Performance

1) Introduction

All experiments results performed for benchmarking on different components i.e. CPU, Disk, Memory and Network are enclosed in this report. Graphs for each performance results are generated so that precise and understandable comparative analysis can be achieved. All the results are tested on chameleon instance having configuration as Disk Size=40GB, Ram=4096MB, CPU=2, Image Name= CC-CentOS7. For measuring time measurements, I went through many timing options which are available in C and finally decided to go with gettimeofday() function which gives measurements in microseconds precision.

2) CPU Benchmark Test

Performance Parameters

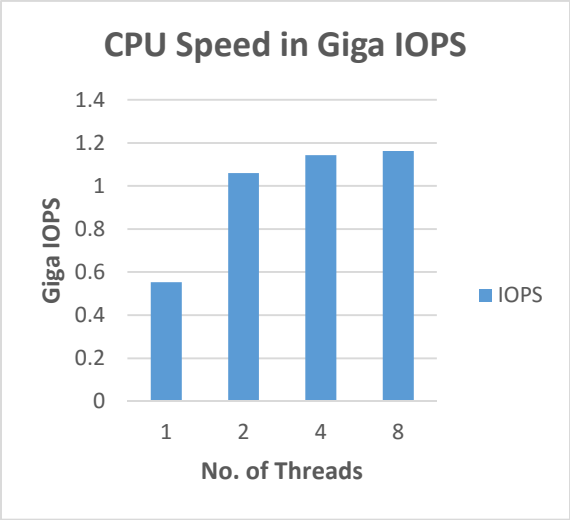
1) **Memory Size to be read/write: 1GB**
2) **Operations performed: CPU Integer and Double Operations, CPU for AVX Instructions, Linpack Benchmark.**

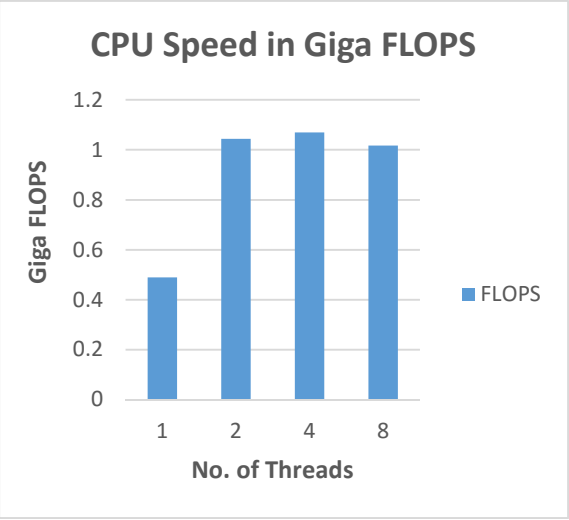
a) CPU Test

The following table shows results for time taken to compute 18*1e8 integer operations and double operations at different concurrency levels.

No. of Threads	Operation Type	IOPS	Total Time
1	Integer Operations	0.553382	2.891311
2	Integer Operations	1.05953	1.510103
4	Integer Operations	1.142756	1.400124
8	Integer Operations	1.162811	1.375976

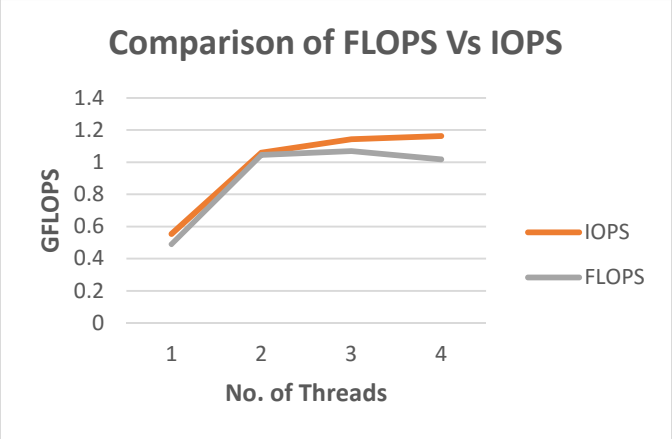
No. of Threads	Operation Type	FLOPS	Total Time
1	Double Operations	0.489817	3.266529
2	Double Operations	1.044355	1.532046
4	Double Operations	1.069634	1.495839
8	Double Operations	1.01654	1.573966





From the graph, it can be depicted that as concurrency level increases, CPU speed increases. But once concurrency level of CPU is achieved (i.e. 4 threads), CPU speed doesn't increases drastically.

The following graph shows comparison results for integer operations and double operations.



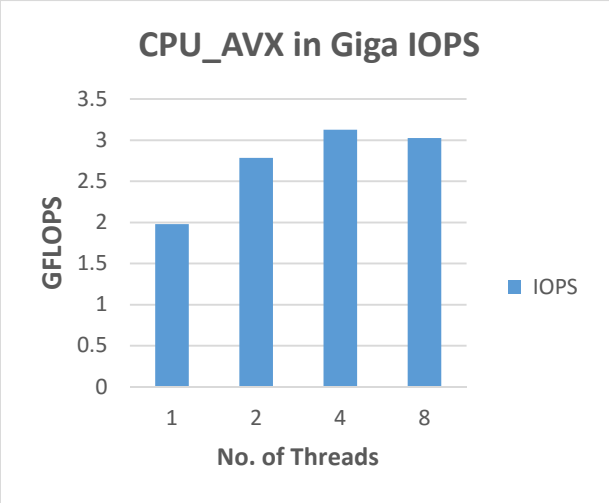
Concluding point made by this comparative analysis, is that CPU speed for integer operations are higher than CPU speed for double operations. The reason for such results is that integer instructions runs faster than double instructions.

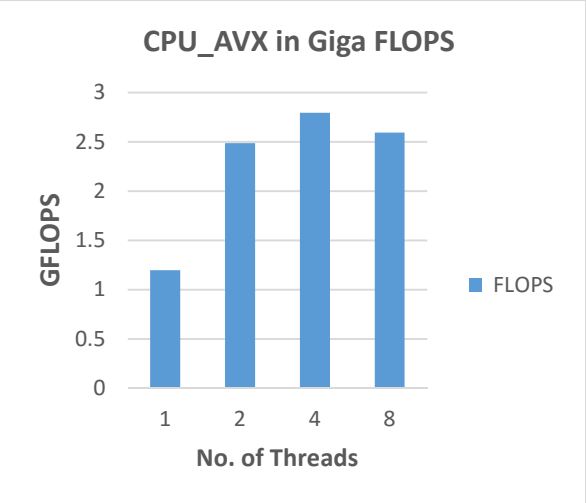
b) CPU AVX Test

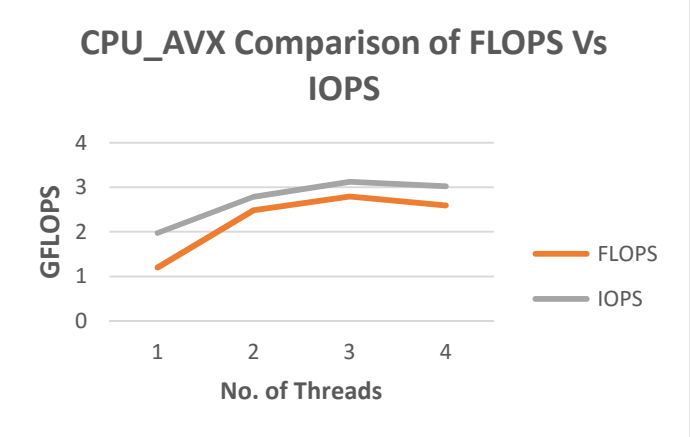
The following table shows results for time taken to compute 18*1e8 avx integer operations and avx double operations at different concurrency levels.

