CS553 Programming Assignment #1

Benchmarking

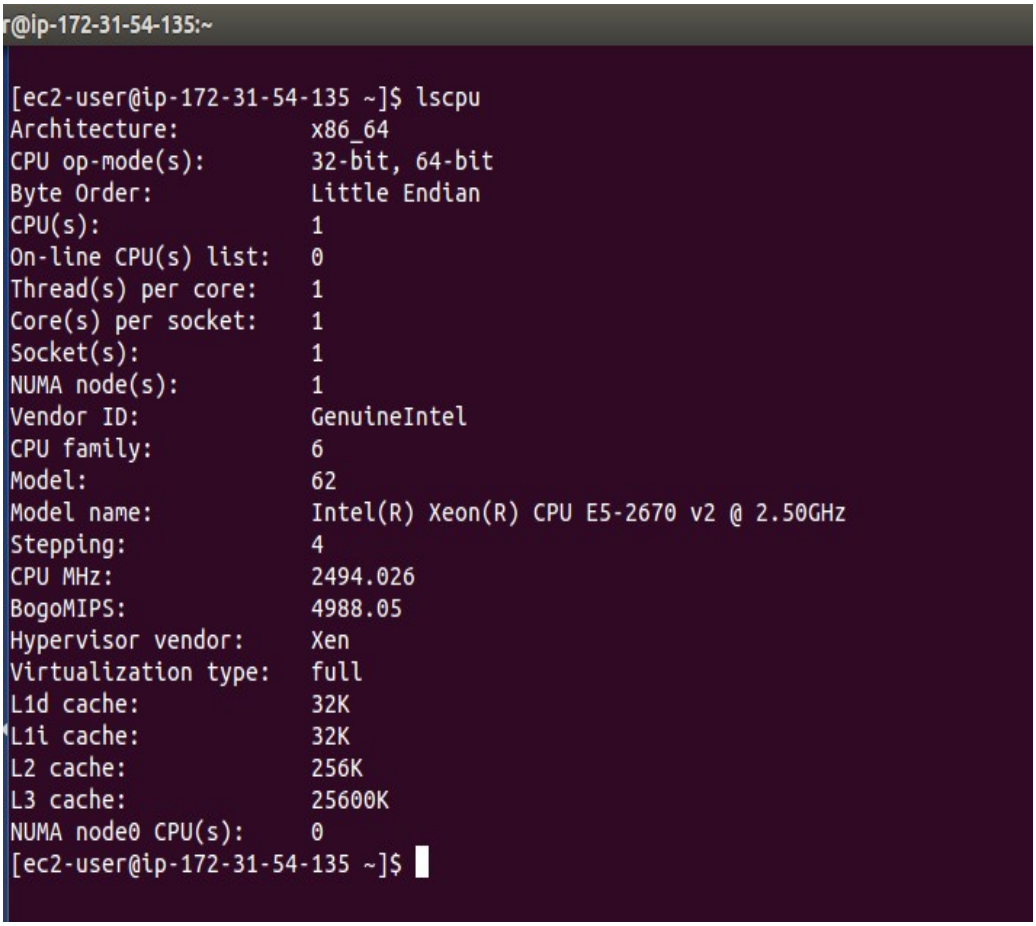
Performance Evaluation

Harsh C. Parikh
A20338453

This document contains all experimental results of all the benchmarks. Each experiment is performed 3 times and average and standard deviation are calculated based on that.

1. Introduction

For all the benchmarks, I experimented on Amazon EC2, with 1 core CPU, Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz. All Programs are done in Python language.



For CPU benchmark, I measured processor's speed in terms of GFLOPS and GIOPS at varying concurrency level for 1,2 and 4 threads. Moreover, I took the 600 samples for IOPS and FLOPS at the interval of 1 sec till 10 minutes.

For Disk benchmark, I measured the throughputs of Sequential Read, Sequential Write, Random Read and Random Write and latency for all of them and varied the block sizes (1B/1KB/1MB) for the number of threads (1 and 2).

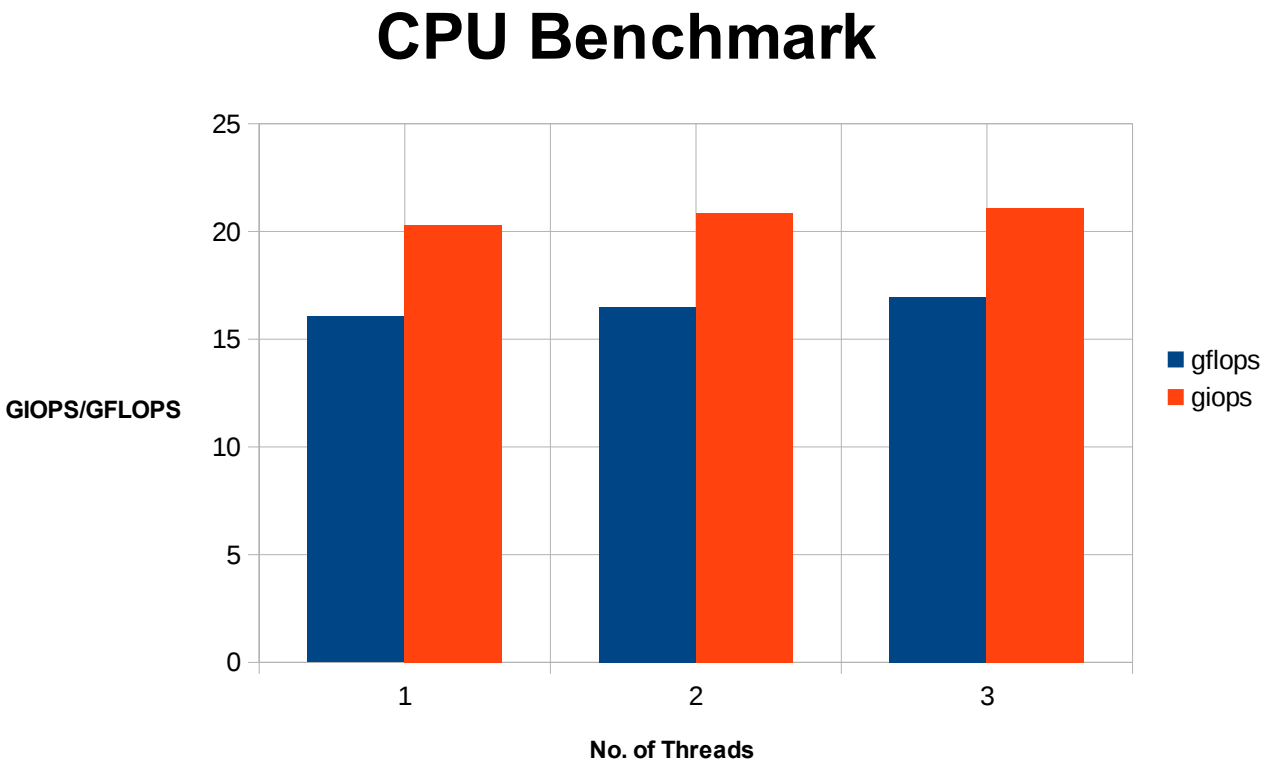
For Network benchmark, I measured throughput and latency for both TCP and UDP protocols and varied the packet size (1B/1KB/64KB) and the number of threads (1 and 2)

2. Experiment Results and Analysis

In this section, all experiment results for each benchmark and explanations for the trede offs in the results.

2.1 CPU Benchmarking

- For CPU benchmark, I am finding GIOPS and GFLOPS value for different number of threads.
- In graph, I had done multiple experiments for 1,2 and 4 threads and according to that I am getting results for GFLOPS and GIOPS as follows.



