CS553 Programming Assignment #1

Benchmarking

TEAM MEMBERS:

- **VAISHNAVI GUDUR (A20384900)**
- **SATYA SUVEEN REDDY MEKALA(A20379568)**

DESIGN DOCUMENT

The assignment involves benchmarking different parts of a computer system, from the CPU, GPU, Memory, Disk, and Network.
The basic design and logic implemented for each of the above mentioned 5 components is explained below:

CPU BENCHMARKING:

1. Program Design:

The main goal of this experiment is to obtain the CPU speed in Giga IOPS and Giga FLOPS by performing arithmetic operations using Integers and Double precision floating point numbers respectively, with varying levels of concurrency.

- The code is designed in such a way that there are 3 methods defined including the main function; main function defines the number of threads (1,2,4 and 8) and calls the second function for each of the number of threads.
- The second function includes thread creation, pthread_join() function suspends the execution of calling thread unless the target thread is terminated. Also calculates the time required to perform the operations and processor speed.
- The third function performs the arithmetic operations.
- To calculate IOPS and FLOPS, 30 arithmetic operations involving addition and subtraction are performed for 10^9 times on both integers and double precision floating point numbers.
- The time required to complete the execution is recorded at each concurrency level by recording the start time at the start of the operation and end time after completion of the operation, and calculating the difference thereafter.
- In case of more than 1 thread, the total time of execution of all the threads is recorded.
- The speed in IOPS and GFLOPS is calculated as the product of number of times the operations performed times the number of arithmetic operations performed divided by the execution time elapsed after performing those operations.
- The same program is run with 1 thread, 2 threads, 4 threads and 8 threads.

- The program is designed to support strong scaling; the number of instructions is kept fixed for the entire code.However, as the number of threads increase, the amount of work per thread is decreased such that each thread performs equal share of work.

2. Performance:

The following formula was used to calculate the processor speed in GIOPS and GFLOPS:

$$\text{Speed} = \frac{\textit{Number of arithmetic instructions} * \textit{Number of times operations performed}}{\textit{Execution time} * 10^{9}}$$

Theoretical Speed:
The theoretical speed of CPU can be calculated as,

GHz * Cores * IPC.

Chameleon Openstack instance has **2.29 GHz, 2 Cores and 16 IPC for Intel Core Processor (Haswell).** I obtained this information using 'lscpu' command on the instance.

Since the processor info was stated as Intel core(Haswell) of family 6 Series 60 stepping 1, the instruction per cycle was decided as 8 IPC .
Thus, theoretical peak performance for 2 threads (1 thread/core) = 2.29 * 2 * 8
= **36.64 GFLOPS**

The results obtained after repeatedly performing the experiment are as follows:
GIOPS:

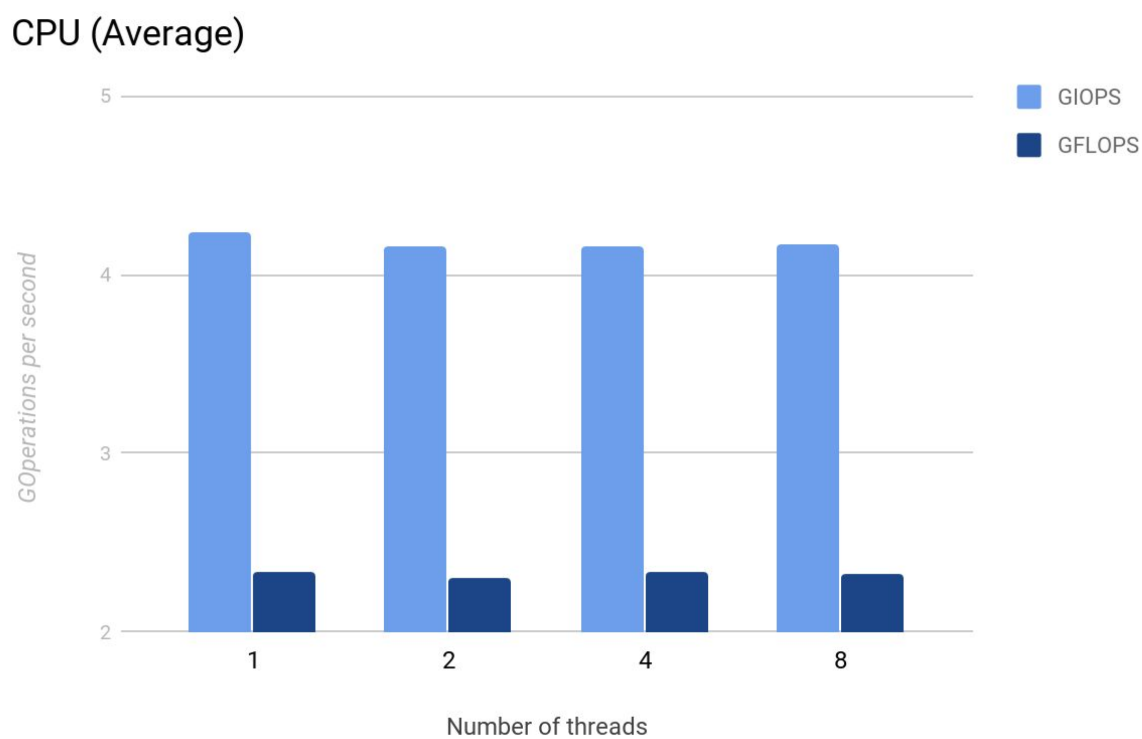
Sr. No.	1 thread		2 threads		4 threads		8 threads	
	GIOPS	TIME	GIOPS	TIME	GIOPS	TIME	GIOPS	TIME
1.	4.189944	7.16	4.010695	7.48	4.195804	7.15	4.137931	7.25
2.	4.231312	7.09	4.160888	7.21	4.149378	7.23	4.243281	7.07
3.	4.297994	6.98	4.304161	6.97	4.115226	7.29	4.137931	7.25

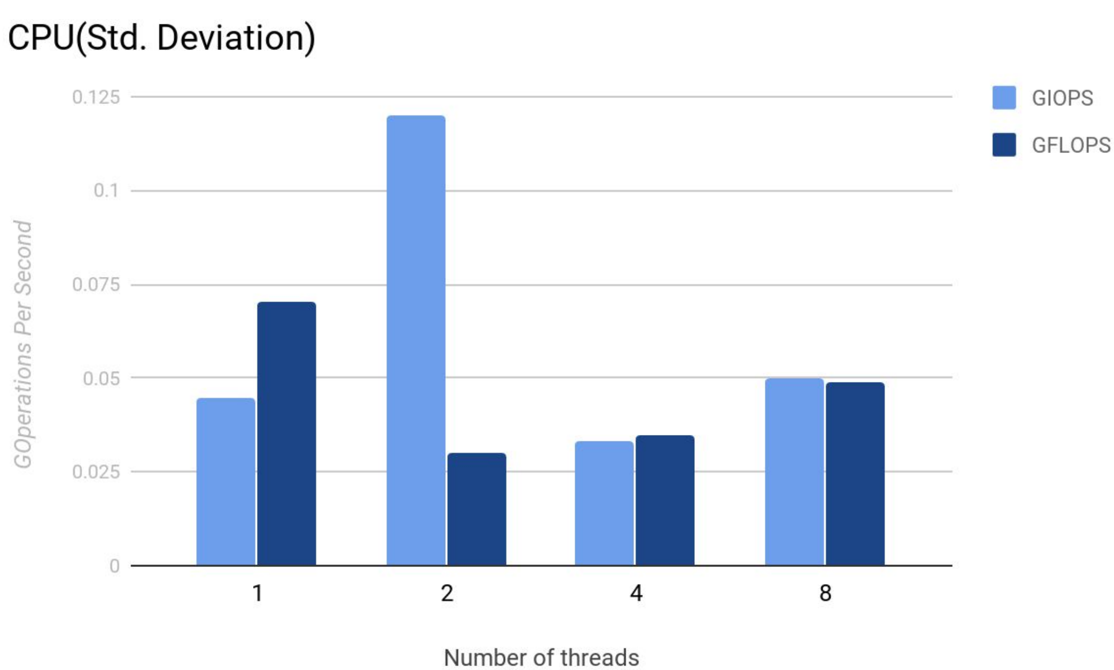
Average	**4.23975**	**7.08**	**4.158581**	**7.22**	**4.153469**	**7.22**	**4.173048**	**7.19**
Standard Deviation	**0.044513**	**0.07**	**0.119818**	**0.21**	**0.033023**	**0.06**	**0.049662**	**0.08**

GFLOPS:

Sr. No.	**1 thread**		**2 threads**		**4 threads**		**8 threads**	
	GFLOPS	**TIME**	**GFLOPS**	**TIME**	**GFLOPS**	**TIME**	**GFLOPS**	**TIME**
1.	2.327386	12.89	2.305919	13.01	2.382844	12.59	2.388535	12.56
2.	2.260739	13.27	2.259036	13.28	2.320186	12.93	2.281369	13.15
3.	2.431118	12.34	2.331002	12.87	2.302379	13.03	2.288330	13.11
Average	**2.339747**	**12.83**	**2.298652**	**13.05**	**2.335136**	**12.85**	**2.319411**	**12.94**
Standard Deviation	**0.070104**	**0.38**	**0.029826**	**0.17**	**0.034508**	**0.188**	**0.048960**	**0.270**

The results obtained of GIOPS and GFLOPS are visualized in the below chart:


CPU (Average)
GIOPS
GFLOPS
GOperations per second
5
4
3
2
1
2
4
8
Number of threads


CPU(Std. Deviation)
GIOPS
GFLOPS
GOperations Per Second
0.125
0.1
0.075
0.05
0.025
0
1
2
4
8
Number of threads

Efficiency:
Efficiency can be calculated as percentage of actual performance by theoretical performance.

$$\text{Efficiency} = \frac{Actual\ \ Performance}{Theoretical\ Performance} * 100$$

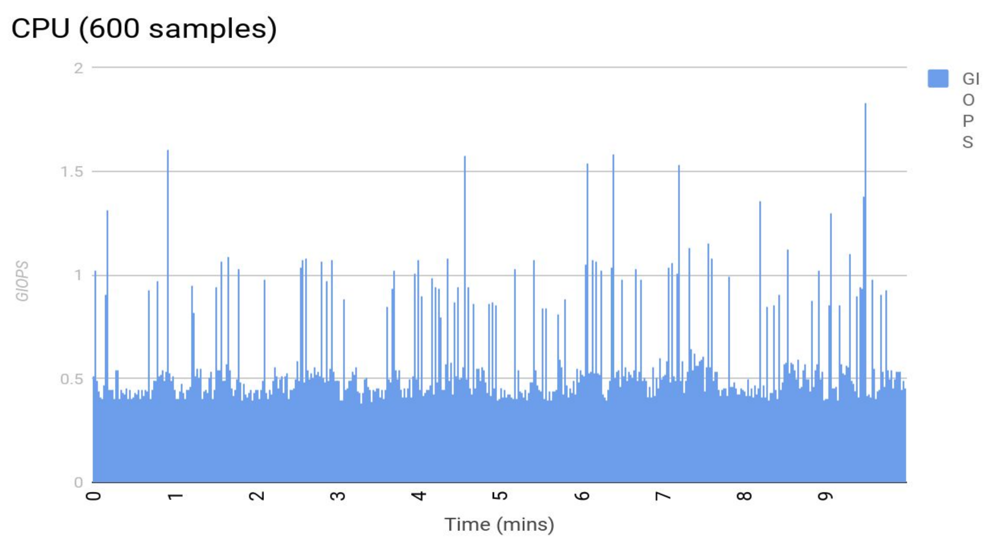
Thus, efficiency in GFLOPS can be calculated as,

