BENCHMARKING

DESIGN DOCUMENT

Shalin A. Chopra

2/12/2016

DESIGN DOCUMENT

1. CPU BENCHMARKING:

- The code for CPU benchmarking is written in C programming language.
- This program calculates the Integer and Floating point operations, in terms of GIOPS and GFLOPS. The main aim is to utilize the complete CPU cycles by executing different arithmetic instructions. Utilizing the CPU's Floating point unit (FPU) completely so that it gives us the maximum FLOPS.
- The program has performs 17 operations which run for 1 Billion times in a loop. The instructions like addition, multiplication, subtraction and division are performed, condition check are accounted as these operations. The function intBM() calculates the GIOPS and floatBM() calculates the GFLOPS.
- There are a maximum of 4 threads, that are executed for each of these functions. All the results are automated using scripts, the final output is directly displayed onto the screen.
- There is a 600 sec performance evaluation, which evaluates the IOPS and FLOPS executed per second, using 4 threads. The entire results are automated the user directly see two files which has values for each second.

2. DISK BENCHMARKING:

- The Disk benchmarking operation is implemented using JAVA programming language.
- The design includes implementation for 3 different block sizes i.e. 1B, 1KB and 1MB each for Sequential and Random operations.
- It implements 4 methods, sequential read & write and random read & write.
- The sequential access is done using a file, and data is read from the file and written into it, in a sequential manner.
- For random access, a random number is generated which lies within the file size, and is seeked to that location onto that file and read and write operations are performed.
- There are 1, 2 threads used to account for concurrency, which helps in calculating Throughput and Latency.

3. MEMORY BENCHMARKING:

- This code is written in C programming language.
- For different block sizes i.e. 1 Byte, 1 KB and 1 MB, sequential and random access to the memory is made.

- This code also implements multi threading to achieve concurrency.
- The disk access are made using memcpy() function which is used to perform read and write operation onto the memory.
- This experiment was run on AWS thus not enough memory was present, the program allocated 50 MB of memory using malloc() function. Larger memory resulted in Segmentation Faults.
- The program also calculates the throughput and latency and helps in finding the Speeds of memory access.

4. **<u>NETWORK BENCHMARKING:</u>**

- This benchmark has been implemented using JAVA programming language.
- The benchmarking is done for both TCP as well as UDP protocol. The code is written to be executed on two different instances of AWS.
- This code does the basic packets transmission from Client to Server and back again, while implementing this we find the RTT of the transmission.
- The packets transmitted are of various sizes i.e. 1 Byte, 1KB and 1MB.
- The TCP being reliable and connection oriented requires pre connection setup and accepting of connection between client and server.
- On the other hand UDP being connection less, the packets are sent and received without and pre established connection.
- The benchmarking helps to find the Network Bandwidth and Latency.

<u>Design Trade-offs made:</u>

CPU Benchmark:

Can be made more efficient and utilize the complete cycles, by implementing more complex instruction such as LINPACK which uses Matrix Multiplication and Linear equation solving. So, that more FLOPS/IOPS can be achieved.

Disk Benchmark:

Since this code is implemented and executed on t2.micro instance which is a single core instance and doesn’t always provide with all the resources available to them as these are shared resources, the data transfer rate is limited by the core. For, future implementation we can extend the design to multi- core environment and achieve better efficiency.

Memory Benchmark:

The memory size was limited and hence most part of it wasn’t accessible to run the benchmark. The benchmark is implemented for 50 MB memory allocation. More memory can be allocated and the cache’s can be isolated so as to achieve better performance.

BENCHMARKING

DESIGN DOCUMENT

Shalin A. Chopra

2/12/2016

PERFORMANCE EVALUATION

This document presents with the performance evaluation for 4 different Benchmarks.

I have evaluated the following benchmarks:

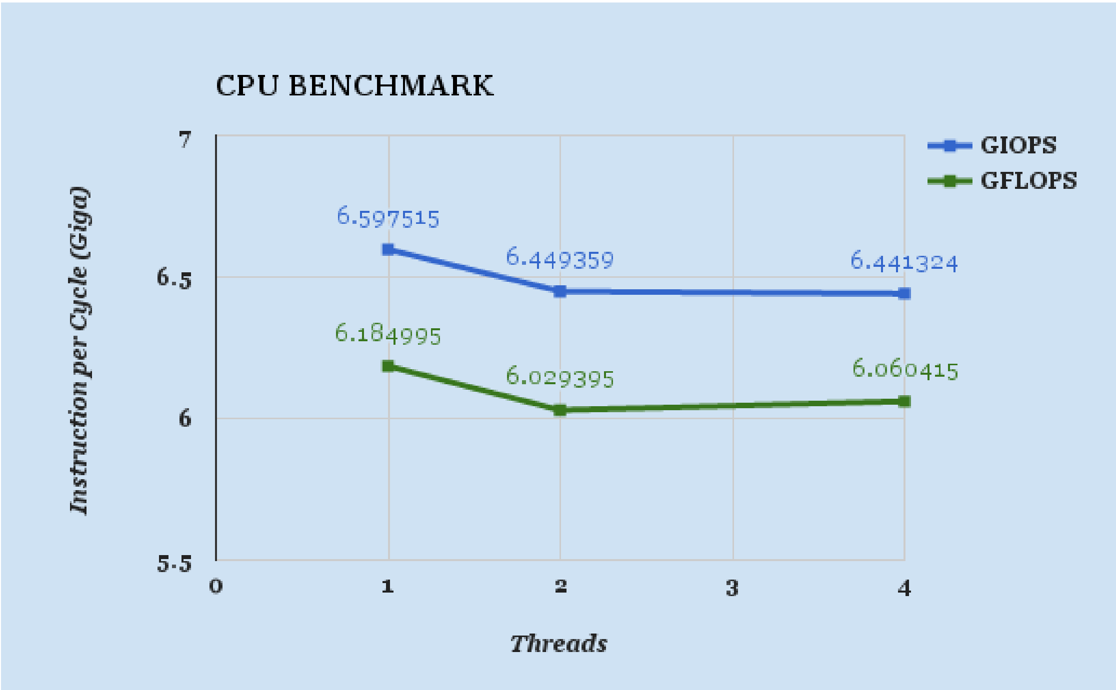
1. **CPU**
2. **DISK**
3. **MEMORY**
4. **NETWORK**

1. CPU BENCHMARKING

Performance is done on AWS EC2 instance i.e. t2.micro, linux version: Ubuntu
The specifications are as follows:

SPECIFICATIONS:	
Model Name	Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz
CPU MHz	2500.056
Cache Size	25600 KB
CPU Cores (vCore)	1
RAM	1 GB
Memory	8 GB

The graphs & tables below show the performance of benchmarking:



The above graph depicts the Average GIOPS and GFLOPS for 1, 2, 4 threads.
GIOPS for Integer operation and GFLOPS for Floating point operations.

As seen in the graph, as the number of thread increased the Number of Instruction per cycle started to decrease. Maximum FLOPS and IOPS were obtained for 1 thread. If we run the experiment for more threads, we see the trend that the Instructions per cycle start to stabilize at a point. It is not always the case that, more the number of threads the better the performance is.

Average & Standard Deviation for 3 Experiments:

Average:

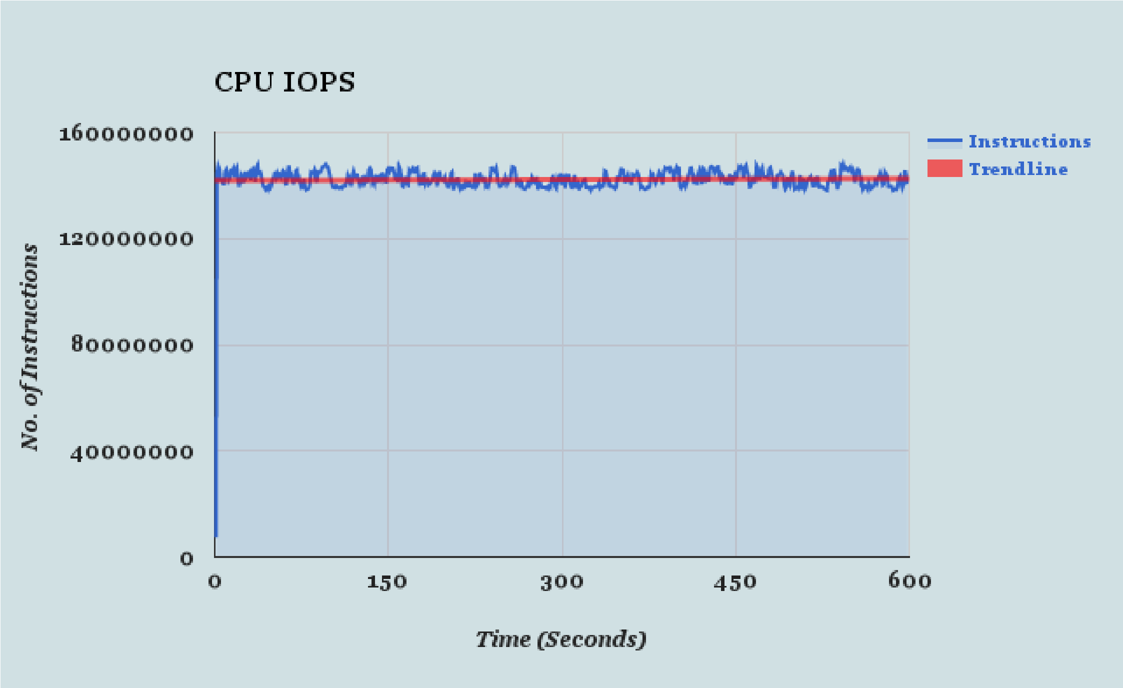
No. of Threads	Average GIOPS	Average GFLOPS
1	6.597515	6.184995
2	6.449359	6.029395
4	6.441324	6.060415

Standard Deviation:

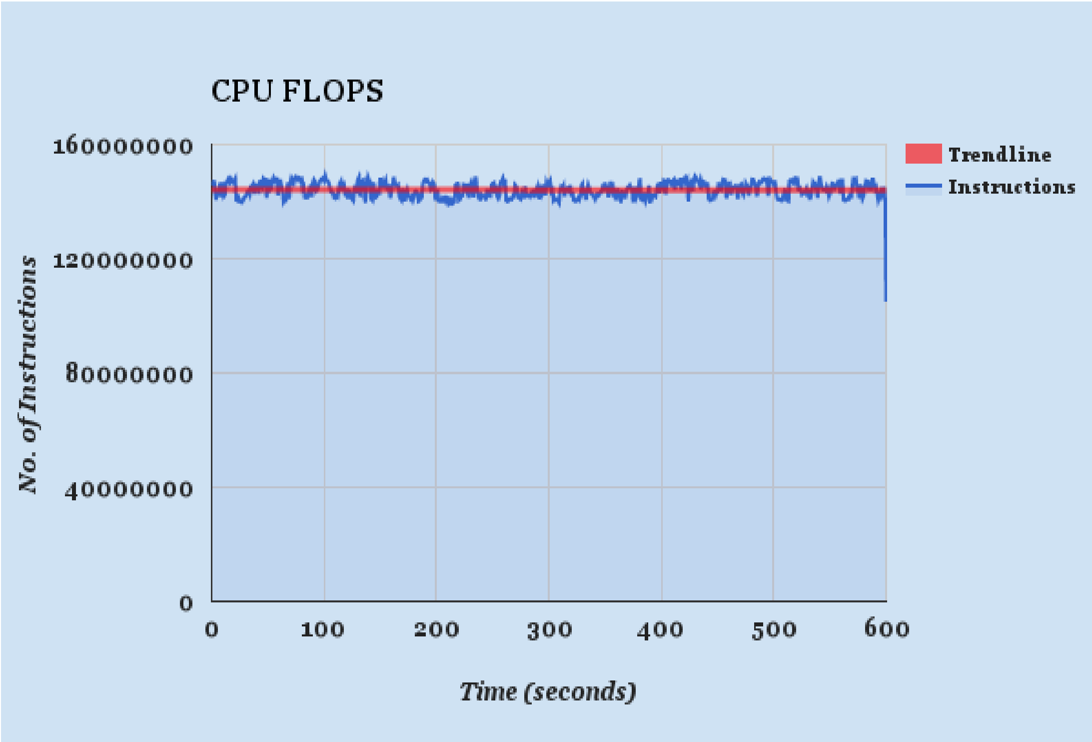
No. of Threads	Standard Deviation
1	0.130069
2	0.048843
4	0.01059

Plotted Values for 600 sec experiment:

Integer Operations per second:



<u>Floating point Operations per second:</u>



The above graphs depicts the number of instructions (operations) executed per second, for 600 seconds having 4 threads running concurrently. We observe that there is a variation for every second, the number of operations doesn't move at a constant rate. This might be due to the CPU prioritization. The Trendline in (red) shows what expected result was.