No of Threads	Operation Type	IOPS
1	Integer Operations	1.977886
2	Integer Operations	2.783974
4	Integer Operations	3.127421
8	Integer Operations	3.026734

No of Threads	Operation Type	FLOPS
1	Double Operations	1.196743
2	Double Operations	2.487355
4	Double Operations	2.793458
8	Double Operations	2.593454







From the above graph it is depicted that CPU speed increases with increase in concurrency level but once concurrency level of CPU is achieved (i.e. 4 threads), CPU speed doesn't increases drastically.

c) Theoretical Performance Calculations

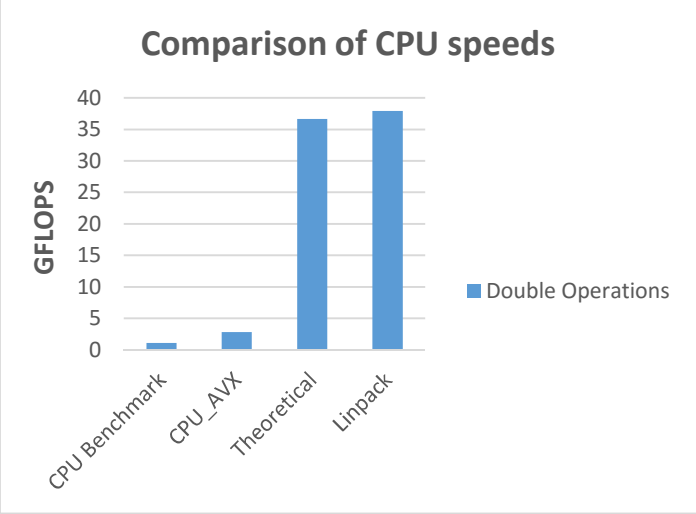
Formula for theoretical performance = CPU Speed * Number of Cores * Instruction per Cycle

We fetched this factors for our instance by running cat /proc/cpuinfo command.

Theoretical Performance for integer precision instructions= 2.30 *2 * 16 = 73.6 GFlops
Theoretical Performance for double precision instructions= 2.30 * 2 * 8 = 36.6 GFlops

d) Linpack Comparison with Theoretical Performance, CPU Benchmark, CPU AVX.

Operation Type	CPU Benchmark	CPU_AVX	Theoretical	Linpack
Double Operations	1.069634	2.793458	36.64	37.94



Conclusion: From above graph, it is concluded that linpack benchmark gives higher performance for CPU speed. CPU Speed obtained from our benchmark is much lower than theoretical performance and linpack performance. The reason behind such results is that linpack utilizes all the parallelism concepts to compute higher computations. Also, linpack performance is almost similar to the theoretical performance.

3) Memory Benchmark

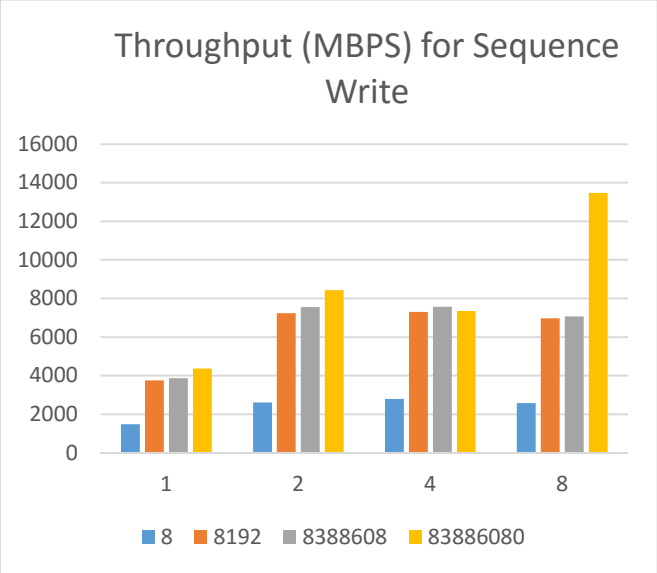
Performance Parameters

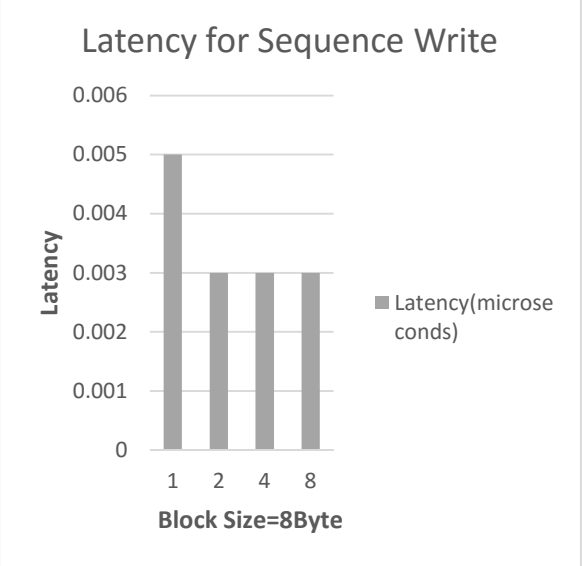
1) **Memory Size to be read/write: 1GB**
2) **Operations performed: Sequence Write, Random Write, Read then Write and Stream Benchmark.**

a) Sequence Write

The following table shows throughput, latency and time taken for writing 1GB of data into memory sequentially. Note that we have shown results for the data written into memory into chunks of 8B,8KB,8MB,80MB block sizes for different concurrency levels(1,2,4,8 Threads).

No of Threads	Block size	Throughput (MBPS)	Operation	Time(seconds)	Latency (microseconds)
1	8	1473.549216	Sequen Write	0.868651	0.000005
	8192	3750.091555	Sequen Write	0.341325	0.002083
	8388608	3870.764839	Sequen Write	0.330684	2.066775
	83886080	4363.209959	Sequen Write	0.293362	18.335125
2	8	2604.224948	Sequen Write	0.491509	0.000003
	8192	7244.121226	Sequen Write	0.176695	0.001078
	8388608	7548.549558	Sequen Write	0.169569	1.059806
	83886080	8436.760548	Sequen Write	0.151717	9.482312
4	8	2799.993	Sequen Write	0.457144	0.000003
	8192	7299.519828	Sequen Write	0.175354	0.00107
	8388608	7569.395986	Sequen Write	0.169102	1.056887
	83886080	7359.790246	Sequen Write	0.173918	10.869875
8	8	2583.150192	Sequen Write	0.495519	0.000003
	8192	6968.641115	Sequen Write	0.18368	0.001121
	8388608	7073.699103	Sequen Write	0.180952	1.13095
	83886080	13480.49541	Sequen Write	0.094952	5.9345



