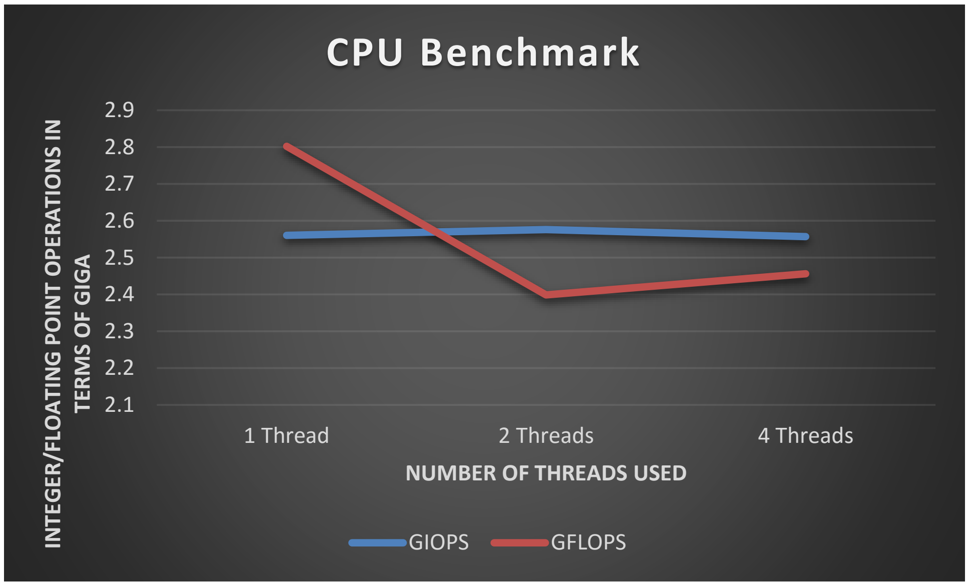
1. CPU
    a. Processor speeds of executing integer and floating point operations are as follows

Type	GIOPS	GFLOPS
1 Thread	2.560905	2.80228
2 Threads	2.576106	2.398774
4 Threads	2.557335	2.456215



The above graphs shows the GIOPS and GFLOPS achieved when the program is executed with single thread, 2 threads and 4 threads. When a single thread is run the Floating point operations give a significant GFLOPs count as it never lets the system idle and since there is only one core available which gives us 1 Hardware thread and 1 logical thread to perform our operations. Since Floating point operations may take more than one cycle to perform a single operation this should give a higher performance than IOPS which is exactly how it is reacting in this benchmark. As we push the application to 2 and 4 threads the performance deteriorates for floating point operations significantly as it takes a lot of execution cycles for each operations and the since the threads run concurrently the distribution of control to each of the threads can cause further drop in performance of the processor.

    b. The above table of values and graph depicts how the processor reacts to 1, 2 and 4 threads
    c. The theoretical performance peak performance is can be calculated by using

    Theoretical Peak = Clock Frequency * IPC * Number of Cores

    The experiment is performed on AWS cloud platform by initiating a t2.micro instance and the instance has the following configuration.

