CS 553 PROGRAMMING ASSIGNMENT-1

EVALUATION DOCUMENT

ROHIT SINGH

A20361198

1 Amazon System Configuration

I have used t2.micro instance to do the following evaluations. I have used the following commands to find the system configurations for CPU, Memory and Disk.

- cat /proc/cpuinfo
- sudo lshw -c memory
- sudo dmidecode 2.9 | grep -i speed
- sudo hdparm -tT /dev/sda

The specifications can be summarized as follows:

- processor : 0
- vendor_id : GenuineIntel
- cpu family : 6
- model : 62
- model name : Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz
- stepping : 4
- microcode : 0x416
- cpu MHz : 2500.092
- cache size : 25600 KB
- cpu cores : 1

Memory

- RAM : 1GB
- description : DIMM RAM
- physical id : 0
- slot : DIMM 0
- size : 1GiB
- width : 64 bits
- Max Speed : 1600 MHz

Disk

- Timing cached reads : 1912 MB in 2.03 seconds = 941.93 MB/sec
- Timing buffered disk reads : 236 MB in 3.01 seconds = 78.37 MB/sec

2 CPU

I performed the evaluation on t2.micro instance on AWS, and the following are data for the GFLOPS and GIOPS for my benchmark and LINPACK.

2.1 AWS Implementation Results 1

GFLOPS

Threads	Iterations			Average	Standard Deviation
	1	2	3		
1	6.088262	6.146171	6.985806	6.40674633	0.502315574
2	6.126962	6.164513	6.984988	6.42548767	0.484905132
4	6.148727	6.166488	6.967125	6.42744667	0.467459507

Table 1. Average and Standard Deviation of GFLOPS for my benchmark

The average GFLOPS for the three experiments comes to about **6.41989356**. The GFLOPS increases with increase in threads. However, there is a slight dip in the slope for 4 threads, which is evident since AWS is a single threaded CPU, and the instruction are executed one after another.

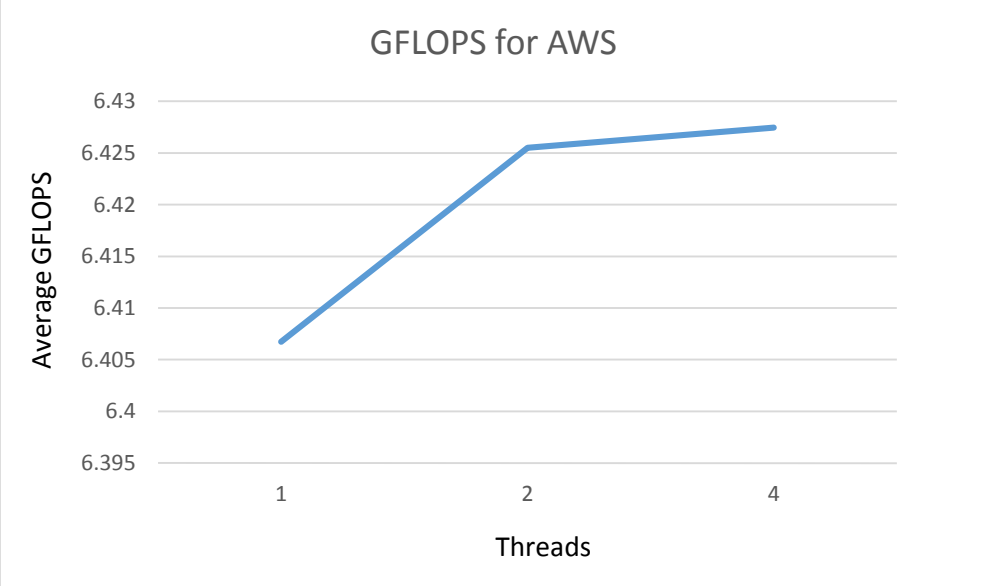


Fig 1. Average GFLOPS vs Threads for my benchmark

GIOPS

Threads	Iterations			Average	Standard Deviation
	1	2	3		
1	7.121369	7.171229	6.995608	7.09606867	0.09050284
2	7.172679	7.177593	6.987513	7.112595	0.108352051
4	7.152859	7.199679	6.993852	7.11546333	0.107888903

Table 2. Average and Standard Deviation of GIOPS for my benchmark

The average GIOPS for the three experiments comes to about **7.10804233**. The GIOPS increases with increase in threads. However, there is a slight dip in the slope for 4 threads, which is evident since AWS is a single threaded CPU, and the instruction are executed one after another.

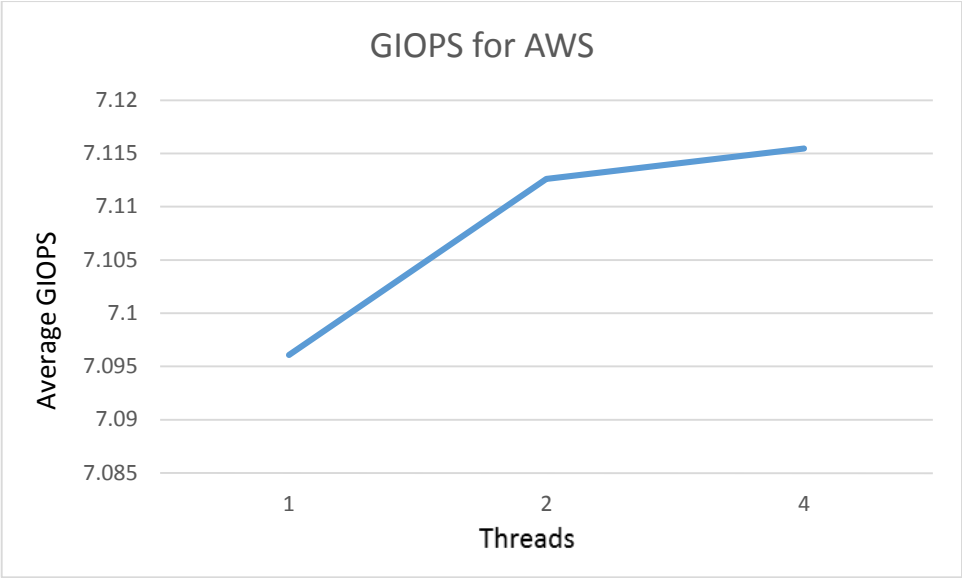


Fig 2. Average GIOPS vs Threads for my benchmark

Theoretical value

$$GFlops = (CPUspeed \in GHz)x(numberofCPUcores)x(CPUinstructionpercycle) \quad (1)$$

Where,
- The CPU speed from Section 1 is 2.5GHz
- The number of cores from Section 1 is 1
- The IPC (Instructions Per Cycle) for Xeon(R) CPU E5-2670 is 8 as done in [1]

Hence, from Equation 1 we get

$$GFlops = (2.5GHz)x(1core)x(8IPC) = 20$$

The efficiency of my benchmark over Theoretical value is 32.10%.

2.2 AWS Implementation Results 2

I have done sampling of the two operation (GFLOPS and GIOPS) every second for 10mins, for 4 threads.

