Benchmarking Design document

Proposal

The objective of this assignment was to delve in the world of benchmarking the four different parts viz. CPU, memory, disk and network with an aim to understand the intricacies and have firsthand feel of the particulars, involved with getting these implemented in the real world.

Design

The overall approach for this design was to create a benchmarking module for each of the four different areas namely, CPU, Memory, Disk and Network. So, naturally this design was divided in to four parts which are mentioned further in the document.

I. CPU Benchmarking

The problem statement for this module of the benchmarking suite required stressing the CPU's FPU unit to get its throughput in terms of operations per second. This meant working with high workload which in this scenario was 1 trillion arithmetic operations to be performed and benchmarks to be calculated for the same. Since we are working with a large payload, POSIX threads were used to cater for division of work to numerous workers in a master-worker analogy.

The computation of Operations per second using a fixed number of iterations was done by using arithmetic operations for addition and subtraction in a loop, which was repeated for 1 trillion times. The total time taken then for all the operations performed in the loop is then used to compute the Giga Operation per second as per below formula:

$$Giga\ Ops/\sec = \frac{(total\ operations\ done\ in\ the\ loop)}{(total\ time\ taken) * 1000000000}$$

To calculate this computation for Single precision – integer operations per second and double precision – floating point operations per second didn't required any special handling but the same was not true for half precision – short operations per second and quarter precision – character operations per second.

Since the counters used to keep track of the iterations performed are using Integer which is of single precision hence, the total time for half precision and quarter precision also included them. To counter for this the time taken by the for loop using integer was extracted in a separate run which was then deducted from the total time calculations for half and quarter precisions before computing the Giga Ops/sec using the above formula.

Since the problem statement advised for strong scaling hence a structure with variables like begin and finish and thread ID were created to track the division of work to the threads. An array of this structure was then created to store these values as and when the division of work is done in the master thread.

II. Memory Benchmarking

In this module, the benchmarking suite required testing the throughput and latency of the RAM in GB/sec and microseconds respectively. The workload for which was 1 GB of data to be read and written, 100 times for different block sizes. Since we are again working with a large payload, POSIX threads were used to cater for division of work to numerous workers in a master-worker analogy.

As the problem statement advised for strong scaling hence the division of work to the threads is being tracked inside the function using finish variable. As read and write access for the memory were to be done sequentially and randomly, multiple memcpy and memset operations were done with the varying block sizes of 1B, 1KB, 1MB and 10MB between 2 blocks called 'workloadsizeblock' and 'blksizeblock'. Since we also had to compute the latency, by doing 100 Million operations for 1 Byte of data, hence changes were done to the for loops such that loop counter changes to 100 Million when block size of 1KB was provided as input to the program. These operations were then timed, and the total time calculated and stored to be used in the computations for throughput and latency.

Finally, the computations for calculating the throughput and latency were done by using the below formulas.

$$Thoroughput\ in\ GB/sec = \frac{(Total\ memory\ Read\ or\ Written\ to\ disk)}{(total\ time\ taken) * (1024 * 1024)}$$

$$Latency\ in\ microseconds = \frac{(Total\ time\ taken) * (1000)}{(total\ memory\ read/written)}$$

This experiment was done using sequential and random read/write patterns, handling was required for sequential and random transfers. For sequential this, handling was done by allocating (block size) * (division of work for that thread) space to workloadsizeblock and then repeated memcpy calls from workloadsizeblock to blksizeblock with block size increments with that size of the workloadsizeblock. For random, the approach was similar with the only change that the random location was calculated by taking absolute of the difference between the work division limits assigned to that thread and random number generator spewing a number between the starting and ending limits of the work assigned to the thread.

III. *Disk Benchmarking*

In this module, the benchmarking suite required testing the throughput and latency of the Disk in MB/sec and milliseconds respectively. The workload for which was 10 GB of data to be read and written for different block sizes. Since we are again working with a large payload, here too POSIX threads were used to cater for division of work to numerous workers in a master-worker analogy.

Since this benchmarking is very similar to memory benchmark, to handle for strong scaling variable like finish is created to track the division of work to the threads. Like Memory benchmarking, here too, we had to do read and writes which need to be done sequentially and randomly, this was done by using functions open, read, write, lseek, etc. for varying block sizes of 1B, 1KB, 1MB and 10MB between 2 blocks called 'workloadsizeblock' and 'blksizeblock'. Since we also had to compute the latency, by doing 1 Million operations for 1 Byte of data for 1 GB workload, hence changes were done to the for loops such that loop counter changes to 1 Million when block size of 1B was provided as input to the program. These operations were then timed, and the total time calculated and stored to be used in the computations for throughput and latency.

Finally, the computations for calculating the throughput and latency were done by using the below formulas.

$$Thoroughput\ in\ MB/sec = \frac{(Total\ memory\ transferred)}{(total\ time\ taken) * (1024 * 1024 * 1024)}$$

$$Latency\ in\ microseconds = \frac{(Total\ time\ taken) * (1000000)}{(total\ memory\ transferred)}$$

This experiment was done using sequential and random read/write patterns, handling was required for sequential and random transfers. For sequential this, handling was done by allocating (block size) * (division of work for that thread) space to workloadsizeblock and then repeated memcpy calls from workloadsizeblock to blksizeblock with block size increments with that size of the workloadsizeblock. For random, the approach was similar with the only change that the random location was calculated by taking absolute of the difference between the work division limits assigned to that thread and random number generator spewing a number between the starting and ending limits of the work assigned to the thread.

