Course: C553
Cloud Computing
PA 1
Benchmarking
Design Document and Performance

Name: Shushupti Vijay Ajmire
Prasanna Shanmuganathan

Design Document

This project aims at Benchmarking using KVM virtual machine m1.medium (2 virtual processors with 4GB RAM and 40GB disk), CentOS 7 Linux for the OS.

CPU Benchmarking

- For CPU Benchmarking, we used strong scaling experiment and design a source code in C language.
- This program will measure the processor speed, in terms of double precision floating point operations per second (GFLOPS) and integer operations per second (GIOPS).
- This program has 16 instruction which include addition operation on both integer and floating point numbers which will run for 1 billion times to measure the speed.
- Main aim of this is to utilize the complete CPU cycle by performing diverse types of instructions and measure the speed.
- There is a performance evaluation which evaluates IOPS and FLOPS executed per second using different levels of concurrency (1 thread, 2 threads, 4 threads, 8 threads)
- Also run this code with AVX instruction and adequate evaluation using bare metal provisioning.
- Ran the Linpack benchmark (http://en.wikipedia.org/wiki/LINPACK) and compare the performance achieved using double precision floating point.

Memory Benchmarking

- For Memory Benchmarking, we used strong scaling experiment and design a source code in C language.
- This program will measure the memory speed of host which will include parameters like read + write operations, sequential write access, random write access.
- This experiment used different block sizes (8B, 8KB, 8MB, 80MB), and varying the concurrency (1 thread, 2 threads, 4 threads, and 8 threads).
- We allocate 1 GB of memory to perform read + write operations in both sequential and random write access.
- The program also measures the throughput (Megabytes per second, MB/sec) and latency (microseconds, us). 8B block case used to measure latency, and the 8KB, 8MB, and 80MB cases used to measure throughput.
- Computed the theoretical memory bandwidth of your memory, based on the type of memory and the speed.
- Ran the Stream benchmark (http://www.cs.virginia.edu/stream/) and compared both the performance to see weather benchmarking is running properly.

Disk Benchmarking

- For Disk Benchmarking, we used strong scaling experiment and design a source code in C language.
- This program will measure the memory speed of host which will include parameters like read + write operations, sequential write access, random write access.
- This experiment used different block sizes (8B, 8KB, 8MB, 80MB), and varying the concurrency (1 thread, 2 threads, 4 threads, and 8 threads).
- It implements 4 methods, sequential read and write, random read and write.
- Allocated a large piece file of 1GB, and perform read + write or read operations on either sequential or random access within this 1GB file.
- Ran the IOZone benchmark (http://www.iozone.org/) and compared the performance achieved with theoretical performance.

Network Benchmarking

- For Network Benchmarking, we used strong scaling experiment and design a source code in Python language.
- It will measure the network speed over the loopback interface card with 1 node, between 2 processes on the same node and 2 nodes, between 2 processes on the different nodes.
- It includes the TCP protocol stack, UDP, fixed packet/buffer size (64KB), and varying the concurrency (1 thread, 2 threads, 4 threads, 8 threads) on both server and client side.
- It will measure the throughput (Megabits per second, Mb/sec) and latency (ms).
- We used fixed amount of data (1 GB) to transfer using socket programing and multithreading.
- Ran the IPerf benchmark (http://en.wikipedia.org/wiki/Iperf) and compared achieved throughput performance over the loopback interface.
- Compared efficiency with the theoretical memory performance.
- Repeated the benchmark across two nodes and compared the efficiency with theoretical network speed.

Performance

CPU Benchmarking

Performance is done on Chameleon testbud using KVM Virtual machine i.e m1.medium(2 virtual processors with 4GB RAM and 40GB disk.

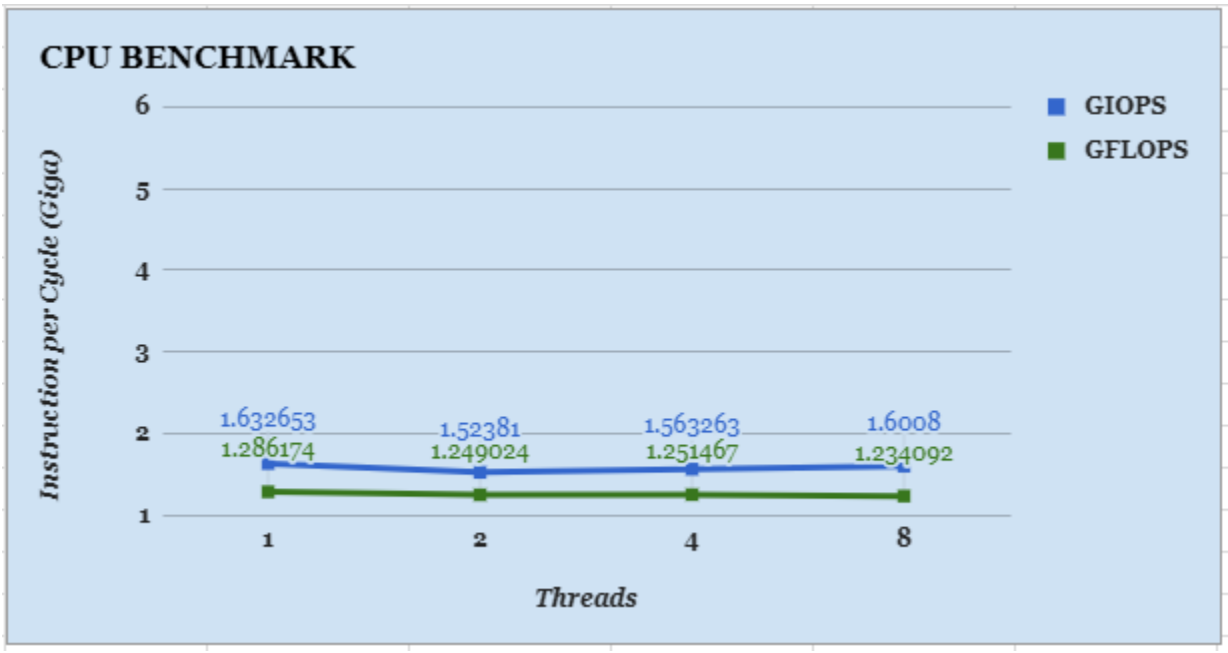
Specifications:

Flavor	m1.medium
Flavor ID	3
RAM	4GB
VCPUs	2 VCPU
Disk	40GB

In this experiment, we are benchmarking CPU performance in terms of GIOPS and GFLOPS. And the experiment is done across 1,2,4 and 8 threads.

Below graphs shows the performance of the experiments:

1. This graph shows the Average GIOPS(Integer operations) and GFLOPS(Floating point operations) for 1,2,4,8 threads :



The above graph shows that the instructions per cycle decreasing after 1 thread and getting stabilized for the rest. It cannot be proven that if the thread count increases, the performance increases as well which can also

been seen above where In case of 1 threads, both GFLOPS and GIOPS have a higher value and then maintaining similar performance for 2,4,8 threads respectively.

