# Benchmarking

CPU/MEMORY/DISK AND NETWORK

Chetan Gupta | CS553 | 30/03/2018

# 1: INTRODUCTION

Benchmarking of the components of the processor, memory, network and hard disk is complete. Several experiments are performed on different parameters. Three tests for each experiment are completed, and the mean and average deviation of the results is mentioned. Each experiment takes several seconds to reduce the effect of the initial load or any other system noise. The general values are used on the graphs. This document illustrates the design strategy for CPU / memory / disk/ network and has a corresponding section under each of them, showing diagrams / trends.

## 1.1 GENERAL POINTS

We have created a single interface for calling all test checks based on command line parameters. Benchmarking code for memory, network and disk is written in C language and CPU is written in C programming.

## 1.2 TESTING PLATFORM

I have use used my CPU and Hyperian cluster for testing my codes, details are as follows:-

# 2: CPU Benchmarking

For CPU benchmarking we have calculated both IOPS and FLOPS with the help of C programming language. Following steps are followed for the calculation:

a. An input is taken from the user about the number of threads which is to be assigned for the benchmarking.

b. The user can choose 1, 2, or 4 threads from options for calculating IOPS/FLOPS(GOPS) of CPU. The threads are parallelized to perform the operations.

c. For calculating the IOPS/FLOPS all the threads do the same calculation for $(2^{31}-1)$ times and the total time taken for performing these operations are recorded.

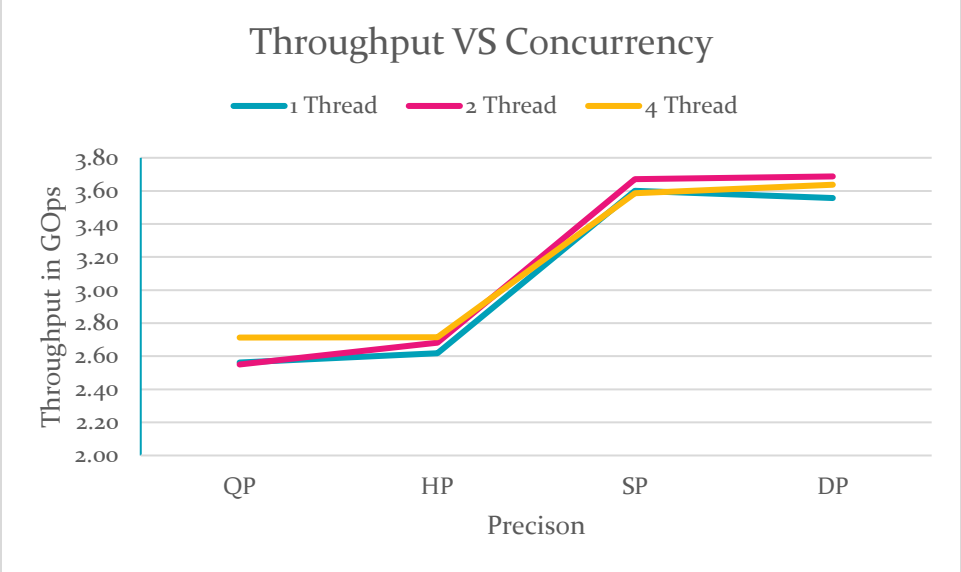
d. Each thread are doing same set of calculations one Trillion times. Total time taken for these operations is recorded, and this is used for calculating the IOPS / FLOPS(GOPs).

e. The sums are calculated, and these are used for plotting the graph using GNU plot.

| | | **Measured Ops/Sec** | **Measured Ops/Sec** | **Ops/Sec (GigaOPS)** | **Efficiency (%)** | **Efficiency (%)** |
|---|---|---|---|---|---|---|

| | | (GigaOPS) | (GigaOPS) | (Hyperion) | | |
|---|---|---|---|---|---|---|
| QP | 1 | 2.562159 | N/A | 588.8 | 0.44 | N/A |
| QP | 2 | 2.5503425 | N/A | 588.8 | 0.43 | N/A |
| QP | 4 | 2.713115 | N/A | 588.8 | 0.46 | N/A |
| HP | 1 | 2.617619 | N/A | 294.4 | 0.89 | N/A |
| HP | 2 | 2.681138667 | N/A | 294.4 | 0.91 | N/A |
| HP | 4 | 2.7148875 | N/A | 294.4 | 0.92 | N/A |
| SP | 1 | 3.6003485 | N/A | 147.2 | 2.45 | N/A |
| SP | 2 | 3.67169 | N/A | 147.2 | 2.49 | N/A |
| SP | 4 | 3.5848745 | N/A | 147.2 | 2.44 | N/A |
| DP | 1 | 3.557330667 | 35.25 | 73.6 | 4.83 | 10.09 |
| DP | 2 | 3.6872475 | 63.8 | 73.6 | 5.01 | 5.78 |
| DP | 4 | 3.637126667 | 70.12 | 73.6 | 4.94 | 5.19 |

### 2.1.1 PERFORMANCE CHART: GIOPS/GFLOPS VS THREADS




**Observations**:

a) GOPS values increases as tnumber of threads increases than becomes constant .