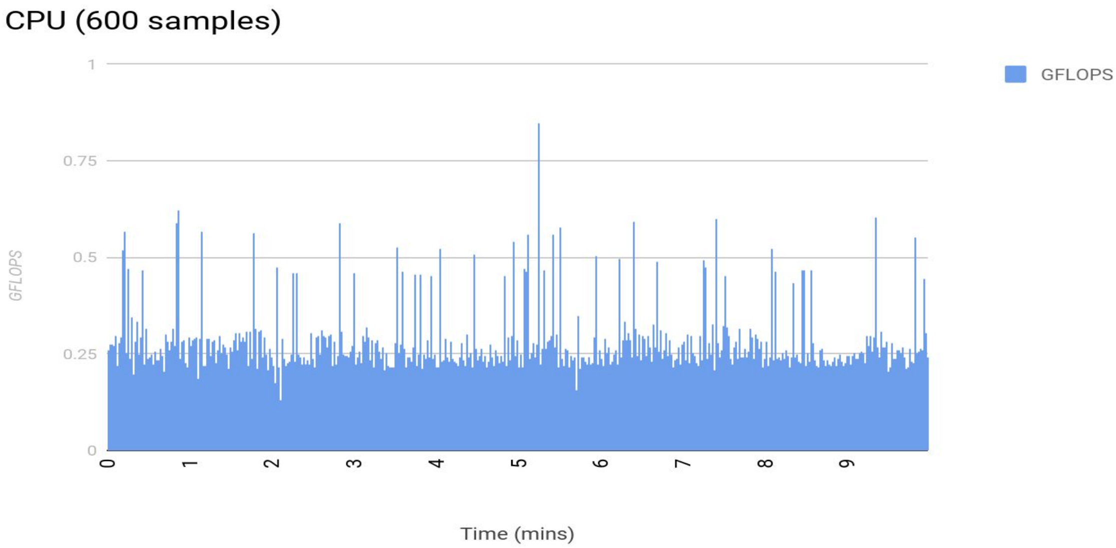
= (2.3397/36.64) * 100

= **6.39%**

CPU BENCHMARKING FOR 600 SAMPLES:

- As a separate experiment, the benchmark code is run on double precision floating point and integer instructions, 8 threads for a 10-minute period for each one, and samples were taken every second on how many instructions per second were achieved during the experiment.
- There are separate code files written for integer instructions and double precision floating point instructions.
- The 600 samples are recorded and the values are plotted with time (0 to 10 min) on the x-axis and FLOPS/IOPS on the y-axis.

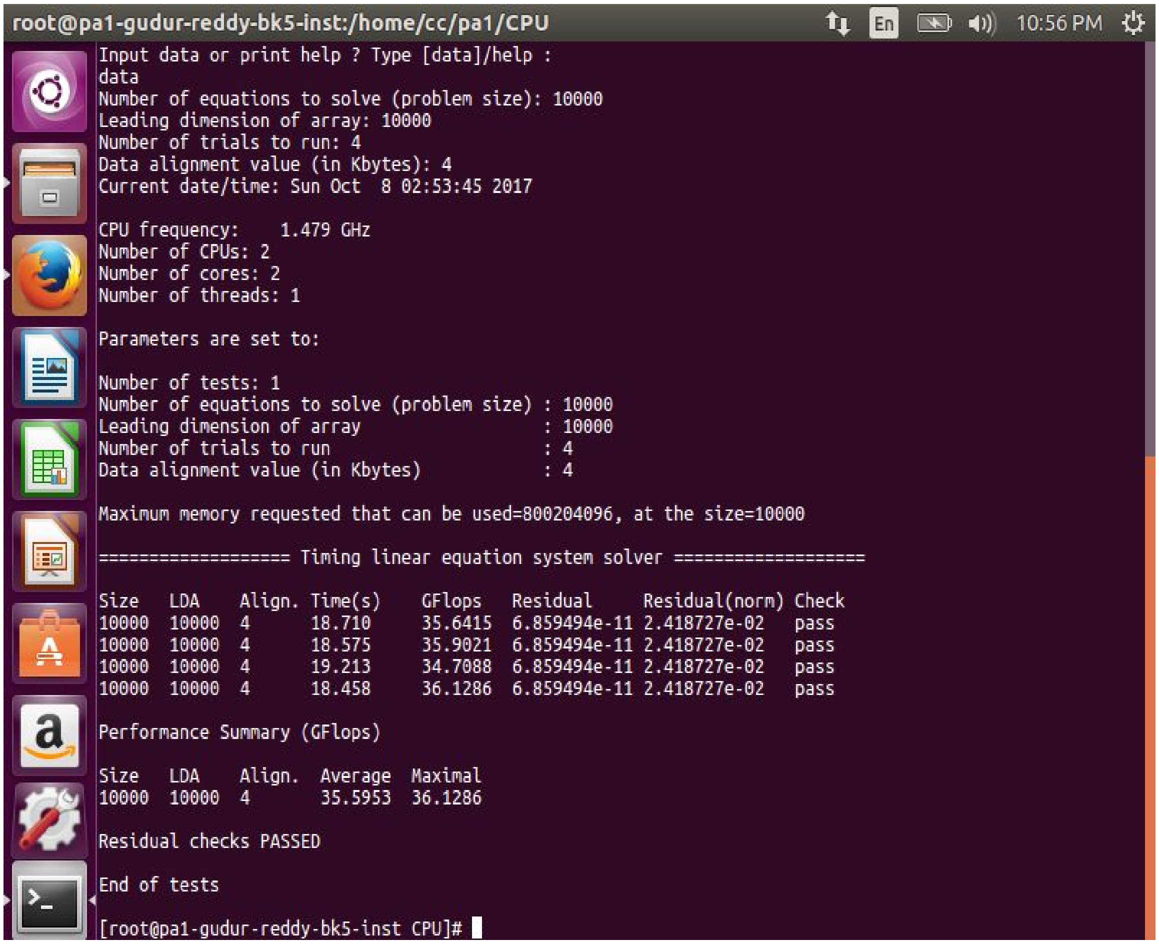




LINPACK BENCHMARK:

- The Linpack benchmark was run across all the nodes of the instance VM and the best GFLOPS performance was recorded.
- The efficiency in the speed achieved compared to both theoretical performance calculated and the benchmark results obtained from our code is reported.

The screenshot below shows the results of Linpack Benchmark run on Chameleon openstack instance:



The parameters set are as follows:
Number of CPUs : 2
Number of Cores : 2
Number of threads : 1

The average peak performance for a problem size of 10000 was found to be **35.5953 GFLOPS.**
Efficiency achieved = $(Linpack\ performance/Theoretical\ speed) * 100$

= (35.5953/36.64) * 100

= **97.15 %**

Efficiency achieved by Linpack benchmark is much higher than efficiency obtained by our code (6.39%) and is close to theoretical performance obtained.

Thus, we conclude that Linpack benchmarks are much better than those obtained in our benchmark. As per Netlib.org, number of factors affect the performance of a processor including problem size,load on the system, accuracy of the clock, compiler options, version of the compiler, size of cache, bandwidth from memory, amount of memory, etc. The problem size used in Linpack is 10000 which is 10^5 times lesser than in our benchmark. Also, Linpack benchmark code uses matrix multiplications which increases the performance by a substantial amount. Linpack itself is built on BLAS, which is a high performance package for computations. Linpack also makes use of AVX instructions which boosts the speed.

3. Conclusion :

Thus, speed increase with increase in number of threads.

The performance achieved by our code was below theoretical speed obtained. Number of factors are responsible : lack of use of AVX instructions being one of them.

Linpack achieved the best CPU speed which is almost equal to theoretical speed.

4. Improvements :

1. GUI can be implemented for real-time tracking of processor speed.
2. Complex operations involving matrix multiplications

MEMORY BENCHMARKING:

1. **Program Design**

The Memory benchmarking program is designed as follows.
The program takes a user input of the block size and the number of threads to run the program on.
It consists of 16 functions, out of which, 4 functions correspond to the block size and is used for the calculation of the latency and throughput.
The other functions (4*3) = 12 functions for read+write and sequential write and random write (using memset(), memcpy()).
The last function is createFile() which is used to create a 1GB file to perform the operations on.

2. **Performance**

The throughput and latency values obtained are as follows:

Throughput - 1 thread (MBps):

SNo	Read + Write				Sequential Write				Random Write			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB
1	800	3289.473	7876.923	2782.608	800	3906.250	20480	4000	114.28	2717.391	4654.545	4000
2	800	3289.473	7314.285	2909.090	400	3906.250	17066.67	4000	133.33	2840.909	4876.1904	4571.428
Avg	**800**	**3289.473**	**5393**	**2845.85**	**600**	**3906.250**	**18773.34**	**4000**	**123.80**	**2779.15**	**4765.377**	**4285.784**
SD	**0**	**0**	**281.319**	**63.241**	**200**	**0**	**1706.665**	**0**	**9.525**	**61.759**	**110.8227**	**285.714**

Latency - 1 thread (usec) :

SNo	Read + Write				Sequential Write				Random Write			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB
1	1250	304	126.953	359.375	1250	256	48.83	250	0.0000009	368	214.843	250
2	1250	304	136.718	343.750	2500	256	58.60	250	0.0000007	352	205.078	218.750
Avg	**1250**	**304**	**131.84**	**351.562**	**1875**	**256**	**53.72**	**250**	**0.00000008**	**360**	**209.96**	**234.375**
SD	**0**	**0**	**4.8825**	**7.8125**	**625**	**0**	**4.885**	**0**	**0**	**8**	**4.8825**	**15.625**

Throughput - 2 threads (MBps):

