Learning and Implementing Odd-even transposition sort in OpenMP
PREPARED BY:

ASIF IMRAN (UB PERSON NUMBER: 50249959)

PREPARED FOR:

PROF. DR. RUSS MILLER

UB DISTINGUISHED PROFESSOR,

DEPARTMENT OF COMPUTER SCIENCE AND

ENGINEERING, UNIVERSITY AT BUFFALO (SUNY)
Overview

- Overview of Odd Even Transposition Sort
- Discussions of Goals and Assumptions
- Obtained results
- PE’s behavior
- Discussion of results
- References
Background

- Aim to enhance knowledge gained on parallel programming
- Implemented the project on OpenMP, different model than MPI, therefore obtaining detailed knowledge on different aspects of parallel programming
- Harness massively parallel computing machines at CCR
- Used OpenMP directive on machines with 2, 4, 8 and 16 cores respectively.
Consider Bubble Sort

- Bubble sort is a $O(N^2)$ sorting algorithm.
- It is simple to understand and implement.

So why discuss it?
- Understandable
- Implementable
- Can be parallelized
Odd Even Transposition sort

• Parallelizable version of Bubble sort
• Requires N passes through the array.
• Each pass through the array analyzes either:
  • Every pair of odd indexed elements and the preceding element, or
  • Every pair of even indexed elements and the preceding element.
• Within each pass, elements that are not in order are swapped.
Pictorial depiction

- Even positions

\((a[0], a[1]), (a[2], a[3]), (a[4], a[5]), \ldots,\)

- Odd positions

\((a[1], a[2]), (a[3], a[4]), (a[5], a[6]), \ldots,\)
Goals of this project

- Run OpenMP code
- One node should have multiple threads
- Aim is to allocate nodes with largest number of cores (i.e. 16 cores).
- Sorted dataset of varying sizes
- Used standard dataset
- There are 2621440 integers in the dataset which is used with a total size of 10.48 MB
- Take integer blocks of varying sizes
Goals of this project (cont)

- We took two groups of data containing sub-groups amongst them:
  - Small Group: 0.00625, 0.0125 and 0.025 million
  - Large Group: 0.05, 0.1, 0.125, 0.25, 0.50, 1.00, 1.5 and 2.00 million
- The reason for using two subgroups was to demonstrate how OpenMP performs to sort various groups of data.
- Provide Runtime graphs when data size and nodes are both doubled
- Provide Runtime graphs when data size is constant and nodes are doubled
Process flow

