CS 553
Cloud Computing
Assignment 1
Performance Evaluation Report

Vivek Agarwal (A20384303)
Shraddha Mantri (A20378714)

CPU BENCHMARKING:

We've evaluated the processor speed in terms of floating point operations per second (Giga FLOPS) and integer operations per second (Giga IOPS) with respect to 1, 2, 4, 8 number of threads. Results mentioned in the graph below are averages of GFLOPS and GIOPS for their respective threads.

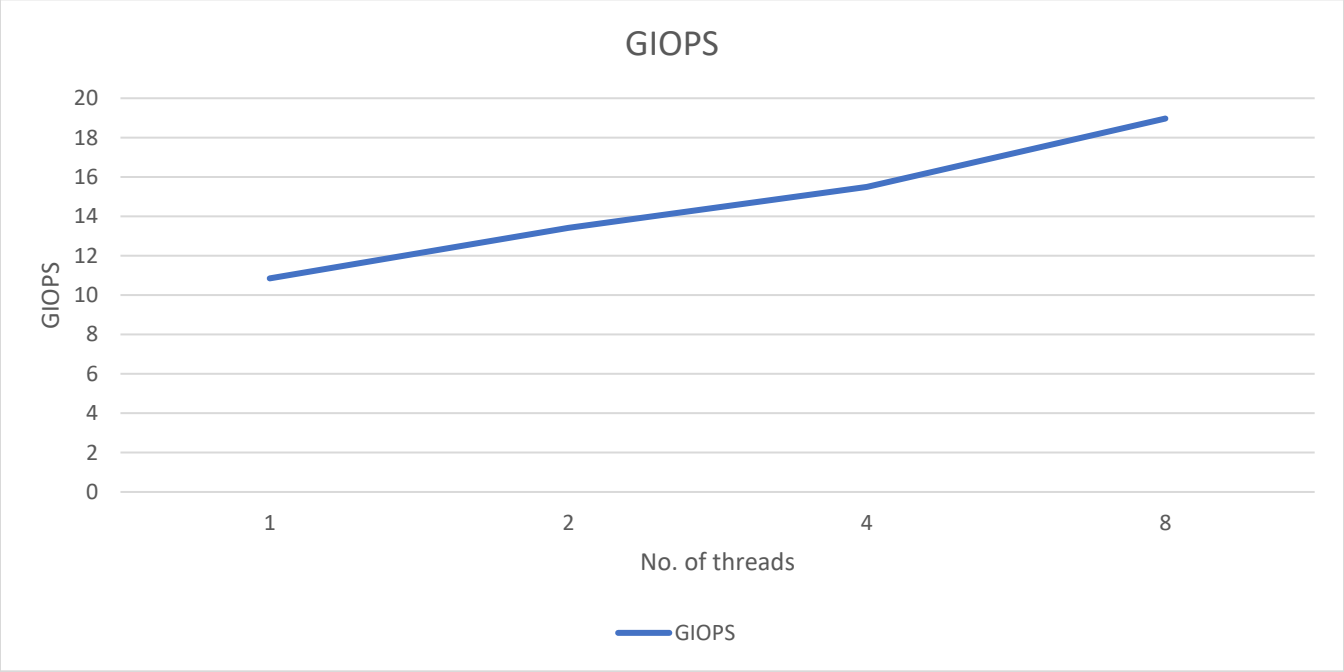
Instance: m1.medium

Operation System: CC-CentOS7-1610.0

No. of cores: 2

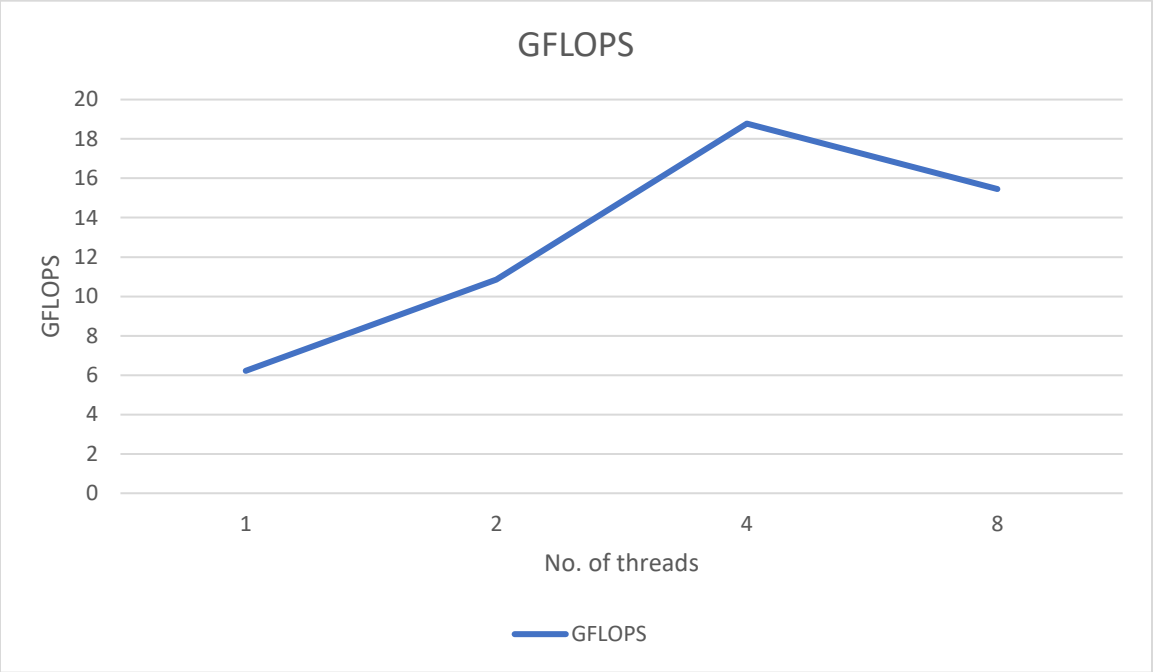
Giga IOPS:

No. of Threads	**GIOPS**
1	10.85
2	13.41
4	15.5
8	18.97



GFLOPS

No. of Threads	GFLOPS
1	6.22
2	10.86
4	18.78
8	15.46



Calculating theoretical peak performance

m1.medium

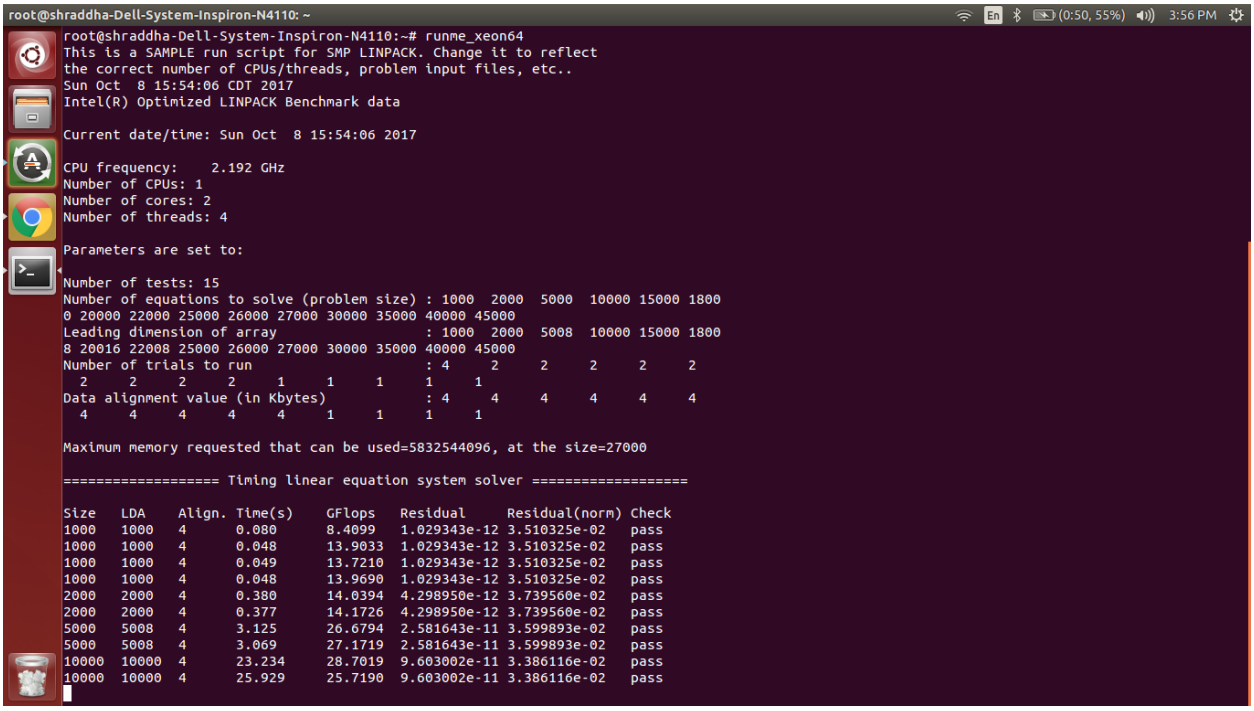
a. CPU speed:2.3GHz
b. Number of cores:2
c. Instruction per cycle: 12

Theoretical peak performance = (CPU speed*no of cores*Instructions/cycle) = 55.2

Efficiency = (FLOPS for 1 thread/Theoretical peak performance)*100 = = 11.12%

Our performance is 11.12% of the theoretical performance and linpack performance is 15.4% of the theoretical performance.

Linpack benchmark:



MEMORY BENCHMARKING:

We've used strong scaling experiments for memory benchmark. We've implemented read + write operations (i.e. memcpy), sequential write access (i.e. memset) and random write access with varying block sizes (8B, 8kB, 8MB, 80MB) and varying number of threads (1 thread, 2 threads, 4 threads and 8 threads). We have performed read + write and write operations for both sequential and random memory access within 1 GB of memory. The results obtained are plotted in following tables and graphs:

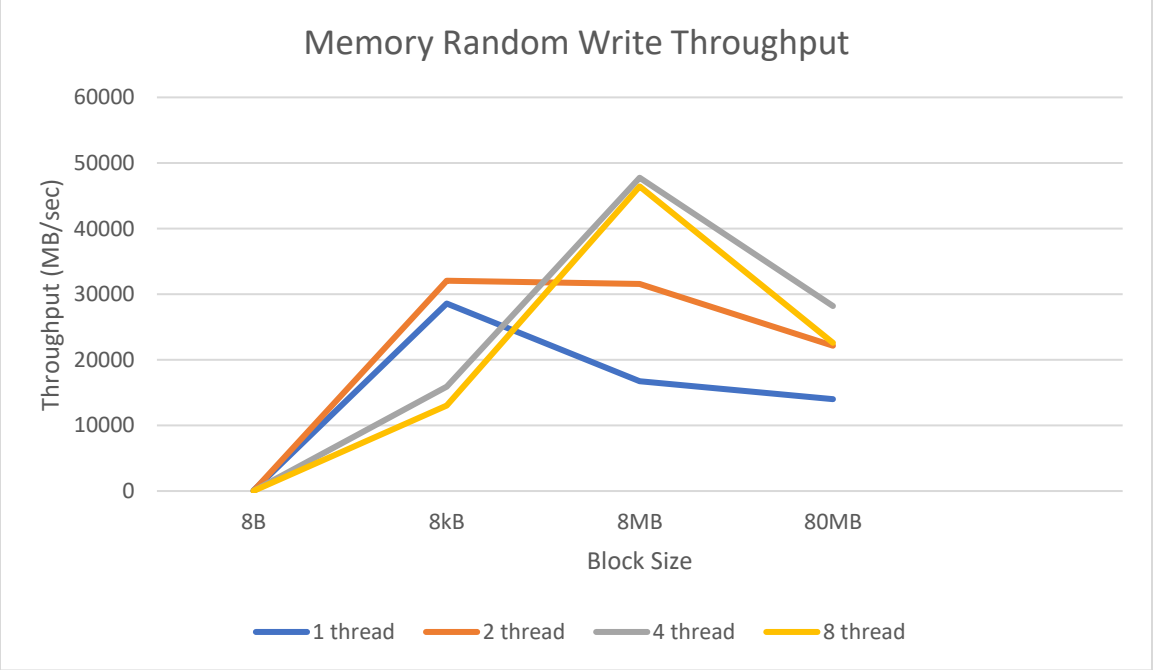
Instance: baremetal

Operation System: CC-CentOS7-1610.0

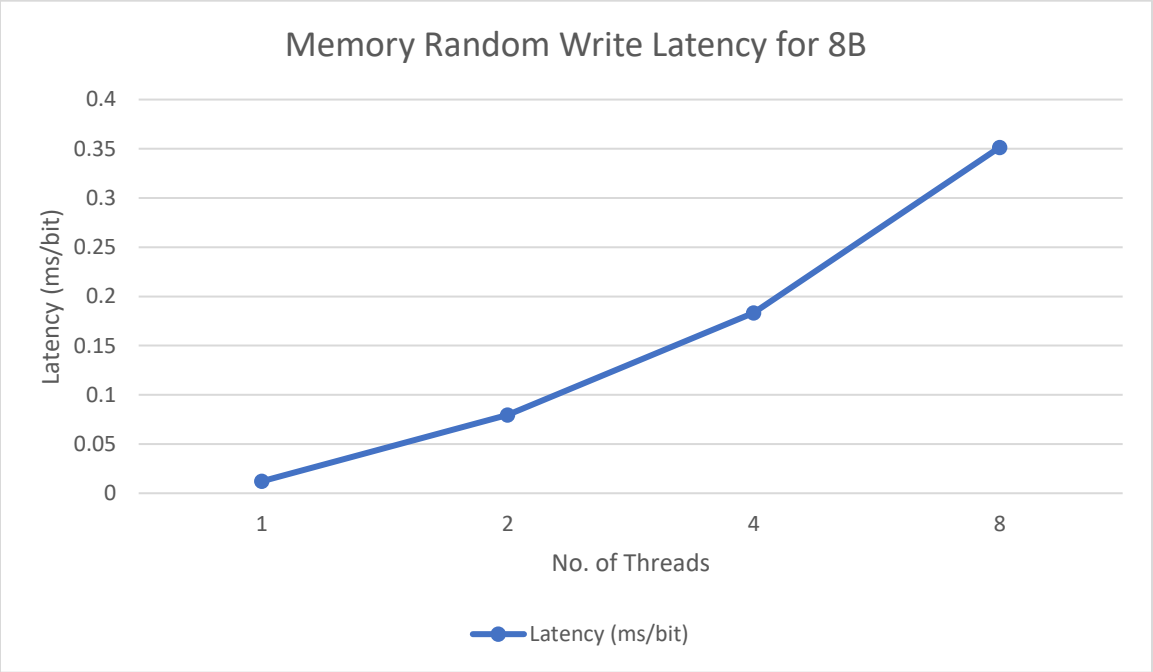
No. of cores: 24

Memory Random Write Throughput:

	8B	**8kB**	**8MB**	**80MB**
1 thread	78.7	28577.8	16711.8	14027.2
2 thread	23.95	32052.08	31555.26	22142.45
4 thread	20.8	15919.9	47750.05	28174.43
8 thread	21.7	13047.91	46425.17	22584.91

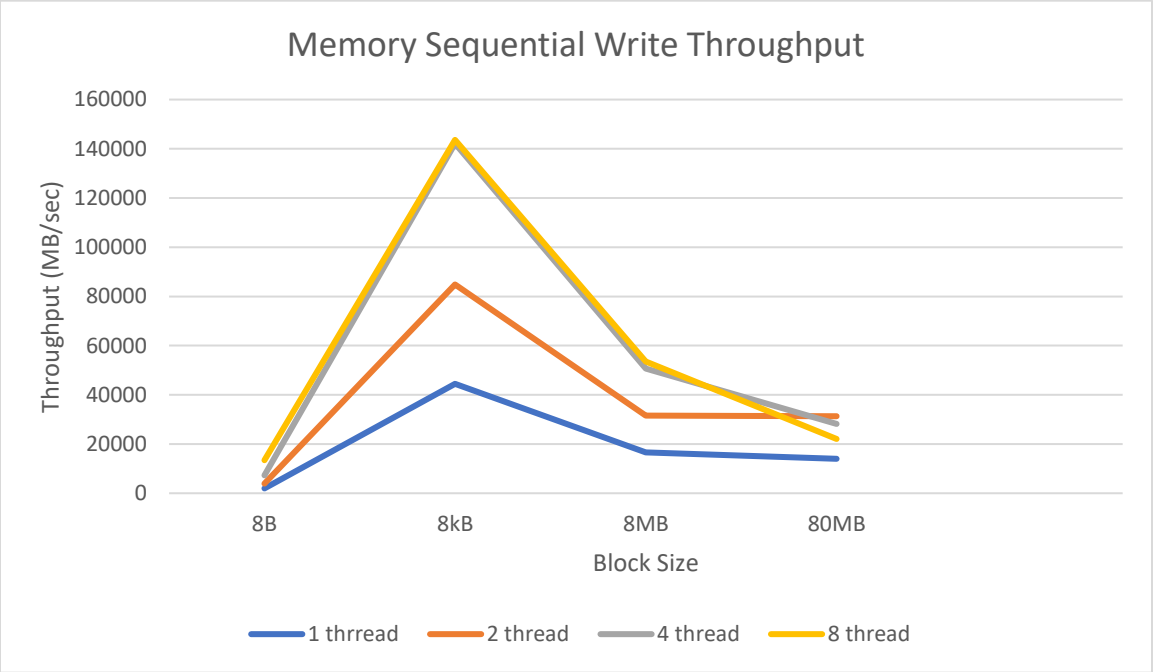


No. of threads	Latency on 8B
1	0.012118
2	0.079620
4	0.183385
8	0.351501

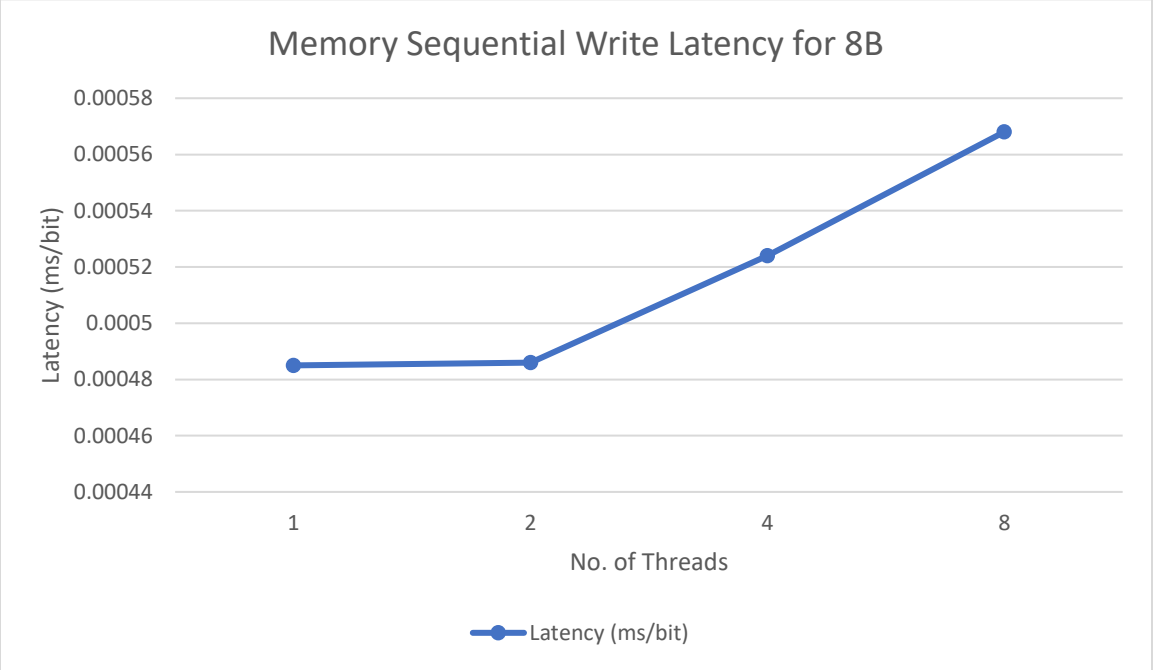


Memory Sequential Write Throughput:

	8B	8kB	8MB	80MB
1 thread	1967.7	44471.46	16649.32	14032.2
2 thread	3921.95	84845.46	31618.6	31597.13
4 thread	7286.65	142143.25	50728.22	28210.144
8 thread	13434.96	143618.5	53483.75781	22070.86

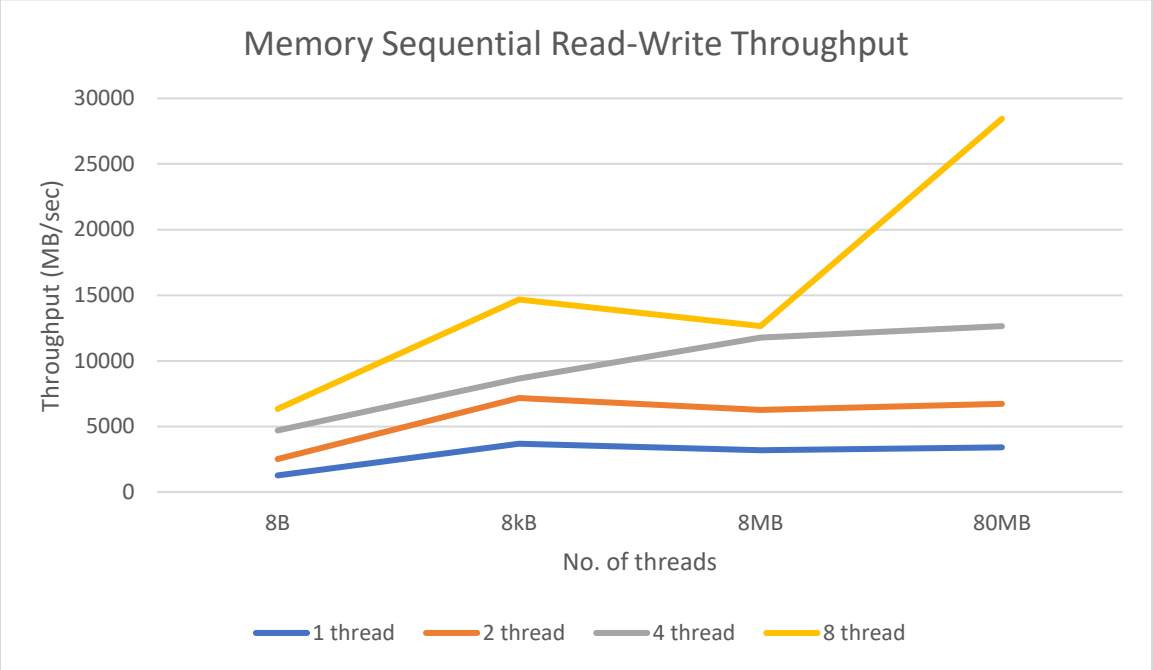


No. of Threads	Latency on 8B
1	0.000485
2	0.000486
4	0.000524
8	0.000568

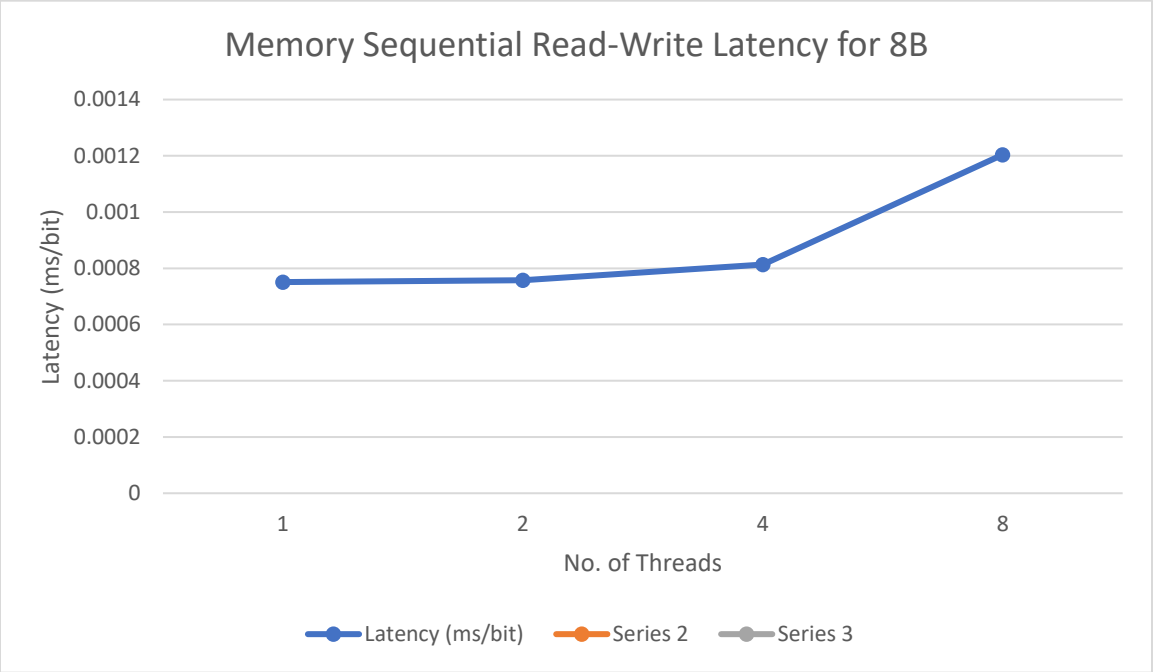


Memory Sequential Read-Write Throughput:

	8B	8kB	8MB	80MB
1 thread	1269.08	3682.1	3193.99	3411.94
2 thread	2516.47	7165.4	6252.4	6721.8
4 thread	4694.03	8647.55	11777.15	12644.94
8 thread	6334.90	14682.05	12648.37	28448.4