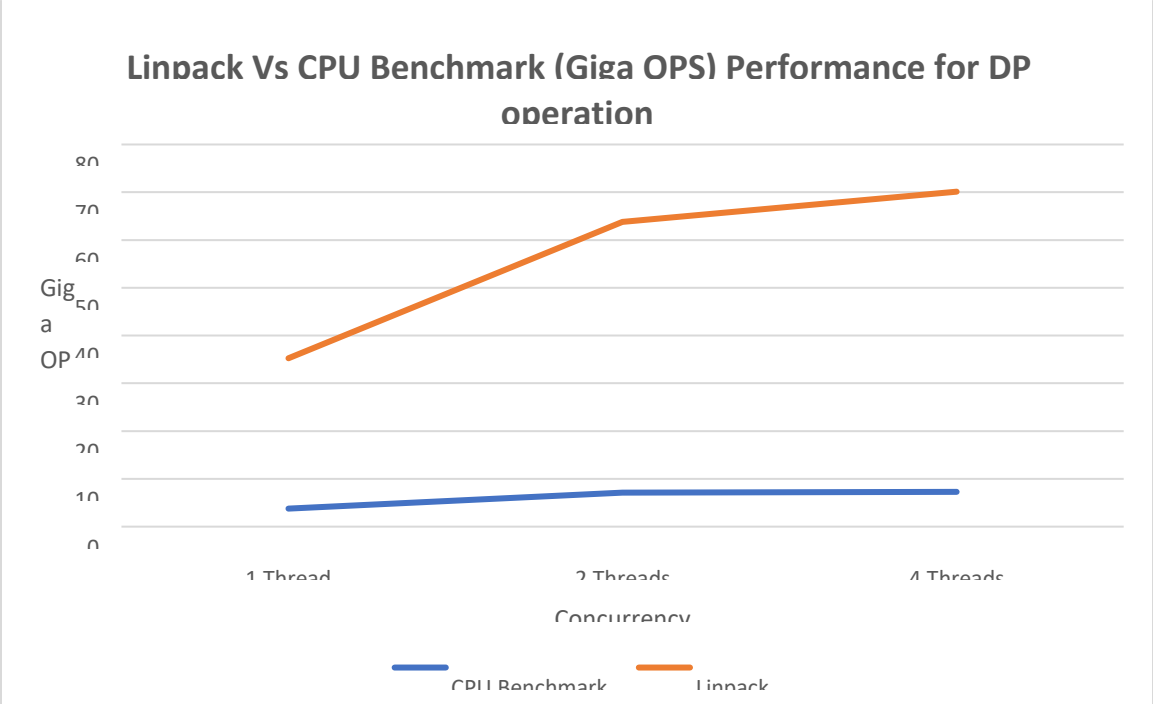
We used 1, 2, 4 threads for this experiment and 4 readings of GOPS is recorded and then plotted. To calculate the height performance we ran 2^31-1 arithmetic operations which include addition, multiplication and

subtraction. It also depicts that the increase in the number of threads improve GigaOPS to double the size of threads. It also shows that for the threads count 2 and 4, the GigaOPS tend to approximately similar; the reason being is that given number of threads similar to the system available threads (or higher than system available threads) provides or thus gives low GigaOPS sometimes due to thread switching.

### Conclusion-

It can be easily concluded from above bar graphs that the GigaOPS for single thread are almost half than that of two threads for each QP, HP, SP, and DP operations.

Below line graph represent comparison of Linpack and CPU benchmark DP floating point operation.




### Conclusion-

From above graph, we can see that ***Linpack*** benchmark gives highest performance for for DP floating point operations. The result of ***Linpack*** benchmark also follows the same pattern in growth of GigaOPS for the given number of threads similar to that of CPU benchmark measured on Hyperion. The reason behind high performance of Linpack benchmark is because of high performance CPU utilization and optimum implementation of parallelism.

## Processor Performance Metrics:

***GigaOPS*** = (No of Operations / Time) / $10^9$

***Theoretical OPS*** = 2.3 * 2 * instructions per second

***Theoretical OPS* QP**= 588.80 GigaOPS

***Theoretical OPS* HP**= 294.40 GigaOPS

***Theoretical OPS* SP**= 147.20 GigaOPS

*Theoretical OPS* DP**= 73.60 GigaOPS**

Maximum Efficiency is achieved by QP operations when the concurrency is 2 threads and efficiency is **18.010923 %.**

### Performance Comparison between CPU Benchmark, Linpack Benchmark and Theoretical peak performance

GigaOPS measured by CPU Benchmark for all operations follow the same growth rate as the number of threads increase which is alike to the Linpack performance in growth rate which gets double when thread size increases from 1 to 2 and remains approximately similar when threads size increases from 2 to 4. However, theoretical throughput is computed based on number of instructions of each operation multiplied by number of cores into system and CPU speed.

## 3. Memory Benchmarking:

For Memory benchmarking we measure the speed of memory with the help of copy operations which are read and write with different block sizes such as 8b, 8KB, 8MB and 80MB. Following steps are followed for the calculation:

a. An input is taken from the user about the memory block size (1 Byte) or (1KB) or (1 MB) or (10MB)), and number of threads are also asked by the user (1 or 2 or 4) to calculate the memory speed and the space is allocated by read and write.
b. Access type such as Sequential or Random are taken as input from the user for memory benchmark.
c. Block by block copy is performed for the sequential access.
d. Copy is performed based on randomly derived blocks size number for random access.
e. Memory blocks are distributed between threads for multiple thread operations and each thread works on one half of the fully allocated memory space.
f. For the evaluation of the Memory benchmark, I used GBps for the measuring the throughput of the memory.
g. I ran the Memory benchmark for concurrency of 1,2,4 threads with Read-Write Sequential and Read-Write Random access patterns with workload of 100GB data with block sizes of 1KB, 1MB and 10MB.

h. I also used the rand () function from ***math.h*** library to generate the random numbers needed for RWR access pattern.

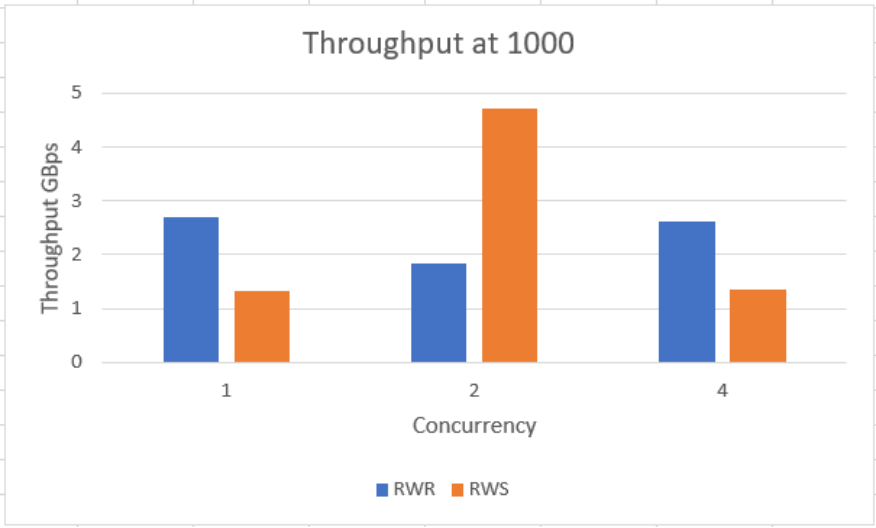
i. For comparison, I ran the ***pmbw*** benchmark to measure memory subsystem performance presented in tabular format below.

