CS553 Cloud Computing

Assignment -1

CWID – A20380536

Performance Document

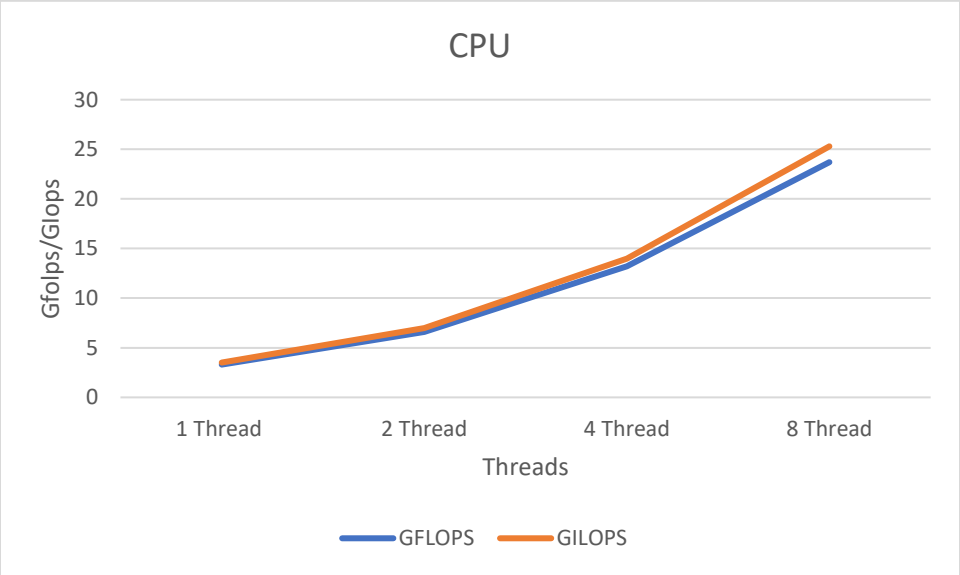
CPU Benchmark:

This program ran for two sets, single precision and double precision.

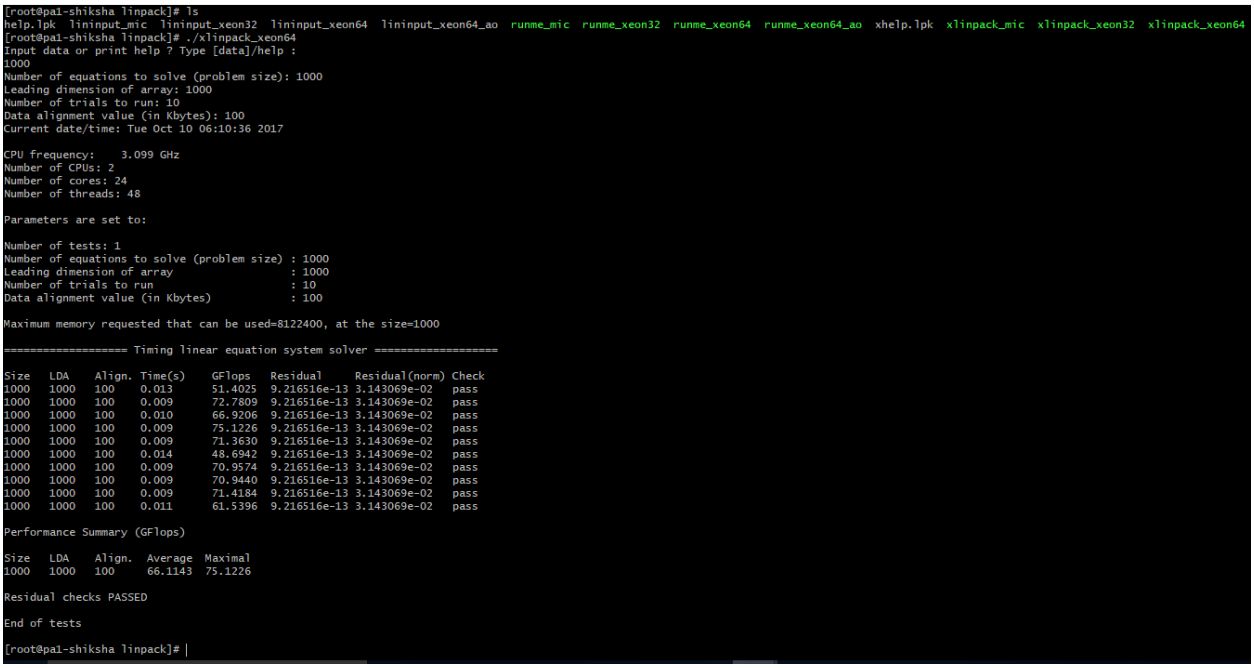
System Configuration:

Chameleon Cloud (bare metal UCI Instance)
Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz *2(dual socket)
Family: Haswell
16 DP CPI 32 SP CPI
OS -Centos 7
RAM: 128 GB
No. of cores = 12 (24 thread) *2 (two CPU)

No_of_threads	GFLOPS	GILOPS
1 Thread	3.305785	3.508772
2 Thread	6.61157	6.986899
4 Thread	13.223141	13.973799
8 Thread	23.703703	25.296444



Linpack benchmark:



- For above experiment we have taken 1e9 number of calculations for 1 operations and one incremental operation of loop (so 2 instructions).
- We have observed that there is increase in number of FLOP as we increase the number of threads. Which interim increases the number of CPU core usage. That helps in using all the core. To verify this, we ran TOP to see the CPU usage for the same.
- We also tried to pin out our processor, but it was observed that we were getting less performance.
- Theoretical peak performance of the test bed:
  = (speed * Instruction per cycle * no of core) /sec
  2.30 * 24 (cores)* 16(dual Precision) = 883.2 GFLOPS

What efficiency do you achieve compared to the theoretical performance?

For GFLOPS: (23.704/883.2) *100= 2.68%
For GIOPS: (25.3/883.2) *100 = 2.86%

Memory Benchmark:

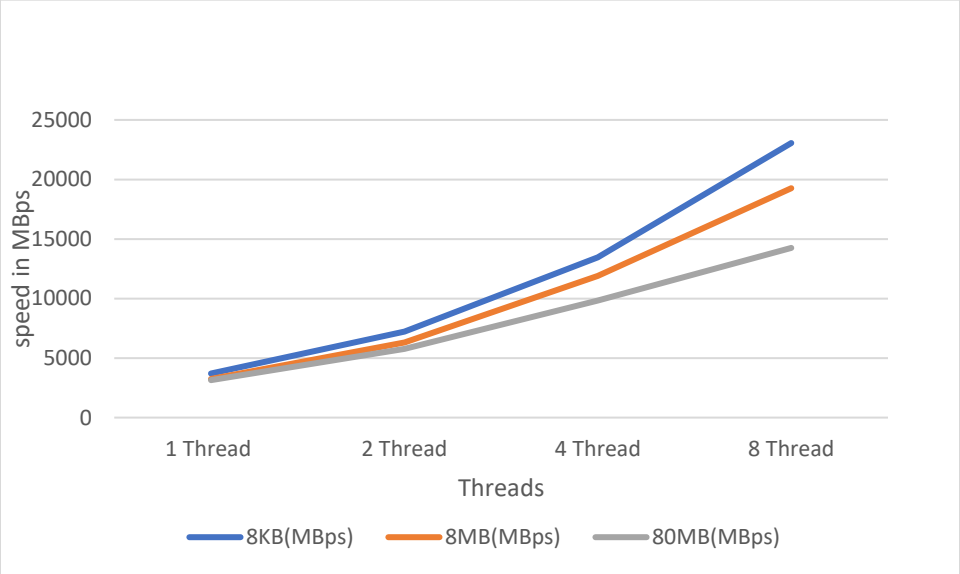
This program ran for 4 Data blocks 8B,8KB,8MB and 80MB for multiple thread categories (1 thread ,2 thread 4 threads and 8 threads) on 1.28GB data.
This Experiment is further categories for different functions:

- Sequential Read and Write
- Sequential Read
- Random Read

System Configuration:
Chameleon Cloud (bare metal UCI Instance)
Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz *2(dual socket)
Family: Haswell
16 DP CPI 32 SP CPI
OS -Centos 7
RAM: 128 GB
No. of cores = 12 (24 thread) *2 (two CPU)

Sequential Read and Write (Throughput):