Apart from the above, since we had to ensure that direct input and output were to be used so while doing file write and read, I have used O_SYNC to do synchronous I/O to guarantee that the call does not return before all the data has been transferred to disk thereby giving as much throughput from disk as possible, within the given framework.

IV. Network Benchmarking

In this module, the benchmarking suite required testing the throughput and latency of the Network in MB/sec and milliseconds respectively. The workload for which was 1 GB of data to be read and written for different block sizes. Since we are again working with a large payload, here too POSIX threads were used to cater for division of work to numerous workers in a master-worker analogy.

Two designs were needed for the protocols UDP and TCP working on 2 instances in a client server fashion. To emulate the protocols the client and server constantly had to send data in record sizes of 1KB and 32KB the using send and recv functions in C. In order to build connections between the server and client sockets were implemented, along with getting addresses from the hostname provided by the batch script. Special checks were put in place to ensure the entire workload is sent between the server and client.

Since we also had to compute the latency, by doing 1 Million operations for 1 Byte of data, hence changes were done to the for loops such that loop counter changes to 1 Million when block size of 1B was provided as input to the program. These operations were then timed, and the total time calculated and stored to be used in the computations for throughput and latency.

Finally, the computations for calculating the throughput and latency were done by using the below formulas.

$$Thoroughput\ in\ MB/sec = \frac{(Total\ bytes\ transferred)}{(total\ time\ taken) * (1024 * 1024)}$$

$$Latency\ in\ microseconds = \frac{(Total\ time\ taken) * (1000)}{(total\ bytes\ transferred)}$$

Generic Coding highlights

- Each of the benchmark module has 2 functions, one containing the actual Benchmarking logic and the other function which is used to create and monitor threads and calls the first function.
- To generate the output for all the different testcases as summarized in the assignment problem document, we also designed bash scripts to automate the submission of batch jobs for all the modules of the benchmarking suite.
- Time duration is computed using 'struct timeval' structure as it gives the duration in seconds and microseconds.

Trade-offs made:

Generic

1. The bash scripts created to submit batch jobs are sub-optimal because Hyperion and Prometheus both had limits of 10 jobs submission per user as permissible limit at any given time.
2. The program currently isn't interactive and doesn't cater to the variable inputs from the user, although an input can be made indirectly with specific parameters by modifying existing input slurm and script files, it isn't user-friendly and complex to do so.

CPU

1. Linear equations could have been used instead of arithmetic operations alone for benchmarking the CPU.
2. Due to limited understanding on how LINPACK works, the current program doesn't have the gains as seen in LINPACK thus inferior in performance when compared with the standard benchmark.

Memory

1. Although, we are using mem* functions to do read and writes on the memory, the program doesn't handle the caching issue due to which the performance shown is not pure memory throughput.
2. Limited knowledge on handling cache issues is also one of the reason the performance is sub-optimal.

Disk

1. In this too, caching issues limit the actual performance that could have been shown by the program.
2. In this program, as the caches were not cleared hence it is a mixture of disk and memory benchmarking.

Improvements that could be done:

- Improved performance can be achieved by understanding LINPACK's internal logic and by implementing AVX and FMA instructions in the code.
- Implementing dropping of caches should be done to get better performance from the disk benchmark. A possible way was to get root access and write 3 in proc fs interface '/proc/sys/vm/drop_caches' to drop both inode and page caches.

Performance report

This part of the document presents the performance evaluation of the benchmark suite.