SNo	Read + Write				Sequential Write				Random Write			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB
1	800	3289.473	6826.67	2909.090	400	3676.470	17066.67	4266.67	133.33	2040.909	4654.545	3764.705
2	800	3289.473	6826.67	3047.619	800	3676.470	20480	4000	114.285	2976.190	4654.545	4266.67
Avg	**800**	**3289.473**	**6826.67**	**2978.355**	**600**	**3676.470**	**18773.34**	**4133.34**	**123.80**	**2508.55**	**4654.545**	**4015.69**
SD	**0**	**0**	**0**	**69.2645**	**200**	**0**	**1706.665**	**133.335**	**9.5225**	**467.6405**	**0**	**250.9825**

Latency - 2 threads (usec) :

SNo	Read + Write				Sequential Write				Random Write			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB

1	1250	304	146	343.75	2500	272	58.593	234.375	0.00000007	352	214.843	265.625
2	1250	304	146	328.125	1250	272	48.82	250	0.00000009	336	214.843	234.375
Avg	**1250**	**304**	**146**	**336**	**1875**	**272**	**53.70**	**242.19**	**0.000000085**	**344**	**214.843**	**250**
SD	**0**	**0**	**0**	**7.8125**	**625**	**0**	**4.885**	**7.8125**	**0**	**8**	**0**	**15.625**

Throughput - 4 threads (MBps):

SNo	Read + Write				Sequential Write				Random Write			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB
1	800	3289.473	7314.285	2909.090	800	3289.473	20480	4000	100	2604.16	4452.174	4000
2	800	3125	7314.285	2782.608	400	3906.25	17066.67	4000	133.33	2976.190	4452.174	3764.705
Avg	**800**	**3207.23**	**7314.285**	**2845.849**	**600**	**3597.861**	**18773.335**	**4000**	**116.67**	**2790.175**	**4452.174**	**3882.3585**
SD	**0**	**82.2365**	**0**	**63.241**	**200**	**308.3885**	**1706.665**	**0**	**16.665**	**186.015**	**0**	**117.6475**

Latency - 4 threads (usec) :

SNo	Read + Write				Sequential Write				Random Write			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB
1	1250	304	136.718	343.75	1250	304	48.828	250	0.000000010	384	224.609	250
2	1250	320	136.718	359.375	2500	256	585.93	250	0.00000007	336	224.609	265.62
Avg	**1250**	**312**	**136.718**	**351.56**	**1875**	**280**	**317.38**	**250**	**0.000000085**	**360**	**224.609**	**257.81**
SD	**0**	**8**	**0**	**7.8125**	**625**	**24**	**268.551**	**0**	**0**	**24**	**0**	**7.81**

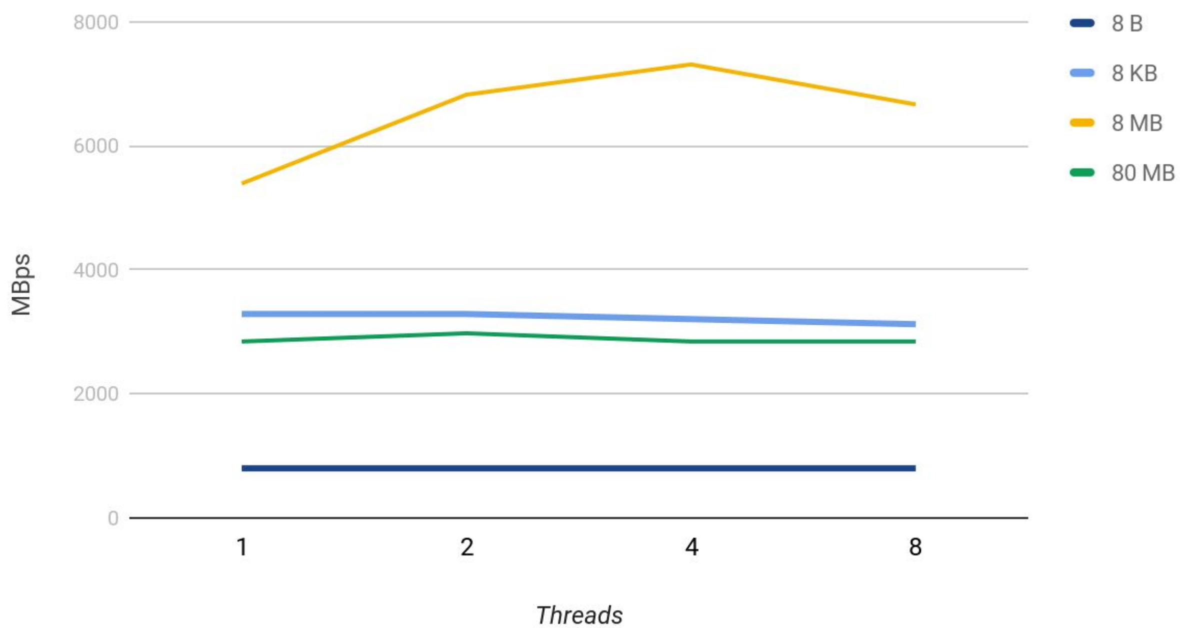
Throughput - 8 threads (MBps):

SNo	Read + Write				Sequential Write				Random Write			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB
1	800	3125	7314.285	2909.090	400	3906.250	17066.67	4000	114.285	2840.909	4654.545	4000
2	800	3125	6023.529	2782.608	400	3906.250	14628.571	4000	133.333	3289.273	5120	3764.705
Avg	**800**	**3125**	**6668.907**	**2845.85**	**400**	**3906.250**	**15847.62**	**4000**	**123.809**	**3065.091**	**4887.272**	**3882.352**
SD	**0**	**0**	**645.378**	**63.241**	**0**	**0**	**1219.0495**	**0**	**9.524**	**224.182**	**232.7275**	**117.647**

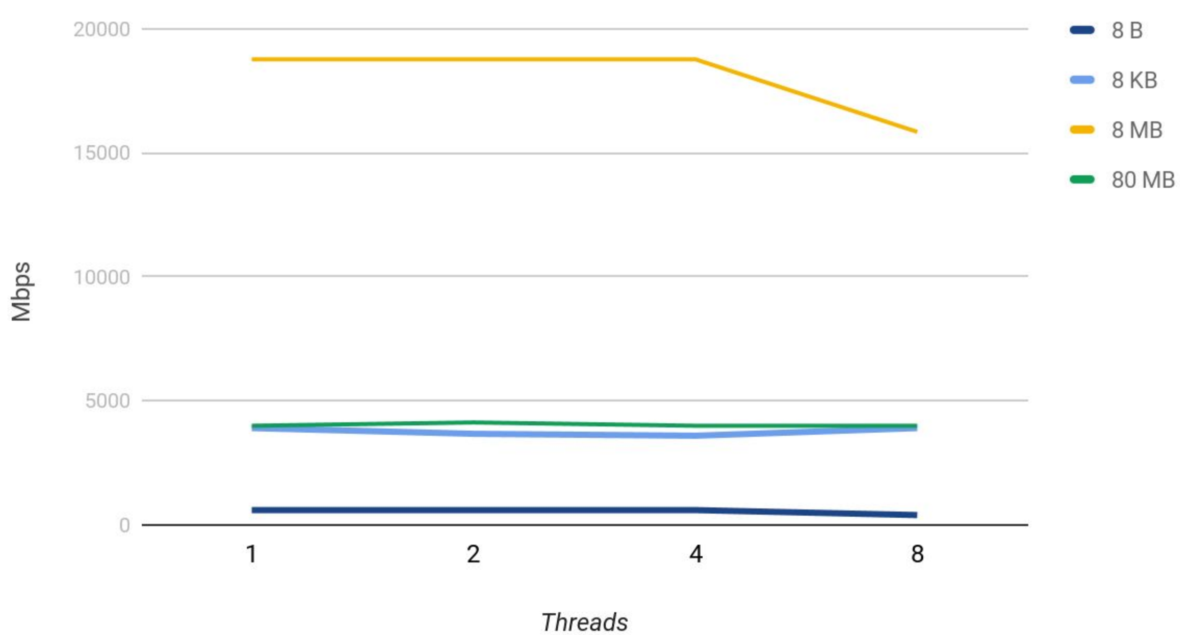
Latency - 8 threads (usec) :

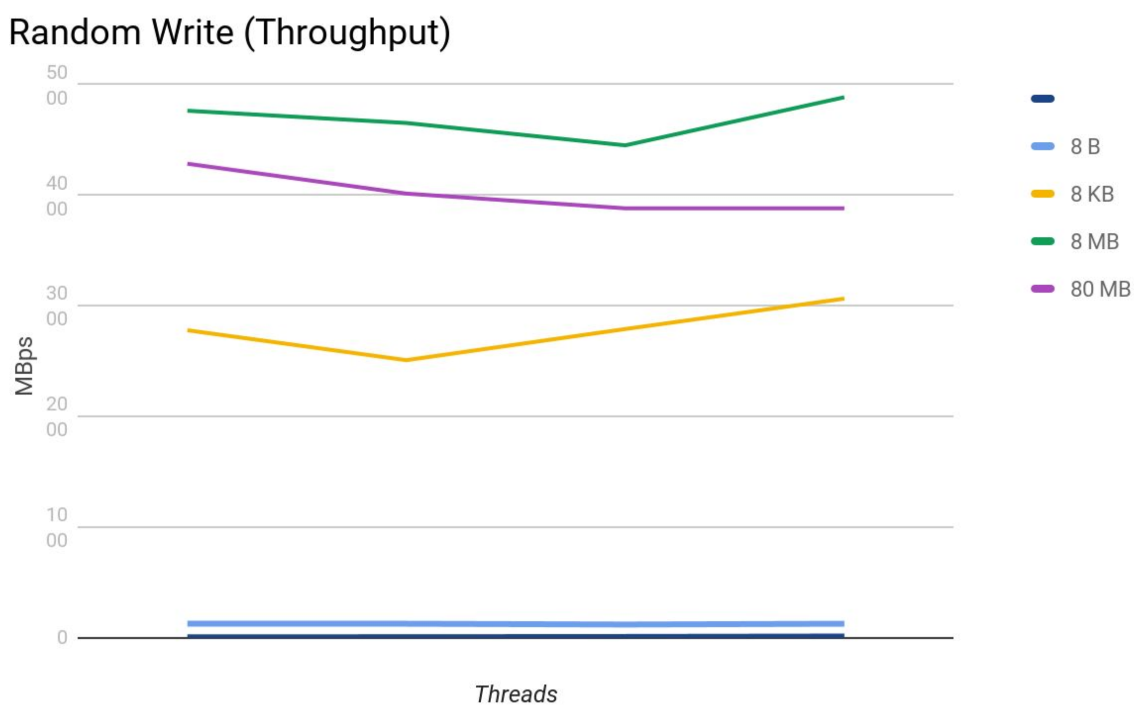
SNo	Read + Write				Sequential Write				Random Write			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB
1	1250	320	136.718	343.750	2500	256	58.593	250	0.00000009	352	214.843	250
2	1250	320	166.015	359.375	2500	256	68.36	250	0.00000007	304	195.312	265.625
Avg	**1250**	**320**	**151.367**	**351.562**	**2500**	**256**	**63.48**	**250**	**0.000000085**	**328**	**205.078**	**257.813**
SD	**0**	**0**	**14.6485**	**7.8125**	**0**	**0**	**4.8835**	**0**	**0**	**24**	**9.7655**	**7.8125**