Below table 2 shows the Memory Throughput measured on Hyperion Cluster.

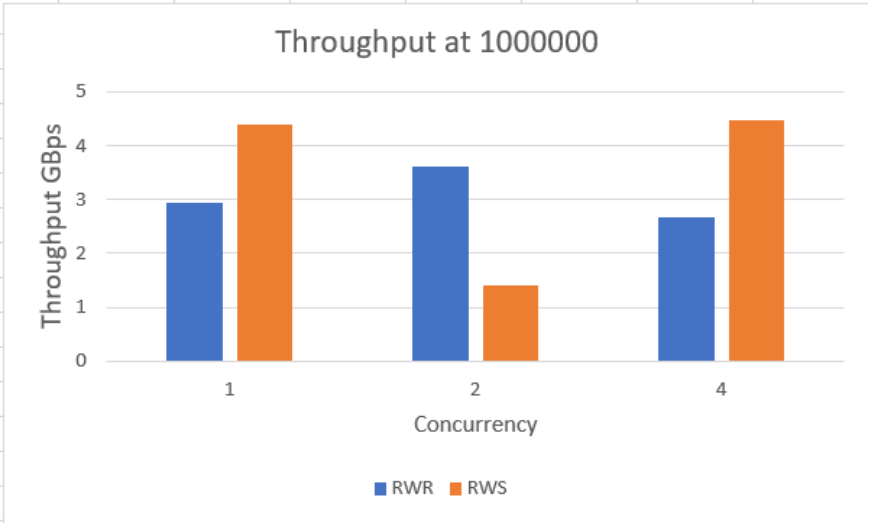
### 3.1 Performance Chart and details:

| Workload | ConCurrency | Block Size | MyRAMBench Measured Throughput (GB/sec) | Pmbw Measured Throughput (GB/sec) | Theoretical Throughput (GB/sec) | MyRAMBench Efficiency (%) | pmbw Efficiency (%) |
|---|---|---|---|---|---|---|---|
| RWS | 1 | 1000 | 2.216058 | 16.5 | 68.256 | 3.246686006 | 24.173699 |
| RWS | 1 | 1000000 | 1.416238667 | 20.7 | 68.256 | 2.074892561 | 30.3270042 |
| RWS | 1 | 10000000 | 2.387083333 | 19.9 | 68.256 | 3.497250547 | 29.1549461 |
| RWS | 2 | 1000 | 1.208943 | 24.9 | 68.256 | 1.771189346 | 36.4803094 |
| RWS | 2 | 1000000 | 2.129539667 | 15 | 68.256 | 3.11993036 | 21.97609 |
| RWS | 2 | 10000000 | 1.351341667 | 36 | 68.256 | 1.97981374 | 52.742616 |
| RWS | 4 | 1000 | 4.797787 | 22.6 | 68.256 | 7.029106599 | 33.1106423 |
| RWS | 4 | 1000000 | 1.337346667 | 34.6 | 68.256 | 1.959310048 | 50.6915143 |
| RWS | 4 | 10000000 | 4.783803333 | 38.7 | 68.256 | 7.008619511 | 56.6983122 |
| RWR | 1 | 1000 | 1.815215667 | 4.2 | 68.256 | 2.659422859 | 6.1533052 |
| RWR | 1 | 1000000 | 6.453729 | 0.61 | 68.256 | 9.455181962 | 0.89369433 |
| RWR | 1 | 10000000 | 2.890362667 | 0.49 | 68.256 | 4.234591342 | 0.71788561 |
| RWR | 2 | 1000 | 4.818662667 | 8.9 | 68.256 | 7.059690967 | 13.0391467 |
| RWR | 2 | 1000000 | 2.886603 | 1.24 | 68.256 | 4.229083158 | 1.81669011 |
| RWR | 2 | 10000000 | 5.943713667 | 0.85 | 68.256 | 8.707972437 | 1.24531177 |
| RWR | 4 | 1000 | 1.777489333 | 19.4 | 68.256 | 2.604151039 | 28.4224098 |
| RWR | 4 | 1000000 | 6.099959 | 3.01 | 68.256 | 8.936883204 | 4.40986873 |
| RWR | 4 | 10000000 | 2.995375667 | 0.91 | 68.256 | 4.388443018 | 1.33321613 |

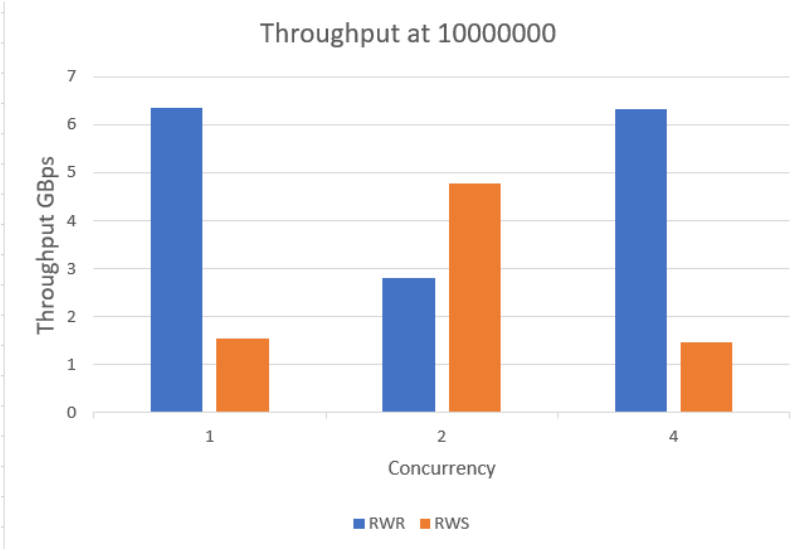
### Throughput for Sequential and Random operation in Log plot: Throughput (Y axis) in Mbps with different Operation (X axis) at 1000Bytes of Block size