- Theoretical peak performance of the processor is given by:

= (CPU speed in GHz) x (number of CPU cores) x (CPU instruction per cycle) x (number of CPUs per node)

=2.50 * 1 *8 * 2

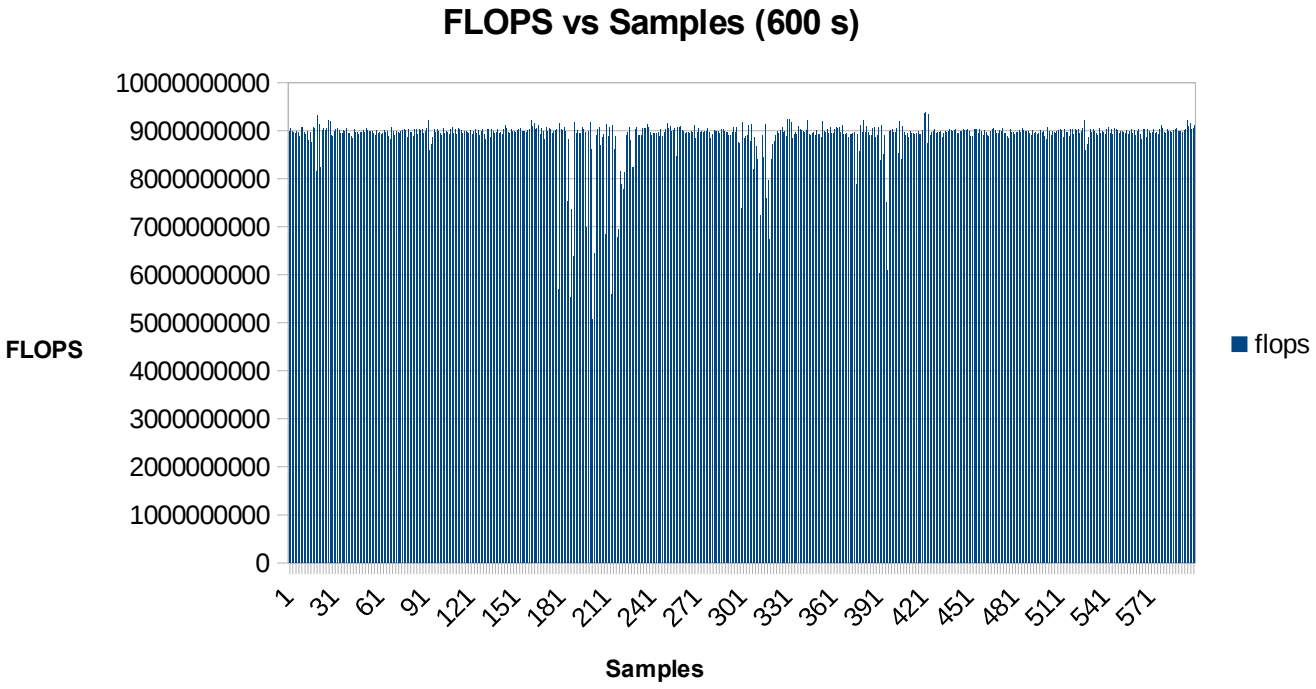
= 40 Gflops

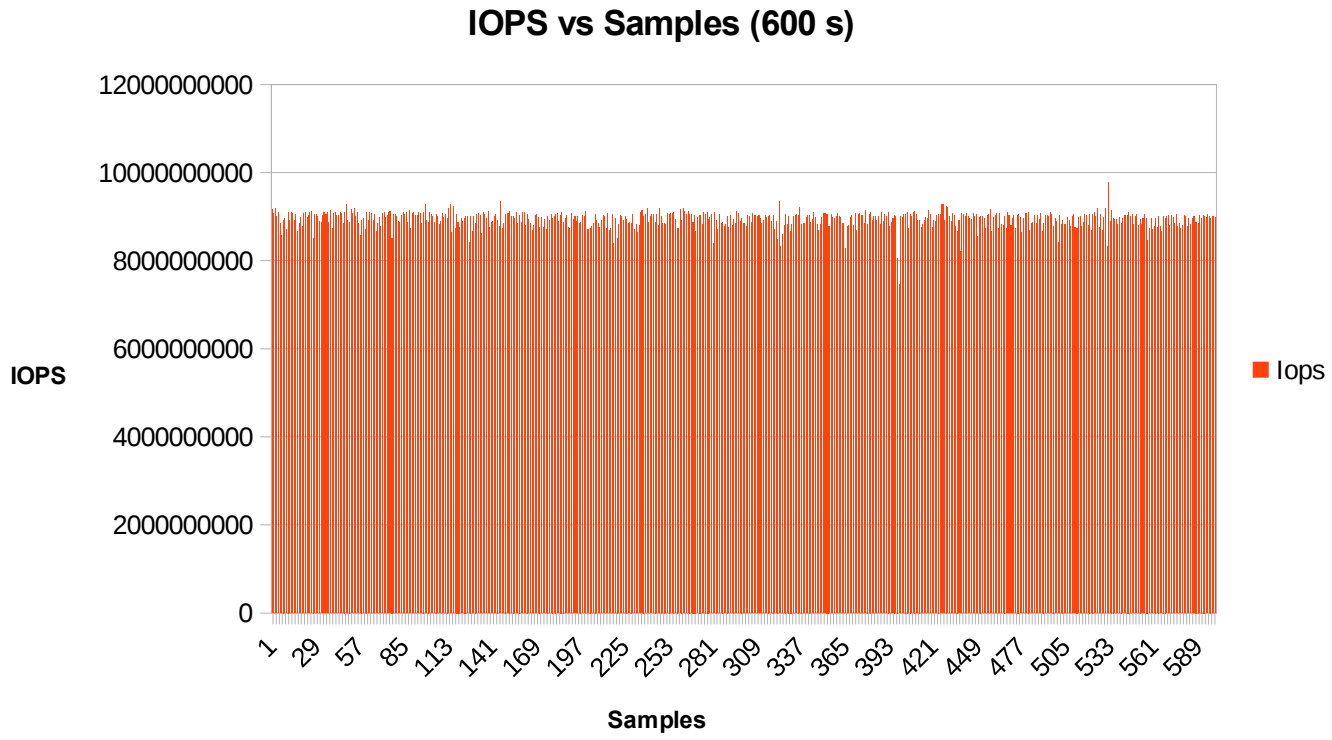
and efficiency would be = (20.88/40)*100 = **52.20 %.**

Average and Standard Deviation for GFLOPS and IOPS are in following tables.

Number of thread	Average (GIOPS)	Standard Deviation(GIOPS)	Average(GFLOPS)	Standard Deviation(GFLOPS)
1	24.2798	0.3541	20.12567	0.3011
2	24.4782	0.3631	20.8832	0.3166
4	24.9537	0.3777	21.43637	0.3199

Now, I have taken **600 samples of IOPS and FLOPS**. Here are the snap-shots of results. These results performed on **4 threads**.





Here, in the graph, I observe there are some glitches. That's because of operations were performed less in that time interval due to other process might use the core. So sometimes due to that we got these types of glitches as if less numbers of operations performed in that period.