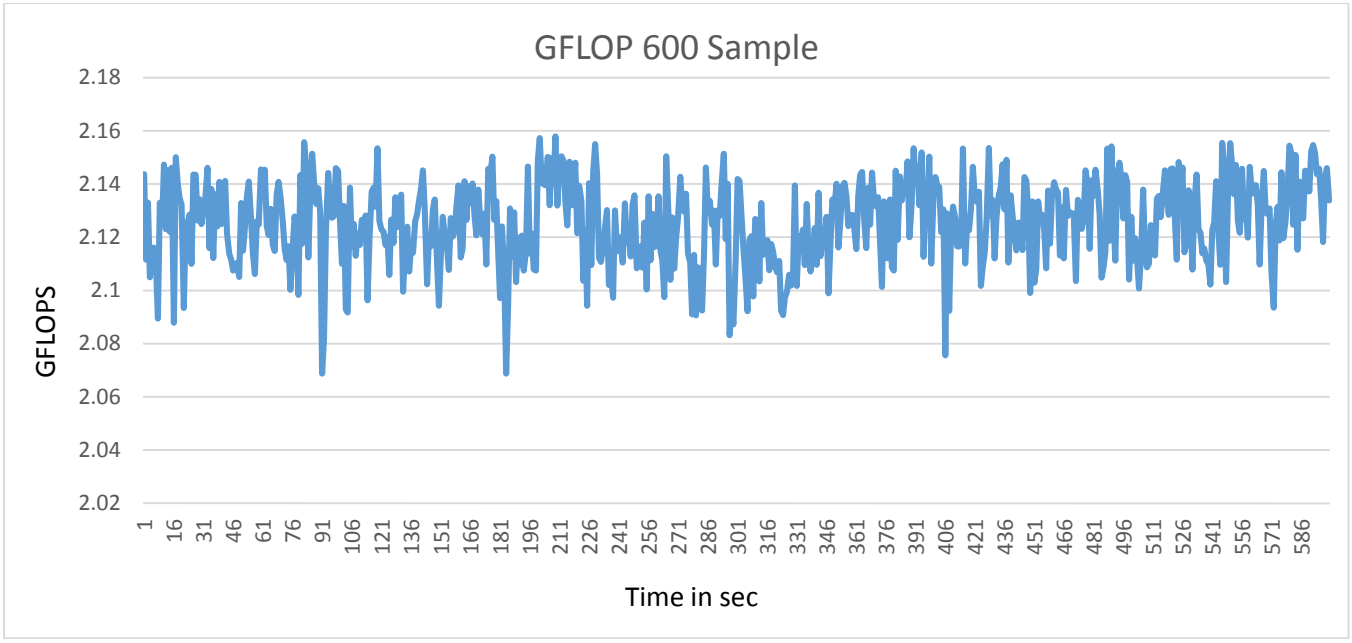


Fig. 3 GFLOPS for 600 s

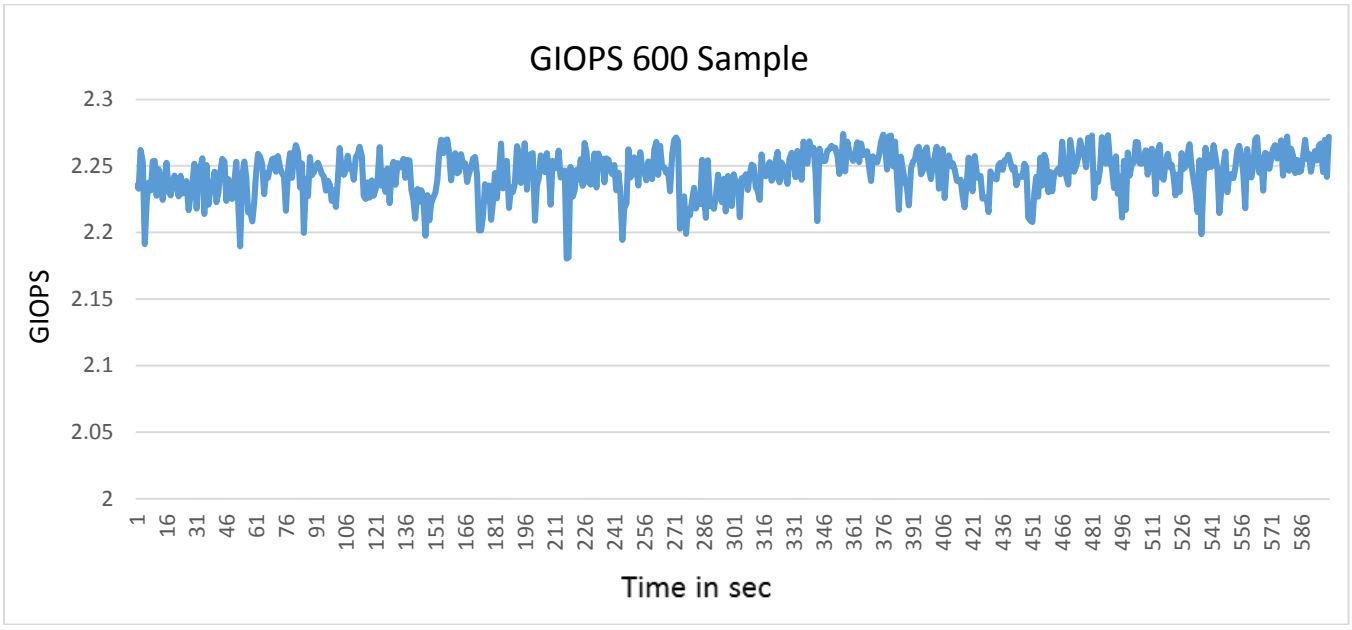
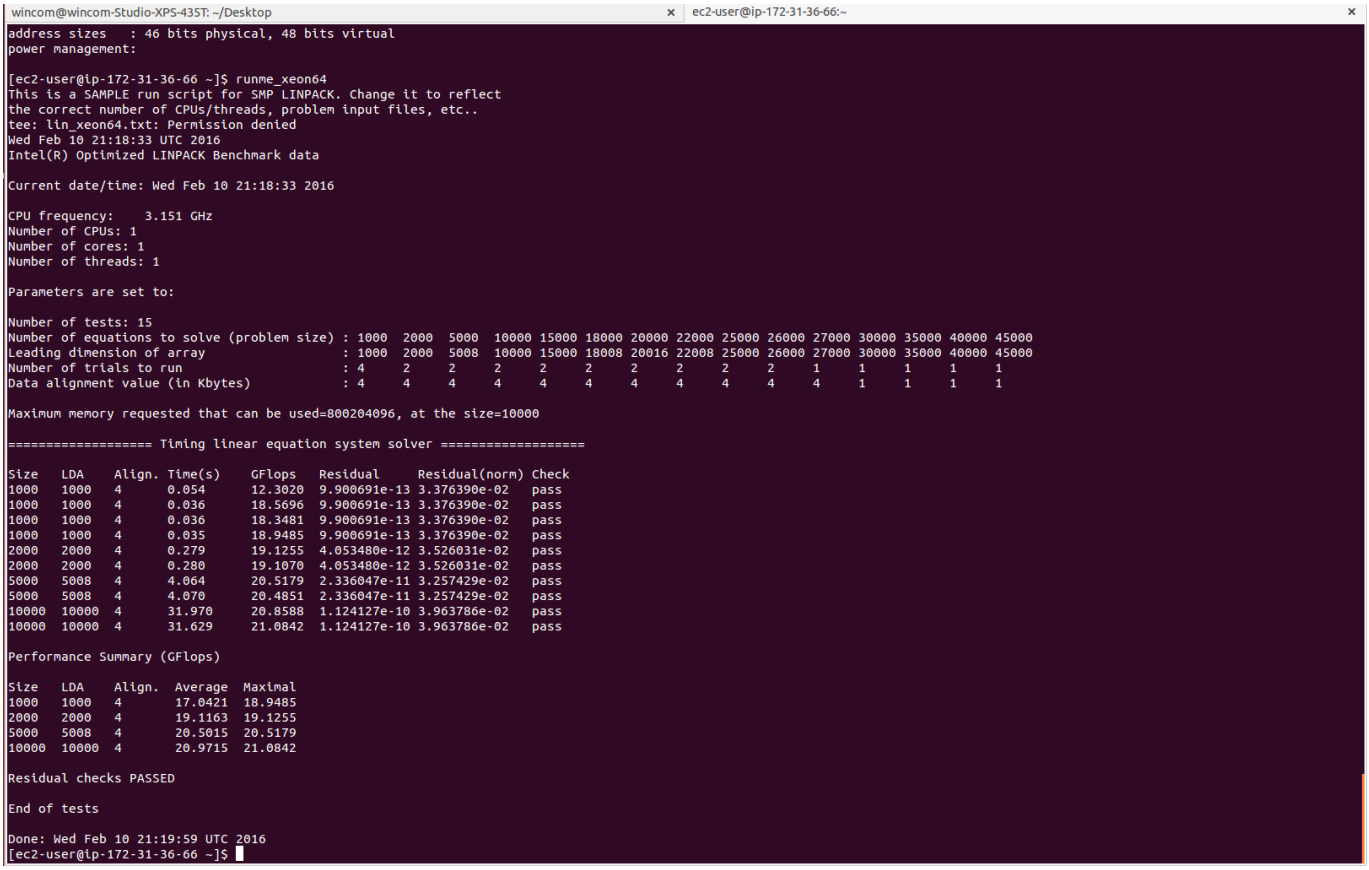