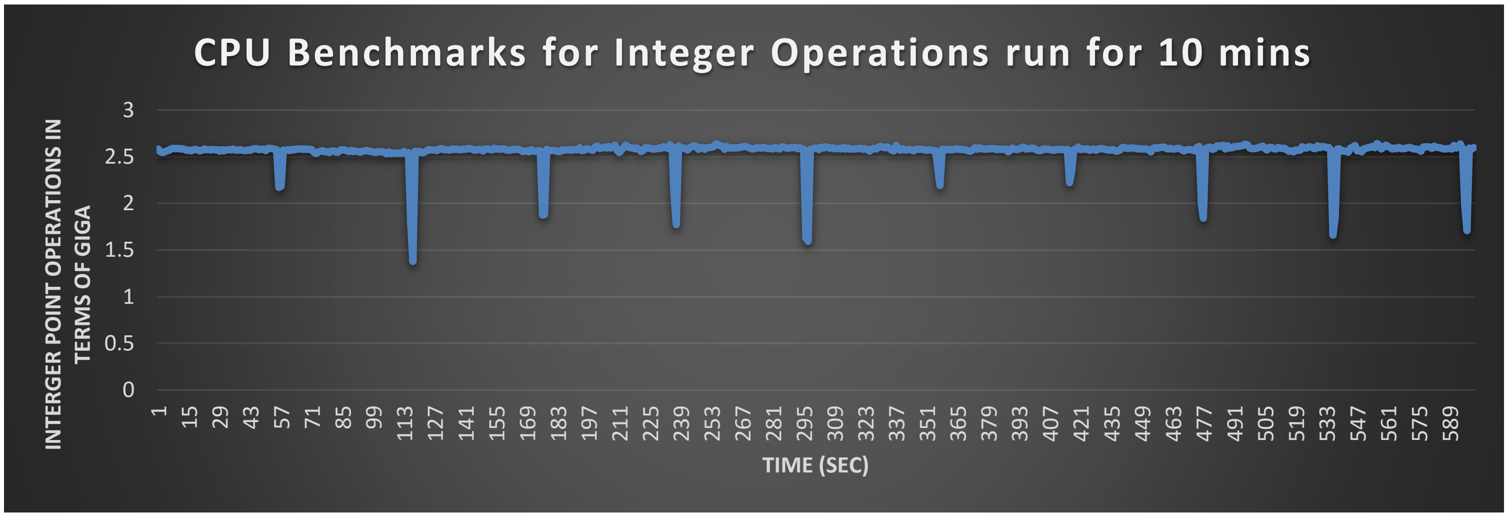
    Clock Frequency = 2.6GHz

Number of Cores = 1

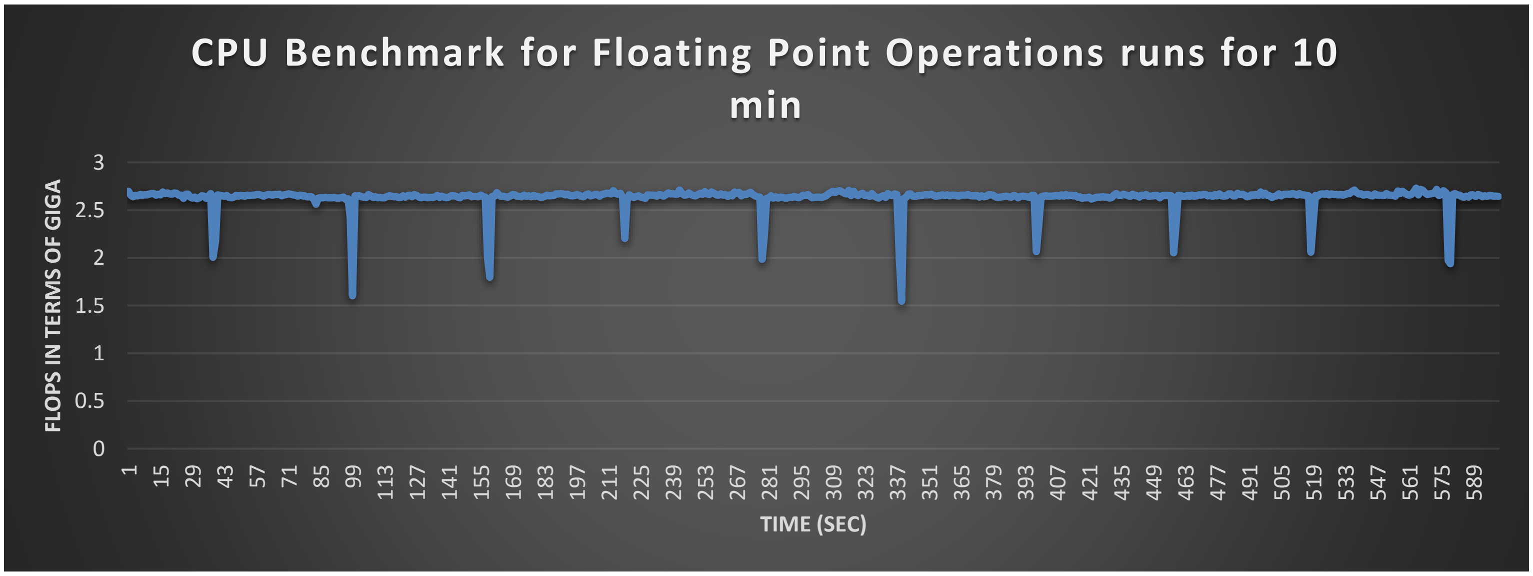
IPC = 16 for Xeon processor which is used in the AWS

Theoretical Peak = 41.6 * 10^9 flops/sec

d. The efficiency achieved by this program is 6.73%
e. Below is the chart which gives the performance test for CPU running for 10 min continuously and samples of the number of instructions executed are extracted every min.



