Snehal Sonawane (A20360111)

PRPGRAMMING ASSIGNMENT 1

Benchmarking

1. Introduction:

The assignment aims at Benchmarking different components of a computer system and thus, learn about the performance.

1. CPU Benchmarking
2. Memory Benchmarking
3. Disk Benchmarking

1.1 CPU Benchmark:

1. CPU Benchmarking is done in C Language. It is also menu driven.
2. Goal of this module is to calculate processor speed in terms of Floating Point Operations Per Second (FLOPS) and Integer Operations Per Second (IOPS). The program runs multiple instructions concurrently.
3. Two functions namely flopsCal and iopsCal calculate the GFLOPS and GIOPS respectively. Time taken to perform operations is calculated using clock() function. The time difference thus obtained is divided by CLOCKS_PER_CYCLE giving time in seconds. Number of (ITERATIONS * No of threads * No of operations) are divided by calculated time to determine processor speed in terms of Giga FLOPS and Giga IOPS. Functions threadFunctionFlopCal and threadFunctionIopsCal are used to create and join threads. For the purpose of multi-threading pthread libraries are used. The loop iterating variables are volatile to prevent the compiler optimizing the loops. It is difficult to calculate CPU speed in single iteration, hence taken 1000000000.
4. It also includes the experiment to calculate the GFLOPS/GIOPS over a period of 10 mins for 4 threads. The functions flops and iops calculate the instructions per second and store them in CPU_LOG.txt and CPU_IOPS.txt respectively. Thread functions like threadFunctionFlop and threadFunctionIops create, join and kill threads.
5. Improvements and extension to the program:
   Improve the overclocking ie. The processor to run faster than expected by giving more work for the CPU. Also, carry out experiments on increasing thread concurrency to give 100% utilization.

1.2 Memory Benchmark:

1. Memory benchmarking program is developed in C language. It is also menu driven.
2. The two operations performed on memory are read and write. Also, read and write operations are performed sequentially and randomly. A random offset is used to perform read and write operations.
3. Memory is allocated using the malloc function.

4. Three block sizes (1 Byte, 1 KB, 1 MB) and concurrency (1 thread/2 thread) is considered.
5. Throughput in terms of MB/sec and latency in terms of milliseconds are calculated for 3 different block sizes of 1 Byte, 1 Kilobyte and 1 Megabyte at varying level of concurrency for 1 thread and 2 threads. Threading is implemented using p_thread() and p_join() functions of C language.
6. clock() function is used to determine latency. Difference of start time and end time is divided by CLOCKS_PER_CYCLE to calculate latency in seconds.
7. Read and Write is performed using C function memcpy().
8. Pointer functions *block_Byte(), *block_Kbyte() and *block_Mbyte() are implemented to perform sequential data transfer of 1 Byte, 1 Kilobyte and 1 Megabyte respectively. Also, *block_Byte_random(), *blcok_KByte_random() and *block_MByte_random() functions are implemented to perform random data transfer of 1 Byte, 1 KB and 1 MB respectively.
9. Improvements and extension to the program:
   To test the read and write operations using functions other than memcpy().
   Improve the code to isolate the effect of cache.

1.3 Disk Benchmark:

1. Disk Benchmarking program is developed in C language.
2. The benchmark aims at measuring the disk speed by performing read and write operations.
3. The functions p_thread() and p_join() are used to create and join multiple threads.
4. clock() function is used to determine latency. Difference of start time and end time is divided by CLOCKS_PER_CYCLE to calculate latency in seconds.
5. Throughput in terms of MB/sec and latency in terms of milliseconds are calculated for 3 different block sizes of 1 Byte, 1 Kilobyte and 1 Megabyte at varying level of concurrency for 1, 2 and 4 threads respectively.
6. Read and Write is performed on data of block size 1 Byte, 1 KB and 1MB using C function fseek, fputc, fread() and fwrite() respectively.
7. fseek sets the indicator to a new position either at beginning of the file or at any random position in the file.
8. Improvements and extension to the program:
   Experiment can be carried out by increasing the concurrency and minimizing the latency as far as possible. Lines of code can be optimized.

2. Performance

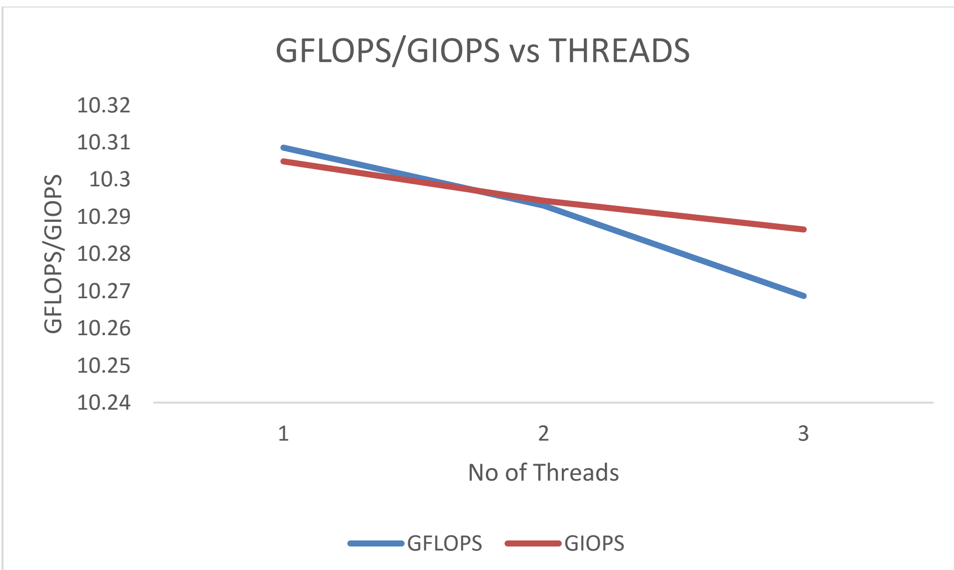
Performance for various modules is measured and plotted on graphs.

2.1 CPU Benchmarking Performance:

1. The floating point operation is in the range of 7-9 GFLOPS for a single thread.
2. The integer point operation is in the range of 9-10 GIOPS for a single thread.
3. Readings

THREADS	GFLOPS	GIOPS
1	10.308567	10.304895
2	10.293027	10.294354
3	10.268674	10.28658

Graph 1: GFLOPS/GIOPS vs No of threads



Observation 1: The GFLOPS and GIOPS decreases as the number of threads increase.
Observation 2: It is observed that GLOPS and GIOPS are maximum when run on single thread.
Observation 3: Optimal performance is obtained at 1 thread

4. Calculating theoretical peak performance
   a. T2 micro instance Specification
      CPU speed: 2.6 GHz
      Number of Cores: 1
      No. of instruction per cycle: 16 (Assumption)

Theoretical Peak Performance = (CPU Speed * No of cores * Instructions per cycle)
= 41.6 GFLOPS

Efficiency = (FLOPS for 1 thread /Theoretical Peak Performance) *100

= (10.308567/41.6) * 100
= 24.7 %

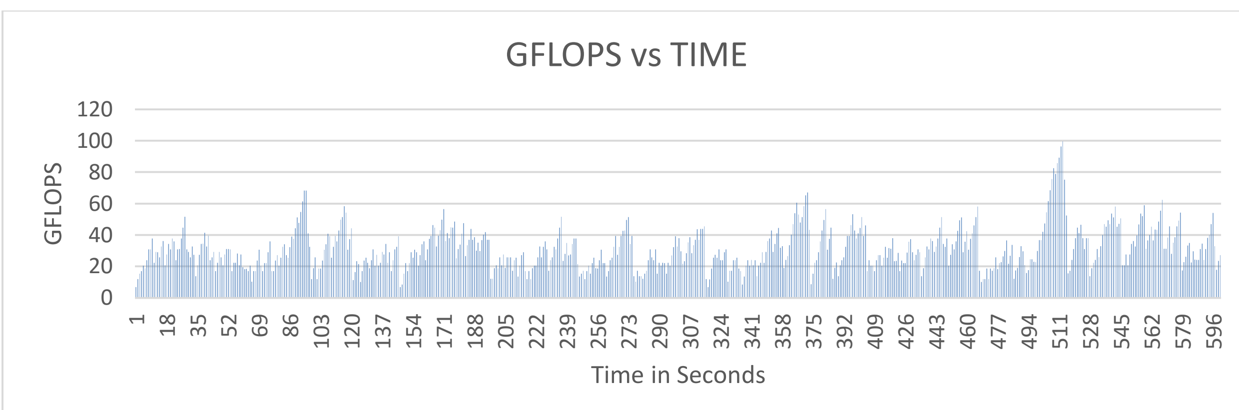
b. Computer Specifications
Processor: 4X Intel(R) core(TM) i3 CPU.
CPU speed: 2.40 GHz
Number of Cores: 2
No. of instruction per cycle: 4
Theoretical Peak Performance = (CPU Speed * No of cores * Instructions per cycle)
= 19.2 GFLOPS