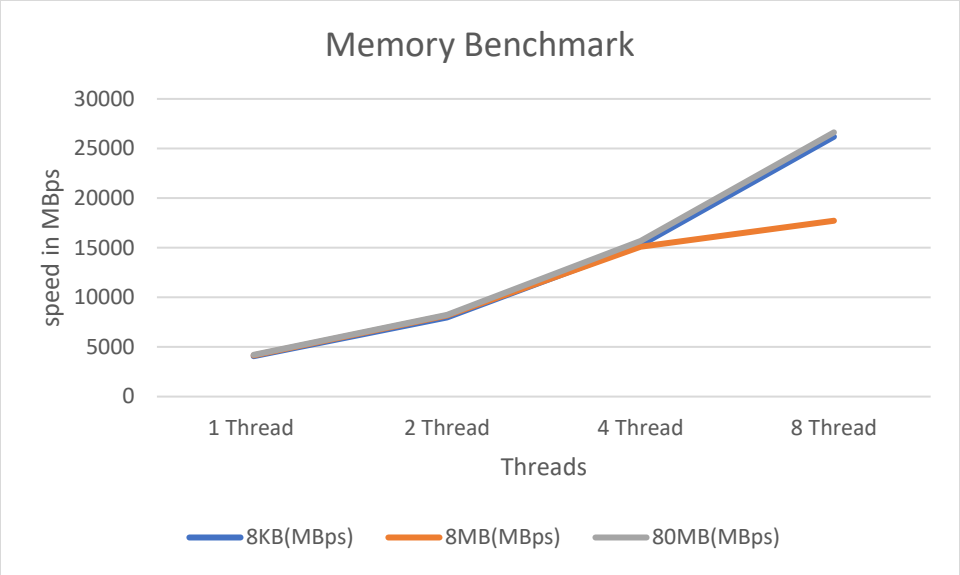
Block_size->	8KB(MBps)	8MB(MBps)	80MB(MBps)
1 Thread	3702.877935	3245.106264	3148.245949
2 Thread	7227.839543	6302.738163	5762.933226
4 Thread	13452.16973	11893.75758	9838.117881
8 Thread	23059.17361	19263.15974	14251.03976



Observation:
There is steady increase in throughput as we increase the number of threads from 1 to 8

Sequential Write (Throughput):

Block_size->	8KB(MBps)	8MB(MBps)	80MB(MBps)
1 Thread	4072.25696	4153.475838	4208.34083
2 Thread	7990.140194	8182.786745	8231.165015
4 Thread	15260.68288	15096.36012	15681.58102
8 Thread	26188.87527	17719.92293	26626.36309

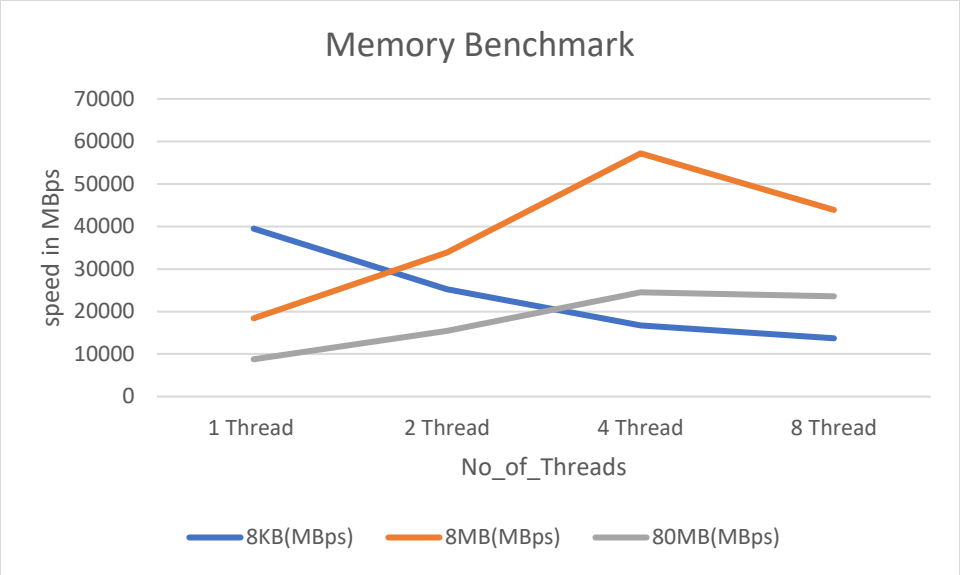


Observation:

- There is increase in throughput as we increase the block-size from 8 KB to 8MB and 80MB.
- Also, as we increase the number of thread, throughput is increasing. But after certain number of threads throughput was not increasing further.