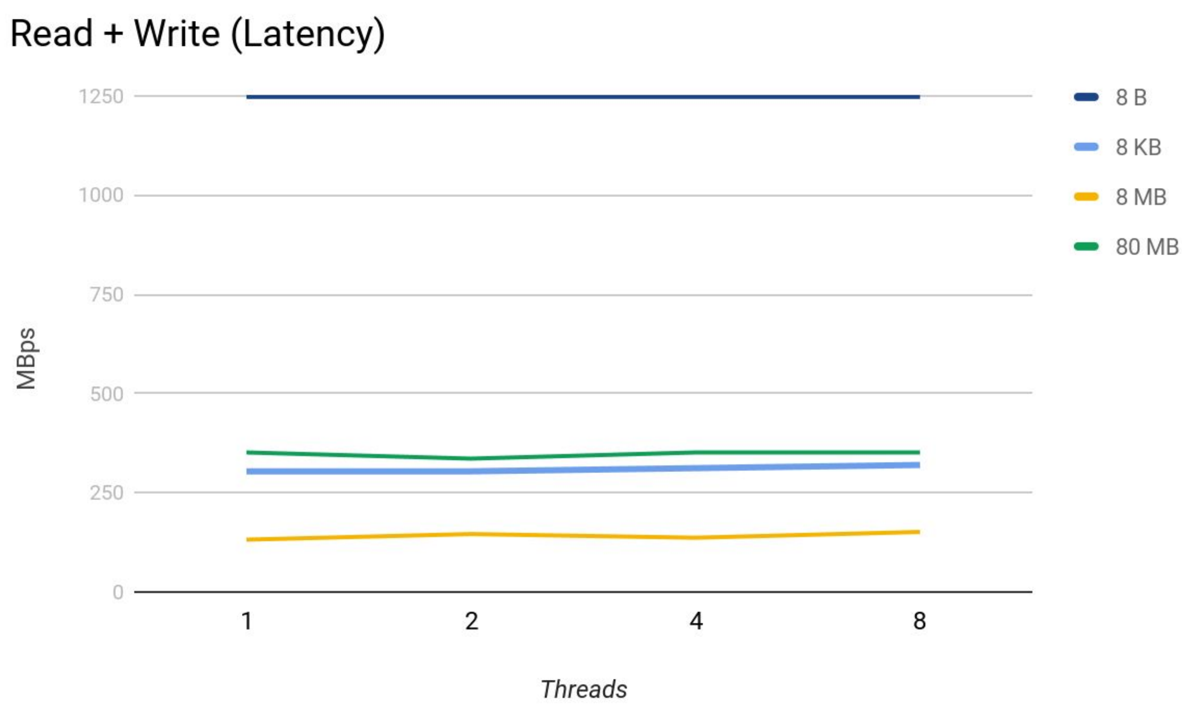
Read + Write (Throughput)

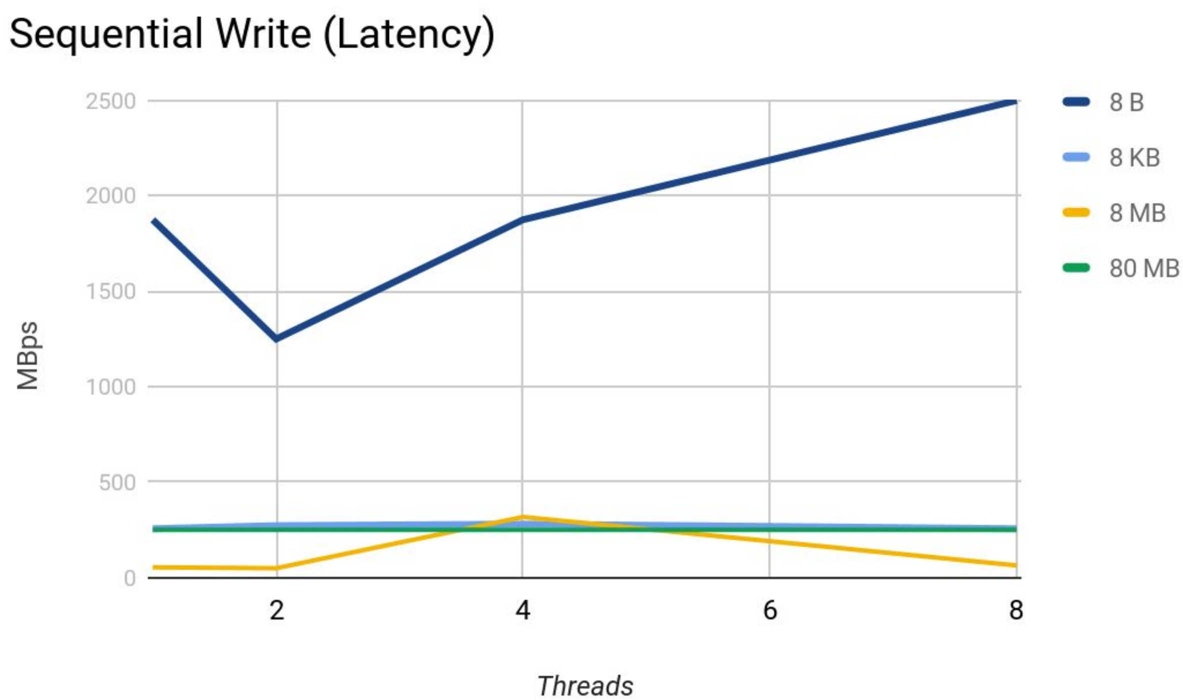


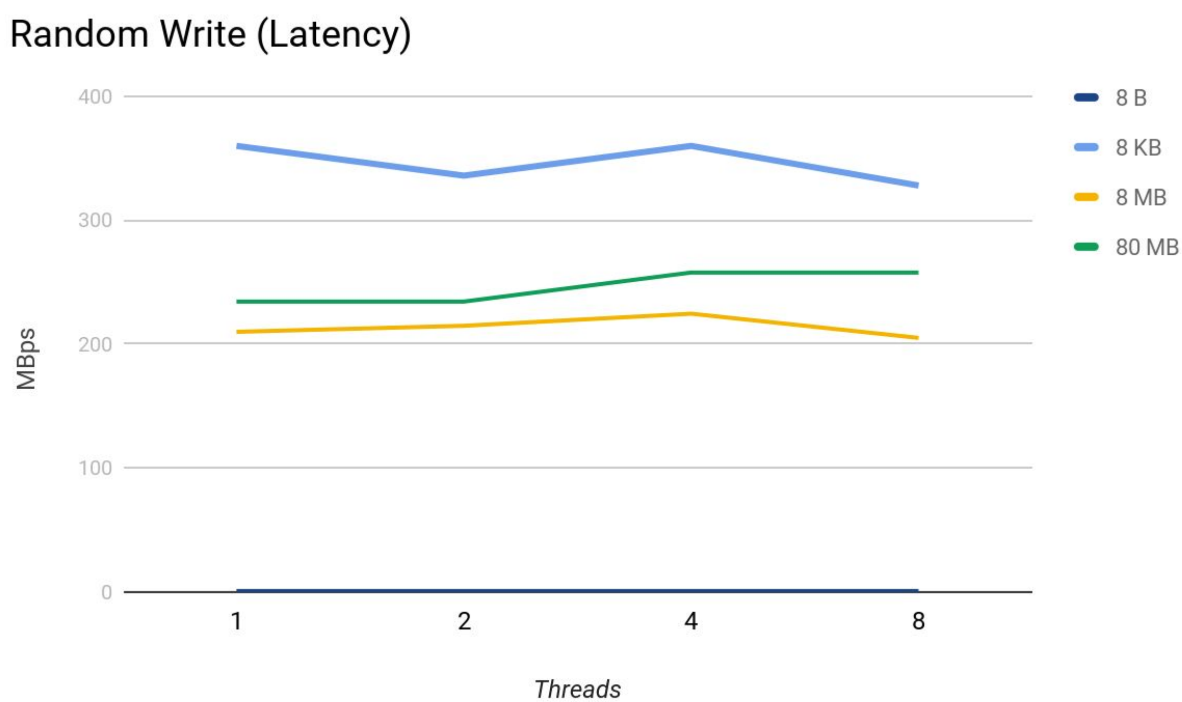
Sequential Write (Throughput)




Random Write (Throughput)
50
00
40
00
30
00
20
00
10
00
0
MBps
Threads
8 B
8 KB
8 MB
80 MB


Read + Write (Latency)
1250
1000
750
500
250
0
MBps
1
2
4
8
Threads
8 B
8 KB
8 MB
80 MB


Sequential Write (Latency)
MBps
2500
2000
1500
1000
500
0
2
4
6
8
Threads
8 B
8 KB
8 MB
80 MB


Random Write (Latency)
MBps
400
300
200
100
0
1
2
4
8
Threads
8 B
8 KB
8 MB
80 MB

Theoretical Memory Bandwidth :

Theoretical memory bandwidth can be roughly calculated as ,

= (2 * dimmSpeed of RAM(DDR3) * max_width of the packet)/8

= (2 * 800Mhz * 64)/8

= **12800 MBps**

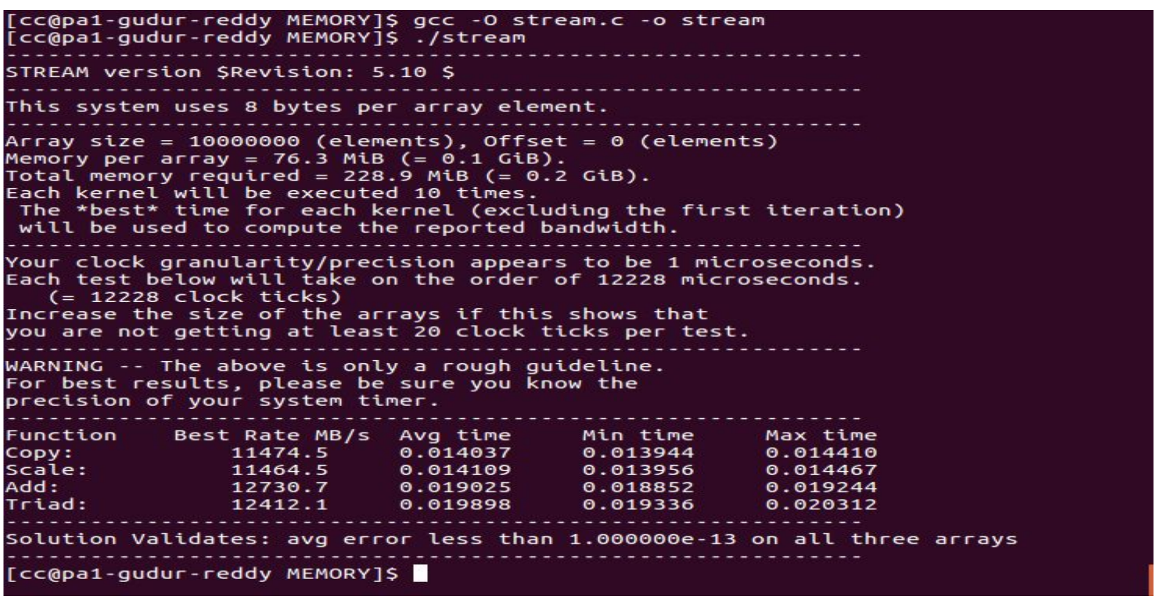
From the Hardware description of the Chameleon cloud, the network bandwidth is known as 100Gbps=12500MBps