Test Results:

	GIOPS				GLOPS			
	1 thread	**2 thread**	**4 thread**	**8 thread**	**1 thread**	**2 thread**	**4 thread**	**8 thread**
Exp 1	1.632653	1.52381	1.563263	1.6008	1.286174	1.249024	1.251467	1.234092
Exp 2	1.596806	1.585728	1.59355	1.606023	1.244168	1.273885	1.24805	1.234806
Exp 3	1.62783	1.59542	1.602589	1.619758	1.269841	1.24031	1.273885	1.237424
Average	1.619096333	1.568319333	1.586467333	1.608860333	1.266727667	1.254406333	1.257800667	1.235440667
Std Dev	0.01588415463	0.03172059564	0.01681778214	0.0079953856	0.01728960463	0.01422550035	0.01145857194	0.001432400627

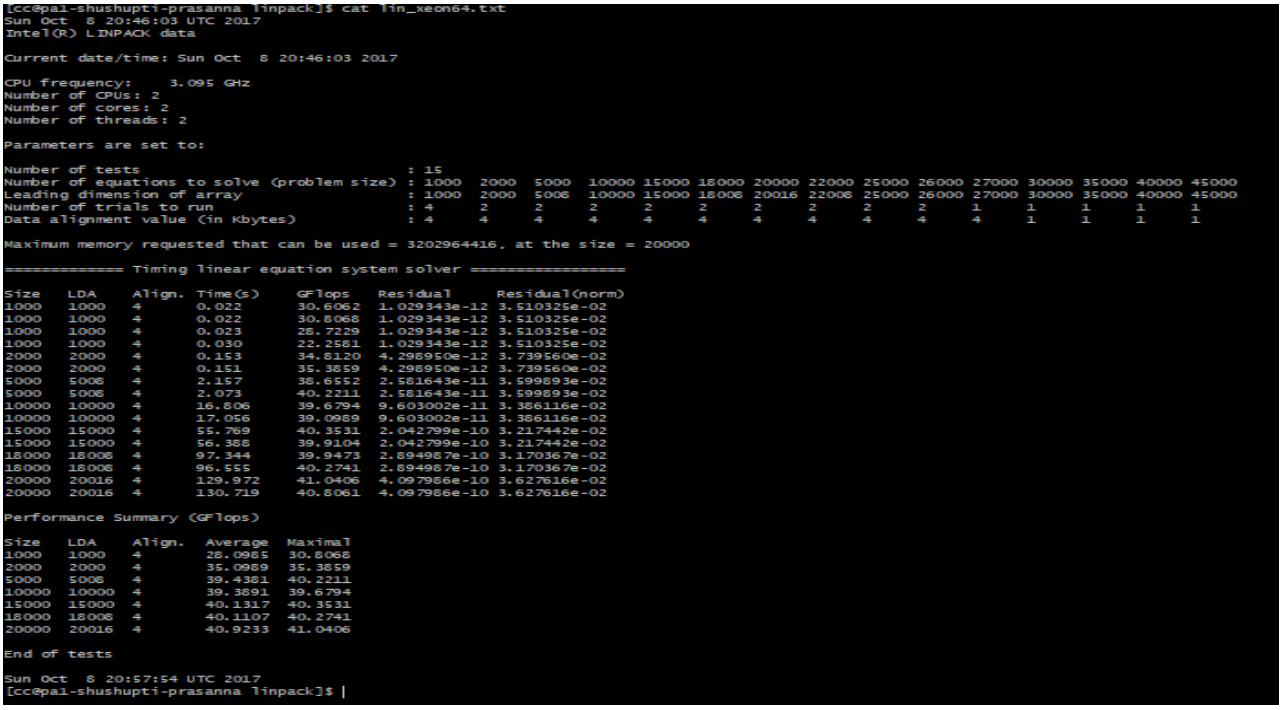
Theoretical Performance

FLOPS= Number of Cores * CLOCK SPEED * IPC

= 2* 2299.996 *10^6* 8 = 35.93 GFLOPS

Linpack Benchmark:

We ran the LINPARK Benchhmark on the system to compare the CPU performance and from the below it can be seen that the maximal value is obtained at 41.0406 GFLOPS which is 95% matching with the theoretical performance

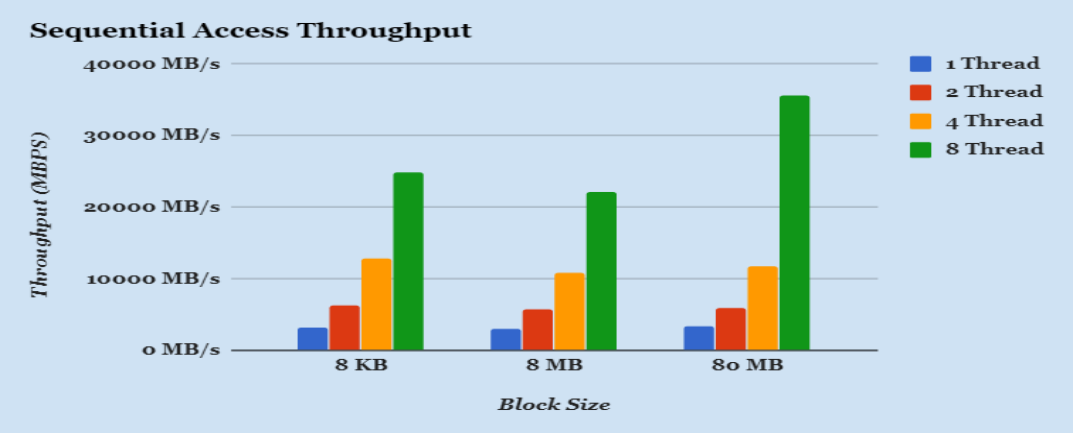


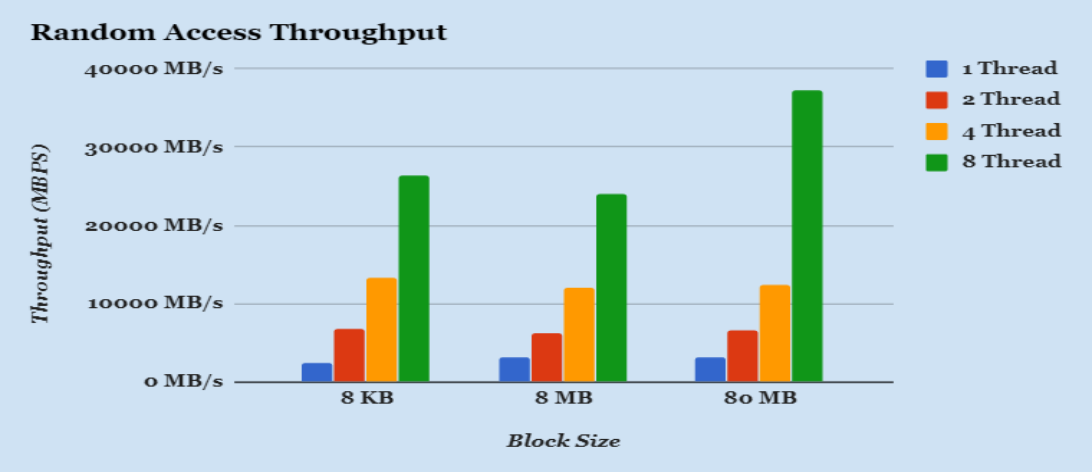
```
[cc@pa1-shushupti-prasanna linpack]$ cat lin_xeon64.txt
Sun Oct  8 20:46:03 UTC 2017
Intel(R) LINPACK data

Current date/time: Sun Oct  8 20:46:03 2017

CPU frequency:    3.095 GHz
Number of CPUs: 2
Number of cores: 2
Number of threads: 2

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 = 3202964416, at the size = 20000

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

Size   LDA    Align. Time(s)    GFlops   Residual     Residual(norm)
1000   1000   4      0.022      30.6062  1.029343e-12 3.510325e-02
1000   1000   4      0.022      30.8068  1.029343e-12 3.510325e-02
1000   1000   4      0.023      28.7229  1.029343e-12 3.510325e-02
1000   1000   4      0.030      22.2581  1.029343e-12 3.510325e-02
2000   2000   4      0.153      34.8120  4.298950e-12 3.739560e-02
2000   2000   4      0.151      35.3859  4.298950e-12 3.739560e-02
5000   5008   4      2.157      38.6552  2.581643e-11 3.599893e-02
5000   5008   4      2.073      40.2211  2.581643e-11 3.599893e-02
10000  10000  4      16.806     39.6794  9.603002e-11 3.386116e-02
10000  10000  4      17.056     39.0989  9.603002e-11 3.386116e-02
15000  15000  4      55.769     40.3531  2.042799e-10 3.217442e-02
15000  15000  4      56.388     39.9104  2.042799e-10 3.217442e-02
18000  18008  4      97.344     39.9473  2.894987e-10 3.170367e-02
18000  18008  4      96.555     40.2741  2.894987e-10 3.170367e-02
20000  20016  4      129.972    41.0406  4.097986e-10 3.627616e-02
20000  20016  4      130.719    40.8061  4.097986e-10 3.627616e-02

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
1000   1000   4       28.0985  30.8068
2000   2000   4       35.0989  35.3859
5000   5008   4       39.4381  40.2211
10000  10000  4       39.3891  39.6794
15000  15000  4       40.1317  40.3531
18000  18008  4       40.1107  40.2741
20000  20016  4       40.9233  41.0406

End of tests

Sun Oct  8 20:57:54 UTC 2017
[cc@pa1-shushupti-prasanna linpack]$
```

