Cloud Computing Programming Assignment 1

Performance Evaluation

This document contains the experimental results for the following benchmark

For CPU benchmark, we measure the performance by calculating processor speed for performing instructions per cycle
For Memory benchmark, we measure the throughputs (sequential read, sequential write, random read and random write) and latency (random read, random write) for various block size (1B, 1KB) and number of threads (1, 2 and 4).
For Disk benchmark, we measured the throughputs (sequential read, sequential write, random read and random write) and latency (random read, random write). Varied the block size (1B/1KB/1MB) and the number of threads (1, 2 and 4).

1. System Domain

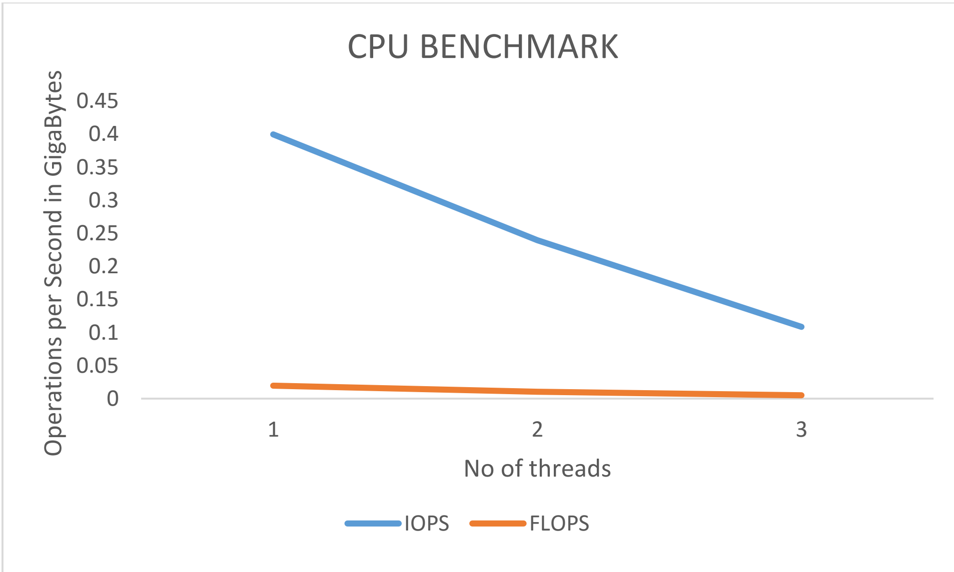
For all benchmark, the experiments were performed on Linux- Ubuntu, Intel i3 processor DDR3 with 2 cores, RAM of 2GB, and DISK 20GB.

2. Program Output Graphs and Results

This part shows the experiment result for each benchmark and give explanations for the trends in the results.

A CPU benchmark Results

CPU benchmark result is to find maximum GIOPS and GFLOPS perfomed by processor for different number of threads.
Below graph shows the experiment results for 1, 2 and 4 threads for GFlops and GIops.
Gflops obtained are between 0.0005 to 0.01 GFLOPS and GIOPS between 0.1 to 0.4 as we vary the no of threads



Thus we derive that as we increase the thread size the No of instructions per second decreases

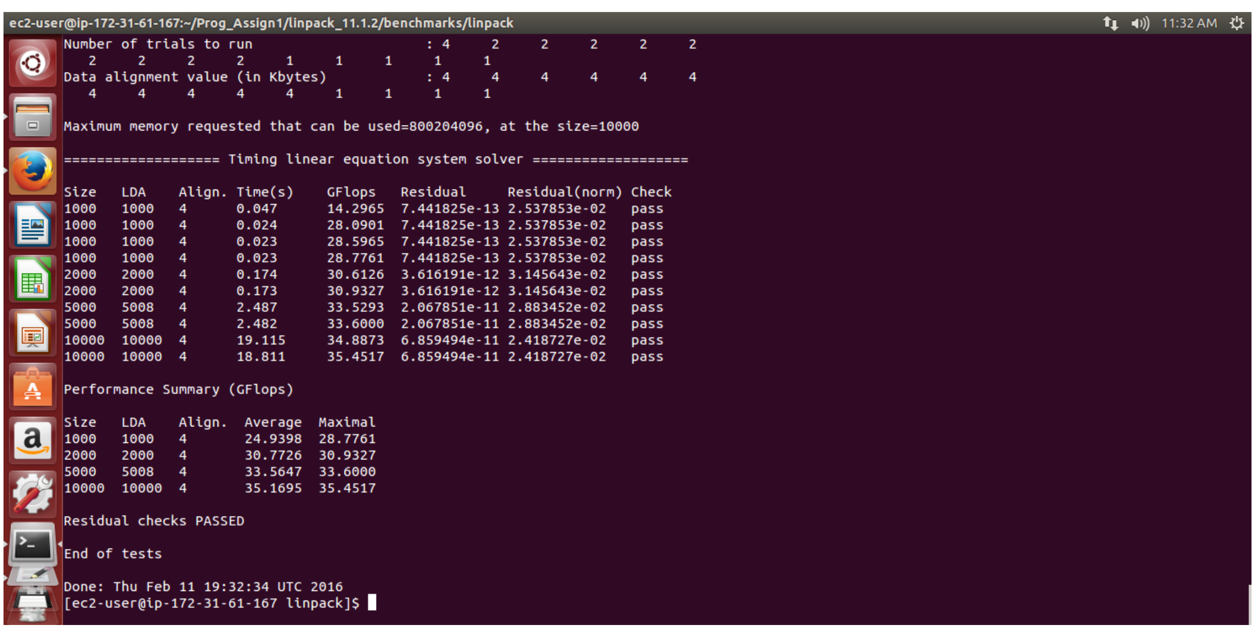
Theoretical peak performance of the processor is given by:
= No of Cores *instruction per cycle * clock rate in Ghz
=1*4*2.5=10 Giga Flops per second