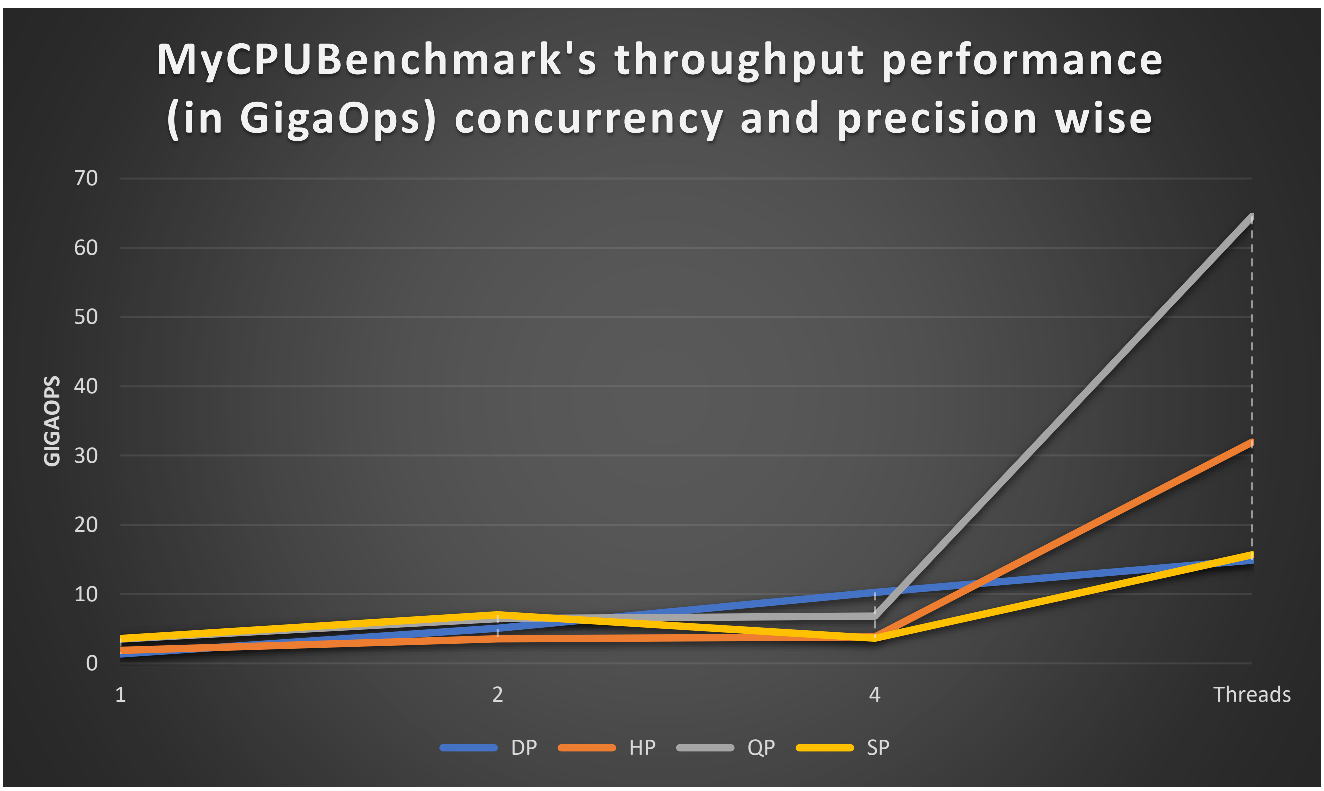
All the performance metrics were obtained from Hyperion cluster.

1. **CPU Benchmark:**
   This benchmark was done on Hyperion cluster which has the below specifications:

Model Name	Intel(R) Xeon(R) CPU E5-2670 v3
CPU MHz	2.30 GHz
Cache Size	256.00 KiB
CPU Cores (vCore)	1

To show case the performance of the benchmarks below visuals are used along with attached data.

The below graph visually showcases the gradual increase in the performance of the benchmarks, as we go from threads 1 to 4, with the lower precisions like Quarter and half showcasing a quadratic increase in the Giga operations per second given by them.

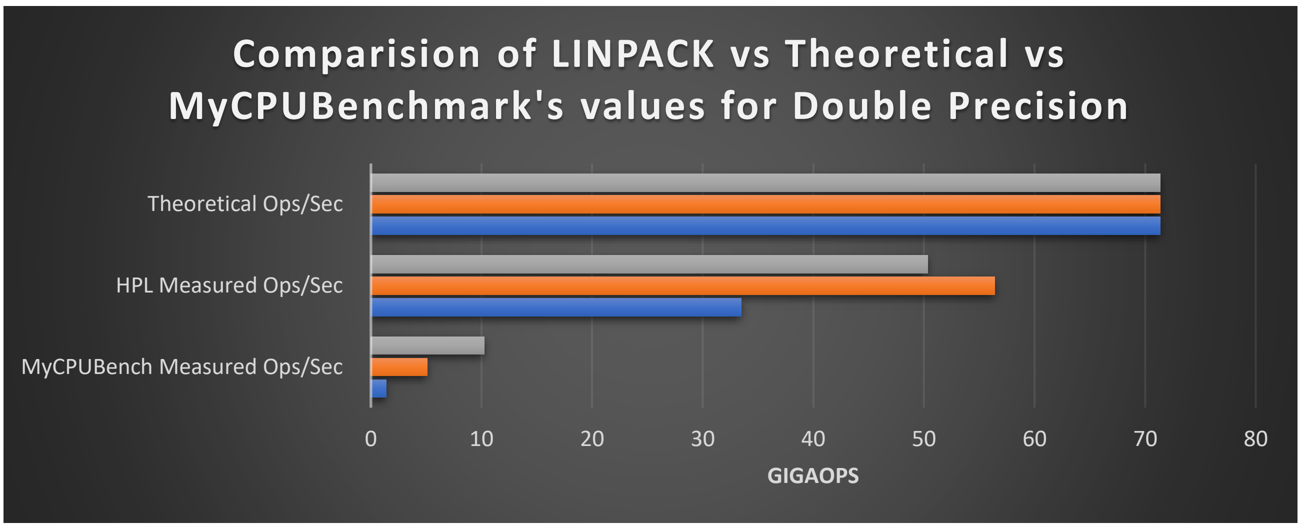


Furthermore, the almost linear increase in the Double precision showcases that the it is the least impacted by the increase in number of threads. I would say this is partly due to the fact that at lower thread count the overhead of working with lower precision is more which increases subtly but soon shows quadratic behavior as now it breaks even with the overheads mentioned previously.

