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)

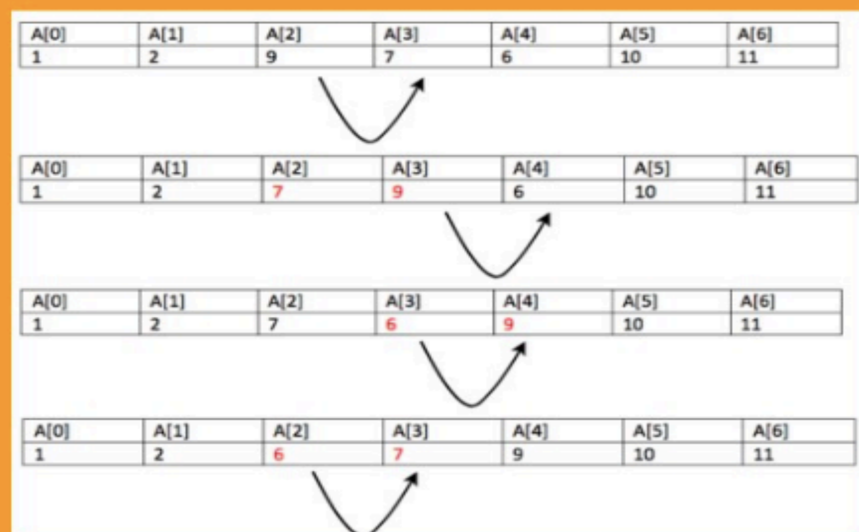
Agenda

- 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 fo Bubble sort



Think of bubble sort….

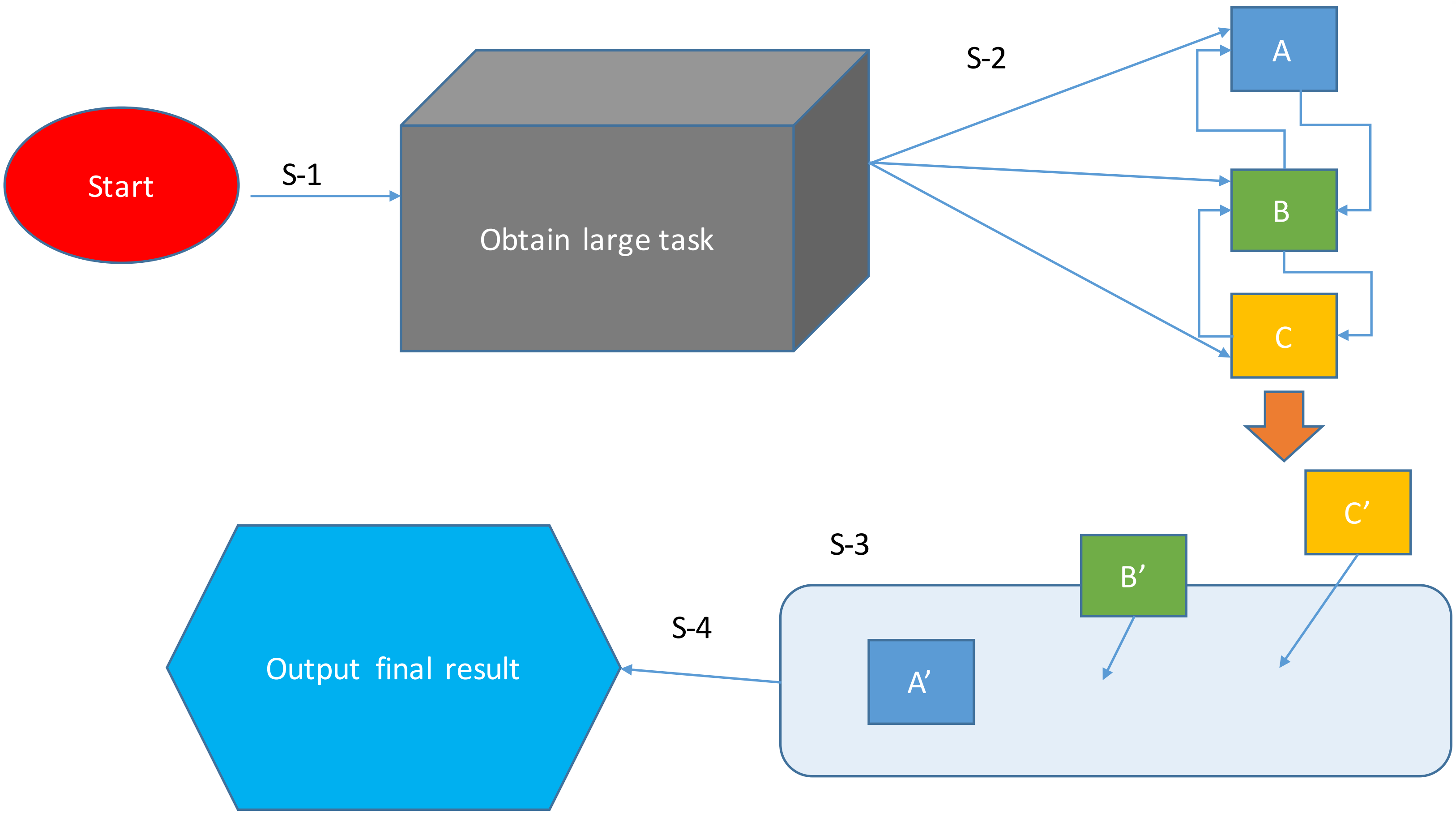
- Unrealistic to parallelize
- Inherently sequential nature of the sort algorithms