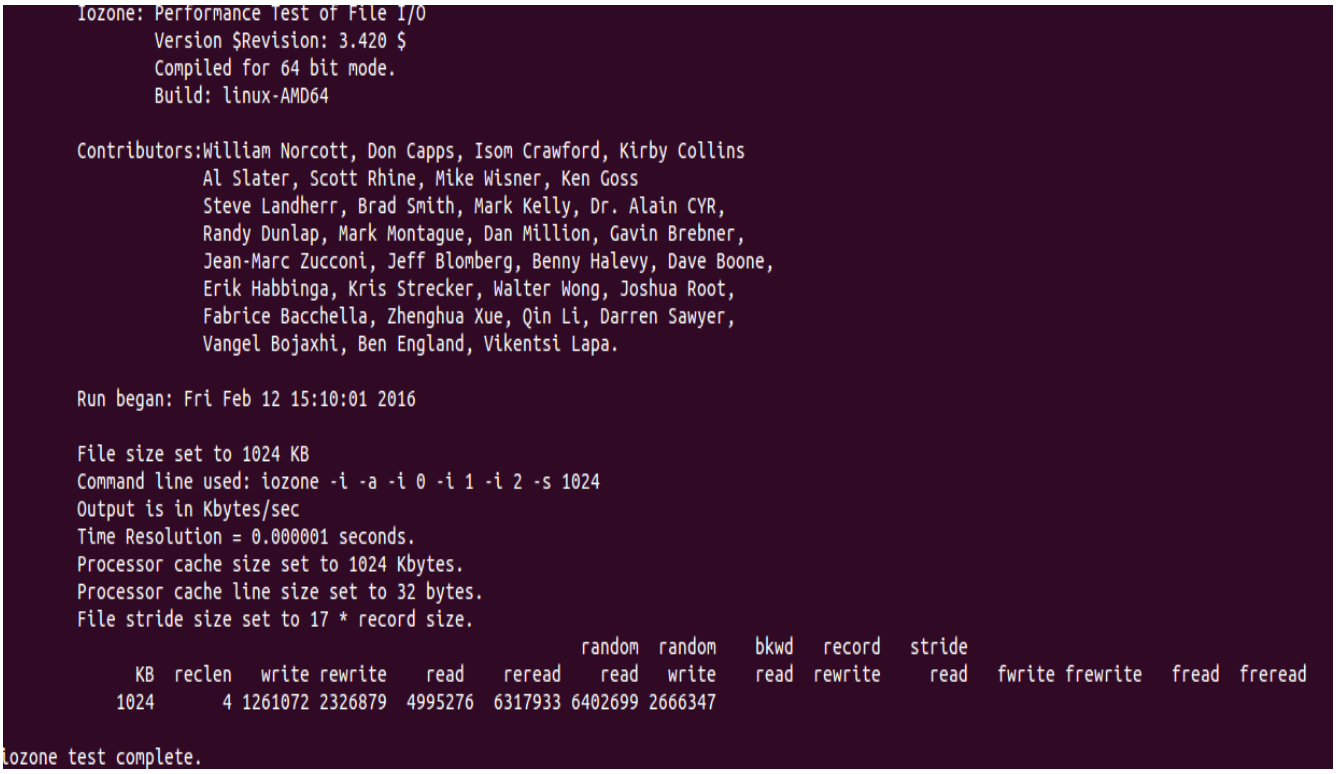
Now, I had run the **Lin-pack benchmark and result is being shown in below screen..**

```
[ec2-user@ip-172-31-59-112 linpack]$ ./runme_xeon64
This is a SAMPLE run script for SMP LINPACK. Change it to reflect
the correct number of CPUs/threads, problem input files, etc..
Thu Feb 11 04:51:24 UTC 2016
Intel(R) Optimized LINPACK Benchmark data

Current date/time: Thu Feb 11 04:51:24 2016

CPU frequency:    2.824 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 1800
0 20000 22000 25000 26000 27000 30000 35000 40000 45000
Leading dimension of array                  : 1000  2000  5008  10000 15000 1800
8 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      0.040      16.6632  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.038      17.6646  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.039      17.1456  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.038      17.6622  9.900691e-13 3.376390e-02   pass
2000   2000   4      0.286      18.6465  4.053480e-12 3.526031e-02   pass
2000   2000   4      0.286      18.6809  4.053480e-12 3.526031e-02   pass
5000   5008   4      4.263      19.5591  2.336047e-11 3.257429e-02   pass
5000   5008   4      4.265      19.5509  2.336047e-11 3.257429e-02   pass
10000  10000  4      33.075     20.1620  1.124127e-10 3.963786e-02   pass
10000  10000  4      32.466     20.5406  1.124127e-10 3.963786e-02   pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
1000   1000   4       17.2839  17.6646
2000   2000   4       18.6637  18.6809
5000   5008   4       19.5550  19.5591
10000  10000  4       20.3513  20.5406

Residual checks PASSED

End of tests

Done: Thu Feb 11 04:52:46 UTC 2016
[ec2-user@ip-172-31-59-112 linpack]$
```