So, in this scenario, the case of having increased efficiency and throughputs due to increase in number of threads is seen in the lower precisions like Quarter & Half precisions, whereas the opposite is true for Single and Double precision as they are now hitting the maximum the CPU’s FPU can handle.

Below graph showcases the performance of CPU benchmark designed in this assignment against LINPACK, and Theoretical values for Double precision. Obviously, the designed benchmark is low when compared to the other two values.

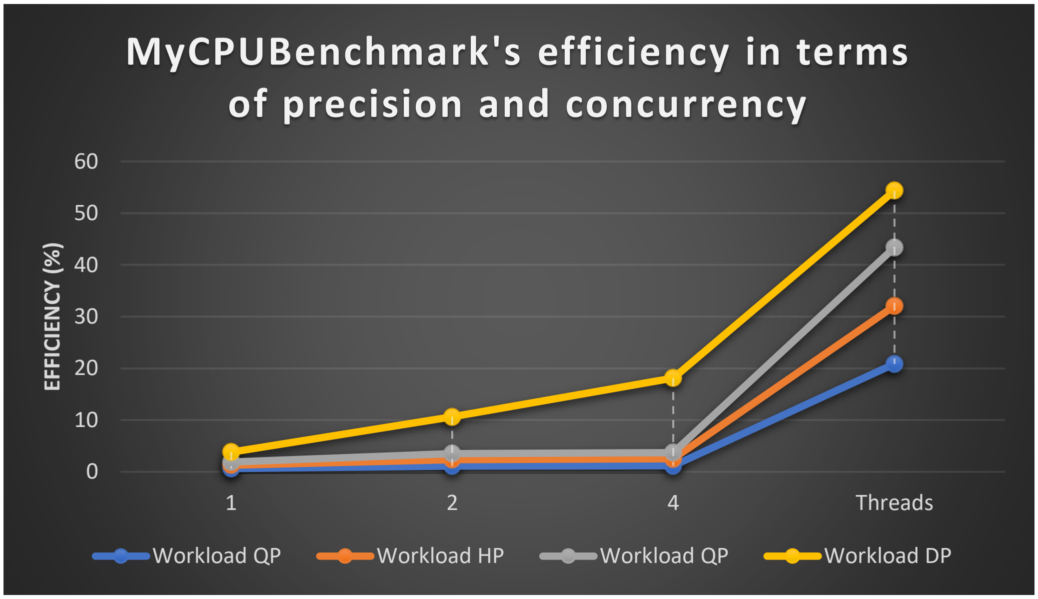
This is due to normal arithmetic operations of addition and multiplication are by themselves not enough to actually stress the CPU’s Floating-point unit to give its maximum performance. Using AVX, FMA instructions while solving matrix multiplications will surely give the throughput as expected in the Linpack benchmark.



The above chart also showcases how close Linpack actual performance is when compared with the theoretical values.

When, I ran the Linpack benchmarks using thread concurrency at 1,2 and 4 with the maximum value of 70 GFLOPS seen with 4 threads for double precision which is much closer to the theoretical value than my program. Due to the issues faced with clusters at both Prometheus and Hyperion I ran the Linpack via slurm jobs rather than the interactive compute nodes as the cluster was slow and running benchmarks on the compute node was becoming quite troublesome.

The below chart expands on the efficiency when compared with precision and concurrency of threads, showcasing how efficient the program is when dealing with multiple precision and number of threads.



Below table lists the thoroughput values in GigaOps/sec for the CPU benchmarks based on the experiments done with the benchmark implemented along with the standard deviation for the 3 times the experiment was repeated.

These throughput value are calculated using the below formulas mentioned in the design document.

GigaOps/sec Throughput values

Workload	Concurrency	MyCPUBench Measured Ops/Sec (GigaOPS)	HPL Measurerd Ops/Sec (GigaOPS)	Theoritical Ops/Sec (GigaOPS)	MyCPU Bench Efficiency(%)	HPL Efficiency(%)	Standarad Deviation
QP	1	3.520475333	NA	588.8	0.597906816	NA	0.024341295
QP	2	6.484087333	NA	588.8	1.101237659	NA	0.141475038
QP	4	6.843448	NA	588.8	1.16227038	NA	0.05388375
HP	1	1.880600667	NA	294.4	0.638790987	NA	0.05460391
HP	2	3.532577	NA	294.4	1.199924253	NA	0.070374278
HP	4	3.805881	NA	294.4	1.292758492	NA	0.026022483
SP	1	3.594858333	NA	147.2	2.442159194	NA	0.1603078
SP	2	7.010844333	NA	147.2	4.762801857	NA	0.353108397