Maximum Practical Performance we got from our output= 0.4

Efficiency= (0.4/10)*100= 4%

4% efficiency achieved as compared to theoretical performance.

Linpack Benchmark Results

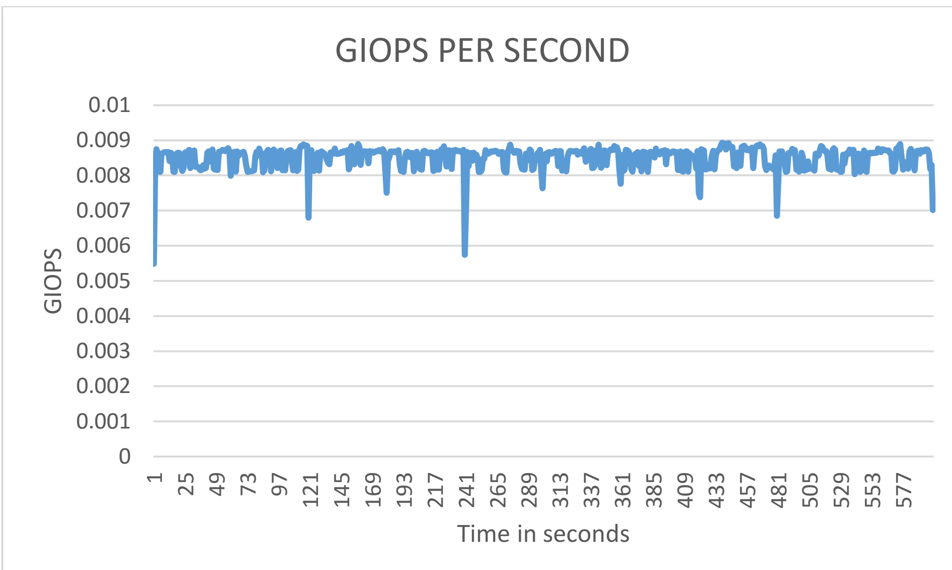


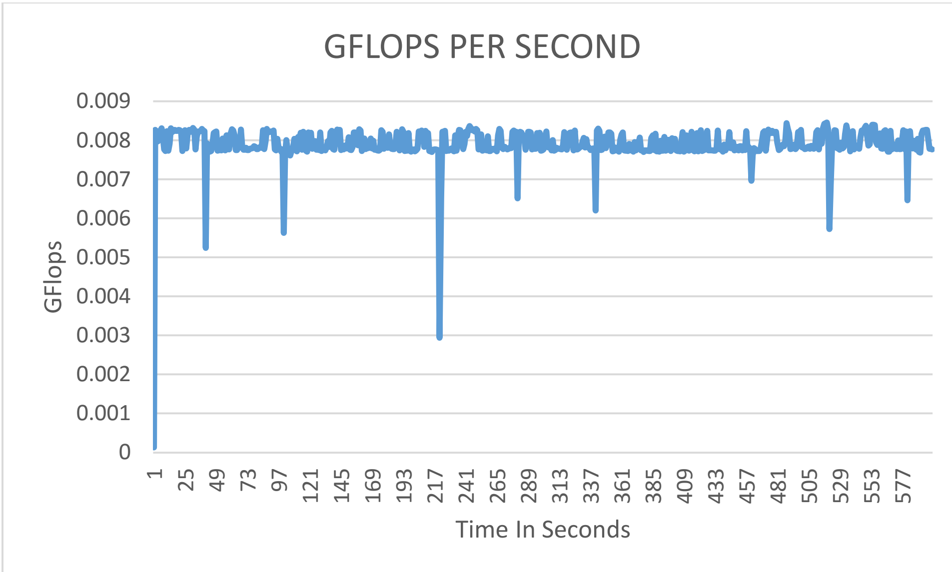
```
ec2-user@ip-172-31-61-167:~/Prog_Assign1/linpack_11.1.2/benchmarks/linpack                11:32 AM
Number of trials to run                    : 4     2     2     2     2     2
   2     2     2     2     1     1     1     1     1
Data alignment value (in Kbytes)           : 4     4     4     4     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      0.047      14.2965  7.441825e-13 2.537853e-02   pass
1000   1000   4      0.024      28.0901  7.441825e-13 2.537853e-02   pass
1000   1000   4      0.023      28.5965  7.441825e-13 2.537853e-02   pass
1000   1000   4      0.023      28.7761  7.441825e-13 2.537853e-02   pass
2000   2000   4      0.174      30.6126  3.616191e-12 3.145643e-02   pass
2000   2000   4      0.173      30.9327  3.616191e-12 3.145643e-02   pass
5000   5008   4      2.487      33.5293  2.067851e-11 2.883452e-02   pass
5000   5008   4      2.482      33.6000  2.067851e-11 2.883452e-02   pass
10000  10000  4      19.115     34.8873  6.859494e-11 2.418727e-02   pass
10000  10000  4      18.811     35.4517  6.859494e-11 2.418727e-02   pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
1000   1000   4       24.9398  28.7761
2000   2000   4       30.7726  30.9327
5000   5008   4       33.5647  33.6000
10000  10000  4       35.1695  35.4517

Residual checks PASSED

End of tests

Done: Thu Feb 11 19:32:34 UTC 2016
[ec2-user@ip-172-31-61-167 linpack]$
```

From code output practical performance we got 0.4 GFLOPS per Second
Practical Value we get from linpack
= 35.4723/18.8=1.886
Efficiency achieved on comparison to theoretical performance
=1.886/10=18%

2) Sample of Graph for 10 minutes plotting Instructions per second based on single thread