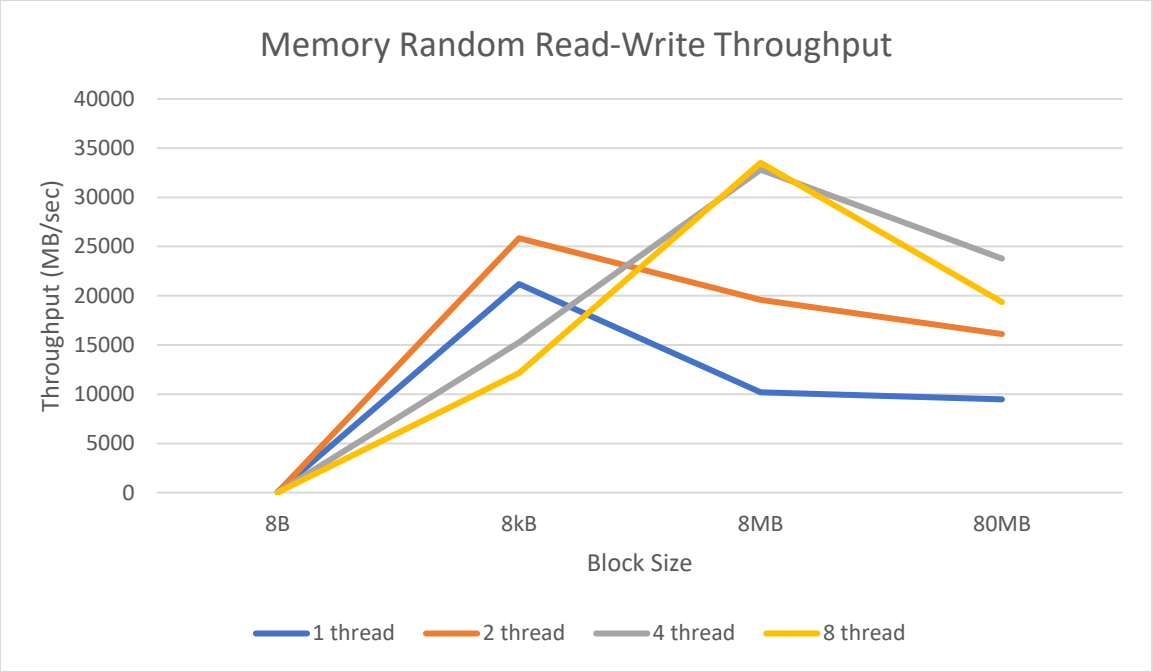


No. of threads	Latency
1	0.000751
2	0.000758
4	0.000813
8	0.001204

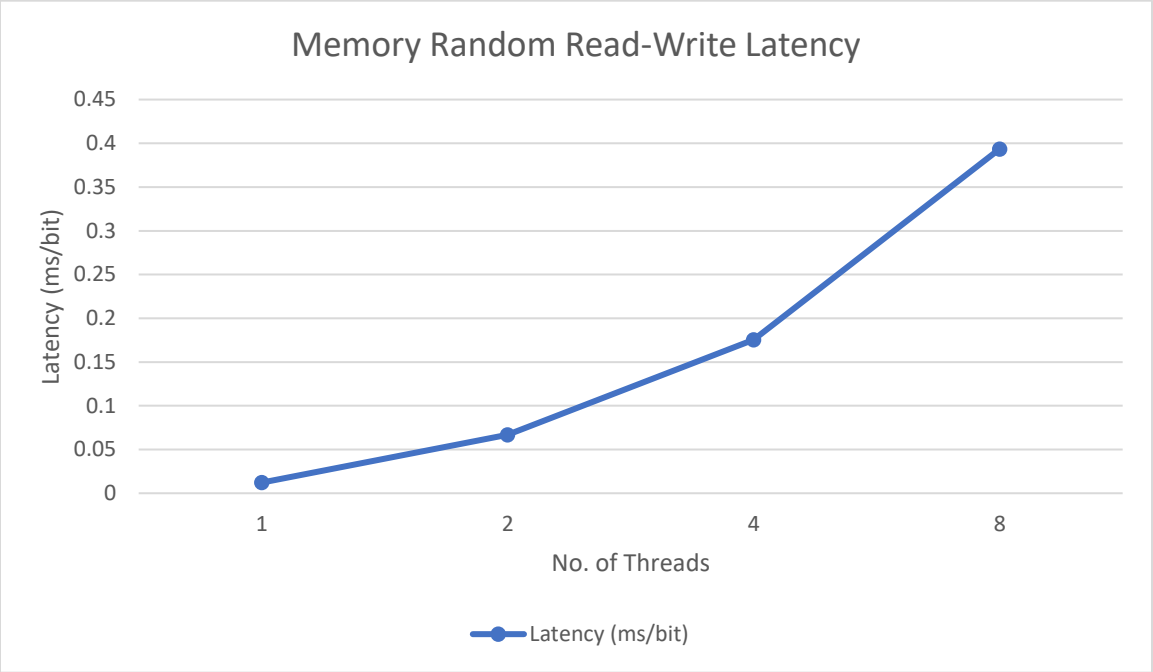


Memory Random Read-Write Throughput:

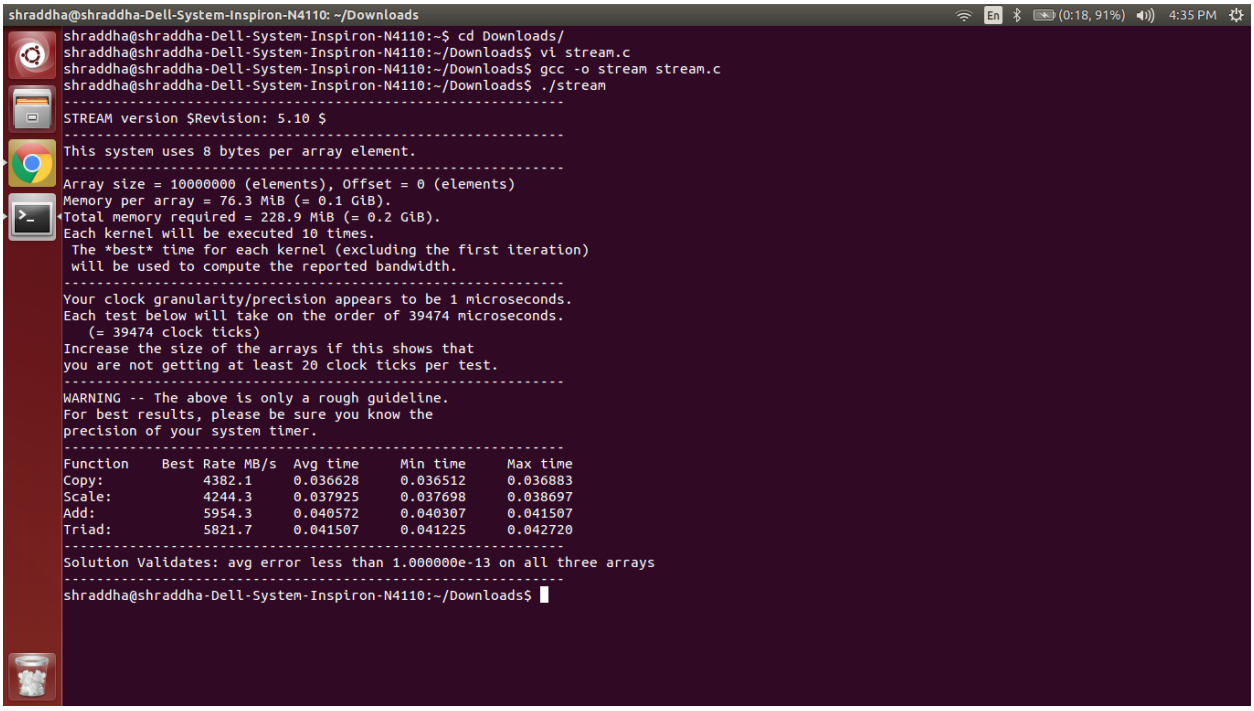
	8B	8kB	8MB	80MB
1 thread	78.47	21191.17	10202.55	9469.20
2 thread	28.57	25848.8	19598.46	16117.10
4 thread	21.73	15263.53	32802.64	23782.42
8 thread	19.38	12148.97	33513.33	19345.94