Fig. 4 GIOPS for 600 s

2.3 LINPACK Output

I ran the LINPACK Benchmark on AWS, and the output is as follows:



```
wincom@wincom-Studio-XPS-435T: ~/Desktop                    ec2-user@ip-172-31-36-66:~
address sizes   : 46 bits physical, 48 bits virtual
power management:

[ec2-user@ip-172-31-36-66 ~]$ 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..
tee: lin_xeon64.txt: Permission denied
Wed Feb 10 21:18:33 UTC 2016
Intel(R) Optimized LINPACK Benchmark data

Current date/time: Wed Feb 10 21:18:33 2016

CPU frequency:    3.151 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     1     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      0.054      12.3020  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.036      18.5696  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.036      18.3481  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.035      18.9485  9.900691e-13 3.376390e-02   pass
2000   2000   4      0.279      19.1255  4.053480e-12 3.526031e-02   pass
2000   2000   4      0.280      19.1070  4.053480e-12 3.526031e-02   pass
5000   5008   4      4.064      20.5179  2.336047e-11 3.257429e-02   pass
5000   5008   4      4.070      20.4851  2.336047e-11 3.257429e-02   pass
10000  10000  4      31.970     20.8588  1.124127e-10 3.963786e-02   pass
10000  10000  4      31.629     21.0842  1.124127e-10 3.963786e-02   pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
1000   1000   4       17.0421  18.9485
2000   2000   4       19.1163  19.1255
5000   5008   4       20.5015  20.5179
10000  10000  4       20.9715  21.0842

Residual checks PASSED

End of tests

Done: Wed Feb 10 21:19:59 UTC 2016
[ec2-user@ip-172-31-36-66 ~]$
```

Fig. 5 Snapshot from LINPACK

On running the LINPACK script it took a set of predefined parameters to check the CPU performance. I have drawn a comparison of the GFLOPS for 4 different set of problem sizes, taking other parameters constant.

Problem Size	GFLOPS
1000	17.04
2000	19.12
5000	20.5
10000	20.97

Table. 3 GFLOPS for LINPACK for different Problem Sizes

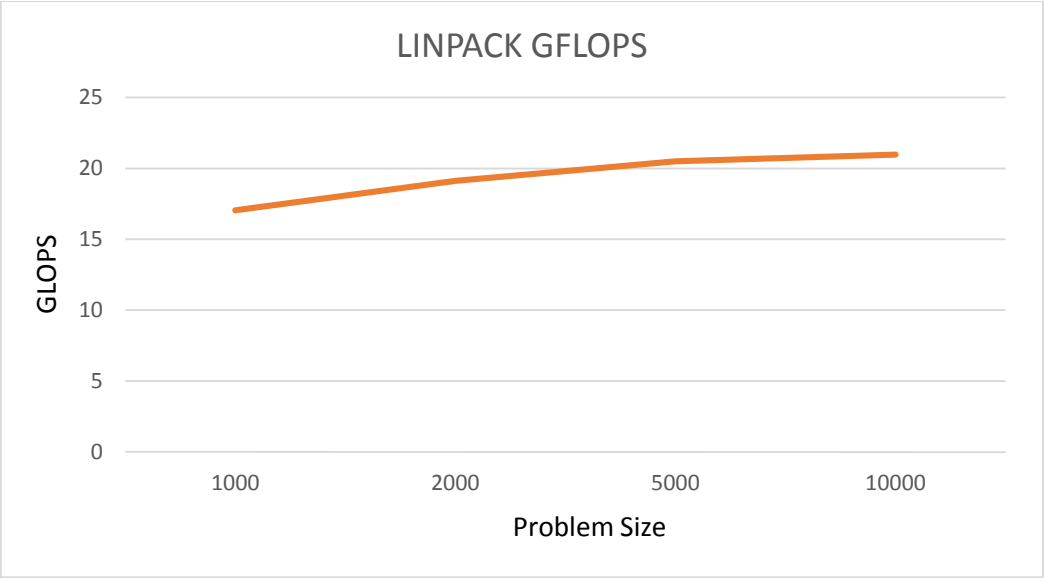


Fig 5. Average GIOPS vs Problem Size for my benchmark

The average GFLOPS for the LINPACK Benchmark comes out to be 19.4075.

Hence, the efficiency of LINPACK over Theoretical value is 97%.

3 MEMORY

I performed the evaluation on t2.micro instance on AWS, and the following are data for the Latency and Throughput for my benchmark and STREAM.

3.1 AWS Implementation Results

A) Sequential Read+Write

Block size	Threads	Iterations					
		1		2		3	
		Latency	Throughput	Latency	Throughput	Latency	Throughput
1B	1	0.00012	8.22	0.00013	7.45	0.00012	8.29
	2	0.00004	23.55	0.00004	24.14	0.00004	23.26
1KB	1	0.00008	12361.55	0.00007	14796.40	0.00008	12682.63
	2	0.00006	17595.72	0.00005	18601.19	0.00005	20135.31
1MB	1	0.04172	23970.47	0.04256	23495.69	0.04255	23500.66
	2	0.04157	24054.36	0.04252	23518.62	0.04219	23702.30

Table. 4 Latency and Throughput for different Block-Sizes and Threads for Sequential Access

Block size	Threads	Average		Standard Deviation	
		Latency	Throughput	Latency	Throughput
1B	1	0.00012	7.99	0.00001	0.46700

	2	0.00004	23.65	0.00000	0.45057
1KB	1	0.00007	13280.19	0.00001	1322.85112
	2	0.00005	18777.41	0.00000	1278.93259
1MB	1	0.04228	23655.61	0.00048	272.69097
	2	0.04209	23758.43	0.00048	272.24436

Table. 5 Average Latency and Throughput for different Block-Sizes and Threads for Sequential Access

