CSE 570: Intro to Parallel and Distributed Programming

OpenMP - Benchmarking

Shri Harsha Adapala

UBID: 5049-5139

Hardware Specifications:

- Experiment run on UB's Vortex CCR HPC Center.
- Operating System: Ubuntu 20.04
- C++ Compiler: gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
- CPU: 8
- Memory: 32GB
- Host Name: cpn-m25

Matrix Vector Multiplication

$$T_1 = O(mn)$$
$$T_p = O\left(m\frac{n}{p}\right)$$

1. Data and its analysis assessing the scalability of the system:

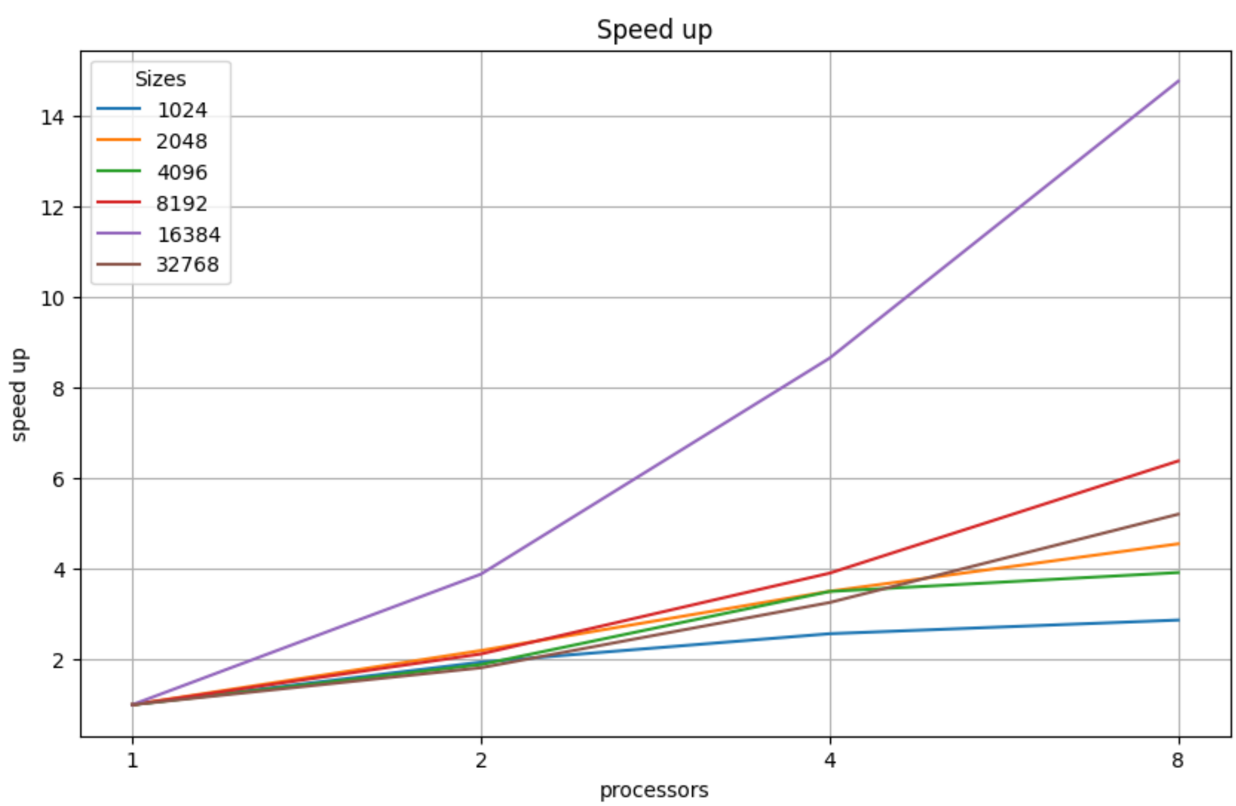
Size = m = n

Execution Time (Seconds):