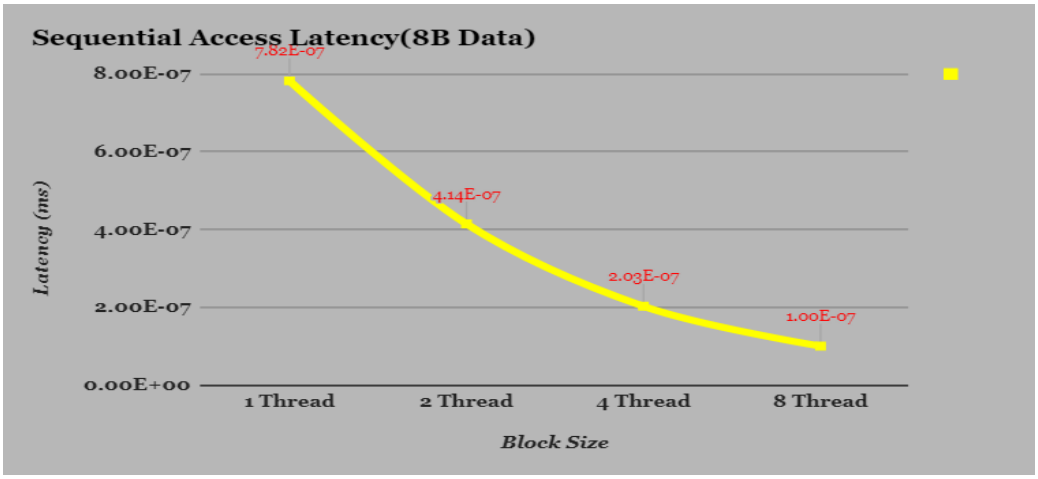
Memory Benchmarking

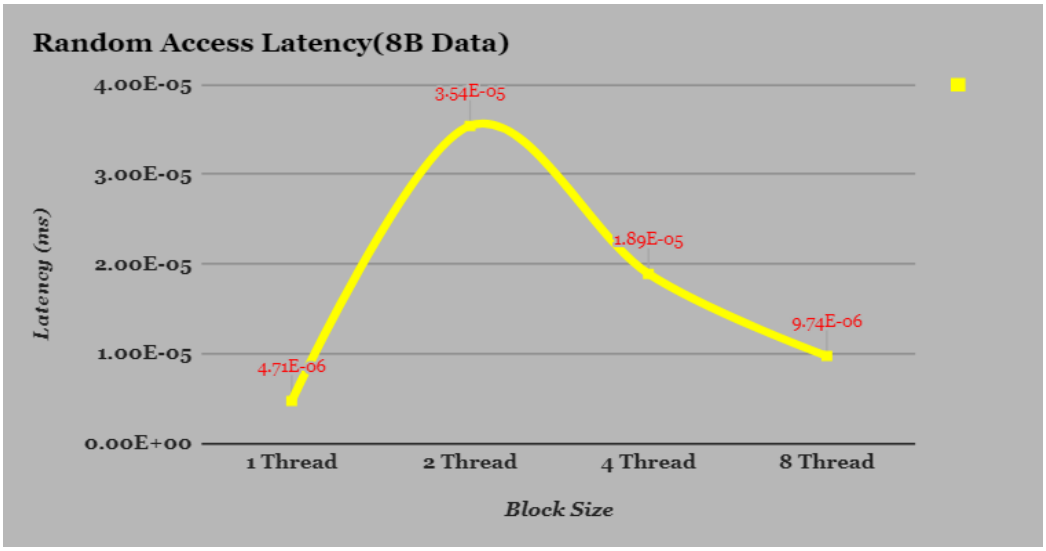
In this experiment, we are performing Memory Benchmarking across 1,2,4 and 8 threads to measure the efficiency of concurrency. Thread operations are done on different block size of data - 8B,8KB,8MB and 80MB and the operations are Sequential and random read+write.

Below are the graphs showing the experimental results of the program we written on C:









Throughputs were measured for 8kb,8MB and 80MB cases whereas latency was plotted for 8B block-size case. As we see from the above graphs, we could see that the throughput increasing as we increase the number of the threads. And the Latency is also decreasing on the thread increase. With sequential read access, the latency has seen a gradual decrease. However, with random access there was increase in latency upon using 2 threads to perform the read+write operation.

Test Results :

Throughput:

We took 3 experiment results for statistics :

Test Results for 8B								
	Sequential R+W				Random R+W			
	1 Thread	2Thread	4Thread	8Thread	1 Thread	2Thread	4Thread	8Thread
Exp 1	1219.047619	2301.123596	4708.045977	9525.581395	202.371542	26.933193	50.44335	97.931859
Exp 2	1000	1684.210526	3368.421053	7111.111111	216.216216	26.845638	89.761571	92.352092
Exp 3	1000	1777.777778	3368.421053	6736.842105	219.178082	27.610009	53.333333	100.946372
Average	1073.015873	1921.0373	3814.962694	7791.178204	212.5886133	27.12961333	64.51275133	97.07677433
Standard Dev.	103.2600379	271.4625842	631.5052453	1235.889732	7.32505165	0.3415664571	17.89255285	3.560317357

Test Results for 8 KB								
	Sequential R+W				Random R+W			
	1 Thread	2Thread	4Thread	8Thread	1 Thread	2Thread	4Thread	8Thread
Exp 1	3200	6206.060606	12800	24824.24242	2438.095238	6826.666667	13212.90323	26425.80645
Exp 2	3200	6400	10666.66667	21333.33333	2666.666667	4571.428571	10666.66667	21333.33333
Exp 3	3200	6400	10666.66667	24824.24242	3200	4571.428571	10666.66667	21333.33333
Average	3200	6335.353535	11377.77778	23660.60606	2768.253968	5323.174603	11515.41219	23030.82437
Standard Dev.	0	91.42390709	1005.662976	1645.630327	319.2331451	1063.129434	1200.307425	2400.614851