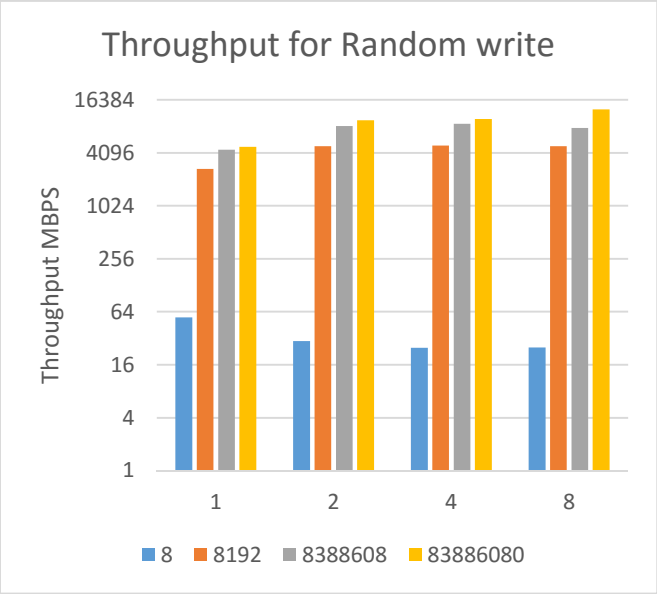


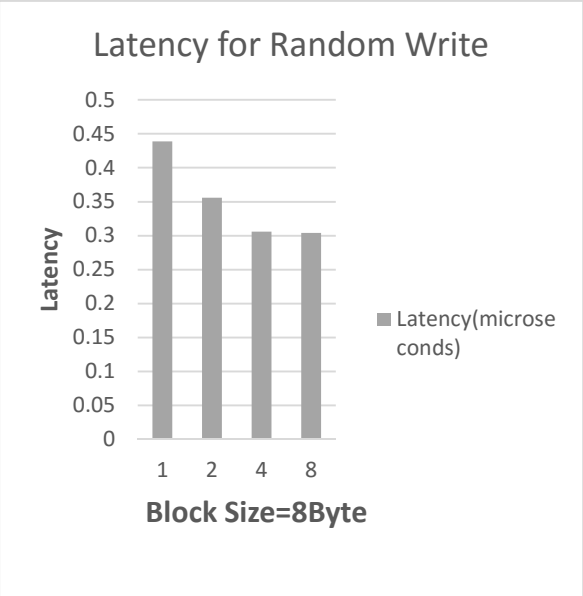
From the above graphs, it is depicted that as block size increase with increase in concurrency level, throughput increases and latency decreases. The reason behind such behavior is that, there is increase in parallelism with increase in number of threads. Hence **strong scaling** is achieved. Also, once concurrency level of CPU is achieved (i.e 4 Threads), throughput and latency doesn't varies drastically.

b) Random Write

The following table shows throughput, latency and time taken for writing 1GB of data into memory randomly. Note that we have shown results for the data written into memory into chunks of 8B,8KB,8MB,80MB block sizes for different concurrency levels(1,2,4,8 Threads).

No of Threads	Block size	Throughput (MBPS)	Operation	Time(seconds)	Latency (microseconds)
1	8	55.044227	Random Write	23.254028	0.000139
	8192	2678.98441	Random Write	0.477793	0.002916
	8388608	4455.27323	Random Write	0.2873	1.795625
	83886080	4781.79332	Random Write	0.267682	16.730125
2	8	29.815767	Random Write	42.930306	0.000256
	8192	4877.95583	Random Write	0.262405	0.001602
	8388608	8270.87103	Random Write	0.15476	0.96725
	83886080	9609.10463	Random Write	0.133207	8.325437
4	8	24.953334	Random Write	51.29575	0.000306
	8192	4943.211	Random Write	0.258941	0.00158
	8388608	8766.58288	Random Write	0.146009	0.912556
	83886080	9902.75188	Random Write	0.129257	8.078563
8	8	25.10858	Random Write	50.978589	0.000304
	8192	4879.70386	Random Write	0.262311	0.001601
	8388608	7858.93217	Random Write	0.162872	1.01795
	83886080	12805.5064	Random Write	0.099957	6.247313





From the above graphs, it is depicted that as block size increase with increase in concurrency level, throughput increases and latency decreases. The reason behind such behavior is that, there is increase in parallelism with increase in number of threads. Hence **strong scaling** is achieved. Also, once concurrency level of CPU is achieved (i.e 4 Threads), throughput and latency doesn't varies drastically.

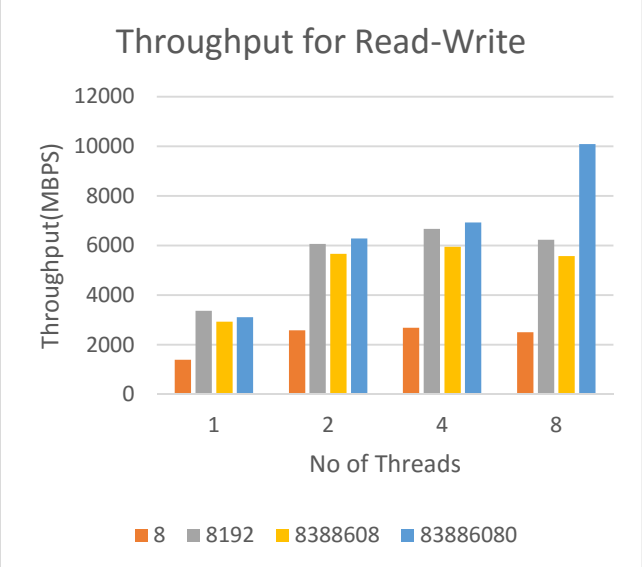
c) Read Write

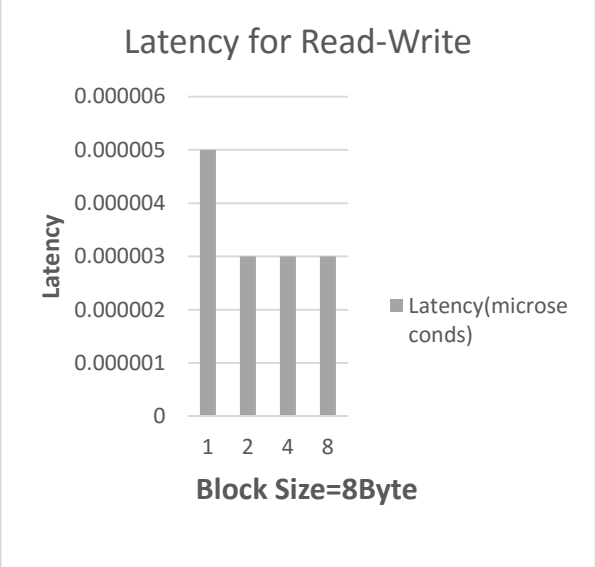
The following table shows throughput, latency and time taken for reading and writing 1GB of data into memory. Note that we have shown results for the data read and written into memory into chunks of 8B,8KB,8MB,80MB block sizes for different concurrency levels(1,2,4,8 Threads).