Note: There are periodic dips in otherwise consistent performance and this could be small power fluctuations at the processor which might occur at constant time intervals. This conclusion is made based on the pattern of the collective low peaks which looks like a half sinusoidal wave.

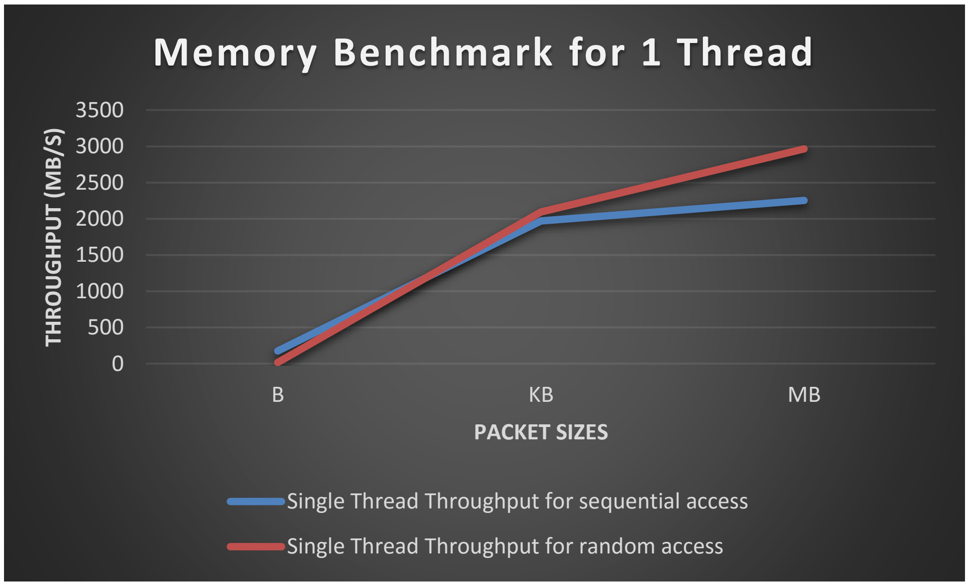


The above chart shows the behavior of the processor over a period of 10 mins with floating point operations being done

2. Memory has been benchmarked and the following table and graphs shows the performance behavior
    a. The code was written and C and the operations performed was memcpy()
    b. The values for the performance are listed below

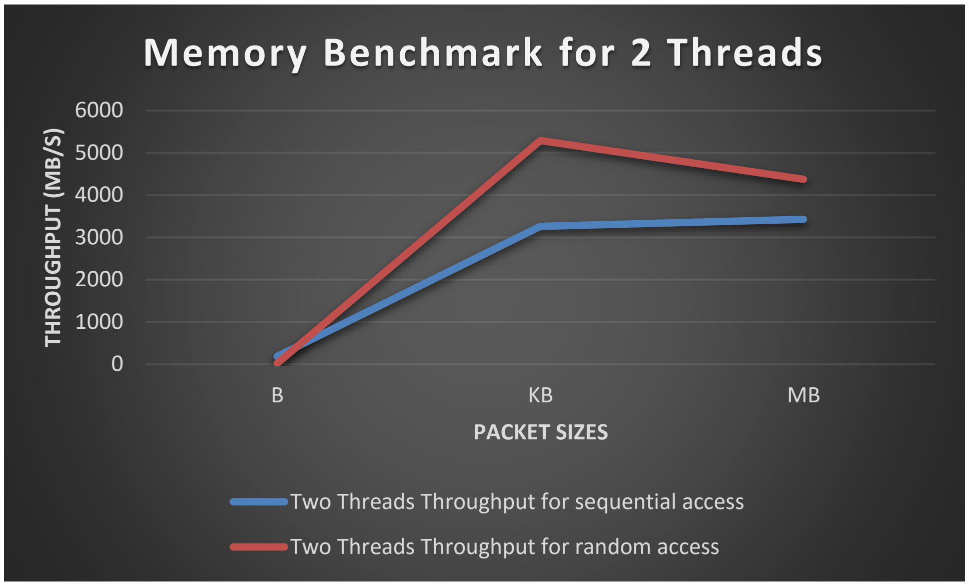
Single Thread		
Test	Throughput for sequential access	Throughput for random access
B	178.037	16.4949
KB	1968.51	2094.76
MB	2255.15	2964.55

Single Thread		
Test	Latency for sequential access(ms)	Latency for random access(ms)
B	5.36E-006	5.78E-005
KB	4.84E-007	2.09E-007
MB	4.23E-007	2.08E-007