Test Results for 8 MB								
	Sequential R+W				Random R+W			
	1 Thread	2Thread	4Thread	8Thread	1 Thread	2Thread	4Thread	8Thread
Exp 1	3011.764706	5688.888889	10778.94737	22140.54054	3103.030303	6206.060606	12047.05882	24094.11765
Exp 2	2666.666667	4571.428571	9142.857143	21333.33333	2666.666667	5333.333333	10666.66667	25600
Exp 3	2666.666667	5333.333333	10666.66667	25600	3200	4571.428571	9142.857143	21333.33333
Average	2781.699347	5197.883598	10196.15706	23024.62462	2989.89899	5370.27417	10618.86088	23675.81699
Standard Dev.	162.6807757	466.146875	746.2047426	1850.642162	231.9628296	667.8467596	1186.117164	1766.794215

Test Results for 80 MB								
	Sequential R+W				Random R+W			
	1 Thread	2Thread	4Thread	8Thread	1 Thread	2Thread	4Thread	8Thread
Exp 1	3413.333333	5851.428571	11702.85714	35617.3913	3200	6606.451613	12412.12121	37236.36364
Exp 2	3200	5851.428571	11702.85714	34133.33333	3303.225806	6400	13653.33333	37236.36364
Exp 3	3103.030303	6023.529412	11377.77778	34133.33333	3303.225806	6826.666667	12800	39009.52381
Average	3238.787879	5908.795518	11594.49735	34628.01932	3268.817204	6611.039427	12955.15152	37827.41703
Standard Dev.	129.6157584	81.12911448	153.2438799	699.5916362	48.66111161	174.2161439	518.4630336	835.8757202

Latency:

Test Results for 8B								
	Sequential R+W				Random R+W			
	1 Thread	2Thread	4Thread	8Thread	1 Thread	2Thread	4Thread	8Thread
Exp 1	7.82E-07	4.14E-07	2.03E-07	1.00E-07	4.71E-06	3.54E-05	1.89E-05	9.74E-06
Exp 2	7.26E-07	4.05E-07	1.98E-07	1.02E-07	4.45E-06	3.60E-05	1.67E-05	5.34E-06
Exp 3	7.54E-07	4.05E-07	1.98E-07	1.02E-07	4.49E-06	3.33E-05	1.79E-05	8.07E-06
Average	7.54E-07	4.08E-07	1.99E-07	1.02E-07	4.55E-06	3.49E-05	1.78E-05	7.72E-06
Standard Dev.	2.28E-08	4.39E-09	2.20E-09	1.10E-09	1.15E-07	1.19E-06	8.87E-07	1.81E-06

STREAM BENCHMARK:

Below is the screenshot of STREAM benchmarking showing the results of Memory Speeds.