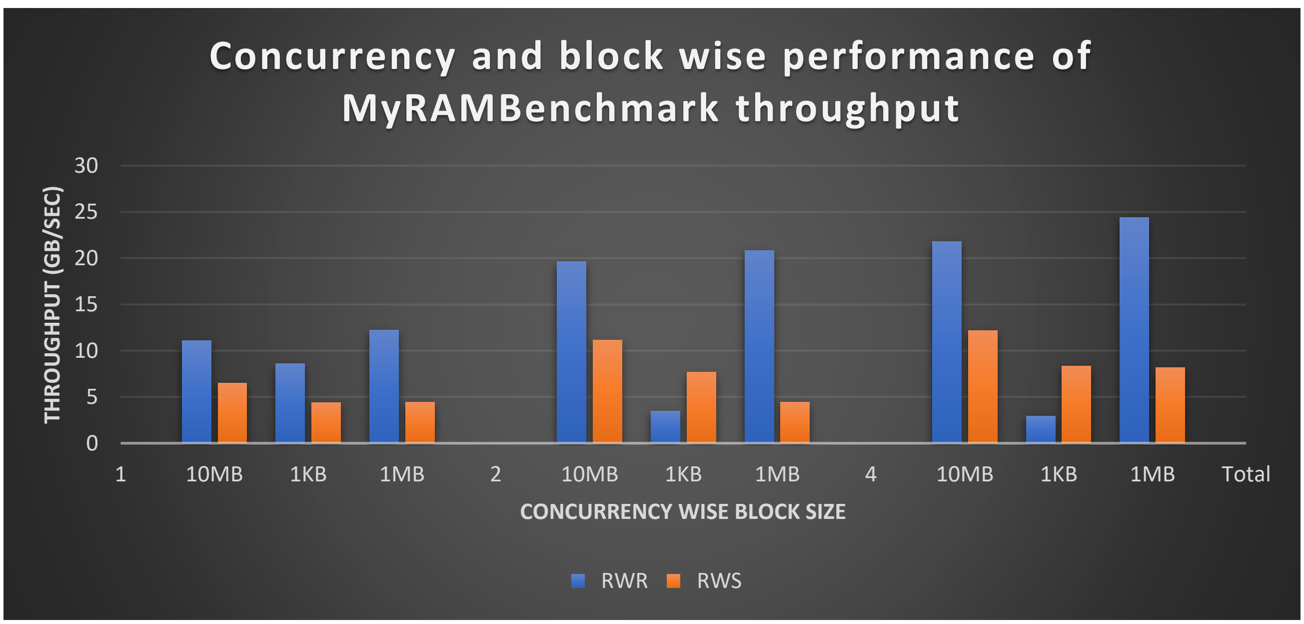
SP	4	3.641644667	NA	147.2	2.473943388	NA	0.068509091
DP	1	1.364665333	38.1844	73.6	1.854164855	51.88097826	0.038020727
DP	2	5.075362	73.4327	73.6	6.895872283	99.77269022	0.198504433
DP	4	10.26228033	70.1092	73.6	13.94331567	95.25706522	0.403845808

2. Memory Benchmark

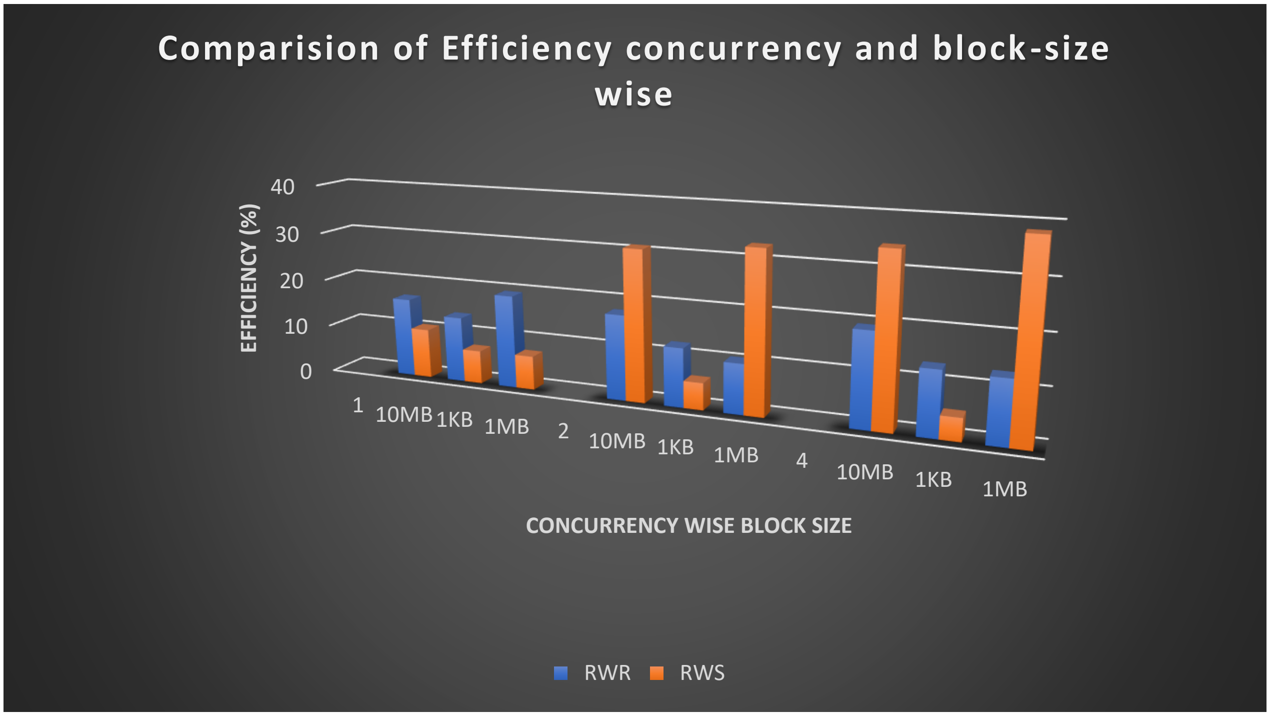
The Memory benchmark was also done on the Hyperion cluster which has the 2133 MHz 12GB RAM sticks of which only 4GB are available to the user or node.