Two Threads		
Test	Throughput for sequential access	Throughput for random access
B	186.23	15.8464
KB	3258.5	5289.33
MB	3426.82	4376.09

Two Threads		
Test	Latency for sequential access(ms)	Throughput for random access(ms)
B	1.02E-005	0.000120365
KB	5.85E-007	3.61E-007
MB	5.57E-007	4.36E-007



The only reason I can think of for random access in the memory module has a higher value is because I have initialized a variable with random address values which is outside the loop and in the time loop the counter used for the loop itself is used as the index of for the random array which could intern cause a sequential pattern. The random values had to calculated outside because the time to calculate rand() is not constant which can cause the accuracy to drop. One other reason that I can think of is the random values generated were not significantly varying through its whole range and this could have made a part of data available in the L1, L2 caches which could have caused the higher value.

c. The measurements of the latency is in milliseconds and Throughput in (MB/S)
d. The theoretical value of the RAM or Memory can be calculated using the below formula

Bandwidth = DDR clock rate * 8;

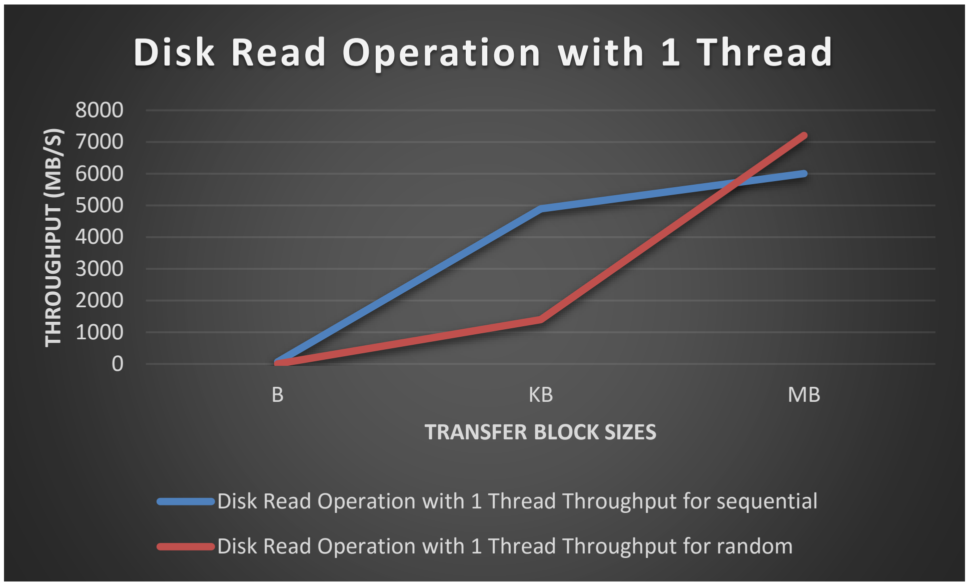
And the memory module used is DDR4 with a frequency of 2133MHz

Bandwidth = 17064 MB/s

3. Disk Benchmarking is done in C programming language and the operations performed is fread() and fwrite()
    a. Sys/time.h header file is included to measure the performance of the read/write operations
    b. The performance is checked for read, write with sequential, random with varying buffer sizes of 1B, 1KB and 1MB on both 1 and 2 threads, and the values are given below in form of tables and graphs

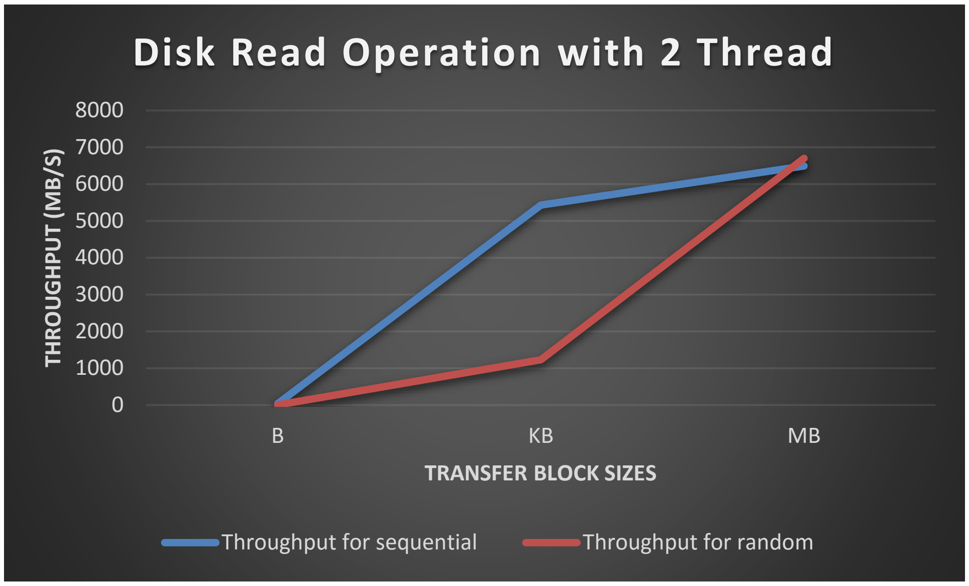
Disk Read Operation with 1 Thread		
Type	Throughput for sequential	Throughput for random
B	64.904	1.92146
KB	4886.72	1398.25
MB	5997.89	7202.77