The result below shows the memory speed having a total of 26.54GB/s

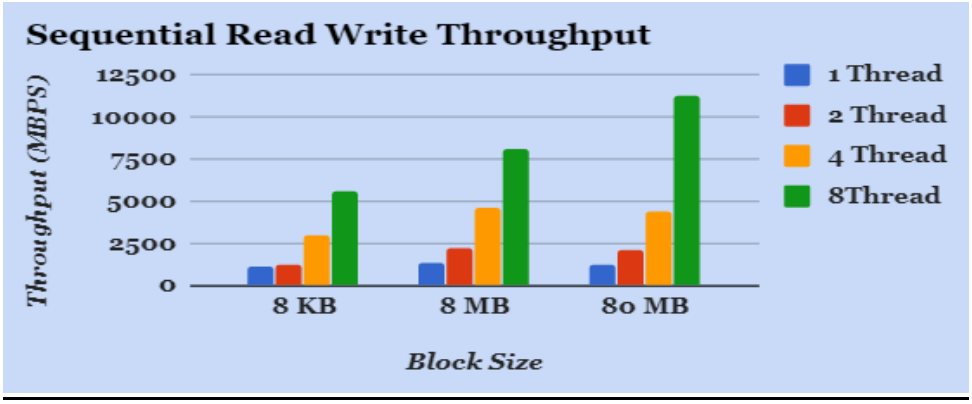
```
[cc@pa1-shushupti-prasanna ~]$ gcc -o stream stream.c
[cc@pa1-shushupti-prasanna ~]$ ./stream
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
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 32410 microseconds.
   (= 32410 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:           5843.3     0.029205     0.027382     0.031202
Scale:          5659.3     0.029841     0.028272     0.031645
Add:            7962.9     0.030738     0.030140     0.032213
Triad:          7673.9     0.032267     0.031275     0.033335
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

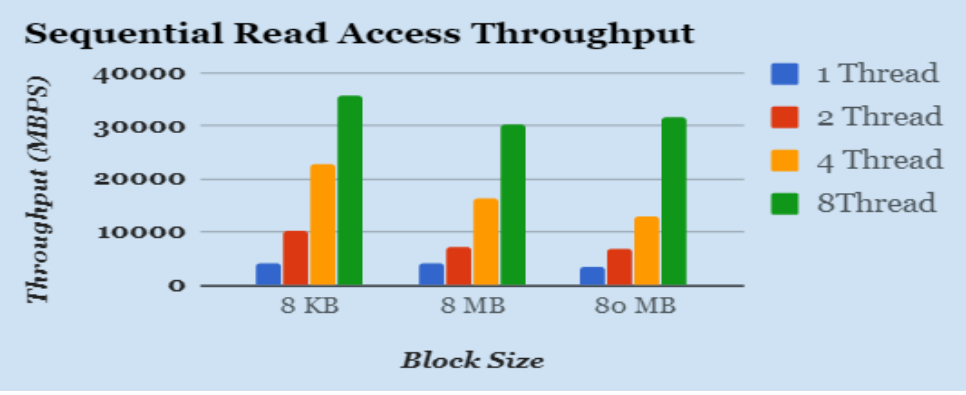
Disk Benchmarking

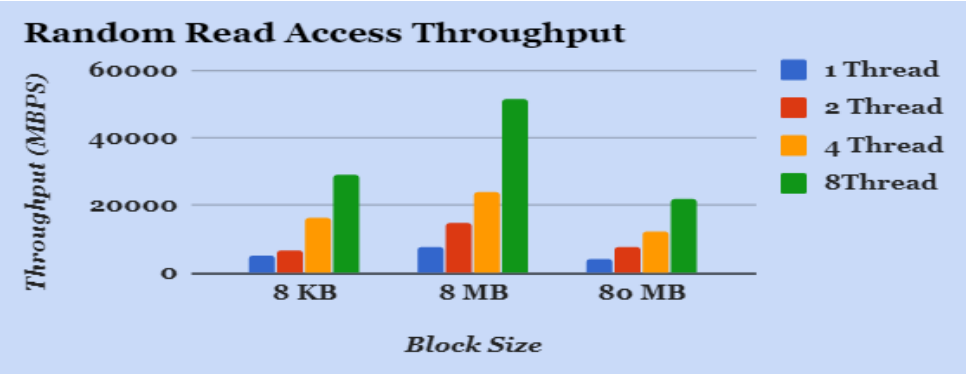
In this experiment, we are performing Disk Benchmarking across 1,2,4 and 8 threads to measure the efficiency of concurrency. Thread operations are done on different block size of data - 8B,8KB,8MB and 80MB and the operations are Sequential read+write, sequential read access and random read access. We have taken a 1GB file to perform the read/write operations on the disk.

Below are the graphs showing the experimental results of the program we written on C:

Throughput are plotted for 8 KB,8 MB and 80 MB cases.



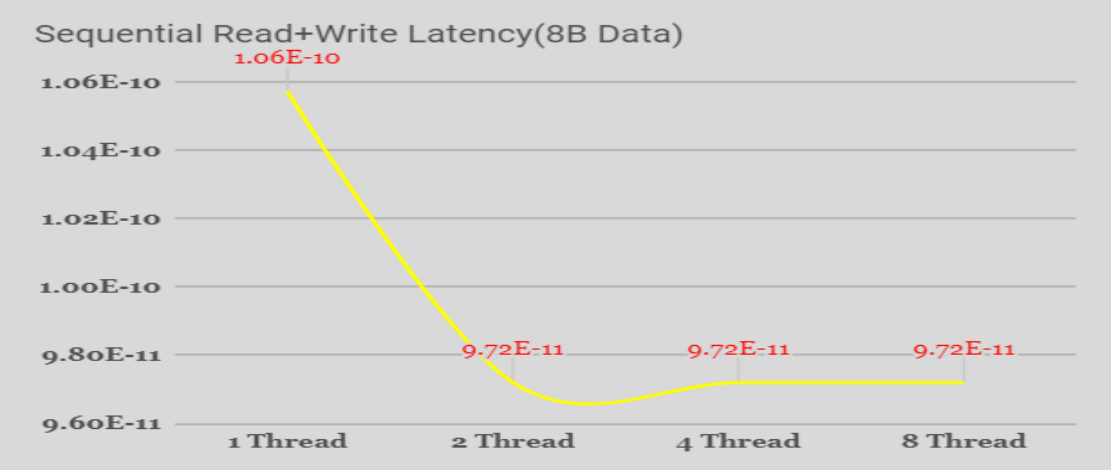


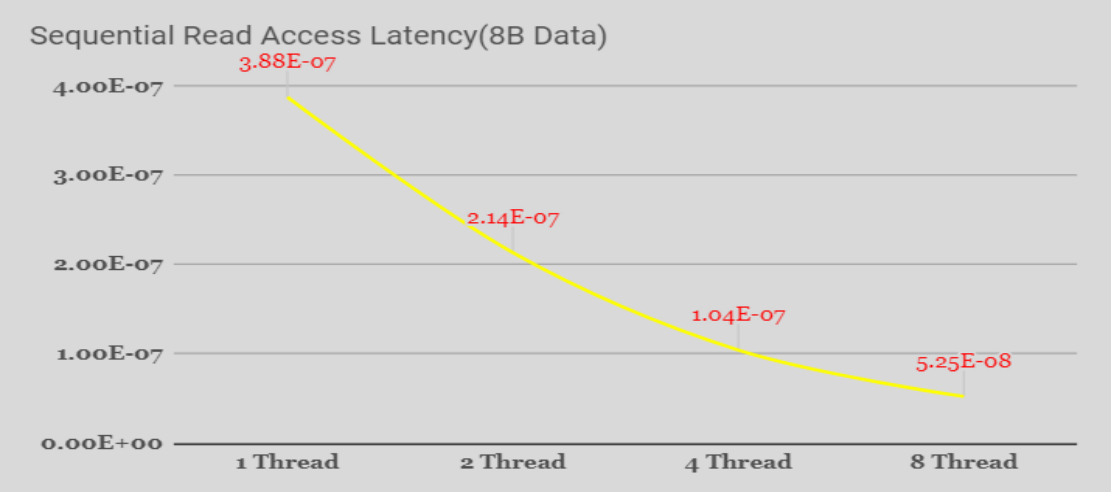


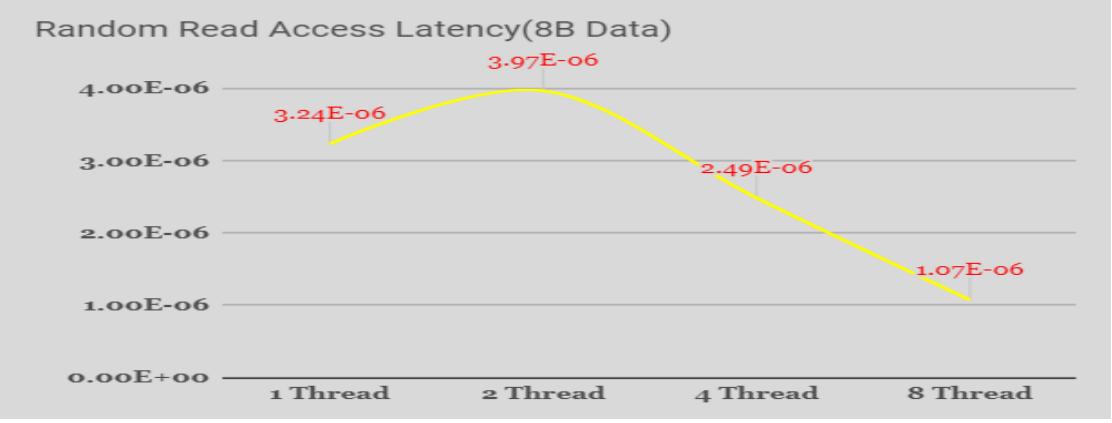
From the above graphs, it is seen that the throughput increases as we increase the number of threads.

Latency:

Latency is plotted considering the 8B case







From the above graphs it can be seen that the latency keeps decreasing as we increase the number of threads. However for random read access, they seem to increase at use of 2 threads but the latency decreased as we increased the threads to 4 and 8.

Test Results:

	Sequential Read+Write - 8KB			
	1 Thread	**2Thread**	**4Thread**	**8Thread**
Exp 1	1101.075269	1197.660819	3011.764706	5610.958904
Exp 2	975.238095	1226.347305	2712.582781	5572.789116
Exp 3	1101.075269	1044.897959	2748.993289	5497.986577
Average	1059.129544	1156.302028	2824.446925	5560.578199
Standard Dev.	59.32021271	79.64035203	133.2851442	46.92203844

Test Results for 8KB								
	Sequential Read				Random Read			
	1 Thread	**2Thread**	**4Thread**	**8Thread**	**1 Thread**	**2Thread**	**4Thread**	**8Thread**
Exp 1	4096	10240	22755.55556	35617.3913	5120	6826.666667	16384	29257.14286
Exp 2	4654.545455	8904.347826	19504.76191	37236.36364	6826.666667	9309.090909	17808.69565	35617.3913
Exp 3	4096	8192	17808.69565	45511.11111	7314.285714	8533.333333	17808.69565	35617.3913
Average	4282.181818	9112.115942	20023.00437	39454.95535	6420.31746	8223.030303	17333.7971	33497.30849
Standard Dev.	263.3008526	848.9019152	2052.524816	4333.05395	940.7664276	1036.926068	671.6079702	2998.249868

	Sequential Read+Write - 8MB			
	1 Thread	**2Thread**	**4Thread**	**8Thread**
Exp 1	1312.820513	2226.086957	4602.247191	8110.891089