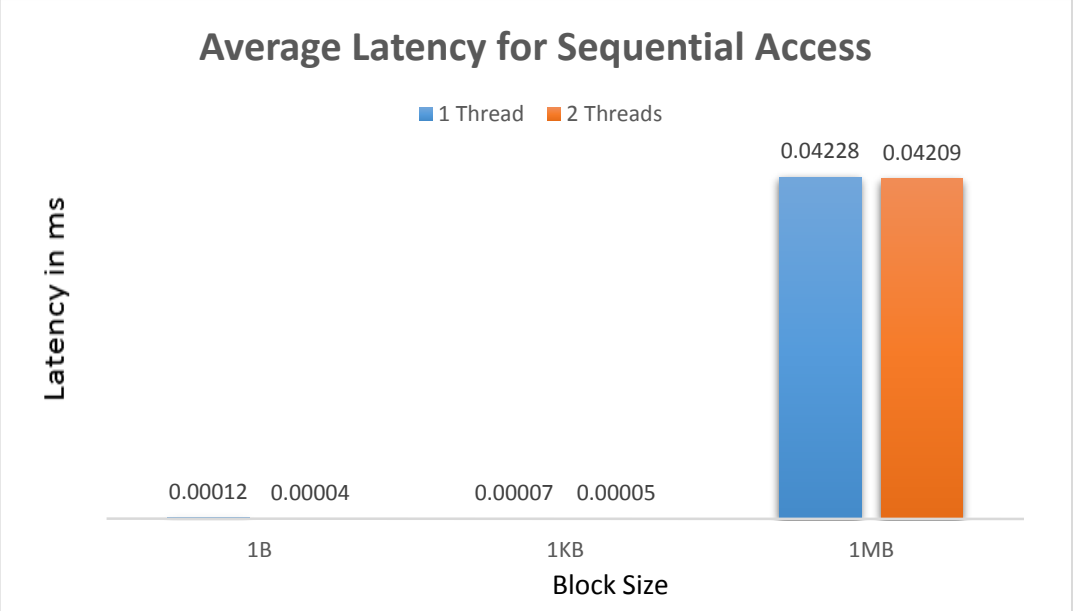


Fig. 6 Average Latency for Sequential Access

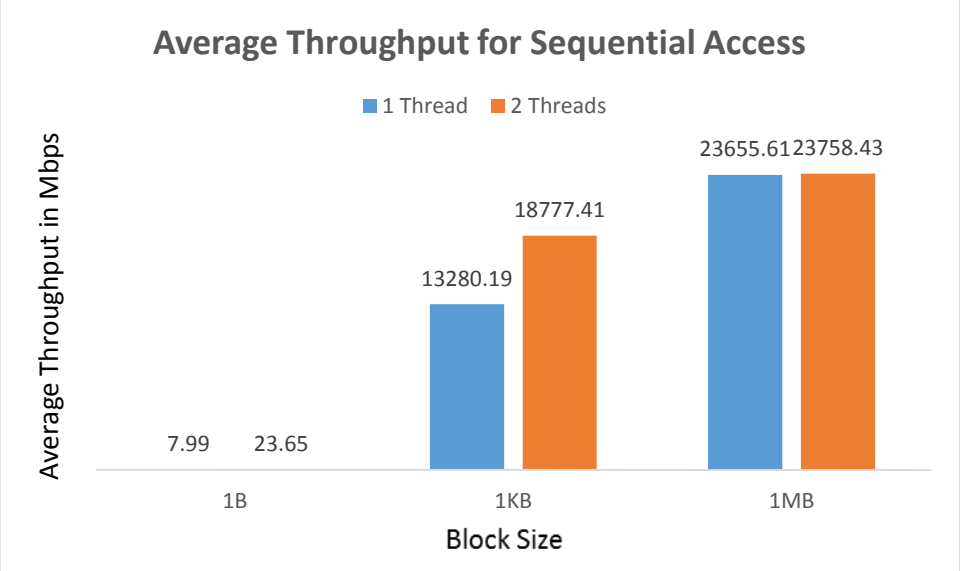


Fig. 7 Average Throughput for Sequential Access

The average latency for the Sequential access comes out to be **0.04219ms** and Throughput is **23707.02 Mbps.**

B) Random Read+Write

Block size	Threads	Iterations					
		1		2		3	
		Latency	Throughput	Latency	Throughput	Latency	Throughput
1B	1	0.00171	0.55901	0.00182	0.52285	0.00186	0.51245
	2	0.00137	0.69841	0.00142	0.67302	0.00141	6.74929
1KB	1	0.00183	532.76730	0.00186	524.75150	0.00193	506.51580
	2	0.00176	553.92090	0.00181	538.49600	0.00184	531.02910
1MB	1	0.08422	11874.37000	0.08854	11294.33000	0.08952	11171.31000
	2	0.09889	10111.99000	0.10136	9866.21400	0.10137	9864.60800

Table. 6 Latency and Throughput for different Block-Sizes and Threads for Random Access

Block size	Threads	Average		Standard Deviation	
		Latency	Throughput	Latency	Throughput
1B	1	0.00180	0.53	0.00008	0.02444
	2	0.00140	2.71	0.00003	3.50083
1KB	1	0.00187	521.34	0.00005	13.45322
	2	0.00181	541.15	0.00004	11.67416
1MB	1	0.08742	11446.67	0.00282	375.47163
	2	0.10054	9947.60	0.00143	142.36472

Table. 7 Average Latency and Throughput for different Block-Sizes and Threads for Random Access

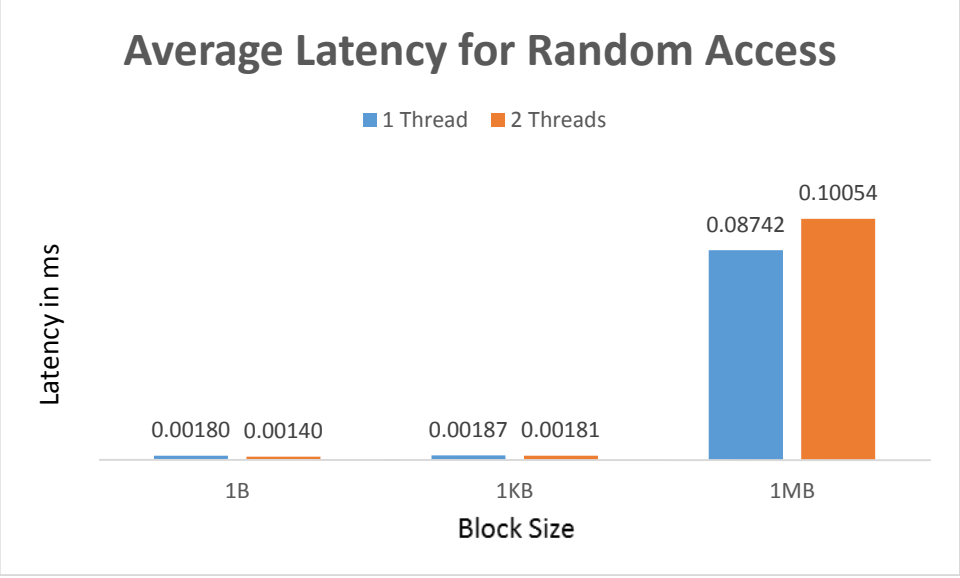


Fig. 8 Average Latency for Random Access

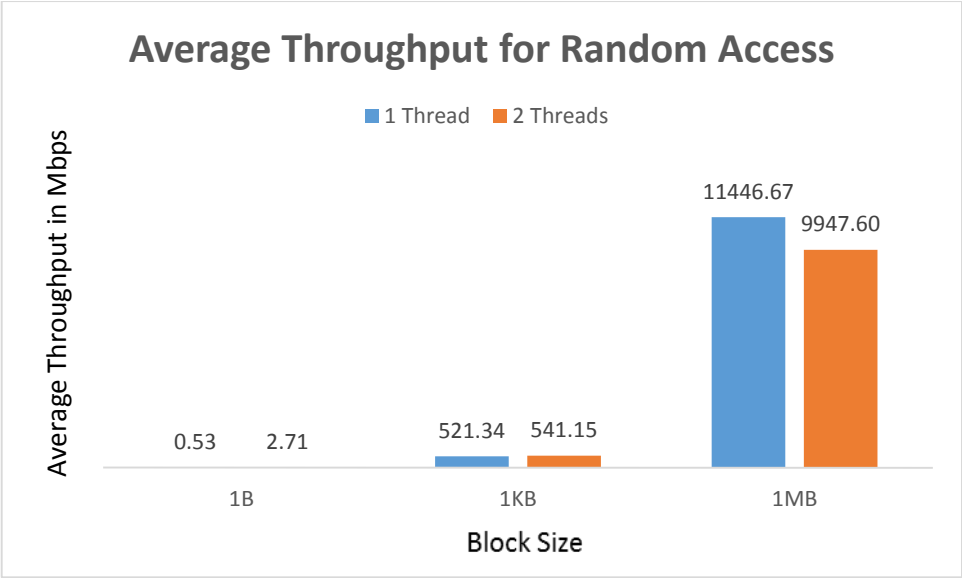


Fig. 9 Average Throughput for Random Access

The average latency for the Random access comes out to be **0.09398ms** and Throughput is **10697.14 Mbps.**

Theoretical value

$$Throughput = (BaseDRAMclockfrequency)x(Numberofdatatransfersperclock)x(Memorybuswidth)x(Numberofinterfaces)$$

Where,
-DRAM clock frequency 1600 MHz from Section 1
- Data transfer per clock is 2. [2]
- Memory width is 64 from Section 1
- Interfaces is 2

$$Throughput = 1600MHzx2x64x2 = 409{,}600Mbps$$

The efficiency of my benchmark over Theoretical value is **5.78%** for Sequential and **2.61%** for Random access. The efficiency is less because the AWS instance scales up and down according to its need. So, one cannot precisely say about the theoretical throughput of the t2.instance.

