Performance Report

Benchmarking the Amazon EC2 instance.

Pavankumar Shetty
CWID-A20354961
CS-553 Spring 2016
Main Campus

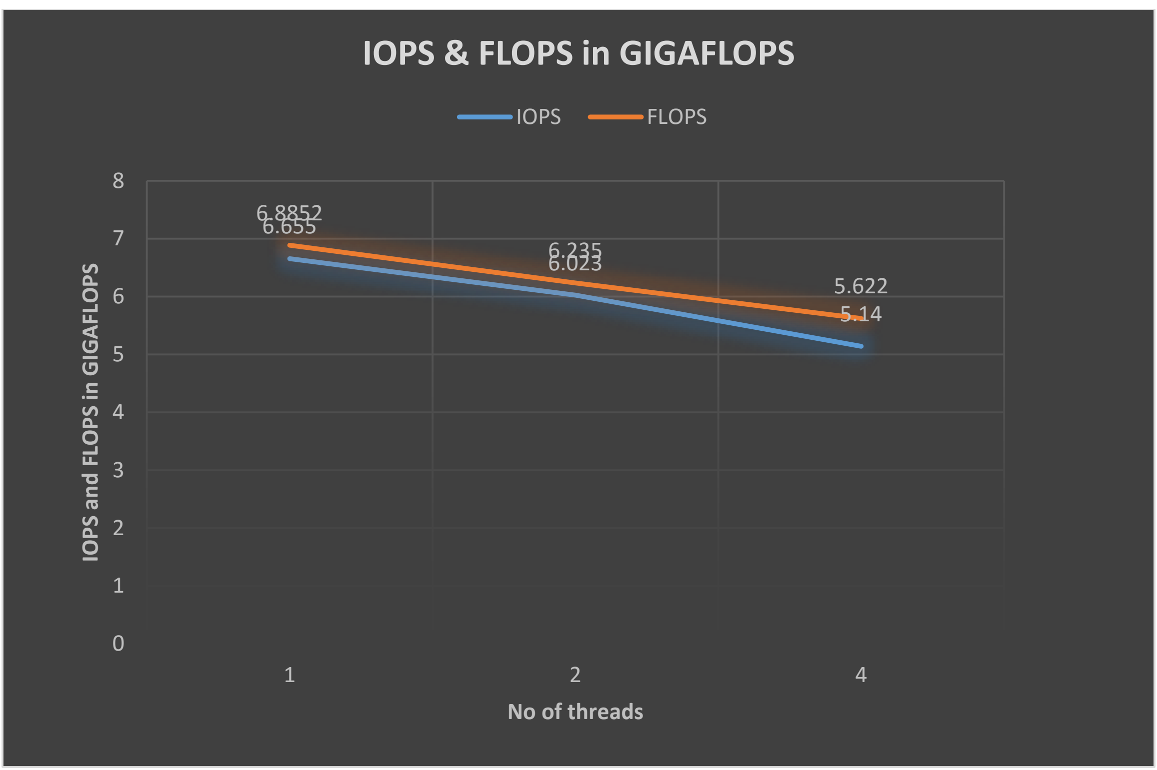
CPU:

Performed experiments to benchmark the CPU of the amazon EC2 t2. micro instances. Designed in C++.
FLOPS I could achieve from 6 to 4 GigaFlops when varying threads from 1 2 and 4. It's been observed that as more number of threads used, FLOPS were found to be reducing and found to be fluctuating over the time.

IOPS:

Input/output operations per second.
IOPS I could achieve from 5 to 4 GigaFlops when varying threads from 1 2 and 4. It's been observed that as more number of threads used, IOPS were found to be reducing.

GRAPH:



X-axis: No of threads
Y- axis: IOPS and FLOPS in GIGAFLOPS

IOPS & FLOPS

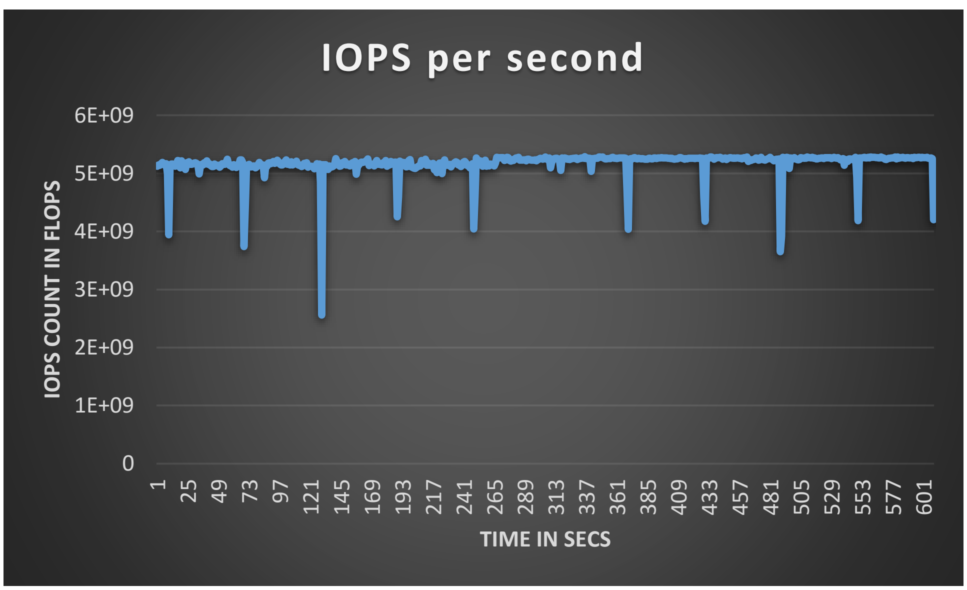
No of threads	IOPS	FLOPS
1	6.655	6.8852
2	6.023	6.235
4	5.14	5.622

---

600 Sample IOPS:

As a separate experiment, ran the benchmark on IOPS and 4 threads for a 10-minute period using a separate thread just to keep track of the number of operations achieved per second and saved it accordingly to a file and plotted a graph of these 600 samples. The values have been consistent and there were few drops in between. This can be inferred from the graph attached below.