Exp 2	1296.202532	2250.549451	4266.666667	8192
Exp 3	1296.202532	2409.411765	4357.446809	8192
Average	1301.741859	2295.349391	4408.786889	8164.963696
Standard Dev.	7.83379137	81.27021707	141.7284372	38.23510732

Test Results for 8MB								
	Sequential Read				Random Read			
	1 Thread	2Thread	4Thread	8Thread	1 Thread	2Thread	4Thread	8Thread
Exp 1	4266.666667	7062.068966	16384	30340.74074	7876.923077	14628.57143	24094.11765	51200
Exp 2	4654.545455	7062.068966	15170.37037	30340.74074	10240	14628.57143	29257.14286	54613.33333
Exp 3	4654.545455	9309.090909	17066.66667	30340.74074	8533.333333	17066.66667	27306.66667	51200
Average	4525.252526	7811.07628	16207.01235	30340.74074	8883.418803	15441.26984	26885.97573	52337.77778
Standard Dev.	182.8478142	1059.256302	784.2101701	3.64E-12	995.9762425	1149.329118	2128.683941	1609.060763

	Sequential Read+Write - 80MB			
	1 Thread	2Thread	4Thread	8Thread
Exp 1	1264.197531	2068.686869	4452.173913	11221.91781
Exp 2	1204.705882	2202.150538	4602.247191	11538.02817
Exp 3	1066.666667	2007.843137	4179.591837	11377.77778
Average	1178.52336	2092.893515	4411.337647	11379.24125
Standard Dev.	82.73956692	81.15135157	174.947778	129.055663

Test Results for 80MB								
	Sequential Read				Random Read			
	1 Thread	2Thread	4Thread	8Thread	1 Thread	2Thread	4Thread	8Thread
Exp 1	3531.034483	6826.666667	12800	31507.69231	4096	7585.185185	12412.12121	22140.54054
Exp 2	3531.034483	6400	13212.90323	31507.69231	3938.461538	7585.185185	11702.85714	18204.44444
Exp 3	3531.034483	6606.451613	12412.12121	31507.69231	4266.666667	7876.923077	14628.57143	32768
Average	3531.034483	6611.039427	12808.34148	31507.69231	4100.376068	7682.431149	12914.51659	24370.99499
Standard Dev.	0	174.2161439	326.9710957	0	134.0249085	137.5265612	1246.127834	6151.177824

Latency:

	Sequential Read+Write - 8B			
	1 Thread	2Thread	4Thread	8Thread
Exp 1	1.19E-06	5.05E-07	2.47E-07	1.25E-07
Exp 2	9.48E-07	4.84E-07	2.44E-07	1.27E-07
Exp 3	8.64E-07	4.90E-07	2.42E-07	1.25E-07
Average	1.00E-06	4.93E-07	2.44E-07	1.25E-07
Standard Dev.	1.38E-07	9.03E-09	2.27E-09	8.09E-10

Test Results for 8B								
	Sequential Read				Random Read			
	1 Thread	2Thread	4Thread	8Thread	1 Thread	2Thread	4Thread	8Thread
Exp 1	3.88E-07	2.14E-07	1.04E-07	5.25E-08	3.24E-06	3.97E-06	2.49E-06	1.07E-06
Exp 2	4.40E-07	2.17E-07	1.06E-07	5.28E-08	3.25E-06	4.39E-06	2.04E-06	9.91E-07
Exp 3	3.86E-07	2.10E-07	1.02E-07	5.09E-08	3.57E-06	4.26E-06	2.29E-06	1.02E-06
Average	4.05E-07	2.14E-07	1.04E-07	5.21E-08	3.35E-06	4.21E-06	2.27E-06	1.03E-06
Standard Dev.	2.50E-08	2.85E-09	1.71E-09	8.32E-10	1.54E-07	1.75E-07	1.83E-07	3.47E-08

IoZone Benchmarking:

The following screenshot shows the performance metrics taken from IoZone benchmarking ran on the system. The speeds are measured in terms of Kilobytes. The first image shows the speeds of all values which gets run when executing the benchmark. The second image is for a specific case where we have given the filesize.

cc@pa1-shushupti-prasanna:~/iozone3_471/src/current