No. of Threads	Latency
1	0.012153
2	0.066752
4	0.175486
8	0.393564



Stream Benchmark:



DISK BENCHMARKING:

We've used strong scaling experiments for memory benchmark. We've implemented read + write operations (i.e. memcpy), sequential write access (i.e. memset) and random write access with varying block sizes (8B, 8kB, 8MB, 80MB) and varying number of threads (1 thread, 2 threads, 4 threads and 8 threads). We have performed read + write and write operations for both sequential and random memory access within 1 GB of memory.
In case of random test cases, we have used fseek to adjust the index location in the file. We are passing the start index as a parameter to the thread function, calculating the end index based on the block size and doing the operations. In case we have 1 GB of file and there are 4 threads to operate, we'll be deviding the start index and the end index (calculation) in the following way: 0-249, 250-499, 500-749, 750-999.
The results obtained are plotted in following tables and graphs.

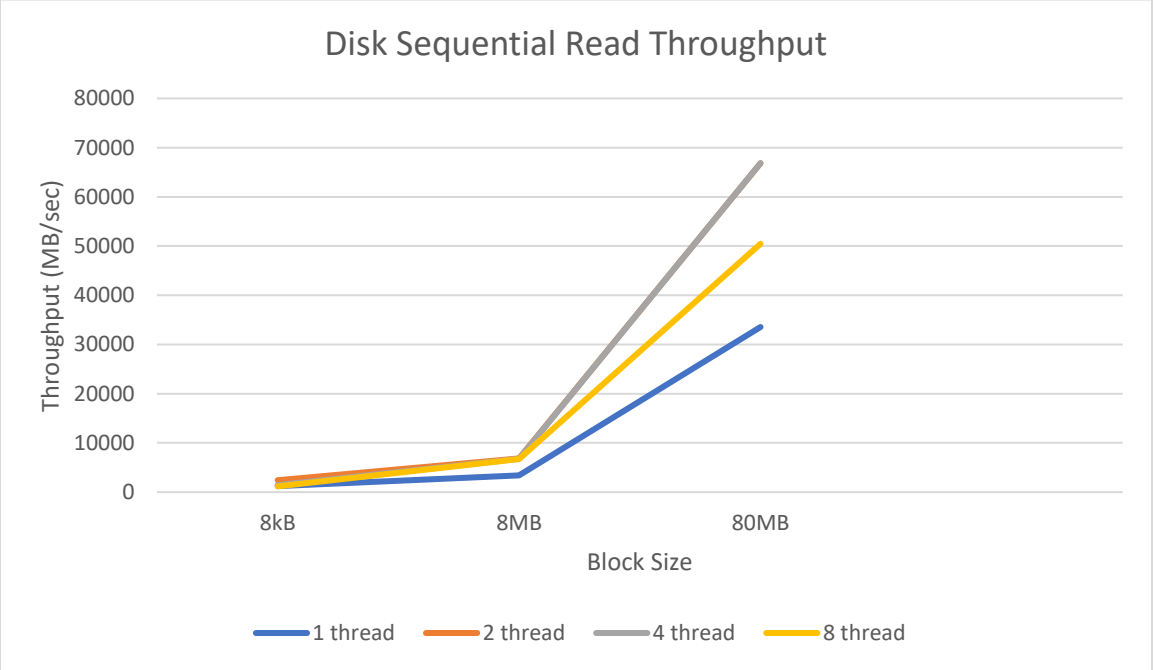
Instance: baremetal

Operation System: CC-CentOS7-1610.0

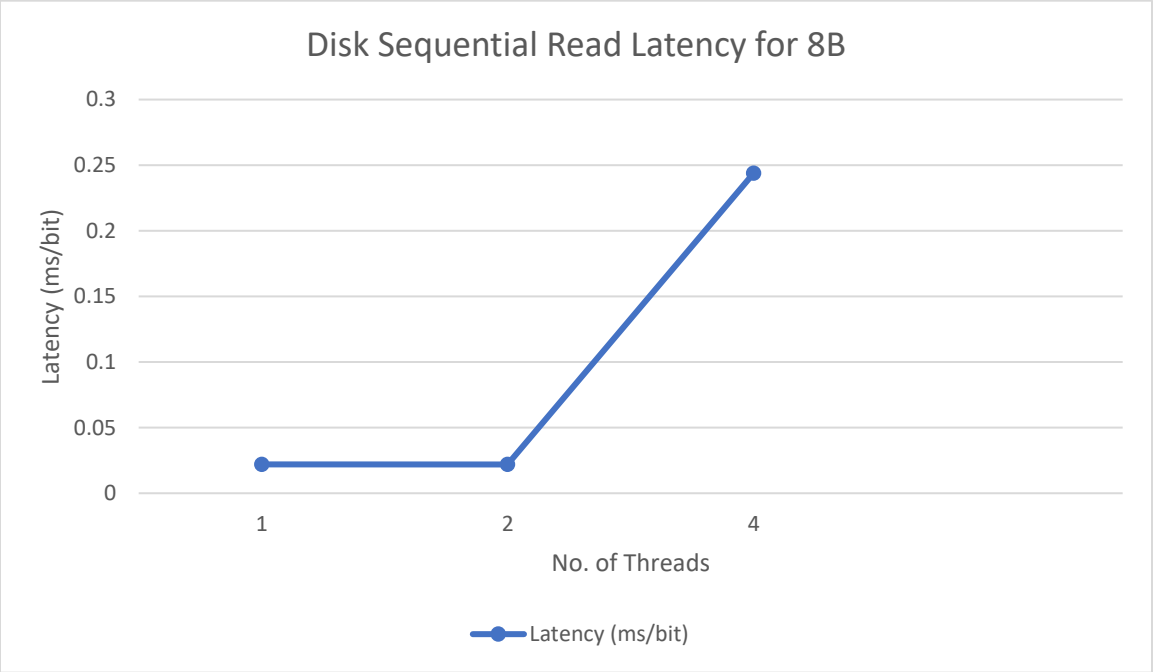
No. of cores: 24

Disk Sequential Read Throughput:

	8B	**8kB**	**8MB**	**80MB**
1 thread	43.40	1219.21	3379.05	33547.37
2 thread	87.01	2414.263	6821.15	66836.36
4 thread	15.64	1409.88	6799.87	66845.094
8 thread		1165.22	6727.21	50467.71