Efficiency achieved = (7314.285/12500) * 100

= **58.514%**

STREAM BENCHMARK:

The stream benchmark was run on openkvm instance. The screenshot of the results are as follows:



```
[cc@pa1-gudur-reddy MEMORY]$ gcc -O stream.c -o stream
[cc@pa1-gudur-reddy MEMORY]$ ./stream
-------------------------------------------------------------
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 12228 microseconds.
   (= 12228 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:           11474.5     0.014037     0.013944     0.014410
Scale:          11464.5     0.014109     0.013956     0.014467
Add:            12730.7     0.019025     0.018852     0.019244
Triad:          12412.1     0.019898     0.019336     0.020312
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
[cc@pa1-gudur-reddy MEMORY]$
```

Thus, the best performance obtained using stream benchmark is **11474 MB/s** using stream benchmark.

Efficiency = (stream performance/theoretical throughput) * 100

= (11474/12500) * 100

= **91.79%**

Thus, the efficiency obtained by stream benchmark is high, close to theoretical value.

3. Conclusion :

Thus, best performance is achieved at concurrency level 4.
Also, the sequential access is faster than the random access because the overhead of accessing any random memory and seeking the file pointer to that random location is avoided.

4. Improvements :

1. GUI can be implemented.
2. Efficient memory handling functions can be implemented.

<u>DISK BENCHMARKING:</u>

1. Program Design

The Disk program is designed as follows.
The program takes a user input of the block size and the number of threads to run the program on.
It consists of 17 functions, out of which, 4 functions correspond to the block size and is used for the calculation of the latency and throughput.
The other functions (4*3) = 12 functions for read+write and sequential read and random read(using fread(), fwrite() and fseek()).
The last function is createFile() which is used to create a 10GB file to perform the operations on.

2. Performance

The throughput and latency values obtained are as follows:

Throughput - 1 thread (MBps):

SNo	Read + Write				Sequential Read				Random Read			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB
1	9.765625	64.873	410.2564	49.2610	19.53125	98.39	1777.778	49.8132	26.041667	86.38	941.1764	49.8132
2	11.38	59.393	489.9782	48.134	22.94	93.06	1834.9	47.846	30.728	85.57	1067.298	46.948
Avg	**10.57**	**62.133**	**450.1141**	**48.6975**	**21.24**	**95.725**	**1806.339**	**54.15**	**28.38**	**85.975**	**1004.237**	**48.3806**

Throughput - 2 threads (MBps):

SNo	Read + Write				Sequential Read				Random Read			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB
1	11.95	78.125	470.588	46.7836	21.478	83.6893	761.904	48.721	35.894	78.125	1066.667	49.93757
2	11.38	74.987	493.2963	48.305	19.383	80.398	794.37	52.7856	29.399	74.834	1205.76823	54.89342
Avg	**11.67**	**76.556**	**481.942**	**47.5443**	**20.43**	**82.04365**	**778.137**	**50.7533**	**32.64**	**76.4795**	**1136.217**	**52.41546**

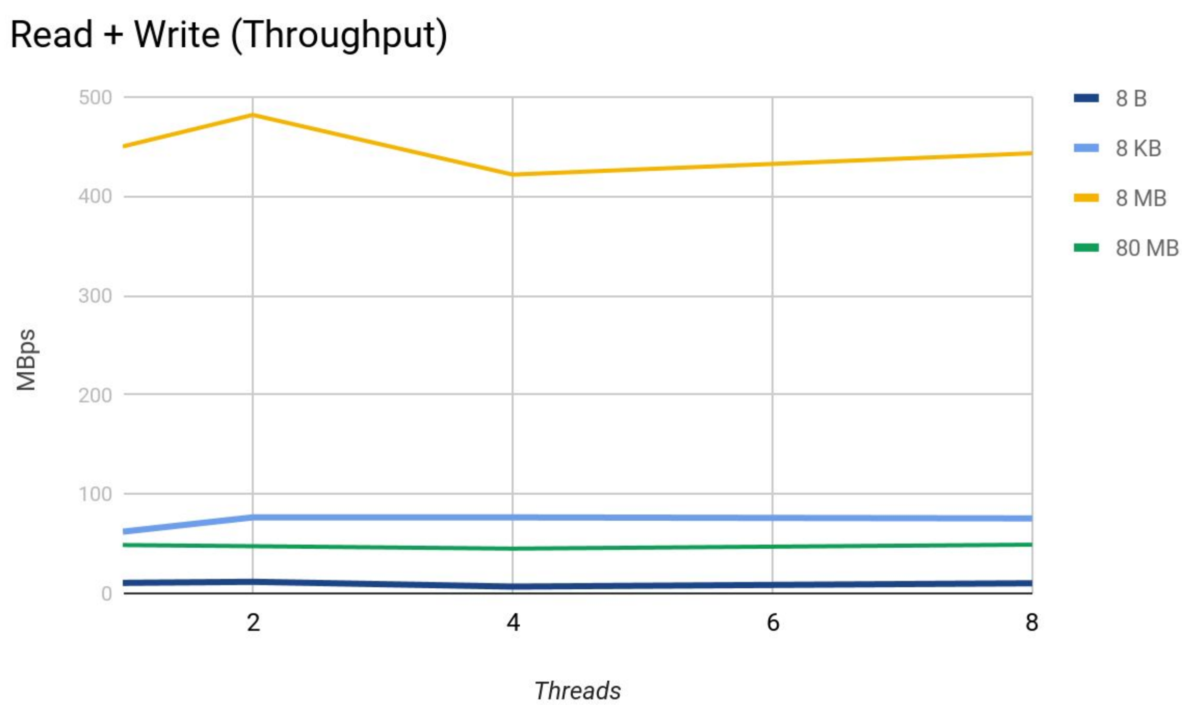
Throughput - 4 threads (MBps):

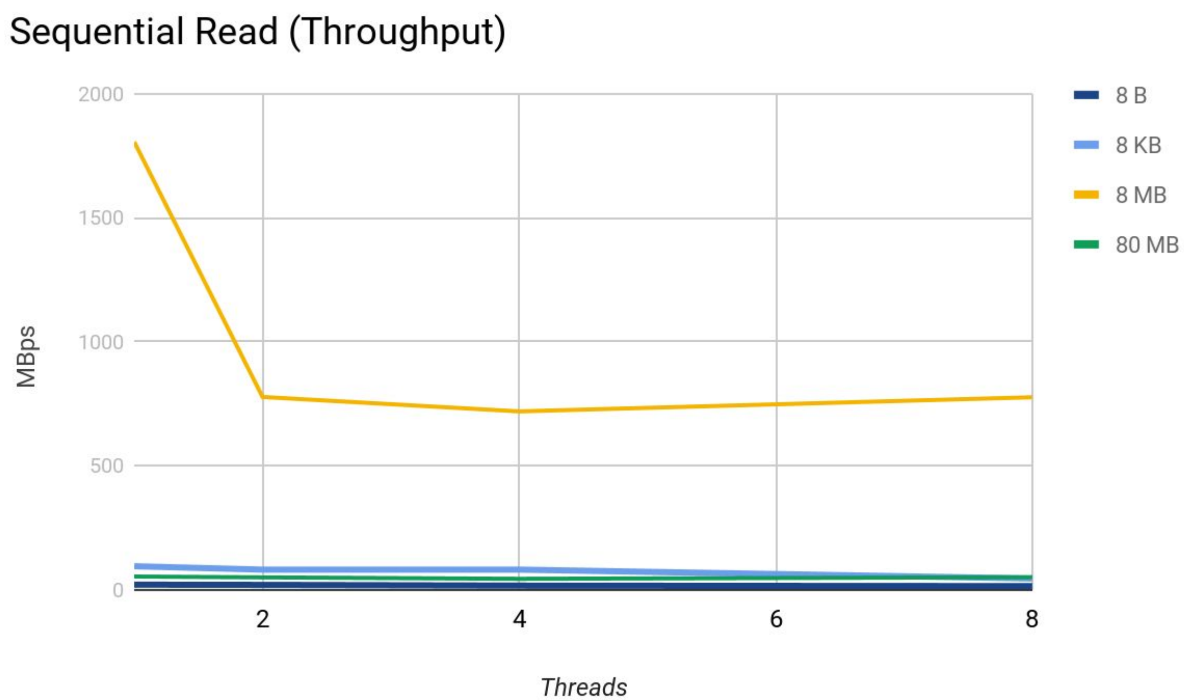
SNo	Read + Write				Sequential Read				Random Read			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB
1	8.373	78.125	457.14	46.457	16.389	83.6893	800.00	46.403	10.38	78.125	1333.3	49.200
2	5.278	74.987	386.4528	43.7289	18.389	80.398	639.924	43.562	7.378	74.834	1194	49.458
Avg	**6.83**	**76.556**	**421.7964**	**45.092**	**17.389**	**82.04365**	**719.962**	**44.9825**	**8.88**	**76.4795**	**1263.665**	**49.329**