Rushit Pandya A20381916
Achal Lathia A20375208

PA-1-PERFORMANCE EVALUATION

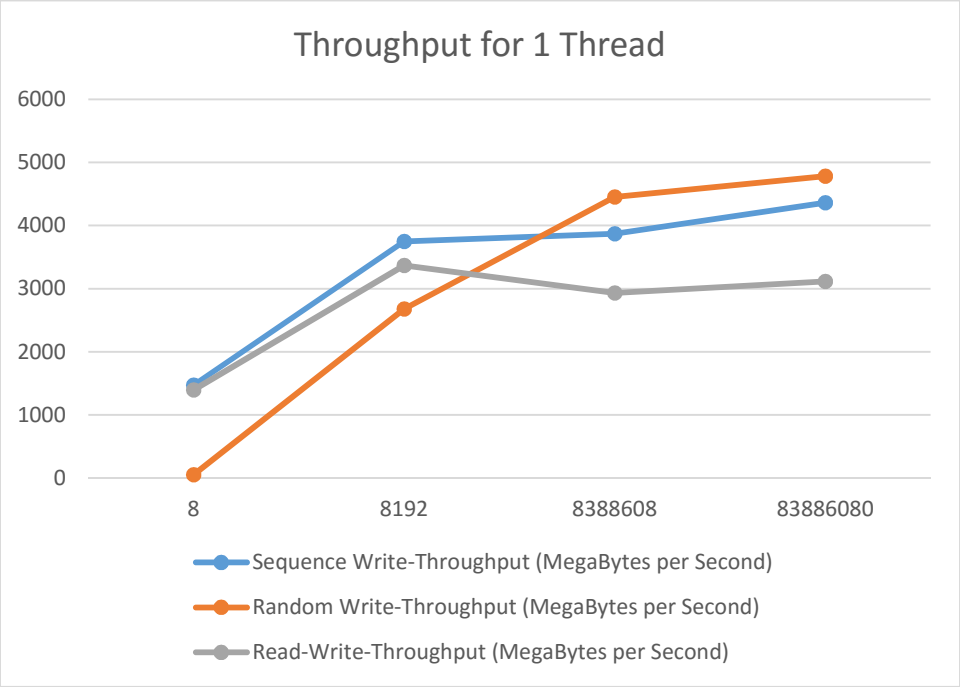
No of Threads	Block size	Throughput (MBPS)	Operation	Time(seconds)	Latency (microseconds)
1	8	1396.47	Read-Write	0.916597	0.000005
	8192	3366.26	Read-Write	0.380244	0.002321
	8388608	2931.866	Read-Write	0.436582	2.728638
	83886080	3116.433	Read-Write	0.410726	25.67038
2	8	2585.9	Read-Write	0.494992	0.000003
	8192	6063.104	Read-Write	0.211113	0.001289
	8388608	5669.612	Read-Write	0.225765	1.411031
	83886080	6285.756	Read-Write	0.203635	12.72719
4	8	2688.364	Read-Write	0.476126	0.000003
	8192	6676.159	Read-Write	0.191727	0.00117
	8388608	5945.193	Read-Write	0.2153	1.345625
	83886080	6936.316	Read-Write	0.184536	11.5335
8	8	2502.938	Read-Write	0.511399	0.000003
	8192	6230.985	Read-Write	0.205425	0.001254
	8388608	5571.031	Read-Write	0.22976	1.436
	83886080	10086.68	Read-Write	0.1269	7.93125

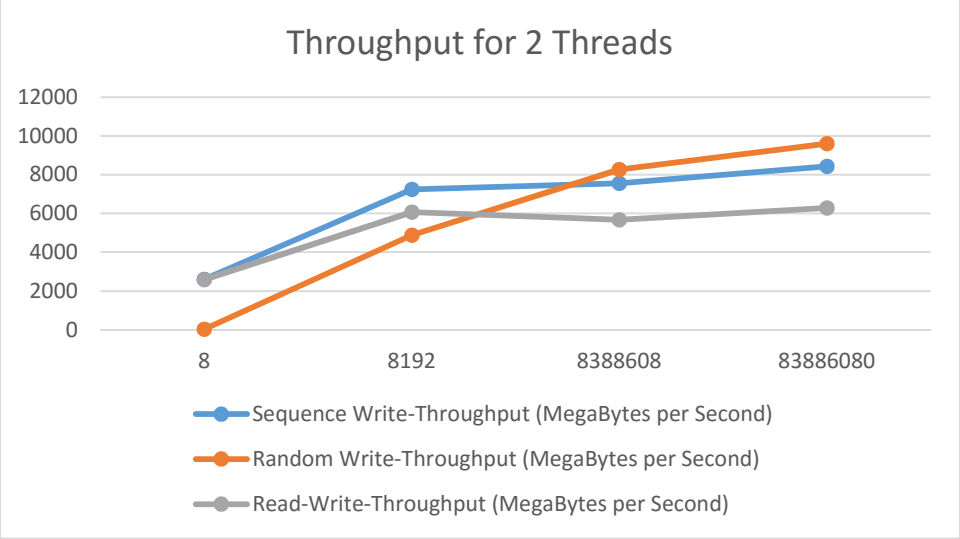


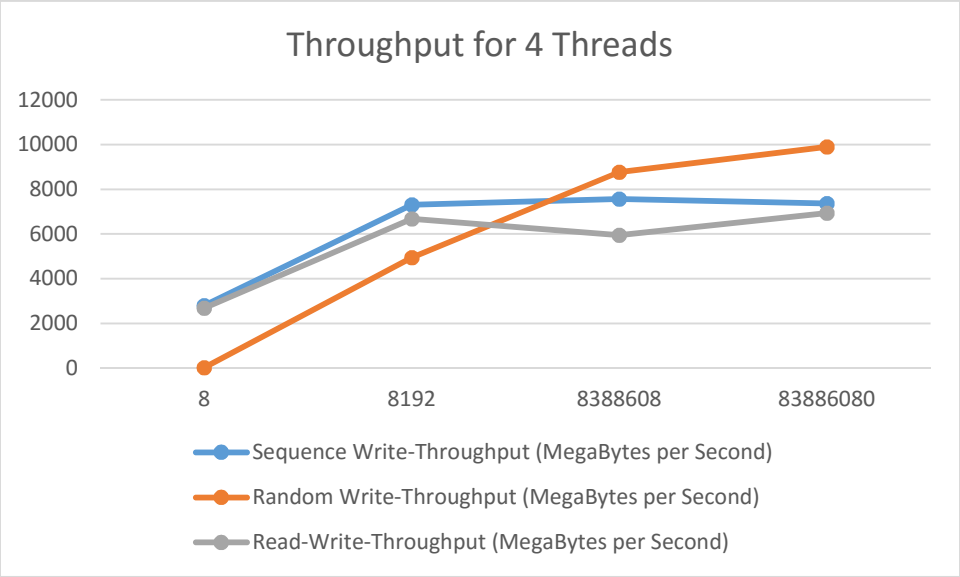


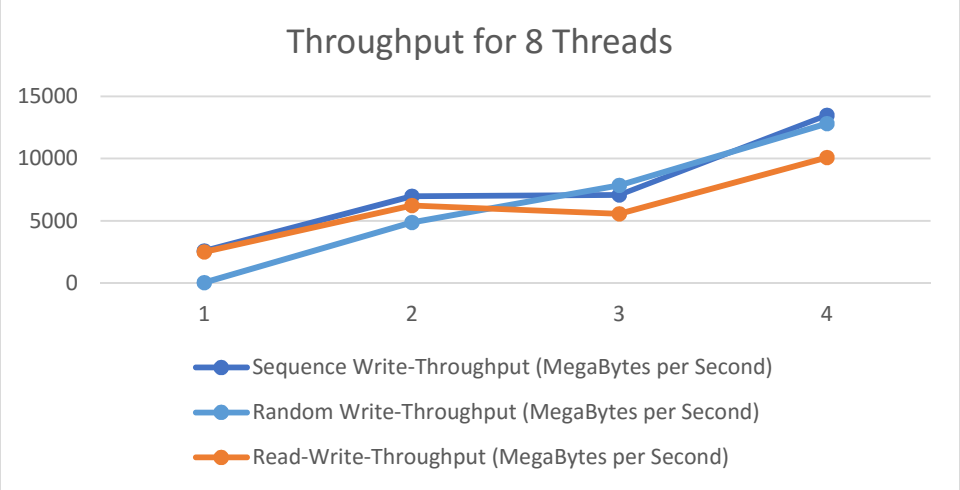
From the above graphs, it is depicted that as that block size increase with increase in concurrency level, throughput increases and latency decreases. The reason behind such behavior is that, there is increase in parallelism with increase in number of threads. Hence **strong scaling** is achieved. Also, once concurrency level of CPU is achieved (i.e 4 Threads), throughput and latency doesn't varies drastically.

d) Comparison Analysis









Above shown are the graphs for throughput for all 3 operations at different concurrency levels.