The below visual aids depict the performance and trends seen in this part of the assignment.

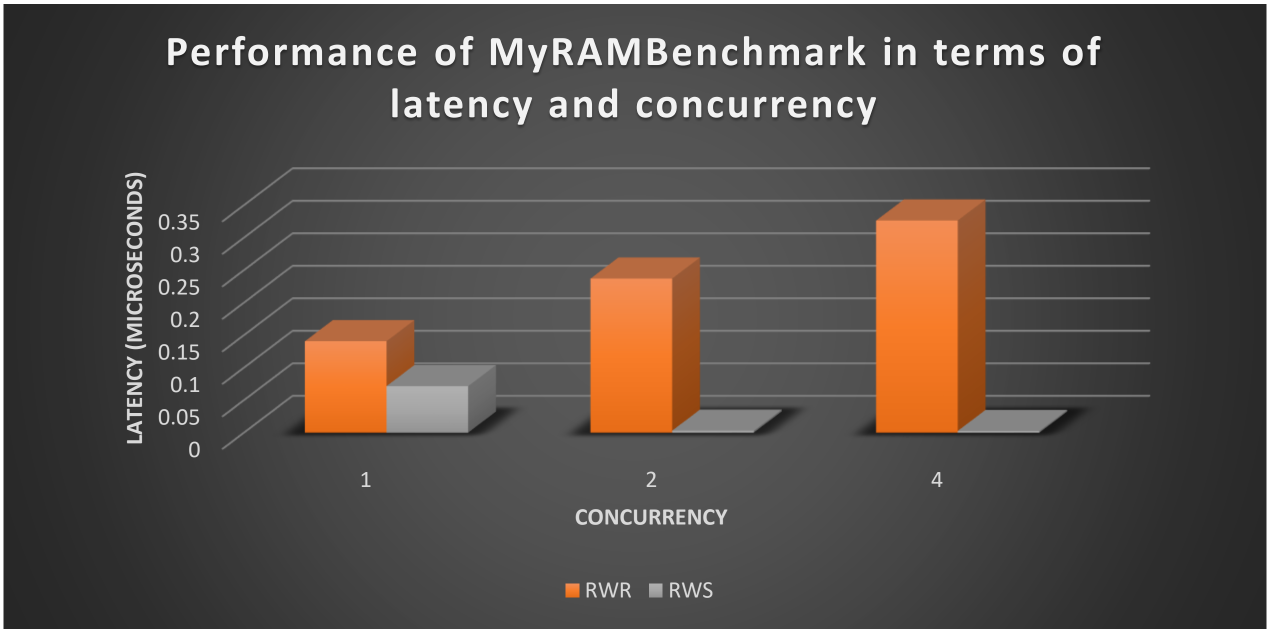
The below graph illustrates the throughput performance of Random Read and Writes as well as Sequential Read and Writes done to the memory, concurrency and block wise. As seen below the Random read and writes showcases the best performance for 1MB block size for 4 threads.



The next graph illustrates the Efficiency of the RAM benchmarking when seen mapped against the concurrency and block sizes. Here too , the best efficiency is seen at 1 MB block size and 4 threads.



Lastly the below graph showcases the performance of the MyRAMBenchmark in terms of latency and concurrency for both Random and Sequential read writes.



Below are the outputs from the benchmarking experiments

Throughput results:

Work-Load	Con-currency	Block Size	MyRAMBench Measured Throughput (GB/sec)	pmbw Measured Throughput(GB/sec)	Theoretical Throughput (GB/sec)	MyRAMBench Efficiency(%)	pmbw Efficiency (%)	Standard Deviation
RWS	1	1KB	4.379744333	16.755738	63.568354	6.889819333	26.35861566	0.08100662
RWS	2	1KB	7.696003	16.161646	63.568354	12.10665767	25.42404279	0.14406986
RWS	4	1KB	8.359899333	14.842147	63.568354	13.15103967	23.34832747	0.12667387
RWS	1	1MB	4.451564667	27.834186	63.568354	7.0028	43.78623111	0.13976822
RWS	2	1MB	4.456042333	29.459142	63.568354	10.276272	46.3424646	0.26543761
RWS	4	1MB	8.177533333	28.464898	63.568354	12.86415833	44.77840945	0.21775537
RWS	1	10MB	6.522010333	34.55994	63.568354	10.259838	54.36658039	1.17521413
RWS	2	10MB	11.13223033	34.273696	63.568354	17.512221	53.9162865	0.59632099
RWS	4	10MB	12.17623167	30.645449	63.568354	19.15454933	48.20865651	2.67423325
RWR	1	1KB	8.627884333	12.843576	63.568354	13.57260933	20.20435542	0.19058895
RWR	2	1KB	3.495027	5.4529531	63.568354	5.542485333	8.578093959	0.01193508
RWR	4	1KB	2.938921667	6.2556131	63.568354	4.623246333	9.840766152	0.09909288
RWR	1	1MB	12.22489533	9.2620952	63.568354	19.231103	14.57029263	0.17266233
RWR	2	1MB	20.84701033	25.186064	63.568354	32.794636	39.62044332	0.59780384
RWR	4	1MB	24.43574333	30.584671	63.568354	38.440107	48.1130461	0.8137159
RWR	1	10MB	11.11102567	23.172895	63.568354	16.407398	36.45350823	1.07384184
RWR	2	10MB	19.624958	32.868451	63.568354	30.87221367	51.70568225	1.65886957
RWR	4	10MB	21.83599367	34.683363	63.568354	34.35041567	54.56073853	1.27941268