Efficiency = (FLOPS for 1 thread /Theoretical Peak Performance) *100

= (8.9/19.2) * 100
= 46.35 %

5. Plot: GFLOPS over a period of 10 mins with four threads running simultaneously.

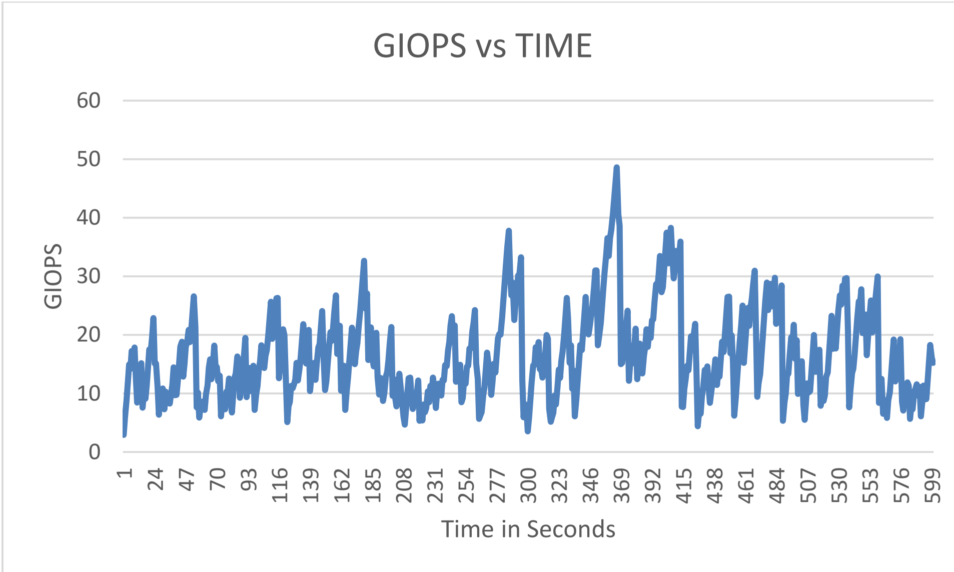
Graph 2: GFLOPS sec vs Time in seconds



Observation 1: The GFLOPS are in the range of 7 to 93
Observation 2: At 511th second the GFLOPS are maximum.

6. Plot: GIOPS over a period of 10 mins with four threads running simultaneously.

Graph 3: Instructions per sec vs Time in seconds

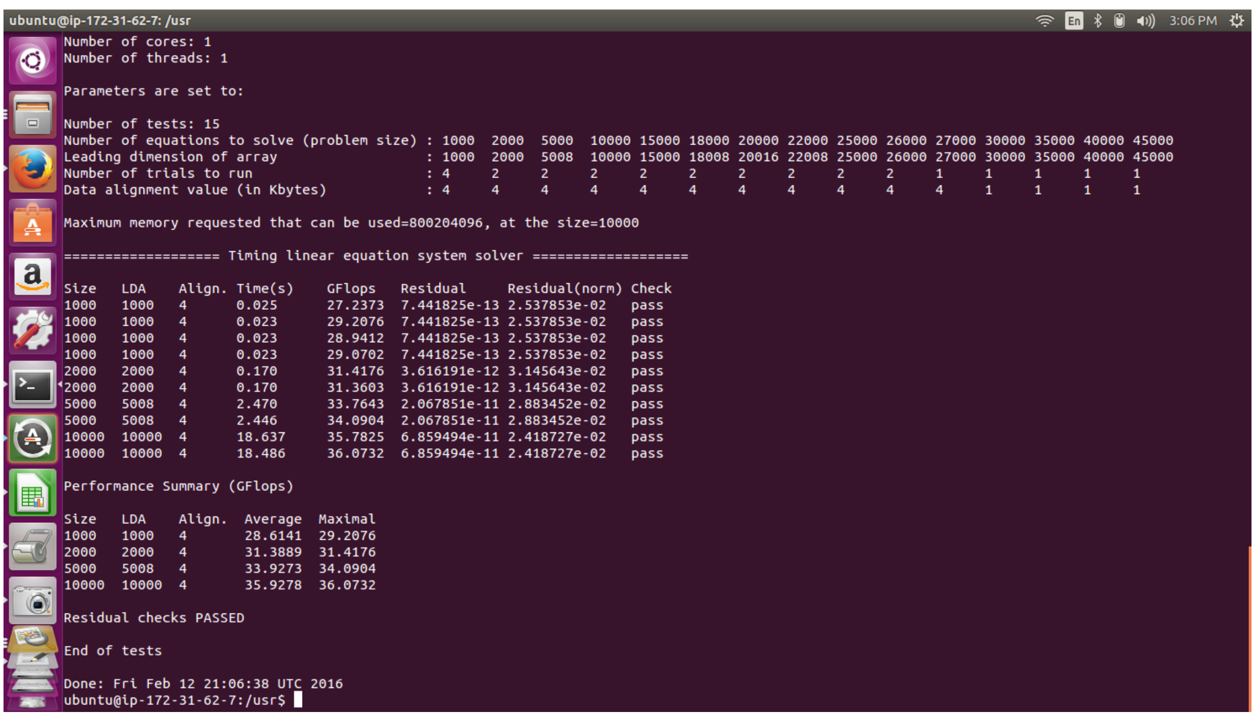


Observation 1: The GIOPS are the range of 2 – 48.

Observation 2: At 369th second the GIOPS are maximum.

Linpack Benchmark:

The following are the observations obtained when LINPACK is run on AMAZON AWS t2 micro instance.



```
ubuntu@ip-172-31-62-7: /usr                                              3:06 PM
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 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     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      0.025      27.2373  7.441825e-13 2.537853e-02   pass
1000   1000   4      0.023      29.2076  7.441825e-13 2.537853e-02   pass
1000   1000   4      0.023      28.9412  7.441825e-13 2.537853e-02   pass
1000   1000   4      0.023      29.0702  7.441825e-13 2.537853e-02   pass
2000   2000   4      0.170      31.4176  3.616191e-12 3.145643e-02   pass
2000   2000   4      0.170      31.3603  3.616191e-12 3.145643e-02   pass
5000   5008   4      2.470      33.7643  2.067851e-11 2.883452e-02   pass
5000   5008   4      2.446      34.0904  2.067851e-11 2.883452e-02   pass
10000  10000  4      18.637     35.7825  6.859494e-11 2.418727e-02   pass
10000  10000  4      18.486     36.0732  6.859494e-11 2.418727e-02   pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
1000   1000   4       28.6141  29.2076
2000   2000   4       31.3889  31.4176
5000   5008   4       33.9273  34.0904
10000  10000  4       35.9278  36.0732

Residual checks PASSED

End of tests

Done: Fri Feb 12 21:06:38 UTC 2016
ubuntu@ip-172-31-62-7:/usr$
```

Average GFLOPS = 31.695

Best performance = 36.0732 GFLOPS

a. Observation 1: Efficiency with respect to theoretical performance of t2 micro instance
Efficiency = (FLOPS for 1 thread /Theoretical Peak Performance) *100
= (31.695/41.6) * 100
= 76.18 %

b. Observation 2: Efficiency of my code with respect to linpack
Efficiency = (FLOPS for 1 thread /Theoretical Peak Performance) *100
= (10.3087/31.695) * 100
= 32.52%