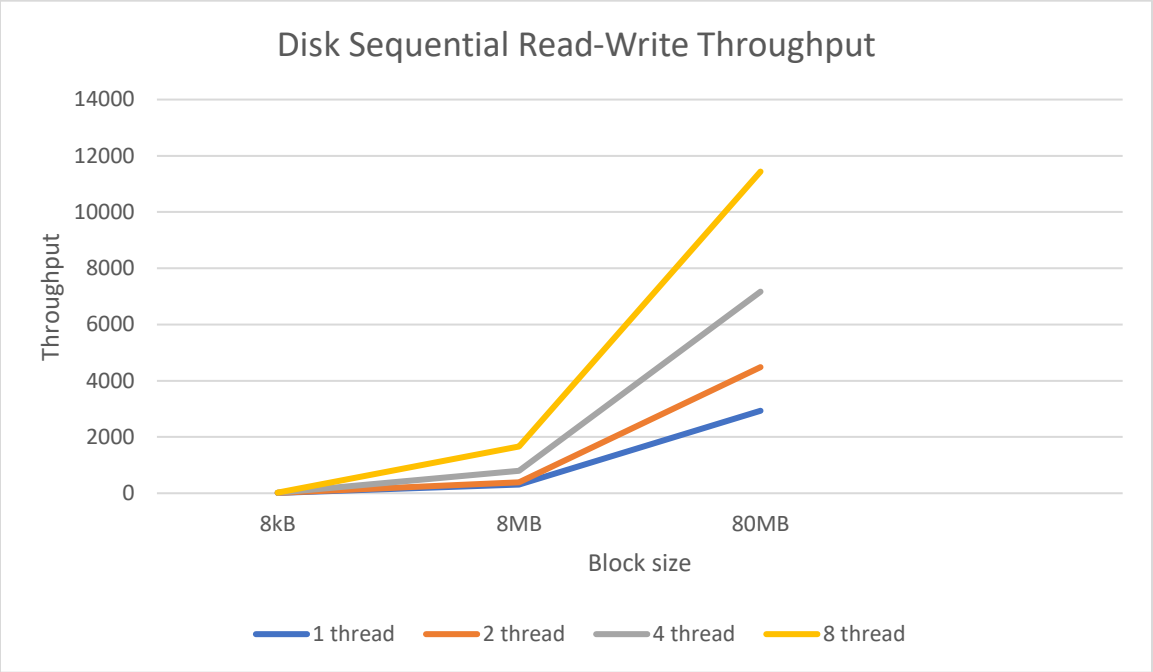
No. of Threads	Latency
1	0.021973
2	0.021921
4	0.243882



Disk sequential read-write throughput:

1GB file:

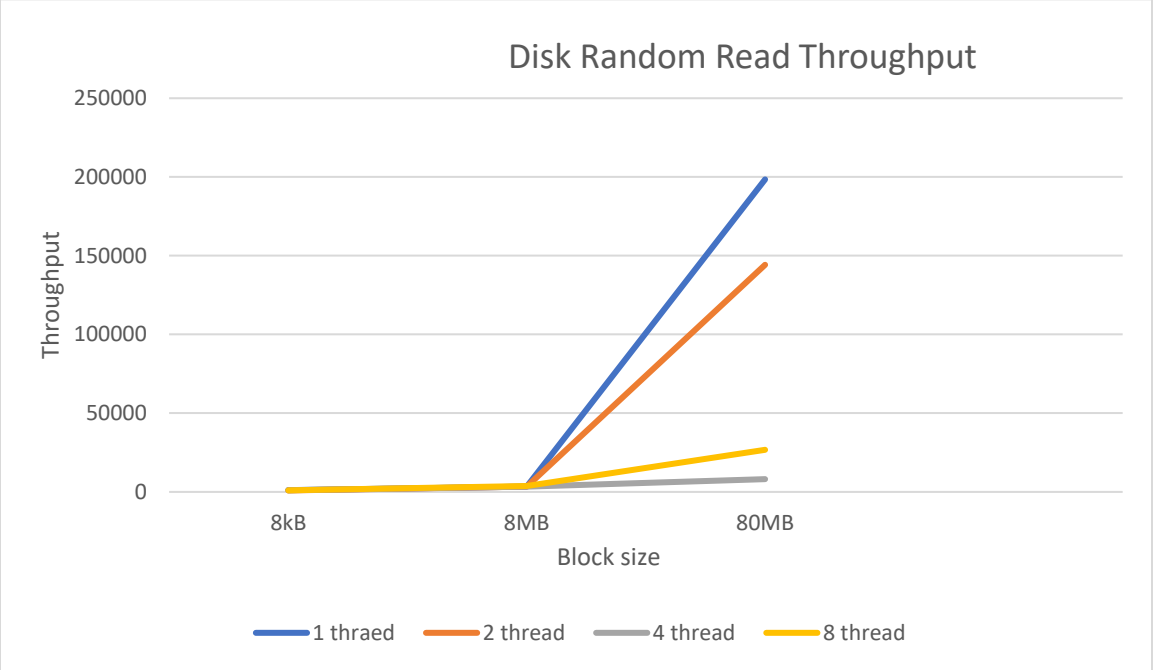
	1 thread	2 thread	4 thread	8 thread
8kB	12.67	14.56	18.34	23.14
8MB	305.19	390.713	802.133	1662.261
80MB	2933.19	4484.305	7169.713	11439.679



Disk random read throughput:

1GB file:

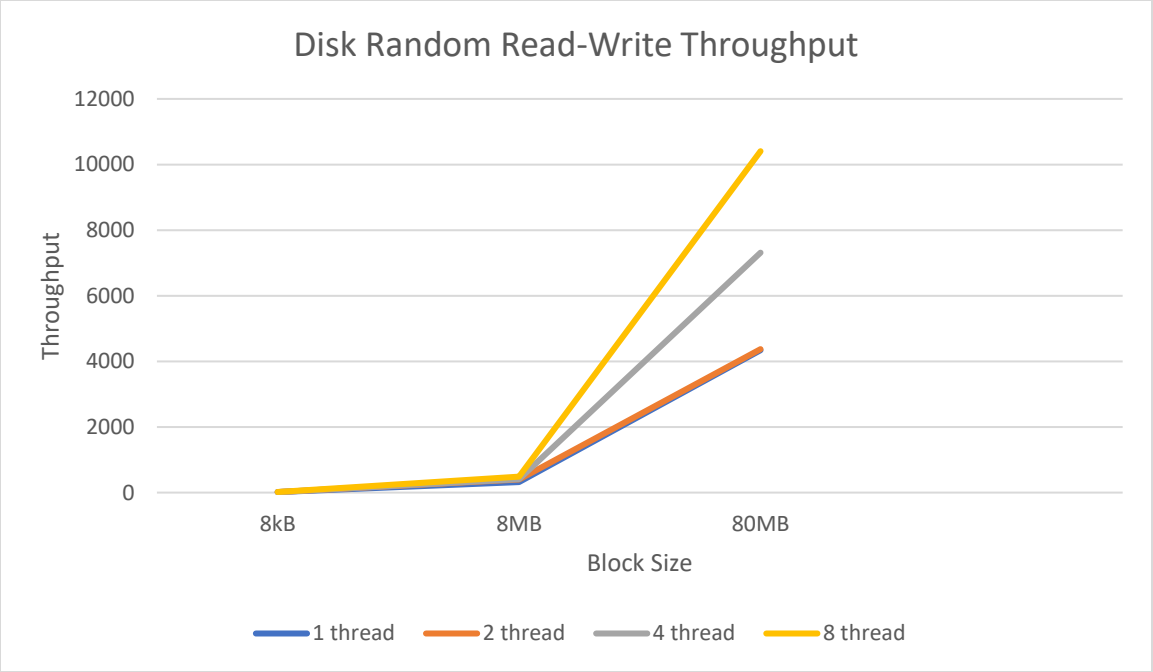
	1 thread	2 thread	4 thread	8 thread
8kB	1137.698	948.41	1089.2	1001.87
8MB	3372.204	3337.505	3363.310	3843.209
80MB	198449.609	144225.359	8109.641	26739.785



