CSE 702: SEMINAR ON PROGRAMMING MASSIVELY PARALLEL SYSTEMS

Learning and Implementing Odd-even transposition sort in OpenMP

University at Buffalo
School of Engineering and Applied Sciences

University at Buffalo
School of Engineering and Applied Sciences

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.

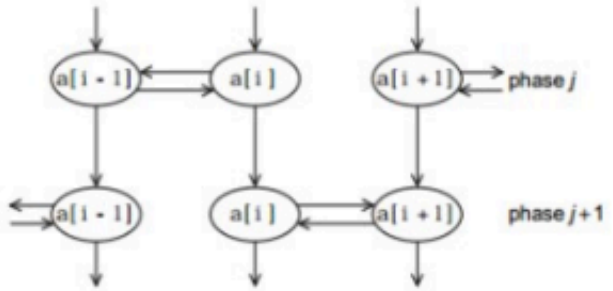
Pictorial depiction

- Even positions

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

- Odd positions

$$(a[1], a[2]), (a[3], a[4]), (a[5], a[6]), \ldots.$$



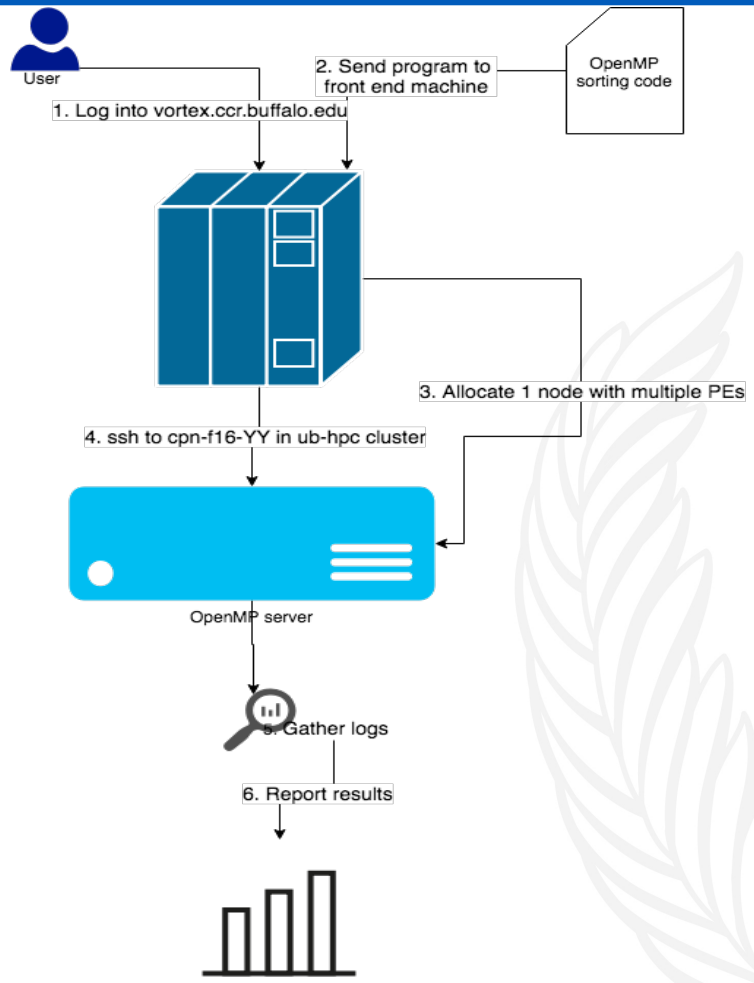
Goals of this project

- 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



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"
```

asif — asifimra@cpn-f16-13:~ — ssh -X asifimra@vortex.ccr.buffalo.edu — 80...