2.2 Memory Benchmarking Performance:

1. Readings for Sequential Read Write operations varying block sizes and varying concurrency.

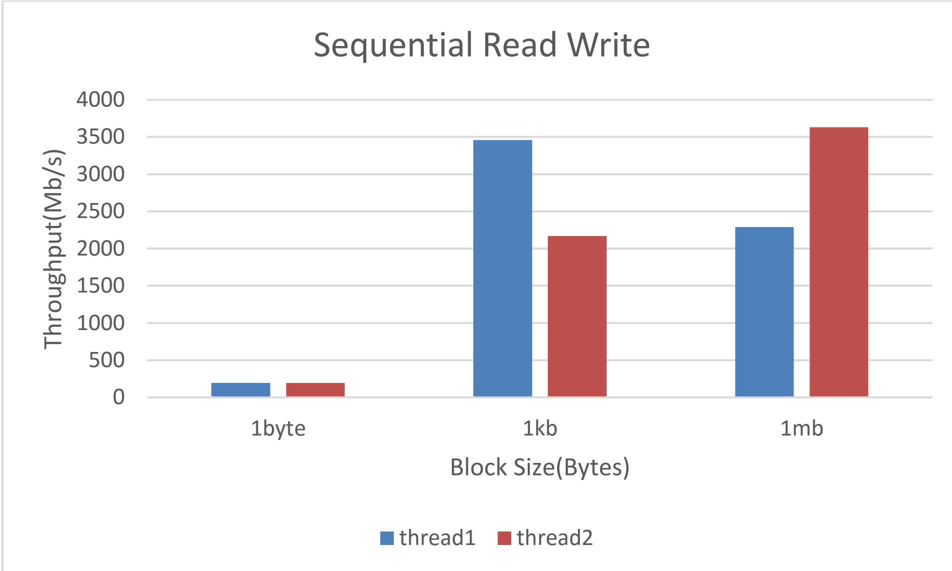
Sequential Read Write	Thread 1	Thread 2
1byte	195.121951	193.89239
1Kb	3457.62712	2170.23585
1 Mb	2288.78276	3630.67

2. Readings for Random Read Write operations varying block sizes and varying concurrency.

Random Read Write	Thread 1	Thread 2
1byte	26.295722	26.053879
1Kb	1610.94177	1354.56368
1 Mb	16447.8887	11616.27

3. Theoretical Calculation
Processor: Intel(R) Core i3-2330M
Clock Speed: 100 MHz
Memory Type: DDR3-1066/1333
Theoretical Maximum Memory Bandwidth is calculated using formula:
Base DRAM clock frequency * lines per clock * memory bus width * number of interfaces = 133000000* 2 * 64 * 2 = 4256 Mb/s

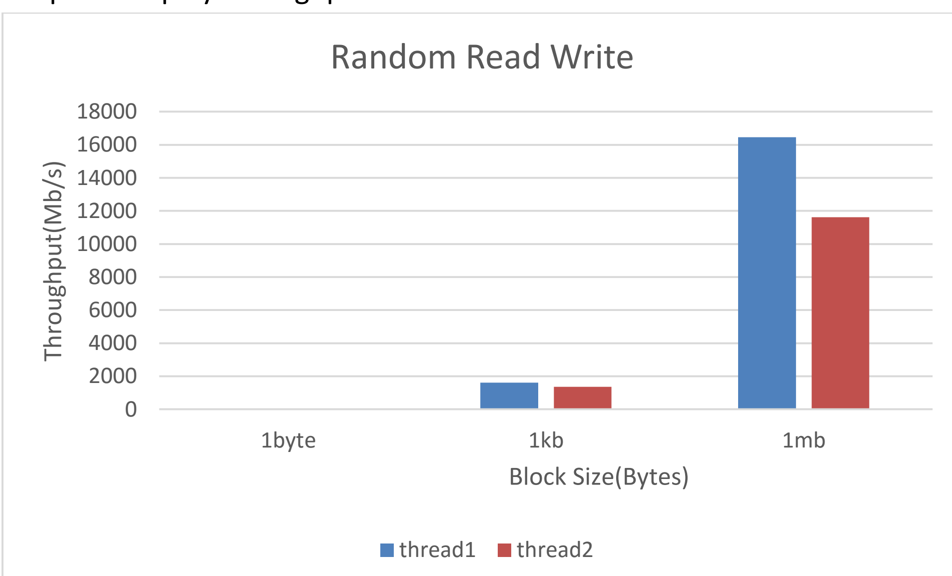
4. Graph 4: Display throughput for Sequential Read & Write



Observation:

It is observed that throughput increases with increase in size of block. Throughput is higher for more number of threads.

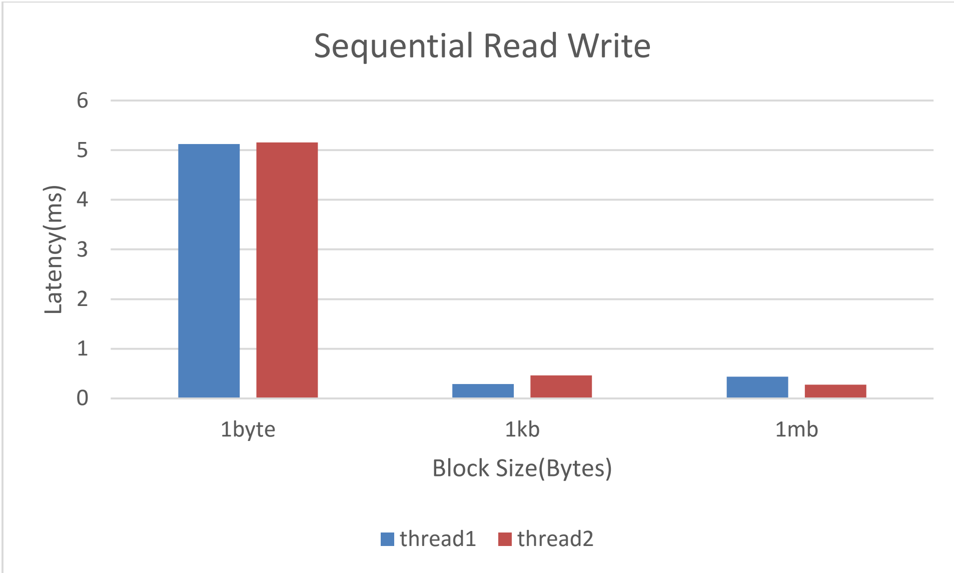
5. Graph 5. Display throughput for Random Read & Write



Observation:

It is observed that throughput increases with increase in size of block. Throughput is higher for single thread.

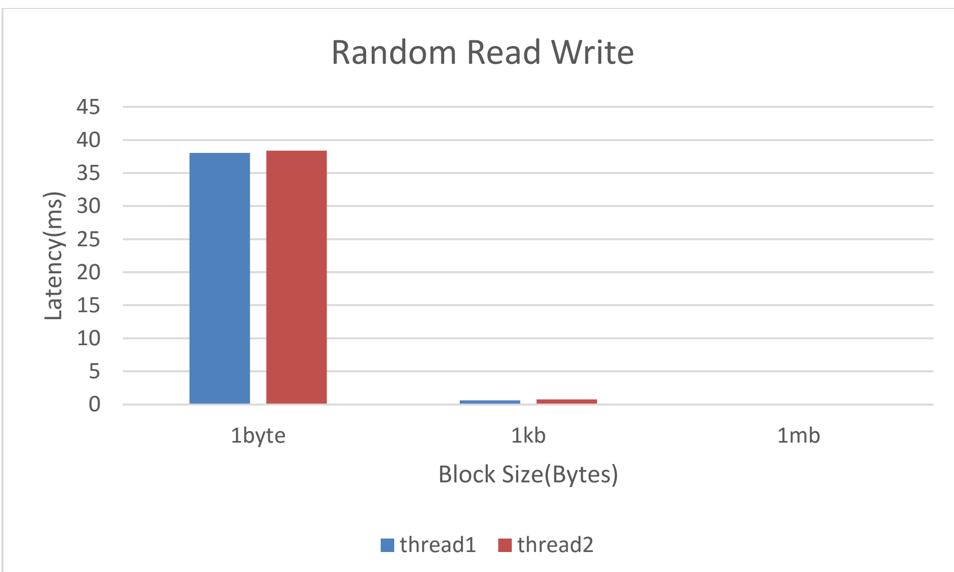
6. Graph 6: Display Latency for Sequential Read & Write



Observation:

It is observed that latency decrease with increase in block size. This is applicable for 1 & 2 threads. Latency is slightly higher for 2 threads.

7. Graph 7: Random Read write latency



Observation:

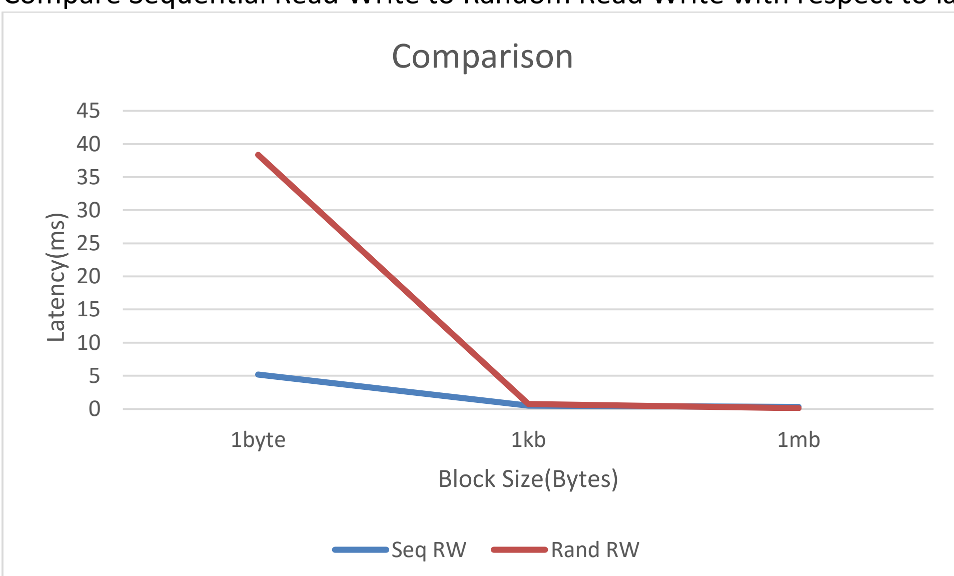
It is observed that latency decrease with increase in block size. This is applicable for 1 & 2 threads. Latency is slightly higher for 2 threads.