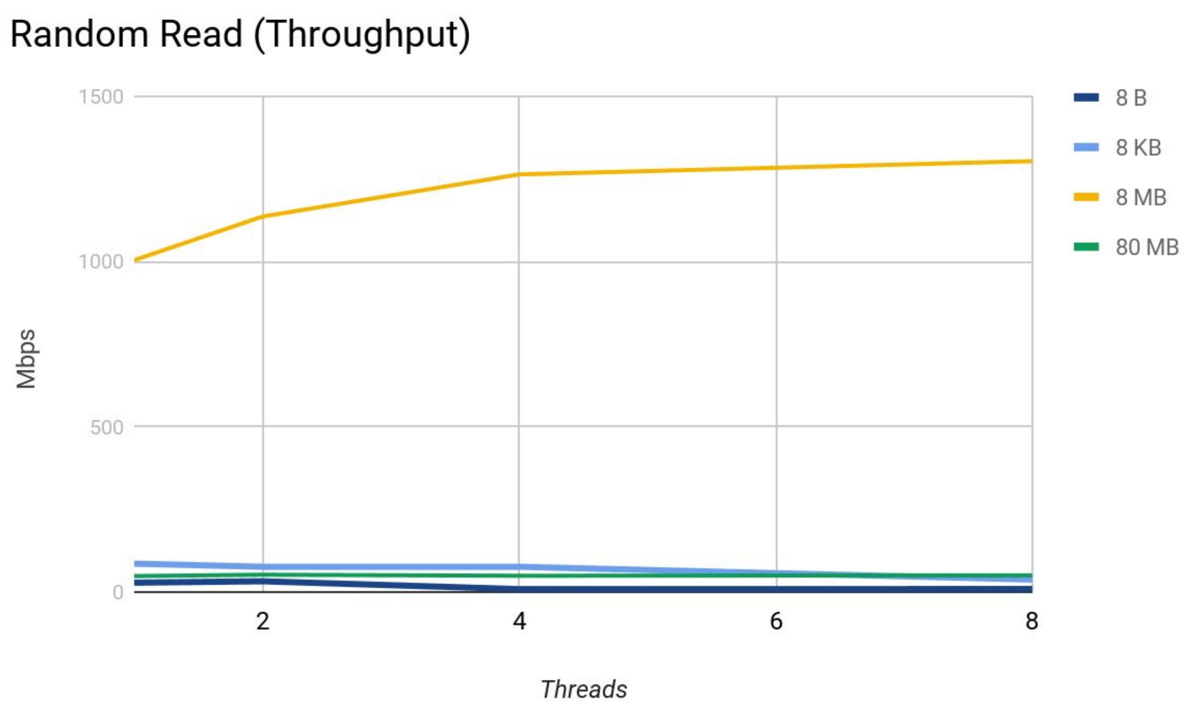
Throughput - 8 threads (MBps) :

SNo	Read + Write				Sequential Read				Random Read			
	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB	8 B	8 KB	8 MB	80 MB
1	10.390	78.125	457.14	47.675	16	49.543	800.00	49.87	9	39.0625	1333.3	49.937

2	10	72.944	429.439	50.435	15.87	42.9534	753.892	52.89	9.34	36.7674	1274.4510	51.34
Avg	**10.2**	**75.5345**	**443.2895**	**49.055**	**15.435**	**46.2482**	**776.95**	**51.38**	**9.17**	**37.9147**	**1303.8755**	**50.6385**







Thus, Random Read and Sequential Read values are almost similar in a Solid State Drive (SSD). Since the values of Random Read and Sequential Read obtained above are similar, we can say that the disk being evaluated is an SSD. The optimal performance was obtained at the concurrency level 2.

Theoretical performance given :

= 6TB inline SAS Drive

Throughput : 6Gbps (Read + Write)

Efficiency achieved = $\frac{\text{(Actual Performance)}}{\text{(Theoretical Performance)}} * 100$

= (3.76/6) * 100 (Since 470MBps = 3.76Gbps)

= **62.67%**

IOZONE BENCHMARK:

The result obtained after performing iozone benchmark for a file size is 8 MB is **271077 KB/s which is 2.168616 Gbps.**

```
[cc@pa1-gudur-reddy-bk6-1 current]$ ./iozone -a -r 8M -s 10G -T
        Iozone: Performance Test of File I/O
                Version $Revision: 3.471 $
                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: Mon Oct  9 16:10:24 2017

        Auto Mode
        Record Size 8192 kB
        File size set to 10485760 kB
        Command line used: ./iozone -a -r 8M -s 10G -T
        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
        10485760    8192   139435   271077   240479    272543    217201    160464    140163   8290533      141783   213304   135893    164936    175763

iozone test complete.
```

Efficiency achieved = (2.168/6) * 100

= **36.13 %**

3. Conclusion

- The throughput is the highest for the 8MB block size.
- The Read + write is slower than random and sequential Read. Also, the Sequential read is higher than the speed of random read. This is because of the fact that the disk head should move more for the random case in a hard disk. Hence the difference in the speeds of random and sequential reads is less in the case of a SSD.

4. Improvements and Extensions

- While the test is in progress, for optimal results, the system should be held idle.
- If the code is written more efficiently by using a better API, better results can be achieved.
- A GUI increases the ease of use of the benchmarking test.

NETWORK BENCHMARKING:

1. Program Design

The main goal of this experiment is to establish TCP/UDP connection between client and server and measure the network speed between one node as well as two nodes.
The metrics used for measurement are throughput and latency, at varying levels of currency.

- The design structure of the program is such that separate codes are written for both TCP and UDP.
- Both client and server send and receive data of the size of 64KB, 100 times.
- The throughput is measured the number of loops times data size divided by the time required to send and receive the data, in Mbits per sec.
- Latency is measured as execution time divided by number of iterations.
- Strong scaling is implemented at both server side and client side.

2. Performance

This experiment was performed on openkvm instance. The results obtained after performing the experiment several times are as follows:

TCP:

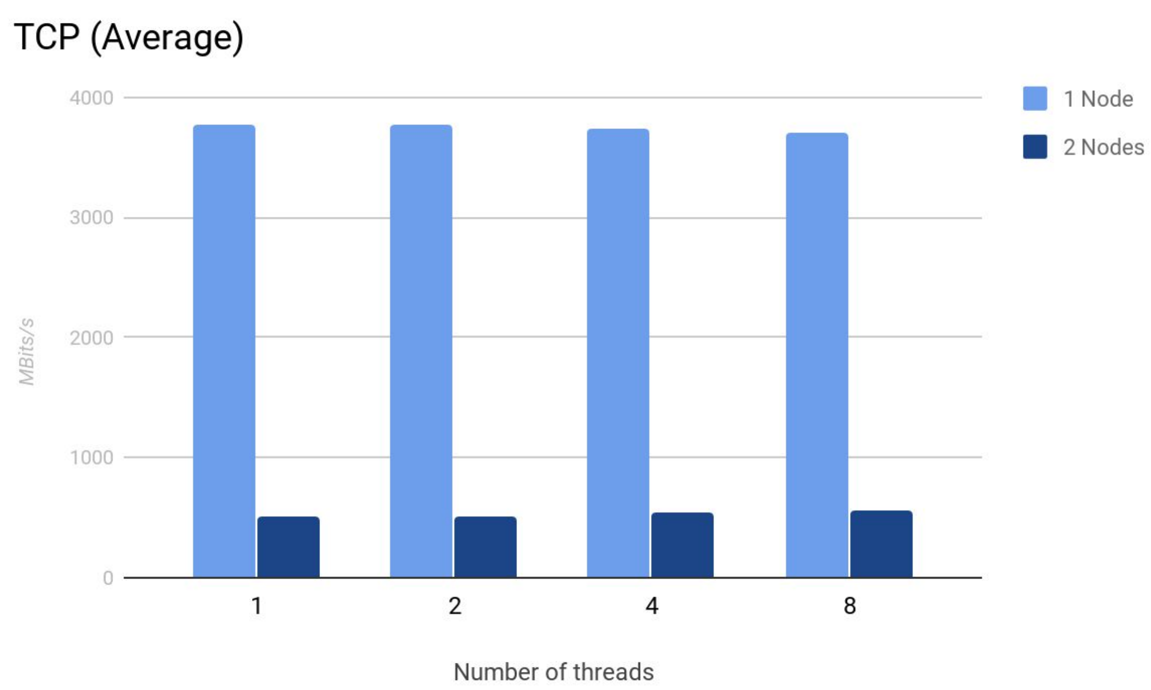
1 Node:

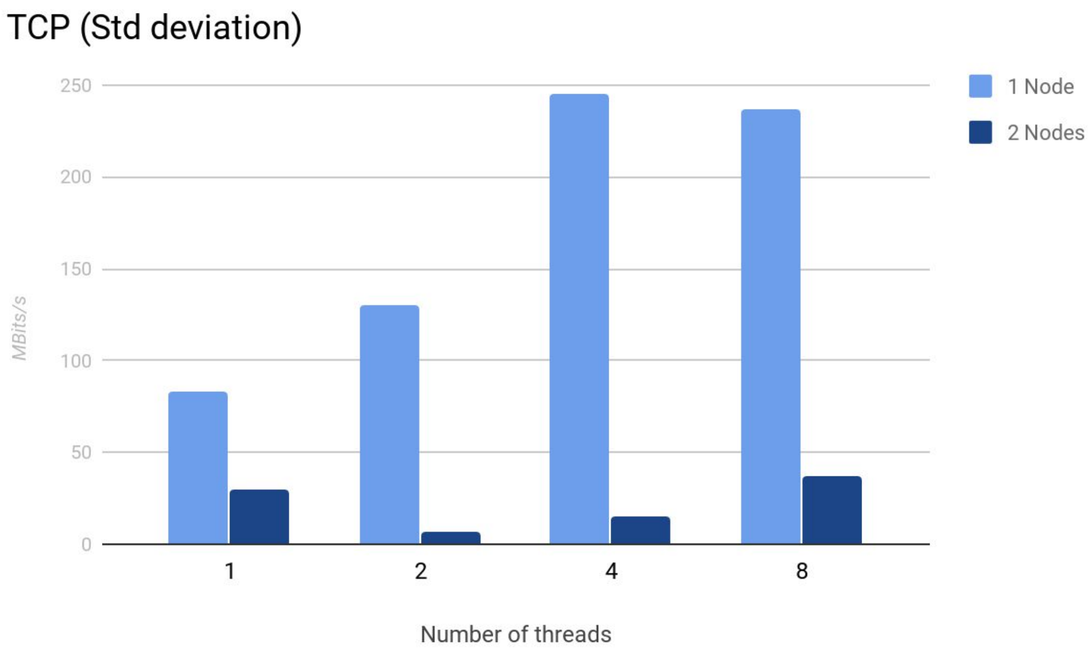
Sr. No.	1 thread		2 threads		4 threads		8 threads	
	Throughput	latency	Throughput	latency	Throughput	latency	Throughput	latency
1.	3759.681	0.133	3681.342	0.136	3527.336	0.142	3377.693	0.148
2.	3683.783	0.136	3963.221	0.126	4082.299	0.122	3910.833	0.128
3.	3885.607	0.129	3691.399	0.135	3603.343	0.138	3843.392	0.130
Average	**3776.357**	**0.132**	**3778.654**	**0.132**	**3737.659**	**0.134**	**3710.639**	**0.135**

Standard Deviation	**83.233**	**0.0028**	**130.573**	**0.0045**	**245.664**	**0.0086**	**237.033**	**0.0089**