GRAPH:



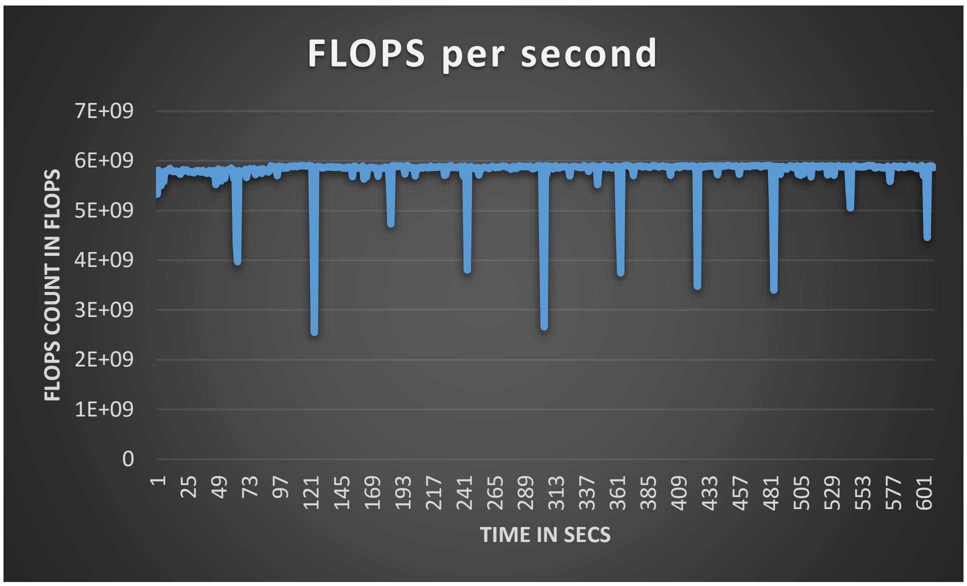
X-axis: TIME in secs

Y- axis: IOPS Count in FLOP per second

600 Sample FLOPS:

As a separate experiment, ran the benchmark on FLOPS and 4 threads for a 10-minute period using a separate thread just to keep track of the number of operations achieved per second and saved it accordingly to a file and plotted a graph of these 600 samples. The values have been consistent and there were few drops in between. This can be inferred from the graph attached below.

GRAPH:



X-axis: TIME in secs

Y- axis: FLOPS Count in FLOP per second

Theoretical Performance:

CPU speed = No of cores * Instructions per Cycle * Clock speed

Speed= 1*4*2.5GHz=**10 GFLOPS**

MEMORY:

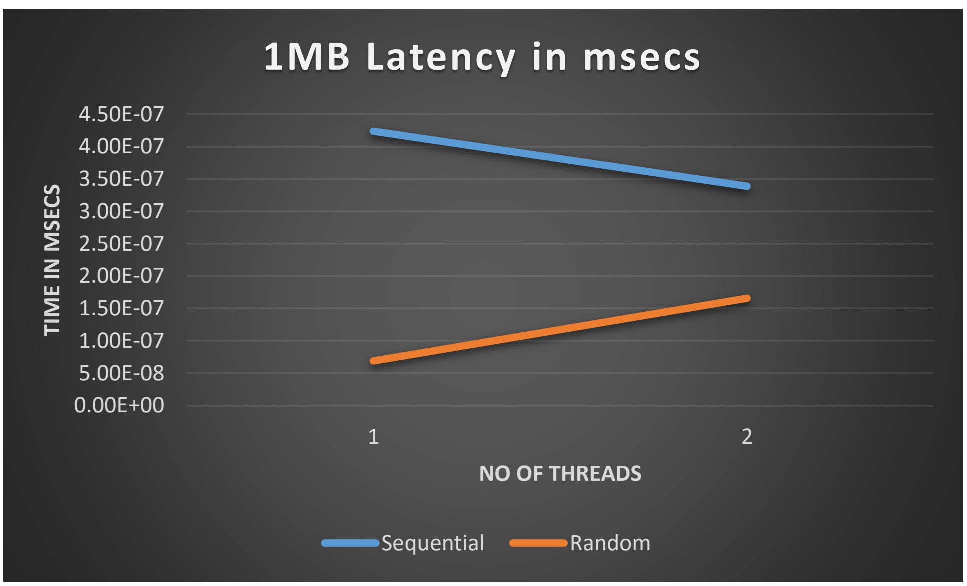
Performed experiments to benchmark the MEMORY of the amazon EC2 t2. micro instances. Designed in C++.

SEQUENTIAL and RANDOM (read + write):

This experiment emphasizes on accessing the memory sequentially, I mean copying the content from source buffer to a destination buffer and also randomly copying the value from a random position of the source buffer, using a customized random number generated. Used memcpy operation to achieve the same. Calculated throughput and Latency for every block size (1B, 1KB, 1MB) transferred with 1, 2 threads and noted down the results and the graph has been plotted for each scenario.

1MB BLOCK SIZE SEQ and RAND access
GRAPH:

a. **Latency in milli secs**



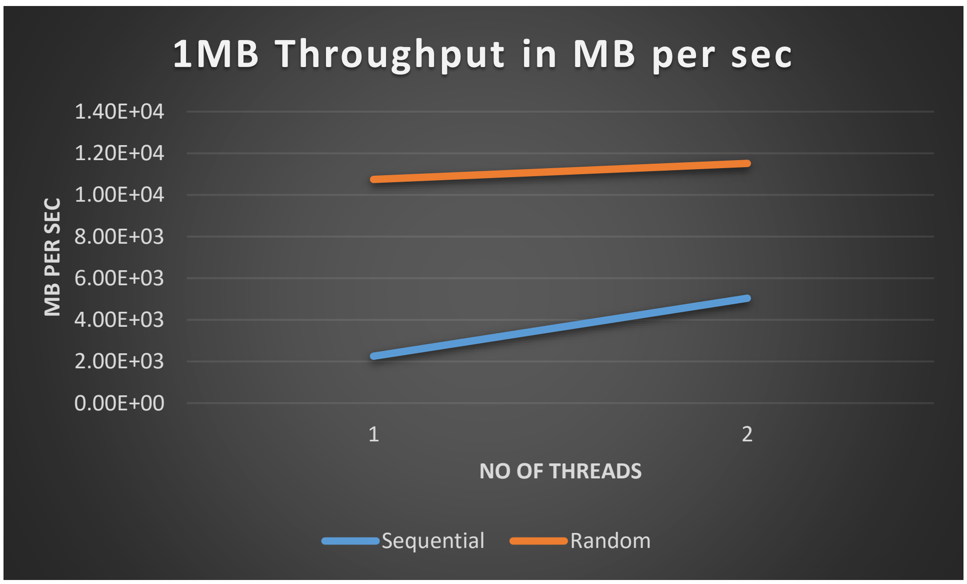
X-axis: No of threads

Y- axis: Time in msecs

Memory 1MB Latency(read+write)

No of threads	Sequential	Random
1	4.24E-07	6.87E-08
2	3.39E-07	1.66E-07

b. Throughput in MB per second



X-axis: No of threads

Y- axis: MB per second

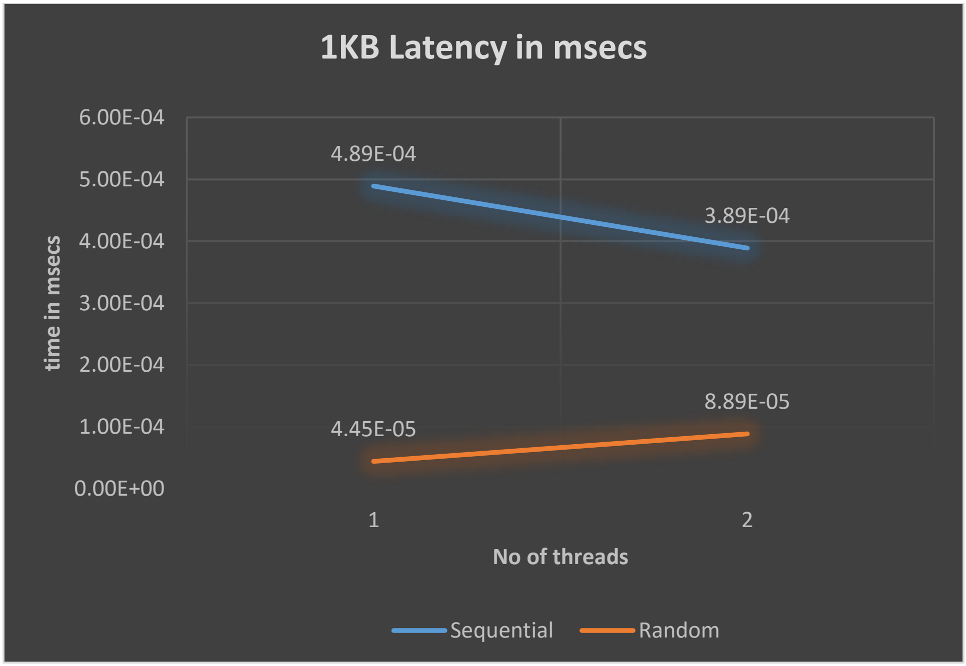
Memory 1MB Throughput(read+write)

No of threads	Sequential	Random
1	2.25E+03	1.07E+04
2	5.03E+03	1.15E+04

1KB BLOCK SIZE SEQ and RAND access

GRAPH:

a. **Latency in milli secs**



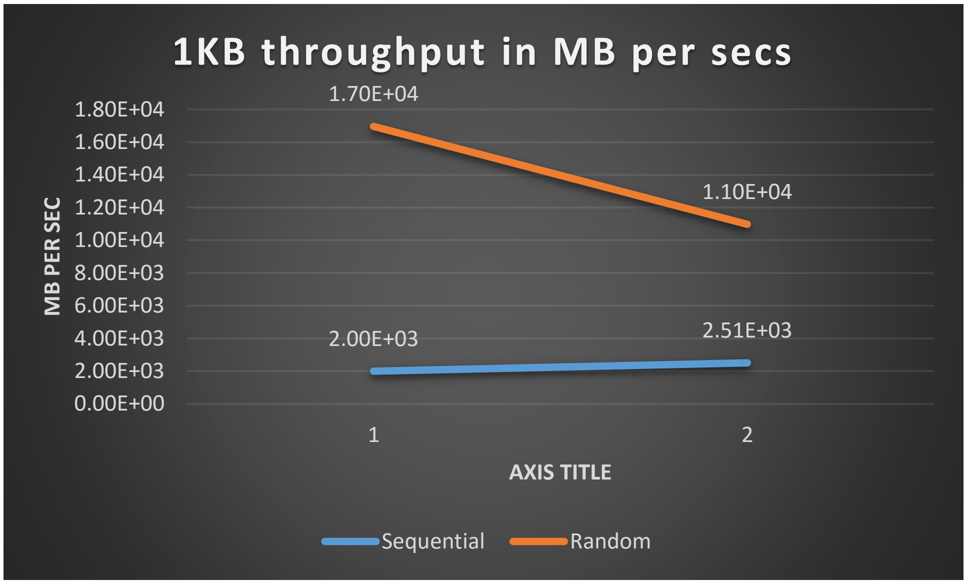
X-axis: No of threads

Y- axis: Time in msecs

Memory 1KB Latency(read+write)