```
        Run began: Sun Oct  8 21:26:51 2017

        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   673098  1232453 3791156   4018152 2801873 1330167  1780008   1638743   2067979  1330167  1421755 2358717  3203069
              64       8   889431  1599680 5860307   5389653 4274062 1722886  2203800   2133730   2892445  1562436  1599680 2772930  4274062
              64      16   811459  1119387 5735102   7100397 4988978 1991276  2278628   1991276   2772930  1562436  1828508 2892445  4564786
              64      32   901377  1734015 5860307   5860307 4274062 1991276  2278628   1232453   3057153  1734015  2203800 3165299  4897948
              64      64   952555  1722886 6421025   7100397 4564786 1991276  2358717   1991276   2801873  1879725  1933893 3363612  5283570
             128       4   853794  1471662 2132084   4135958 3199360 1487977  2202044   1905110   2464907  1598754  1504659 2969325  3560017
             128       8   962469  1802755 5325799   5603747 4407601 1975201  2920861   2511022   4267461  1852520  1878447 4012317  4717434
             128      16   809997  1618028 5784891   6406138 5122535 2326073  3468030   2727923   4407601  2202044  2238774 4135958  5325799
             128      32  1185654  2511022 5545860   6045455 5325799 2511022  3657016   2905056   4759253  2727923  2286447 4104338  2608629
             128      64   955616  2166499 5847904   6727225 5325799 2558895  3468030   2784517   2367096  1878447  2969325 3895854  3380677
             128     128  1022989  2166499 6727225   6727225 5847904 2673584  2248149   2621367   4012317  2409592  1939522 4012317  5122535
             256       4   886067  1327564 4281170   4496299 3454704 1292410  2872459   2114473   3009318  1752173  1610276 3410808  3756894
             256       8   873809  2015259 5320671   5938650 4840911 2330140  3605511   1985448   3823789  1802167  1766587 3705040  5569035
             256      16  1157291  2371308 6249745   4734192 5569035 2639446  3935921   2819657   3605511  2210228  2210228 4422226  5347168
             256      32  1195962  1652404 3935921   5569035 5117791 2665657  3935921   3078337   4572895  2413957  1740810 4553502  5242734
             256      64  1219045  2419396 5687020   5142301 3823789 2849590  1487577   2975955   5117791  2371308  2719671 4652146  5320671
             256     128  1206714  2305128 3879045   5810112 5455847 3067188  4117018   3368013   4907284  2019049  3454704 4572895  5320671
             256     256   870974  2392442 5455847   5569035 5320671 2783115  2783115   2719671   4332998  2783115  2726577 4907284  5117791
             512       4   932550  1662389 4030519   4163357 3239989 1505098  2942541   2178404   3029721  1594501  1547395 2975154  2497638
             512       8  1007827  2073249 5115422   5624545 4784885 2275345  3510074   2584820   4195896  2132967  1712772 4871723  5331314
             512      16  1221684  2317080 5822804   5951911 4447925 2613128  4494470   3459188   4742616  2285029  1848401 5227492  5624545
             512      32  1171691  2073249 3678421   5624545 5227492 2845081  4871723   2910635   5115422  2438090  1940253 4131319  5384786
             512      64  1078159  2544997 5822804   5951911 5744919 3029721  3970696   3481620   5398323  2668325  2651850 5019764  5624545
             512     128  1248678  2706713 4532414   3877684 5509113 2497638  3877684   3556580   5331314  2285029  2926501 4660280  6414119
             512     256  1289930  2753870 5760329   5886650 5439343 3239989  4973263   3264616   5278892  1821745  3762197 4784885  5278892
             512     512  1017858  2535981 3684733   5684095 5384786 2958758  4742616   3182373   4571003  3177664  2975154 5019764  5384786
            1024       4  1021928  1638171 4029784   4249053 2737730 1646334  2572135   2160657   3020783  1837965  1525848 3835457  3941039
            1024       8  1120288  2173780 6128613   5593820 3880507 2333199  4410497   2985091   4550690  2031864  1954207 5200941  5356618
            1024      16  1311117  2305644 5821271   6059442 5991815 2869422  5789881   3778101   4876180  2491561  2480051 3738636  4079544
            1024      32  1311117  2573677 5623115   4995276 5479632 2737730  5417427   3970183   4949226  2708381  2708381 5789881  4574926
            1024      64  1343938  2432298 5821271   5917516 5782087 2976816  5330028   3908759   4920874  2358827  2653170 5925680  5444898
            1024     128  1399100  2375790 4232305   5853003 4654248 3210456  4130547   3271594   5660167  2716948  2715230 4300102  5623115
            1024     256  1374474  2370545 4762630   5472650 5479632 3103735  4995276   3579719   5336651  2708381  2985091 5194651  4898425
            1024     512  1238168  2791104 5500686   5564829 5536137 3083680  5048117   3379719   5226256  2102484  3880507 3631168  5390231
            1024    1024  1192117  2552265 4805258   5593820 5251818 3092561  5096034   3130886   5652717  3314514  2730767 4926518  4634161
            2048       4   941176  1823453 3962502   3901313 3121562 1859773  2920945   2397544   3324528  1725646  1344529 3938881  3730203
            2048       8  1265122  2098276 4686767   5640860 4284671 2484934  4086930   3271351   5041616  2387548  2164908 5349298  4911886
            2048      16  1371798  2412357 5904523   6058611 4490751 2824887  5212953   3374151   5719737  2479196  2260624 5750369  5916724
            2048      32  1145965  2400894 5578583   5107566 5517666 2805512  4934459   3764533   5251194  2522144  2423929 4807425  5596757
            2048      64  1261036  1832009 4983122   5920802 5689430 3209024  5593112   3374151   5461535  2209455  1933883 5389574  6464402
            2048     128  1163663  2578935 6058611   6243566 5578583 3374151  4707314   4094723   5251194  2676988  2470639 4751580  5644567
            2048     256  1380173  2060032 4889518   5475461 5489457 3002627  4623699   3483621   5447681  2540795  2531808 5567735  5852228
            2048     512  1444219  1890887 5492967   5535444 5032754 2994254  4807425   3414387   4643695  2920945  2572756 4601407  5475461
            2048    1024  1183707  2045804 4831762   5640860 5611381 2981782  4772701   3418463   5447681  2235910  3682233 4720248  5582208
            2048    2048  1204792  2619037 5159722   5769681 6189580 3112514  3885431   2933915   5403134  2999482  2884652 5293261  5156624
```

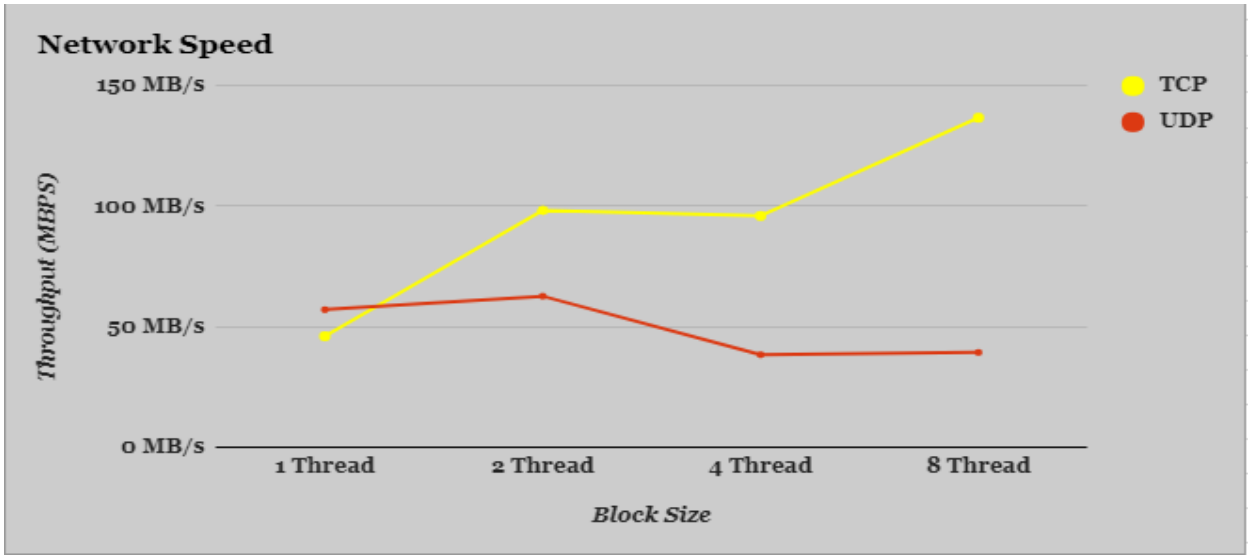
```
[cc@pa1-shushupti-prasanna current]$ ./iozone -g# -s 8192
        Iozone: Performance Test of File I/O
                Version $Revision: 3.471 $
                Compiled for 64 bit mode.
                Build: linux-ia64

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

        Run began: Tue Oct 10 02:04:32 2017

        Using maximum file size of 4 kilobytes.
        File size set to 8192 kB
        Command line used: ./iozone -g# -s 8192
        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
            8192       4   653838  1373576 4222755   4576535 3497955 1858005  2583282   2383476   3415886  1984069  1460016 3291223  4011735