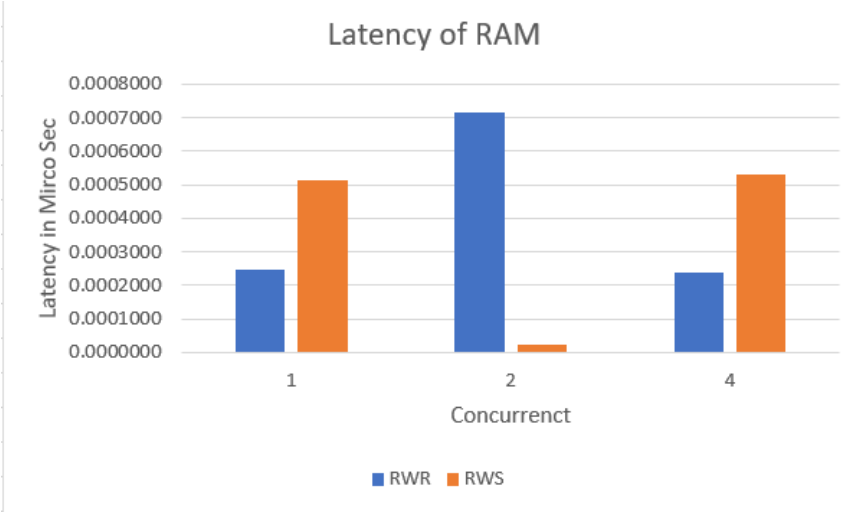

Throughput for Sequential and Random operation in Log plot: Throughput (Y axis) in Mbps with different Operation (X axis) at 100000Bytes of Block size




Throughput for Sequential and Random operation in Log plot: Throughput (Y axis) in Mbps with different Operation (X axis) at 1000000Bytes of Block size




Throughput for Sequential and Random operation in Log plot: Latency (Y axis) in Microseconds with different concunrrency (X axis) at 1Bytes of Block size




**Observations**:

a) Read /write speed in sequential access is much more compared to random access. This is because sequential access will have more cache hits as data is accessed sequentially
b) Both random and sequential access performed best with block size of 8KB.

## Conclusion-

Above tables gave us brief about that the throughput and latency for RWS and RWR access patterns. For RWS access pattern, throughput gets doubled for given block sizes as number of threads increase from 1 to 2, remains approximately similar when increased from 2 to 4 threads.

Additionally, RWS gives us less latency as compared to the ***pmbw*** measured and theoretical latency. Above tables gave us brief about that the throughput and latency for RWS and RWR access patterns. For RWR access pattern, throughput gets doubled for block sizes of 1MB and 10 MB as number of threads increase from 1 to 2, remains approximately similar when increased from 2 to 4 threads. In addition to this, RWR gives us high latency as compared to theoretical latency but less than ***pmbw*** measured, the reason being decrease in latency efficiency is that it involves randomly reading and writing data to memory which increases execution time. The reason behind improvement in the throughput is the block size which if increased causes decrease in the memory read and write operations. For block size of 1KB, RWR seems to be not changing considering its throughput for given number of threads.

**Conclusion:**

From the above comparison graphs, we can say growth in number of threads decreases latency for sequential read-write access pattern because concurrent multiple threads behavior. But growth in number of threads increases latency for random read-write access pattern due to selection of random location for reading data and writing it to memory which will increase the overall latency.

## Memory Performance Metrics:

*Workload = 1GB*

***Throughput*** = ((100 * Workload) / Time) / Workload GBps

***Theoretical Throughput*** = (2133 * 2 * 64 *2) / 8) / 1000 GBps

***Throughput Efficiency =*** (Throughput / Theoretical Throughput) *100

***Latency =*** (execution time * 1000000) / 100000000 microseconds

*Theoretical Latency* **= 0.014 microseconds**

***Latency Efficiency*** = *((Theoretical Latency - Latency)/ Theoretical Latency) * 100*

Maximum Throughput Efficiency is achieved by RWR access pattern when the concurrency is 2 threads and efficiency is **21.767445 %.**

Maximum Latency Efficiency is achieved by RWS access pattern when the concurrency is 4 threads and efficiency is **79.19 %.**

### Performance Comparison between Memory Benchmark, pmbw Benchmark and Theoretical performance

Throughput measured by Memory Benchmark for RWS access pattern increases as the number of threads increase which gets double when thread size increases from 1 to 2 and

remains approximately similar when threads size increases from 2 to 4. RWR also follows the same pattern that of the RWS except for block size 1KB. **pmbw** benchmark seems to have variations in the throughput for RWS access pattern but for RWR access pattern generates higher throughput for 1 thread and significantly decreases as the number of threads increases. For large block sizes, we get higher throughput due to less number of computations/operations to perform.

Growth rates in threads affect the performance of memory that decreases latency for RWS access pattern but increases for RWR access pattern.