From practical performance, linpack gives me around **20.35 Gflops.**

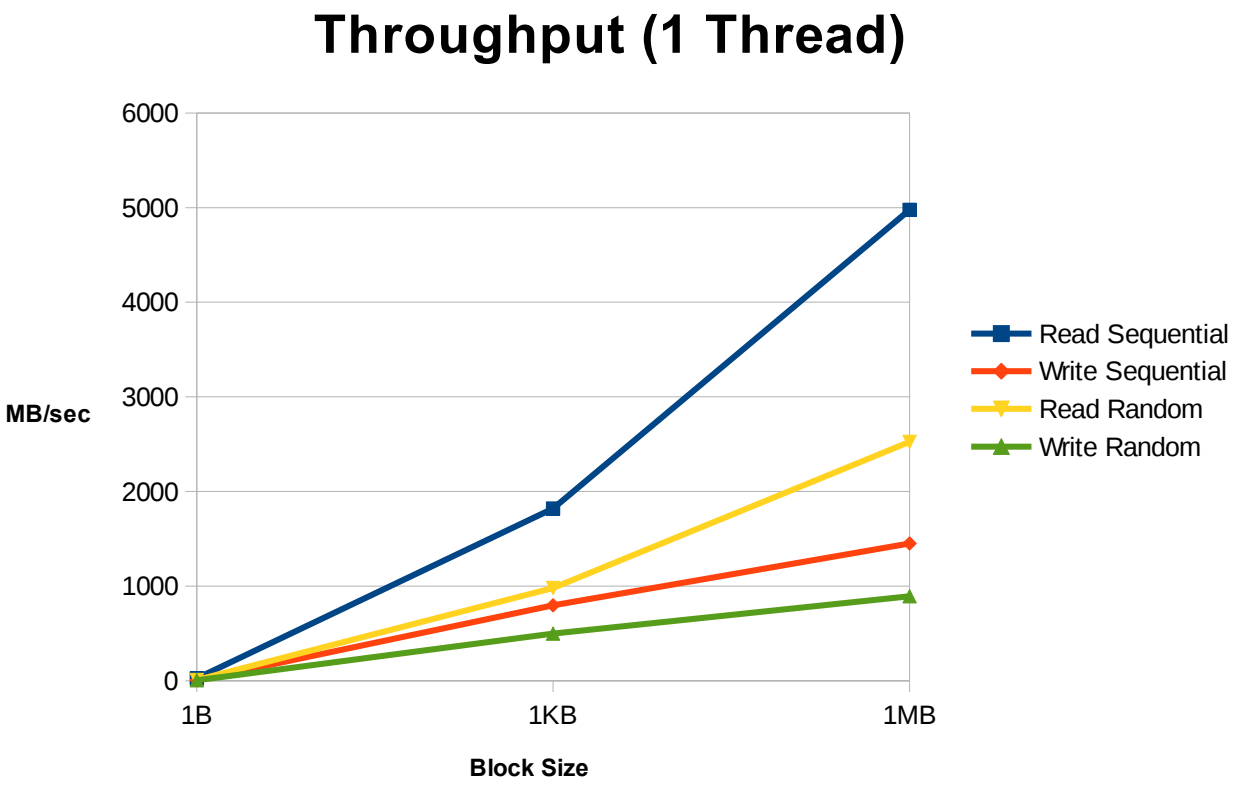
Efficiency = (20.53/40)*100 = 51.32 %

2.2 Disk Benchmark

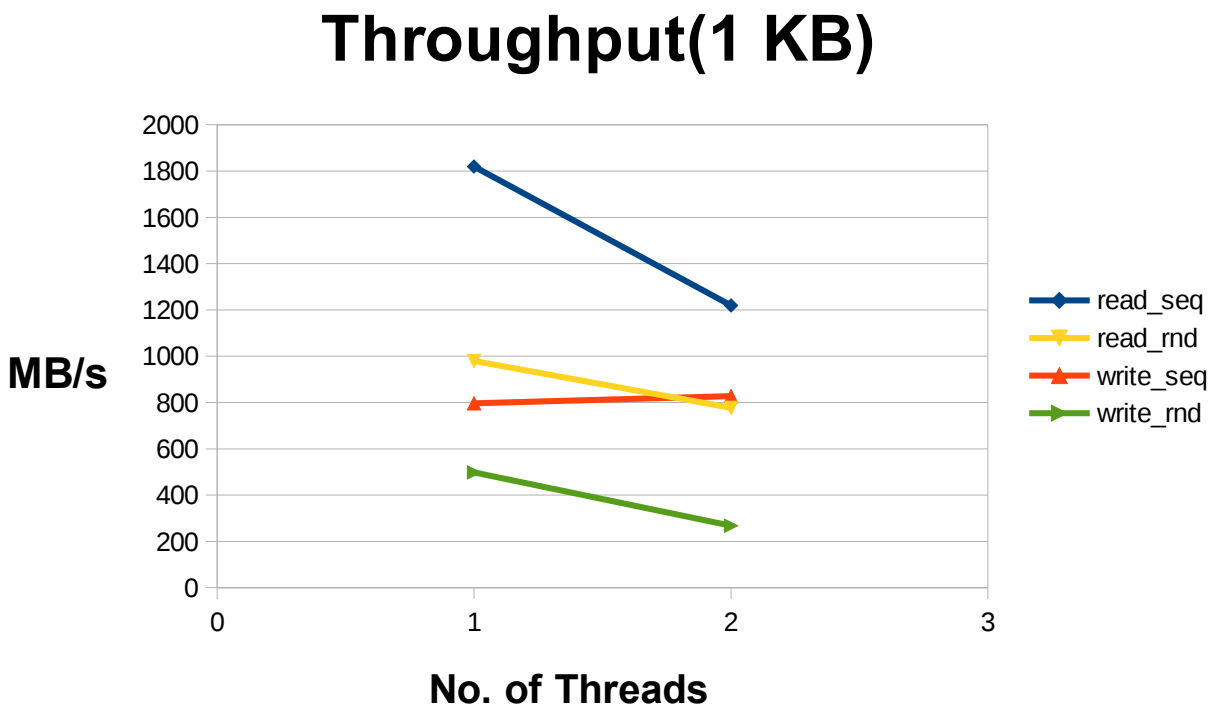


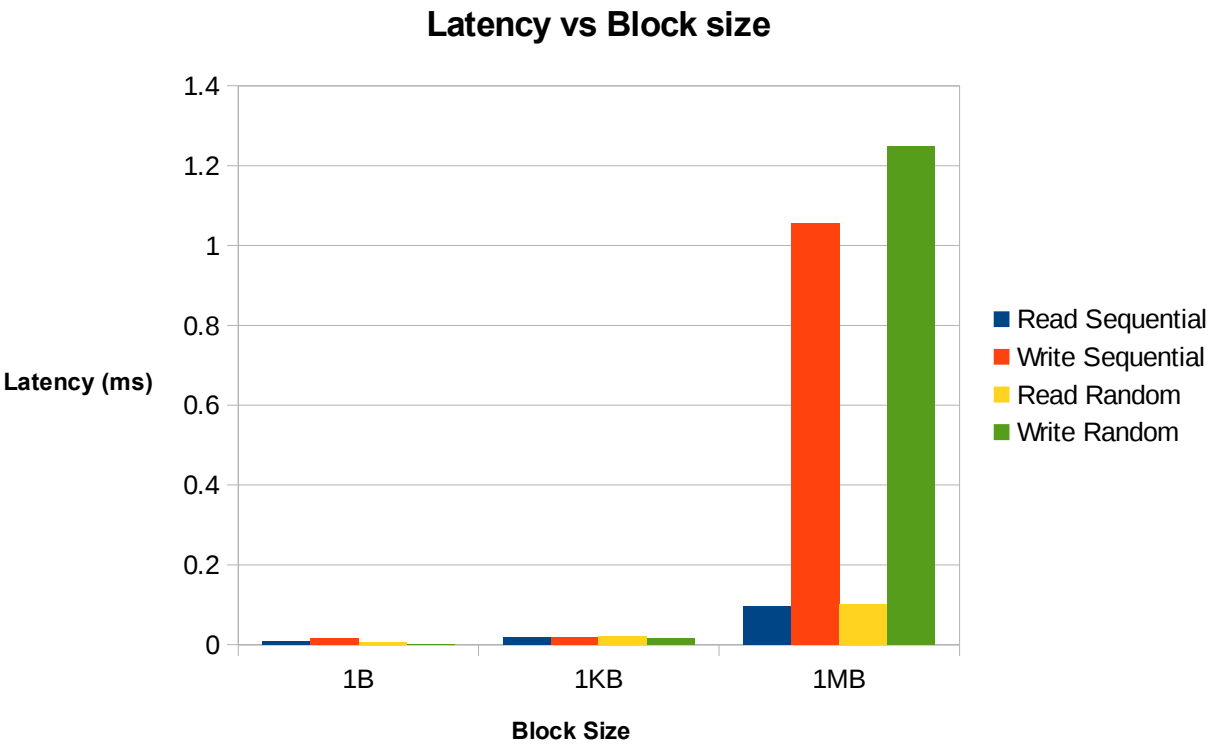
From practical Throughput for reading 1MB or 1024 KB, I'm getting around = 4.977 GB/sec = 4977615 B/sec and as per theoretical Throughput, it would be around 6 GB/sec (as per wikipedia, our EC2 instance version is 3.0) . So efficiency would be around **82.83%.** Moreover on IOzone, throughput is around 6.40 GB/sec.

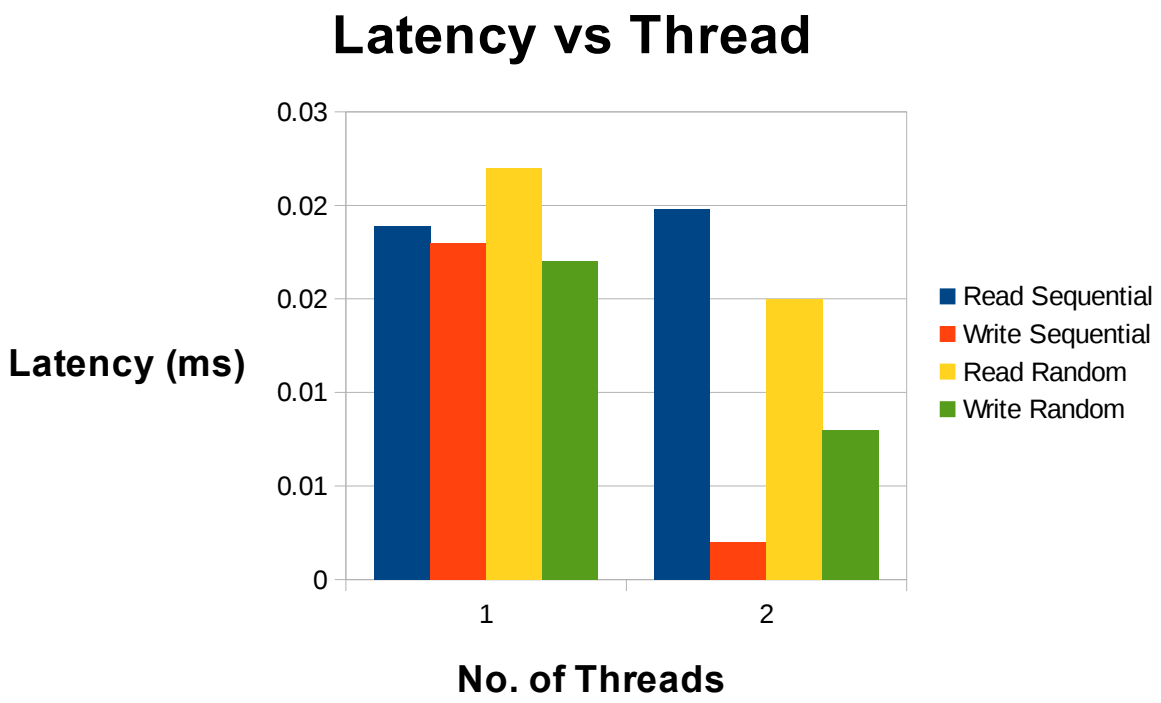
- And latency is = 0.747 ms .
- Here I have calculated the throughput of single thread and multi threaded and block size for sequential and random read/write. And their results are shown.
- Here, I'm getting less throughput for 2 threads compare to 1 thread because when in 2 threads, after completion of one thread, other can access the file. So 2 threads can't access a file at the same time. (Read it somewhere from Amazon Notes).