Random Write (Throughput):

Block_size->	8KB(MBps)	8MB(MBps)	80MB(MBps)
1 Thread	39490.5326	18405.66747	8775.226685
2 Thread	25229.16721	33918.96635	15461.41505
4 Thread	16708.87291	57186.84127	24519.24117
8 Thread	13711.16689	43911.35195	23548.68466

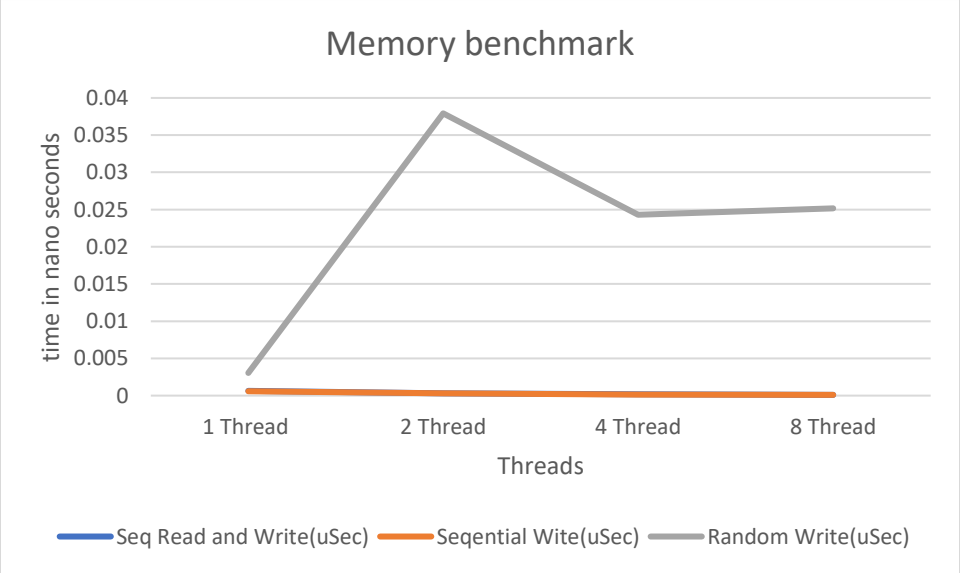


Observation:

- There is increase in throughput as we increase the block-size from 8 KB to 8MB and 80MB.
- Also, as we increase the number of thread throughput is increasing.

Latency for 8B data:

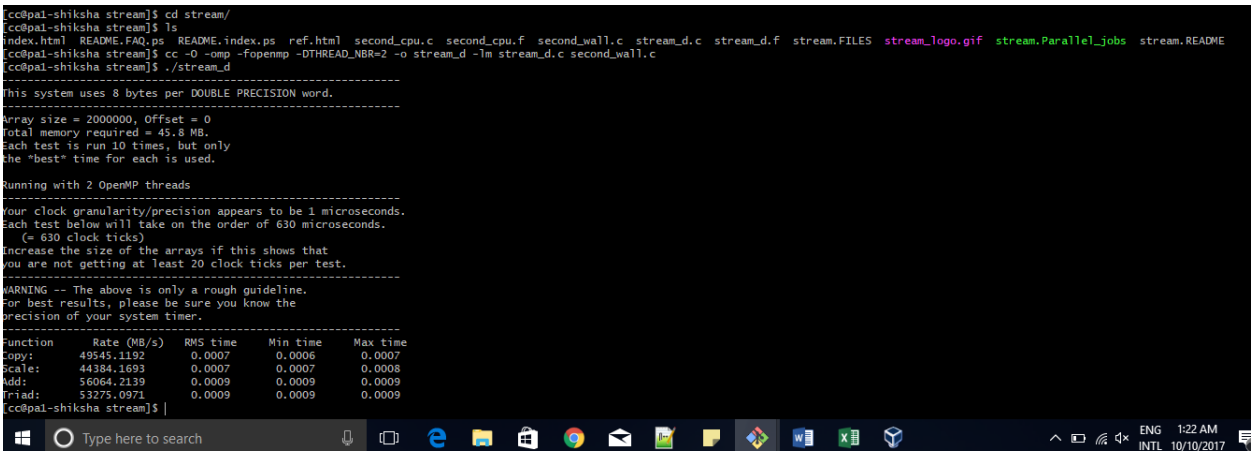
Function->	Seq Read and Write(uSec)	Seqential Wite(uSec)	Random Write(uSec)
1 Thread	0.000636	0.000586	0.003047
2 Thread	0.000319	0.000295	0.037926
4 Thread	0.000189	0.00015	0.024299
8 Thread	0.000095	0.000101	0.025148



Observation:

- Latency was reducing as we increased the number of threads.
- Random function was taking more time and hence it was having high latency.