8. Graph 8: Compare Sequential Read Write to Random Read Write with respect to throughput

Observation:

It is observed that throughput increases with increase in block size. This is applicable for 1 & 2 threads. Throughput is higher for Rand Read Write.

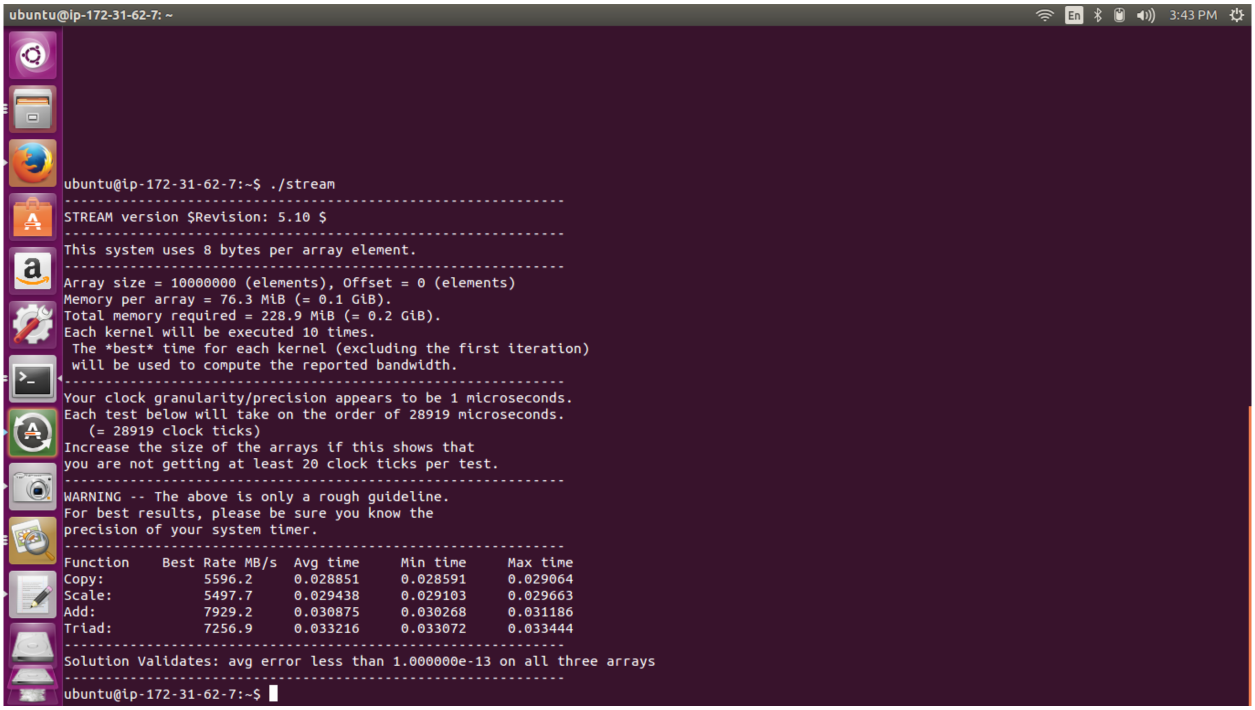
9. Compare Sequential Read Write to Random Read Write with respect to latency



Observation:

It is observed that latency decreases with increase in block size. This is applicable for 1 & 2 threads. Latency is highest to Rand Read Write 1 Byte of data.

Stream Benchmark:



Best performance achieved is 5596.2 MB/s

The theoretical performance calculated for my system is 4256 Mb/s. But the stream benchmark is executed on Amazon t2 micro instance. The specifications must be higher than the ones achieved on my system. Stream has achieved 75% or more efficiency.

2.3 Disk Benchmarking Performance:

1. Readings

Random write	Thread 1	Thread 2
1byte	1.264756	1.24779
1Kb	821.686068	834.43091
1 Mb	3471.01701	3595.82884

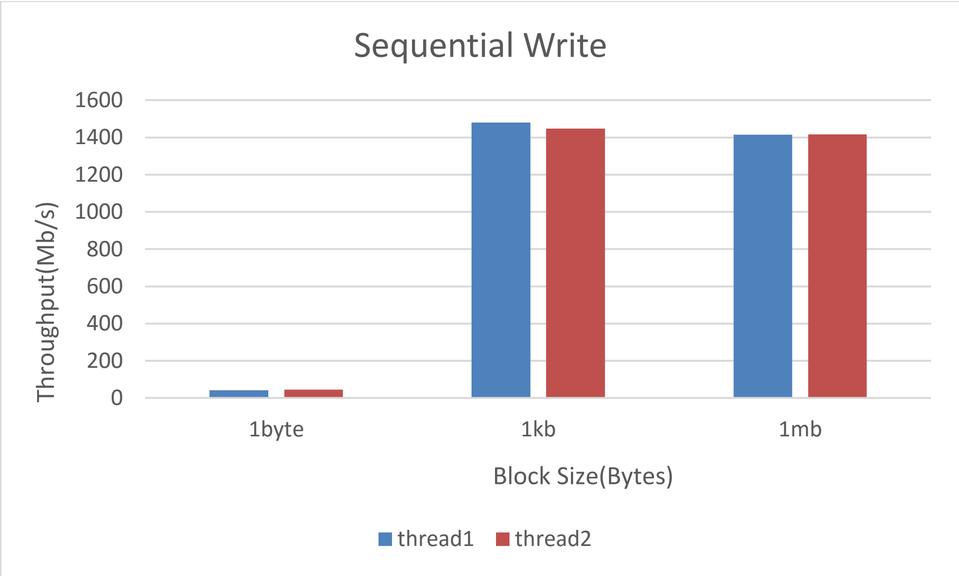
Sequential write	Thread 1	Thread 2
1byte	41.777726	45.702168
1Kb	1479.95531	1447.47579
1 Mb	1415.42817	1416.93234

Random read	Thread 1	Thread 2
1byte	6.178667	6.40438
1Kb	4770.14451	4775.95168
1 Mb	8928.57143	10422.0948

Sequential Read	Thread 1	Thread 2
1byte	33.288505	32.852159
1Kb	3837.03871	4287.06525
1 Mb	5649.71751	7446.01638

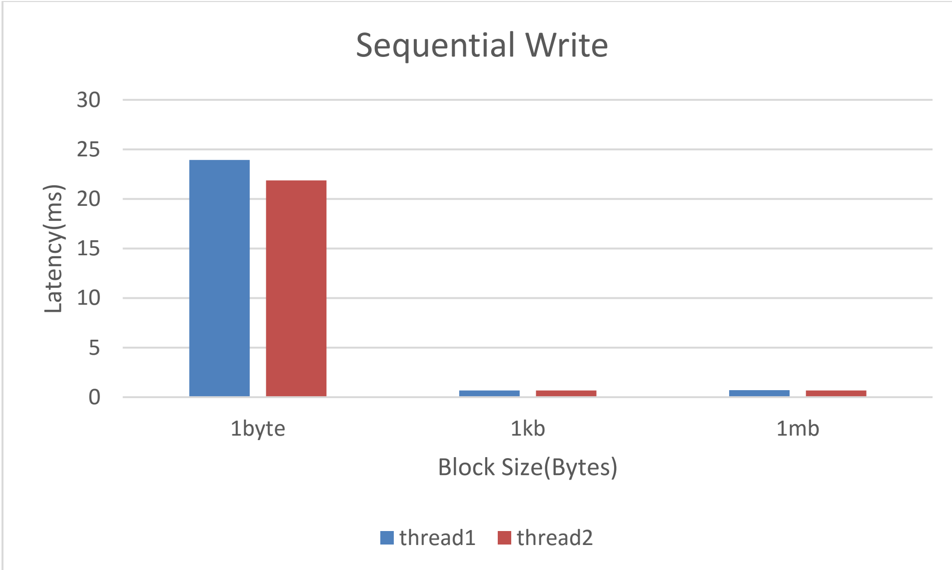
2. Sequential Write operation for varying block sizes, for varying concurrency (1 thread/ 2 thread). It is observed that the throughput increases with the block size. Also, the throughput values are almost same for 1 and 2 threads. Throughput is maximum for 1 kb.