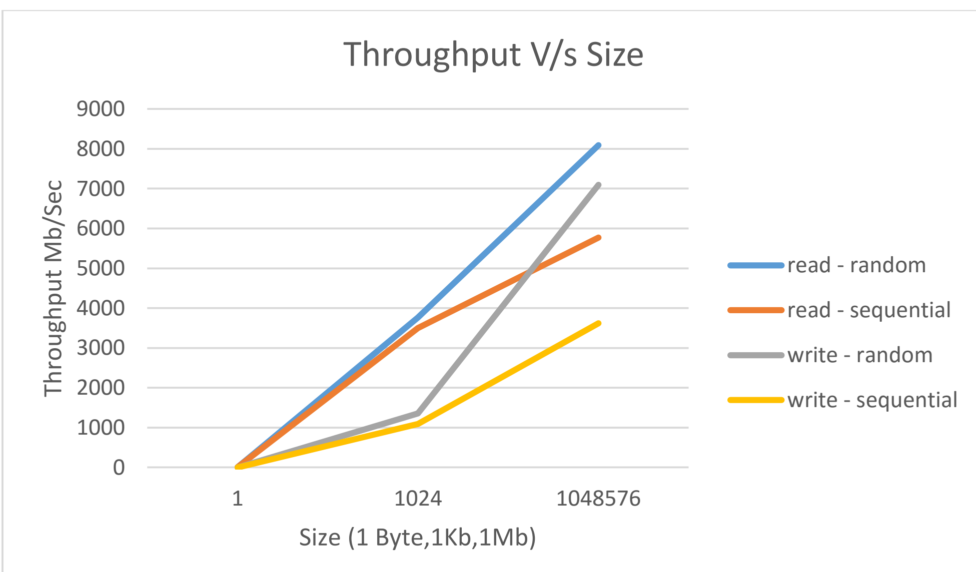
THREAD /TIME





2) Disk benchmark

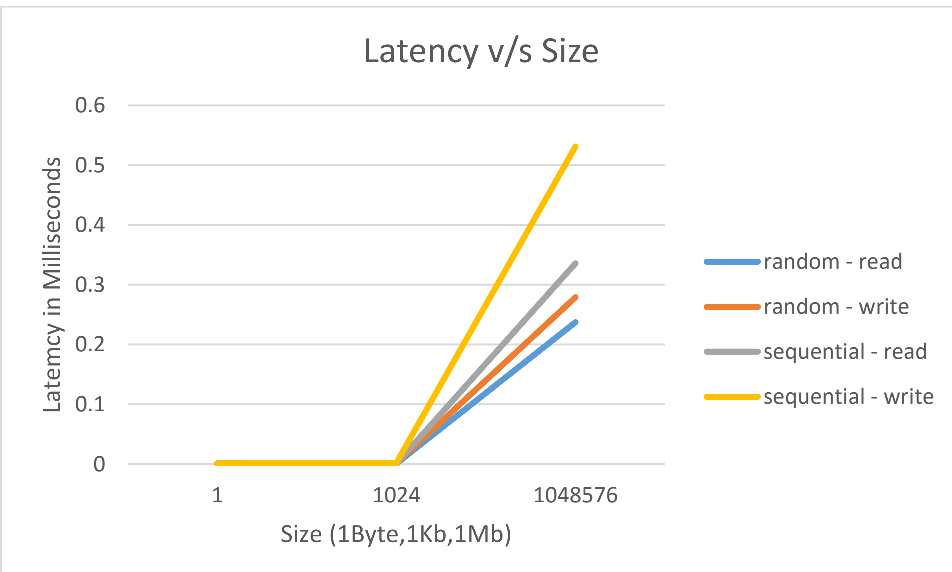
A Throughput V/s size



The graph displays the implementation of sequential and random throughput for disk read and write.
As the size increases throughput increases exponentially for random and sequential read and write of disk
As the bytes increase reading randomly gives more throughput megabytes read per second than random read