2 Nodes:

Sr. No.	**1 thread**		**2 threads**		**4 threads**		**8 threads**	
	Throughput	**latency**	**Throughput**	**latency**	**Throughput**	**latency**	**Throughput**	**latency**
1.	550.964	0.908	498.529	1.003	528.306	0.946	609.377	0.820
2.	481.245	1.039	515.533	0.969	536.032	0.932	530.571	0.942
3.	500.265	1.000	503.829	0.992	562.809	0.888	532.305	0.929
Average	**510.824**	**0.982**	**505.96**	**0.988**	**542.382**	**0.922**	**557.417**	**0.897**
Standard Deviation	**29.42**	**0.0549**	**7.104**	**0.0141**	**14.784**	**0.0247**	**36.747**	**0.0547**





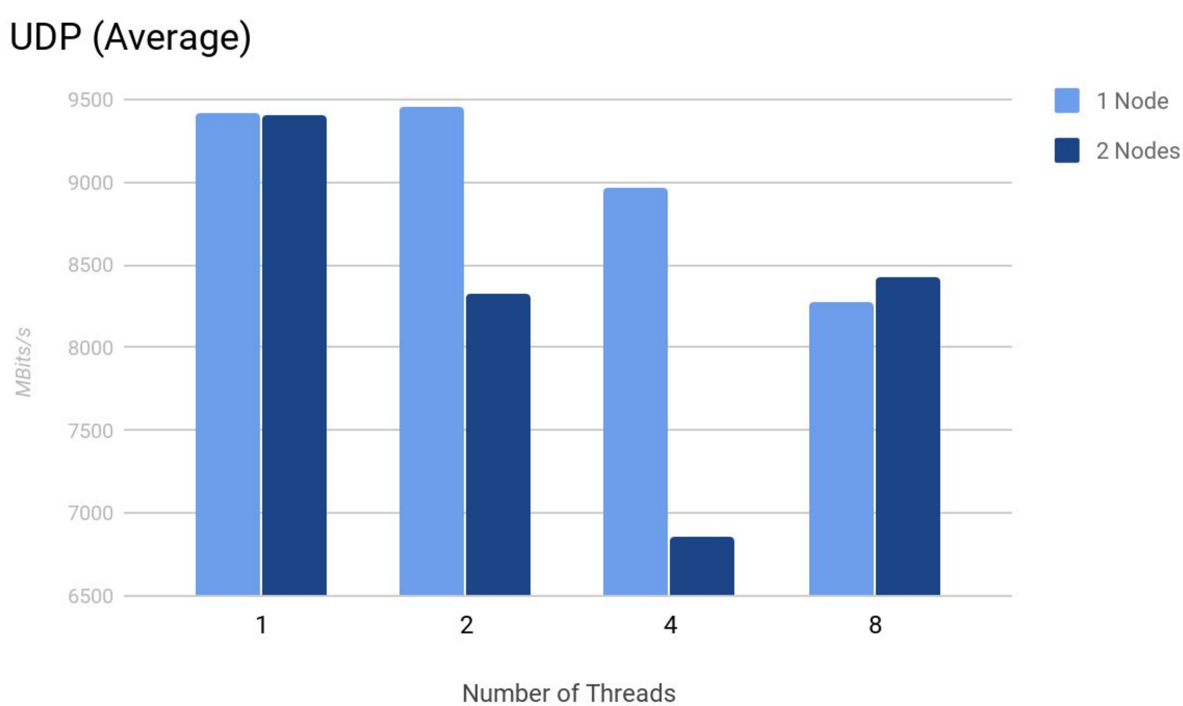
UDP:

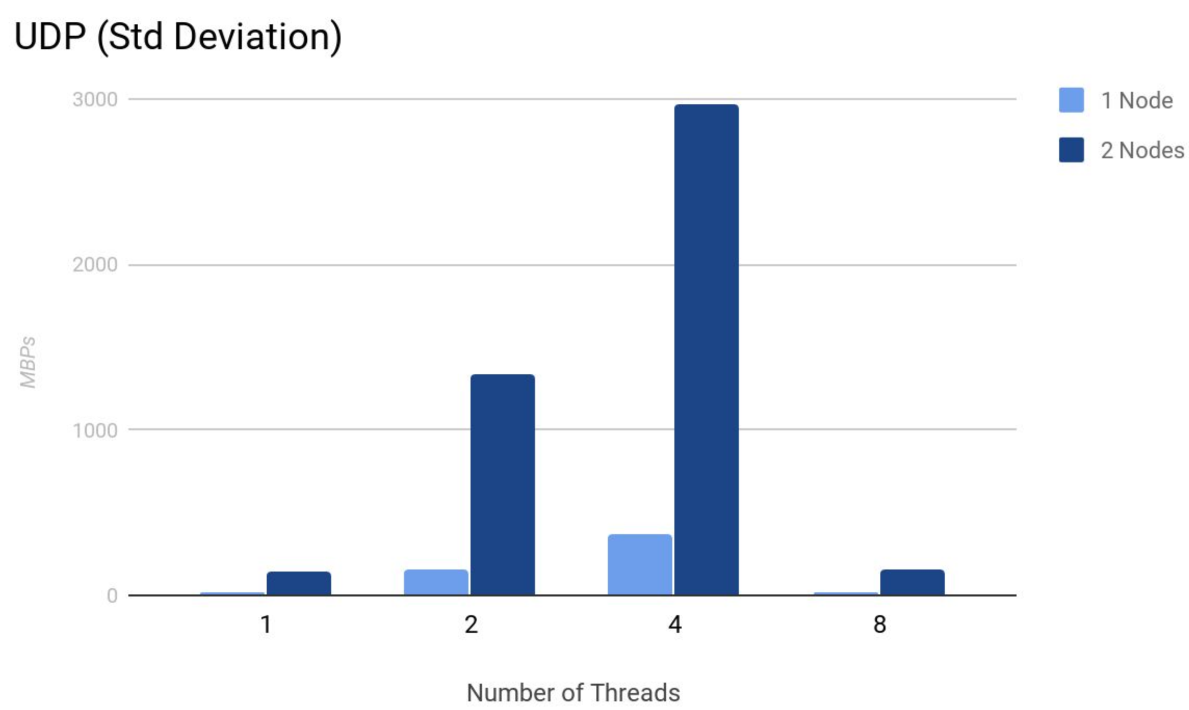
1 Node:

Sr. No.	1 thread		2 threads		4 threads		8 threads	
	Throughput	latency	Throughput	latency	Throughput	latency	Throughput	latency
1.	9403.798	0.0531	9516.588	0.0525	8484.642	0.0589	8271.898	0.0604
2.	9433.961	0.0530	9602.358	0.0520	9380.863	0.0533	8264.462	0.0605
3.	10.864.840	0.0460	9237.021	0.0541	9031.792	0.0553	8298.754	0.0602
Average	**9418.8795**	**0.0507**	**9451.989**	**0.0528**	**8965.765**	**0.0558**	**8278.371**	**0.0603**
Standard Deviation	**15.081**	**0.0033**	**155.986**	**0.00089**	**368.847**	**0.00231**	**14.728**	**0.000124**

2 Nodes:

Sr. No.	1 thread		2 threads		4 threads		8 threads	
	Throughput	**latency**	**Throughput**	**latency**	**Throughput**	**latency**	**Throughput**	**latency**
1.	9442.870	0.0529	9267.840	0.0539	8716.875	0.0573	8385.0410	0.0596
2.	9543.805	0.0523	6434.178	0.0771	2668.089	0.1874	8272.667	0.0604
3.	9209.799	0.0543	9281.603	0.0538	9196.249	0.0543	8638.562	0.0578
Average	**9398.824**	**0.05316**	**8327.873**	**0.0616**	**6860.404333**	**0.0996**	**8432.09**	**0.0592**
Standard Deviation	**139.868**	**0.000837**	**1339.056**	**0.01096**	**2970.86**	**0.06204**	**153.035**	**0.00108**





For TCP, Thus, the best throughput performance, 1 node obtained over the loopback interface is **3776.357 Mbits/s , 3778.654 MBits/s, 3737.659 MBits/s, 3710.639 MBits/s** for **1, 2 , 4 and 8 threads** respectively.

For 2 nodes, throughput performance obtained is **510.824 Mbits/s , 505.96 MBits/s, 542.382 MBits/s, 557.417 MBits/s** for **1,2,4**, and **8** threads respectively.

For UDP, Thus, the best throughput performance, 1 node obtained over the loopback interface is **9418.8795 MBits/s, 9451.989 Mbits/s, 8965.765 MBits/s and 8278.371 MBits/s** for **1, 2 , 4 and 8 threads** respectively.

For 2 nodes, throughput performance obtained is **9398.824 MBits/s, 8327.873 MBits/s, 6860.404333 MBits/s, 8432.09 MBits/s** for **1,2,4**, and **8** threads respectively.

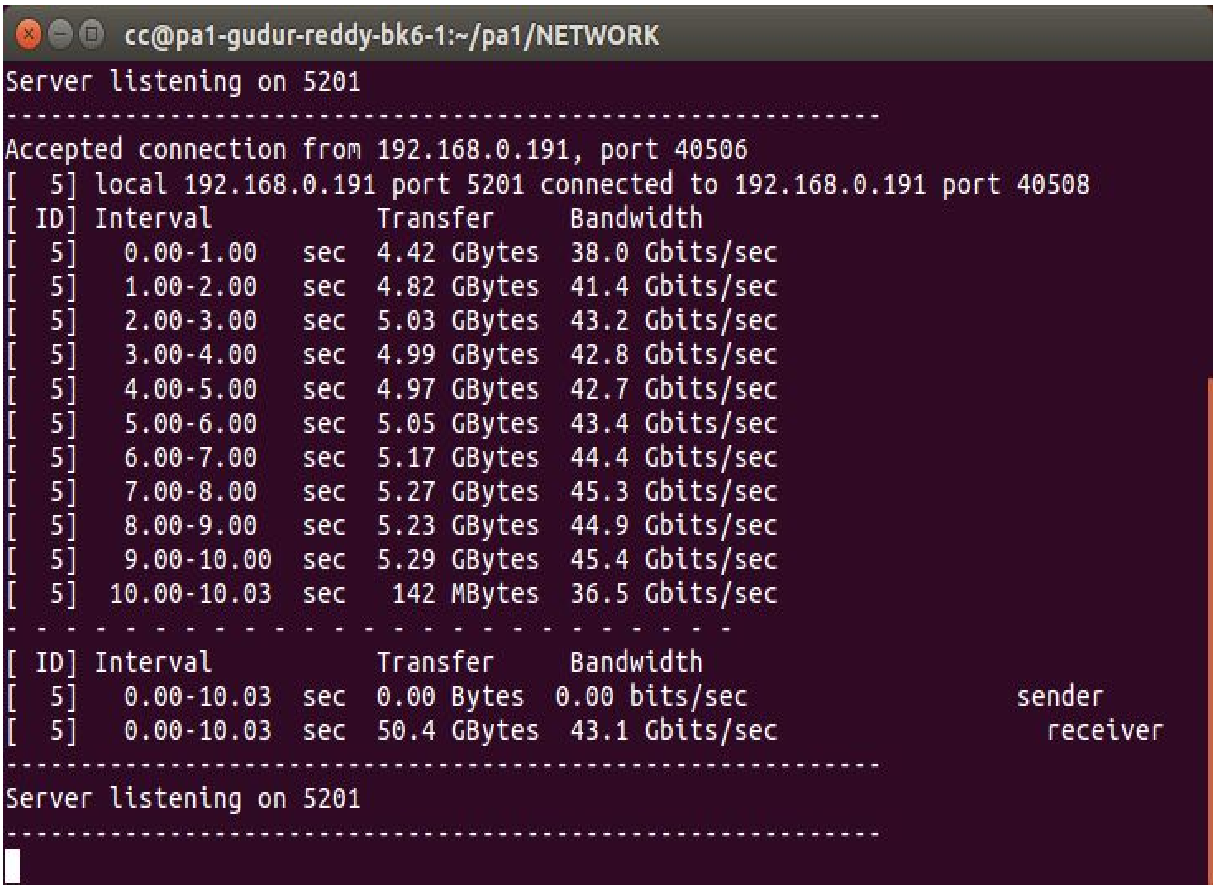
Theoretical Network Speed = 12500 MBits/s

IPerf Benchmark:

IPerf benchmark is run on two separate Baremetal instances and performance is recorded.