```
top - 23:08:37 up 18 days, 15:27,  2 users,  load average: 8.04, 2.78, 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
KiB Mem : 13181312+total, 15373364 free,  2909152 used, 11353060+buff/cache
KiB Swap: 13195673+total, 13184179+free,   114944 used. 12649500+avail Mem
```

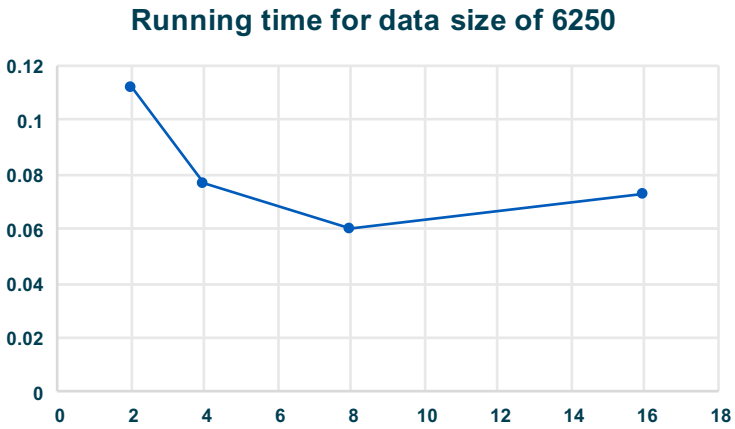
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
L7332	asifimra	20	0	139480	8744	664	R	1591	.0	25:24.72	sortOpenMP
3383	root	20	0	332120	46496	4596	S	2.0	.0	173:13.17	perl
32012	root	20	0	266520	165324	2072	S	1.3	.1	41:42.82	pmdaproc
L1994	root	20	0	5640664	199368	6444	S	0.3	0.2	37:48.12	dsm_om_con+