Conclusion: we can make comparative analysis from above 4 graphs that for any number of threads, throughput of sequence write > throughput of random write > throughput of read write. The reason behind such result is, reading or writing into memory sequentially is always faster than reading or writing it randomly. As every time random block is selected for writing into memory, it is difficult for the CPU to fetch the location and write at the particular random location. On the other side, sequential writing into memory is always faster because CPU has to fetch and write into the blocks in sequence. Also, throughput for read-write will be lower than sequence write and random write because read and write both operations have to be performed.

e) Stream Benchmark

We have ran stream benchmark with different parameters. For 1 thread, 2 threads, 4threads, 8threads stream benchmark outputs 7496.5 MBPS, 14852.9MBPS, 14179.8MBPS, 11264.4MBPS respectively. However, this results are for ADD operation. Generally, it

performs 8 bytes operations per array element. However, throughput for 1 thread stream benchmark are similar to our benchmark throughput, but for other threads stream benchmark have higher performance than our benchmarks performance.

f) Theoretical Performance

Theoretical Memory Bandwidth can calculated as (Base DRAM Clock frequency) * (Number of data transfers per clock) * (Memory Bus Width) * (Number of Interfaces).

While searching specification for Intel Xeon E3-12xx (Sandy Bridge) having 2.3GHZ frequency, we found memory bandwidth= 21GB/s. The link for specifications is https://ark.intel.com/products/53401. It is clearly stated that theoretical performance is higher than stream benchmark and our benchmark.

4) Disk Benchmark

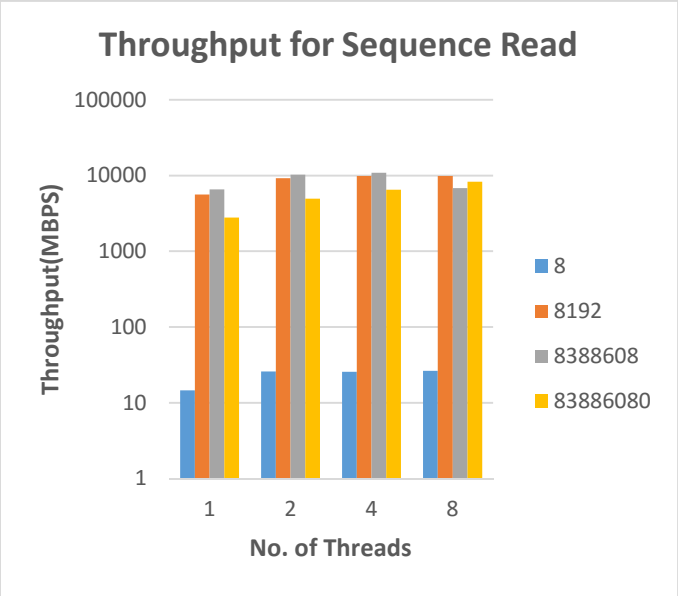
Performance Parameters

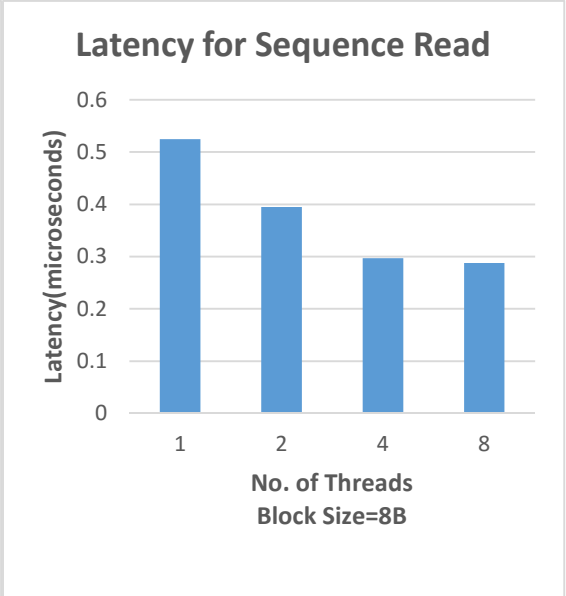
1) **File Size to be read/write: 1GB**
2) **Operations performed: Sequence Read, Random Read, Read then Write and Iozone Benchmark.**

a) Sequence Read

The following table shows throughput, latency and time taken for reading 1GB of data from file sequentially. Note that we have shown results for the data read from file into chunks of 8B,8KB,8MB,80MB block sizes for different concurrency levels(1,2,4,8 Threads).

Threads	Block_Size	Throughput (MBPS)	Operation	Time(seconds)	Latency(microseconds)
1	8	14.53074	Sequen Read	88.08915	0.000525
1	8192	5613.493	Sequen Read	0.228022	0.001392
1	8388608	6585.921	Sequen Read	0.194354	1.214712
1	83886080	2797.35	Sequen Read	0.457576	28.5985
2	8	25.86813	Sequen Read	49.48175	0.000295
2	8192	9218.78	Sequen Read	0.138847	0.000847
2	8388608	10304.05	Sequen Read	0.124223	0.776394
2	83886080	4958.05	Sequen Read	0.258166	16.13538
4	8	25.65913	Sequen Read	49.88478	0.000297
4	8192	9890.815	Sequen Read	0.129413	0.00079
4	8388608	10945.97	Sequen Read	0.116938	0.730863
4	83886080	6517.278	Sequen Read	0.196401	12.27506
8	8	26.47055	Sequen Read	48.35562	0.000288
8	8192	9854.492	Sequen Read	0.12989	0.000793
8	8388608	6839.47	Sequen Read	0.187149	1.169681
8	83886080	8286.56	Sequen Read	0.154467	9.654187





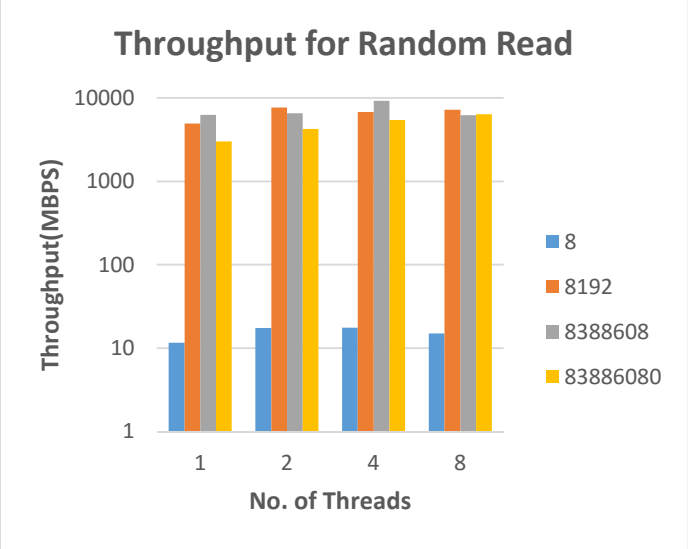
From the above graphs, it is depicted that as block size increase with increase in concurrency level, throughput increases and latency decreases. The reason behind such behavior is that, there is increase in parallelism with increase in number of threads. Hence **strong scaling** is achieved. Also, once concurrency level of CPU is achieved (i.e 4 Threads), throughput and latency doesn't varies drastically.

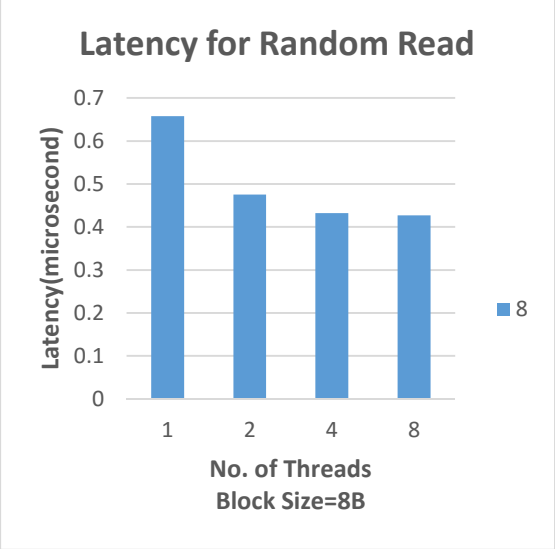
b) Random Read

The following table shows throughput, latency and time taken for reading 1GB of data from file randomly. Note that we have shown results for the data read from file into chunks of 8B,8KB,8MB,80MB block sizes for different concurrency levels(1,2,4,8 Threads).