Random writes are more efficient than sequential writes for throughput
Read is performed faster than write.
Read throughput is more than write throughput

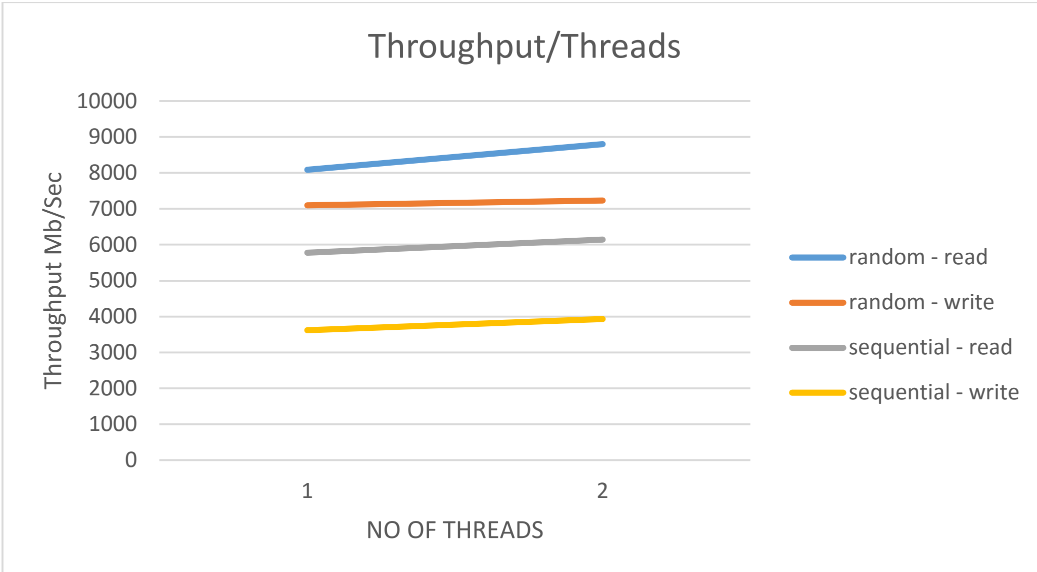
B Latency V/s Size

Graph shows the implementation and comparison of number of threads in sequential read, sequential write, random read and random write with respect to latency in milliseconds.