Stream benchmark:



Theoretical peak = (Clock speed* Memory interface)
=2133*10^6*64/8=940.653 GB/s

what efficiency do you achieve compared to the theoretical performance?
For latency: (0.000586/0.0007) = 83.7%

Disk Benchmark:

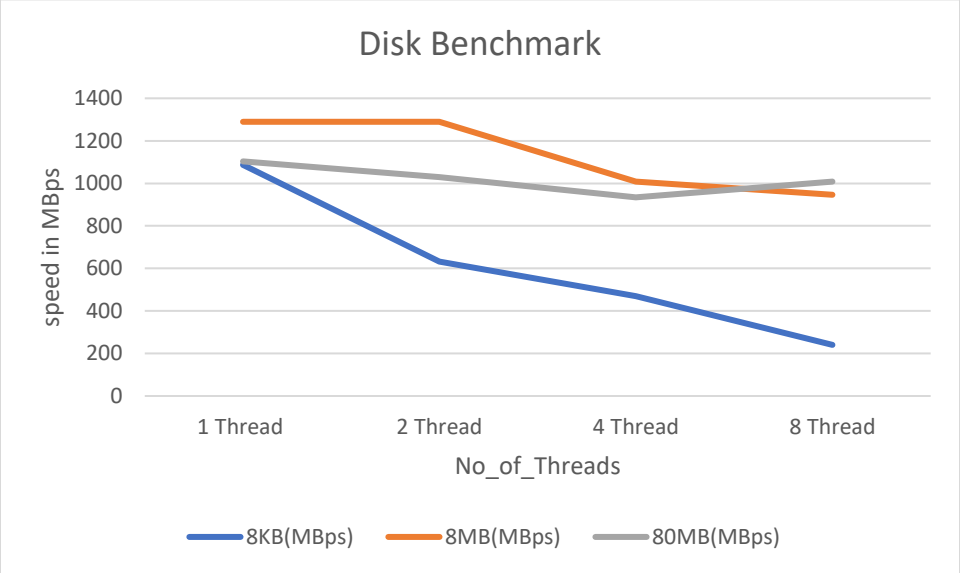
This program ran for 4 Data blocks 8B,8KB,8MB and 80MB for multiple thread categories (1 thread ,2 thread 4 threads and 8 threads) on 5GB file.
This Experiment is further categories for different functions:

- Sequential Read and Write (from File1 to File2);
- Sequential Read (from File1)
- Random Read (from File1)

System Configuration:
Chameleon Cloud (bare metal UCI Instance)
Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz *2(dual socket)
Family: Haswell
16 DP CPI 32 SP CPI
OS -Centos 7
RAM: 128 GB
No. of cores = 12 (24 thread) *2 (two CPU)

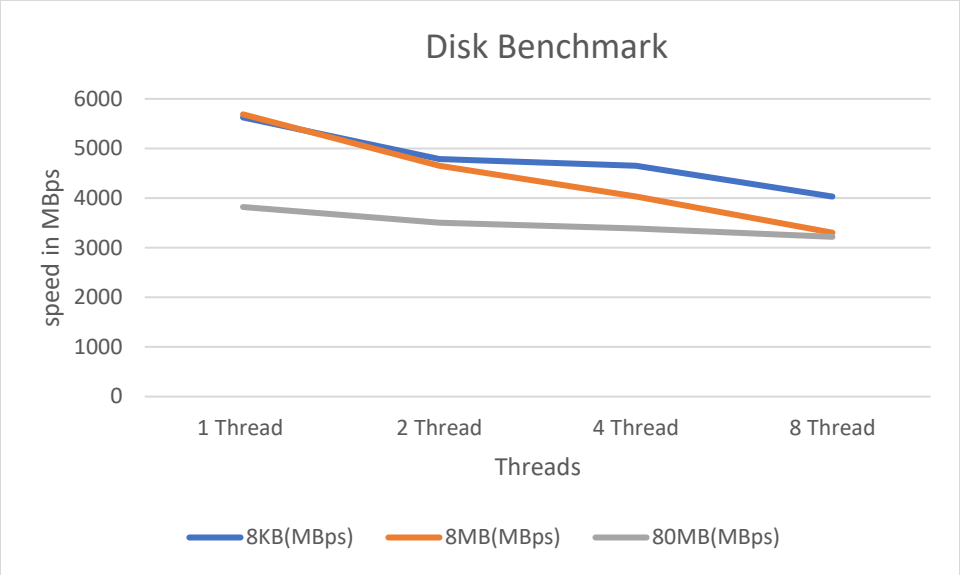
Sequential Read and Write (Throughput):

Block_size->	8KB(MBps)	8MB(MBps)	80MB(MBps)
1 Thread	1087.048832	1289.672544	1103.448276
2 Thread	632.098765	1289.672544	1030.181087
4 Thread	469.724771	1007.874016	934.306569
8 Thread	240.488492	946.395564	1007.874016