No of threads	Sequential	Random
1	4.89E-04	4.45E-05
2	3.89E-04	8.89E-05

b. Throughput in MB per second



X-axis: No of threads

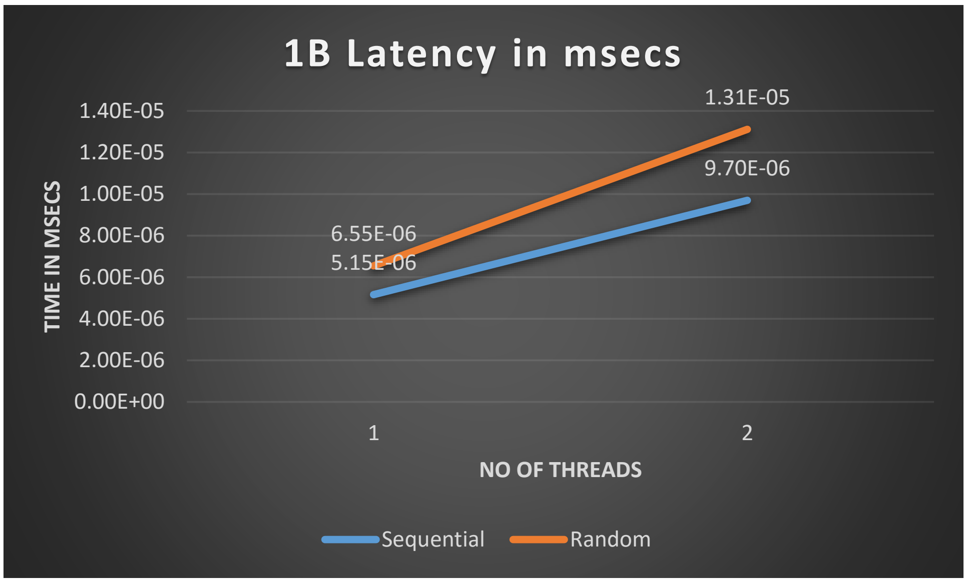
Y- axis: MB per second

Memory 1KB Throughput(read+write)		
No of threads	**Sequential**	**Random**
1	2.00E+03	1.70E+04
2	2.51E+03	1.10E+04

1B BLOCK SIZE SEQ and RAND access

GRAPH:

a. **Latency in milli secs**



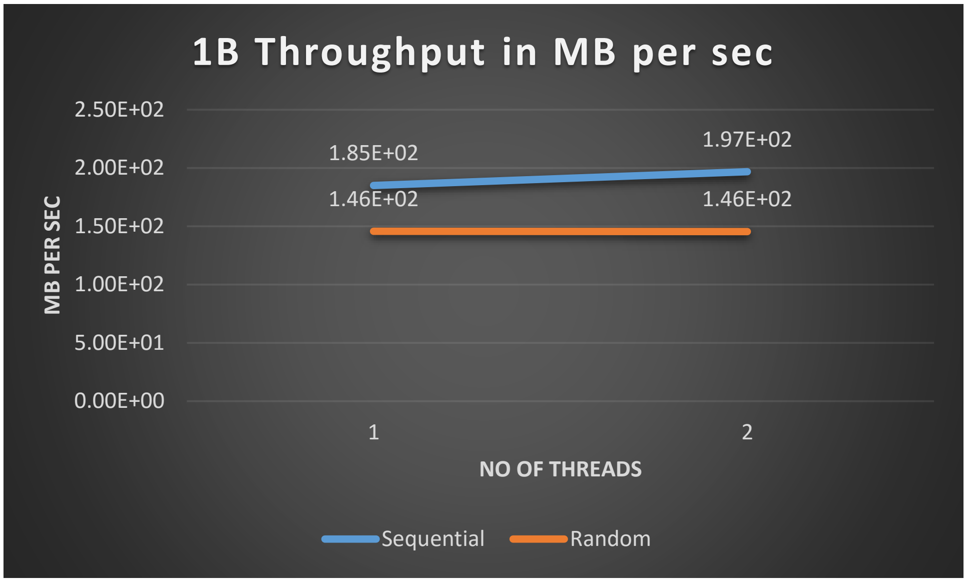
X-axis: No of threads

Y- axis: Time in msecs

Memory 1B Latency(read+write)

No of threads	Sequential	Random
1	5.15E-06	6.55E-06
2	9.70E-06	1.31E-05

b. Throughput in MB per second



X-axis: No of threads

Y- axis: MB per second

Memory 1B Throughput(read+write)		
No of threads	**Sequential**	**Random**
1	1.85E+02	1.46E+02
2	1.97E+02	1.46E+02

With the graphs above, we can infer the data copying sequentially is much quicker than reading and writing it randomly. And also the **random function** takes more amount of time when asked to position itself in the memory and copy the bytes to and fro memory.

Theoretical Performance:

Memory Bandwidth = Data transfer per clock * Clock frequency * mem-bus width * Number of Interfaces

=> 1*1200*64*2 => **19.2 GBPS.**

DISK:

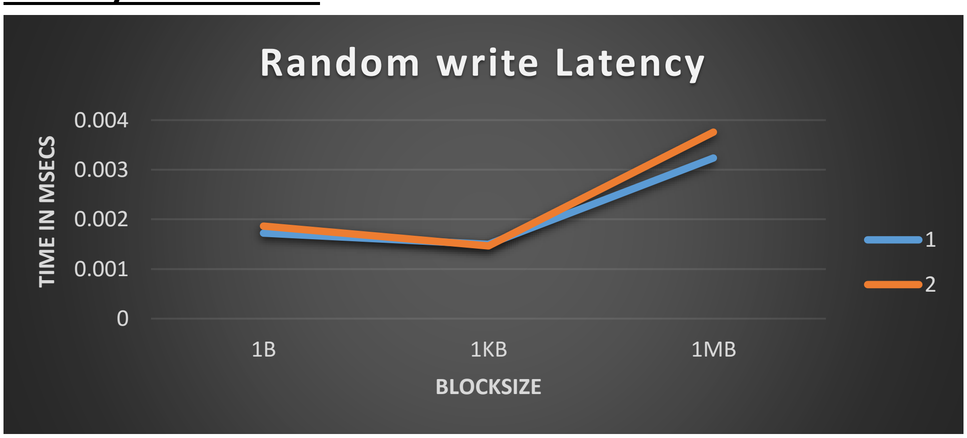
Performed experiments to benchmark the DISK of the amazon EC2 t2. micro instances. Designed in JAVA.