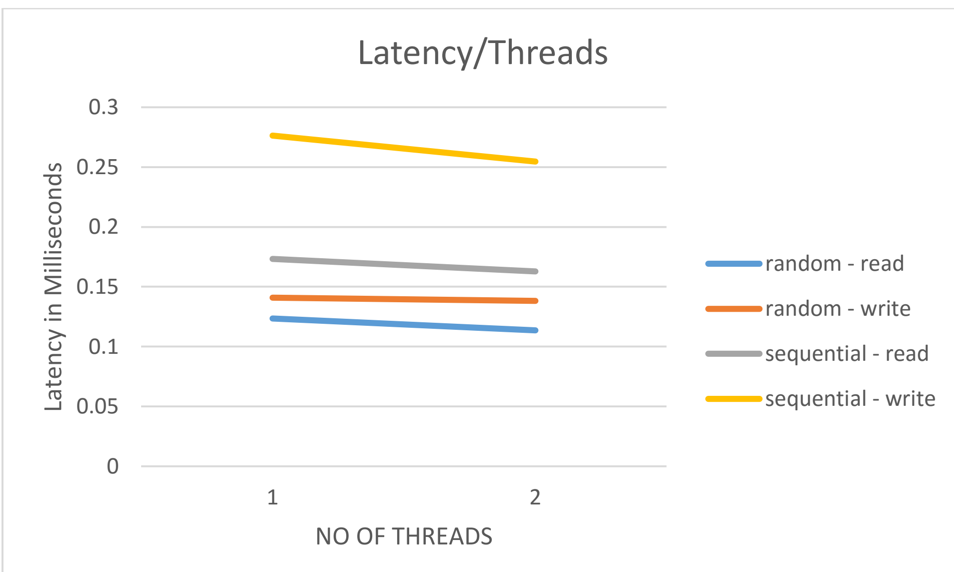


Sequential Latency takes more time than random latency to reach the disk for both reading and writing. Hence, we can say that sequential access is less efficient.
Write takes more time than read to reach disk.
Therefore random access is more efficient with respect to latency and throughput.

Throughput on increasing no of threads for 1 MB

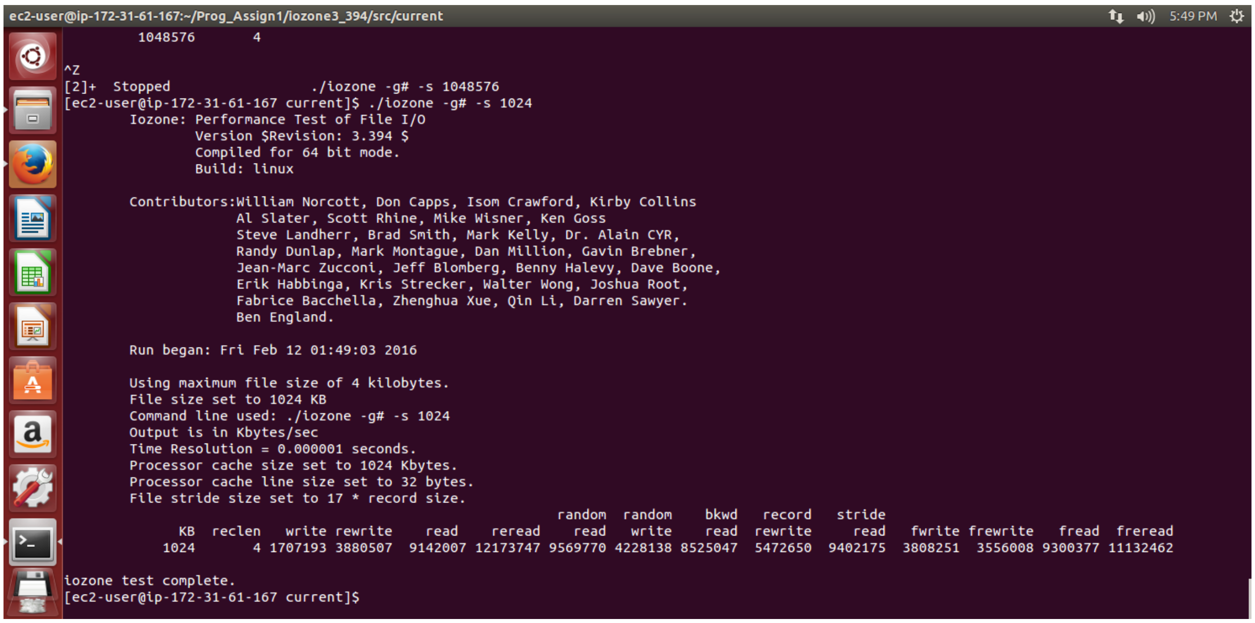


Latency on increasing no of threads for 1 MB



C IOZONE Benchmark

Below screen shot shows the execution of Iozone benchmark for 1024 kb (1 MB)



Theoretical Performance of disk by Iozone:

Read Throughput 9300377 kb/sec =9082.39 MB/sec
Write Throughput 3808251/1024 =3718.99 MB/Sec

Practical output maximum throughput achieved read sequential= 7095.97
Practical output maximum throughput achieved write sequential=3619.9

Efficiency achieved for read (%) = 7095.97 *100/9082.39= 78.12 %
Efficiency achieved for write (%) = 3619.9 *100/3718.99= 97.33 %