- 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]

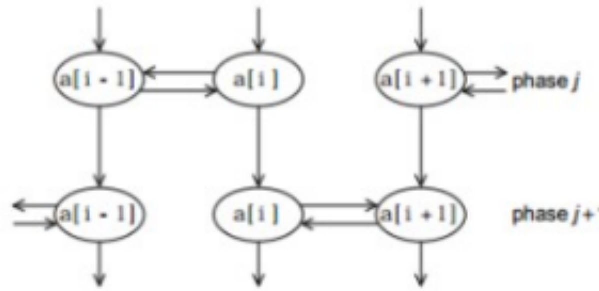
Ian Foster's method



Pictorial depiction of odd-even sort mechanism

- Even positions

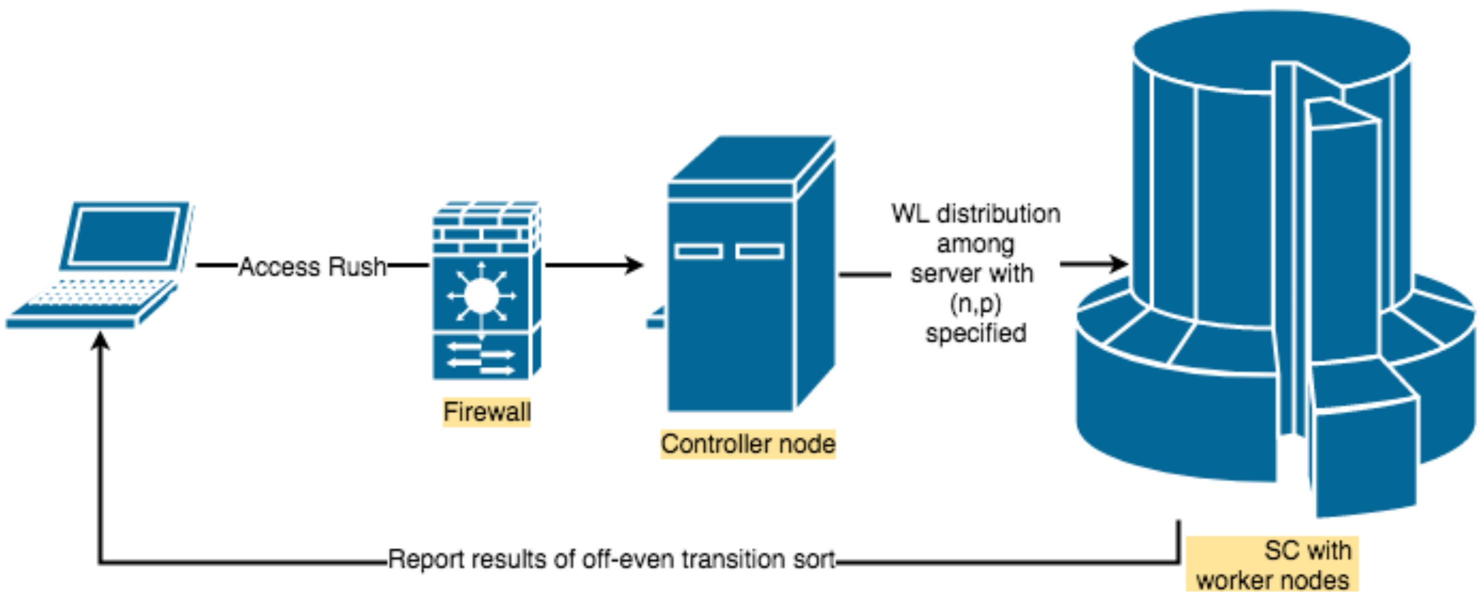
$$(a[0], a[1]), (a[2], a[3]), (a[4], a[5]), \ldots,$$