Threads	Block_Size	Throughput (MBPS)	Operation	Time(seconds)	Latency(microseconds)
1	8	11.59267	Random Read	110.4146	0.000658
1	8192	4920.673	Random Read	0.260127	0.001588
1	8388608	6272.235	Random Read	0.204074	1.275462
1	83886080	3022.725	Random Read	0.423459	26.46619
2	8	17.52295	Random Read	73.04708	0.000435
2	8192	7696.563	Random Read	0.166308	0.001015
2	8388608	6536.014	Random Read	0.195838	1.223988
2	83886080	4245.341	Random Read	0.301507	18.84419
4	8	17.65632	Random Read	72.49528	0.000432

4	8192	6796.003	Random Read	0.188346	0.00115
4	8388608	9239.476	Random Read	0.138536	0.86585
4	83886080	5460.704	Random Read	0.234402	14.65013
8	8	15.05224	Random Read	85.03719	0.000507
8	8192	7194.447	Random Read	0.177915	0.001086
8	8388608	6201.43	Random Read	0.206404	1.290025
8	83886080	6371.614	Random Read	0.200891	12.55569





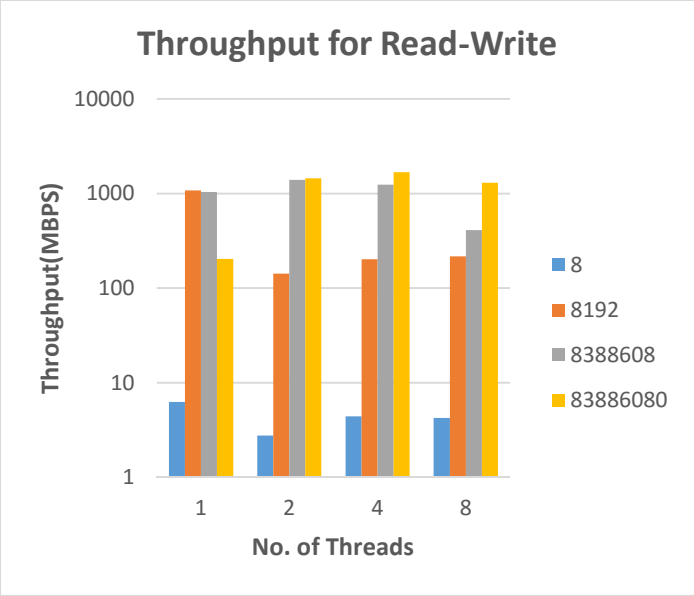
From the above graphs, it is depicted that as block size increase with increase in concurrency level, throughput increases and latency decreases. The reason behind such behavior is that, there is increase in parallelism with increase in number of threads. Hence **strong scaling** is achieved. Also, once concurrency level of CPU is achieved (i.e 4 Threads), throughput and latency doesn't varies drastically.

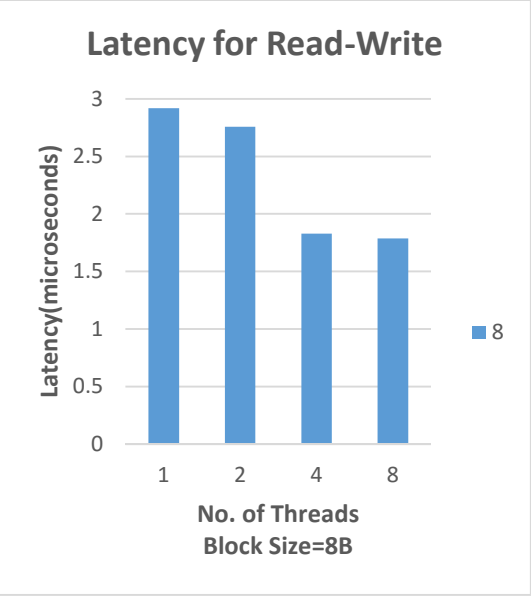
c) **Read-Write**

The following table shows throughput, latency and time taken for reading and writing 1GB of data into file. Note that we have shown results for the data read and written into file into chunks of 8B,8KB,8MB,80MB block sizes for different concurrency levels(1,2,4,8 Threads).

Threads	Block_Size	Throughput (MBPS)	Operation	Time(seconds)	Latency(microseconds)
1	8	6.264041	Read Write	204.3409	0.001218
1	8192	1080.629	Read Write	1.184495	0.00723
1	8388608	1037.501	Read Write	1.233734	7.710838

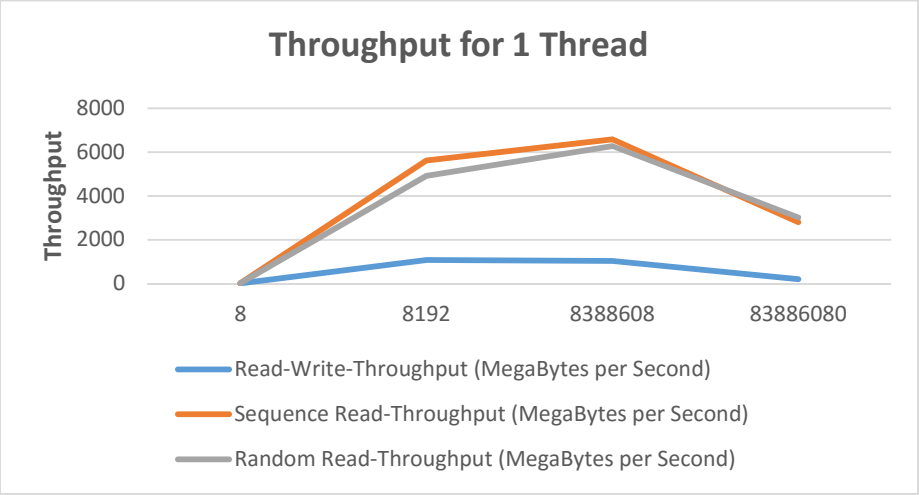
1	83886080	204.1564	Read Write	6.269704	391.8565
2	8	2.768483	Read Write	462.347	0.002756
2	8192	141.9975	Read Write	9.014243	0.055019
2	8388608	1395.67	Read Write	0.917122	5.732012
2	83886080	1451.792	Read Write	0.881669	55.10431
4	8	4.411409	Read Write	290.1568	0.001729
4	8192	201.8253	Read Write	6.34212	0.038709
4	8388608	1241.67	Read Write	1.03087	6.442938
4	83886080	1681.836	Read Write	0.761073	47.56706
8	8	4.272281	Read Write	299.6057	0.001786
8	8192	216.4247	Read Write	5.914296	0.036098
8	8388608	410.2893	Read Write	3.11975	19.49844
8	83886080	1296.896	Read Write	0.986972	61.68575