The screenshots of client and server are given below:

1 Node:



```
cc@pa1-gudur-reddy-bk6-1:~/pa1/NETWORK
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.0.191, port 40506
[  5] local 192.168.0.191 port 5201 connected to 192.168.0.191 port 40508
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec  4.42 GBytes  38.0 Gbits/sec
[  5]   1.00-2.00   sec  4.82 GBytes  41.4 Gbits/sec
[  5]   2.00-3.00   sec  5.03 GBytes  43.2 Gbits/sec
[  5]   3.00-4.00   sec  4.99 GBytes  42.8 Gbits/sec
[  5]   4.00-5.00   sec  4.97 GBytes  42.7 Gbits/sec
[  5]   5.00-6.00   sec  5.05 GBytes  43.4 Gbits/sec
[  5]   6.00-7.00   sec  5.17 GBytes  44.4 Gbits/sec
[  5]   7.00-8.00   sec  5.27 GBytes  45.3 Gbits/sec
[  5]   8.00-9.00   sec  5.23 GBytes  44.9 Gbits/sec
[  5]   9.00-10.00  sec  5.29 GBytes  45.4 Gbits/sec
[  5]  10.00-10.03  sec   142 MBytes  36.5 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-10.03  sec  0.00 Bytes  0.00 bits/sec                  sender
[  5]   0.00-10.03  sec  50.4 GBytes  43.1 Gbits/sec                  receiver
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

Fig : server side

```
vaishnavi@vaishnavi-VirtualBox:~$ ssh -i cloud.key cc@129.114.33.25
Last login: Mon Oct  9 18:14:48 2017 from 68.61.156.76
[cc@pa1-gudur-reddy-bk6-1 ~]$ cd pa1/NETWORK/
[cc@pa1-gudur-reddy-bk6-1 NETWORK]$ sudo rpm -Uvh iperf3-3.1.3-1.fc24.x86_64.rpm

warning: waiting for transaction lock on /var/lib/rpm/.rpm.lock
Preparing...                          ################################# [100%]
        package iperf3-3.1.3-1.fc24.x86_64 is already installed
[cc@pa1-gudur-reddy-bk6-1 NETWORK]$ clear
[cc@pa1-gudur-reddy-bk6-1 NETWORK]$ iperf3 -c 192.168.0.191
Connecting to host 192.168.0.191, port 5201
[  4] local 192.168.0.191 port 40508 connected to 192.168.0.191 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  4.54 GBytes  39.0 Gbits/sec    0   2.12 MBytes
[  4]   1.00-2.00   sec  4.83 GBytes  41.5 Gbits/sec    0   2.12 MBytes
[  4]   2.00-3.00   sec  5.03 GBytes  43.2 Gbits/sec    0   2.12 MBytes
[  4]   3.00-4.00   sec  4.99 GBytes  42.8 Gbits/sec    0   2.12 MBytes
[  4]   4.00-5.00   sec  4.96 GBytes  42.6 Gbits/sec    0   2.12 MBytes
[  4]   5.00-6.00   sec  5.06 GBytes  43.5 Gbits/sec    0   2.12 MBytes
[  4]   6.00-7.00   sec  5.16 GBytes  44.3 Gbits/sec    0   2.12 MBytes
[  4]   7.00-8.00   sec  5.29 GBytes  45.4 Gbits/sec    0   2.12 MBytes
[  4]   8.00-9.00   sec  5.22 GBytes  44.9 Gbits/sec    0   2.12 MBytes
[  4]   9.00-10.00  sec  5.29 GBytes  45.5 Gbits/sec    0   2.12 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  50.4 GBytes  43.3 Gbits/sec    0             sender
[  4]   0.00-10.00  sec  50.4 GBytes  43.3 Gbits/sec                  receiver

iperf Done.
[cc@pa1-gudur-reddy-bk6-1 NETWORK]$
```

Fig : client side

Efficiency achieved = (4330/12500) * 100

= **34.64%**

2 Nodes:

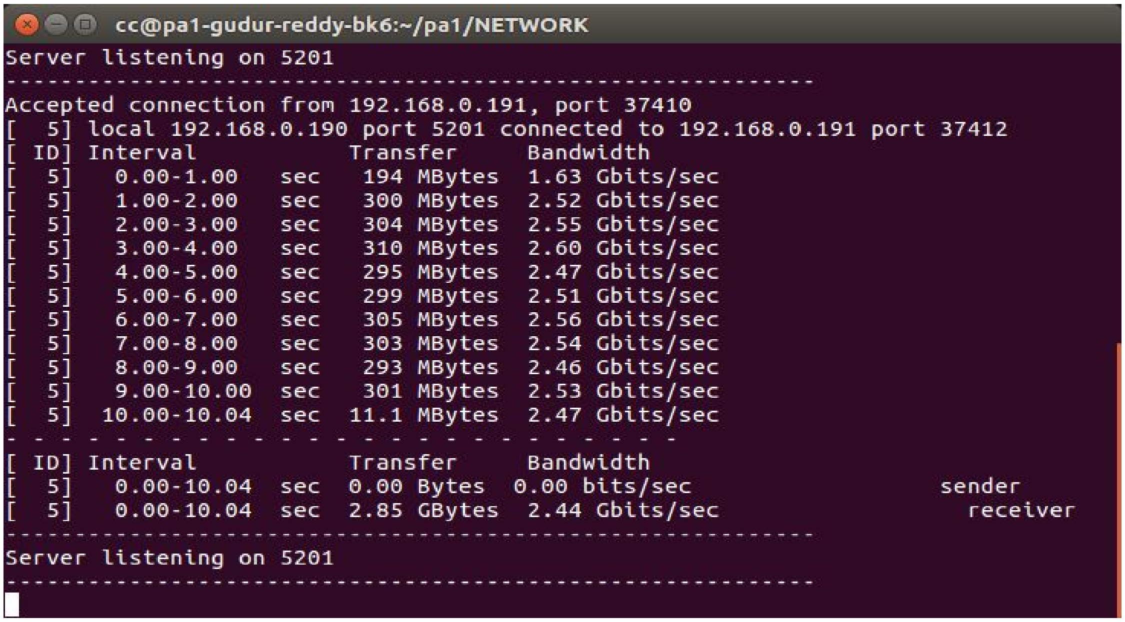


Fig : Server side instance

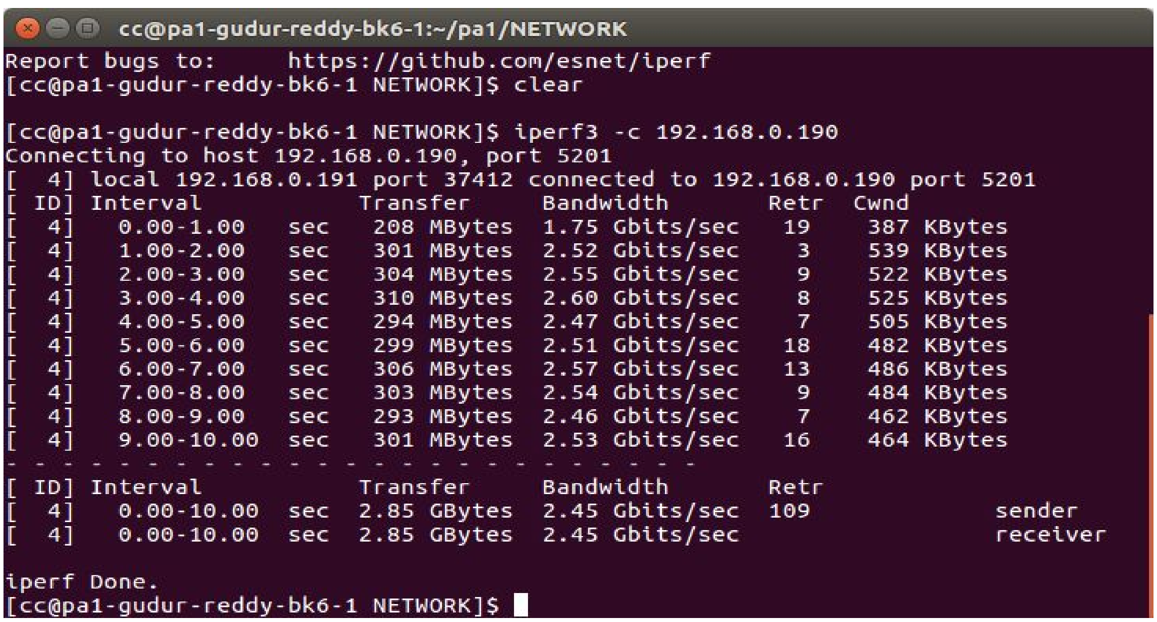


Fig : Client side instance

Thus, the best performance achieved by IPerf benchmark is **2.45 Gbits/sec** i.e 2450 Mbits/sec.

$$\text{Efficiency achieved} = \frac{\text{(Actual performance)}}{\text{(Theoretical Performance)}} * 100$$

= (2450/12500) * 100
= **20 %**

3. Conclusion :

Thus, network speed achieved is greater on a loopback interface on one node as compared to that achieved on two nodes. This may be due to a variety of factors including network traffic, congestion.

UDP transmission gives greater network speed however, it is unreliable compared to TCP.

IPerf benchmark achieved greater efficiency on one node.

4. Improvements and Extensions:

1. GUI can be implemented.
2. Varying size of data can be transmitted and the program can be tested on all levels to observe the deviation.