L7336	asifimra	20	0	172400	2496	1648	R	0.3	0.0	0:03.98	top
1	root	20	0	192072	3612	2280	S	0.0	0.0	4:19.82	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.25	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:47.69	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:+
8	root	rt	0	0	0	0	S	0.0	0.0	0:00.55	migration/0
9	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
10	root	20	0	0	0	0	S	0.0	0.0	5:23.54	rcu_sched
11	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	lru-add-dr+
12	root	rt	0	0	0	0	S	0.0	0.0	1:31.48	watchdog/0
13	root	rt	0	0	0	0	S	0.0	0.0	0:19.04	watchdog/1
14	root	rt	0	0	0	0	S	0.0	0.0	0:00.22	migration/1

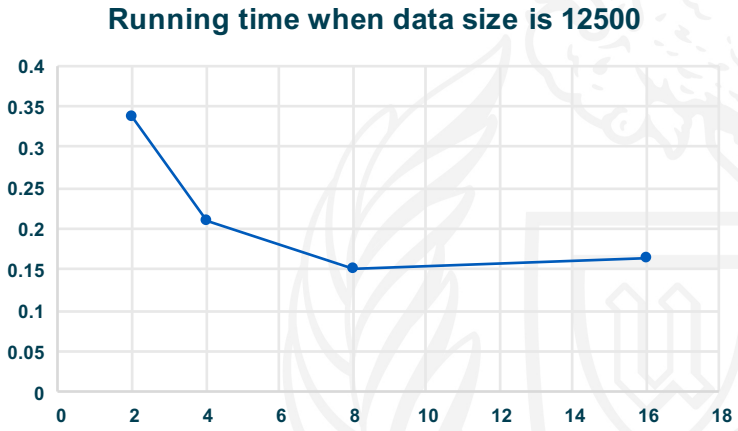
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU
29309	asifimra	20	0	81720	16476	660	R	800.0

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU
37786	asifimra	20	0	48936	16452	668	R	400.0

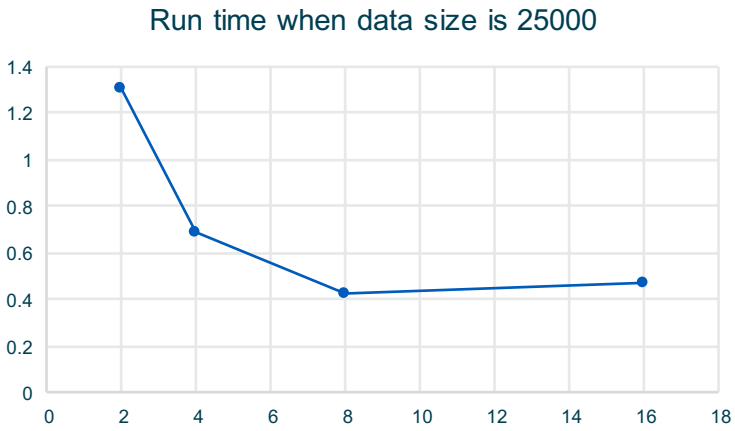
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU
31088	asifimra	20	0	32544	16432	668	R	200.0

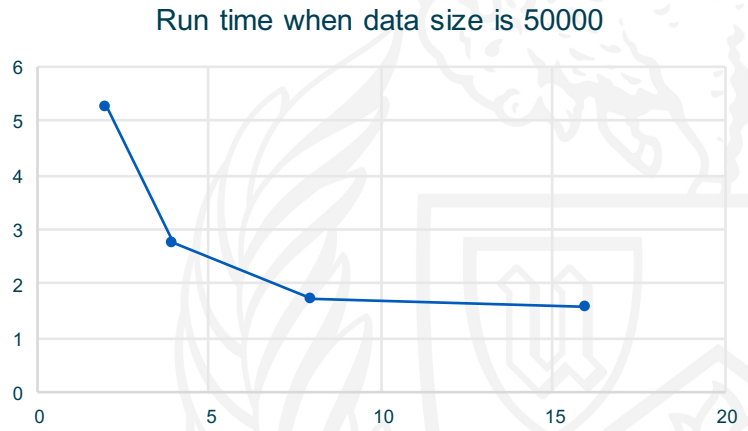
Obtained results



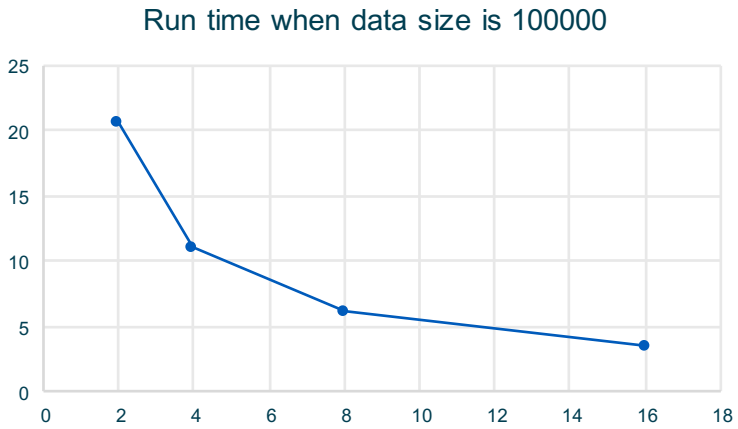


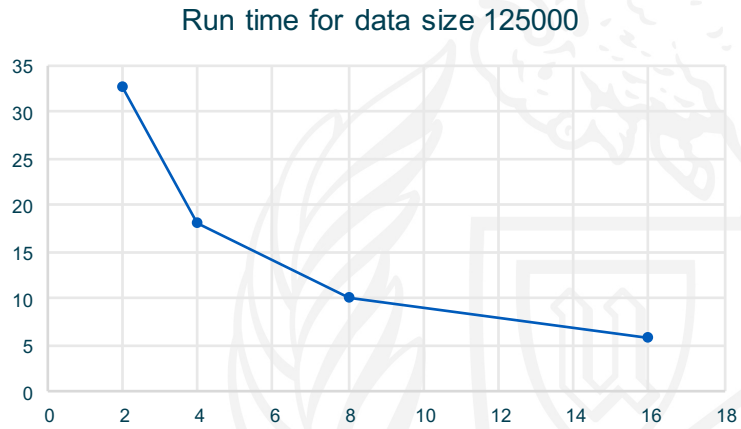
Obtained results [cont]