Size	1024	2048	4096	8192	16384	32768
1	0.003691	0.01582	0.054229	0.199714	1.37366	4.68156
2	0.001902	0.0072	0.02875	0.09417	0.353672	2.57834
4	0.001437	0.004513	0.015488	0.051133	0.158715	1.43721
8	0.001286	0.003474	0.013835	0.031284	0.092996	0.898901

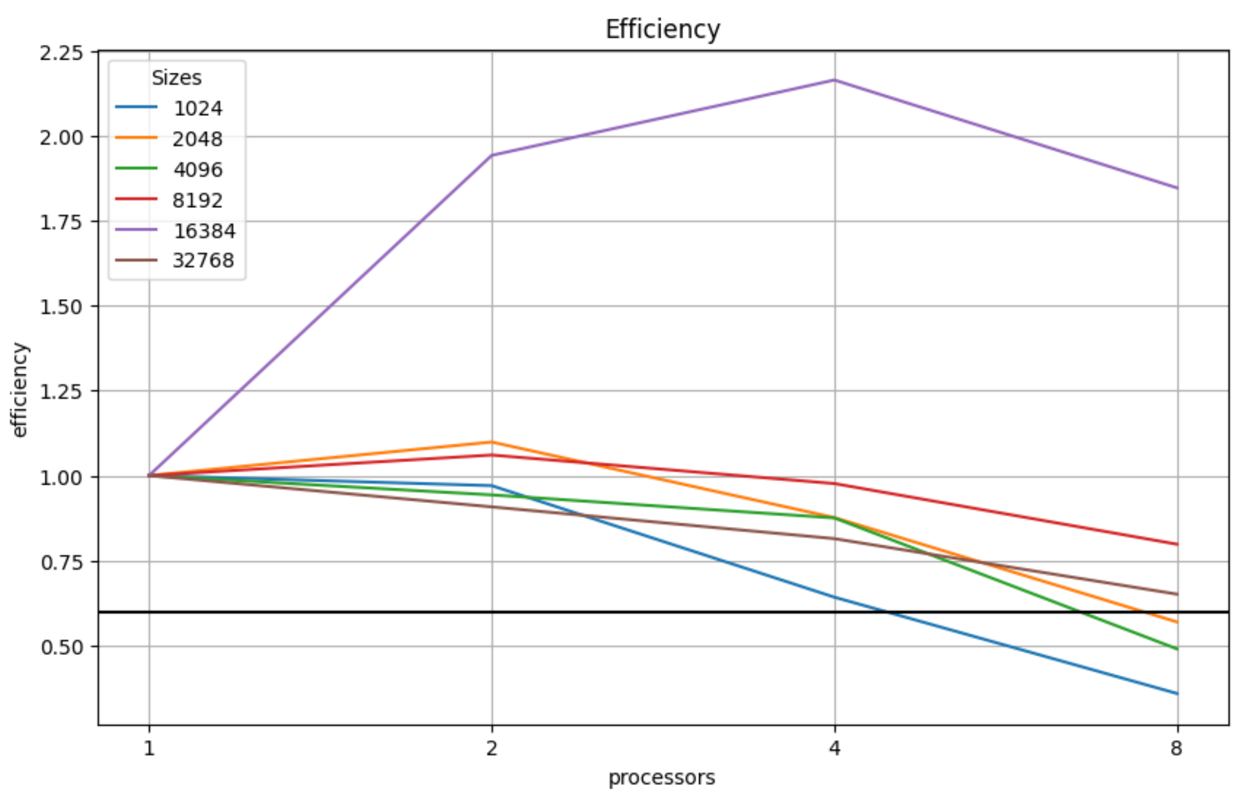
Speed Up:

Size	1024	2048	4096	8192	16384	32768
1	1	1	1	1	1	1
2	1.94	2.197	1.88	2.12	3.88	1.81
4	2.56	3.504	3.50	3.9	8.65	3.25
8	2.8	4.5	3.91	6.38	14.77	5.20



Efficiency:

Size	1024	2048	4096	8192	16384	32768
1	1	1	1	1	1	1
2	0.97	1.09	0.94	1.06	1.94	0.90
4	0.64	0.87	0.87	0.97	2.16	0.814
8	0.35	0.56	0.48	0.79	1.84	0.65



Observations:

- We can see almost all speedups super-linear up to 4 processors. After that Strong-Scalability of the took a dent for the small sizes like 1024, 2048 and 4096.
- Algorithm is highly Weekly-Scalable.
- The algorithm is highly efficient for large problem sizes, but their efficiency took a dent for small problem sizes (1024, 2048, 4096) after 4 processors.