3.2 STREAM Output

I ran the STREAM Benchmark on AWS, and the output is as follows:

```
[ec2-user@ip-172-31-36-66 ~]$ ls
c     ccc          cpumod.c   d       -dm             iozone.sh                  m30          Sample FLOPS.txt
-c2   cpt          cpumodt.c  dd      dt              lin_xeon64.txt             memory.c     Sample IOPS.txt
c3    cpumod2.c    -ct        -dd     File.txt        lshw-2.14-1.el4.rf.i386.rpm  mm         stream-scaling-master
c600  cpumod3.c    ctt        disk.c  iozone3_394     m                          mm2
cc3   cpumod600.c  cu         dm      iozone3_394.tar m10                        prog1-v2.pdf
[ec2-user@ip-172-31-36-66 ~]$ cd stream-scaling-master/
[ec2-user@ip-172-31-36-66 stream-scaling-master]$ ./stream
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 14432814 (elements), Offset = 0 (elements)
Memory per array = 110.1 MiB (= 0.1 GiB).
Total memory required = 330.3 MiB (= 0.3 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.
-------------------------------------------------------------
Number of Threads requested = 1
Number of Threads counted = 1
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 13683 microseconds.
   (= 13683 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:           14011.7     0.016590     0.016481     0.016744
Scale:          11301.6     0.020634     0.020433     0.020868
Add:            12313.4     0.028265     0.028131     0.028592
Triad:          12480.7     0.027882     0.027754     0.028158
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

Fig. 10 STREAM output snapshot

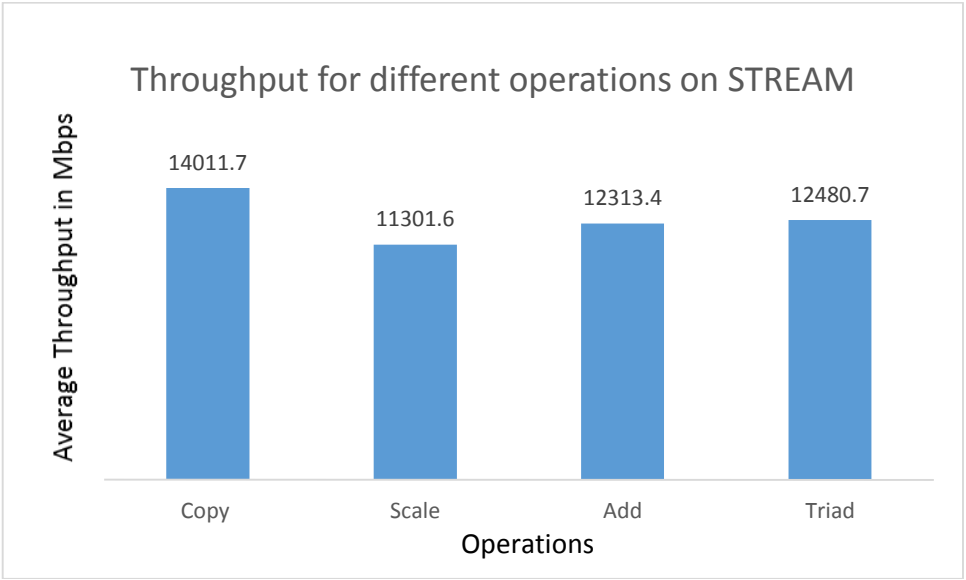


Fig. 11 Throughput for different operations on STREAM

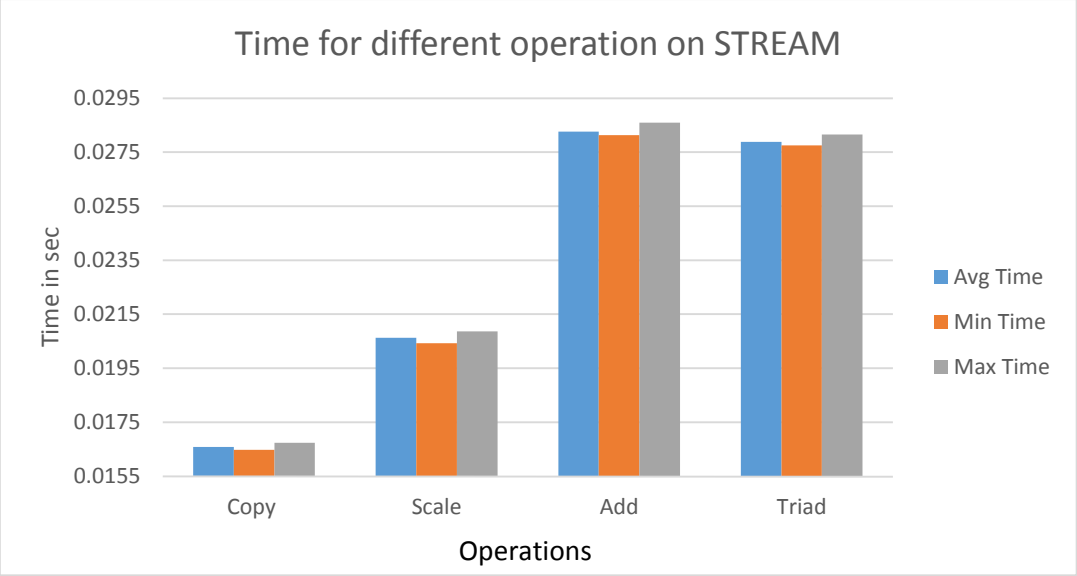


Fig. 12 Time for different operations on STREAM

The efficiency of STREAM over Theoretical value is 3%, which may because of the scaling up and down of the AWS. As soon as a big memory is given to be read or write the threshold is exceeded and the resources scales up.

4 DISK

I performed the evaluation on t2.micro instance on AWS, and the following are data for the Latency and Throughput for my benchmark and IOZONE.

4.1 AWS Implementation Results

A) Sequential Write

Block size	Threads	Iterations					
		1		2		3	
		Latency	Throughput	Latency	Throughput	Latency	Throughput
1B	1	1.00600	0.00095	1.02400	0.00093	1.06400	0.00090
	2	0.05900	0.01616	0.05950	0.01603	0.06050	0.01576
1KB	1	0.09200	10.61481	0.09400	10.38896	0.09400	10.38896
	2	0.06850	14.25639	0.07150	13.65822	0.06850	14.25639
1MB	1	22.77600	43.90587	22.79600	43.86735	22.84500	43.77325
	2	22.53200	44.38132	22.63450	44.18034	22.55600	44.33410

Table. 8 Latency and Throughput for different Block-Sizes and Threads for Sequential Write

Block size	Threads	Average		Standard Deviation	
		Latency	Throughput	Latency	Throughput
1B	1	1.03133	0.00093	0.02969	0.00003
	2	0.05967	0.01599	0.00076	0.00020
1KB	1	0.09333	10.46424	0.00115	0.13039
	2	0.06950	14.05700	0.00173	0.34535
1MB	1	22.80567	43.84882	0.03550	0.06822
	2	22.57417	44.29859	0.05361	0.10509