Disk Read Operation with 1 Thread		
Type	Latency for sequential access(ms)	Latency for random access(ms)
B	1.47E-005	0.000496328
KB	1.95E-007	6.82E-007
MB	1.59E-007	1.32E-007



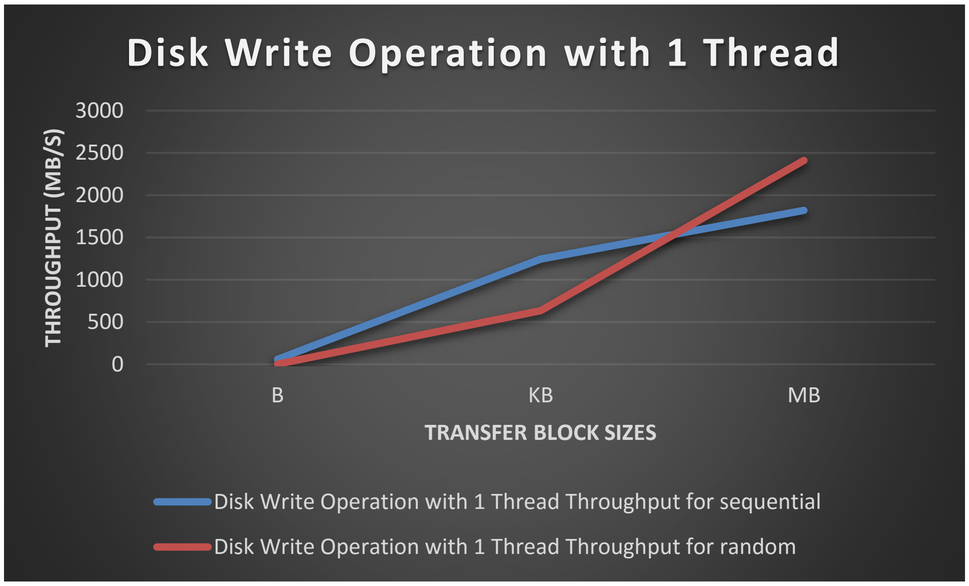
Disk Read Operation with 2 Thread		
Type	Throughput for sequential	Throughput for random
B	39.5282	1.50906
KB	5422.51	1227.27
MB	6490.17	6693.2

Disk Read Operation with 2 Thread		
Type	Latency for sequential access(ms)	Latency for random access(ms)
B	2.41E-005	0.000631968
KB	1.76E-007	7.77E-007
MB	1.47E-007	1.42E-007