Graph 9: Throughput vs Block Size for Sequential Write



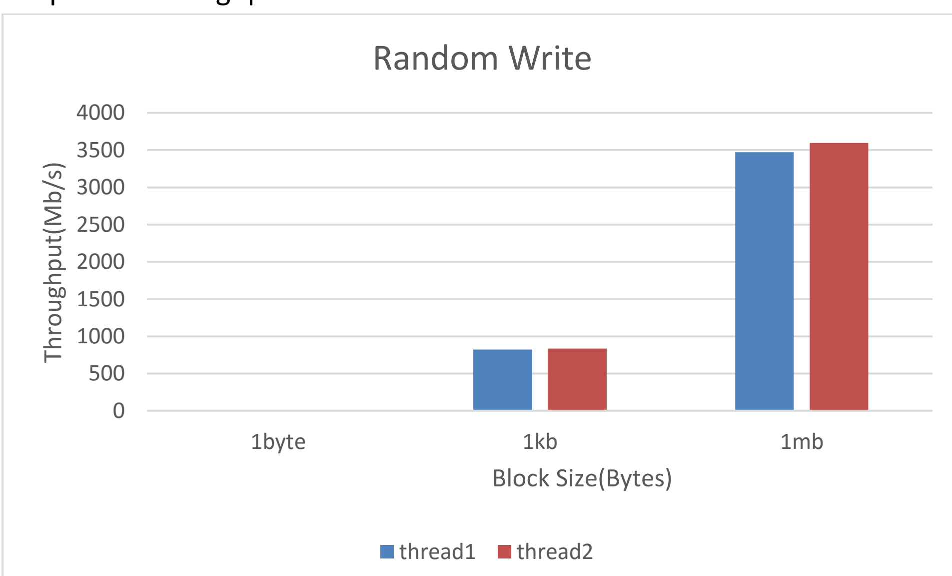
3. Sequential Write Operation for varying block size, for varying concurrency (1 thread, 2 thread). It is observed that the latency is maximum for 1 Byte and decreases with increase in block size. Also, the Latency is almost same for 1 and 2 threads.

Graph 10: Latency vs Block Size for Sequential Write



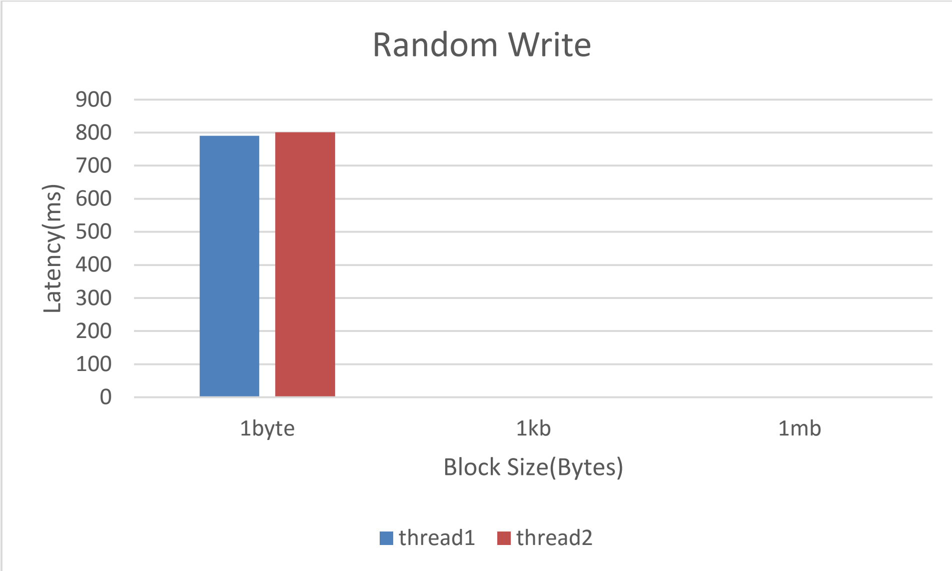
4. Random Write Operation for varying block size, for varying concurrency (1 thread, 2 thread). It is observed that the throughput is extremely high for 1 MB and decreases drastically with decrease in block size. Also, the throughput is almost same for 1 and 2 threads.

Graph 11: Throughput vs Block Size for Random Write



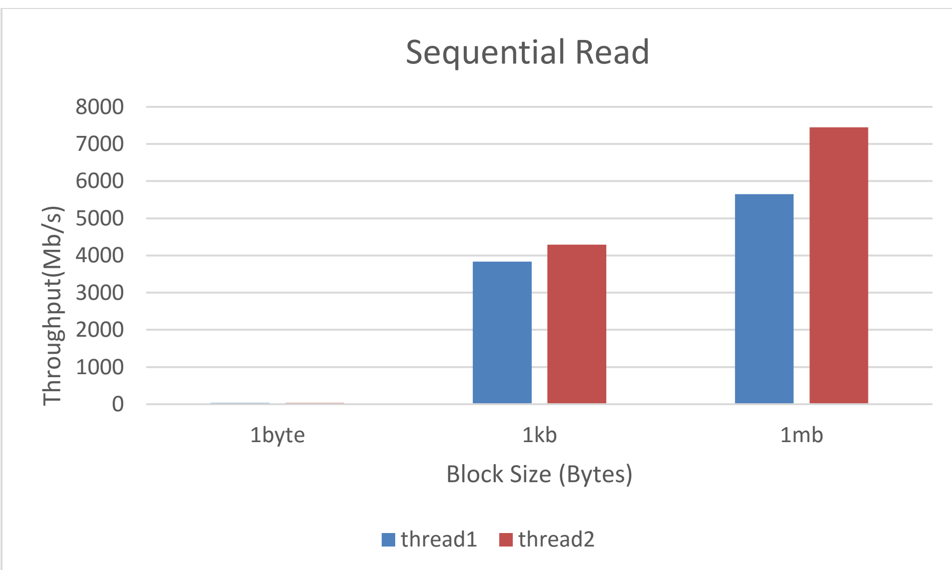
5. Random Write Operation for varying block size, for varying concurrency (1 thread, 2 thread). It is observed that the latency is maximum for 1 Byte and is almost negligible as the block size increases. Also, the Latency is almost same for 1 and 2 threads.

Graph 12: Latency vs Block Size for Random Write



6. Sequential Read Operation for varying block size, for varying concurrency (1 thread, 2 thread). It is observed that the throughput is extremely high for 1 MB and decreases drastically with decrease in block size. Also, the throughput is almost same for 1 and 2 threads. Throughput is maximum for thread no 2 with increase in block size.

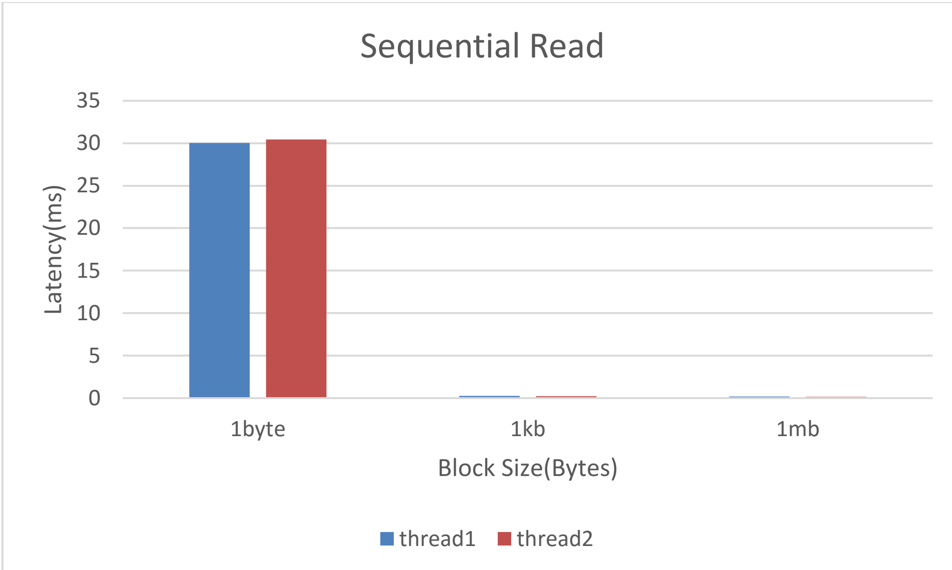
Graph 8: Throughput vs Block Size for Sequential Read