Table. 9 Average Latency and Throughput for different Block-Sizes and Threads for Sequential Write

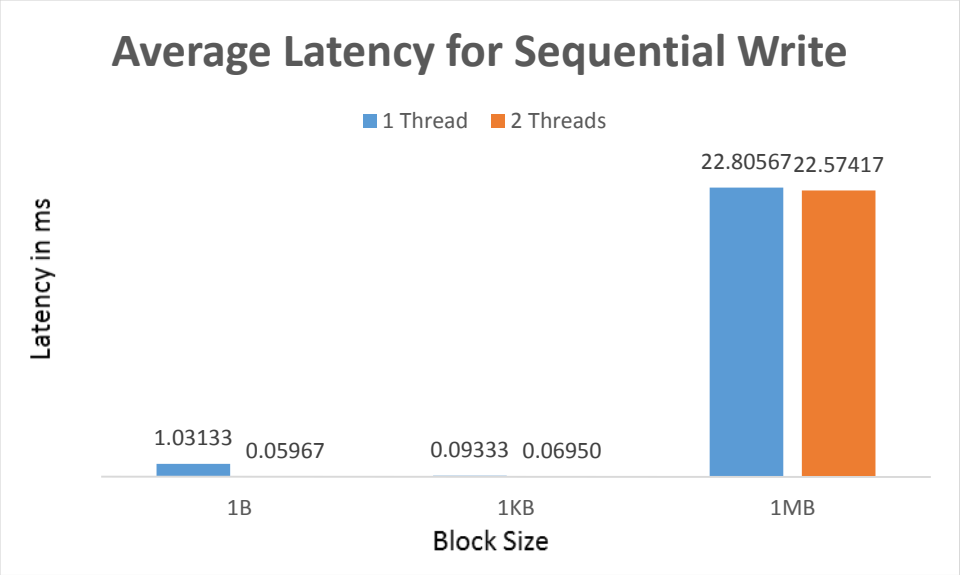


Fig. 13 Average Latency for Sequential Write

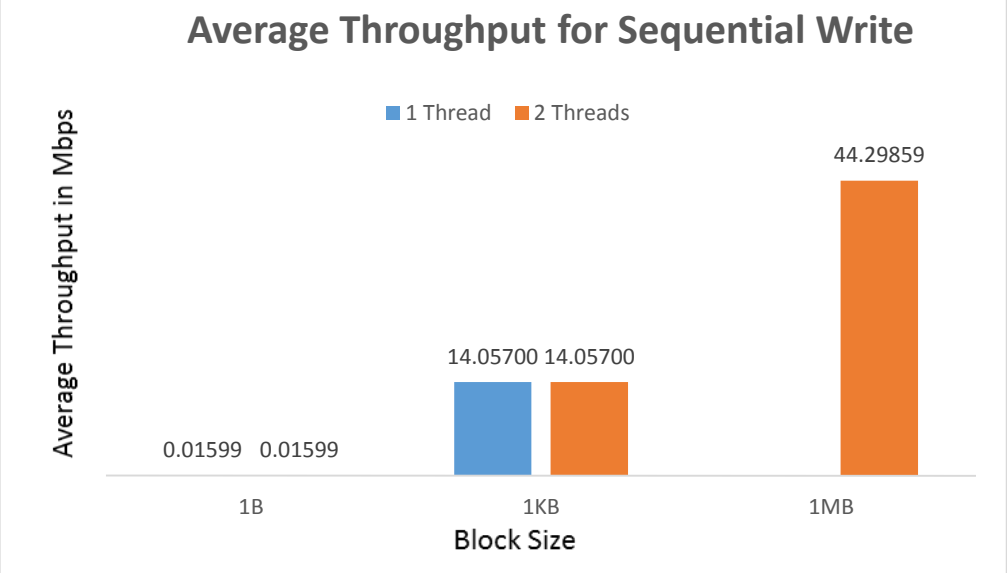


Fig. 14 Average Throughput for Sequential Write

The average latency for the Sequential Write comes out to be **22.68992ms** and Throughput is **44.07371Mbps**, for a Block size of 1MB**.**

B) Sequential Read

Block size	Threads	Iterations					
		1		2		3	
		Latency	Throughput	Latency	Throughput	Latency	Throughput
1B	1	0.11500	0.00829	0.12100	0.00788	0.10200	0.00935
	2	0.03600	0.02649	0.03750	0.02543	0.04200	0.02271
1KB	1	0.10400	9.39002	0.10600	9.21285	0.10500	9.30060
	2	0.08500	11.48897	0.08650	11.28974	0.09250	10.55743
1MB	1	54.89900	18.21527	55.05400	18.16398	55.06600	18.16003
	2	54.84000	18.23487	55.40800	18.04794	55.46450	18.02955

Table. 10 Latency and Throughput for different Block-Sizes and Threads for Sequential Read

Block size	Threads	Average		Standard Deviation	
		Latency	Throughput	Latency	Throughput
1B	1	0.11267	0.00851	0.00971	0.00076
	2	0.03850	0.02488	0.00312	0.00195
1KB	1	0.10500	9.30116	0.00100	0.08859
	2	0.08800	11.11205	0.00397	0.49053
1MB	1	55.00633	18.17976	0.09315	0.03082
	2	55.23750	18.10412	0.34540	0.11361

Table. 11 Average Latency and Throughput for different Block-Sizes and Threads for Sequential Read

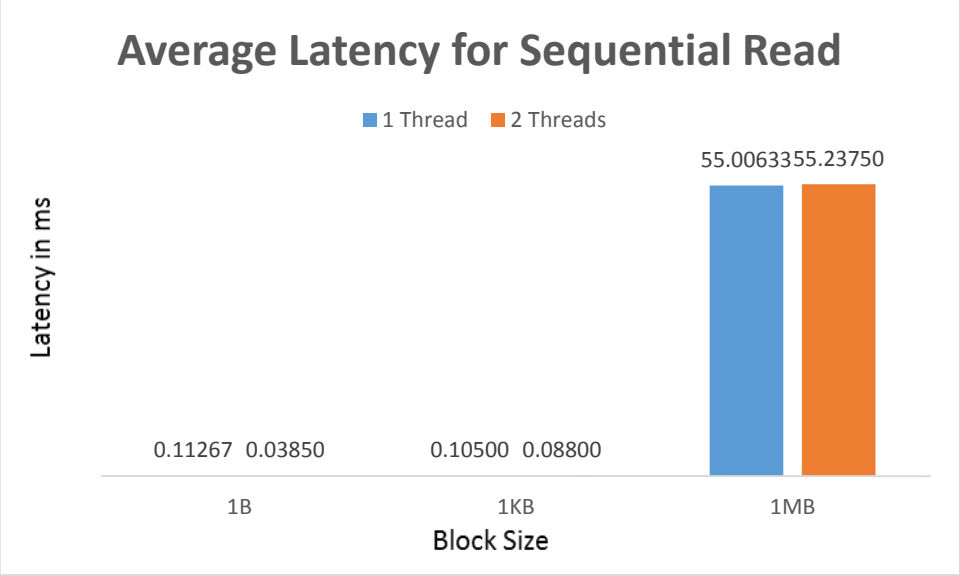


Fig. 15 Average Latency for Sequential Read

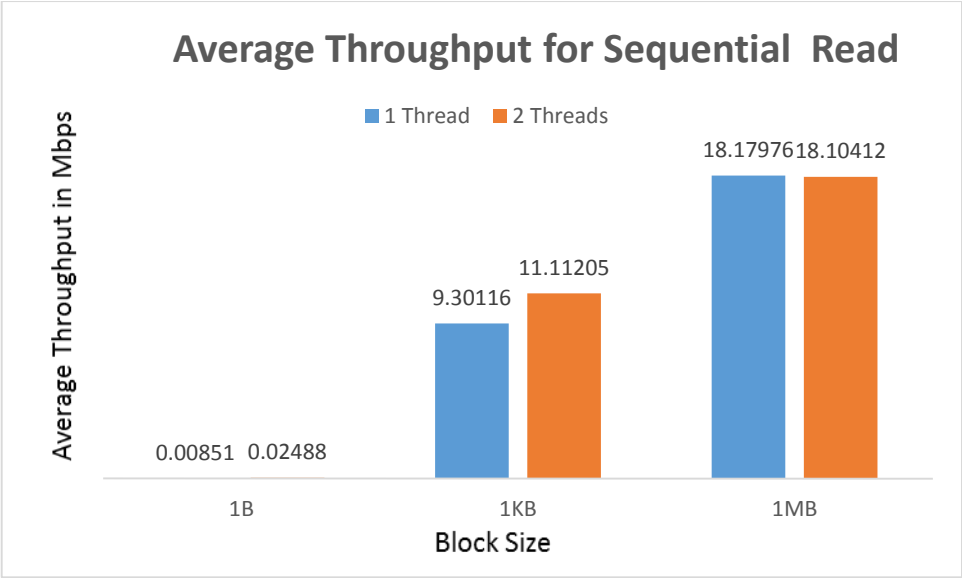


Fig. 16 Average Throughput for Sequential Read

The average latency for the Sequential Read comes out to be **55.12192ms** and Throughput is **18.14194 Mbps**, for a Block size of 1MB**.**

C) Random Write

Block size	Threads	Iterations					
		1		2		3	
		Latency	Throughput	Latency	Throughput	Latency	Throughput
1B	1	1.84000	0.00052	1.87000	0.00051	1.83000	0.00052
	2	1.39900	0.00068	1.41650	0.00067	1.39750	0.00068
1KB	1	1.40400	0.69556	1.41300	0.69113	1.47200	0.66343
	2	1.35600	0.72018	1.38550	0.70484	1.35200	0.72231
1MB	1	24.79200	40.33559	24.84700	40.24631	24.35900	41.05259
	2	24.55650	40.72242	24.63750	40.58853	24.39050	40.99957

Table. 12 Latency and Throughput for different Block-Sizes and Threads for Random Write

Block size	Threads	Average		Standard Deviation	
		Latency	Throughput	Latency	Throughput
1B	1	1.84667	0.00052	0.02082	0.00001
	2	1.40433	0.00068	0.01056	0.00001
1KB	1	1.42967	0.68337	0.03694	0.01741
	2	1.36450	0.71578	0.01830	0.00953
1MB	1	24.66600	40.54483	0.26729	0.44199
	2	24.52817	40.77017	0.12591	0.20964

Table. 13 Average Latency and Throughput for different Block-Sizes and Threads for Random Write

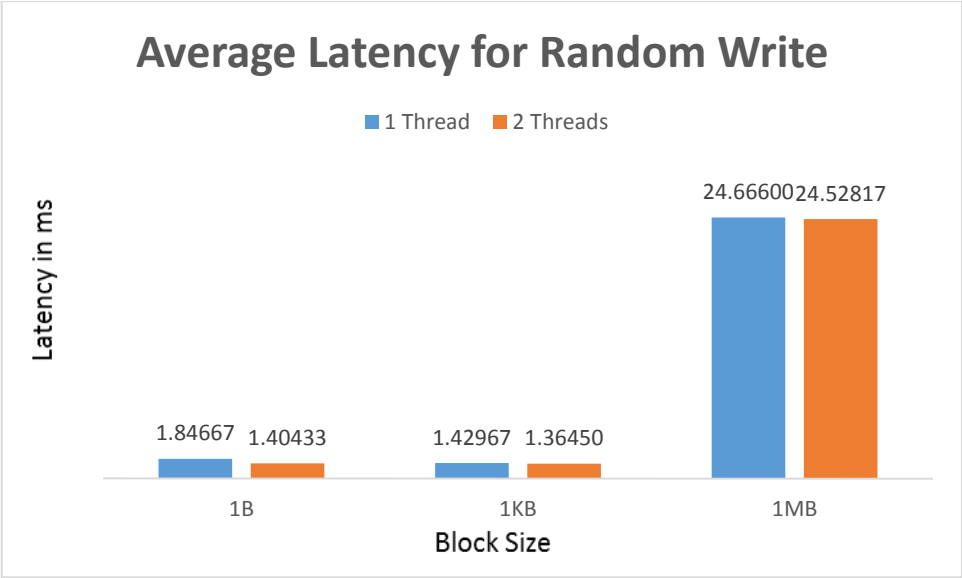


Fig. 17 Average Latency for Random Write

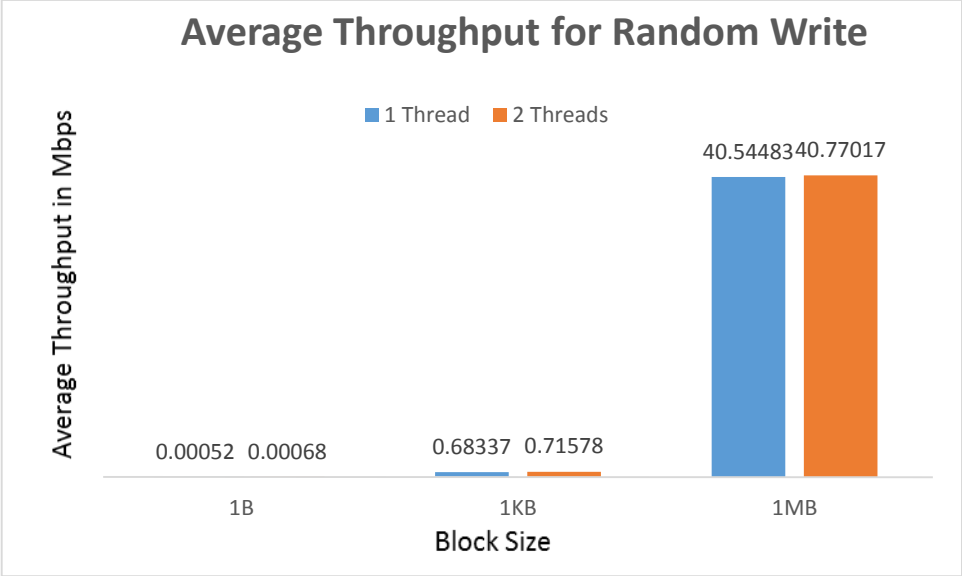


Fig. 18 Average Throughput for Random Write

The average latency for the Random Write comes out to be **24.59708ms** and Throughput is **40.65750 Mbps**, for a Block size of 1MB**.**

D) Random Read

Block size	Threads	Iterations					
		1		2		3	