Disk random read-write throughput:

1GB file:

	1 thread	2 thread	4 thread	8 thread
8kB	15.3	I6.3	15.9	18.1
8MB	319.523	399.085	412.2	484.2
80MB	4339.976	4372.238	7313.554	10406.081



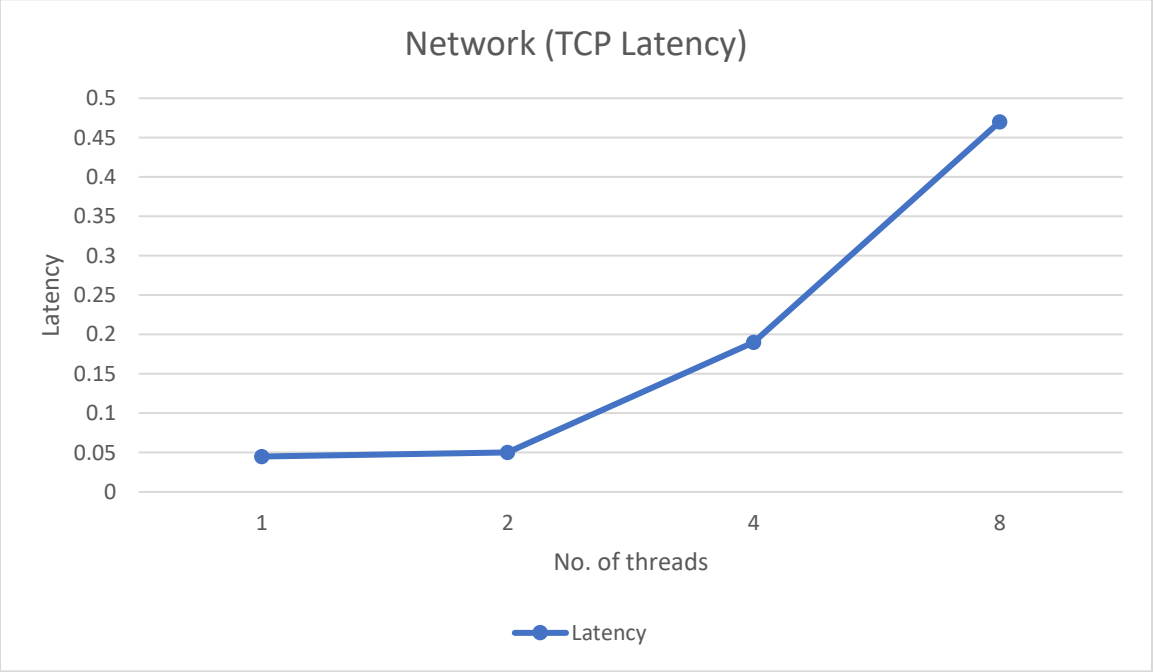
NETWORK BENCHMARKING:

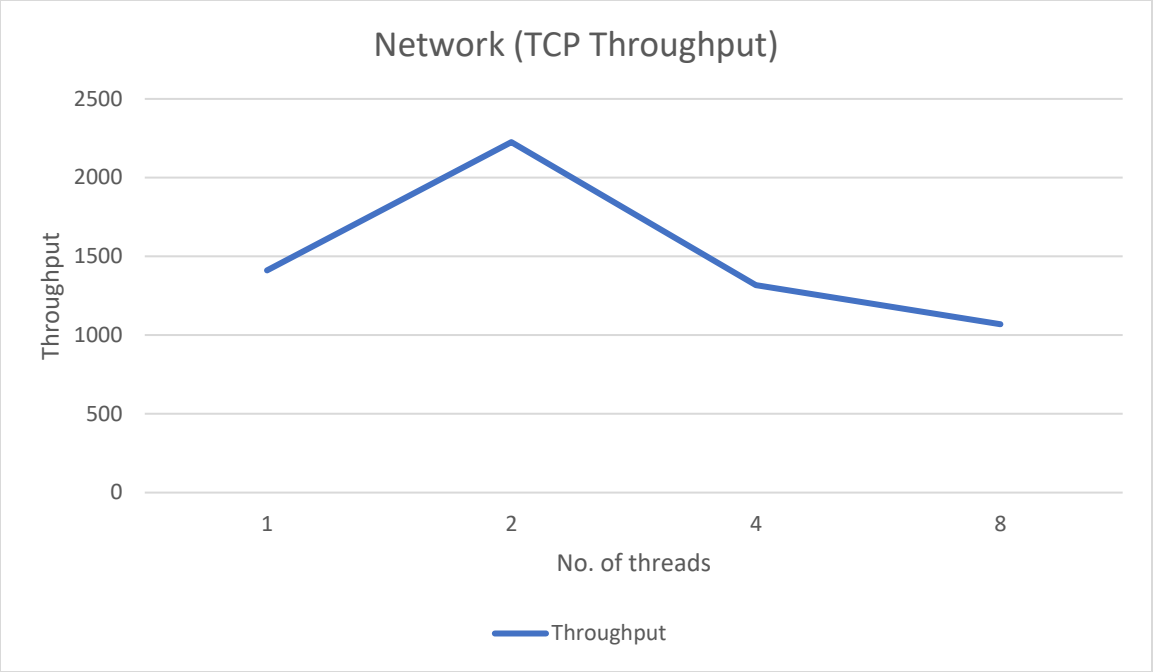
We've implemented TCP server-client communication where client is sending the data and server is receiving and replying (acknowledging). From the graphs, we see that latency increases the number of threads. Throughput increases as we go from 1 thread to 2 thread and then gradually decreases.

Instance: baremetal

Operation System: CC-CentOS7-1610.0
No. of cores: 24

No. of threads	Latency	Throughput
1	0.045	1411.19
2	0.05	2225.48
4	0.1943	1317.25
8	0.47	1069.03




Network (TCP Throughput)
2500
2000
1500
1000
500
0
Throughput
1
2
4
8
No. of threads
Throughput