Matrix-Matrix Multiplication

$$T_1 = O(mnk)$$
$$T_p = O\left(mk\frac{n}{p}\right)$$

Size: m = k = n

Execution Time (Seconds):

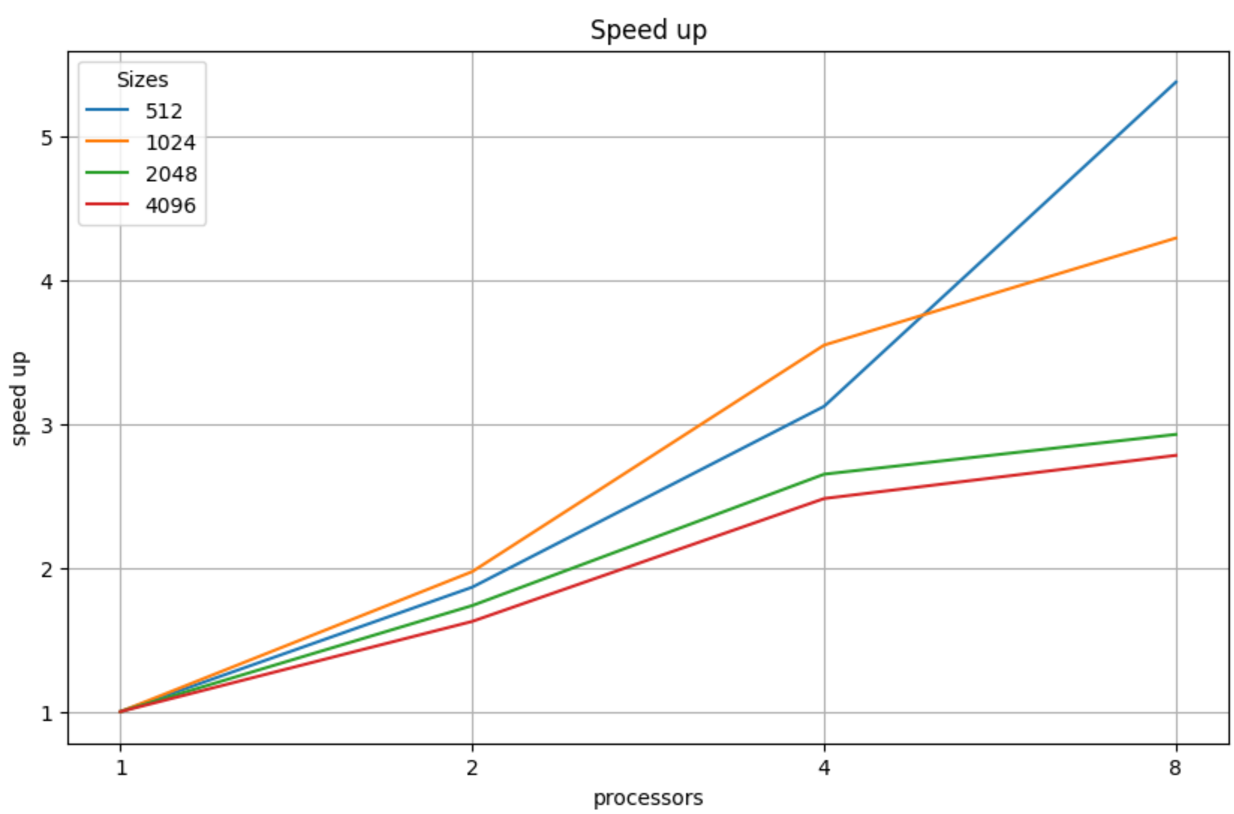
Size	512	1024	2048	4096
1	0.25	2.1058	50.803	413.995
2	0.133977	1.06664	29.2306	254.312
4	0.08002	0.593165	19.1574	166.765
8	0.046464	0.490279	17.3453	148.753