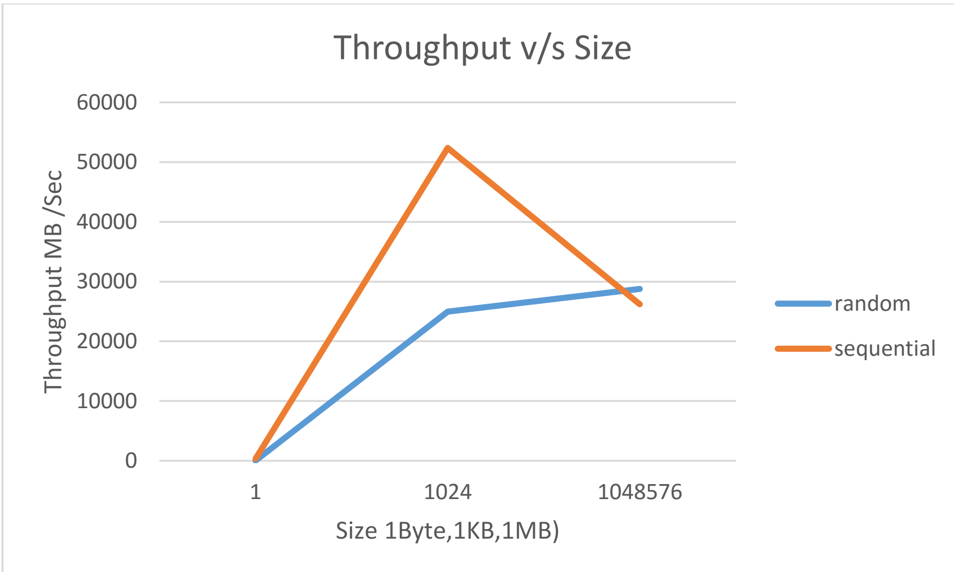
Theoretical Throughput for processor
Transfer rate=6 Gb/sec
Cache Size=16
Revolution per minute =5400
Latency Achieved in program output= 0.53
Efficiency Achieved= 0.53/7.3=7%
Theoretical Throughput for sequential write=80.8 mb/sec

D Program Output Values

size	threads	Access	throughput	latency	operation
1	1	sequential	4.465004	0.000214	read
1	2	sequential	4.360963	0.000219	read
1	1	random	3.210147	0.000297	read
1	2	random	3.067479	0.000311	read
1024	1	sequential	3495.892	0.000279	read
1024	2	sequential	3478.261	0.000281	read
1024	1	random	3766.478	0.000259	read
1024	2	random	3612.39	0.00027	read
1048576	1	sequential	5774.505	0.173175	read
1048576	2	sequential	6142.742	0.162794	read
1048576	1	random	8088.979	0.123625	read
1048576	2	random	8802.333	0.113606	read
1	1	sequential	1.594541	0.000598	write
1	2	sequential	1.557635	0.000612	write
1	1	random	1.319972	0.000722	write
1	2	random	1.304455	0.000731	write
1024	1	sequential	1092.344	0.000894	write
1024	2	sequential	1115.084	0.000876	write
1024	1	random	1353.592	0.000721	write
1024	2	random	1347.232	0.000725	write
1048576	1	sequential	3619.909	0.27625	write
1048576	2	sequential	3926.573	0.254675	write
1048576	1	random	7095.973	0.140925	write
1048576	2	random	7234.254	0.138231	write

3) Memory benchmark results

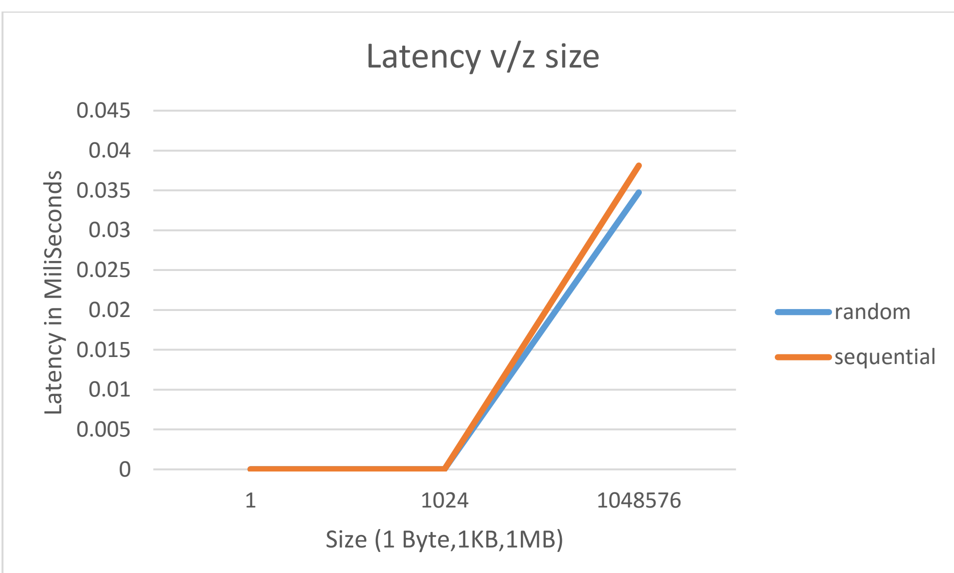
A Throughput V/s Size



As we increase the memory size Sequential memory access gives more throughput. Random access is less efficient for copying bytes of data than sequential as we increase the memory size