- 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

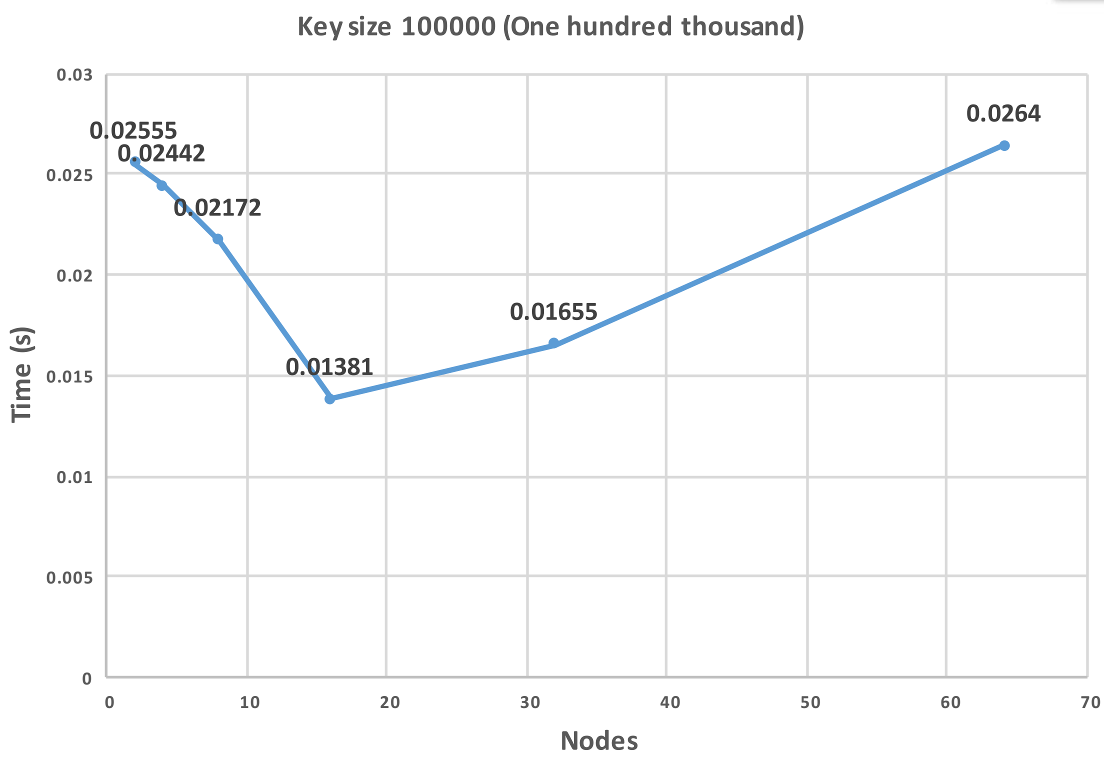
Script for running SLURM jobs

```
!/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 srun
mpirun –np 16 ./oddeven2
echo "All Done!"
```