Latency results:

Work-load	Con-currency	Block Size	MyRAMBench Measured Latency(us)	pmbw Measured Latency (us)	Theoritical Latency (us)	MyRAMBench Efficiency (%)	PMBW Efficiency (%)	Standard Deviation
RWS	1	1B	0.007177333	0.006319	0.01406	51.049992	44.943101	0.00029666
RWS	2	1B	0.003247667	0.003148	0.01406	23.09752967	22.38975818	0.00019356
RWS	4	1B	0.002832	0.001874	0.01406	20.14460433	13.32859175	0.00013945
RWR	1	1B	0.140941	0.143937	0.01406	9.715076333	1023.733997	0.00563812
RWR	2	1B	0.237287333	0.067617	0.01406	16.87674933	480.9174964	0.00670277
RWR	4	1B	0.326609333	0.033241	0.01406	23.22968267	236.4224751	0.00697526

Similar to Linpack, the PMBW benchmarks were submitted by me via slurm jobs the

3. Disk Benchmark

The Memory benchmark was also done on the Hyperion cluster which has the Seagate Constellation 2 SATA Hard drives. The detailed specifications for the same can be found at the below link:

https://www.seagate.com/files/www-content/product-content/constellation-fam/constellation/constellation-2/en-us/docs/constellation2-fips-ds1719-4-1207us.pdf

The below table showcases the results obtained from running the benchmark on Hyperion cluster.

Thoroughput results:

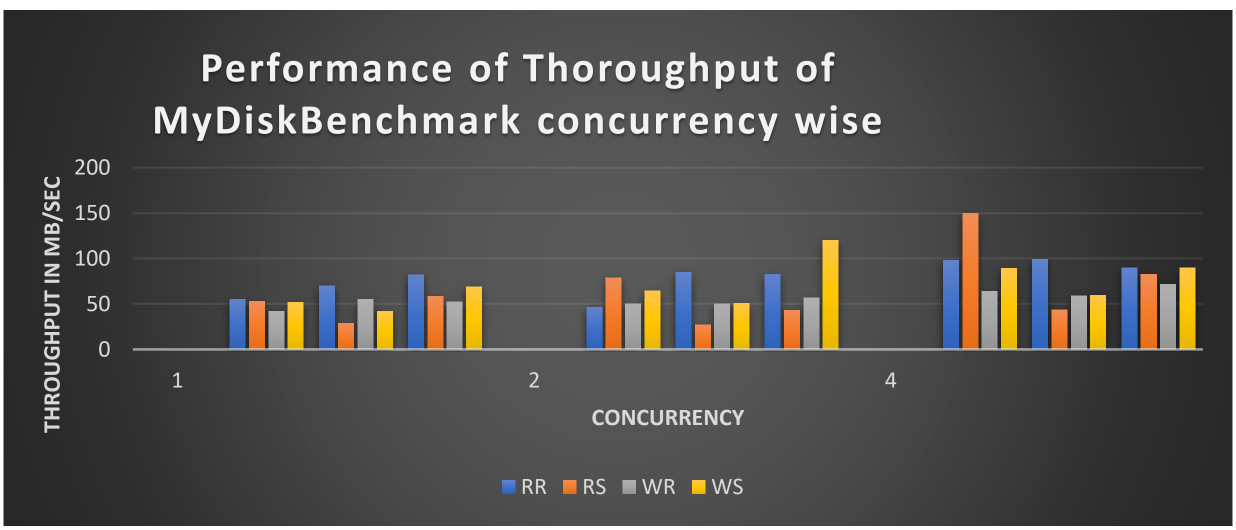
Work-load	Con-currency	Block Size	MyDisk Bench Measured Throughput(MB/sec)	IOZoneMeasured Throughput (MB/sec)	Theoretical Throughput (MB/sec)	MyDiskBench Efficiency (%)	IOZone Efficiency(%)	Standard Deviation
RR	1	1MB	53.75619	313.18	600	8.959365	52.19667	11.37474
RR	2	1MB	44.72444	353.18	600	7.454074	58.86333	37.21378
RR	4	1MB	96.50222	303.18	600	16.0837	50.53	17.50689
RR	1	10MB	68.39111	526.36	600	11.39852	87.72667	7.653445