- The graph shows the trade offs between sequential and random read/write throughput according to block size. It's going upwards as we increase the block sizes And if we increase number of threads then throughput is sometimes decreasing because if one can't write in a file simultaneously. So that
- Sequential throughput is faster than random throughput for both read and write operations.
- Read throughput performs better in both sequential and random cases.
- Another Graph shows the trade offs between number of threads and throughput in sequential read, sequential write, random read and random write with respect to latency in milliseconds.
- Sequential operation works faster than random throughput for both read and write.
- In other graphs, I have used 1 KB block to measure trade-off between latency and thread. So, one can understand from graph that if multi-threaded operation takes lesser time for both of them.


Throughput(1 KB)
MB/s
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
1
2
3
No. of Threads
read_seq
read_rnd
write_seq
write_rnd


Latency vs Block size
Latency (ms)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1B
1KB
1MB
Block Size
Read Sequential
Write Sequential
Read Random
Write Random



Throughput result of Sequential Read/Write Operations (results in MB/sec):

	Read			Write		
Threads	1b	1k	1m	1b	1k	1m
1	24.138	1819.839	4977.615	15.442	796.307	1451.383
2	22.339	1219.013	3988.177	18.996	827.156	1290.241

Random Read/Write Operations (results in MB/sec):

Threads	Read			Write		
	1b	1k	1m	1b	1k	1m
1	18.45	979.985	2523.345	14.998	497.897	892.291
2	20.562	776.589	2069.252	15.945	267.586	579.468

Average and Standard Deviation for 1 Byte are in following tables(results in MB/sec & latency in ms).

For Sequential Read (1 B):

Number of Thread	Average(Throughput-Sequential Read)	Standard Deviation(Throughput-Sequential Read)	Average(Latency-Sequential Read)	Standard Deviation((Latency-Sequential Read)
1	24.138	4.22	0.0043	0.0036
2	22.339	3.88	0.0042	0.0027

For Random read (1 B):

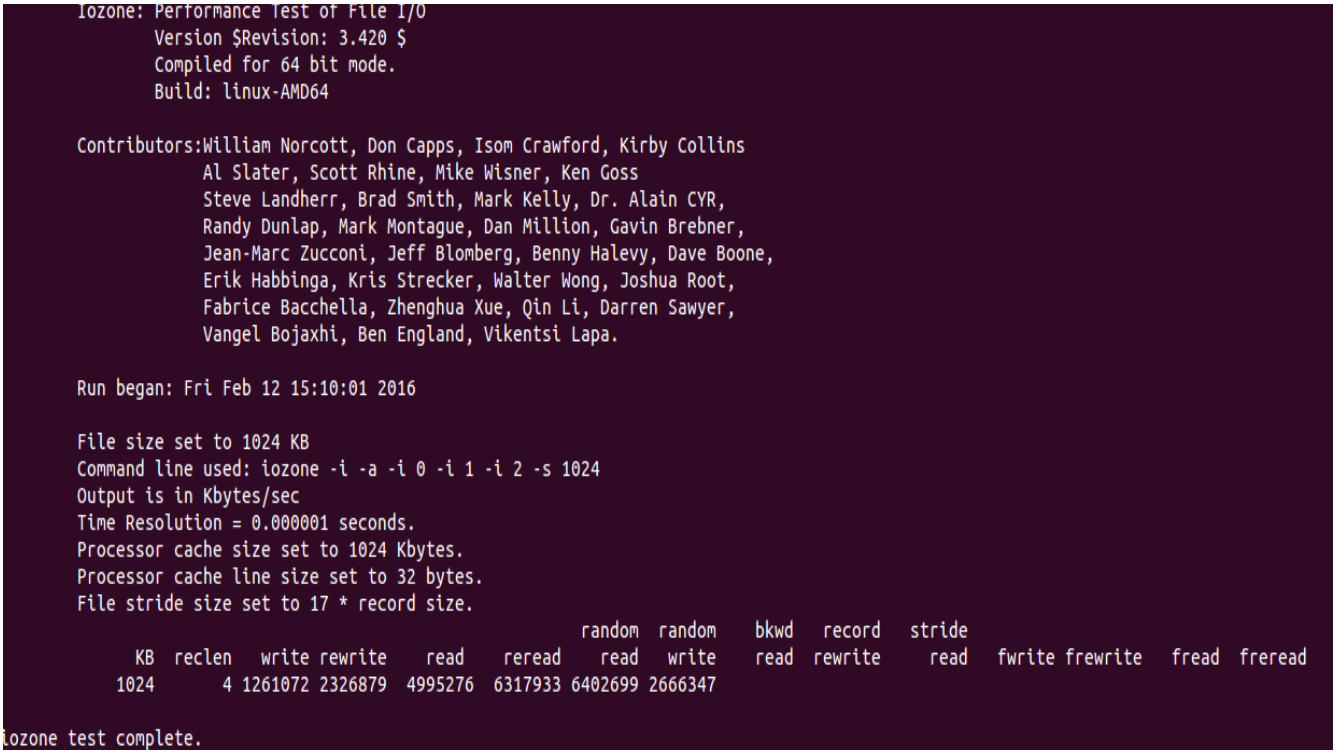
Number of Thread	Average(Throughput-Random Read)	Standard Deviation(Throughput-Random Read)	Average(Latency-Random Read)	Standard Deviation((Latency-Random Read)
1	18.45	6.99	0.0163	0.006
2	20.562	5.29	0.0071	0.004

For Sequential write (1 B):

Number of Thread	Average(Throughput-Sequential Write)	Standard Deviation(Throughput-Sequential Write)	Average(Latency-Sequential write)	Standard Deviation((Latency-Sequential Write)
1	15.442	2.88	0.081	0.007
2	18.996	4.22	0.096	0.006

For Random write (1 B):

Number of Thread	Average(Throughput-Random write)	Standard Deviation(Throughput-Random write)	Average(Latency-Random write)	Standard Deviation((Latency-Random write)
1	14.998	7.12	0.0101	0.006
2	15.545	3.65	0.0123	0.008