Million FLOPS:

Size	512	1024	2048	4096
1	1064.21	1019.8	338.166	331.983
2	2003.59	2013.31	587.735	540.434
4	3354.6	3620.38	896.773	824.148
8	5777.28	4380.13	990.464	923.939

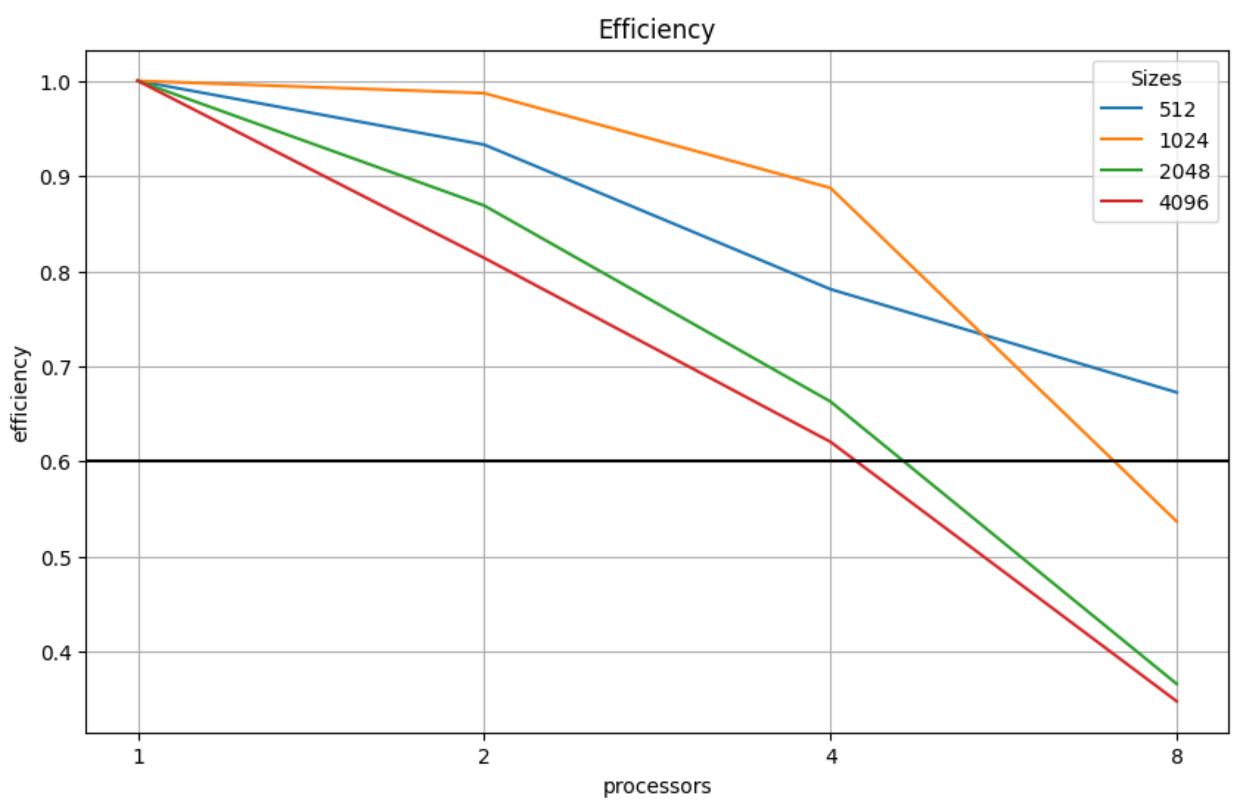
Speed Up:

Size	512	1024	2048	4096
1	1	1	1	1
2	1.86	1.97	1.73	1.62
4	3.12	3.55	2.65	2.48
8	5.38	4.29	2.92	2.78



Efficiency:

Size	512	1024	2048	4096
1	1	1	1	1
2	0.93	0.98	0.86	0.81
4	0.78	0.88	0.66	0.62
8	0.67	0.53	0.36	0.34



Observations:

- We can see almost all speedups super-linear at 4 processors for small problem sized. Algorithm is Strongly-Scalable only for small problem sizes.
- Algorithm is not Weekly-Scalable.
- The algorithm is highly efficient for small problem sizes and less processors.