iozone test complete.
```

Performance tested for 8MB(8192 KB) file size :

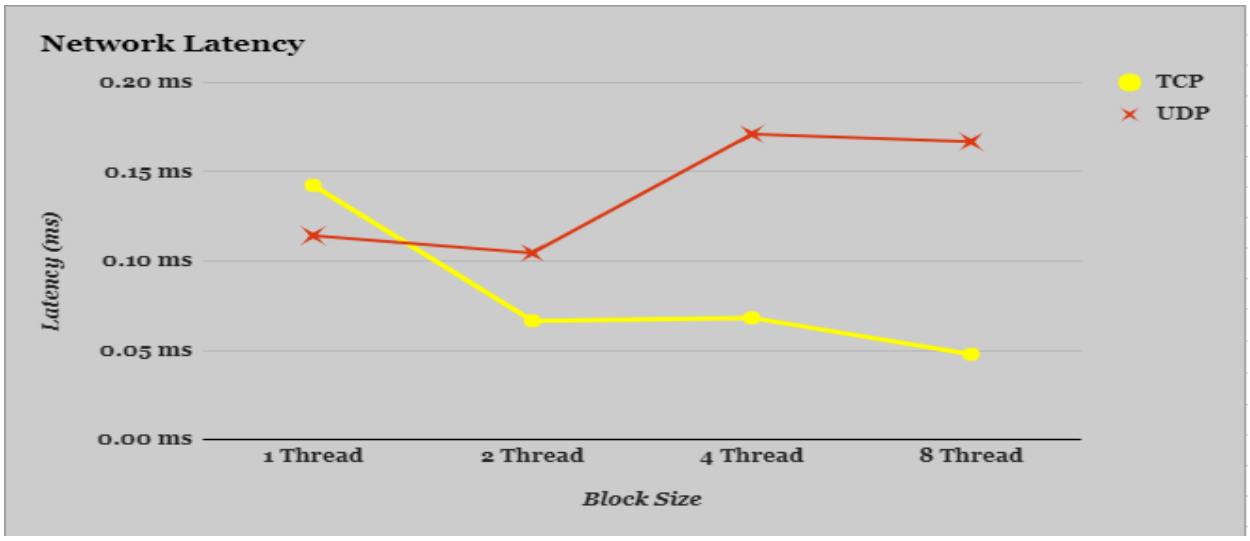
From the above we can note down the speed performances, sequential write -0.62 Gb/s, sequential read - 4.03 Gb/s and random read -3.33 Gb/s

Network Benchmarking

- Network Benchmark evaluation is performed for both TCP and UDP using fixed scaling.
- It includes the TCP protocol stack, UDP, fixed packet/buffer size (64KB), and varying the concurrency (1 thread, 2 threads, 4 threads, 8 threads);
- Size of data is 1GB.
- **The throughput and latency for each of them is shown in the below graph:**



- Here we can see that throughput is increasing as the threads are increasing for TCP connection. But, for UDP, throughput is decreasing. As UDP being Connectionless protocol is slower as compared to TCP. There might be lost of packets in case of UDP. Big Data size is also a reason for decreasing throughput.



- Here we can see that latency is decreasing as the threads are increasing for TCP connection. But, for UDP its increasing. Latency gives us the idea about the continuous sending of data, so it should be decreasing. UDP is a connectionless protocol, hence its taking time to send the data.

- **Throughput and Latency for network:**

Network Throughput	1 Thread	2 Thread	4 Thread	8 Thread
TCP	45.99979	98.34523	95.98722	136.7811
UDP	57.02899	62.63849	38.31422	39.26602

Network Latency	1 Thread	2 Thread	4 Thread	8 Thread
TCP	0.14247	0.066639	0.068276	0.047913
UDP	0.114258	0.104626	0.171049	0.166903

- **Mean Throughput and Latency for three experiments:**

Mean Value	Thread 1	Thread 2	Thread 4	Thread 8
TCP	43.48793	99.47256	94.94335	134.8511
UDP	57.2036	73.71618	36.71421	43.46394

Mean Value	Thread 1	Thread 2	Thread 4	Thread 8
TCP	0.095307	0.049078	0.06776	0.035685
UDP	0.157589	0.101295	0.191048	0.267643

- **Standard Deviation of Throughput and Latency for three experiments:**

SD	Thread 1	Thread 2	Thread 4	Thread 8
TCP	4.481194	4.165044	5.60816	3.342858
UDP	6.271823	13.59424	3.23209	3.661663

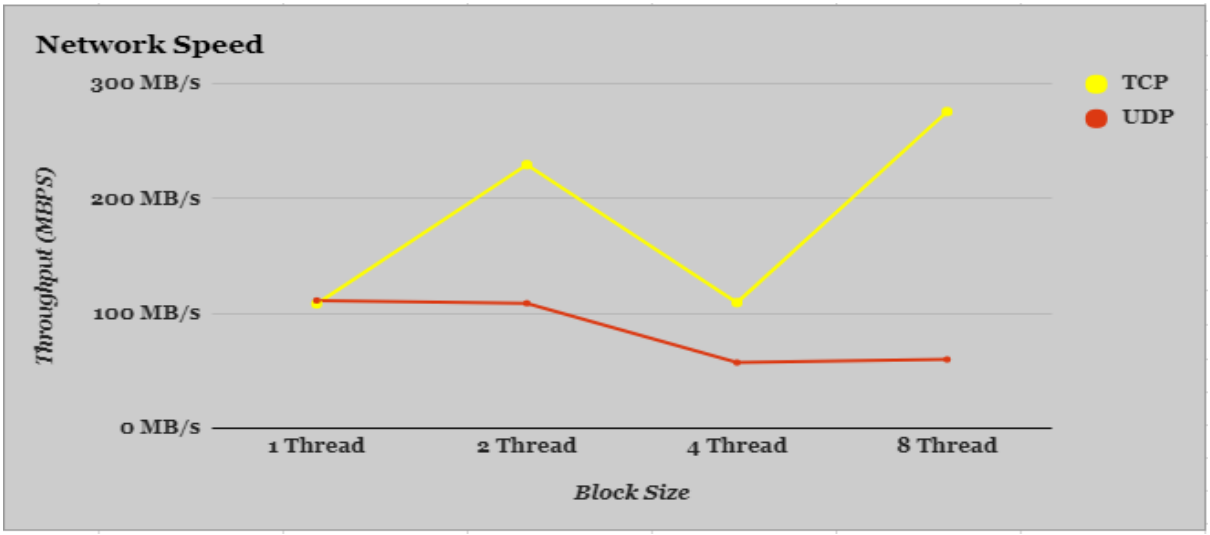
SD	Thread 1	Thread 2	Thread 4	Thread 8
TCP	0.051063	0.035739	0.006243	0.014802
UDP	0.075052	0.005768	0.034639	0.174487

- **IPERF Benchmarking for TCP and UDP:**

```
------------------------------------------------------------
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
```

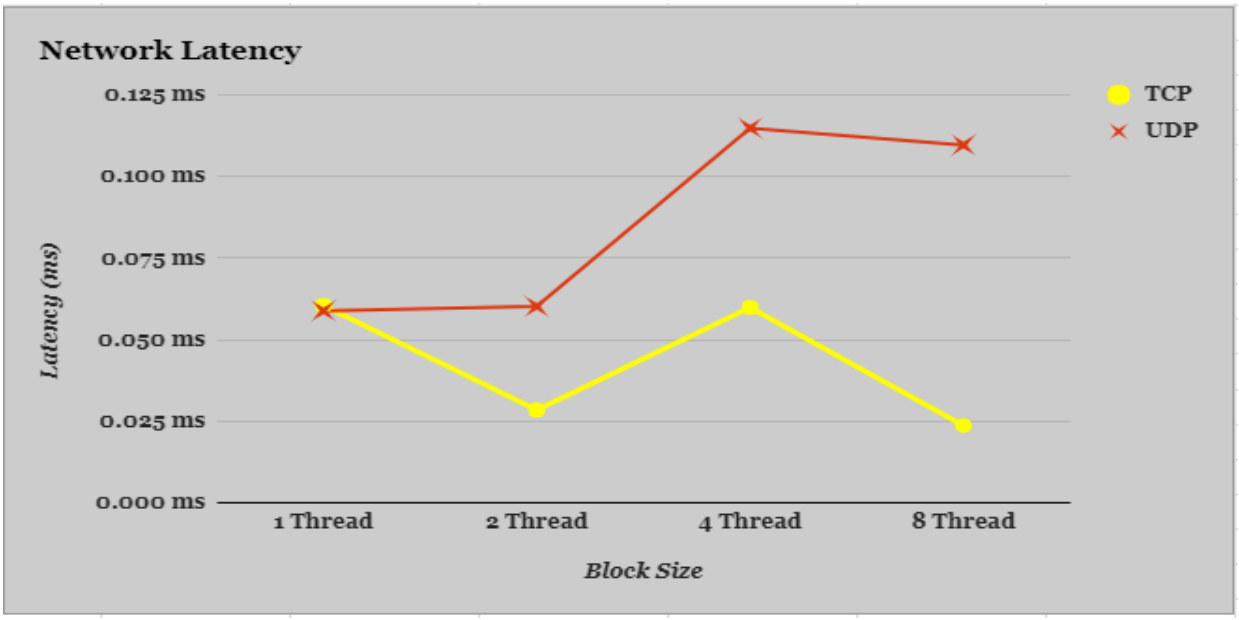
```
------------------------------------------------------------
Client connecting to 52.26.161.78, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 172.31.25.167 port 50114 connected with 52.26.161.78 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%)
```

- **Network Benchmarking on two different nodes:**



Network Throughput	1 Thread	2 Thread	4 Thread	8 Thread
TCP	108.2749328	229.7124246	109.3656836	276.0400275
UDP	111.2610902	108.8259378	57.06838789	59.76927625

- We can see that throughput is increasing as the threads are increasing for TCP connection. But, for UDP, throughput is decreasing.



Network Latency	1 Thread	2 Thread	4 Thread	8 Thread
TCP	0.06052739844	0.02852958438	0.05992373281	0.02374148438
UDP	0.05890289219	0.0602209375	0.1148376578	0.1096483078

- Here we can see that latency is decreasing as the threads are increasing for TCP connection. But, for UDP its increasing. Latency gives us the idea about the continuous sending of data, so it should be decreasing.