SEQUENTIAL and RANDOM (read + write):

This experiment emphasizes on reading and writing to disk memory sequentially (Sequential Access), I mean reading the content from file and also writing it to the other file sequentially and also randomly copying the content from a random position of the file, using a customized random number generated (Random Access). Calculated throughput and Latency for every block size (1B, 1KB, 1MB) transferred with 1, 2 threads and noted down the results and the graph has been plotted for each scenario.

<u>RANDOM WRITE access with varying block sizes(1B,1KB,1MB) and no of threads 1,2ss</u>
<u>GRAPH:</u>

a. **<u>Latency in milli secs</u>**



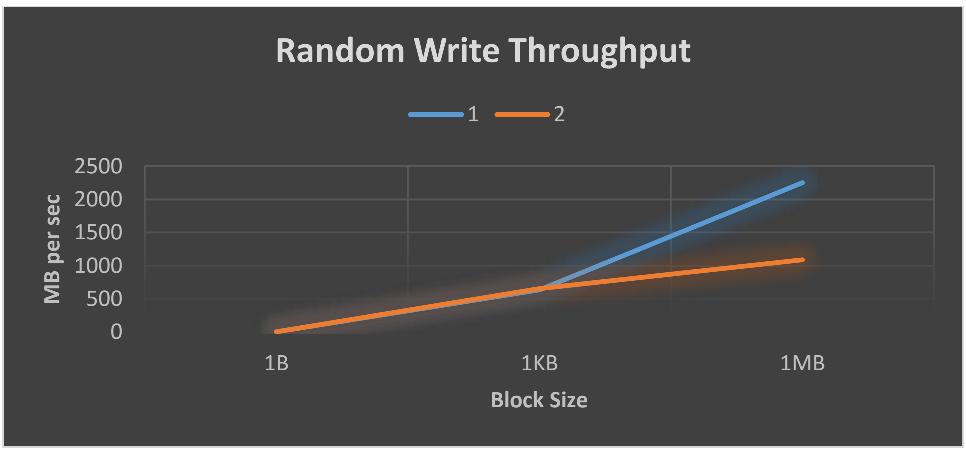
<u>X-axis:</u> block sizes
<u>Y- axis:</u> Time in msecs
Blue line represents thread count #1

Orange line represents thread count #2

Random Write Latency in msecs		**Block size**	
No of threads	**1B**	**1KB**	**1MB**
1	0.00172335	1.50E-03	3.24E-03
2	0.00186297	1.47E-03	3.76E-03

---

b. <u>Throughput in MB per second</u>



<u>X-axis:</u> block sizes

<u>Y- axis:</u> Time in msecs

Blue line represents thread count #1

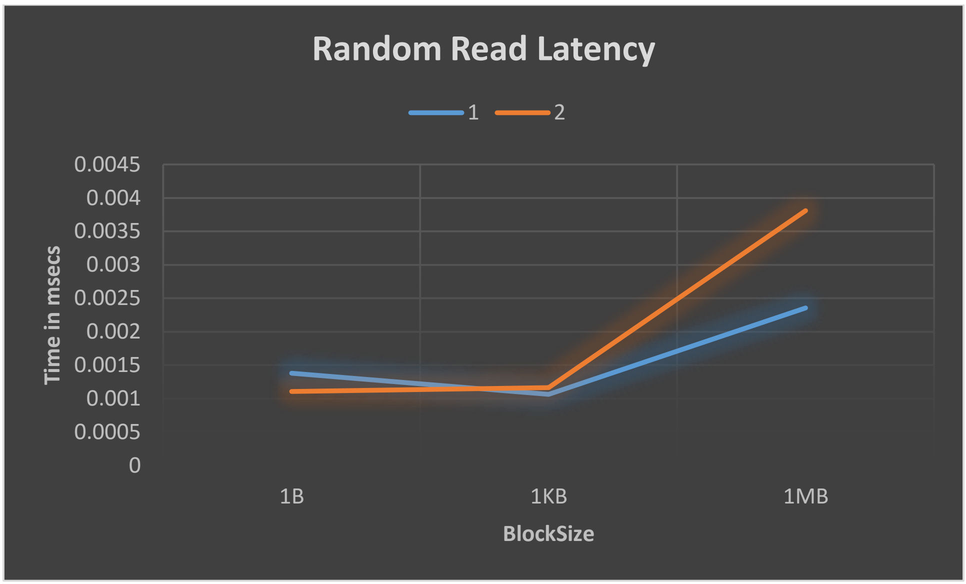
Orange line represents thread count #2

Random Write Throughput in mbps		Block size	
No of threads	1B	1KB	1MB
1	0.553384	636.3007	2249.8757
2	0.53921316	653.21628	1089.3235

RANDOM READ access with varying block sizes(1B,1KB,1MB) and no of threads 1,2

GRAPH:

a. **Latency in milli secs**



X-axis: block sizes

Y- axis: Time in msecs

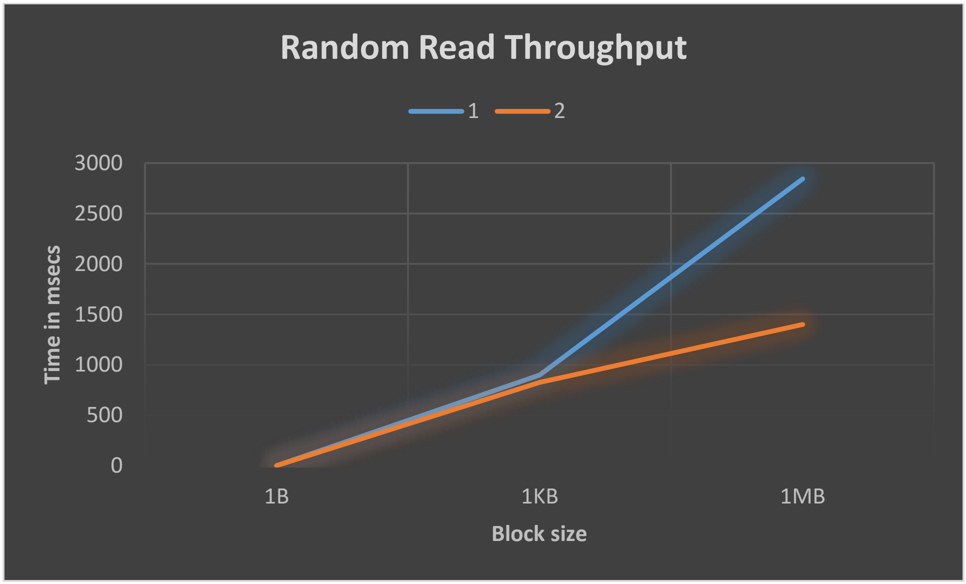
Blue line represents thread count #1

Orange line represents thread count #2

Random Read Latency in msecs

		Block size	
No of threads	**1B**	**1KB**	**1MB**
1	0.00137692	1.06E-03	2.35E-03
2	0.00110668	1.17E-03	3.81E-03

b. Throughput in MB per second



X-axis: block sizes

Y- axis: Time in msecs

Blue line represents thread count #1

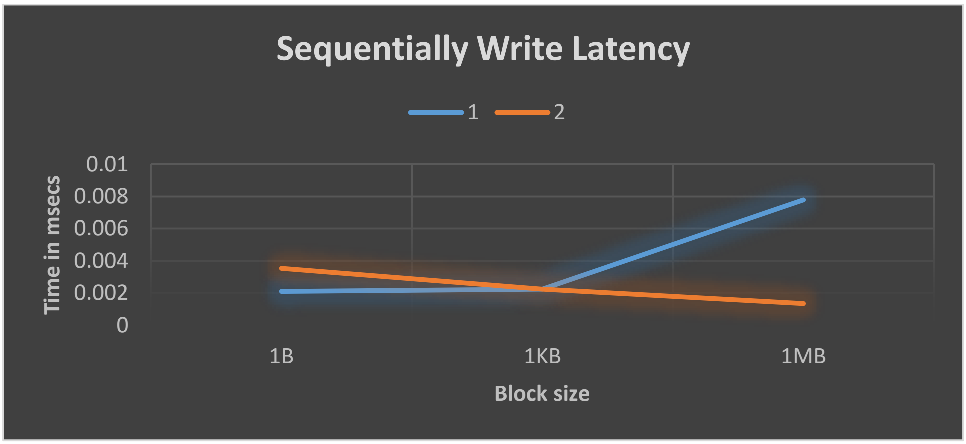
Orange line represents thread count #2

Random Read Throughput in msecs			
		Block size	
No of threads	**1B**	**1KB**	**1MB**
1	0.69261418	8.97E+02	2.84E+03
2	0.8670573	8.26E+02	1401.3412

SEQUENTIAL WRITE access with varying block sizes(1B,1KB,1MB) and no of threads 1,2

GRAPH:

a. **Latency in milli secs**



X-axis: block sizes

Y- axis: Time in msecs

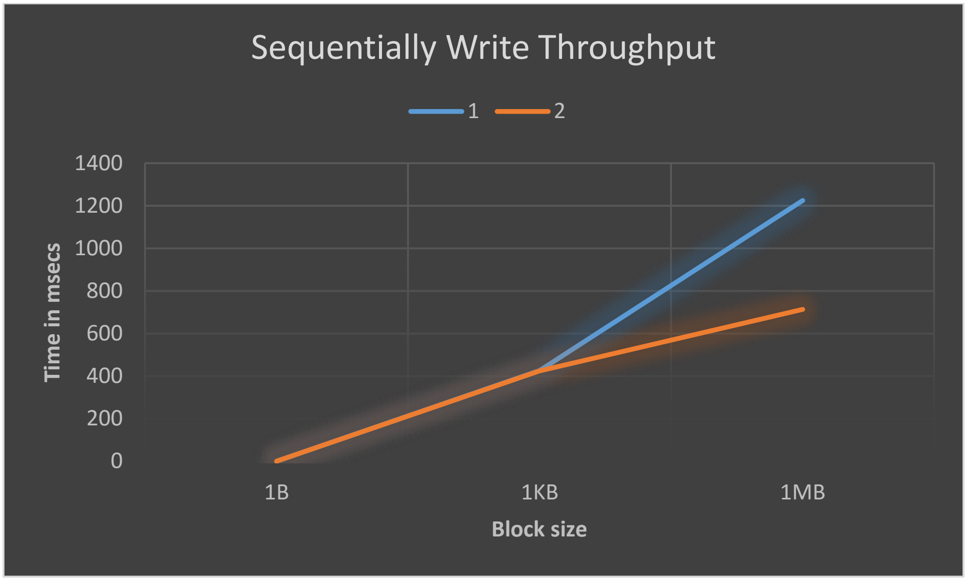
Blue line represents thread count #1

Orange line represents thread count #2

Sequentially Write Latency in msecs

		Block size	
No of threads	**1B**	**1KB**	**1MB**
1	0.00210588	2.24E-03	7.78E-03
2	0.00353461	2.25E-03	1.35E-03