B Latency V/s Size

As we increase the size of memory latency for accessing the memory increases exponentially.

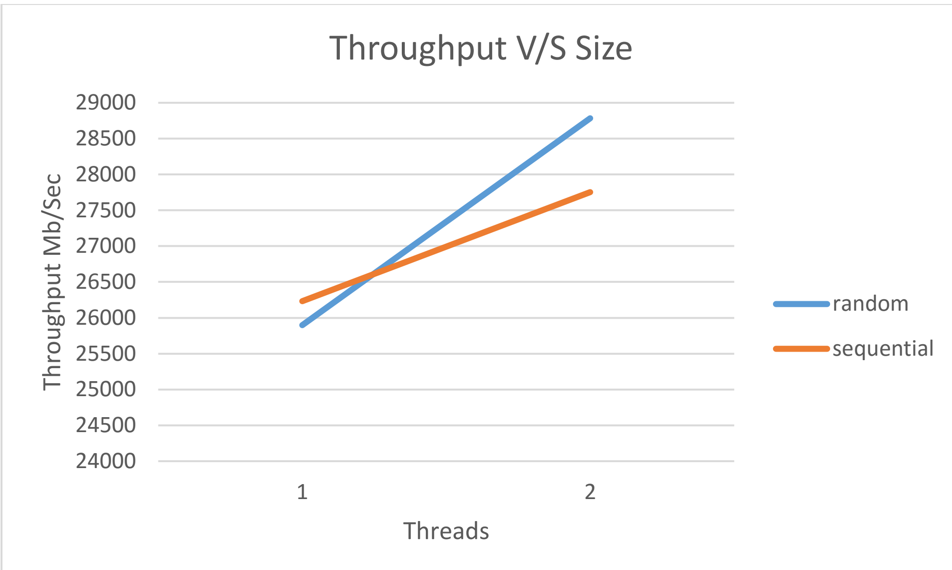


s

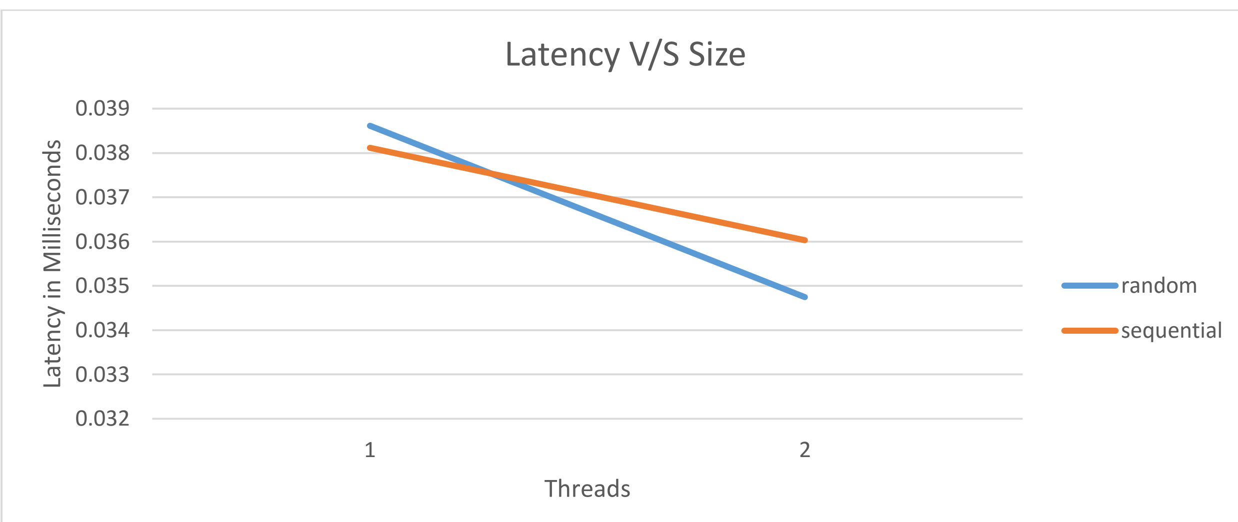
Latency increases exponentially for both random and sequential access as we increase the memory bytes to copy