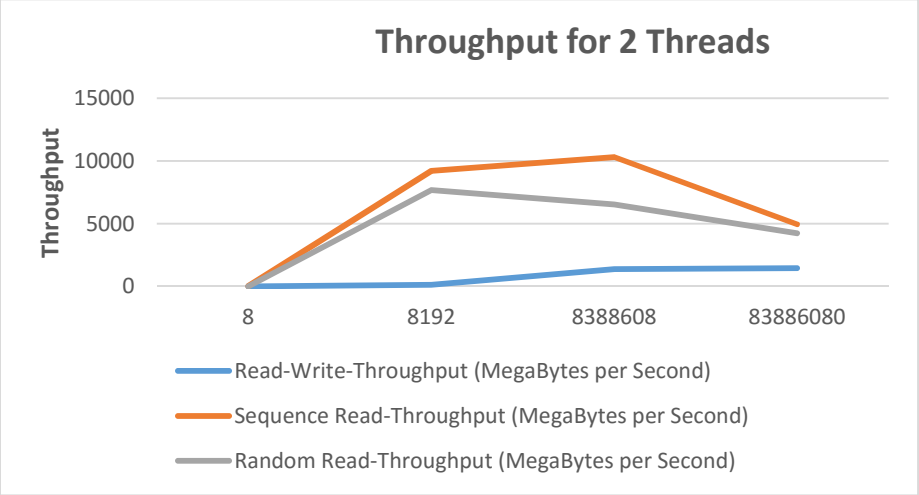


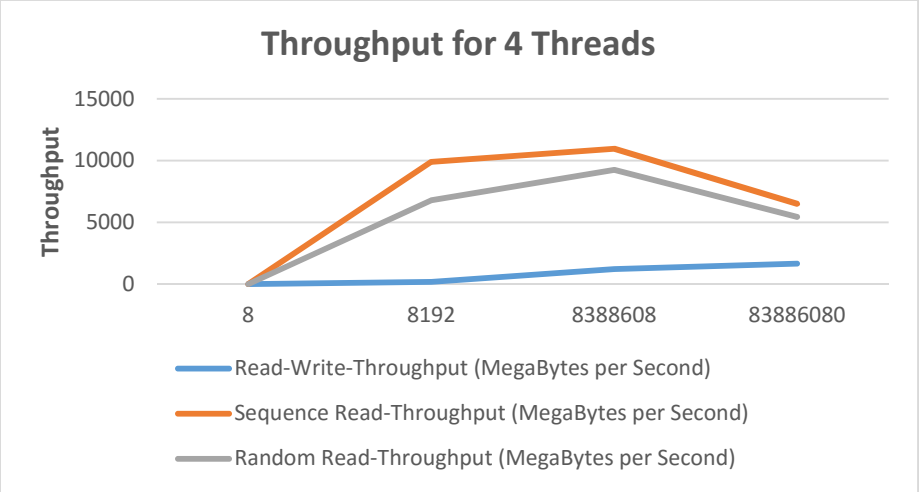


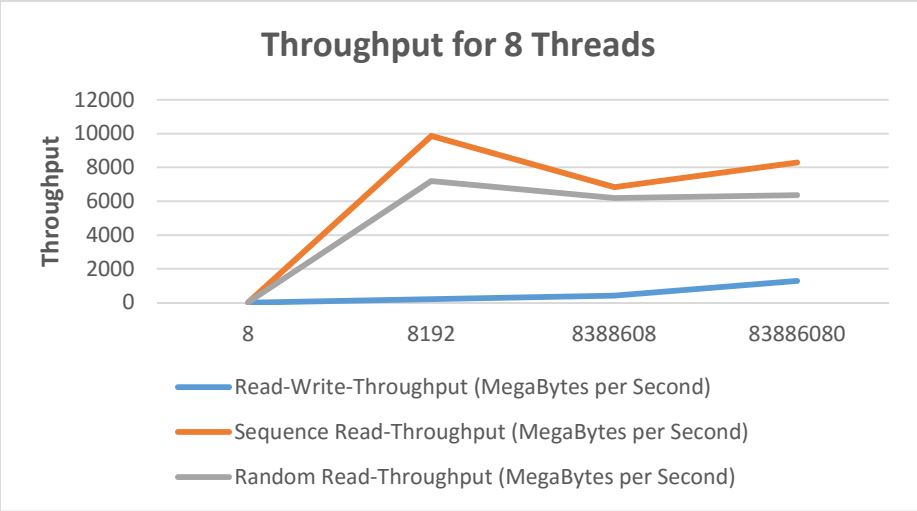
From the above graphs, it is depicted that as block size increase with increase in concurrency level, throughput increases and latency decreases. The reason behind such behavior is that, there is increase in parallelism with increase in number of threads. Hence **strong scaling** is achieved. Also, once concurrency level of CPU is achieved (i.e 4 Threads), throughput and latency doesn't varies drastically.

d) Comparative Analysis









Above shown are the graphs for throughput for all 3 operations at different concurrency levels.

Conclusion: we can make comparative analysis from above 4 graphs that for any number of threads, throughput of sequence read > throughput of random read > throughput of read write. The reason behind such result is, reading or writing into file sequentially is always faster than reading or writing it randomly. As every time random block is selected for writing into file, it is difficult for the CPU to fetch the location and write at the particular random location. On the other side, sequential writing into file is always faster because CPU has to fetch and write into the blocks in sequence. Also, throughput for read-write will be lower than sequence write and random write because read and write both operations have to be performed.

e) Iozone Benchmark

We ran iozone benchmark in an automatic mode and fetched all the results.

1) For writing into 50MB of file, bandwidth of disk is 2450MBPS, 2101MBPS and 2209MBPS for 1KB, 8KB, 16KB record length in iozone bechmark. However, we have calculated bandwidth for writing a file of 1GB in 8KB chunks size which is 1341.8 MBPS. We can clearly argue that our benchmark results are slightly lower than Iozone Benchmark results.
2) For reading 50MB of file, bandwidth of disk is 6547MBPS, 7893MBPS and 4678MBPS for 1KB, 8KB, 16KB record length in iozone benchmark. However, we have calculated bandwidth for reading a file of 1GB in 8KB chunks size which is 3701 MBPS. We can clearly argue that our benchmark results are slightly lower than Iozone Benchmark results.
3) For reading 50MB of file randomly, bandwidth of disk is 6926MBPS, 7218MBPS and 5689MBPS for 1KB, 8KB, 16KB record length in iozone benchmark. However, we have calculated bandwidth for reading a file of 1GB in 8KB chunks size which is 2301 MBPS. We can clearly argue that our benchmark results are slightly lower than Iozone Benchmark results.

5) Network Benchmark

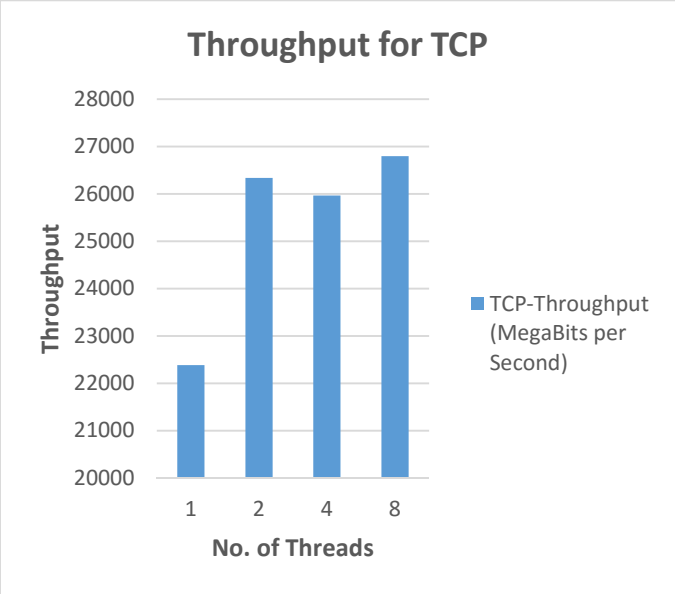
Performance Parameters

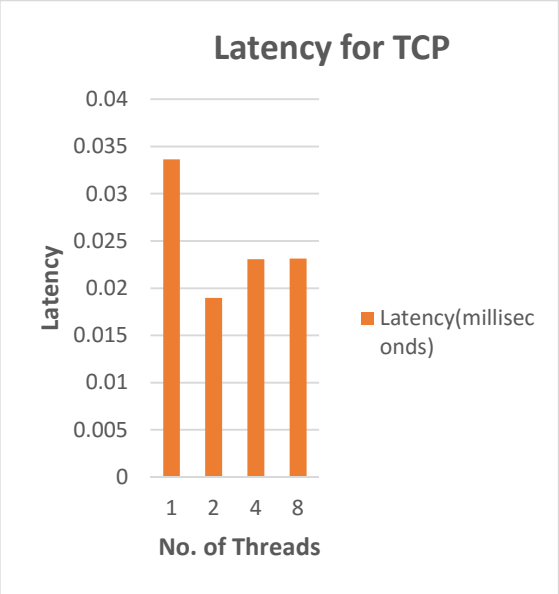
1) **Total Data to be transferred: 1GB**
2) **Packet Size: 64KB (Note: For UDP, we have used 63KB packet size as it is maximum size that can be send)**
3) **Operations performed: TCP Benchmark, UDP Benchmark and Iperf Benchmark.**