b. Throughput in MB per second



X-axis: block sizes

Y- axis: Time in msecs

Blue line represents thread count #1

Orange line represents thread count #2

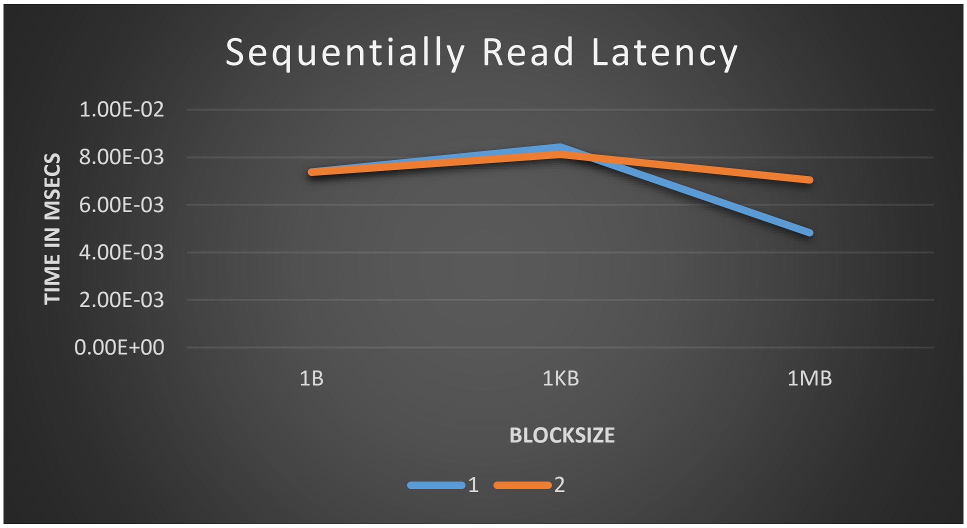
Sequentially Write Throughput in mbps

		Block size	
No of threads	**1B**	**1KB**	**1MB**
1	0.45286261	425.610155	1225.4325
2	0.26766495	424.229203	713.45287

SEQUENTIAL READ access with varying block sizes(1B,1KB,1MB) and no of threads 1,2

GRAPH:

a. **Latency in milli secs**



X-axis: block sizes

Y- axis: Time in msecs

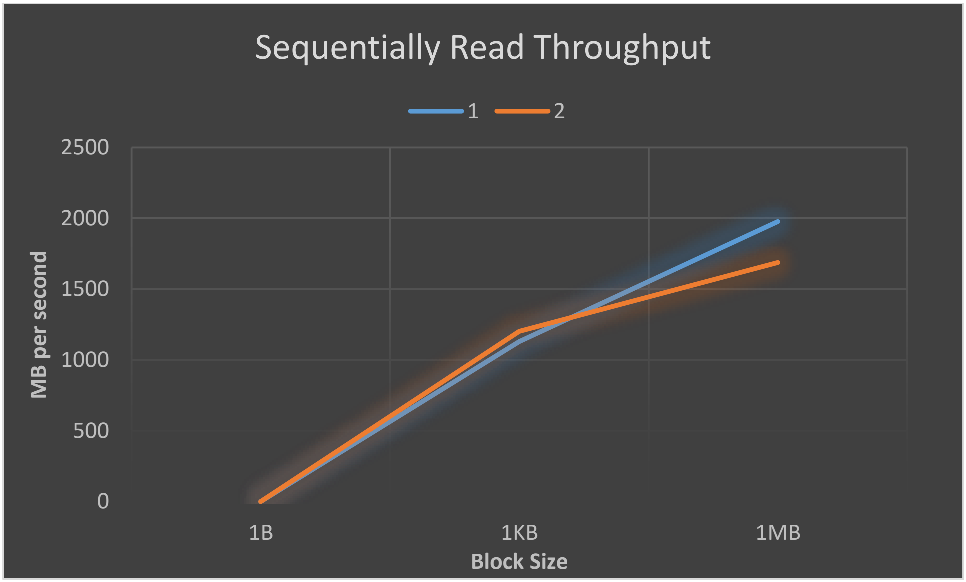
Blue line represents thread count #1

Orange line represents thread count #2

Sequentially Read Latency in msecs

		Block size	
No of threads	**1B**	**1KB**	**1MB**
1	7.38E-03	8.44E-03	4.82E-03
2	0.00737555	8.13E-03	7.05E-03

b. **Throughput in MB per second**



X-axis: block sizes

Y- axis: Time in msecs

Blue line represents thread count #1

Orange line represents thread count #2

Sequentially Read Throughput in msecs

		Block size	
No of threads	**1B**	**1KB**	**1MB**
1	1.29253936	1.13E+03	1.98E+03
2	1.2889	1.20E+03	1687.2469

With the graphs above, we can infer the data access happening sequentially is much quicker than accessing it randomly. Disk hardware may be a reason for this as well. And also the **seek operation** takes more amount of time when asked to position itself in the disk memory.

Disk's Theoretical Performance:

Model number	WD5000BPVX
Interface	SATA 6 GBPS
Data transfer rates	
Interface speed	6 GBPS
Internal transfer rate (max)	144 MBPS
Average latency (millisecs)	5.5
Rotational speed (RPM)	5400
Cache (MB)	8

EXTRA CREDITS:

1. Linpack:

```
[ec2-user@ip-172-31-51-19 linpack]$ ./runme_xeon64
This is a SAMPLE run script for SMP LINPACK. Change it to reflect
the correct number of CPUs/threads, problem input files, etc..
Sat Feb 13 02:02:26 UTC 2016
Intel(R) Optimized LINPACK Benchmark data

Current date/time: Sat Feb 13 02:02:26 2016

CPU frequency:    2.992 GHz
Number of CPUs: 1
Number of cores: 1
Number of threads: 1

Parameters are set to:

Number of tests: 15
Number of equations to solve (problem size) : 1000  2000  5000  10000 15000 18000 20000 22000 25000 26000 27000 30000 35000 40000 45000
Leading dimension of array                  : 1000  2000  5008  10000 15000 18008 20016 22008 25000 26000 27000 30000 35000 40000 45000
Number of trials to run                     : 4     2     2     2     2     2     2     2     2     2     1     1     1     1     1
Data alignment value (in Kbytes)            : 4     4     4     4     4     4     4     4     4     4     4     1     1     1     1

Maximum memory requested that can be used=800204096, at the size=10000

=================== Timing linear equation system solver ===================

Size   LDA    Align. Time(s)    GFlops   Residual     Residual(norm) Check
1000   1000   4      0.025      26.4607  9.632295e-13 3.284860e-02   pass
1000   1000   4      0.025      26.5258  9.632295e-13 3.284860e-02   pass
1000   1000   4      0.025      26.5799  9.632295e-13 3.284860e-02   pass
1000   1000   4      0.025      26.7420  9.632295e-13 3.284860e-02   pass
2000   2000   4      0.185      28.8596  4.746648e-12 4.129002e-02   pass
2000   2000   4      0.189      28.3287  4.746648e-12 4.129002e-02   pass
5000   5008   4      2.461      33.8818  2.651185e-11 3.696863e-02   pass
5000   5008   4      2.467      33.7997  2.651185e-11 3.696863e-02   pass
10000  10000  4      18.180     36.6807  9.014595e-11 3.178637e-02   pass
10000  10000  4      18.683     35.6941  9.014595e-11 3.178637e-02   pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
1000   1000   4       26.5771  26.7420
2000   2000   4       28.5941  28.8596
5000   5008   4       33.8408  33.8818
10000  10000  4       36.1874  36.6807

Residual checks PASSED

End of tests
```