7. Sequential Read Operation for varying block size, for varying concurrency (1 thread, 2 thread). It is observed that the latency is extremely high for 1 Byte and decreases

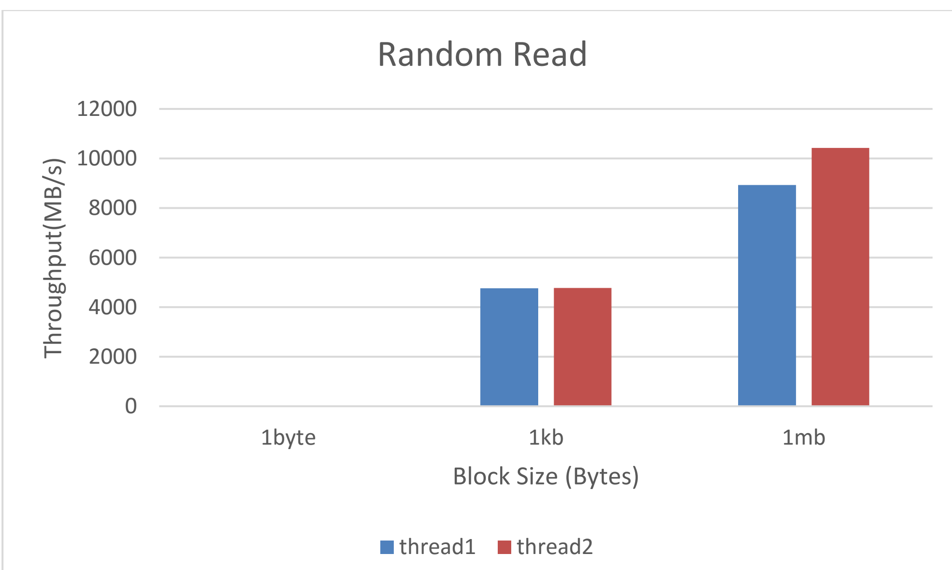
drastically with increase in block size. Also, the latency is almost same for 1 and 2 threads.

Graph 9: Latency vs Block Size for Sequential Write



8. Random Read Operation for varying block size, for varying concurrency (1 thread, 2 thread). It is observed that the throughput is extremely high for 1 MB and decreases drastically with decrease in block size. Also, the throughput is almost same for 1 and 2 threads.

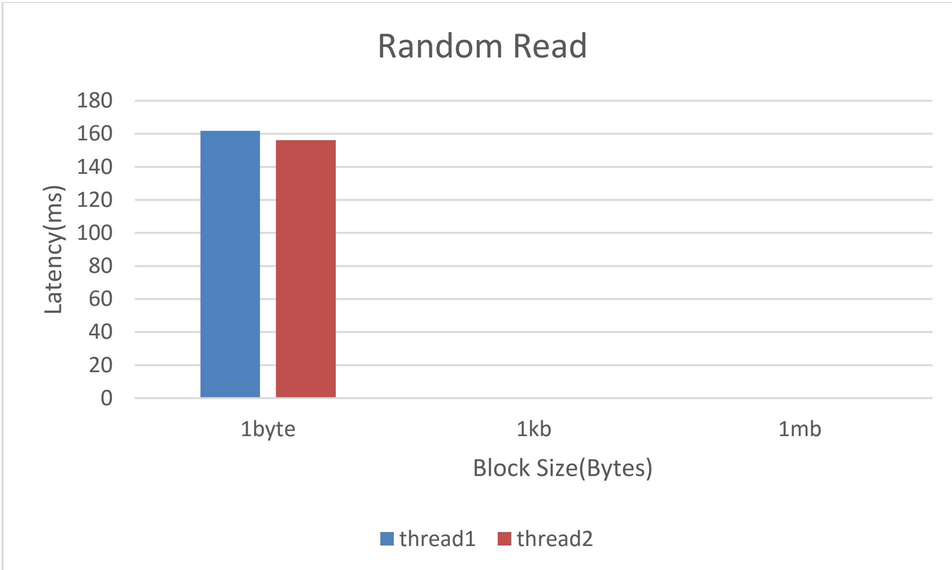
Graph 10: Throughput vs Block Size for Sequential Write



9. Random Read Operation for varying block size, for varying concurrency (1 thread, 2 thread). It is observed that the latency is extremely high for 1 Byte and decreases

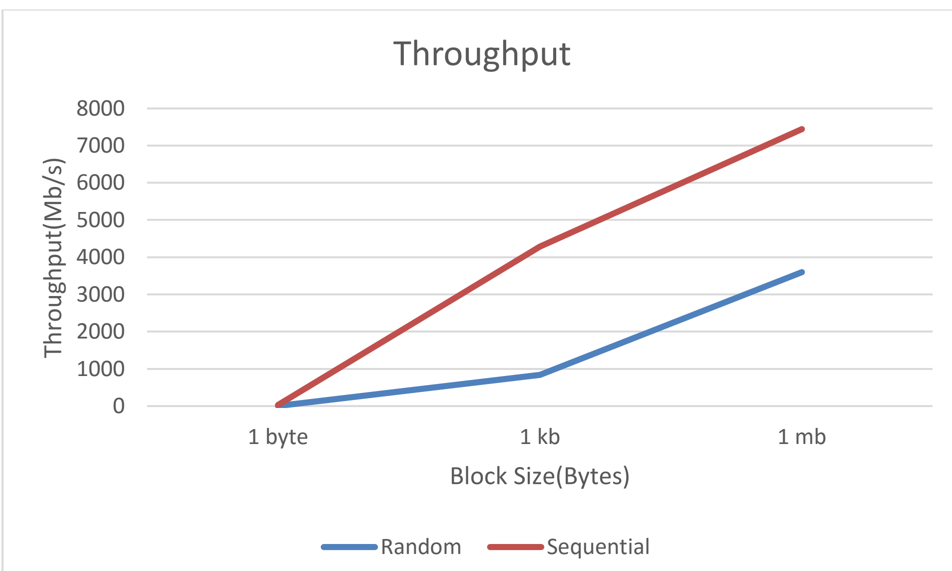
drastically with increase in block size. Also, the latency is almost same for 1 and 2 threads.

Graph 11: Latency vs Block Size for Sequential Write



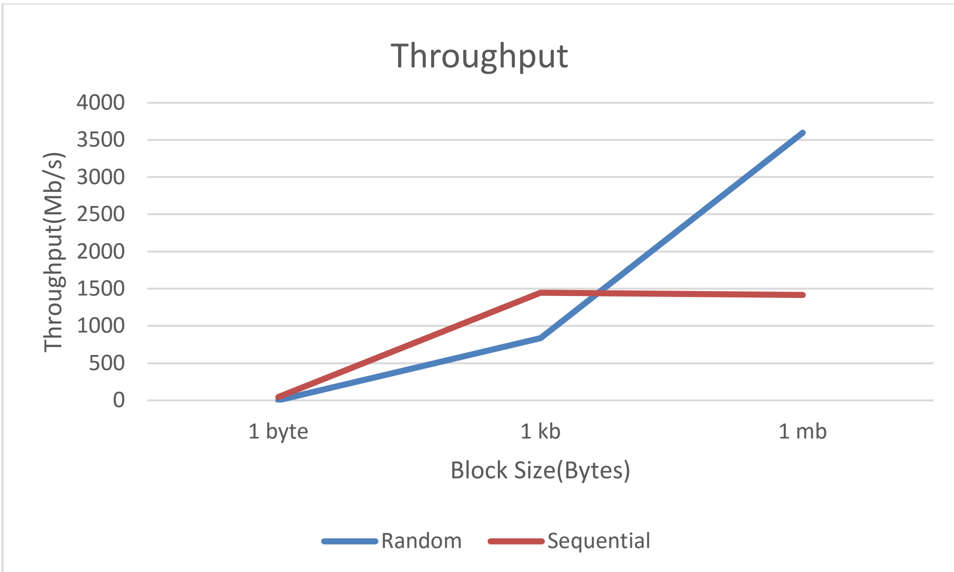
10. Throughput for Sequential Access Read Operation is more than throughput for Random Access Read Operation for 1 thread. Same observed for 2 threads.

Graph 12: Throughput vs Block Size for Comparing Sequential and Random Access throughput.