Theoretical Performance:
Number of Cores * Number of Instruction per cycle * Clock Speed = 1*4*2.5
= 10 GFLOPS

Compared to theoretical performance efficiency achieved is around = 62.5%

LINPACK BENCHMARK:

I ran the LINPACK Benchmark, at two different time intervals, having different values. For 1st the maximal value obtained is: **36.9150 GFLOPS**
For 2nd the maximal value obtained is: **20.778 GFLOPS**

This variation might be due to AWS's CPU priority and utilization as amazon doesn't provide with 100% CPU utilization of the instances as these are shared, it provides only around 10% utilization.

```
Intel(R) Optimized LINPACK Benchmark data

Current date/time: Sat Feb 13 01:33:58 2016

CPU frequency:    2.825 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      0.037      17.8627  9.632295e-13 3.284860e-02   pass
1000   1000   4      0.025      27.0416  9.632295e-13 3.284860e-02   pass
1000   1000   4      0.025      27.1198  9.632295e-13 3.284860e-02   pass
1000   1000   4      0.025      26.8425  9.632295e-13 3.284860e-02   pass
2000   2000   4      0.187      28.6389  4.746648e-12 4.129002e-02   pass
2000   2000   4      0.187      28.6115  4.746648e-12 4.129002e-02   pass
5000   5008   4      2.455      33.9686  2.651185e-11 3.696863e-02   pass
5000   5008   4      2.439      34.1811  2.651185e-11 3.696863e-02   pass
10000  10000  4      18.143     36.7564  9.014595e-11 3.178637e-02   pass
10000  10000  4      18.065     36.9150  9.014595e-11 3.178637e-02   pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
1000   1000   4       24.7167  27.1198
2000   2000   4       28.6252  28.6389
5000   5008   4       34.0749  34.1811
10000  10000  4       36.8357  36.9150

Residual checks PASSED

End of tests
```

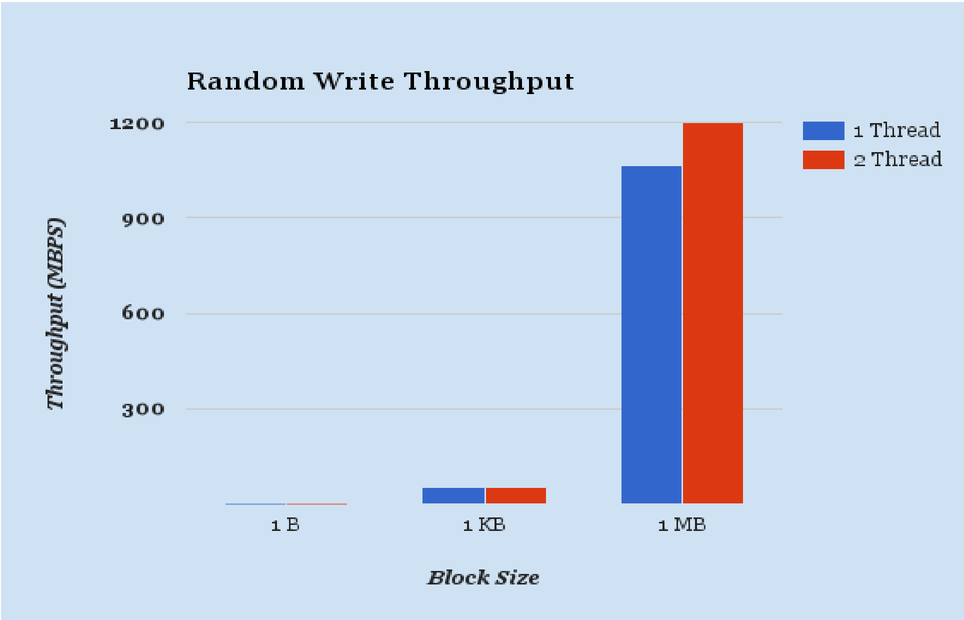
```
the correct number of CPUs/threads, problem input files, etc..
./runme_xeon64: 37: [: -gt: unexpected operator
Sat Feb 13 03:35:29 UTC 2016
Intel(R) Optimized LINPACK Benchmark data

Current date/time: Sat Feb 13 03:35:29 2016

CPU frequency:    2.929 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      0.051      13.1048  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.038      17.4613  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.038      17.6799  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.038      17.7428  9.900691e-13 3.376390e-02   pass
2000   2000   4      0.286      18.6734  4.053480e-12 3.526031e-02   pass
2000   2000   4      0.280      19.0483  4.053480e-12 3.526031e-02   pass
5000   5008   4      4.140      20.1390  2.336047e-11 3.257429e-02   pass
5000   5008   4      4.113      20.2752  2.336047e-11 3.257429e-02   pass
10000  10000  4      32.187     20.7185  1.124127e-10 3.963786e-02   pass
10000  10000  4      32.106     20.7708  1.124127e-10 3.963786e-02   pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
1000   1000   4       16.4972  17.7428
2000   2000   4       18.8609  19.0483
5000   5008   4       20.2071  20.2752
10000  10000  4       20.7446  20.7708

Residual checks PASSED

End of tests

Done: Sat Feb 13 03:36:49 UTC 2016
ubuntu@ip-172-31-25-167:~/l_mklb_p_11.3.1.002/benchmarks_11.3.1/linux/mkl/benchmarks/linpack$
```

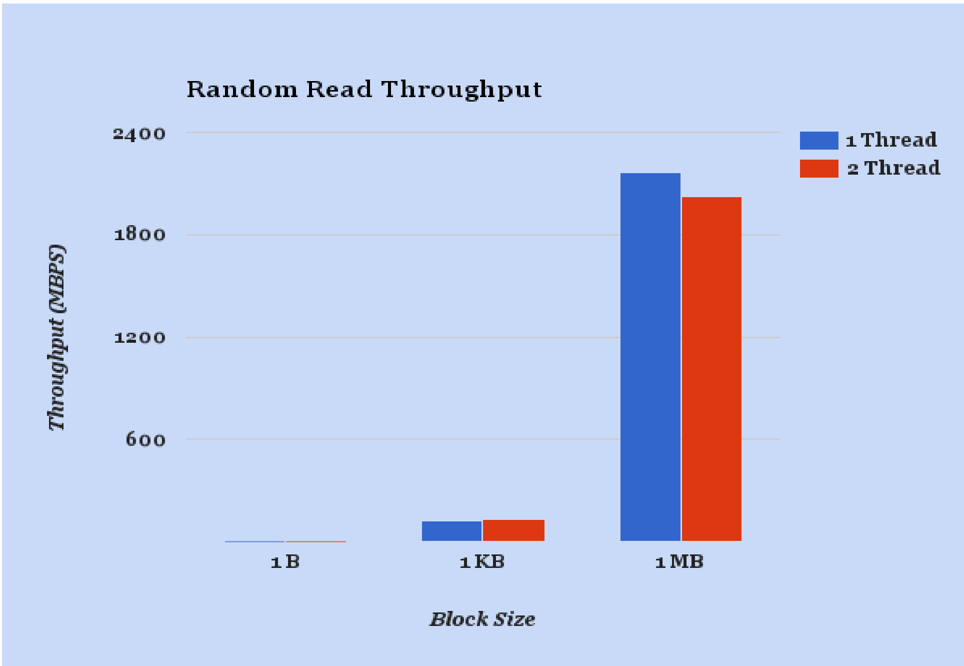
2. DISK BENCHMARKING

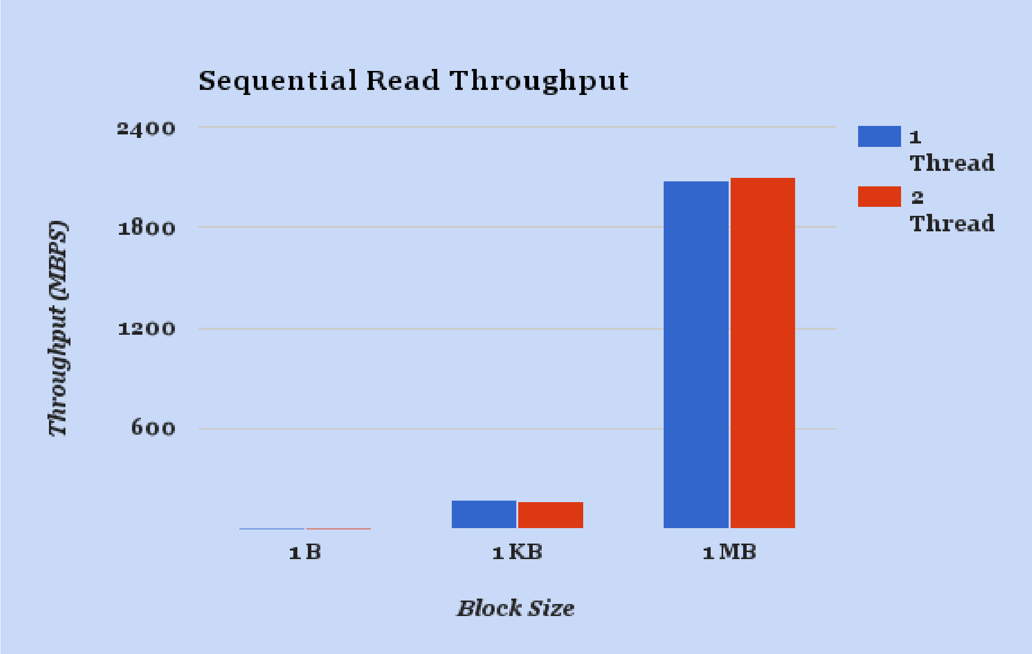
Amazon provides with Disk Space of around 8GB and Cache of size 25600Kbytes. The disk benchmarking is evaluated on the basis of these specifications.