2D-Convolution

$$T_1 = O(mnk^2)$$
$$T_p = O\left(\frac{mn}{p}k^2\right)$$

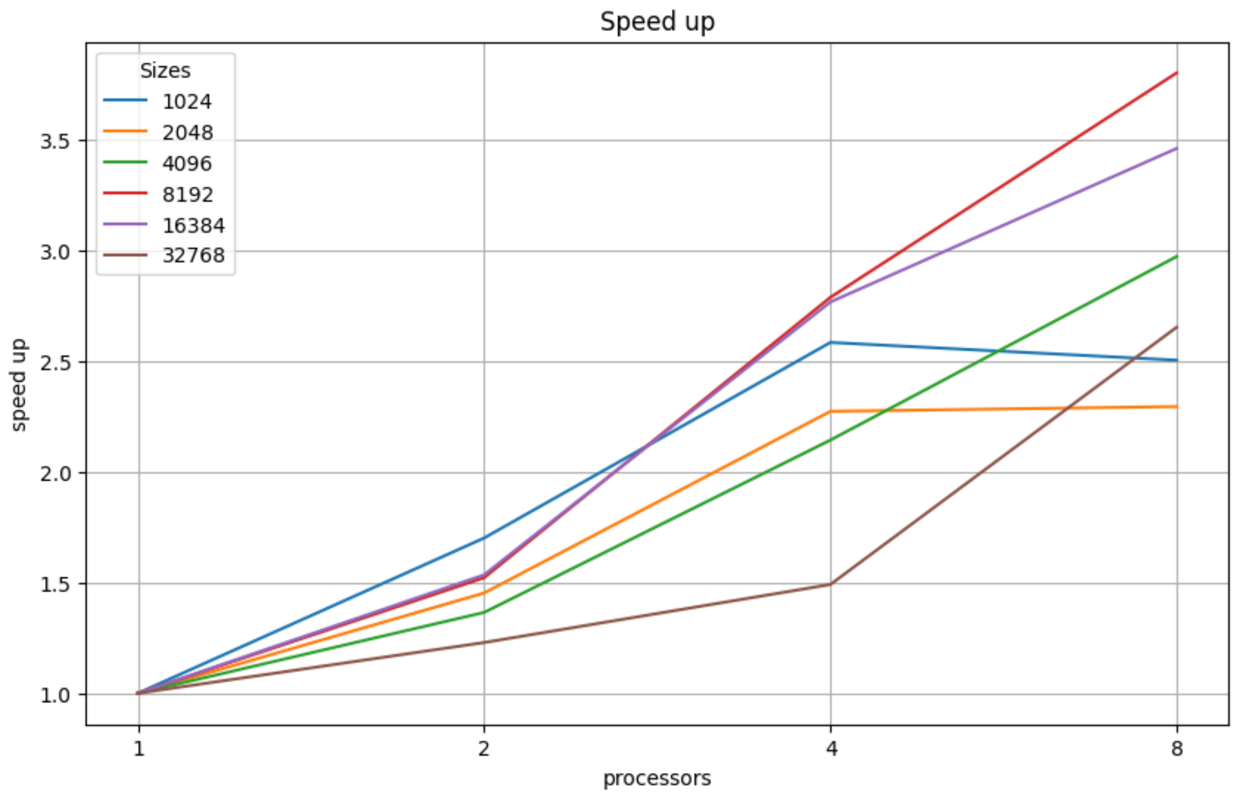
Size: m = n; kernel_size=3 ;

Execution Time (Seconds):

Size	1024	2048	4096	8192	16384	32768
1	0.009302	0.027533	0.072712	0.25031	0.981586	6.87654
2	0.005471	0.018959	0.053285	0.164545	0.639869	5.5961
4	0.003602	0.01212	0.033945	0.08985	0.355011	4.61309
8	0.003717	0.012006	0.024481	0.065897	0.283879	2.59363