a) TCP Benchmark

The following table shows throughput and latency for sending 1GB of data from client end to server end using TCP socket connection in 64KB chunk size. Note that we have shown results for the data sent into chunks of 64KB for different concurrency levels (1, 2, 4, 8 Threads).

Threads	Operation	Throughput (Mbps)	Latency (milliseconds)
1	TCP	22387.5136	0.03363
2	TCP	26340.836	0.018982
4	TCP	25967.9592	0.023071
8	TCP	26798.5568	0.023142





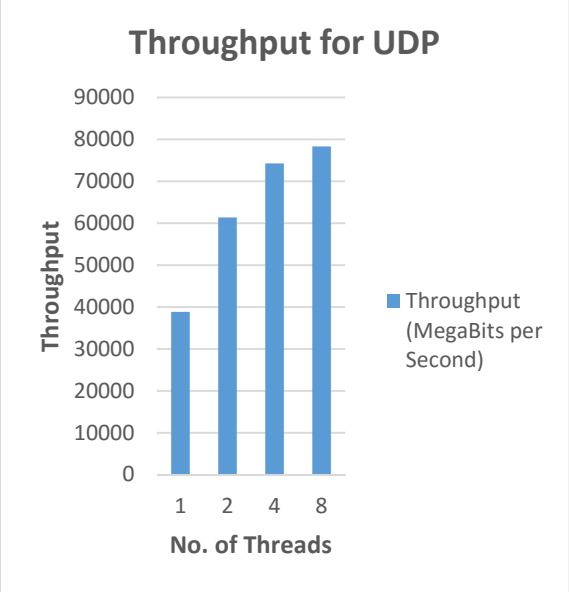
From the above graphs, it is depicted that as concurrency level increases, throughput increases and latency decreases. The reason behind such behavior is that, there is increase in parallelism

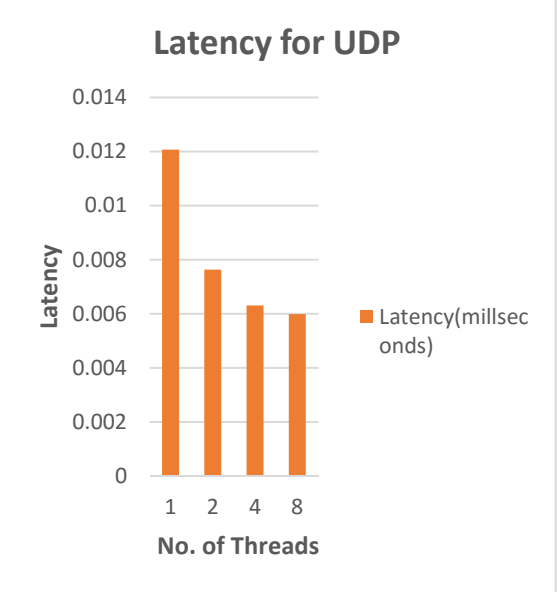
with increase in number of threads. Hence **strong scaling** is achieved. Also, once concurrency level of CPU is achieved (i.e 4 Threads), throughput and latency doesn't varies drastically.

b) UDP Connection

The following table shows throughput and latency for sending 1GB of data from client end to server end using UDP datagram connection in 63KB chunk size. Note that we have shown results for the data sent into chunks of 63KB for different concurrency levels (1, 2, 4, 8 Threads).

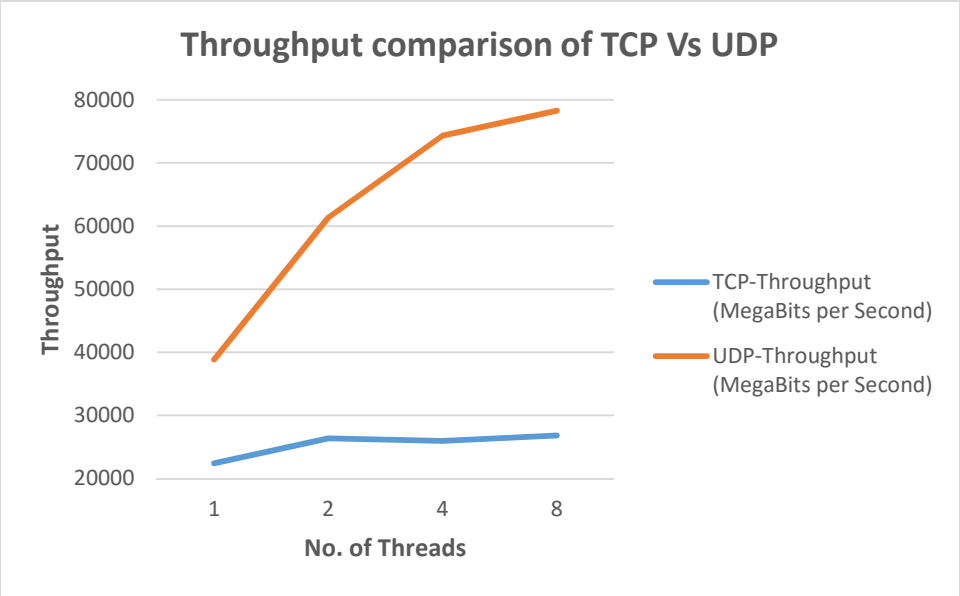
Threads	Operation	Throughput (Mbps)	Latency(milliseconds)
1	UDP	38824.64	0.012074
2	UDP	61363.3	0.007639
4	UDP	74303.85	0.006309
8	UDP	78298.69	0.005987





From the above graphs, it is depicted that as concurrency level increases, throughput increases and latency decreases. The reason behind such behavior is that, there is increase in parallelism with increase in number of threads. Hence **strong scaling** is achieved. Also, once concurrency level of CPU is achieved (i.e 4 Threads), throughput and latency doesn't varies drastically.

c) Comparative Analysis



Above shown are the graphs for throughput for all 2 operations at different concurrency levels.

Conclusion: we can make comparative analysis from above graph that UDP is faster than TCP as throughput for UDP is higher than TCP's throughput. The reason behind such results is that UDP is connection-less protocol and light-weight which means that it doesn't wait for any acknowledgement from the receiving end whereas TCP is a connection-oriented protocol and heavy-weight which means it waits for acknowledgement from the receiving end. Hence UDP transmission is always faster than TCP transmission.

d) IPERF Benchmark

We ran IPERF benchmark for both TCP and UDP connection. As maximum data that can be transferred between iperf client and server is 5GB, bandwidth obtained for sending 10packets of 5GB data in TCP connection is 43Gbits/sec. Whereas for UDP connection, bandwidth is 32.5Gbits/sec for sending 37.8 GB data. However, IPERF bandwidth performance is light higher than our performance benchmark for TCP and UDP network.