Parallel Odd-Even Transposition Sort using MPI

CSE 633: Parallel Algorithms

Course Instructor: Dr. Russ Miller
UB Distinguished Professor

Department of Computer Science & Engineering
State University of New York at Buffalo

Prepared by: Asif Imran (UB Person number: 50249959)
Agenda

• Overview of the project
• Proposed algorithm with justification
• Architecture of the solution
• Experimentation in CCR
• Obtained results and analysis
• Challenges
• Learnings
• Conclusion and Future Work
Overview of the project

Odd-Even Transposition sorting

Think of Bubble sort
Think of bubble sort....

- Unrealistic to parallelize
- Inherently sequential nature of the sort algorithms

- Why Odd-Even Transposition sort?
  - Bigger opportunity to parallelize
  - Key idea is to decouple the compare swaps
  - Consists of two different phases of sequence
  - For example: During even phases, compare swaps are executed on the even pairs and vice versa.
Goal of the project

• Design, implementation, and analyze parallel solution of interest on modern large-scale multiprocessor/multi-core systems. [1]

• Acclimatization to real life high performance multiprocessor computing environment and obtaining knowledge on how to use them.

• Use Foster’s method [2]

• Use Amdahl’s law for calculation of speedup [2]
Ian Foster’s method

Start

Obtain large task

S-1

A

B

C

S-2

A'

B'

C'

Output final result

S-4

S-3
Pictorial depiction of odd-even sort mechanism

• Even positions

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

• Odd positions

\[(a[1], a[2]), (a[3], a[4]), (a[5], a[6]), \ldots,\]
Architecture of Odd-Even Transposition sort
Experimentation

• Involved allocation of resources followed by execution of code to collect run-time
• Used script file
• Specified number of servers
• Specified number of CPUs
• Specified number of tasks per process
• Obtained –exclusive access to the resources
• Calculated speedup values using Amdahl’s law
Script for running SLURM jobs

```
!/bin/sh
SBATCH --salloc
SBATCH --partition=general-compute --qos=general-compute
SBATCH --time=1:00:00
SBATCH --nodes=16
SBATCH --ntasks-per-node=1
SBATCH --constraint=IB
SBATCH --job-name= "Odd_Even"
SBATCH --mail-user=asifimra@buffalo.edu
SBATCH --mail-type=ALL
SBATCH --requeue
#
# The initial srun will trigger the SLURM prologue on the compute nodes.
#
I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
mpirun --np 16 ./oddeven2
echo "All Done!"
```
## Server Configuration [4]

<table>
<thead>
<tr>
<th>Type of Node</th>
<th>Approximate # of Nodes</th>
<th># Cores per Node</th>
<th>Clock Rate</th>
<th>RAM</th>
<th>Network*</th>
<th>SLURM TAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute</td>
<td>372</td>
<td>12</td>
<td>2.40GHz</td>
<td>48GB</td>
<td>Infiniband (QL)</td>
<td>IB CPU-E5645</td>
</tr>
</tbody>
</table>
**Key size: 100000**

<table>
<thead>
<tr>
<th>Processors</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.02555</td>
</tr>
<tr>
<td>4</td>
<td>0.02442</td>
</tr>
<tr>
<td>8</td>
<td>0.02172</td>
</tr>
<tr>
<td>16</td>
<td>0.01381</td>
</tr>
<tr>
<td>32</td>
<td>0.01655</td>
</tr>
<tr>
<td>64</td>
<td>0.0264</td>
</tr>
</tbody>
</table>
Key size: 200000

<table>
<thead>
<tr>
<th>Processors</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.896</td>
</tr>
<tr>
<td>4</td>
<td>1.6833</td>
</tr>
<tr>
<td>8</td>
<td>1.2287</td>
</tr>
<tr>
<td>16</td>
<td>1.1688</td>
</tr>
<tr>
<td>32</td>
<td>0.934</td>
</tr>
<tr>
<td>64</td>
<td>1.07</td>
</tr>
<tr>
<td>128</td>
<td>1.311</td>
</tr>
<tr>
<td>256</td>
<td>1.610</td>
</tr>
<tr>
<td>Processors</td>
<td>Speedup</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>2</td>
<td>30.609</td>
</tr>
<tr>
<td>4</td>
<td>19.447</td>
</tr>
<tr>
<td>8</td>
<td>10.799</td>
</tr>
<tr>
<td>16</td>
<td>4.649</td>
</tr>
<tr>
<td>32</td>
<td>2.873</td>
</tr>
<tr>
<td>64</td>
<td>1.329</td>
</tr>
<tr>
<td>128</td>
<td>0.901</td>
</tr>
</tbody>
</table>