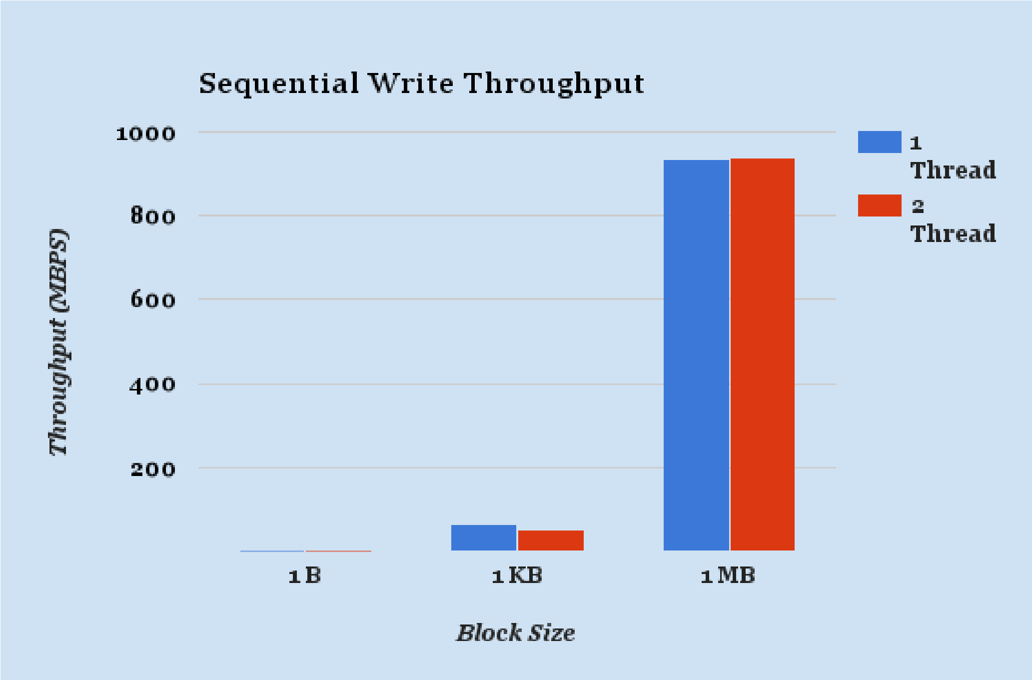
The graphs for Throughput and Latency for, Sequential Read & Write as well as Random Read & Write are shown below:

Throughput:





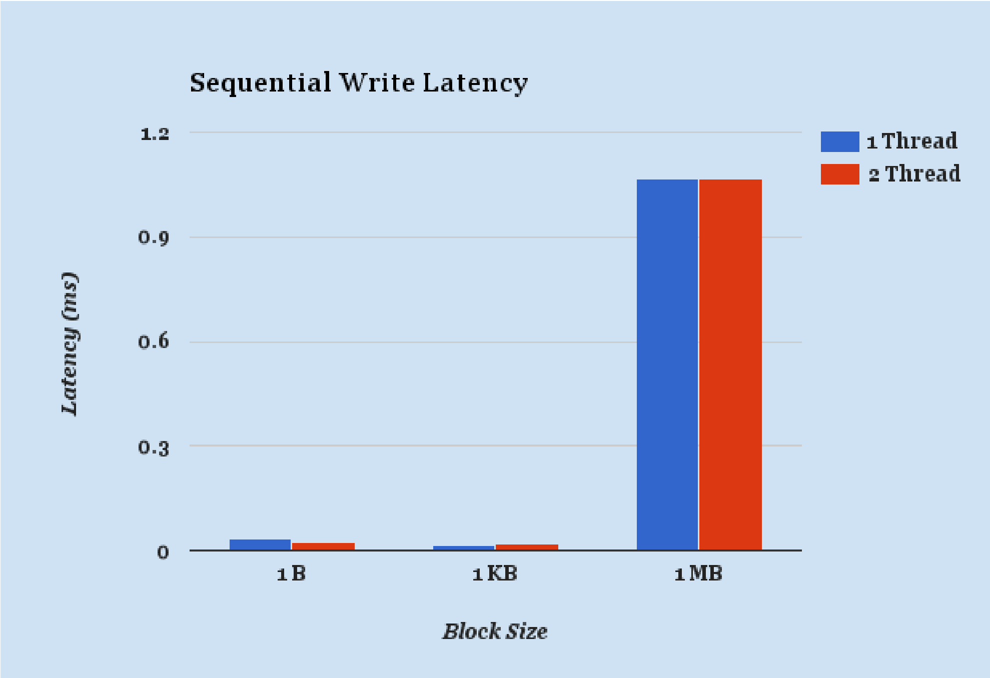


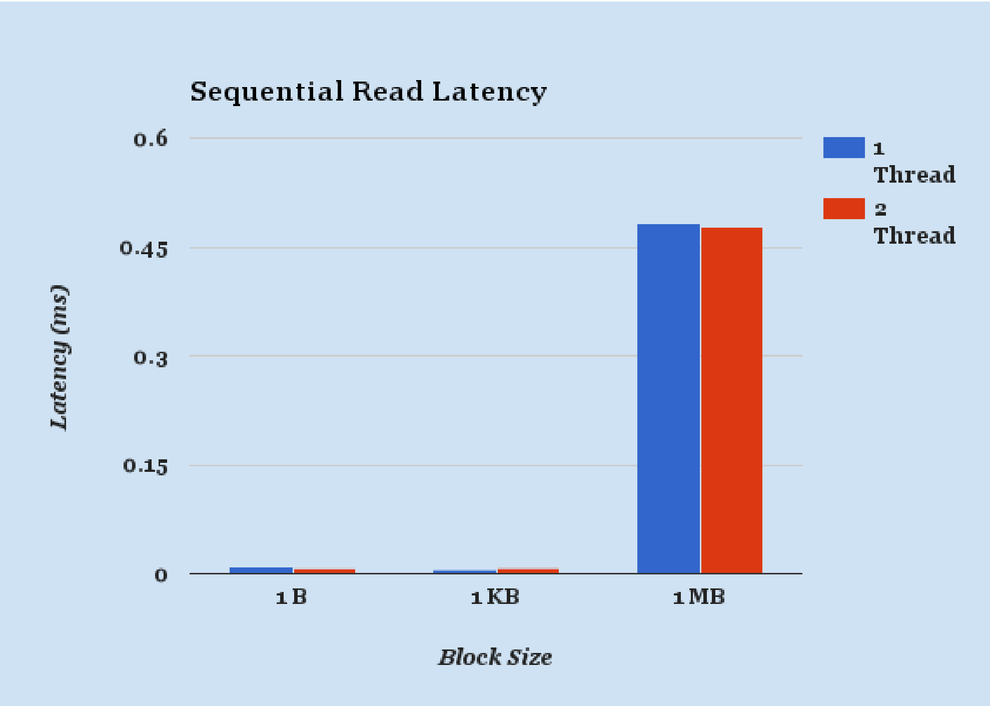


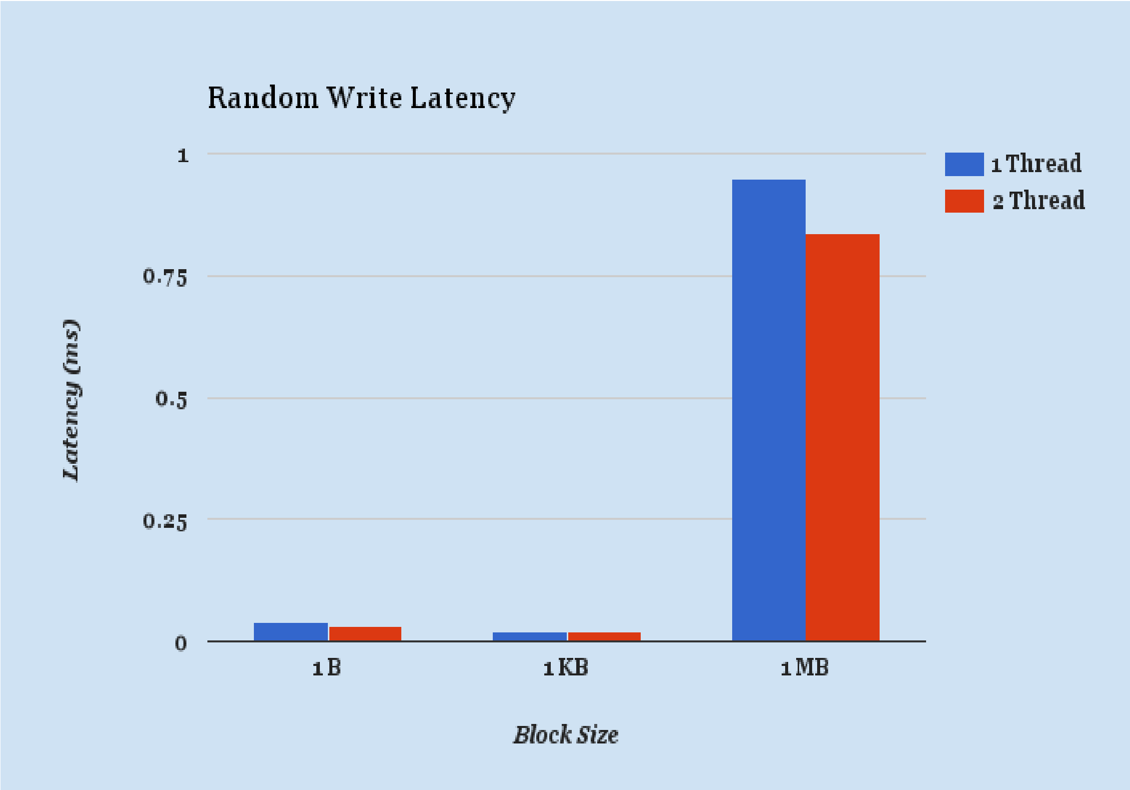
As we increase the number of threads, we can see the Throughput increases; the speeds for Sequential are a bit higher for some cases than Random disk access.

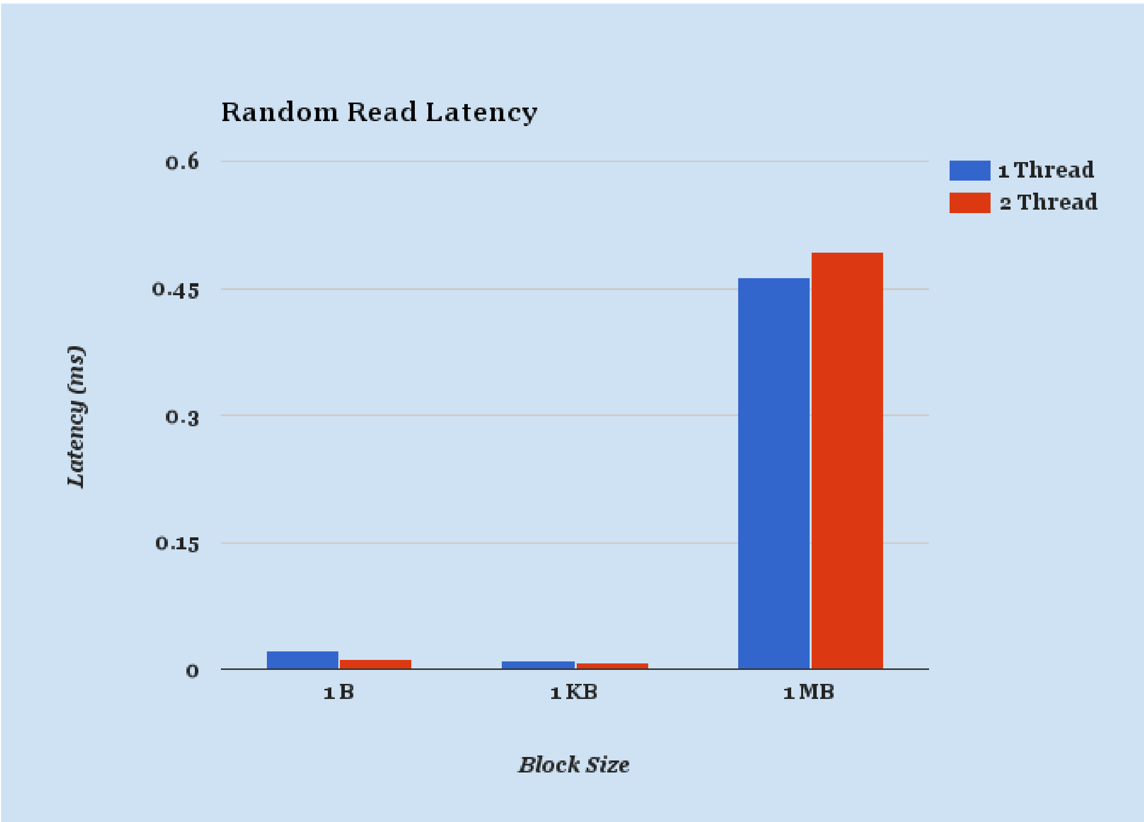
Latency:

As we see the graphs below the latency decreases as we increase the number of threads, as the data to access from disk also increases with the increase in number of threads. The Bar graphs shows and increase in latency, this is because the latencies are calculated for 1B, 1 KB and 1 MB as a whole. The other line graph shows the latencies with common 1Byte factor, for all block sizes.