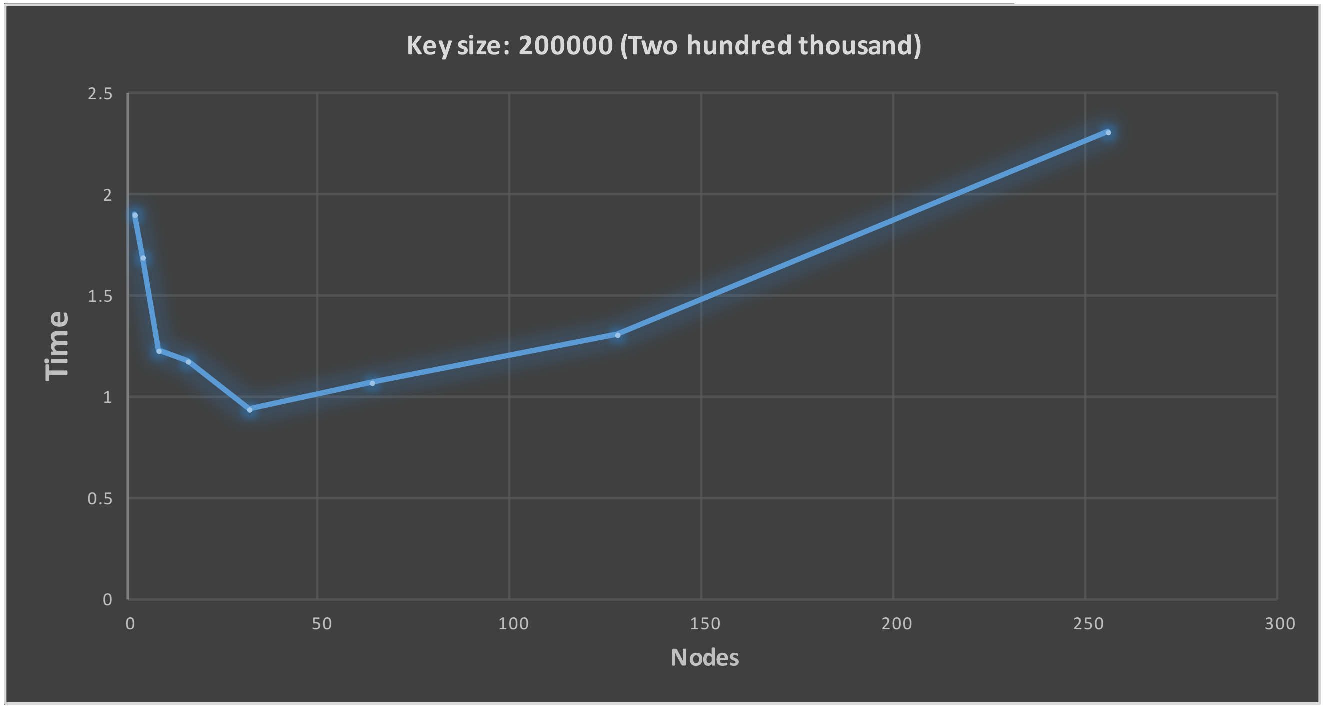
Server Configuration [4]

Type of Node	Approximate # of Nodes	# Cores per Node	Clock Rate	RAM	Network*	SLURM TAGS
Compute	372	12	2.40GHz	48GB	Infiniband (QL)	IB CPU-E5645