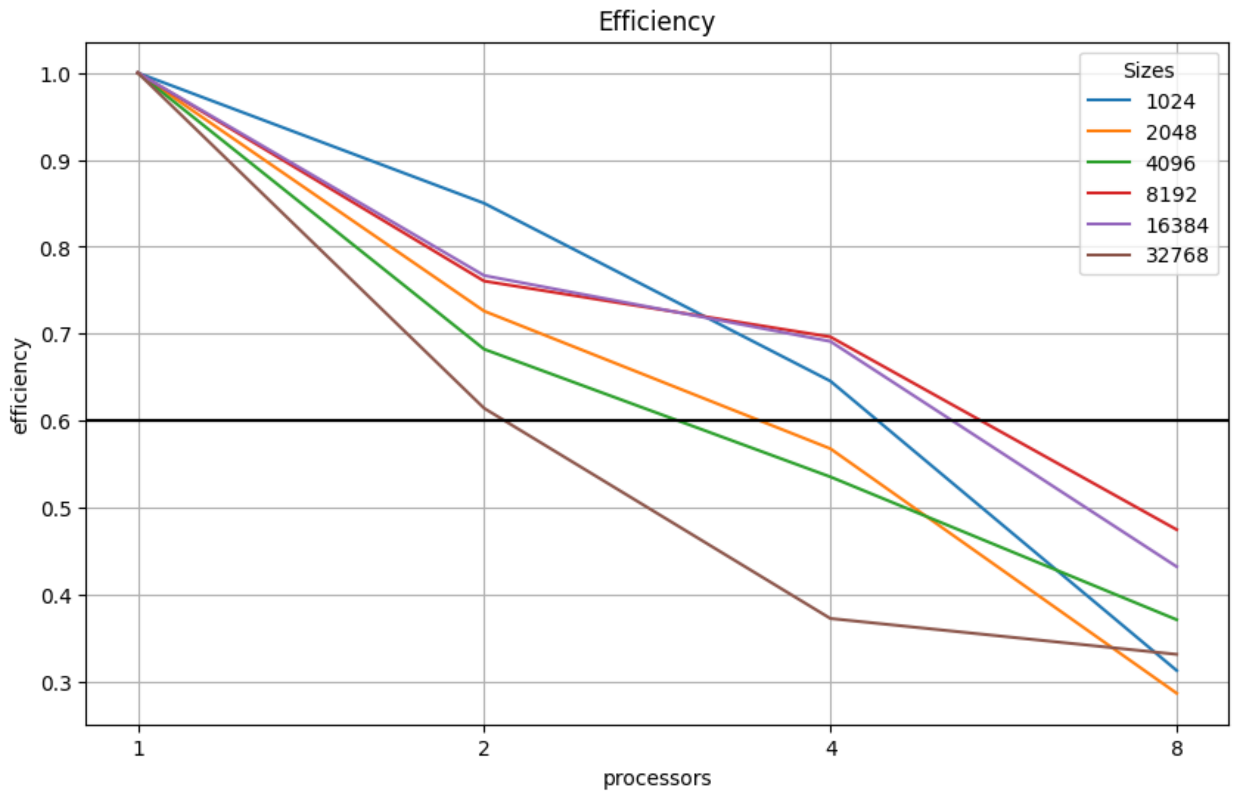
Speed Up:

Size	1024	2048	4096	8192	16384	32768
1	1	1	1	1	1	1
2	1.7	1.45	1.36	1.52	1.53	1.22
4	2.58	2.27	2.14	2.78	2.76	1.49
8	2.5	2.29	2.9	3.7	3.4	2.6



Efficiency:

Size	1024	2048	4096	8192	16384	32768
1	1	1	1	1	1	1
2	0.85	0.76	0.68	0.76	0.76	0.61
4	0.64	0.56	0.53	0.69	0.61	0.37
8	0.31	0.28	0.37	0.47	0.43	0.33



Observations:

- We can see almost all speedups sub-linear.
- Algorithm is not Strongly-Scalable.
- Algorithm is not Weekly-Scalable.
- The algorithm is highly efficient for small problem sizes and less processors.