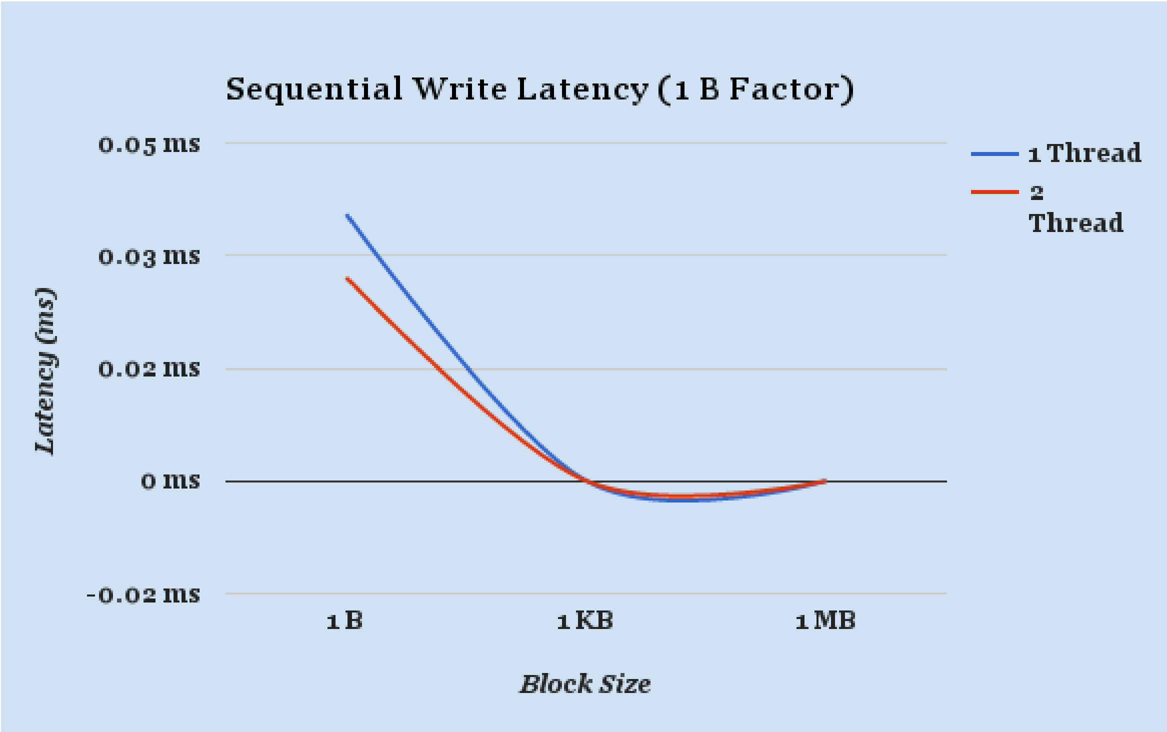


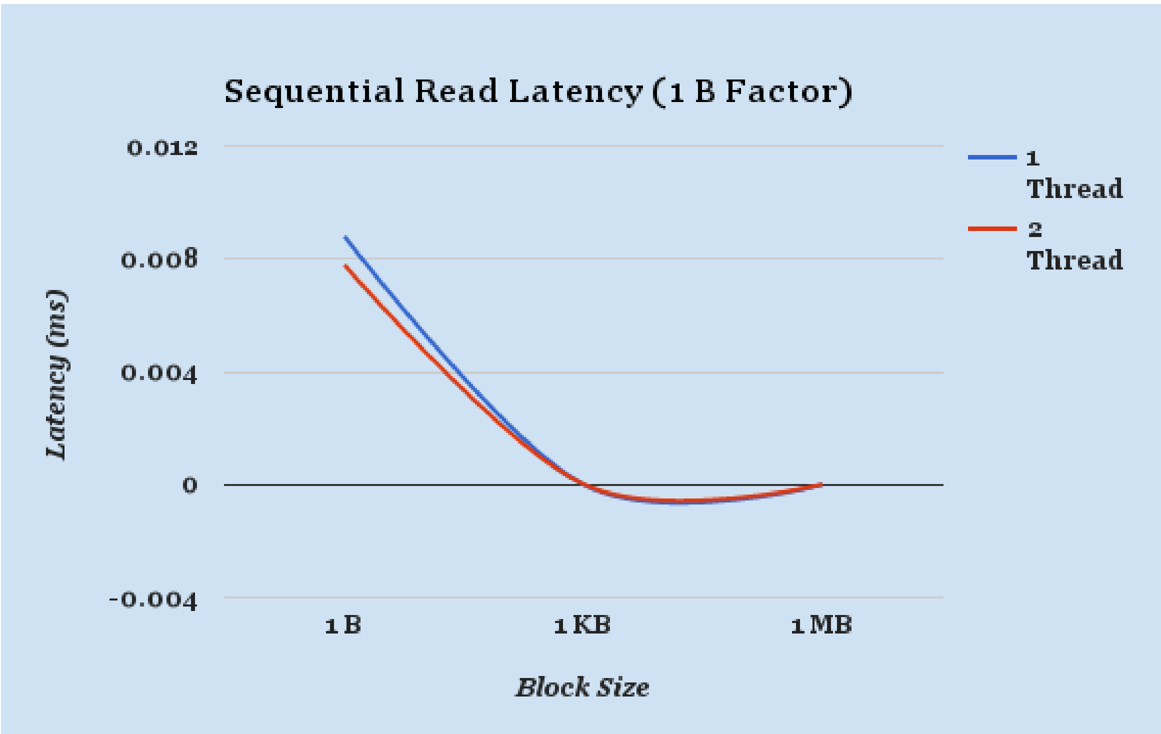


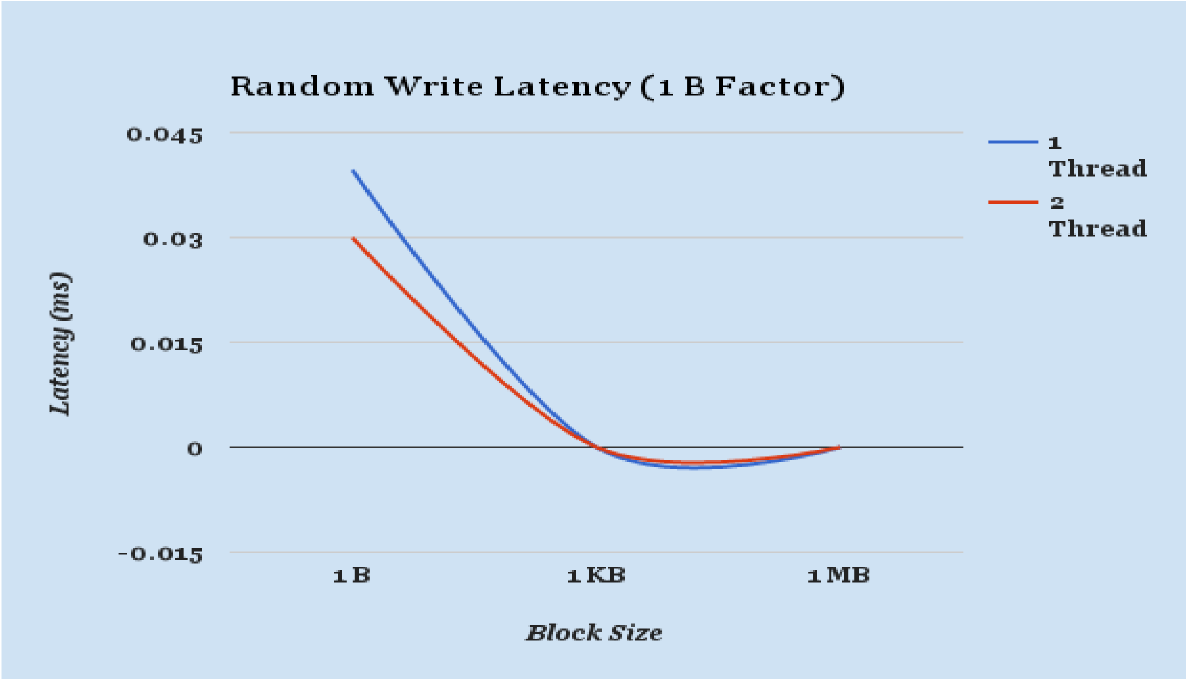

Random Write Latency
1
0.75
0.5
0.25
0
Latency (ms)
1 Thread
2 Thread
1 B
1 KB
1 MB
Block Size

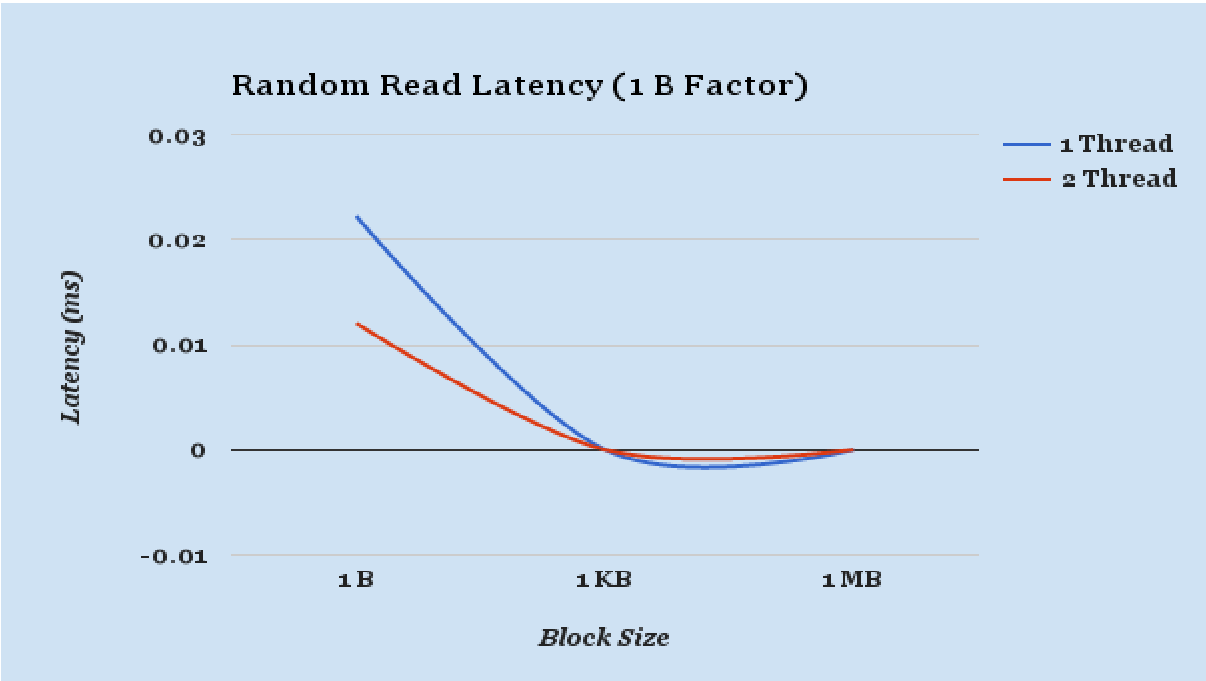

Random Read Latency
0.6
0.45
0.3
0.15
0
Latency (ms)
1 Thread
2 Thread
1 B
1 KB
1 MB
Block Size

These latencies graph are considered with the 1 Byte factor, i.e. all the values are for 1Byte of data transfer, from different Block sizes.