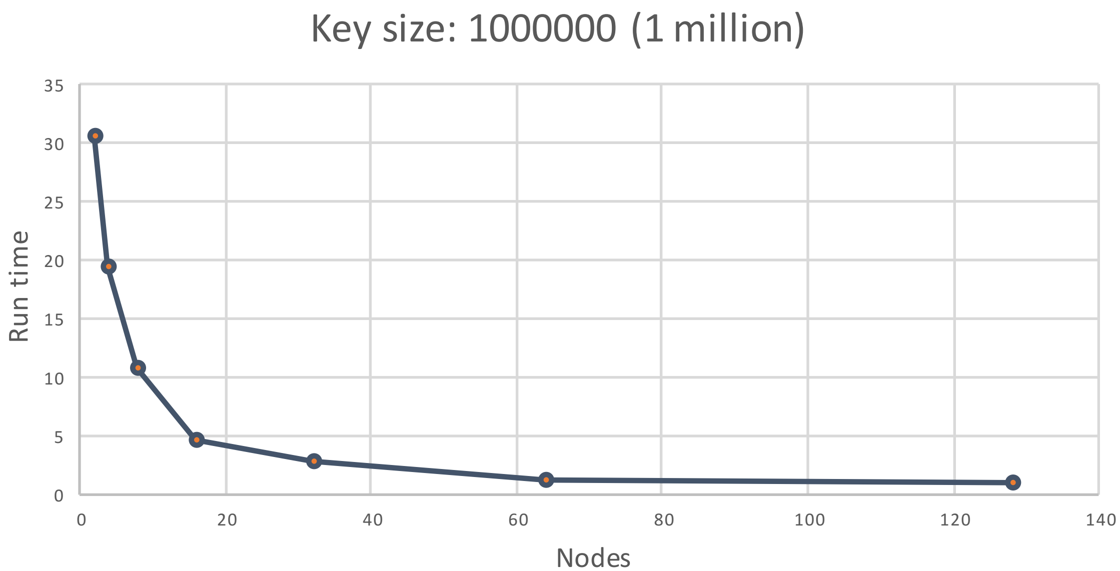
Key size: 100000	
Processors	Time
2	0.02555
4	0.02442
8	0.02172
16	0.01381
32	0.01655
64	0.0264



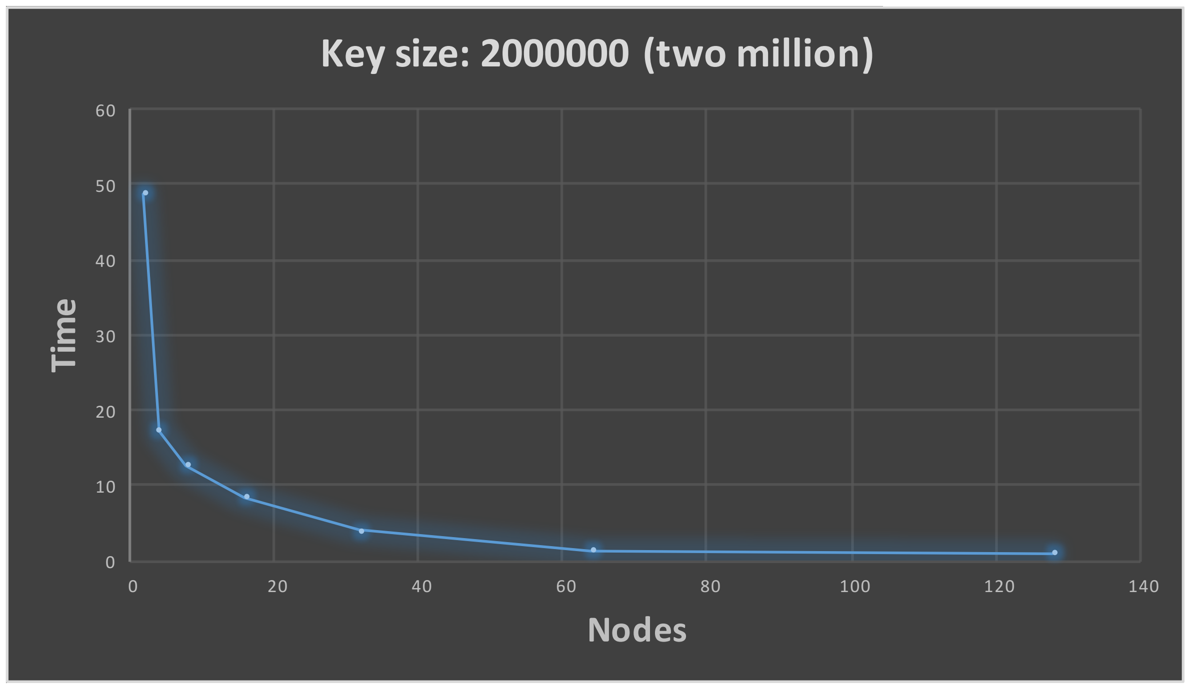
Key size: 200000	
Processors	Time
2	1.896
4	1.6833
8	1.2287
16	1.1688
32	0.934
64	1.07
128	1.311
256	1.610