Key size: 1000000 (1 million)
### Key size: 200000 (2 million)

<table>
<thead>
<tr>
<th>Processors</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>48.905</td>
</tr>
<tr>
<td>4</td>
<td>17.312</td>
</tr>
<tr>
<td>8</td>
<td>12.688</td>
</tr>
<tr>
<td>16</td>
<td>8.491</td>
</tr>
<tr>
<td>32</td>
<td>4.142</td>
</tr>
<tr>
<td>64</td>
<td>1.464</td>
</tr>
<tr>
<td>128</td>
<td>0.996</td>
</tr>
</tbody>
</table>

![Graph showing speedup and time relationship with different processors](Key_size_2000000_two_million.png)
### Speedup

#### Key size: 100000

<table>
<thead>
<tr>
<th>Processors</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4.618395303</td>
</tr>
<tr>
<td>4</td>
<td>4.832104832</td>
</tr>
<tr>
<td>8</td>
<td>5.432780847</td>
</tr>
<tr>
<td>16</td>
<td>8.544532947</td>
</tr>
<tr>
<td>32</td>
<td>7.129909366</td>
</tr>
<tr>
<td>64</td>
<td>4.46969697</td>
</tr>
</tbody>
</table>

![Speedup Graph](image-url)
Speedup [cont]

<table>
<thead>
<tr>
<th>Processors</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3.372</td>
</tr>
<tr>
<td>4</td>
<td>3.798</td>
</tr>
<tr>
<td>8</td>
<td>5.203</td>
</tr>
<tr>
<td>16</td>
<td>5.47</td>
</tr>
<tr>
<td>32</td>
<td>6.845</td>
</tr>
</tbody>
</table>
Speedup

• Amdahl’s law

\[ T_{\text{parallel}} = 0.9 \times T_{\text{serial}}/p + 0.1 \times T_{\text{serial}} = \frac{18}{p} + 2, \]

\[ S = \frac{T_{\text{serial}}}{0.9 \times \frac{T_{\text{serial}}}{p} + 0.1 \times T_{\text{serial}}} = \frac{20}{\frac{18}{p} + 2} \]
SLURM Job details for CPU = 2

[[asifimra@rush:~]$ scontrol show job 8751334
JobId=8751334 JobName=odd_even
  UserId=asifimra(549081) GroupId=cse633s18(8200175) MCS_label=N/A
  Priority=50214 Nice=0 Account=cse633s18 QOS=general-compute
  JobState=TIMEOUT Reason=TimeLimit Dependency=(null)
  Requeue=0 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
  RunTime=00:15:00 TimeLimit=00:15:00 TimeMin=N/A
  PreemptTime=0 None SuspendTime=0 SecsPreSuspend=0
  Partition=general-compute AllocNodeList=drv-k07-1x:37483
  ReqNodeList=(null) ExecNodeList=(null)
  NodeList=cpn-d14-[12,30]
  BatchHost=cpn-d14-12
  NumNodes=2 NumCPUs=2 NumTasks=2 CPUs/Task=1 ReqB:S:C:T=0:0:0:*:*
  TRES=cpu2,mem=46000M,node=2
  Socs/Node== NtasksPerNode:S=C=1:* CoreSpec=*
  MinCPUsNode=1 MinMemoryNode=23000M MinTmpDiskNode=0
  Features=10 DelayBoot=0:00:00
  Grps=(null) Reservation=(null)
  OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
  Command=/user/asifimra/myscript.sh
  WorkDir=/user/asifimra
  StdErr=/user/asifimra/test-srun.out
  StdIn=/dev/null
  StdOut=/user/asifimra/test-srun.out
  Power=
Challenges

• Long time to provision 64, 126 and 256 cores
• Unexpected service unavailability due to emergency.
Learning from the course

• Viewed the difference in run time as cores are increased
• Noticed how high performance computing systems and parallelization can speed up performance compared to sequential runs.
• Knowledge on MPI, Intel MPI and Open MPI systems
• Visit and seeing CCR infrastructure
Conclusion and future goals

• Results show that there should be an optimum number of CPU’s which need to be allocated for the data load
• Each physical server initiated 1 process only

• Future Goal:
  • Extend this code to OpenMP and compare performance in CSE 702
References


Thank you