Sequential Read (Throughput):

Block_size->	8KB(MBps)	8MB(MBps)	80MB(MBps)
1 Thread	5626.373626	5688.888889	3820.895522
2 Thread	4785.046729	4654.545455	3506.849315
4 Thread	4654.545455	4031.496063	3390.728477
8 Thread	4031.496063	3303.225806	3220.125786

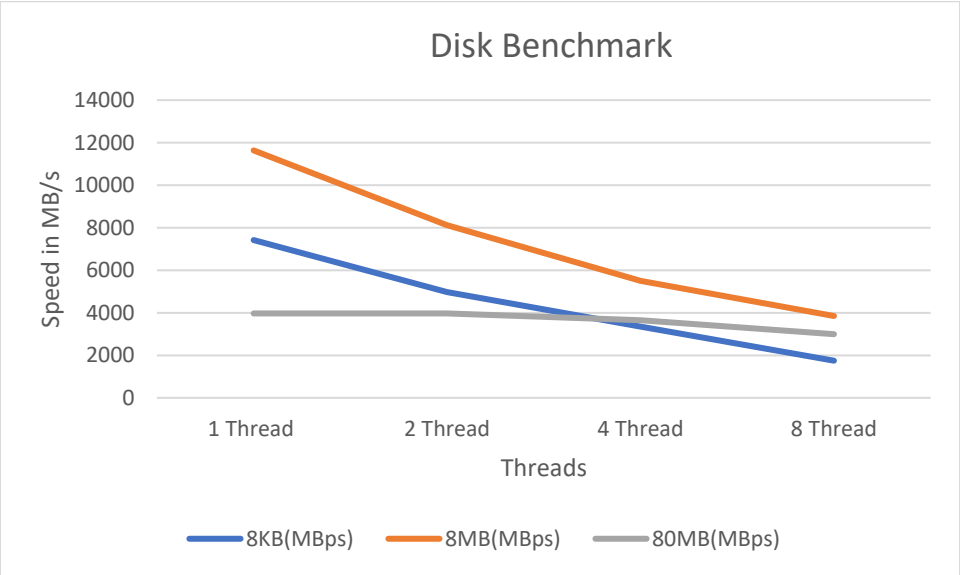


Observation:

After no of thread (4), there was decrement in rate of speed for reading.

Random Read (Throughput):

Block_size->	8KB(MBps)	8MB(MBps)	80MB(MBps)
1 Thread	7420.289855	11636.36364	3968.992248
2 Thread	4970.873786	8126.984127	3968.992248
4 Thread	3346.405229	5505.376344	3657.142857
8 Thread	1747.440273	3849.62406	2994.152047

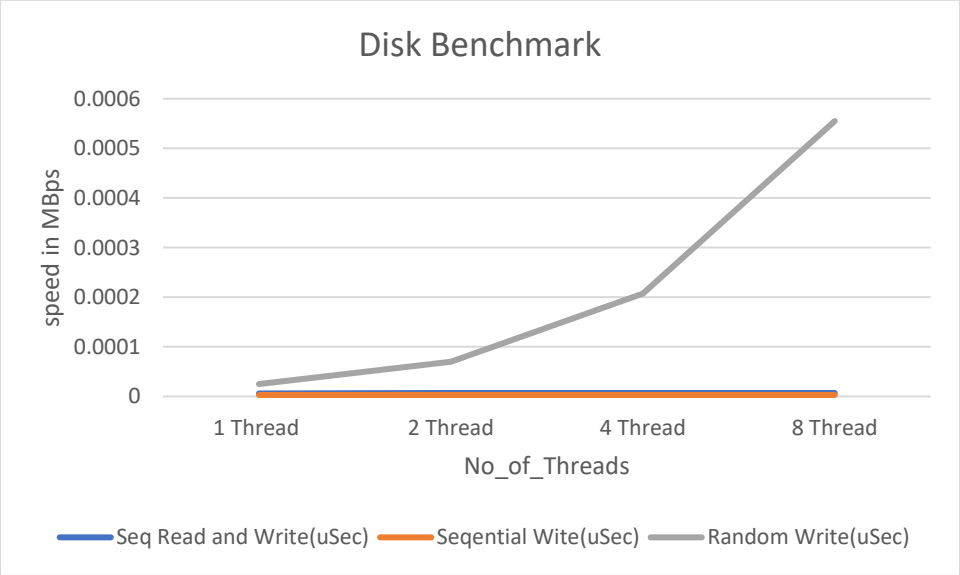


Observation:

This is more of like sequential read, but it is observed that random read is little slower than the sequential read when compared.