Key size: 1000000 (1 million)	
Processors	Speedup
2	30.609
4	19.447
8	10.799
16	4.649
32	2.873
64	1.329
128	0.901

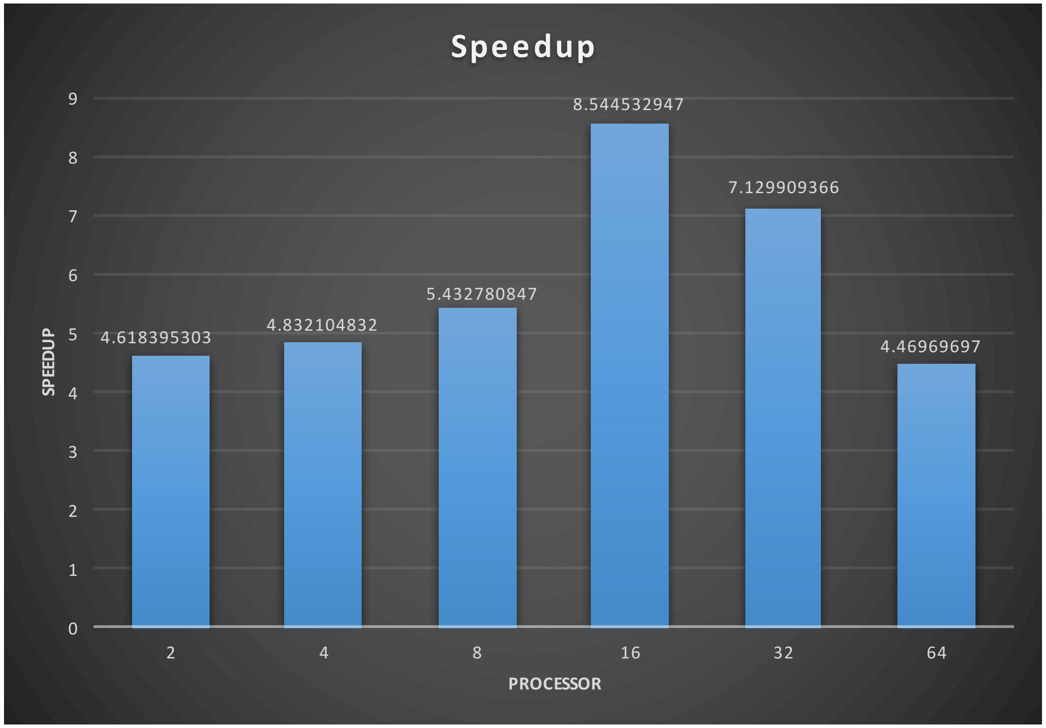


Key size: 200000 (2 million)	
Processors	Speedup
2	48.905
4	17.312
8	12.688
16	8.491
32	4.142
64	1.464
128	0.996



