702 SEMINAR
PROGRAMMING MASSIVELY
PARALLEL SYSTEMS

ODD-EVEN TRANSPOSITION SORT
USING MPI

Instructor: Dr. Russ Miller
UB Distinguished Professor

By Hao Wang
• Background
• Sequential algorithm
• Parallel algorithm
• Goal to achieve
• Experiment Results
• Observation
• Reference
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.\(^\text{(1)}\)

\(^\text{(1)}\) https://en.wikipedia.org/wiki/Odd–even_sort
• **Sequential algorithm**

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

• **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 \( \frac{n}{p} \) 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

```bash
#!/bin/sh
#SBATCH --partition=general-compute --qos=general-compute
#SBATCH --time=07:00:00
#SBATCH --nodes=10
#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}

module load intel/12.1
module load intel-mpi/5.1.1

# The first run will trigger the SLURM prologue on the compute nodes.
# Key1: 100,000
# /nodes: 2:50000 4:25000 8:12500 16:6250 32:3125 64:1563
# Key2: 200,000
# /nodes: 2:100000 4:50000 8:25000 16:12500 32:6250 64:3125
# Key3: 1,000,000
# /nodes: 2:500000 4:250000 8:125000 16:62500 32:31250 64:15625
# Key4: 2,000,000
# /nodes: 2:1000000 4:5000000 8:2500000 16:1250000 32:625000 64:312500

mpicc --node $nodes$sep$key main.c
#arg1: key_size, arg2: nums_nodes
mpirun /user/kwangj/node_$nodes$sep$key 200000 $nodes

```

• **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
<table>
<thead>
<tr>
<th>Number of processors</th>
<th>100,000 (s)</th>
<th>200,000 (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.030</td>
<td>0.06465</td>
</tr>
<tr>
<td>4</td>
<td>0.0170</td>
<td>0.03499</td>
</tr>
<tr>
<td>8</td>
<td>0.0092</td>
<td>0.0188</td>
</tr>
<tr>
<td>16</td>
<td>0.0055</td>
<td>0.01096</td>
</tr>
<tr>
<td>32</td>
<td>0.0037</td>
<td>0.0070</td>
</tr>
<tr>
<td>64</td>
<td>0.0023</td>
<td>0.0059</td>
</tr>
</tbody>
</table>
EXPERIMENT RESULT

- KeySize: 100,000
  - Running time(s): 0.06465, 0.03499, 0.0188, 0.01096, 0.007, 0.0061, 0.0018
- KeySize: 200,000
  - Running time(s): 0.03, 0.017, 0.0092, 0.0055, 0.0037, 0.007, 0.0018

Number of nodes:
- 2, 4, 8, 16, 32, 64
<table>
<thead>
<tr>
<th>Number of processors</th>
<th>1,000,000 (s)</th>
<th>2,000,000 (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.2864</td>
<td>0.5768</td>
</tr>
<tr>
<td>4</td>
<td>0.1571</td>
<td>0.3003</td>
</tr>
<tr>
<td>8</td>
<td>0.09425</td>
<td>0.17028</td>
</tr>
<tr>
<td>16</td>
<td>0.05493</td>
<td>0.10324</td>
</tr>
<tr>
<td>32</td>
<td>0.03490</td>
<td>0.06815</td>
</tr>
<tr>
<td>64</td>
<td>0.0278</td>
<td>0.04853</td>
</tr>
</tbody>
</table>
EXPERIMENT RESULT

Running time (s)

KeySize: 1,000,000
KeySize: 2,000,000

2 4 8 16 32 64
Fixed keySize and scale Results

<table>
<thead>
<tr>
<th>Number of processors :</th>
<th>1 million per node (s)</th>
<th>10 million per node (s)</th>
<th>20 million per node (s)</th>
<th>80 million per node (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5768</td>
<td>7.147839</td>
<td>14.78549</td>
<td>63.7654</td>
</tr>
<tr>
<td>4</td>
<td>0.609313</td>
<td>7.445081</td>
<td>15.37767</td>
<td>66.1508</td>
</tr>
<tr>
<td>8</td>
<td>0.702924</td>
<td>7.895275</td>
<td>16.28316</td>
<td>70.1465</td>
</tr>
<tr>
<td>16</td>
<td>0.786155</td>
<td>7.91641</td>
<td>16.76659</td>
<td>68.5454</td>
</tr>
<tr>
<td>32</td>
<td>0.865806</td>
<td>9.510384</td>
<td>19.7052</td>
<td>90.932</td>
</tr>
<tr>
<td>64</td>
<td>1.142172</td>
<td>12.579163</td>
<td>26.4245</td>
<td>124.3056</td>
</tr>
</tbody>
</table>
FIXED KEY SIZE PER NODE AND SCALE RESULT
FIXED KEY SIZE PER NODE AND SCALE RESULT

- 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

0 10 20 30 40 50 60 70 80 90 100

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/](http://mpitutorial.com/) MPI Tutorials
- [https://en.wikipedia.org/wiki/Odd%E2%80%93even_sort](https://en.wikipedia.org/wiki/Odd%E2%80%93even_sort)
Thank you