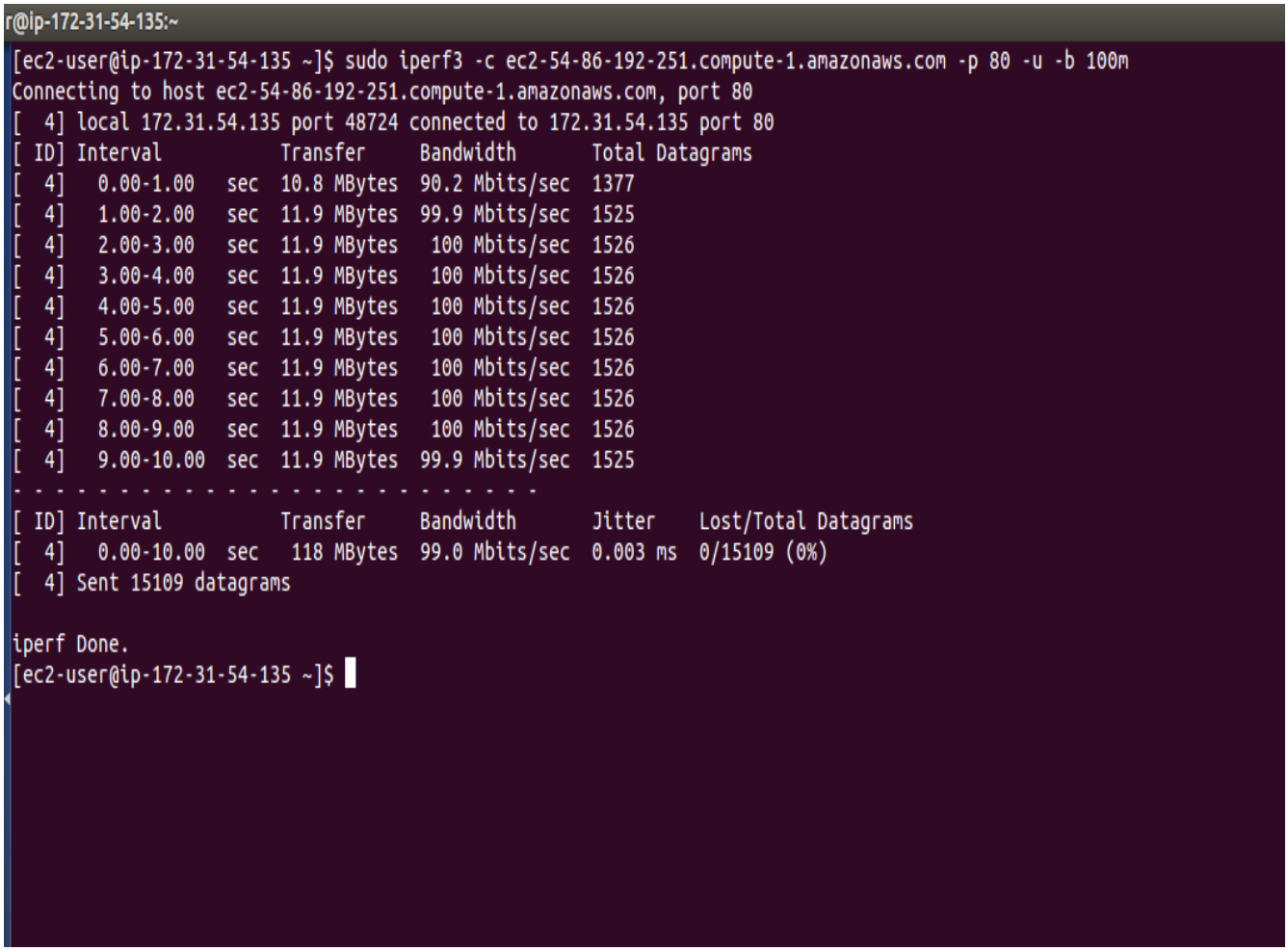


2.3 Network Benchmark

- The following graph shows the trade offs between Latency and throughput according to block size. It's going upwards as we increase the block sizes in both TCP and UDP .
- In Network, I observer that TCP throughput is lesser than UDP because UDP is a connectionless protocol and so in UDP, there is no guarantee of receiving bytes.
- On the other side, TCP sends an acknowledgement from server. So for that reason, in TCP, we are getting higher latency compare to UDP.
- The graph shows the trade offs between Latency and throughput according to block size. It's going upwards as we increase the block sizes in both TCP and UDP .
- In Network, I observer that TCP throughput is lesser than UDP because UDP is a connectionless protocol and so in UDP, there is no guarantee of receiving packages.
- On the other side, TCP sends an acknowledgement from server. So for that reason, in TCP, we are getting higher latency compare to UDP.

These are two snap-shots of UDP and TCP Iperf performance.

UDP Graph: -



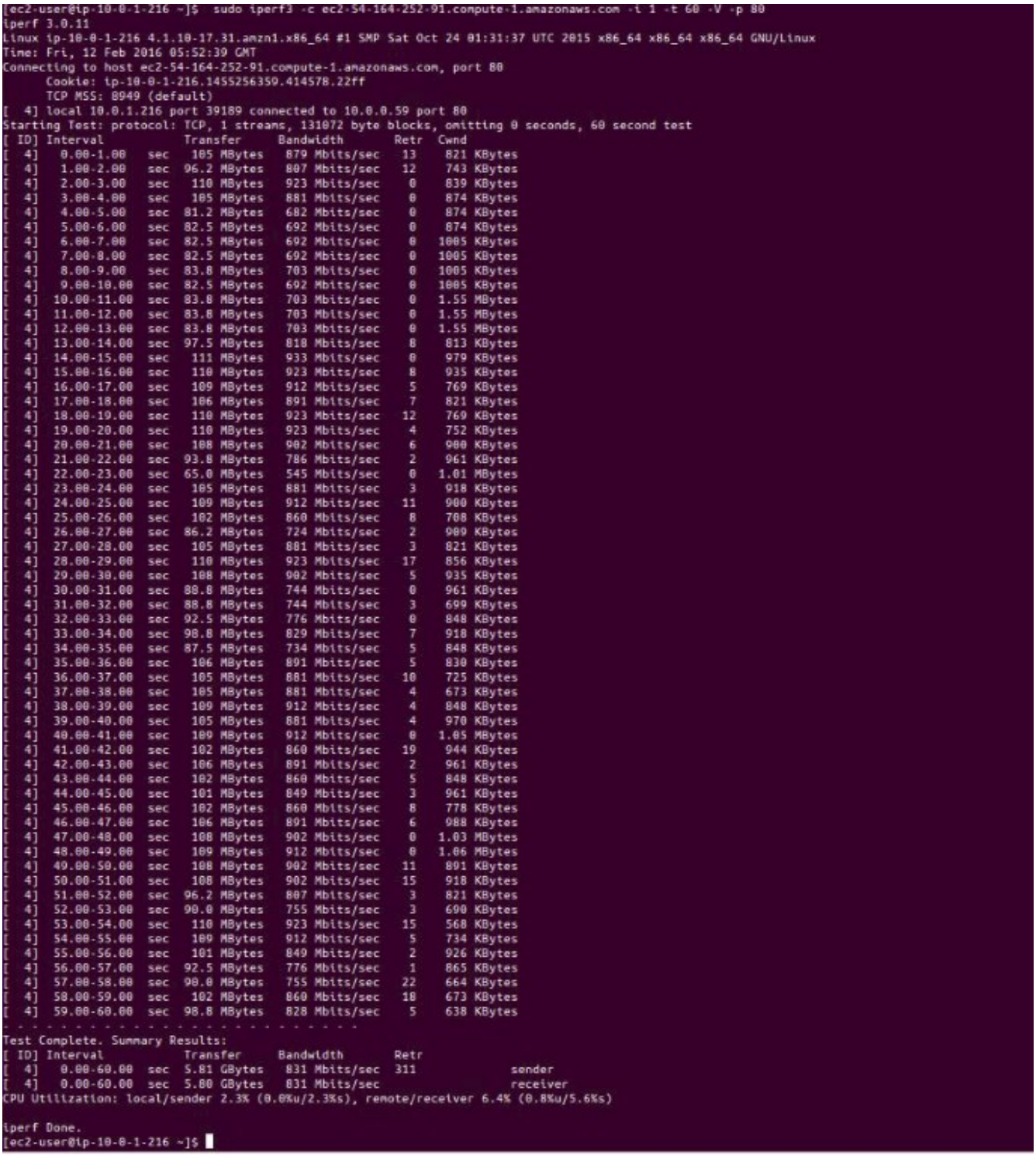
Here, in this Iperf, they have given Bandwidth and packets those are being transferred from transmitter to receiver side. In UDP, we can't send more than 65536 bytes package at the same time.