## 4. Disk Benchmarking:

For Disk benchmarking, we analyze multiple read and write operations on disk in both sequential and random fashion. User also has an option to select different numbers of thread to analyze the change in the performance with different threads. Different block sizes of data is written and also read. Following steps are followed for the calculation:

a. For operations such as read/write, block size, sequential/random access and number of threads, the user will give the input for the operations.
b. For threads to do simultaneous read/write, create the required number of threads to parallelize the operations.
c. 10GB space should be provided for the operations to be performed.
d. Same string of data is read/written in a loop on all over the space of 10GB for sequential read/write.
e. As compared to sequential experiments the random access is slow, the number of read/write operations are less. The disk space is accessed in a random fashion which is 10GB.
f. When multiple thread are involved, each thread write to/ read from its own file of 10GB size to work on.

g. I ran the disk benchmark for concurrency of 1,2,4,8,16,32,64,128 threads with Random Read and Random Write access pattern while concurrency of 1,2,4 threads for Read Sequential and Write Sequential access patterns with workload of 10GB data with block sizes of 1KB, 1MB, 10MB and 100MB.

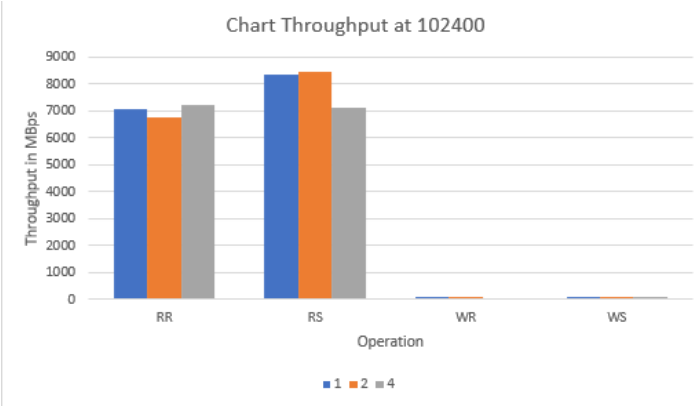
h. I also used the rand () function from math.h library to generate the random numbers needed for RR and WR access patterns.

i. For comparison, I ran the ***iozone*** benchmark to measure disk performance presented in tabular format below.

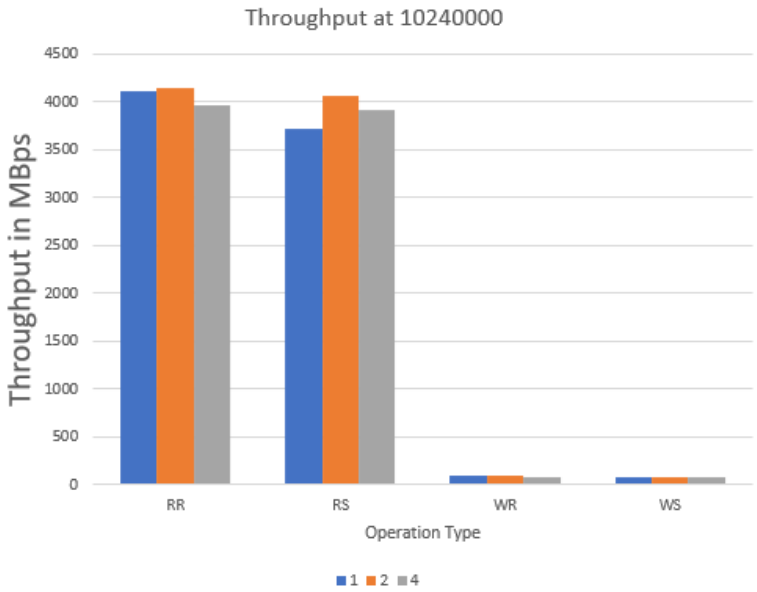
## 4.1 Performance measurement:

| Work-Load | Con-Currency | Block Size | MyDiskBench Measured Throughput (MB/sec) | IOZone Measured Throughput (MB/sec) | Theoretical Throughput (MB/sec) | MyDiskBench Efficiency (%) | IOZone Efficiency (%) |
|---|---|---|---|---|---|---|---|
| RS | 1 | 1024000 | 83.3673999 | 250 | 372 | 22.41059137 | 67.204301 |
| RS | 1 | 10240000 | 57.60887207 | 250 | 372 | 15.48625593 | 67.204301 |
| RS | 1 | 102400000 | 59.64 | 250 | 372 | 16.03225806 | 67.204301 |
| RS | 2 | 1024000 | 64.25495484 | 266 | 744 | 8.636418661 | 35.752688 |
| RS | 2 | 10240000 | 40.62854859 | 276 | 744 | 5.460826423 | 37.096774 |
| RS | 2 | 102400000 | 41.72107056 | 296 | 744 | 5.607670774 | 39.784946 |
| RS | 4 | 1024000 | 55.24890625 | 276 | 1488 | 3.71296413 | 18.548387 |
| RS | 4 | 10240000 | 40.92319458 | 290 | 1488 | 2.75021469 | 19.489247 |
| RS | 4 | 102400000 | 36.47611328 | 300 | 1488 | 2.451351699 | 20.16129 |
| WS | 1 | 1024000 | 0.77867577 | 255 | 172 | 0.452718471 | 148.25581 |
| WS | 1 | 10240000 | 0.80457352 | 250 | 172 | 0.467775302 | 145.34883 |
| WS | 1 | 102400000 | 0.815240635 | 230 | 172 | 0.473977113 | 133.72093 |
| WS | 2 | 1024000 | 0.779047695 | 299 | 344 | 0.226467353 | 86.918604 |
| WS | 2 | 10240000 | 0.78782349 | 315 | 344 | 0.229018456 | 91.569767 |
| WS | 2 | 102400000 | 0.792292095 | 360 | 344 | 0.230317469 | 104.65116 |
| WS | 4 | 1024000 | 0.812079695 | 350 | 688 | 0.118034839 | 50.872093 |
| WS | 4 | 10240000 | 0.776237605 | 370 | 688 | 0.112825233 | 53.779069 |
| WS | 4 | 102400000 | 0.81950409 | 360 | 688 | 0.119113967 | 52.325581 |
| RR | 1 | 1024000 | 70.82223145 | 135 | 540 | 13.11522805 | 25 |
| RR | 1 | 10240000 | 58.89396729 | 148 | 540 | 10.90629024 | 27.407407 |
| RR | 1 | 102400000 | 51.6678003 | 146 | 540 | 9.568111167 | 27.037037 |
| RR | 2 | 1024000 | 49.92111694 | 159 | 1080 | 4.622325643 | 14.722222 |
| RR | 2 | 10240000 | 41.38459595 | 169 | 1080 | 3.831907032 | 15.648148 |
| RR | 2 | 102400000 | 39.71565064 | 170 | 1080 | 3.677375059 | 15.74074 |
| RR | 4 | 1024000 | 50.24557374 | 160 | 2160 | 2.326183969 | 7.4074074 |
| RR | 4 | 10240000 | 40.79599243 | 162 | 2160 | 1.888703353 | 7.5 |
| RR | 4 | 102400000 | 37.92998291 | 165 | 2160 | 1.756017727 | 7.6388888 |
| WR | 1 | 1024000 | 0.77867577 | 240 | 410 | 0.18992092 | 58.536585 |
| WR | 1 | 10240000 | 0.84928627 | 242 | 410 | 0.207142993 | 59.02439 |
| WR | 1 | 102400000 | 0.84145653 | 248 | 410 | 0.2052333 | 60.487804 |
| WR | 2 | 1024000 | 0.772429925 | 280 | 820 | 0.094198771 | 34.146341 |
| WR | 2 | 10240000 | 0.83750626 | 283 | 820 | 0.10213491 | 34.512195 |
| WR | 2 | 102400000 | 0.86670681 | 305 | 820 | 0.105695952 | 37.195121 |
| WR | 4 | 1024000 | 0.475235985 | 201 | 1640 | 0.028977804 | 12.256097 |
| WR | 4 | 10240000 | 0.84975918 | 210 | 1640 | 0.051814584 | 12.804878 |
| WR | 4 | 102400000 | 0.927718085 | 211 | 1640 | 0.056568176 | 12.865853 |