The Average and Standard Deviation for 3 Experiments:

Average (Latency in ms):

Block Size	1 B	1KB	1MB
Seq. Write	**0.035500667**	**0.015134667**	**1.067327**
Seq. Read	**0.008803**	**0.005443333**	**0.481972667**
Ran. Write	**0.03964**	**0.018343**	**0.94679**
Ran. Read	**0.02223**	**0.0093596**	**0.462244**

Average (Throughput in MB/s):

Block Size	1 B	1KB	1MB
Seq. Write	**0.0269241**	**64.54552182**	**934.7162732**
Seq. Read	**0.108385**	**179.455102**	**2077.08631**
Ran. Write	**0.0240529**	**53.25690**	**1066.74833**
Ran. Read	**0.0463590**	**118.6682**	**2161.63714**

Standard Deviation (Latency):

Block Size	Seq. Write	Seq. Read	Ran. Write	Ran. Read
1 B	**0.002081**	**0.0002344**	**0.0002344**	**0.0002173**
1 KB	**0.000332**	**0.0001107**	**0.0001107**	**0.0004130**
1 MB	**0.015442**	**0.0195914**	**0.0195914**	**0.1190715**

Standard Deviation (Throughput):

Block Size	Seq. Write	Seq. Read	Ran. Write	Ran. Read
1 B	**0.0015896**	**0.0028439**	**0.0001315**	**0.01407758**
1 KB	**1.4087563**	**3.6764208**	**1.1944984**	**1.74758713**
1 MB	**12.644688**	**52.153752**	**12.004160**	**52.1278257**

IOZone Benchmarking:

The following screen shots shows the performance of IOZone Benchmarking tool on AWS. The result shows the different Speeds in Kbytes. The 1st screen shows the speeds of all the values which gets run by default when executing IOZone. The 2nd screen shows the values for specific file size provided by us.

```
        Run began: Sat Feb 13 01:45:55 2016

        Auto Mode
        Command line used: ./iozone -a
        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
              64       4  1279447  3541098 10821524 12902017  7100397  3791156  5860307   4564786  10821524  3541098  4018152  6421025  7940539
              64       8  1679761  4018152 15972885 15972885 12902017  4564786  7100397   5283570  22737791  4897948  4274062  9318832  9006179
              64      16  2006158  4564786 15972885 15972885 15972885  4897948  7100397   5389653  10821524  3057153  4564786  7940539 15972885
              64      32  2006158  4274062 15972885 12902017 12902017  4274062  6271021   4897948  15972885  4643754  6421025  9318832  9006179
              64      64  1933893  4564786 20962191 20962191 15972885  4988978  9318832   5389653   9006179  4564786  4564786  8182586 15972885
             128       4  1642783  4012317 11720614 11720614  9795896  4012317  7082197   5122535   7476717  3759450  4012317  9129573 10779307
             128       8  2175277  2066432 16365173 15881078 12542043  4934216  9129573   2409592   5847904  4596273  4557257 10567140 10779307
             128      16  2132084  3759450  6114306 14200794 14200794  5122535  8548124   6114306  12842051  3759450  4596273  4596273  6114306
             128      32  1111981  3867787  5122535 15881078 15881078  5545860  9977956   6406138  10567140  5122535  5847904 11470204 14200794
             128      64  1142750  2326073 18012359 18012359 15881078  5847904  9977956   5847904   5325799  3199360  3895854 12842051 14586625
             128     128  1102844  2367096  7082197  7082197  4717434  5603747  9795896   5545860   4934216  5545860  5784891  4934216 14200794
             256       4  1842366  4132864 11569783 12812277 10245071  4332998  8888172   4477549   8534922  3935921  4197489  9868433 10245071
             256       8  2065659  4734192 14164395 15835569 12812277  4496299  9518507   6719046  12090911  4840911  4819184 12228612 12090911
             256      16  2305128  4819184 15164624 14953435 13454450  5687020 10651598   6554972  15164624  3994490  4819184 12090911 12812277
             256      32  2639446  5320671 14953435 14164395 14164395  5810112 11091721   6554972  15164624  5217259  5217259 11695808 13454450
             256      64  2486631  5540300 14953435 16072608 15164624  5971678 11207494   6719046  13454450  5117791  6213578 13454450 15164624
             256     128  2557711  5455847 12812277 17096249 14164395  6107548 11091721   6595234  12812277  4197489  7735574  5117791 14164395
             256     256  2632974  4132864 13454450 13454450 12812277  5455847 10245071   5687020  11207494  5217259  5569035 11207494 12661199
             512       4  1889050  4195896 11620224 11943357  9510316  4195896  9304292   5566231   8528336  4131319  4091959  9859630 11374040
             512       8  2370799  4522868  7540170 15471149 13438092  5278892 11620224   7410080  11620224  4838792  4375425 13109944 14240069
             512      16  2141475  5331314 14628067 15583417 13872122  5760329 11943357   6319739  12797441  5019764  4961773 13109944 14628067
             512      32  2695115  5699180 14240069 15037801 14240069  6086873 12797441   6999490  15471149  5115422  5164632 13109944 14628067
             512      64  2548017  5807059 15037801 15037801 14628067  6174377 12797441   7022379  14146265  5164632  5566231 13438092 15037801
             512     128  2815243  5566231 13783088 14628067 13872122  6104175 11877300   6632013  12499490  5566231  6035551  9638369 14240069
             512     256  2532990  5398323 12499490 12499490 12146009  5684095 11080601   6246213  11620224  4228947  7309196 11138071 12797441
             512     512  2524058  5331314 13109944 12797441 11374040  5684095 11620224   5495016   7758090  3710197  5951911 11620224 11877300
            1024       4  1858644  4178773 11773301 12492426  9464330  3941039  9855234   5917516   9677584  3891053  4162573 10769573 10134284
            1024       8  2437821  5120336 15295157 16037624 12348754  5018624 13142265   6963241  12037272  5048117  4228138 12790037 14183905
            1024      16  2326879  5175871 13472053 15080341 13818817  5277632 11773301   6963241  12639479  4493556  5018624 13264026 14183905
            1024      32  2744728  5564829 13472053 14618393 14044758  5660167 13472053   6569179  12492426  4920874  4854136 13643232 14668318
            1024      64  2694787  5958564 12348754 15516181 13863422  6025439 13102174   6609617  15080341  5018624  5120336 14044758 15295157
            1024     128  2835325  5720477 13643232 14044758 12790037  5917516  8822754   6355328  13818817  5096034  5330028  9485232 13818817
            1024     256  2437821  4805258 12037272 12790037 11903823  5720477  8474583   6393168  12173747  4589593  6093831 11368190 13142265
            1024     512  2688041  5652717 12944224 12790037 12492426  5690162 11903823   5885083  11773301  3863055  7160597 11368190 12944224
            1024    1024  2553783  4972145 11368190 11520658 13142265  5277632 11773301   6025439  12456195  5251818  5782087 12173747 12492426
            2048       4  1885905  4221501 11706006  7207495  9062970  4130162  7556202   5754221   9101380  3743207  4138120  9796850 11260973
            2048       8  2062505  5119743 13836752 12959958 12118884  5549749  6873016   6895083  11440955  4764758  3483621  7207495 11770166
            2048      16  2060526  5131978 12580349  6990474 12805399  5319484 12327589   6523312   6298503  4900677  4552633 13489098 14539356
            2048      32  2488533  5277002 12882215 14944066 13926483  5970183 10240672   6807652   9308497  5277002  5135046 13467949  7189398
```

```
ubuntu@ip-172-31-23-104:~/iozone3_434/src/current$ ./iozone -g# -s 1024
        Iozone: Performance Test of File I/O
                Version $Revision: 3.434 $
                Compiled for 64 bit mode.
                Build: linux

        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, Vikentsi Lapa,
                     Alexey Skidanov.

        Run began: Sat Feb 13 01:49:37 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  1577984  4110780 11398360 11903823  8542002  4392454  8457895   5950309   8262639  4044965  3970183 10429596  9743447

iozone test complete.
```

As seen above, the performance for 1MB (1024KB) of file for iozone benchmarking tool is:

Sequential Write: 1.5048 Gbps, Sequential Read: 10.87 Gbps, Random Write: 4.188 Gbps, Random Read: 8.146 Gbps

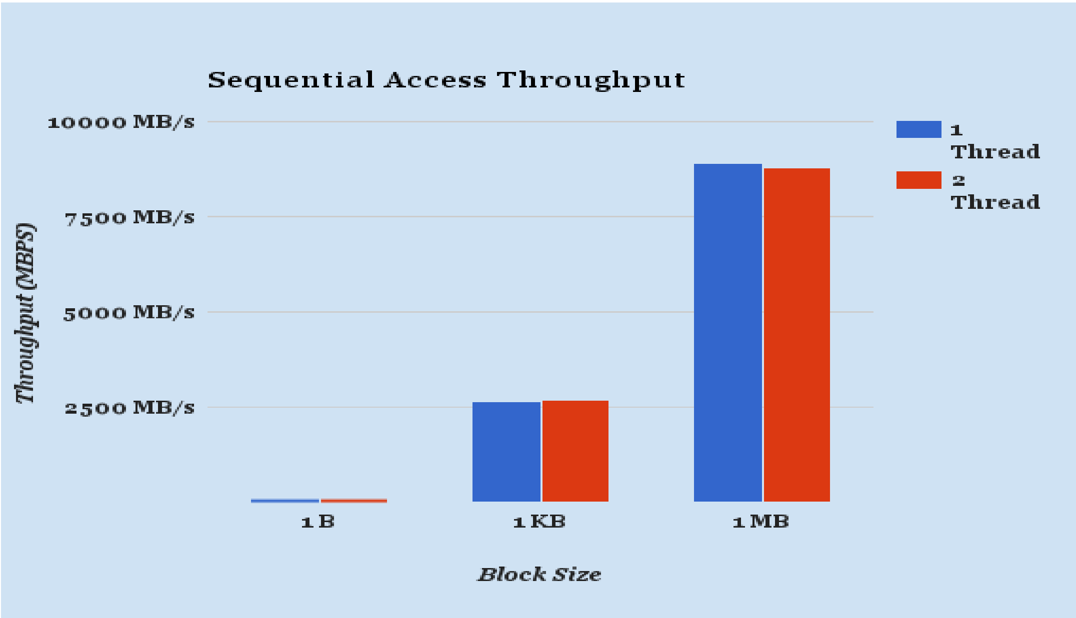
The efficiency of the tool when compared with theoretical performance for sequential write is about: **51%.**