Moreover, in UDP, we need to send packets one by one and in TCP, we can send number of bytes together. So in screen-shot, we get the idea about that.

Efficiency comparison between Iperf and My result.

Efficiency = (759/979)*100 = 77.52%

TCP Graph:



```
[ec2-user@ip-10-0-1-216 ~]$  sudo iperf3 -c ec2-54-164-252-91.compute-1.amazonaws.com -i 1 -t 60 -V -p 80
iperf 3.0.11
Linux ip-10-0-1-216 4.1.10-17.31.amzn1.x86_64 #1 SMP Sat Oct 24 01:31:37 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
Time: Fri, 12 Feb 2016 05:52:39 GMT
Connecting to host ec2-54-164-252-91.compute-1.amazonaws.com, port 80
      Cookie: ip-10-0-1-216.1455256359.414578.22ff
      TCP MSS: 8949 (default)
[  4] local 10.0.1.216 port 39189 connected to 10.0.0.59 port 80
Starting Test: protocol: TCP, 1 streams, 131072 byte blocks, omitting 0 seconds, 60 second test
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   105 MBytes   879 Mbits/sec   13    821 KBytes
[  4]   1.00-2.00   sec  96.2 MBytes   807 Mbits/sec   12    743 KBytes
[  4]   2.00-3.00   sec   110 MBytes   923 Mbits/sec    0    839 KBytes
[  4]   3.00-4.00   sec   105 MBytes   881 Mbits/sec    0    874 KBytes
[  4]   4.00-5.00   sec  81.2 MBytes   682 Mbits/sec    0    874 KBytes
[  4]   5.00-6.00   sec  82.5 MBytes   692 Mbits/sec    0    874 KBytes
[  4]   6.00-7.00   sec  82.5 MBytes   692 Mbits/sec    0   1005 KBytes
[  4]   7.00-8.00   sec  82.5 MBytes   692 Mbits/sec    0   1005 KBytes
[  4]   8.00-9.00   sec  83.8 MBytes   703 Mbits/sec    0   1005 KBytes
[  4]   9.00-10.00  sec  82.5 MBytes   692 Mbits/sec    0   1005 KBytes
[  4]  10.00-11.00  sec  83.8 MBytes   703 Mbits/sec    0   1.55 MBytes
[  4]  11.00-12.00  sec  83.8 MBytes   703 Mbits/sec    0   1.55 MBytes
[  4]  12.00-13.00  sec  83.8 MBytes   703 Mbits/sec    0   1.55 MBytes
[  4]  13.00-14.00  sec  97.5 MBytes   818 Mbits/sec    8    813 KBytes
[  4]  14.00-15.00  sec   111 MBytes   933 Mbits/sec    0    979 KBytes
[  4]  15.00-16.00  sec   110 MBytes   923 Mbits/sec    8    935 KBytes
[  4]  16.00-17.00  sec   109 MBytes   912 Mbits/sec    5    769 KBytes
[  4]  17.00-18.00  sec   106 MBytes   891 Mbits/sec    7    821 KBytes
[  4]  18.00-19.00  sec   110 MBytes   923 Mbits/sec   12    769 KBytes
[  4]  19.00-20.00  sec   110 MBytes   923 Mbits/sec    4    752 KBytes
[  4]  20.00-21.00  sec   108 MBytes   902 Mbits/sec    6    900 KBytes
[  4]  21.00-22.00  sec  93.8 MBytes   786 Mbits/sec    2    961 KBytes
[  4]  22.00-23.00  sec  65.0 MBytes   545 Mbits/sec    0   1.01 MBytes
[  4]  23.00-24.00  sec   105 MBytes   881 Mbits/sec    3    918 KBytes
[  4]  24.00-25.00  sec   109 MBytes   912 Mbits/sec   11    900 KBytes
[  4]  25.00-26.00  sec   102 MBytes   860 Mbits/sec    8    708 KBytes
[  4]  26.00-27.00  sec  86.2 MBytes   724 Mbits/sec    2    909 KBytes
[  4]  27.00-28.00  sec   105 MBytes   881 Mbits/sec    3    821 KBytes
[  4]  28.00-29.00  sec   110 MBytes   923 Mbits/sec   17    856 KBytes
[  4]  29.00-30.00  sec   108 MBytes   902 Mbits/sec    5    935 KBytes
[  4]  30.00-31.00  sec  88.8 MBytes   744 Mbits/sec    0    961 KBytes
[  4]  31.00-32.00  sec  88.8 MBytes   744 Mbits/sec    3    699 KBytes
[  4]  32.00-33.00  sec  92.5 MBytes   776 Mbits/sec    0    848 KBytes
[  4]  33.00-34.00  sec  98.8 MBytes   829 Mbits/sec    7    918 KBytes
[  4]  34.00-35.00  sec  87.5 MBytes   734 Mbits/sec    5    848 KBytes
[  4]  35.00-36.00  sec   106 MBytes   891 Mbits/sec    5    830 KBytes
[  4]  36.00-37.00  sec   105 MBytes   881 Mbits/sec   10    725 KBytes
[  4]  37.00-38.00  sec   105 MBytes   881 Mbits/sec    4    673 KBytes
[  4]  38.00-39.00  sec   109 MBytes   912 Mbits/sec    4    848 KBytes
[  4]  39.00-40.00  sec   105 MBytes   881 Mbits/sec    4    970 KBytes
[  4]  40.00-41.00  sec   109 MBytes   912 Mbits/sec    0   1.05 MBytes
[  4]  41.00-42.00  sec   102 MBytes   860 Mbits/sec   19    944 KBytes
[  4]  42.00-43.00  sec   106 MBytes   891 Mbits/sec    2    961 KBytes
[  4]  43.00-44.00  sec   102 MBytes   860 Mbits/sec    5    848 KBytes
[  4]  44.00-45.00  sec   101 MBytes   849 Mbits/sec    3    961 KBytes
[  4]  45.00-46.00  sec   102 MBytes   860 Mbits/sec    8    778 KBytes
[  4]  46.00-47.00  sec   106 MBytes   891 Mbits/sec    6    988 KBytes
[  4]  47.00-48.00  sec   108 MBytes   902 Mbits/sec    0   1.03 MBytes
[  4]  48.00-49.00  sec   109 MBytes   912 Mbits/sec    0   1.06 MBytes
[  4]  49.00-50.00  sec   108 MBytes   902 Mbits/sec   11    891 KBytes
[  4]  50.00-51.00  sec   108 MBytes   902 Mbits/sec   15    918 KBytes
[  4]  51.00-52.00  sec  96.2 MBytes   807 Mbits/sec    3    821 KBytes
[  4]  52.00-53.00  sec  90.0 MBytes   755 Mbits/sec    3    690 KBytes
[  4]  53.00-54.00  sec   110 MBytes   923 Mbits/sec   15    568 KBytes
[  4]  54.00-55.00  sec   109 MBytes   912 Mbits/sec    5    734 KBytes
[  4]  55.00-56.00  sec   101 MBytes   849 Mbits/sec    2    926 KBytes
[  4]  56.00-57.00  sec  92.5 MBytes   776 Mbits/sec    1    865 KBytes
[  4]  57.00-58.00  sec  90.0 MBytes   755 Mbits/sec   22    664 KBytes
[  4]  58.00-59.00  sec   102 MBytes   860 Mbits/sec   18    673 KBytes
[  4]  59.00-60.00  sec  98.8 MBytes   828 Mbits/sec    5    638 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Test Complete. Summary Results:
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-60.00  sec  5.81 GBytes   831 Mbits/sec  311             sender
[  4]   0.00-60.00  sec  5.80 GBytes   831 Mbits/sec                  receiver
CPU Utilization: local/sender 2.3% (0.0%u/2.3%s), remote/receiver 6.4% (0.8%u/5.6%s)

iperf Done.
[ec2-user@ip-10-0-1-216 ~]$
```