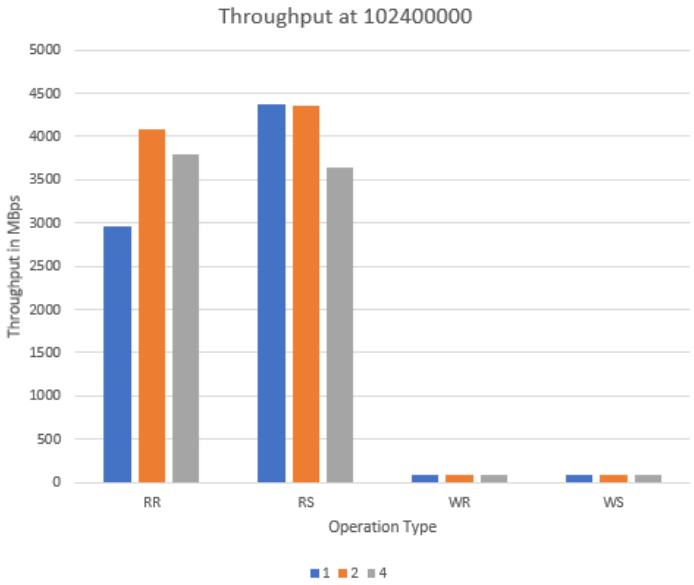
*Throughput in Mbps (Y axis**à** log plot**) for all operations at 102400 blocksize***




*Throughput in Mbps (Y axis**à** log plot**) for all operations at 10240000 blocksize***



*Throughput in Mbps (Y axis**à** log plot**) for all operations at 102400000 blocksize***



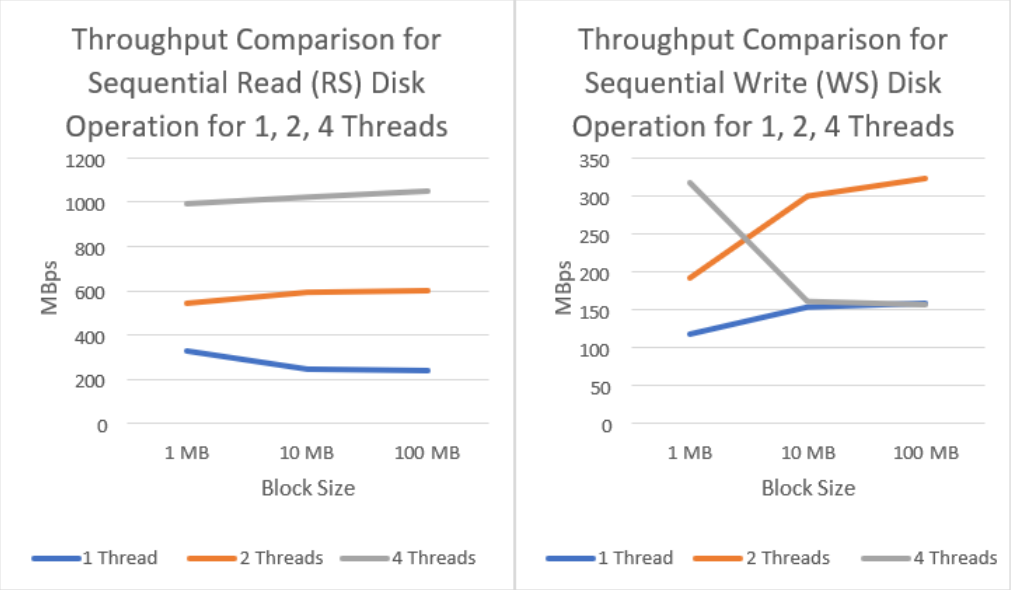

## Conclusion-

The table above along with the bar graphs show us that as the number of thread increases up to number of available threads in the system, the throughput increases. We can also see that as the block size increases the throughput increases, the reason being that increase in block size decreases number of disk read or write operations

Below line graphs represent throughput (Mbps) comparison for RS, WS, RR, WR against the concurrency of 1, 2 and 4 threads.

## Conclusion –

As can be seen from above line charts that for RS and RR operations, throughput increases with the increase in threads. For WS and WR operations, however throughput varies.