		Latency	Throughput	Latency	Throughput	Latency	Throughput
1B	1	0.26200	0.00364	0.27400	0.00348	0.27500	0.00347
	2	0.17550	0.00543	0.18250	0.00523	0.18750	0.00509
1KB	1	0.34700	2.81430	0.36000	2.71267	0.37600	2.59724
	2	0.30500	3.20184	0.32150	3.03752	0.30450	3.20710
1MB	1	177.91500	5.62066	181.94700	5.49611	178.58200	5.59967
	2	178.37450	5.60618	181.75750	5.50184	179.16100	5.58157

Table. 14 Latency and Throughput for different Block-Sizes and Threads for Random Read

Block size	Threads	Average		Standard Deviation	
		Latency	Throughput	Latency	Throughput
1B	1	0.27033	0.00353	0.00723	0.00010
	2	0.18183	0.00525	0.00603	0.00018
1KB	1	0.36100	2.70807	0.01453	0.10860
	2	0.31033	3.14882	0.00967	0.09643
1MB	1	179.481	5.57215	2.16122	0.06668
	2	179.764	5.56320	1.77036	0.05455

Table. 15 Average Latency and Throughput for different Block-Sizes and Threads for Random Read

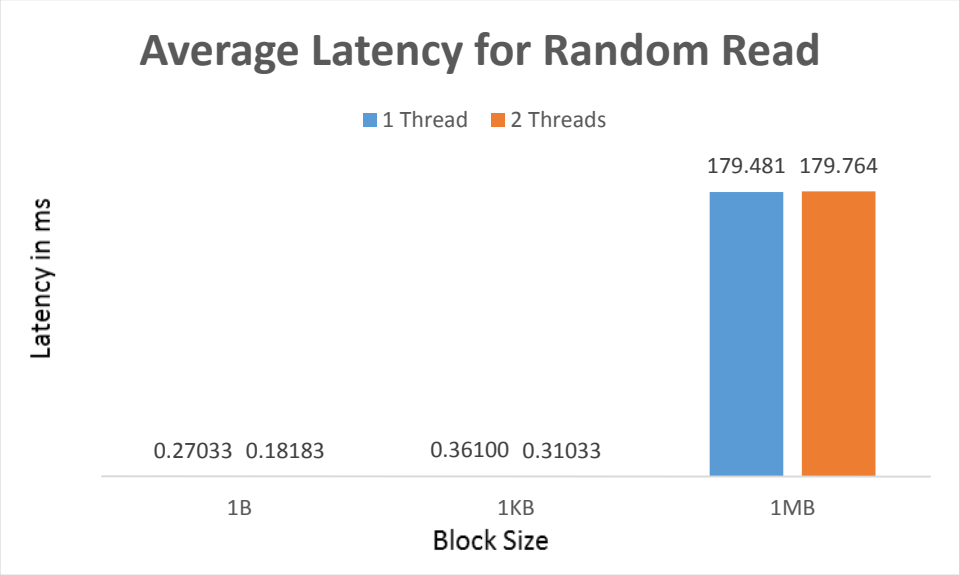


Fig. 19 Average Latency for Random Read

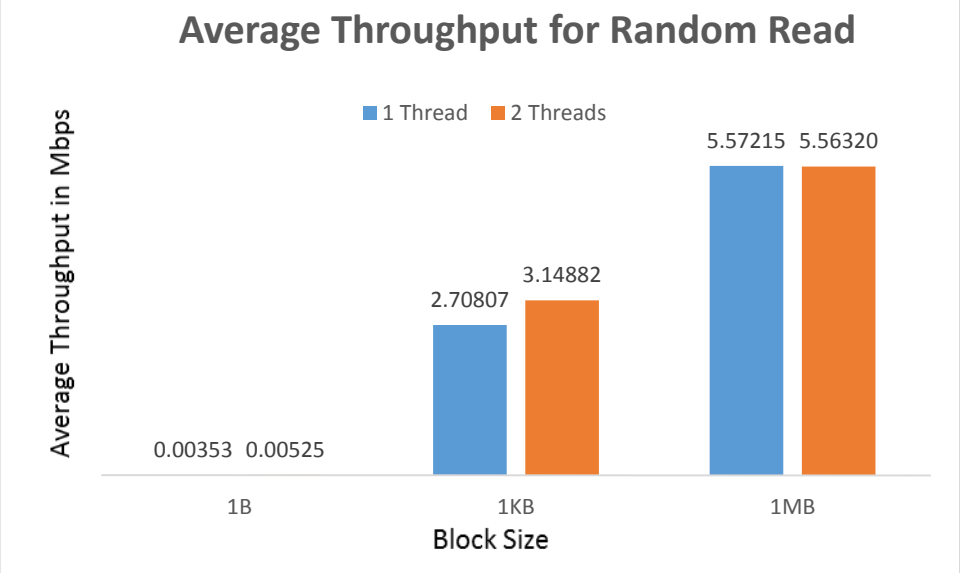


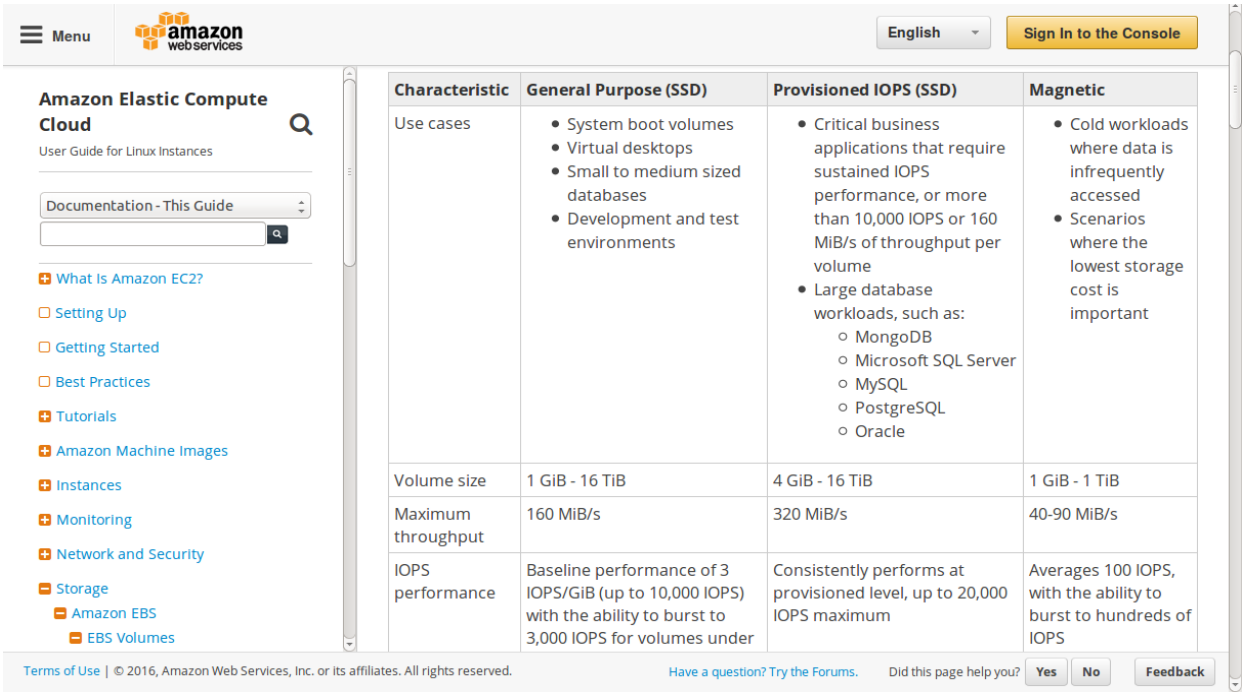
Fig. 20 Average Throughput for Random Read

The average latency for the Random Read comes out to be **179.623 ms** and Throughput is **5.56767 Mbps**, for a Block size of 1MB**.**

Theoretical value

40-90 MiB/s

As we can see from the amazon document mentioned in [3] and a snapshot given below:



Characteristic	General Purpose (SSD)	Provisioned IOPS (SSD)	Magnetic
Use cases	• System boot volumes • Virtual desktops • Small to medium sized databases • Development and test environments	• Critical business applications that require sustained IOPS performance, or more than 10,000 IOPS or 160 MiB/s of throughput per volume • Large database workloads, such as: ○ MongoDB ○ Microsoft SQL Server ○ MySQL ○ PostgreSQL ○ Oracle	• Cold workloads where data is infrequently accessed • Scenarios where the lowest storage cost is important
Volume size	1 GiB - 16 TiB	4 GiB - 16 TiB	1 GiB - 1 TiB
Maximum throughput	160 MiB/s	320 MiB/s	40-90 MiB/s
IOPS performance	Baseline performance of 3 IOPS/GiB (up to 10,000 IOPS) with the ability to burst to 3,000 IOPS for volumes under	Consistently performs at provisioned level, up to 20,000 IOPS maximum	Averages 100 IOPS, with the ability to burst to hundreds of IOPS

Fig. 21 Snapshot of the Amazon Document

Moreover, as one can see in Section 1 with sudo hdparm -tT /dev/sda command I got the throughput as 78.37 MB/sec. Which falls in the range as mentioned in [3].

Hence I have considered the upper bound of 90Mbps as the theoretical value.

The efficiency of my benchmark over Theoretical value are:

- **48.97%** for Sequential Write for 1MB Block Size
- **20.16%** for Sequential Read for 1MB Block Size
- **45.17%** for Random Write for 1MB Block Size
- **6.19%** for Random Read for 1MB Block Size

The efficiency is less because the AWS instance scales up and down according to its need. So, one cannot precisely say about the theoretical throughput of the t2.instance, the value will vary from 40 to 90 Mbps.

4.2 IOZONE Output