Throughput (1 Thread) in MB/sec:

Number of blocks	Average(Throughput-TCP)	Standard Deviation(Throughput-TCP)	Average(Throughput-UDP)	Standard Deviation(Throughput-UDP)
1 B	1.023	0.324	2.085	1.1445
1 KB	328.614	27.76	452.225	29.897
64 KB	759.314	85.14	1310.124	110.255

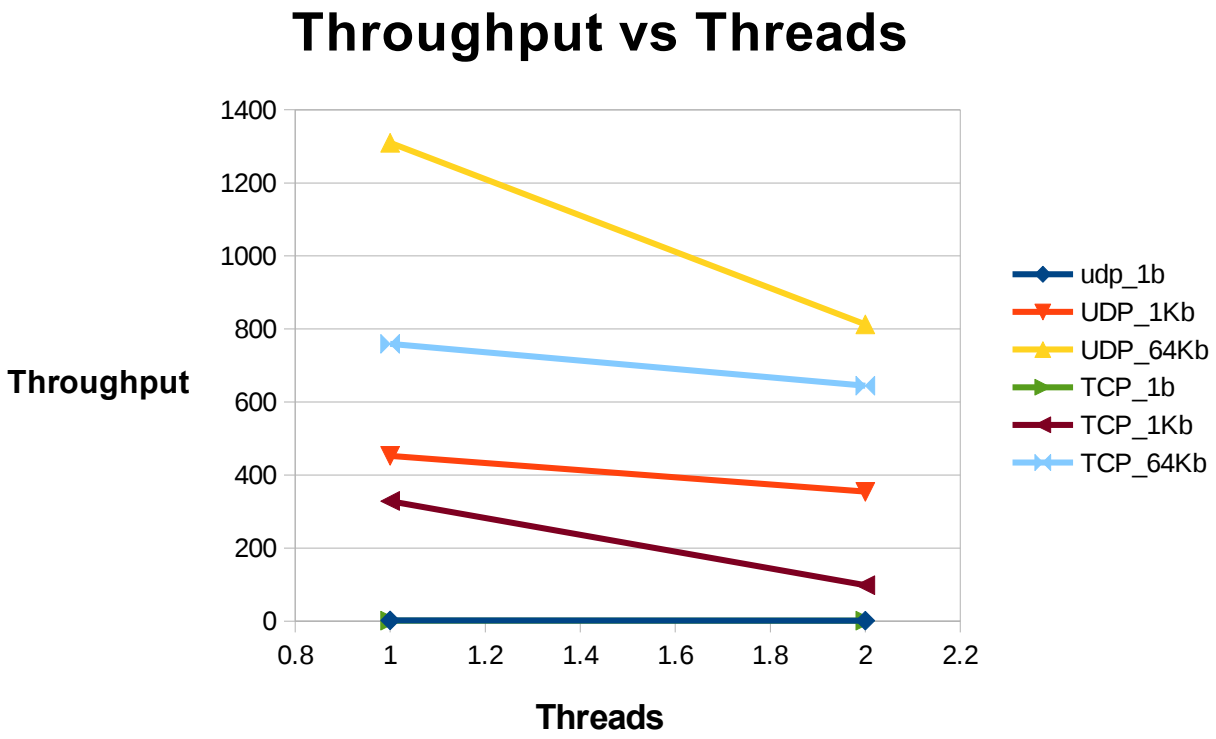
Latency (1 Thread) in ms:

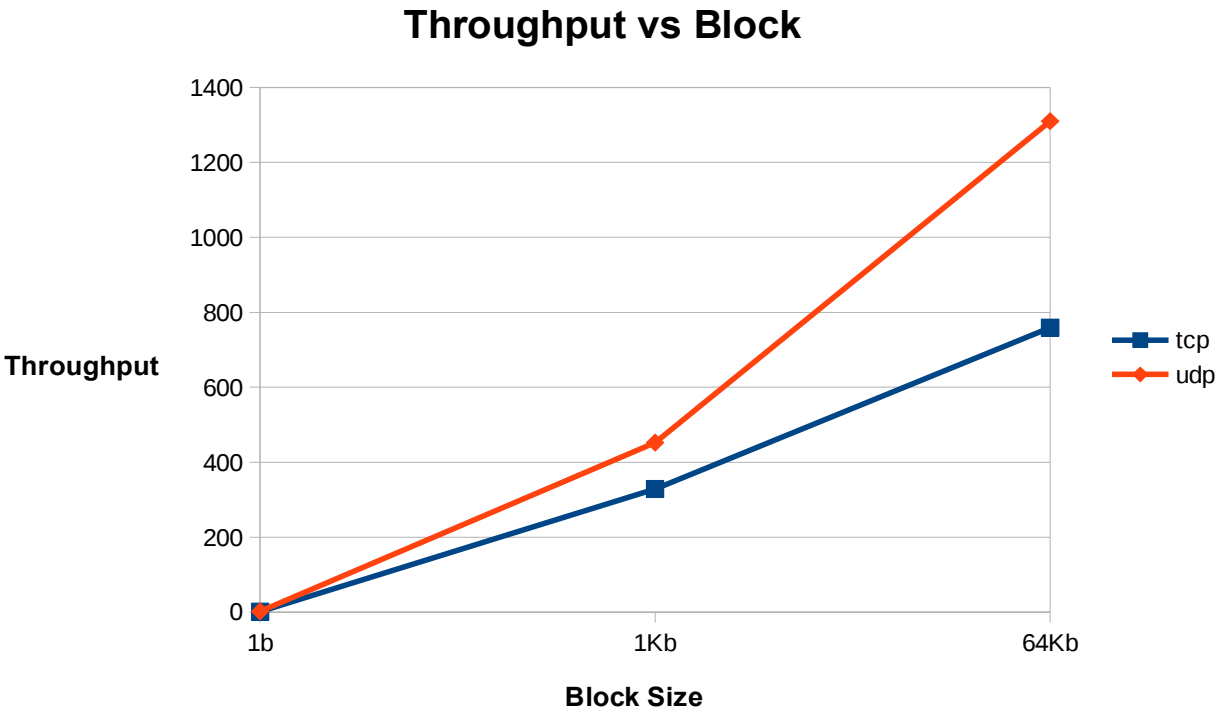
Number of blocks	Average(Latency-TCP)	Standard Deviation(Latency-TCP)	Average(Latency-UDP)	Standard Deviation(Latency-UDP)
1 B	0.147	0.005	0.077	0.0025
1 KB	0.506	0.002	0.498	0.05
64 KB	12.372	1.992	8.0079	1.56

Throughput result of TCP and UDP (results in MB):

	UDP		
Thread	1b	1Kb	64Kb
1	2.023	452.25	1310.14
2	1.085	354.233	812.5

	TCP		
Thread	1b	1Kb	64Kb
1	1.023	328.614	759.314
2	1.069	98.01	644.259


Throughput vs Threads
1400
1200
1000
800
600
400
200
0
0.8
1
1.2
1.4
1.6
1.8
2
2.2
Throughput
Threads
udp_1b
UDP_1Kb
UDP_64Kb
TCP_1b
TCP_1Kb
TCP_64Kb


Throughput vs Block
1400
1200
1000
800
600
400
200
0
1b
1Kb
64Kb
Throughput
Block Size
tcp
udp