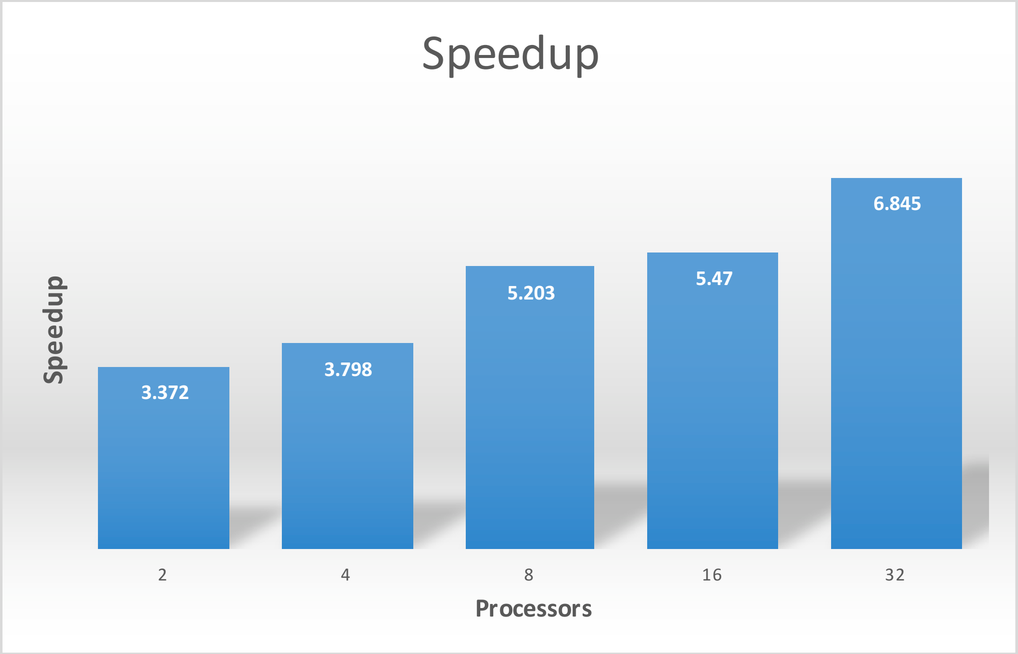
Speedup

Key size: 100000	
Processors	Speedup
2	4.618395303
4	4.832104832
8	5.432780847
16	8.544532947
32	7.129909366
64	4.46969697



Speedup [cont]

Key size: 200000	
Processors	Speedup
2	3.372
4	3.798
8	5.203
16	5.47
32	6.845



Speedup

- Amdahl's law

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

$$S = \frac{T_{\text{serial}}}{0.9 \times T_{\text{serial}}/p + 0.1 \times T_{\text{serial}}} = \frac{20}{18/p + 2}$$

SLURM Job details for CPU = 2

```
[[asifimra@rush:~]$ scontrol show job 8751334
 JobId=8751334 JobName=odd_even
    UserId=asifimra(549091) GroupId=cse633s18(89200175) 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=0:15
    RunTime=00:15:08 TimeLimit=00:15:00 TimeMin=N/A
    SubmitTime=2018-04-24T22:34:55 EligibleTime=2018-04-24T22:34:55
    StartTime=2018-04-24T22:41:39 EndTime=2018-04-24T22:56:47 Deadline=N/A
    PreemptTime=None SuspendTime=None SecsPreSuspend=0
    Partition=general-compute AllocNode:Sid=srv-k07-14:37483
    ReqNodeList=(null) ExcNodeList=(null)
    NodeList=cpn-d14-[12,36]
    BatchHost=cpn-d14-12
    NumNodes=2 NumCPUs=2 NumTasks=2 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
    TRES=cpu=2,mem=46000M,node=2
    Socks/Node=* NtasksPerN:B:S:C=1:0:*:* CoreSpec=*
    MinCPUsNode=1 MinMemoryNode=23000M MinTmpDiskNode=0
    Features=IB DelayBoot=00:00:00
    Gres=(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

[1] https://www.cse.buffalo.edu//faculty/miller/teaching.shtml

[2] Pacheco, P.S., 1997. *Parallel programming with MPI*. Morgan Kaufmann.

[3] Foster, I., Zhao, Y., Raicu, I. and Lu, S., 2008, November. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*(pp. 1-10). IEEE.

[4] Academic Compute Cluster (UB-HPC). Link: https://www.buffalo.edu/ccr/support/research_facilities/general_compute.html

Thank you