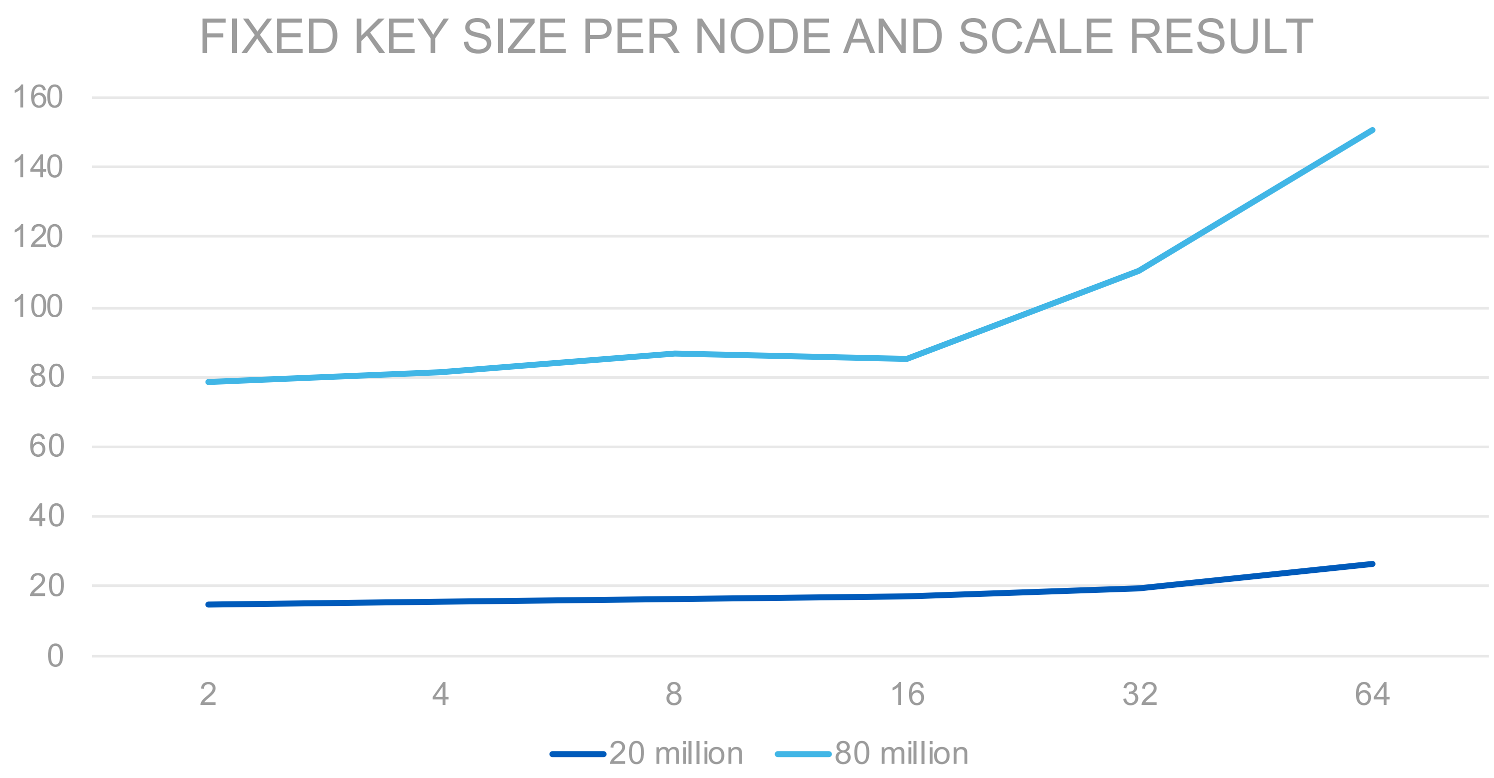
Number of processors :	1,000,000 (s)	2,000,000 (s)
2	0.2864	0.5768
4	0.1571	0.3003
8	0.09425	0.17028
16	0.05493	0.10324
32	0.03490	0.06815
64	0.0278	0.04853


EXPERIMENT RESULT
Running time(s)
0.5768
0.3003
0.17028
0.10324
0.06815
0.059
0.2864
0.1571
0.09425
0.05493
0.0349
0.02
2
4
8
16
32
64
KeySize:1,000,000
KeySize:2,000,000

Fixed keySize and scale Results

Number of processors :	1 million per node (s)	10 million per node (s)	20 million per node (s)	80 million per node (s)
2	0.5768	7.147839	14.78549	63.7654
4	0.609313	7.445081	15.37767	66.1508
8	0.702924	7.895275	16.28316	70.1465
16	0.786155	7.91641	16.76659	68.5454
32	0.865806	9.510384	19.7052	90.932
64	1.142172	12.579163	26.4245	124.3056