## Conclusion –

As can be seen from above line charts that for RS and RR operations, throughput increases with the increase in threads. For WS and WR operations, however throughput varies.

| Worklo ad | Concurre ncy | Bloc k Size | MyDiskBe nch Measured Latency(m s) | IOZONE Measured Latency( ms) | Theoretic al Latency( ms) | MyDiskBe nch Efficiency | IOZON E Efficien cy |
|---|---|---|---|---|---|---|---|
| RR | 1 | 1KB | 0.0329675 | 1.3 | 0.5 | 6.5935 | -160 |
| RR | 2 | 1KB | 0.0315535 | 1.3 | 0.5 | 6.3107 | -160 |
| RR | 4 | 1KB | 0.0319245 | 2 | 0.5 | 6.3849 | -300 |
| RR | 8 | 1KB | 0.034287 | 1.2 | 0.5 | 6.8574 | -140 |
| RR | 16 | 1KB | 0.030365 | 0.6 | 0.5 | 6.073 | -20 |
| RR | 32 | 1KB | 0.033113 | 1.1 | 0.5 | 6.6226 | -120 |
| RR | 64 | 1KB | 0.0408095 | 1.1 | 0.5 | 8.1619 | -120 |
| RR | 128 | 1KB | 0.0335895 | 0.56 | 0.5 | 6.7179 | -12 |
| WR | 1 | 1KB | 0.1087795 | 1.9 | 0.5 | 21.7559 | -280 |
| WR | 2 | 1KB | 0.1116235 | 1.9 | 0.5 | 22.3247 | -280 |
| WR | 4 | 1KB | 0.1113145 | 3.1 | 0.5 | 22.2629 | -520 |
| WR | 8 | 1KB | 0.120874 | 0.75 | 0.5 | 24.1748 | -50 |
| WR | 16 | 1KB | 0.132994 | 0.81 | 0.5 | 26.5988 | -62 |
| WR | 32 | 1KB | 0.151828 | 0.5 | 0.5 | 30.3656 | 0 |
| WR | 64 | 1KB | 0.1508375 | 0.55 | 0.5 | 30.1675 | -10 |
| WR | 128 | 1KB | 0.1629035 | 0.67 | 0.5 | 32.5807 | -34 |

## Conclusion-

From table, we can say that as number of threads increases the latency increases for Random-Write operation and for Random-Read, latency tends to be constant. Disk IOPS tends to be falling

gradually as the number of thread increases for Random-Read operation while IOPS for Random-Write tends to be constant over the number of threads.

## Disk Performance Metrics:

***Workload = 10GB***

***Throughput =*** **(**Workload / time) / 1000000 Mbps

*Theoretical Throughput* RS**= 372 Mbps** *Theoretical Throughput* WS**= 172 Mbps** *Theoretical Throughput* RR**= 540 Mbps** *Theoretical Throughput* WR**= 410 Mbps**

***Throughput Efficiency =*** (Throughput / Theoretical Throughput) *100

***Latency =*** (execution time * 1000) / 1000000 milliseconds

*Theoretical Latency* **= 0.5 milliseconds**

***Latency Efficiency*** *= ((Theoretical Latency - Latency)/ Theoretical Latency) * 100*

***Disk IOPS*** = 1000000 / execution time

***Theoretical IOPS*** = (*Theoretical Throughput* * 1000000) /1000

### Performance Comparison between Disk Benchmark, IOZONE Benchmark and Theoretical performance

Throughput measured by Disk Benchmark for RR and RS access pattern increases as the number of threads increase which gets double when thread size increases from 1 to 2 and remains approximately similar when threads size increases from 2 to 4. However, for Sequential and Random Write operations, throughput changes. Block size is also a significant factor in determining the throughput. Thread count increase also increases the latency in WR while keeps the constant in RR whereas disk IOPS gets reduced with threads increase in RR while disk IOPS becomes constant in WR. IOZONE efficiency computed for RS and RR is less than the MyDiskBench efficiency but greater for WS and WR operations.

## . Network Benchmarking:

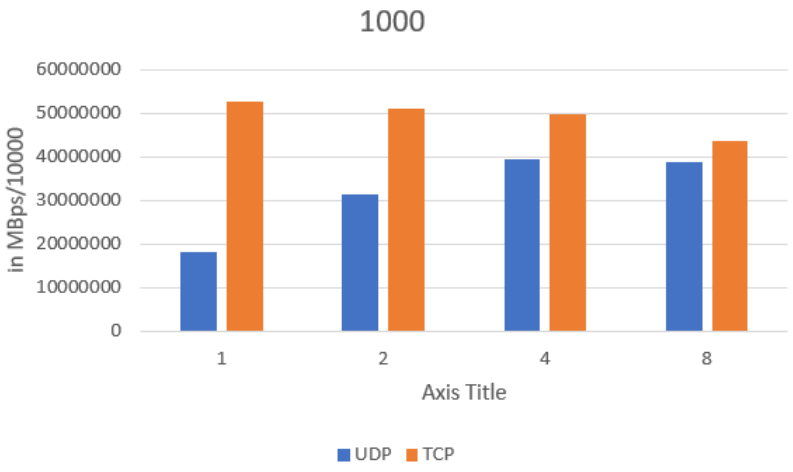
a. We are using strong scaling to measure N/W speed for loopback network.
b. Multithreading (up to 8 threads) concept we are using along with socket programming.
c. Calculating Latency (ms) and Throughput (Mb/sec) for loopback N/W and for this we are using 64 Kb buffer and sending 8k ping pong message .
d. Both UDP and TCP protocols are used.
e. Socket programming, we are using in C for both UDP and TCP.

f. I have not ran this code over Hyperian or any cluster performance matric has been taken through run over my personal system.