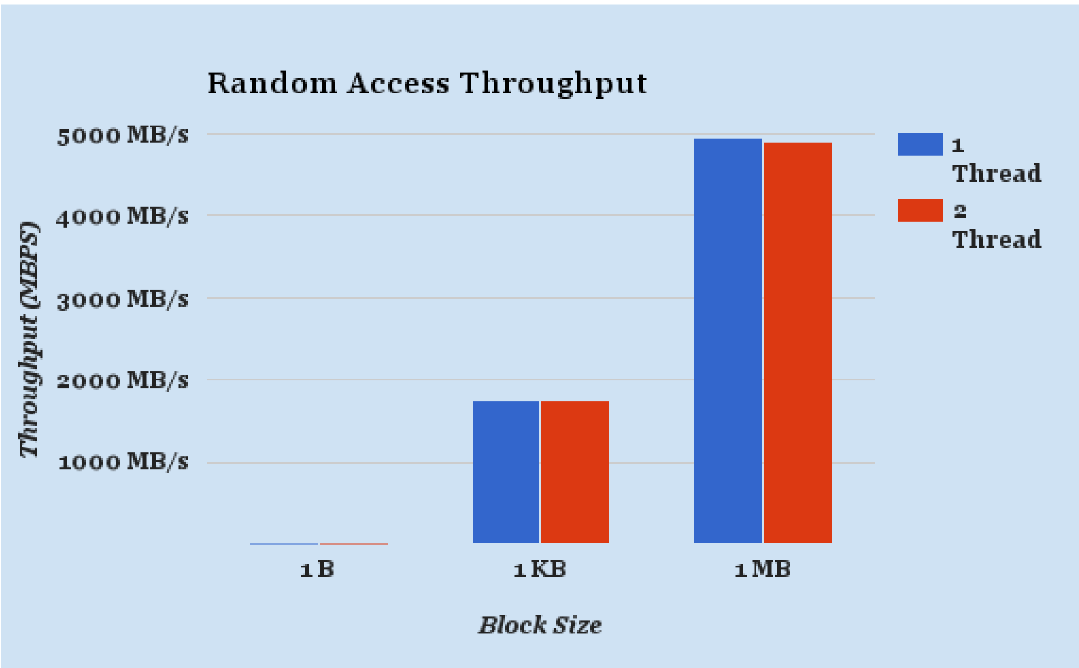
3. MEMORY BENCHMARKING

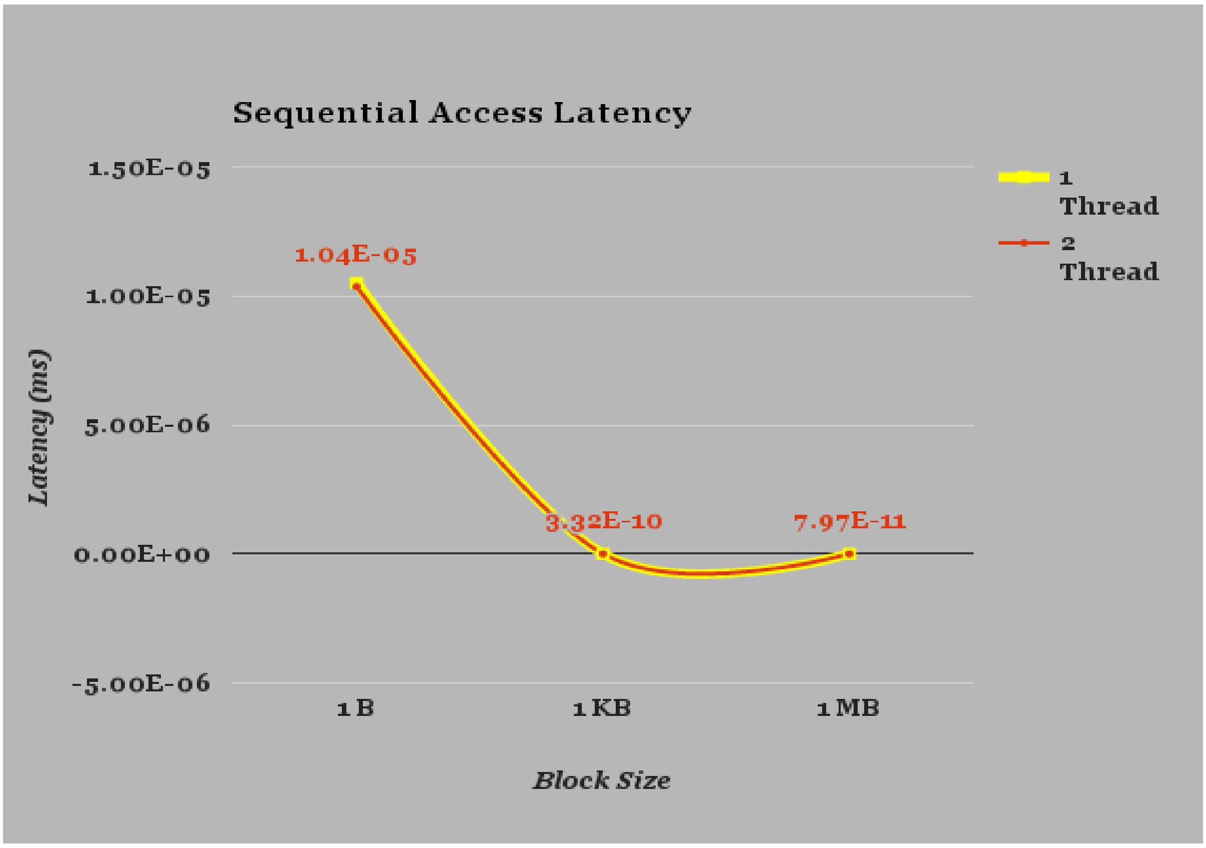
Amazon for memory doesn't specify its Base DRAM Clock frequency, which is essential for calculating the Theoretical performance. The Data transfer per clock is 2, Memory Bus width being 64 and Number of Interfaces are 2.

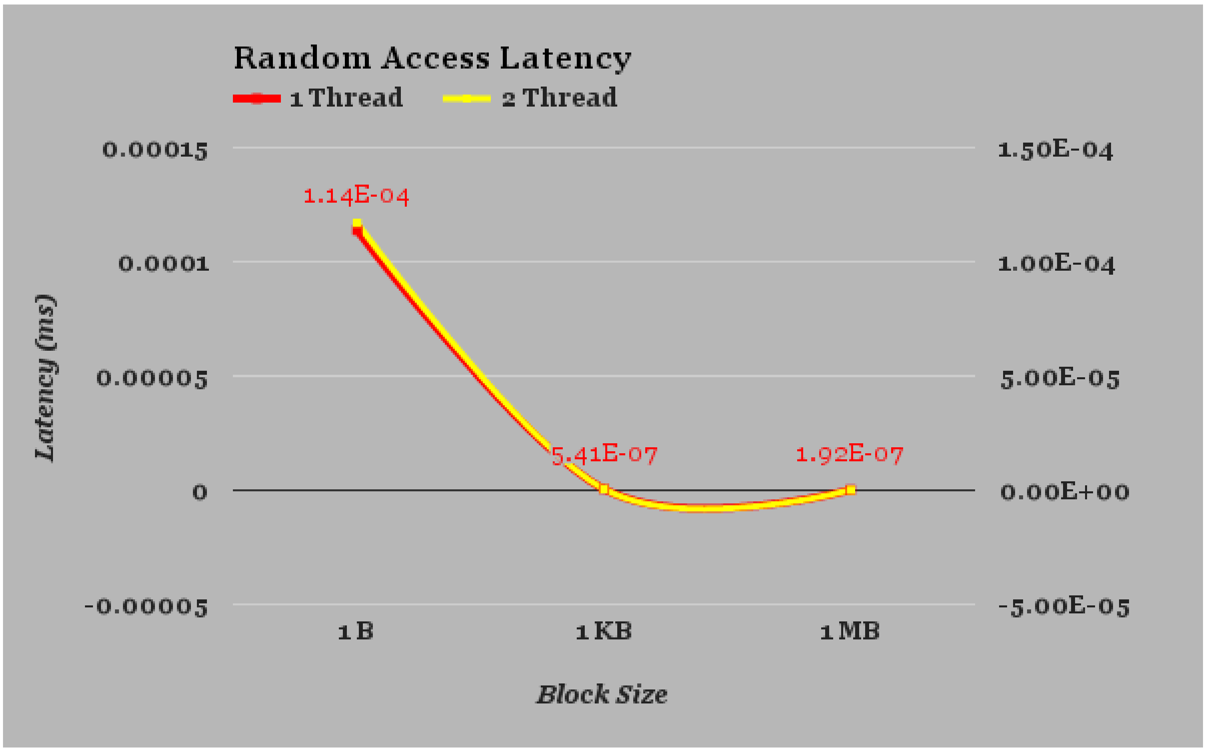
Theoretical Performance (considering my system): Base DRAM Clock freq. * No. of Data transfer per clock * Number of Interfaces = 1600 * 2 * 64 * 2 = 51.2 GB/s

The following graph shows the Throughput and Latency for Sequential and Random access of memory:









As, we can see the latency starts to decrease as we increase the number of threads, at some point of time, the memory speed becomes too fast that the latency tends to become towards power of E -15 (minus 15).

The Average and Standard Deviation for 3 Experiments:

Average (Latency in ms):

Block Size	1 B	1KB	1MB
Seq. Access	**1.050E-05**	**1.3E-09**	**2.215E-10**
Ran. Access	**0.000113766**	**5.41167E-07**	**1.92233E-07**

Average (Throughput in MB/s):

Block Size	1 B	1KB	1MB
Seq. Access	**90.796756**	**2654.4614**	**8906.71**
Ran. Access	**8.38496**	**1762.3905**	**4931.50911**

Standard Deviation (Latency):

Block Size	Seq. Access	Ran. Access
1 B	**1062E-07**	**2.233E-06**
1 KB	**1.6462E-09**	**6.997E-09**
1 MB	**1.0332E-11**	**2.0793E-09**

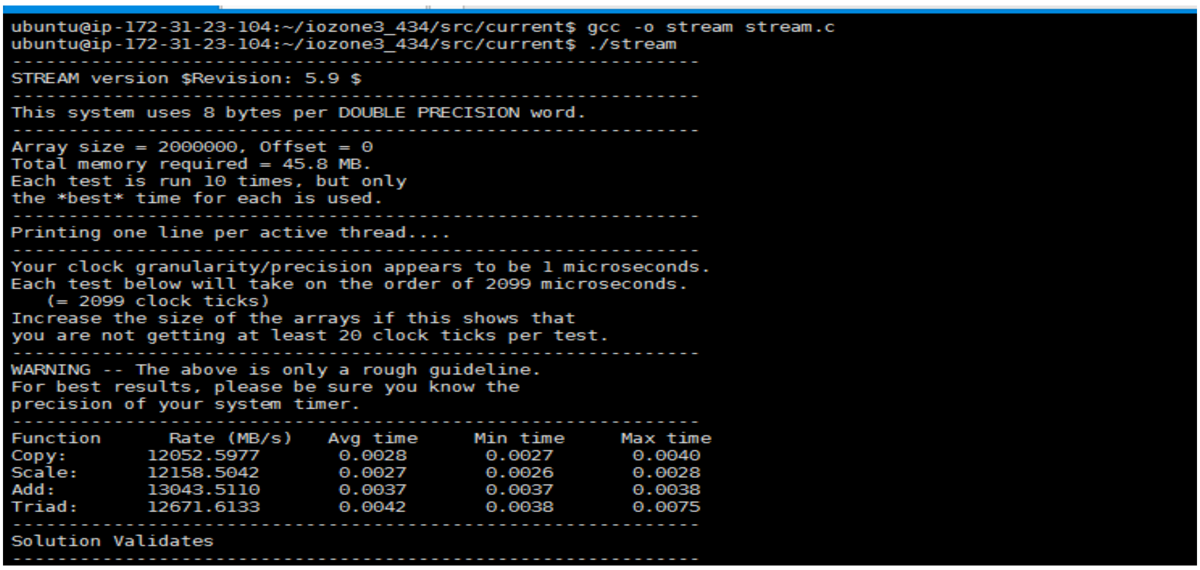
Standard Deviation (Throughput):

Block Size	Seq. Access	Ran. Access
1 B	**0.91697326**	**0.16612**
1 KB	**152.3571**	**22.7043**
1 MB	**121.122**	**53.0298**

STREAM Benchmarking:

The following screen shots of STREAM benchmarking toll shows the results for Memory Speeds achieved when evaluated on AWS.