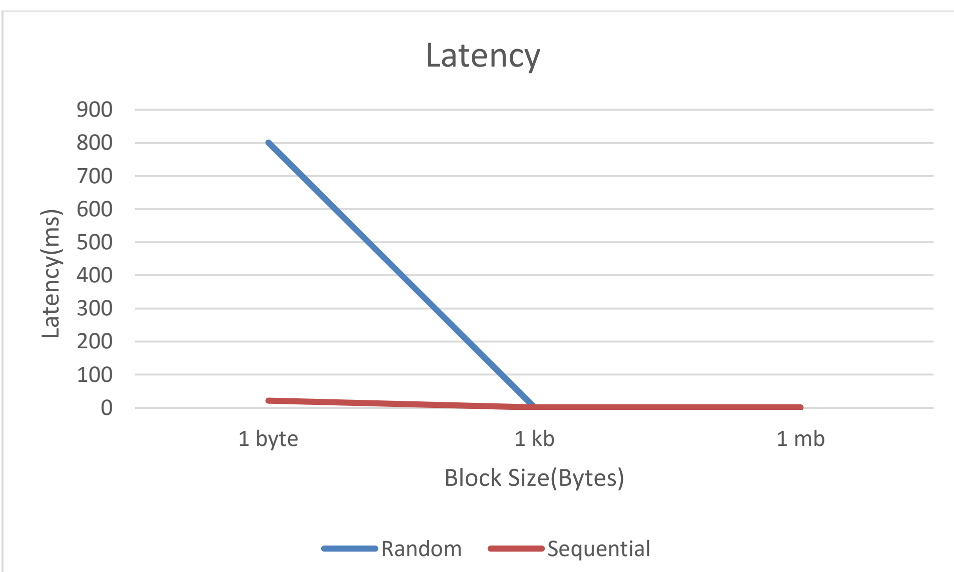
11. Throughput for Random Access Write Operation is more than throughput for Sequential Access Write Operation for 1Mb for 1 thread. Similar observations are made for 2 threads.

    Graph 13: Throughput vs Block Size for Comparing Sequential and Random Access throughput for Write operation 1 thread.



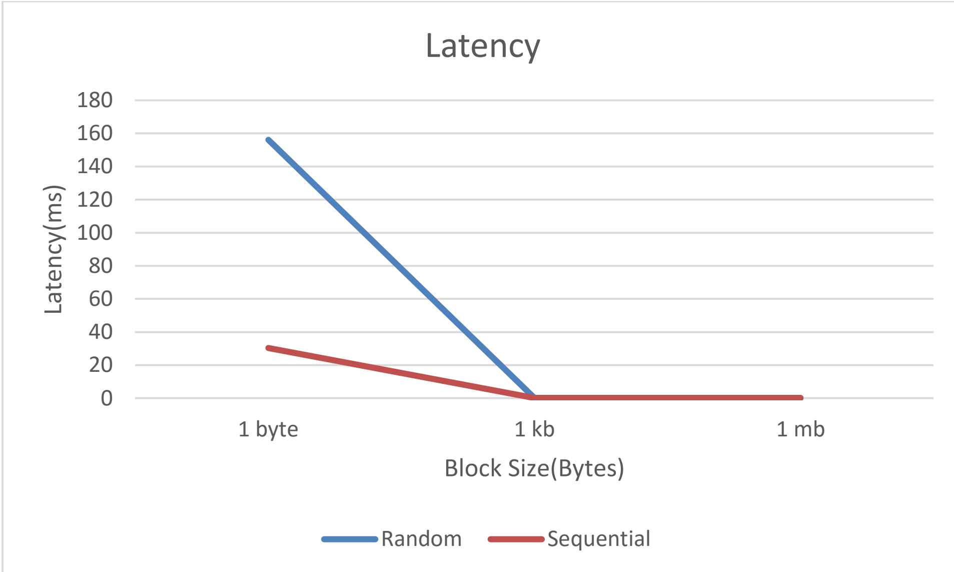
12. Latency for Random Access Write Operation is more than latency for Sequential Access Write Operation for 1 thread for 1 byte. Similar observation for 2 threads.

    Graph 14: Latency vs Block Size for Comparing Sequential and Random Access throughput for Write operation for 1 thread. Similar observation for 2 threads.



13. Latency for Random Access Read Operation is more than latency for Sequential Access Read Operation for 1 thread. Similar observation for 2 threads.

Graph 15: Latency vs Block Size for Comparing Sequential and Random Access throughput for Write operation for 1 thread.

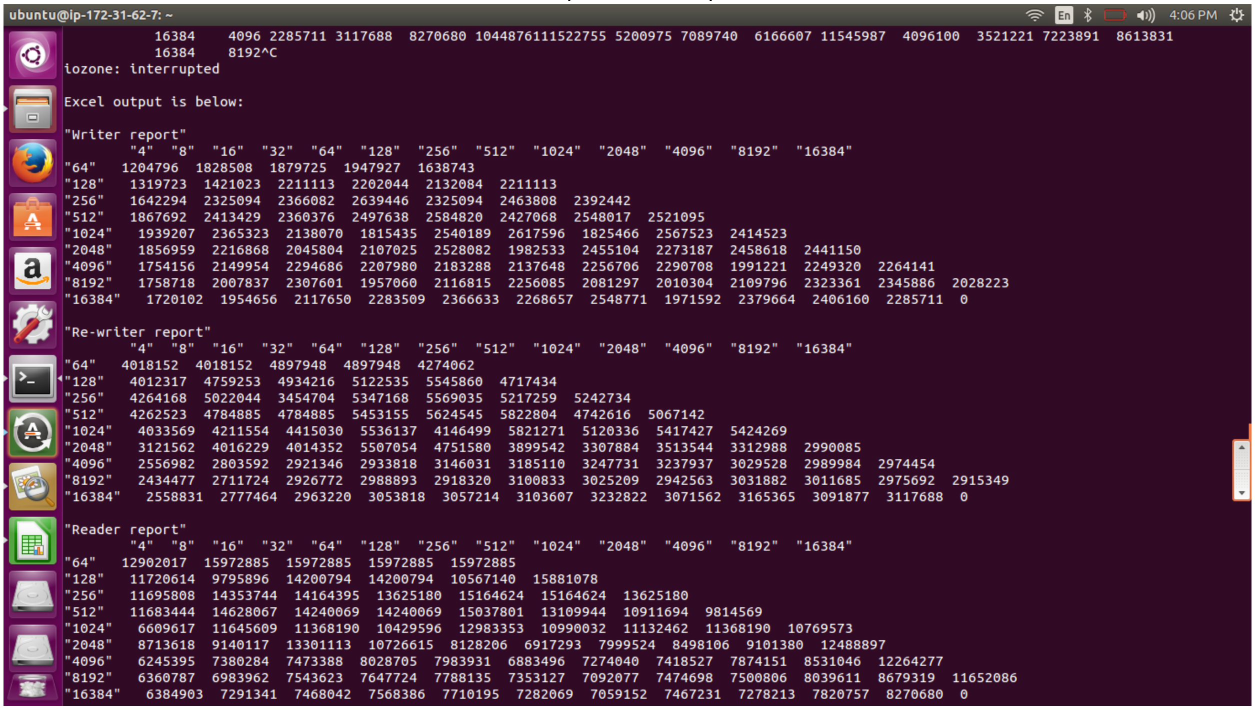


14. Optimal number of concurrency for best performance is 2.
15. More the number of threads more throughput.

IOZone Benchmark:

1. IOZone Benchmark is run for verifying outcomes of the implemented benchmark program

Operation	IOZone(Mb/s)	Observation(Mb/s)	Efficiency
Random Write	5710.61	828.055	14.5%
Random Read	10454.984	4773.04	45.65%
Sequential Read	10769.573	4062.0475	37.71%
Sequential Write	2458.618	1463.71	59.53%