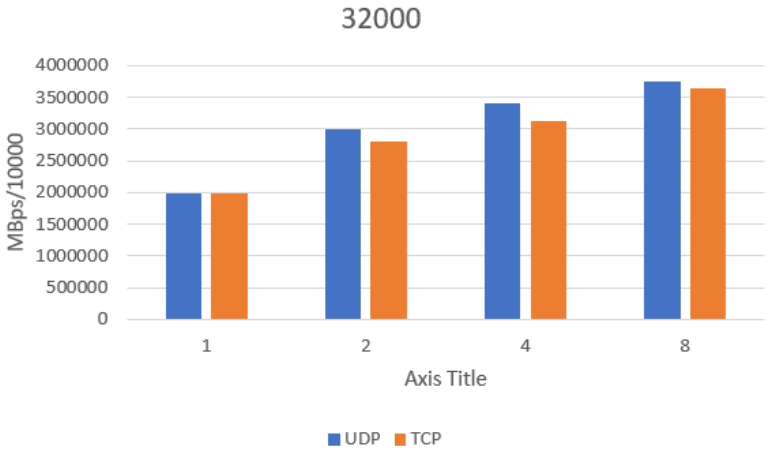
### 5.1 Performance measurement:

| Proto-col | Con-currency | Block Size | MyNETBench Measured Throughput (Mb/sec) | Iperf Measured Throughput (Mb/sec) | Theoretical Throughput (Mb/sec) | MyNETBench Efficiency (%) | Iperf Efficiency (%) |
|---|---|---|---|---|---|---|---|
| TCP | 1 | 1KB | 1821.8656 | 2010.41 | 10000 | 18.218656 | 20.1041 |
| TCP | 1 | 32KB | 5274.81 | 2163.52 | 10000 | 52.7481 | 21.6352 |
| TCP | 2 | 1KB | 198.715425 | 3562.63 | 10000 | 1.98715425 | 35.6263 |
| TCP | 2 | 32KB | 5098.866 | 4120.25 | 10000 | 50.98866 | 41.2025 |
| TCP | 4 | 1KB | 279.815425 | 4562.33 | 10000 | 2.79815425 | 45.6233 |
| TCP | 4 | 32KB | 4975.81 | 4765.22 | 10000 | 49.7581 | 47.6522 |
| TCP | 8 | 1KB | 312.388925 | 4412.32 | 10000 | 3.12388925 | 44.1232 |
| TCP | 8 | 32KB | 4375.0104 | 5120.62 | 10000 | 43.750104 | 51.2062 |
| UDP | 1 | 1KB | 364.47945 | 1262.66 | 10000 | 3.6447945 | 12.6266 |
| UDP | 1 | 32KB | 198.7197625 | 6000.35 | 10000 | 1.987197625 | 60.0035 |
| UDP | 2 | 1KB | 3148.1254 | 1432.2 | 10000 | 31.481254 | 14.322 |
| UDP | 2 | 32KB | 300.56185 | 9620.32 | 10000 | 3.0056185 | 96.2032 |
| UDP | 4 | 1KB | 3937.0604 | 3302.32 | 10000 | 39.370604 | 33.0232 |
| UDP | 4 | 32KB | 341.34975 | 8010.32 | 10000 | 3.4134975 | 80.1032 |
| UDP | 8 | 1KB | 3874.2412 | 1595.36 | 10000 | 38.742412 | 15.9536 |
| UDP | 8 | 32KB | 374.336425 | 9162.35 | 10000 | 3.74336425 | 91.6235 |

TCP/UPD throughput plot : Throughput in Mb/sec vs Concurrency at 1000 B blocksize:

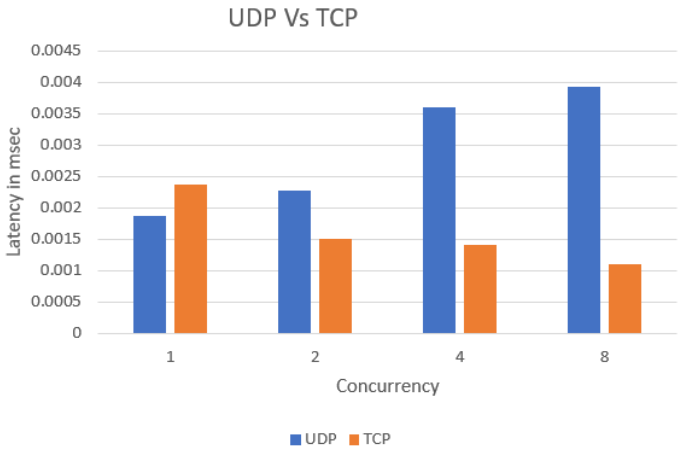



TCP/UPD throughput plot : Throughput in Mb/sec vs Concurrency at 32000 B blocksize:




As from the graph , it can be seen that TCP throughput increase with the increase in thread count but reaches a maximum limit after concurrency level 2. This is due to the sequential copy of data from client buffer to server buffer which involves mutex locks and I have used 1GB buffer at Server and Client side to store the data which is copied in chunks of 1kb or 32kb . Another reason, could be less bandwidth on the hyperion network at the time when the test was run. As UDP follows a connection less service ,there are packet drops involved when sending data.

TCP/UDP latency plot : Latency in us vs block size




It can be seen clearly that for TCP as the no of threads increases , the latency decreases with increase in concurrency. This is not always true for UDP as there is variation in the latency from 1 to 8 threads.

**Network Benchmark Metrics:**

Workload = 1GB
Throughput = ((Workload * 100) / time) / 1048576)
Theoretical Throughput TCP= 10000 Mbps
Theoretical Throughput UDP= 10000 Mbps
Throughput Efficiency = (Throughput / Theoretical Throughput) *100
Latency = (execution wall time * 1000) / 1000000 milliseconds
Theoretical Latency = 0.0007 milliseconds
Latency Efficiency = ((Theoretical Latency - Latency)/ Theoretical Latency) * 100

**Conclusion:**

This performance benchmark on TCP and UDP verifies that , better throughput can be achieved with UDP than TCP due to continuous packet stream between Server and Client with either of them not waiting for any ACK packet from the other, which is the case with TCP. This ACK of sent packets in each TCP opened window delays the transfer of next stream of packets.