1. Log into vortex.ocf.buffalo.edu
2. Send program to front end machine
3. Allocate 1 node with multiple PEs
4. ssh to cpn-115-YY in ub-hpc cluster
5. Gather logs
6. Report results
SLURM output

```
[asifimra@vortex2:~]$ export | grep SLURM
declare -x SLURM_CLUSTER_NAME="ub-hpc"
declare -x SLURM_JOBID="9947720"
declare -x SLURM_JOB_CPUS_PER_NODE="16"
declare -x SLURM_JOB_ID="9947720"
declare -x SLURM_JOB_NAME="bash"
declare -x SLURM_JOB_NODELIST="cpn-f16-13"
declare -x SLURM_JOB_NUM_NODES="1"
declare -x SLURM_JOB_PARTITION="general-compute"
declare -x SLURM_MEM_PER_CPU="2800"
declare -x SLURM_NNODES="1"
declare -x SLURM_NODELIST="cpn-f16-13"
declare -x SLURM_NODE_ALIASES="(null)"
declare -x SLURM_NPROCS="16"
declare -x SLURM_NTASKS="16"
```
```
**top** - 23:08:37 up 18 days, 16:27, 2 users, load average: 0.04, 0.78, 1.39
Tasks: 279 total, 2 running, 276 sleeping, 0 stopped, 1 zombie
Cpu(s): 99.6 us, 0.1 sy, 0.0 ni, 0.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
Mem: 13181312+total, 15373354 free, 2009152 used, 11353060+buff/cache
Swap: 13195673+total, 13184179+free, 114944 used, 12649508+avail Mem

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>17332</td>
<td>asifim</td>
<td>20</td>
<td>0</td>
<td>139480</td>
<td>8744</td>
<td>664</td>
<td>R</td>
<td>1591</td>
<td>0.0</td>
<td>25:24.72 sortOpenMP</td>
</tr>
<tr>
<td>3383</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>332120</td>
<td>46496</td>
<td>4596</td>
<td>S</td>
<td>2.0</td>
<td>0.0</td>
<td>173:13.17 perl</td>
</tr>
<tr>
<td>32012</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>266520</td>
<td>165324</td>
<td>2072</td>
<td>R</td>
<td>1.3</td>
<td>0.1</td>
<td>41:42.02 pmddproc</td>
</tr>
<tr>
<td>11994</td>
<td>asifim</td>
<td>20</td>
<td>0</td>
<td>5648664</td>
<td>199368</td>
<td>644</td>
<td>S</td>
<td>0.3</td>
<td>0.2</td>
<td>37:48.12 dsm Omコン</td>
</tr>
<tr>
<td>17336</td>
<td>asifim</td>
<td>20</td>
<td>0</td>
<td>172480</td>
<td>2496</td>
<td>1648</td>
<td>R</td>
<td>0.3</td>
<td>0.0</td>
<td>0:09:98 top</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>192072</td>
<td>3612</td>
<td>2200</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>4:19:82 systemd</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:25 kthreadd</td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:47:69 ksoftirqd/0</td>
</tr>
<tr>
<td>5</td>
<td>root</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00 kworker/0+</td>
</tr>
<tr>
<td>8</td>
<td>root</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:55 migration/0</td>
</tr>
<tr>
<td>9</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00 rcu_bh</td>
</tr>
<tr>
<td>10</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>5:23:54 rcu_sched</td>
</tr>
<tr>
<td>11</td>
<td>root</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00 lrur-add-dr+</td>
</tr>
<tr>
<td>12</td>
<td>root</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>1:31:48 watchdog/0</td>
</tr>
<tr>
<td>13</td>
<td>root</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:19:04 watchdog/1</td>
</tr>
<tr>
<td>14</td>
<td>root</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:22 migration/1</td>
</tr>
<tr>
<td>15</td>
<td>root</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00:00 rcuosallen</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>29309</td>
<td>asifim</td>
<td>20</td>
<td>0</td>
<td>81720</td>
<td>16476</td>
<td>660</td>
<td>R</td>
<td>800.0</td>
</tr>
<tr>
<td>37786</td>
<td>asifim</td>
<td>20</td>
<td>0</td>
<td>48936</td>
<td>16452</td>
<td>668</td>
<td>R</td>
<td>400.0</td>
</tr>
<tr>
<td>31888</td>
<td>asifim</td>
<td>20</td>
<td>0</td>
<td>32544</td>
<td>16432</td>
<td>668</td>
<td>R</td>
<td>200.0</td>
</tr>
</tbody>
</table>
```
Obtained results

Running time for data size of 6250

Running time when data size is 12500
Obtained results [cont]

Run time when data size is 25000

Run time when data size is 50000
Obtained Results [cont]

Run time when data size is 100000

Run time for data size 125000
Obtained Results [cont]

Run time for datasize 250000

Run time when data size is 500000
Obtained Results [cont]

Run time for data size 1 million

Run time when data size is 1.5M
Results [cont]

Run time when data size is 2M
Results (cont.) with 32 threads (i.e. PEs)

- For 32 cores:
  - Difficult to obtain a single server with 32 cores by `salloc` command in CCR.
  - Experiments were run in vortex1 front end of CCR which was equipped with 32 cores.
  - It was conducted during off-peak hours (between 3:00 AM – 4:00 AM EST)
Double data size and PEs (threads)

Along X-axis, a value of P implies $2^P$ PEs
Speedup
Outcome

- Presented results of the runtimes in a multitude of dimensions
- Discussions
  - OpenMP’s runtime is desirable for considerably large datasets
  - Performance degrades when large number of PEs are used for data of size 25000 or lower
  - Desirable behavior when we double data and PEs

*Made good use of free resources during thanksgiving*
References

- Dr. Russ Miller’s webpage:
  https://cse.buffalo.edu/faculty/miller/teaching.shtml

- Parallel Odd Even Transposition sort:

Thank you