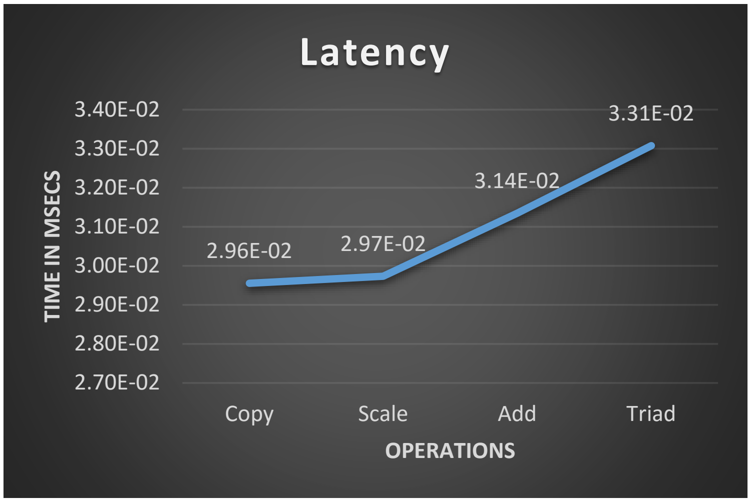
The maximum GFLOPS achieved 36.6807 when this value compared with my theoretical performance, I have achieved 27.7%.

2. Stream:

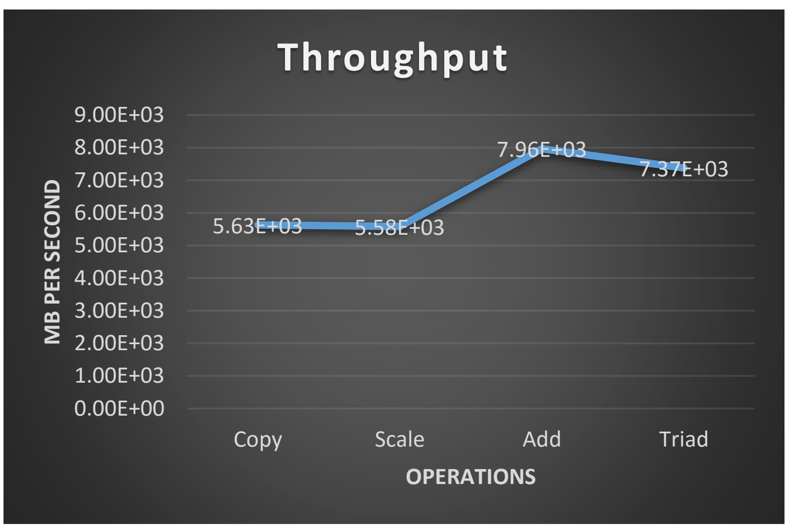
```
[ec2-user@ip-172-31-51-19 ~]$ gcc stream.c -o stream.o
[ec2-user@ip-172-31-51-19 ~]$
[ec2-user@ip-172-31-51-19 ~]$
[ec2-user@ip-172-31-51-19 ~]$
[ec2-user@ip-172-31-51-19 ~]$ ./stream.o
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 28414 microseconds.
   (= 28414 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           5630.2     0.029551     0.028418     0.029945
Scale:          5583.5     0.029736     0.028656     0.030489
Add:            7961.3     0.031350     0.030146     0.032085
Triad:          7372.0     0.033083     0.032556     0.033334
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
[ec2-user@ip-172-31-51-19 ~]$
```

Graphs:

	Latency
Copy	2.96E-02
Scale	2.97E-02
Add	3.14E-02
Triad	3.31E-02



	Throughput
Copy	5.63E+03
Scale	5.58E+03
Add	7.96E+03
Triad	7.37E+03



3. IOZone:

```
[root@ip-172-31-51-19 current]#
[root@ip-172-31-51-19 current]# ./iozone -g# -s 1024
        Iozone: Performance Test of File I/O
                Version $Revision: 3.414 $
                Compiled for 64 bit mode.
                Build: linux-ia64

        Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
                     Al Slater, Scott Rhine, Mike Wisner, Ken Goss
                     Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
                     Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
                     Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
                     Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
                     Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
                     Vangel Bojaxhi, Ben England.

        Run began: Sat Feb 13 03:36:26 2016

        Using maximum file size of 4 kilobytes.
        File size set to 1024 KB
        Command line used: ./iozone -g# -s 1024
        Output is in Kbytes/sec
        Time Resolution = 0.000001 seconds.
        Processor cache size set to 1024 Kbytes.
        Processor cache line size set to 32 bytes.
        File stride size set to 17 * record size.
                                                              random  random    bkwd   record   stride
              KB  reclen   write rewrite    read    reread    read   write    read  rewrite     read   fwrite frewrite   fread  freread
            1024       4 1695065 4304412  7755368 12828238 9569770 4215688 9220512  5758828  9464330  4178773  416257311018226 11773301

iozone test complete.
[root@ip-172-31-51-19 current]#
```