Latency for 8B data:

Function->	Seq Read and Write(uSec)	Seqential Wite(uSec)	Random Write(uSec)
1 Thread	0.000006	0.000003	0.000025
2 Thread	0.000007	0.000003	0.00007
4 Thread	0.000007	0.000003	0.000207
8 Thread	0.000007	0.000003	0.000555



IOZone benchmark:

```
     ce->myaiocb.aio_nbytes);
     ^
libasync.c:1573:5: warning: format '%d' expects argument of type 'int', but argument 5 has type 'size_t' [-Wformat=]

Building fileop for Linux

cc -Wall -c -O3  fileop.c -o fileop_linux.o
fileop.c: In function 'validate':
fileop.c:1377:15: warning: variable 'size1' set but not used [-Wunused-but-set-variable]
  register int size1;
               ^

Building the pit_server

cc -c   pit_server.c  -o pit_server.o
cc  -O3  iozone_linux.o libasync.o libbif.o -lpthread \
        -lrt -o iozone
cc  -O3 -Dlinux fileop_linux.o -o fileop
cc  -O3 -Dlinux pit_server.o -o pit_server
[cc@pa1-shiksha current]$ ./iozone -g# -s 81920
        Iozone: Performance Test of File I/O
                Version $Revision: 3.394 $
                Compiled for 64 bit mode.
                Build: linux

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

        Run began: Tue Oct 10 06:26:39 2017

        Using maximum file size of 4 kilobytes.
        File size set to 81920 KB
        Command line used: ./iozone -g# -s 81920
        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
           81920       4 1942514 2675287  6202252  6259196 5262408 2624206 5601357  6127700  5609770  2831858  2799442 6125297  6181275

iozone test complete.
[cc@pa1-shiksha current]$
```

what efficiency do you achieve compared to the theoretical performance?

fread= 6125297

throughput= 6125297/1024= 5981.73 MBps

My experiment for 80MB, throughput= 3820.90 MBps

Efficiency= (3820.90/5981.73) * 100= 63.87%

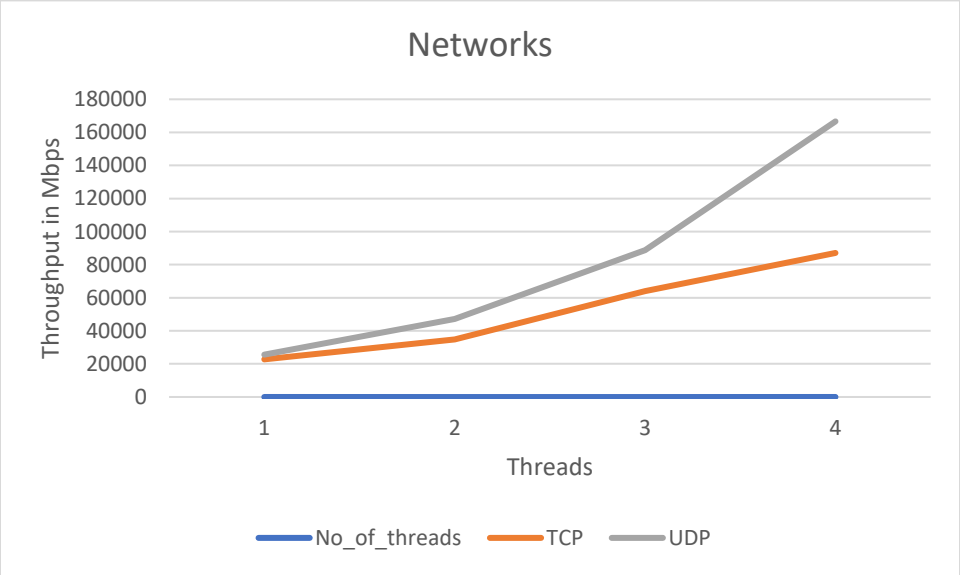
Network:

In this part, we are calculating Network speed between two nodes for different number of threads i.e. 1, 2, 4, 8.

There are two operations i.e. communications through TCP and UDP.