The graph shows the comparison between sequential and random access throughput and latency for multiple threads.
As we increase threads throughput for accessing the memory and latency increases

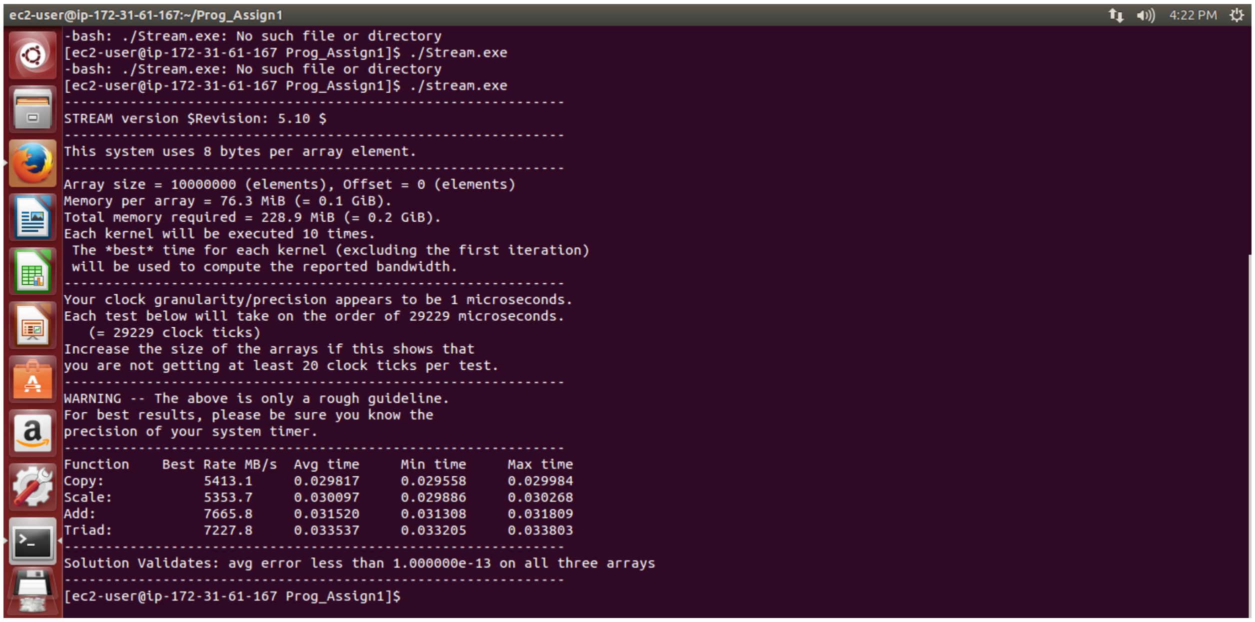
Throughput on increasing no of threads for 1 MB



Latency on increasing no of threads for 1 MB



C Stream Benchmark



```
ec2-user@ip-172-31-61-167:~/Prog_Assign1                                    4:22 PM
-bash: ./Stream.exe: No such file or directory
[ec2-user@ip-172-31-61-167 Prog_Assign1]$ ./Stream.exe
-bash: ./Stream.exe: No such file or directory
[ec2-user@ip-172-31-61-167 Prog_Assign1]$ ./stream.exe
-------------------------------------------------------------
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 29229 microseconds.
   (= 29229 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:           5413.1     0.029817     0.029558     0.029984
Scale:          5353.7     0.030097     0.029886     0.030268
Add:            7665.8     0.031520     0.031308     0.031809
Triad:          7227.8     0.033537     0.033205     0.033803
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
[ec2-user@ip-172-31-61-167 Prog_Assign1]$
```

Average Latency for Stream=0.029
Average Latency Achieved from Program=0.034
Efficiency achieved= (0.034/0.029)*100

Clocks =33
Theoretical memory throughput = 8 *clock_rate=8*33=264 mb/sec

D Program Output Values

size	threads	access	throughput	latency
1	1	sequential	330.6147	0.000003
1	2	sequential	332.9622	0.000003
1	1	random	24.4916	0.000039
1	2	random	20.4465	0.000047
1024	1	sequential	52390.31	0.000019
1024	2	sequential	56677.29	0.000017
1024	1	random	24949.32	0.000039
1024	2	random	24756.3	0.000039
1048576	1	sequential	26233.81	0.038119
1048576	2	sequential	27753.69	0.036031
1048576	1	Random	25898.35	0.038612
1048576	2	Random	28782.15	0.034744