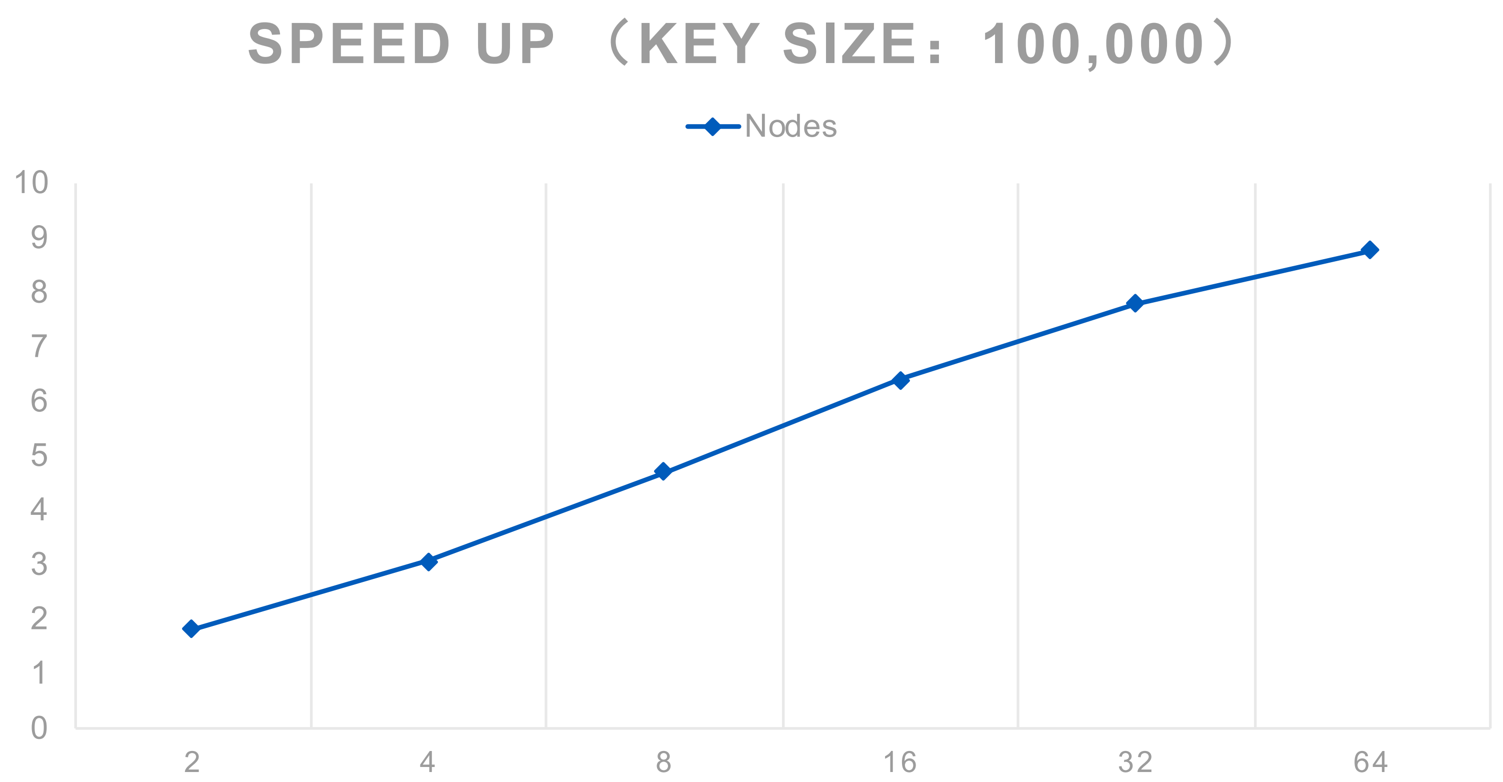

FIXED KEY SIZE PER NODE AND SCALE RESULT
16
14
12
10
8
6
4
2
0
2
4
8
16
32
64
1 million
2 million


FIXED KEY SIZE PER NODE AND SCALE RESULT
160
140
120
100
80
60
40
20
0
2
4
8
16
32
64
20 million
80 million

Speed up

- The speedup is defined as the ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on p processors.

$$S = \frac{T_S}{T_P}$$


SPEED UP（KEY SIZE：100,000）
Nodes
10
9
8
7
6
5
4
3
2
1
0
2
4
8
16
32
64

Observation and Learning

- Record the difference in running time of different size of input and different number of nodes
- Parallel solution can speed up the Odd–even transposition running time
- For the same key size, when the nodes are doubles, the time required to sort the array decreases nearly by the factor of two. However, there is an additional overhead because of communication time between nodes.
- If the key size per node are fixed, while the nodes and total key size are doubled, the running time does not be the same since there is communication time between each nodes while in the sorting process.
- Learning how to work on the CCR

Reference:

- http://mpitutorial.com/ MPI Tutorials
- https://ubccr.freshdesk.com/support/solutions/articles/13000026245-tutorials-and-training-documents CCR Tutorials
- https://en.wikipedia.org/wiki/Odd%E2%80%93even_sort

Thank you