TCP Benchmark:

TCP	Block_size(64KB)	Data size(4GB)
No_of_threads	TCP	UDP
1	22755.55465	25600.00057
2	34859.57456	47148.20192
4	63937.56246	88862.37145
8	87091.03018	166624.7431

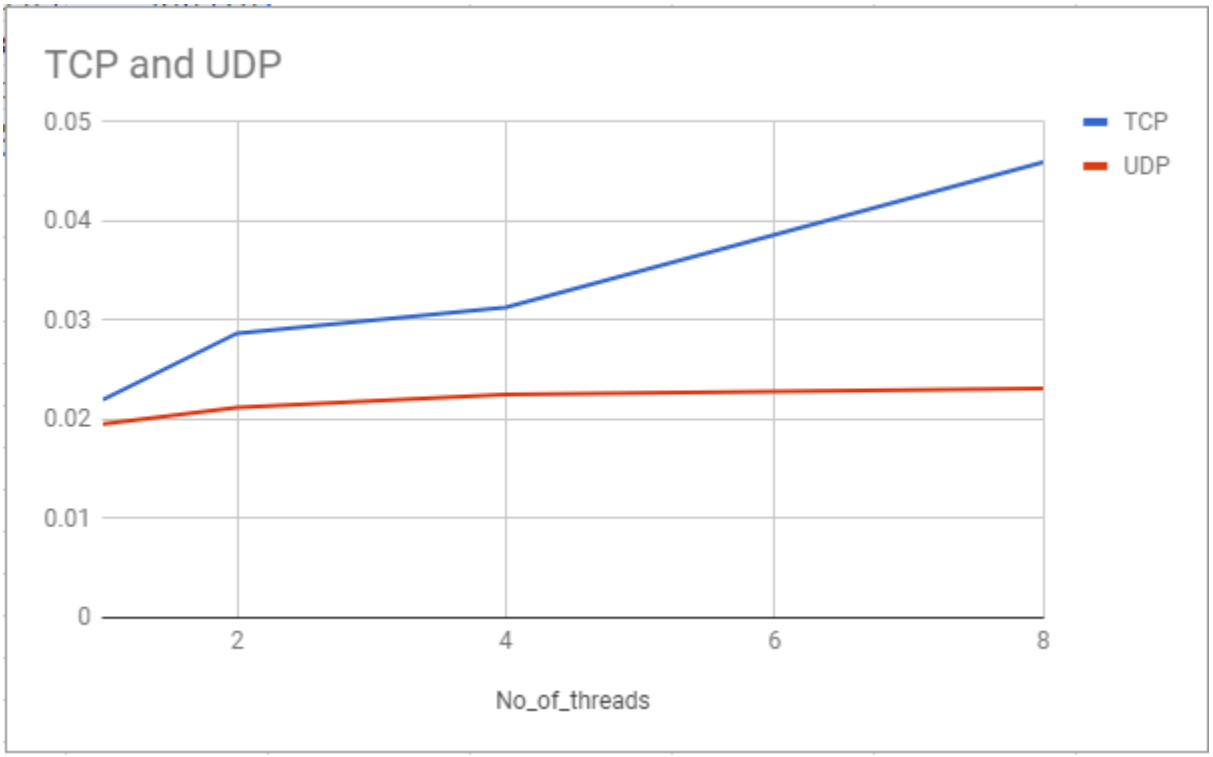


Observation:

As we increase the number of threads, the throughput is increasing for the transfer of fixed amount of data.

UDP Benchmark:

UDP	Block_size(64KB)	Data size(4GB)
No_of_threads	TCP	UDP
1	0.021973	0.019531
2	0.028687	0.02121
4	0.031281	0.022507
8	0.045929	0.023101



Observation:

- Latency is increasing for TCP as the number of threads increasing.
- Latency is increasing in the starting and after that it became static.

IPerf benchmark:

```
- - - - - - - - - - - - - - - - - - - - - - - - -
Test Complete. Summary Results:
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  34.6 GBytes  29.7 Gbits/sec    0             sender
[  4]   0.00-10.00  sec  34.6 GBytes  29.7 Gbits/sec                  receiver
[  6]   0.00-10.00  sec  34.6 GBytes  29.7 Gbits/sec    0             sender
[  6]   0.00-10.00  sec  34.6 GBytes  29.7 Gbits/sec                  receiver
[SUM]   0.00-10.00  sec  69.1 GBytes  59.4 Gbits/sec    0             sender
[SUM]   0.00-10.00  sec  69.1 GBytes  59.4 Gbits/sec                  receiver
CPU Utilization: local/sender 99.5% (0.9%u/98.7%s), remote/receiver 70.3% (1.2%u/69.2%s)
snd_tcp_congestion cubic
rcv_tcp_congestion cubic

iperf Done.
[cc@pa1-shiksha current]$
```