```
        ->   OWIN                (32bit)   <-
        ->   Windows (95/98/NT)  (32bit)   <-

[ec2-user@ip-172-31-36-66 current]$
[ec2-user@ip-172-31-36-66 current]$ make linux
cc  -O3  iozone_linux.o libasync.o libbif.o -lpthread \
        -lrt -o iozone
cc  -O3 -Dlinux fileop_linux.o -o fileop
cc  -O3 -Dlinux pit_server.o -o pit_server
[ec2-user@ip-172-31-36-66 current]$
[ec2-user@ip-172-31-36-66 current]$ ./iozone -g# -s 1024
        Iozone: Performance Test of File I/O
                Version $Revision: 3.394 $
                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.
                     Ben England.

        Run began: Thu Feb 11 18:44:10 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 1617808 4470172 12790037 13142265 9655828 4374559 9402175  5782087  9946527  4199201  41464991023084 5 12071103

iozone test complete.
[ec2-user@ip-172-31-36-66 current]$
```

Fig. 22 IOZONE output snapshot

From the IOZONE Benchmark I got the following Throughput for the 4 scenarios for 1MB block-size are:

- Sequential Read: 12944.224 Kbps
- Sequential Write: 1648.862 Kbps
- Random Read: 10039.528 Kbps
- Random Write: 4392.454 Kbps

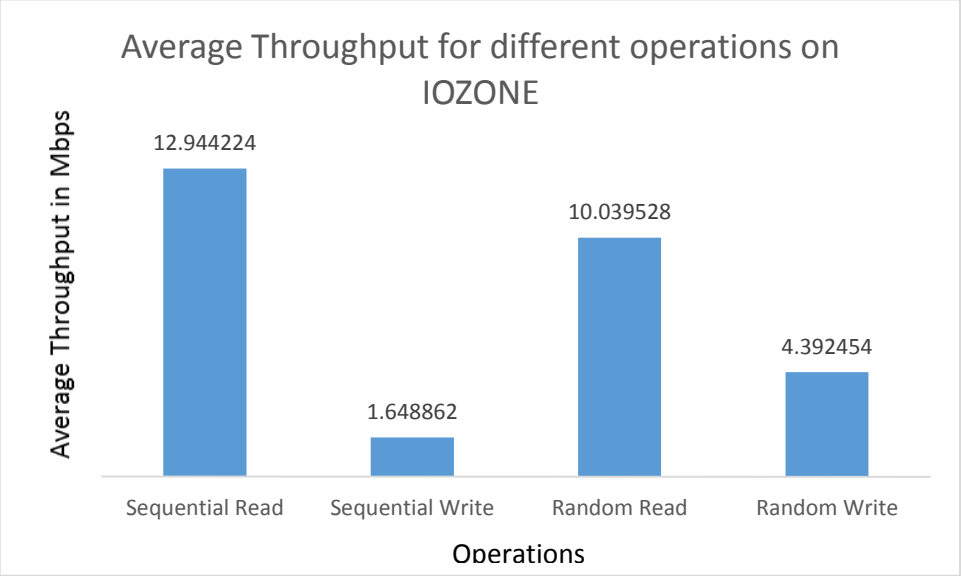


Fig. 23 Average Throughput for different operations on IOZONE

This value are abrupt may be because of the scaling up and down of the AWS. As soon as a big memory is given to be read or write the threshold is exceeded and the resources scales up.

5 References

[1] https://saiclearning.wordpress.com/2014/04/08/how-to-calculate-peak-theoretical-performance-of-a-cpu-based-hpc-system/

[2] https://en.wikipedia.org/wiki/Memory_bandwidth

[3] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html