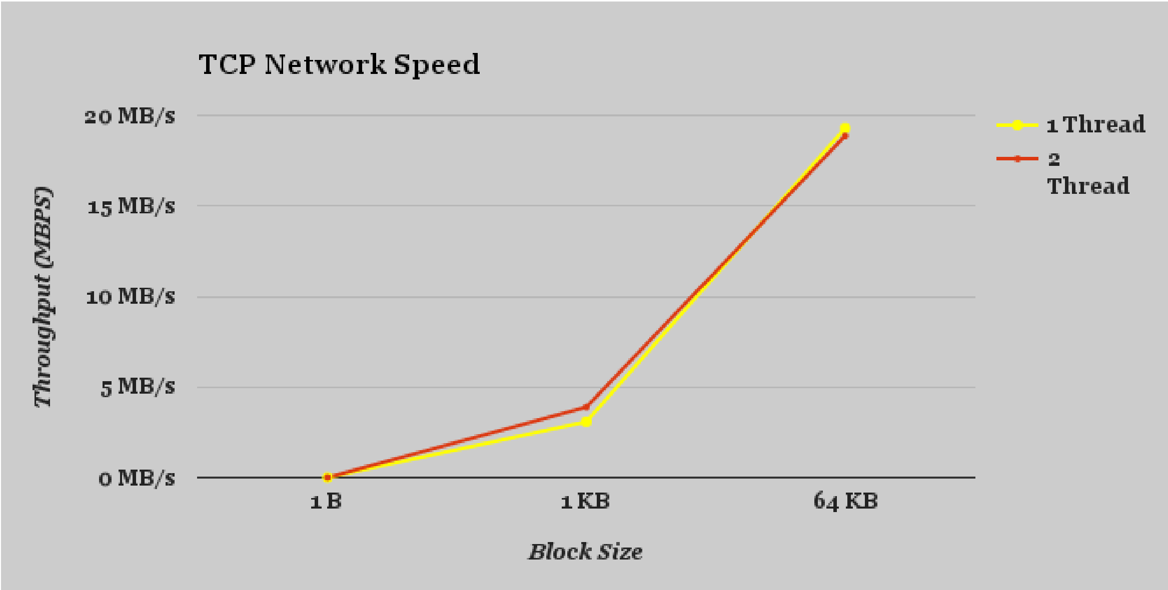
The result is divided into 4 parts, having a total of 48.75 GB/s, so the performance for it when compared with theoretical performance shows efficiency of: 95.21 %

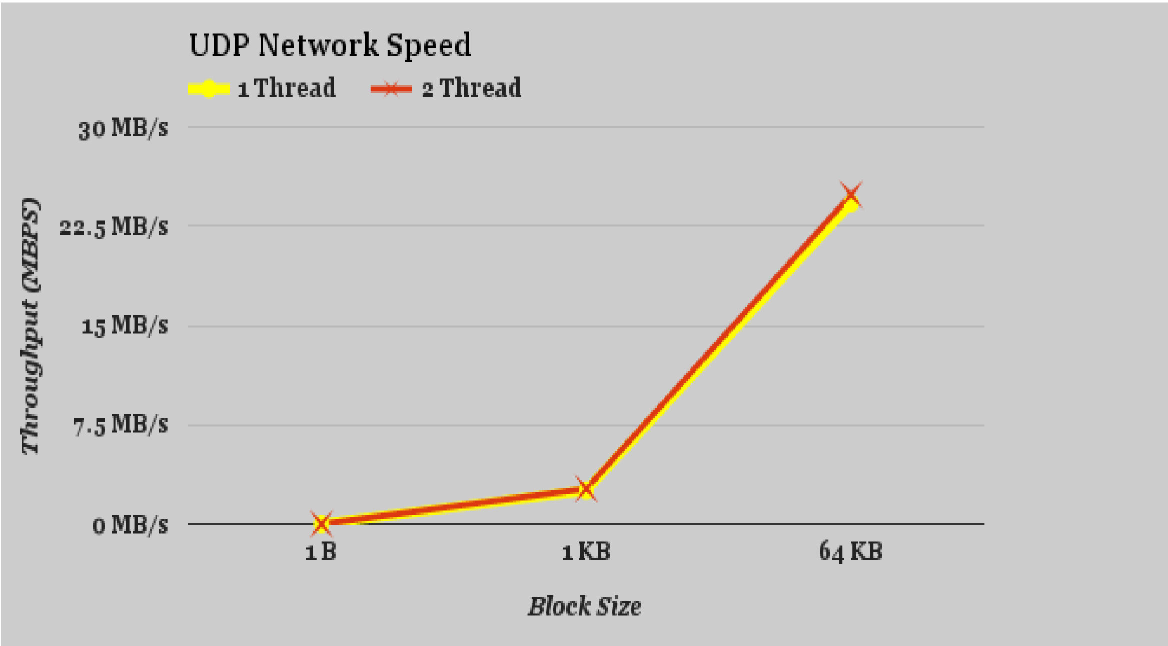


```
ubuntu@ip-172-31-23-104:~/iozone3_434/src/current$ gcc -o stream stream.c
ubuntu@ip-172-31-23-104:~/iozone3_434/src/current$ ./stream
-------------------------------------------------------------
STREAM version $Revision: 5.9 $
-------------------------------------------------------------
This system uses 8 bytes per DOUBLE PRECISION word.
-------------------------------------------------------------
Array size = 2000000, Offset = 0
Total memory required = 45.8 MB.
Each test is run 10 times, but only
the *best* time for each is used.
-------------------------------------------------------------
Printing one line per active thread....
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 2099 microseconds.
   (= 2099 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      Rate (MB/s)   Avg time     Min time     Max time
Copy:        12052.5977      0.0028       0.0027       0.0040
Scale:       12158.5042      0.0027       0.0026       0.0028
Add:         13043.5110      0.0037       0.0037       0.0038
Triad:       12671.6133      0.0042       0.0038       0.0075
-------------------------------------------------------------
Solution Validates
-------------------------------------------------------------
```

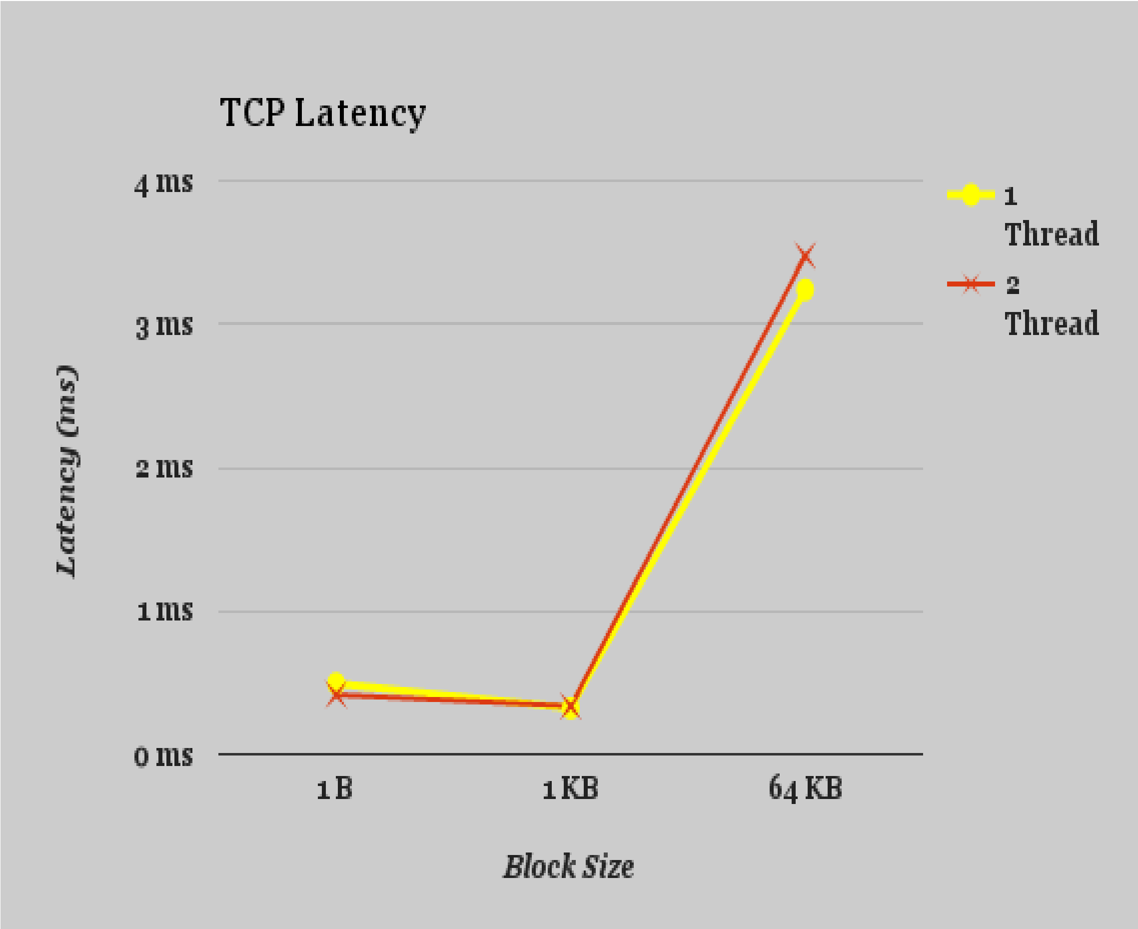
4. NETWORK BENCHMARK

The network benchmark evaluation is performed for TCP as well as UDP. Different Packet sizes are transferred ranging from 1B, 1KB, 64KB and the RTT is calculated accordingly.
The throughput and latency for each of them is shown below in the graphs:

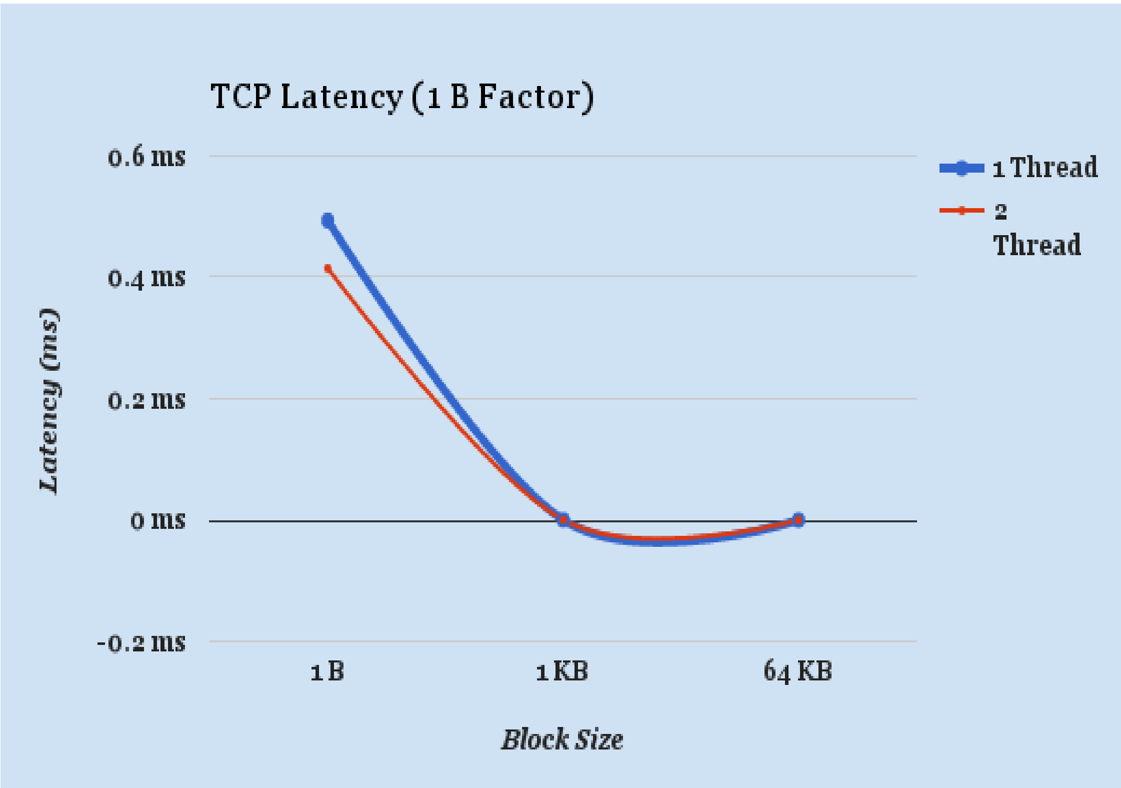




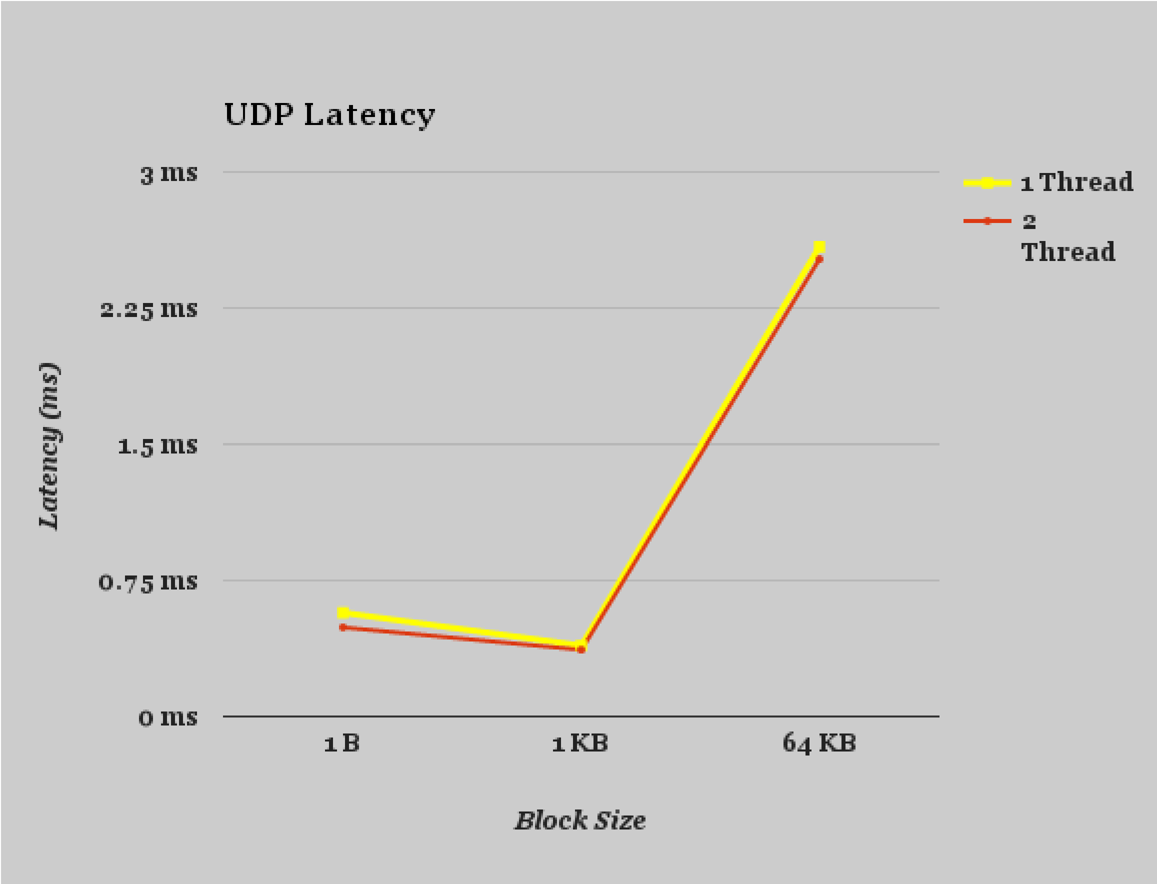
As seen, the network speed i.e. throughput increases with the increase in Packet size to sent for the TCP and UDP network protocol. UDP being connection less protocol is faster when compare with TCP network. There might be a lot of data which is lost when transmitted for UDP.

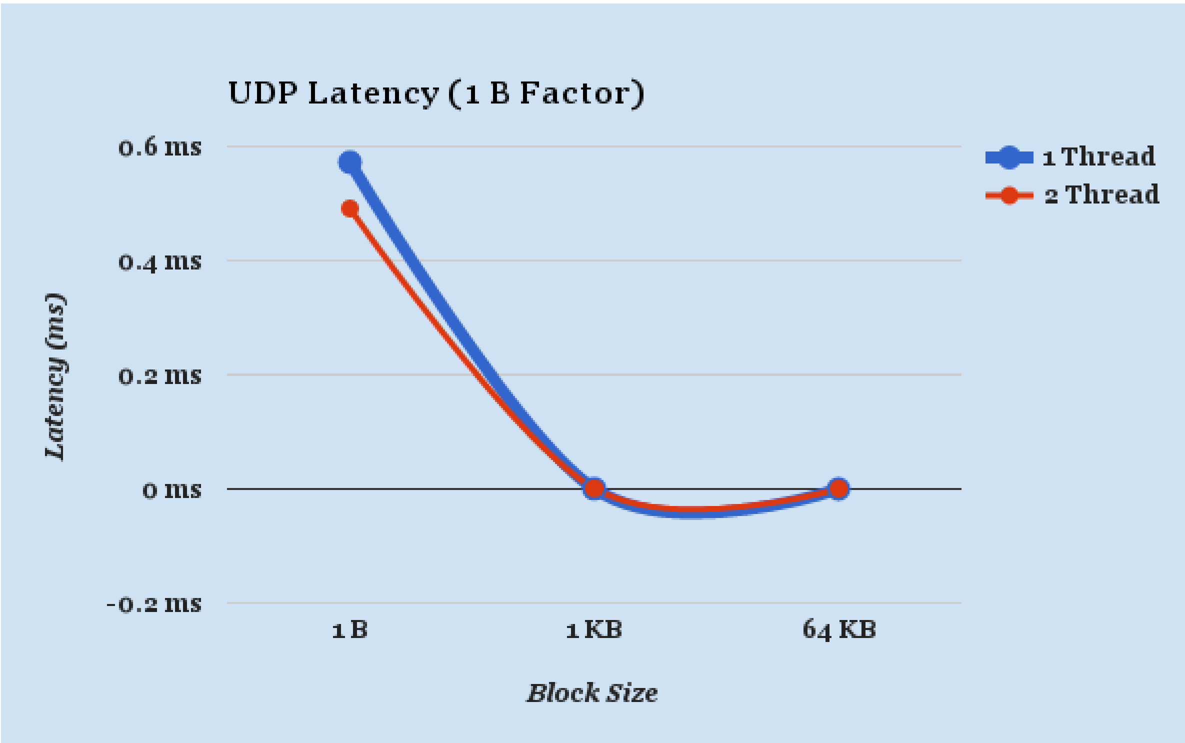


TCP Latency
4 ms
3 ms
2 ms
1 ms
0 ms
Latency (ms)
1 B
1 KB
64 KB
Block Size
1 Thread
2 Thread



TCP Latency (1 B Factor)
0.6 ms
0.4 ms
0.2 ms
0 ms
-0.2 ms
Latency (ms)
1 B
1 KB
64 KB
Block Size
1 Thread
2 Thread





The above graphs are for Latency, the 1B factor graphs are for comparing all the Packet sizes with the same 1Byte factor for latency. The other two graphs are for latency as a whole, i.e. the entire packet size latency is considered. This is the reason why the other two graph shows that latency is in increasing order. In actual, the 1 Byte factor graphs tells the real numbers when comparing latency. We observe that latency decreases with the increase in Packet Size to sent across network for both the cases of TCP and UDP.

The Average and Standard Deviation for 3 Experiments:

Average (Latency in ms):

Block Size	1 B	1KB	64KB
TCP	**0.493708**	**0.3216916**	**3.240265**
UDP	**0.5723611**	**0.391104**	**2.588017**

Average (Throughput in MB/s):

Block Size	1 B	1KB	64KB
TCP	**0.0020187**	**3.0817792**	**19.3021101**
UDP	**0.001669**	**2.5429939**	**24.2412297**

Standard Deviation (Latency):

Block Size	TCP	UDP
1 B	**0.134623**	**0.03145243**
1 KB	**0.050008**	**0.06583269**
64KB	**0.135084**	**0.16675977**

Standard Deviation (Throughput):

Block Size	TCP	UDP
1 B	**0.00047992**	**8.89916E-05**
1 KB	**0.44158978**	**0.41199362**
64KB	**0.8165913**	**1.58918945**

IPERF BENCHMARKING:
TCP & UDP:

```
ubuntu@ip-172-31-25-167:~$ iperf -c 52.26.161.78 -n 1
------------------------------------------------------------
Client connecting to 52.26.161.78, TCP port 5001
TCP window size:  325 KByte (default)
------------------------------------------------------------
[  3] local 172.31.25.167 port 46111 connected with 52.26.161.78 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 0.0 sec   128 KBytes  18.7 Gbits/sec
ubuntu@ip-172-31-25-167:~$ iperf -c 52.26.161.78 -n 64
------------------------------------------------------------
Client connecting to 52.26.161.78, TCP port 5001
TCP window size:  325 KByte (default)
------------------------------------------------------------
[  3] local 172.31.25.167 port 46112 connected with 52.26.161.78 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 0.0 sec   128 KBytes  18.7 Gbits/sec
ubuntu@ip-172-31-25-167:~$ iperf -c 52.26.161.78 -n 1024
------------------------------------------------------------
Client connecting to 52.26.161.78, TCP port 5001
TCP window size:  325 KByte (default)
------------------------------------------------------------
[  3] local 172.31.25.167 port 46113 connected with 52.26.161.78 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 0.0 sec   128 KBytes  15.7 Gbits/sec
ubuntu@ip-172-31-25-167:~$ iperf -c 52.26.161.78 -n 65536
------------------------------------------------------------
Client connecting to 52.26.161.78, TCP port 5001
TCP window size:  325 KByte (default)
------------------------------------------------------------
[  3] local 172.31.25.167 port 46114 connected with 52.26.161.78 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 0.0 sec   128 KBytes  16.4 Gbits/sec
```

```
error: No address associated with hostname
ubuntu@ip-172-31-25-167:~$ iperf -c 52.26.161.78 -u -n 1
------------------------------------------------------------
Client connecting to 52.26.161.78, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 172.31.25.167 port 50114 connected with 52.26.161.78 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 0.0 sec  1.44 KBytes  1.04 Mbits/sec
[  3] Sent 1 datagrams
[  3] Server Report:
[  3]  0.0- 0.0 sec  1.44 KBytes  1.04 Mbits/sec   0.000 ms    0/    1 (0%)
ubuntu@ip-172-31-25-167:~$ iperf -c 52.26.161.78 -u -n 1024
------------------------------------------------------------
Client connecting to 52.26.161.78, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 172.31.25.167 port 56925 connected with 52.26.161.78 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 0.0 sec  1.44 KBytes  1.04 Mbits/sec
[  3] Sent 1 datagrams
[  3] Server Report:
[  3]  0.0- 0.0 sec  1.44 KBytes  1.05 Mbits/sec   0.000 ms    0/    1 (0%)
ubuntu@ip-172-31-25-167:~$ iperf -c 52.26.161.78 -u -n 65536
------------------------------------------------------------
Client connecting to 52.26.161.78, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 172.31.25.167 port 55050 connected with 52.26.161.78 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 0.5 sec  64.6 KBytes  1.05 Mbits/sec
[  3] Sent 45 datagrams
[  3] Server Report:
[  3]  0.0- 0.5 sec  64.6 KBytes  1.05 Mbits/sec   0.041 ms    0/   45 (0%)
```