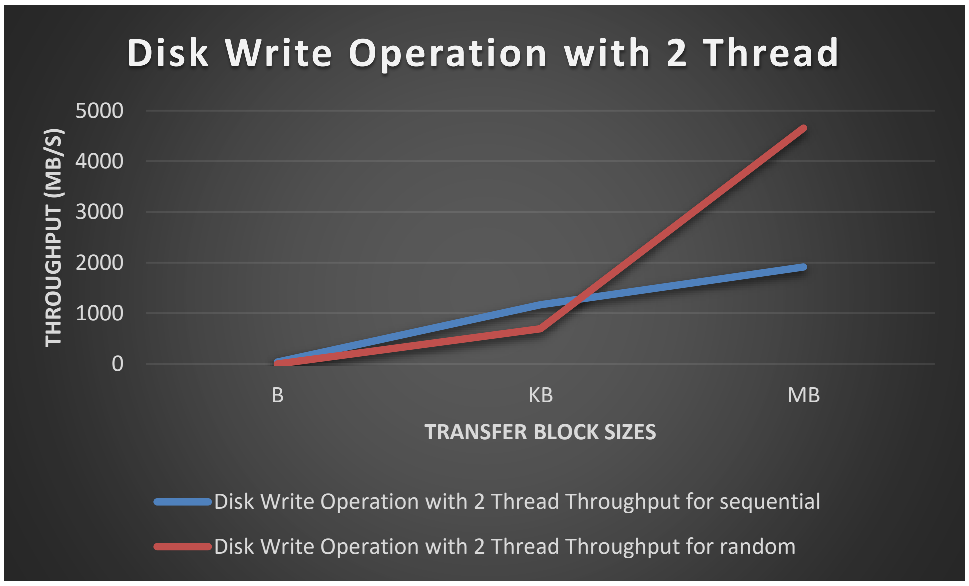
Disk Write Operation with 1 Thread		
Type	Throughput for sequential	Throughput for random
B	56.7311	1.01038
KB	1244.23	631.074
MB	1817.21	2407.89

Disk Write Operation with 1 Thread		
Type	Latency for sequential access(ms)	Latency for random access(ms)
B	1.68E-005	0.000943876
KB	7.66E-007	1.51E-006
MB	5.25E-007	3.96E-007



Disk Write Operation with 2 Thread		
Type	Throughput for sequential	Throughput for random
B	38.097	1.01678
KB	1171.23	698.492
MB	1913.5	4654.39

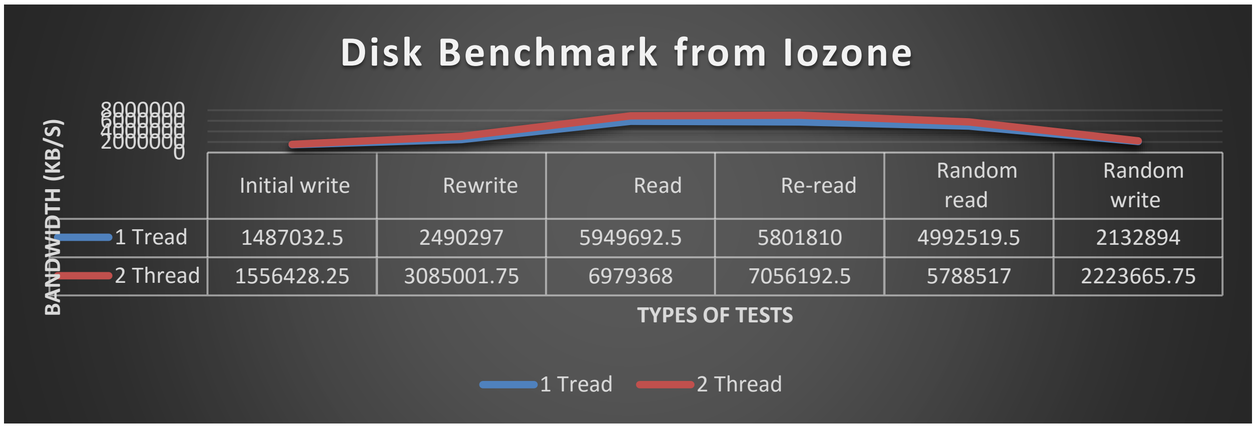
Disk Write Operation with 2 Thread		
Type	Latency for sequential access(ms)	Latency for random access(ms)
B	2.50E-005	0.000937934
KB	8.14E-007	1.37E-006
MB	4.98E-007	2.05E-007



If we observe the graphs carefully for all the read and write operations to disk sequential has a higher score for 1KB buffer size which could mean that the optimal packet transfer size incorporated in the architecture of the t2.micro instances could be using 1KB buffer size which in turn helps improve the throughput.

c. The metrics latency and throughput are measured in ms and MB/s respectively.

Extra Credits Questions



	Initial write	Rewrite	Read	Re-read	Random read	Random write
1 Tread	1487032.5	2490297	5949692.5	5801810	4992519.5	2132894
2 Thread	1556428.25	3085001.75	6979368	7056192.5	5788517	2223665.75

Type	1 Tread	2 Thread
Initial write	1487032.5	1556428.3
Rewrite	2490297	3085001.8
Read	5949692.5	6979368
Re-read	5801810	7056192.5
Random read	4992519.5	5788517
Random write	2132894	2223665.8

Memory and CPU benchmarks are also executed but was not able to fill in with graphs. Kindly find their respective results in the respective benchmark folders.