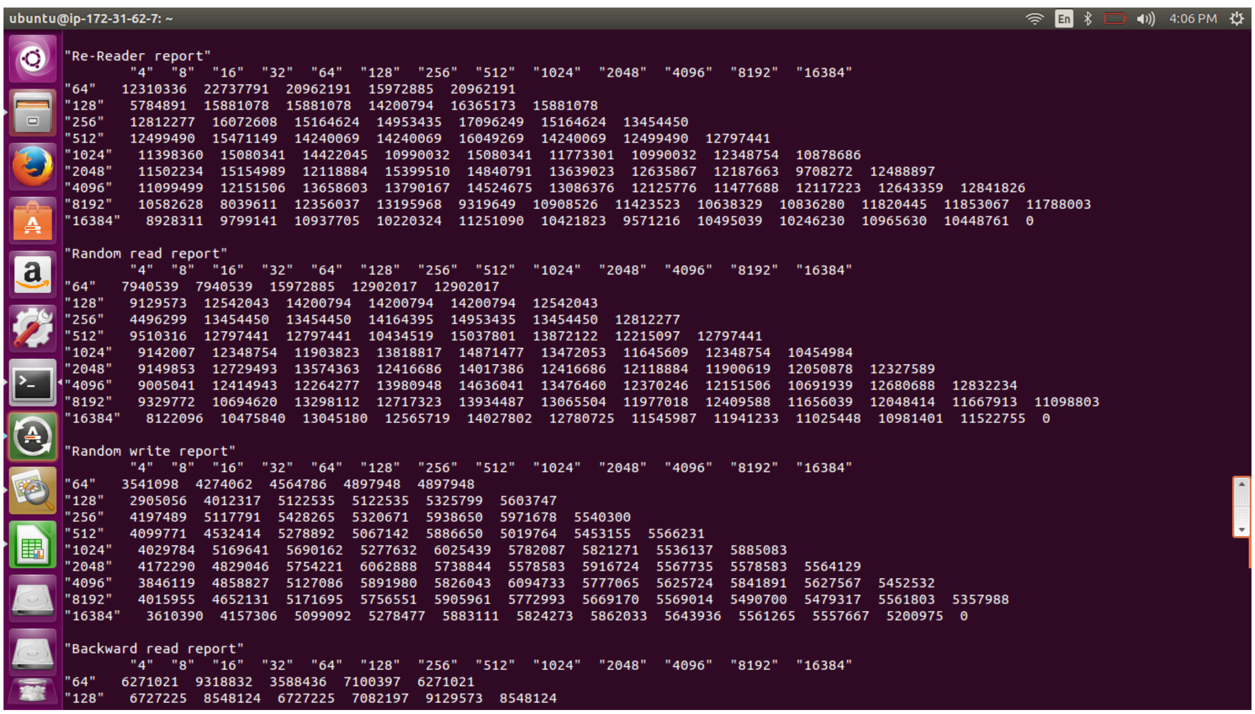
```
           16384    4096 2285711 3117688  8270680 1044876111522755 5200975 7089740  6166607 11545987  4096100  3521221 7223891  8613831
           16384    8192^C
iozone: interrupted

Excel output is below:

"Writer report"
        "4"  "8"  "16"  "32"  "64"  "128"  "256"  "512"  "1024"  "2048"  "4096"  "8192"  "16384"
"64"   1204796  1828508  1879725  1947927  1638743
"128"   1319723  1421023  2211113  2202044  2132084  2211113
"256"   1642294  2325094  2366082  2639446  2325094  2463808  2392442
"512"   1867692  2413429  2360376  2497638  2584820  2427068  2548017  2521095
"1024"   1939207  2365323  2138070  1815435  2540189  2617596  1825466  2567523  2414523
"2048"   1856959  2216868  2045804  2107025  2528082  1982533  2455104  2273187  2458618  2441150
"4096"   1754156  2149954  2294686  2207980  2183288  2137648  2256706  2290708  1991221  2249320  2264141
"8192"   1758718  2007837  2307601  1957060  2116815  2256085  2081297  2010304  2109796  2323361  2345886  2028223
"16384"   1720102  1954656  2117650  2283509  2366633  2268657  2548771  1971592  2379664  2406160  2285711  0

"Re-writer report"
        "4"  "8"  "16"  "32"  "64"  "128"  "256"  "512"  "1024"  "2048"  "4096"  "8192"  "16384"
"64"   4018152  4018152  4897948  4897948  4274062
"128"   4012317  4759253  4934216  5122535  5545860  4717434
"256"   4264168  5022044  3454704  5347168  5569035  5217259  5242734
"512"   4262523  4784885  4784885  5453155  5624545  5822804  4742616  5067142
"1024"   4033569  4211554  4415030  5536137  4146499  5821271  5120336  5417427  5424269
"2048"   3121562  4016229  4014352  5507054  4751580  3899542  3307884  3513544  3312988  2990085
"4096"   2556982  2803592  2921346  2933818  3146031  3185110  3247731  3237937  3029528  2989984  2974454
"8192"   2434477  2711724  2926772  2988893  2918320  3100833  3025209  2942563  3031882  3011685  2975692  2915349
"16384"   2558831  2777464  2963220  3053818  3057214  3103607  3232822  3071562  3165365  3091877  3117688  0

"Reader report"
        "4"  "8"  "16"  "32"  "64"  "128"  "256"  "512"  "1024"  "2048"  "4096"  "8192"  "16384"
"64"   12902017  15972885  15972885  15972885  15972885
"128"   11720614  9795896  14200794  14200794  10567140  15881078
"256"   11695808  14353744  14164395  13625180  15164624  15164624  13625180
"512"   11683444  14628067  14240069  14240069  15037801  13109944  10911694  9814569
"1024"   6609617  11645609  11368190  10429596  12983353  10990032  11132462  11368190  10769573
"2048"   8713618  9140117  13301113  10726615  8128206  6917293  7999524  8498106  9101380  12488897
"4096"   6245395  7380284  7473388  8028705  7983931  6883496  7274040  7418527  7874151  8531046  12264277
"8192"   6360787  6983962  7543623  7647724  7788135  7353127  7092077  7474698  7500806  8039611  8679319  11652086
"16384"   6384903  7291341  7468042  7568386  7710195  7282069  7059152  7467231  7278213  7820757  8270680  0
```



```
"Re-Reader report"
        "4"  "8"  "16"  "32"  "64"  "128"  "256"  "512"  "1024"  "2048"  "4096"  "8192"  "16384"
"64"   12310336  22737791  20962191  15972885  20962191
"128"   5784891  15881078  15881078  14200794  16365173  15881078
"256"   12812277  16072608  15164624  14953435  17096249  15164624  13454450
"512"   12499490  15471149  14240069  14240069  16049269  14240069  12499490  12797441
"1024"   11398360  15080341  14422045  10990032  15080341  11773301  10990032  12348754  10878686
"2048"   11502234  15154989  12118884  15399510  14840791  13639023  12635867  12187663  9708272  12488897
"4096"   11099499  12151506  13658603  13790167  14524675  13086376  12125776  11477688  12117223  12643359  12841826
"8192"   10582628  8039611  12356037  13195968  9319649  10908526  11423523  10638329  10836280  11820445  11853067  11788003
"16384"   8928311  9799141  10937705  10220324  11251090  10421823  9571216  10495039  10246230  10965630  10448761  0

"Random read report"
        "4"  "8"  "16"  "32"  "64"  "128"  "256"  "512"  "1024"  "2048"  "4096"  "8192"  "16384"
"64"   7940539  7940539  15972885  12902017  12902017
"128"   9129573  12542043  14200794  14200794  14200794  12542043
"256"   4496299  13454450  13454450  14164395  14953435  13454450  12812277
"512"   9510316  12797441  12797441  10434519  15037801  13872122  12215097  12797441
"1024"   9142007  12348754  11903823  13818817  14871477  13472053  11645609  12348754  10454984
"2048"   9149853  12729493  13574363  12416686  14017386  12416686  12118884  11900619  12050878  12327589
"4096"   9005041  12414943  12264277  13980948  14636041  13476460  12370246  12151506  10691939  12680688  12832234
"8192"   9329772  10694620  13298112  12717323  13934487  13065504  11977018  12409588  11656039  12048414  11667913  11098803
"16384"   8122096  10475840  13045180  12565719  14027802  12780725  11545987  11941233  11025448  10981401  11522755  0

"Random write report"
        "4"  "8"  "16"  "32"  "64"  "128"  "256"  "512"  "1024"  "2048"  "4096"  "8192"  "16384"
"64"   3541098  4274062  4564786  4897948  4897948
"128"   2905056  4012317  5122535  5122535  5325799  5603747
"256"   4197489  5117791  5428265  5320671  5938650  5971678  5540300
"512"   4099771  4532414  5278892  5067142  5886650  5019764  5453155  5566231
"1024"   4029784  5169641  5690162  5277632  6025439  5782087  5821271  5536137  5885083
"2048"   4172290  4829046  5754221  6062888  5738844  5578583  5916724  5567735  5578583  5564129
"4096"   3846119  4858827  5127086  5891980  5826043  6094733  5777065  5625724  5841891  5627567  5452532
"8192"   4015955  4652131  5171695  5756551  5905961  5772993  5669170  5569014  5490700  5479317  5561803  5357988
"16384"   3610390  4157306  5099092  5278477  5883111  5824273  5862033  5643936  5561265  5557667  5200975  0

"Backward read report"
        "4"  "8"  "16"  "32"  "64"  "128"  "256"  "512"  "1024"  "2048"  "4096"  "8192"  "16384"
"64"   6271021  9318832  3588436  7100397  6271021
"128"   6727225  8548124  6727225  7082197  9129573  8548124
```