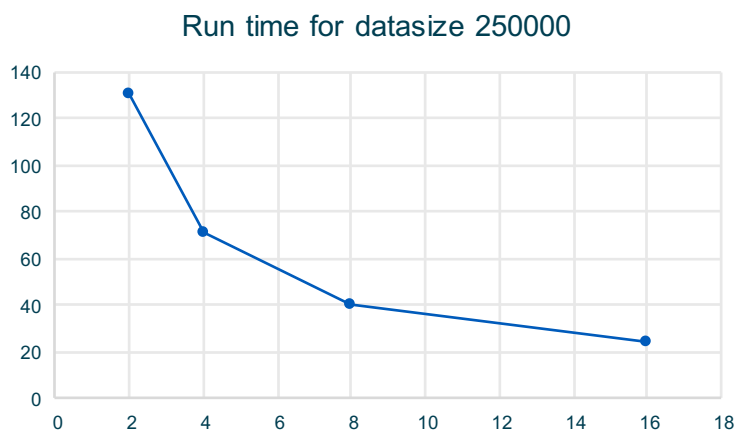


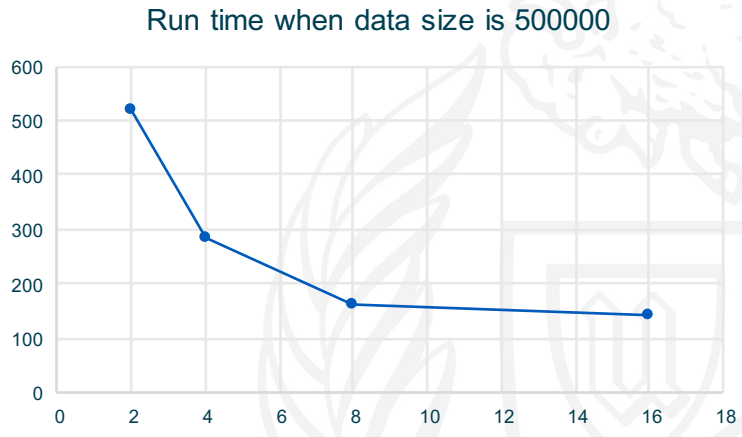
Obtained Results [cont]



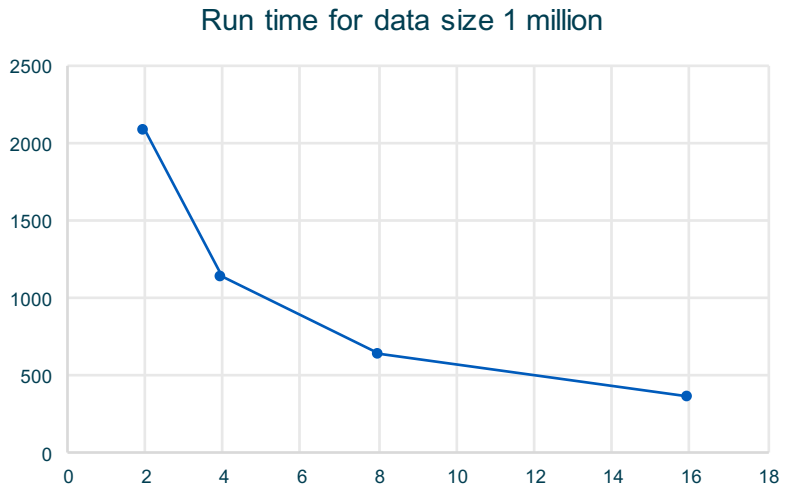


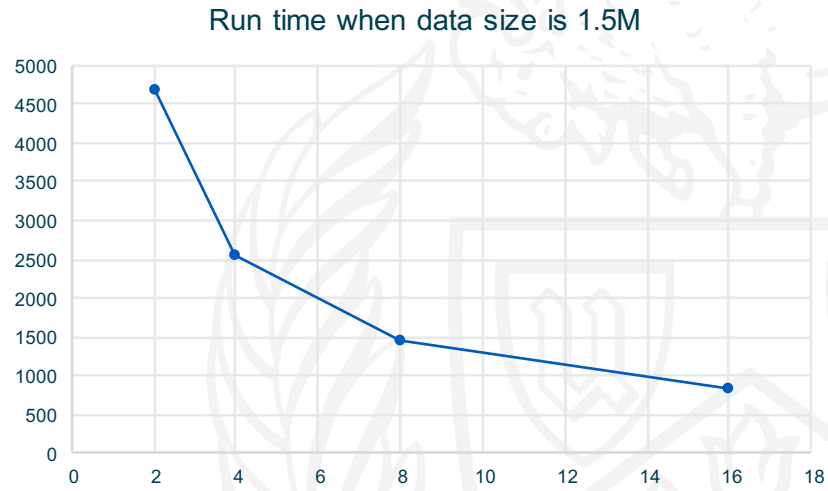
Obtained Results [cont]



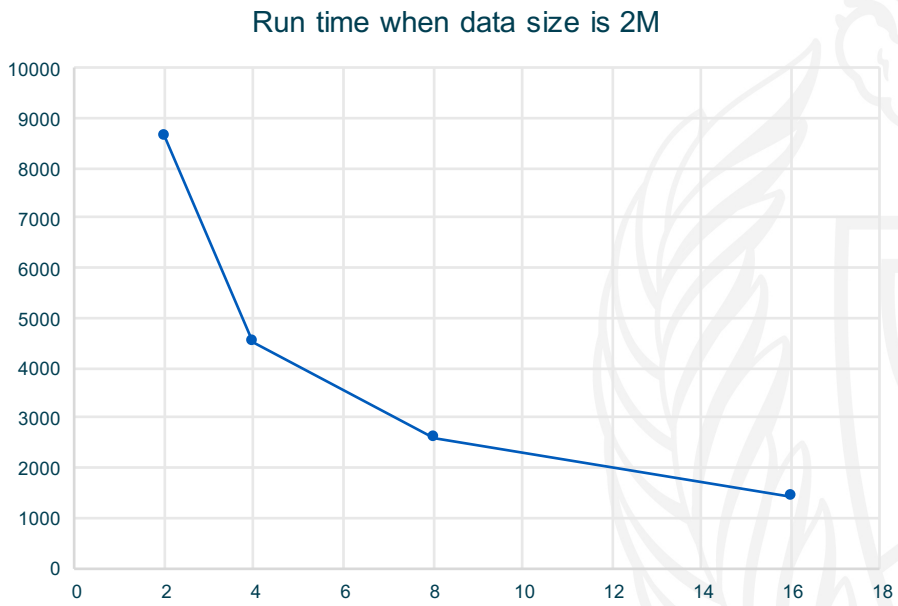


Obtained Results [cont]

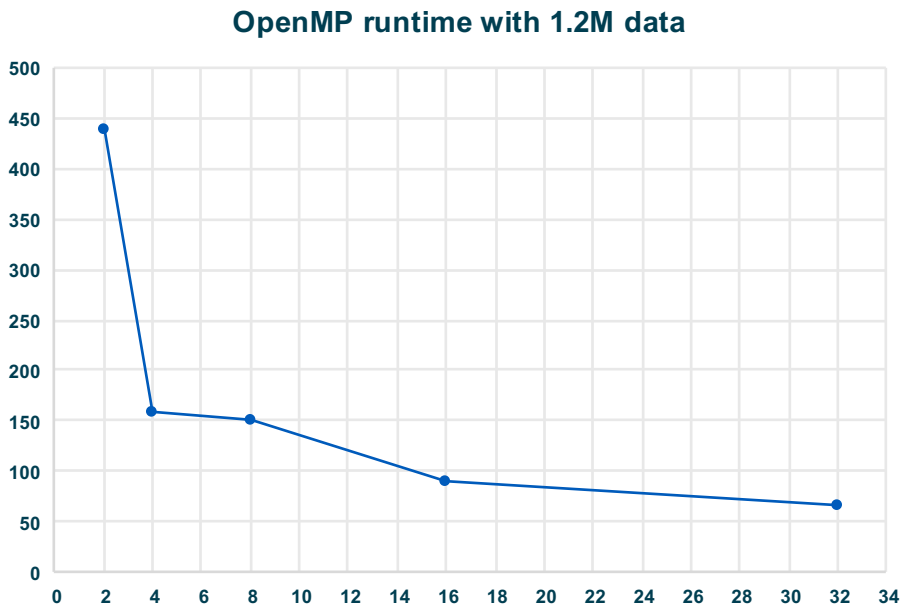




Results [cont]

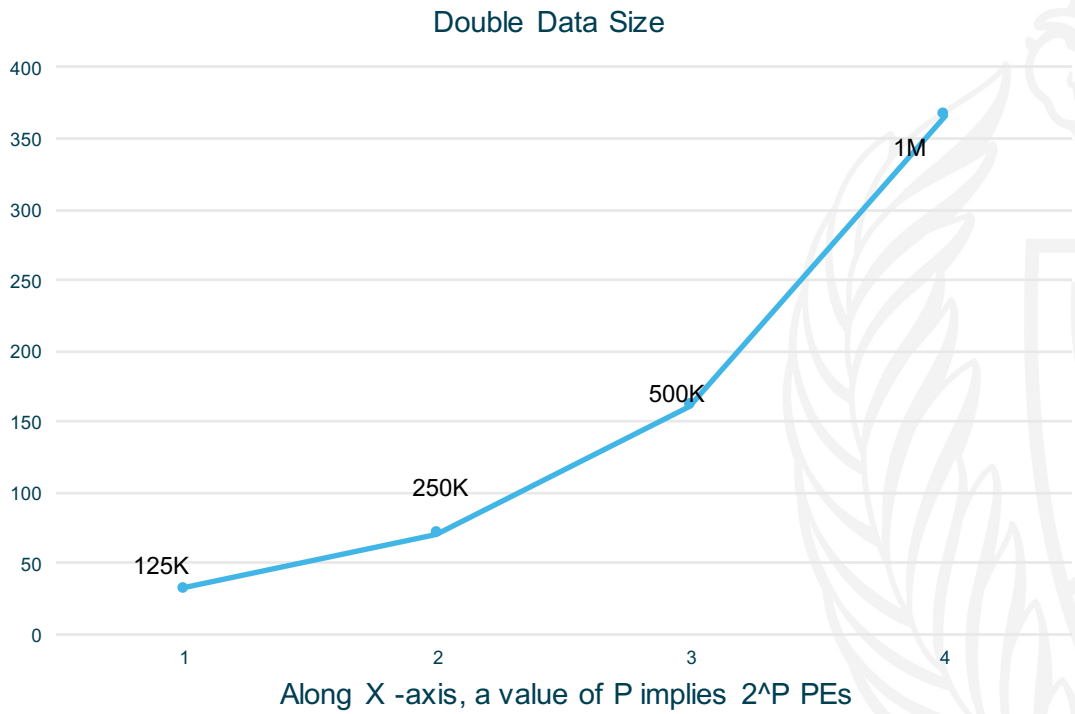


Results (cont.) with 32 threads (i.e. PEs)

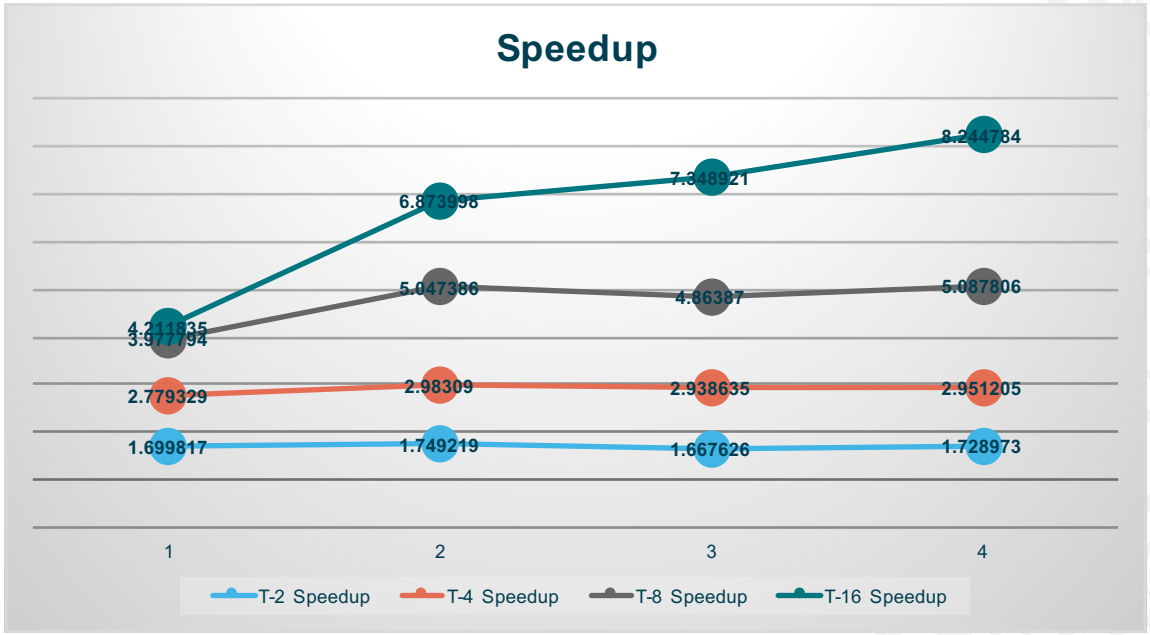


- **For 32 cores:**
  - Difficult to obtain a single server with 32 cores by salloc command in CCR.
  - Experiments were run in vortex1 front end of CCR which was equipped with 32 cores.
  - It was conducted during off-peak hours (between 3:00 AM – 4:00 AM EST)

Double data size and PEs (threads)



Speedup


Speedup
8.244784
7.348921
6.873998
5.047386
4.86387
5.087806
4.211835
3.877794
2.779329
2.98309
2.938635
2.951205
1.699817
1.749219
1.667626
1.728973
1
2
3
4
T-2 Speedup
T-4 Speedup
T-8 Speedup
T-16 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: https://cse.buffalo.edu/faculty/miller/Courses/CSE633/Asif-Imran-Spring-2018.pdf
- Parallel Computing Sorting https://cs.nyu.edu/courses/spring14/CSCI-UA.0480-003/lecture11.pdf

Thank you