RR	2	10MB	83.05778	406.36	600	13.84296	67.72667	41.68933
RR	4	10MB	97.73788	566.36	600	16.28965	94.39333	342.6706
RR	1	100MB	80.67037	452.72	600	13.44506	75.45333	187.492
RR	2	100MB	81.16889	332.72	600	13.52815	55.45333	342.6706
RR	4	100MB	88.33051	532.72	600	14.72175	88.78667	42.58838
RS	1	1MB	51.42628	310.52	600	8.571047	51.75333	1.542875
RS	2	1MB	77.32781	325.52	600	12.88797	54.25333	13.78345
RS	4	1MB	148.3362	301.52	600	24.7227	50.25333	32.19568
RS	1	10MB	27.11404	321.04	600	4.519007	53.50667	19.32195
RS	2	10MB	25.6744	501.04	600	4.279067	83.50667	0.121074
RS	4	10MB	41.83216	561.04	600	6.972027	93.50667	11.18196
RS	1	100MB	56.6192	442.08	600	9.436534	73.68	11.69876
RS	2	100MB	41.56502	302.08	600	6.927504	50.34667	211.402
RS	4	100MB	81.05684	302.08	600	13.50947	50.34667	10.08252
WR	1	1MB	40.54739	284.25	600	6.757899	47.375	0.520862
WR	2	1MB	48.39925	219.25	600	8.066541	36.54167	0.28584
WR	4	1MB	62.19073	184.25	600	10.36512	30.70833	3.118532
WR	1	10MB	53.3278	598.5	600	8.887967	99.75	4.133656
WR	2	10MB	48.31124	468.5	600	8.051874	78.08333	4.133656
WR	4	10MB	57.36965	588.5	600	9.561609	98.08333	4.133656
WR	1	100MB	50.65512	426	600	8.442521	71	1.050117
WR	2	100MB	55.22669	367	600	9.204448	61.16667	8.090561
WR	4	100MB	70.09879	483	600	11.68313	80.5	23.20778

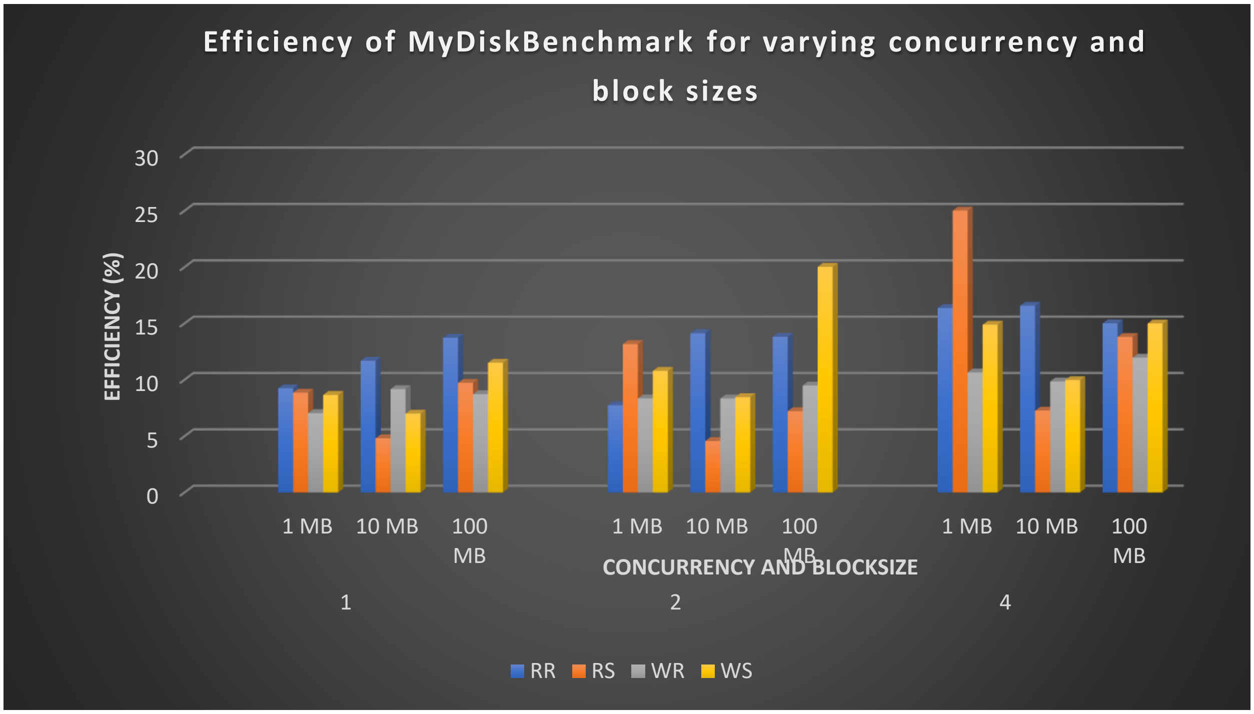
WS	1	1MB	50.21623	185.63	600	8.369371	30.93833	1.083086
WS	2	1MB	63.03881	195.63	600	10.50647	32.605	3.951009
WS	4	1MB	87.61114	105.63	600	14.60186	17.605	1.202421
WS	1	10MB	40.25868	371.26	600	6.70978	61.87667	3.214788
WS	2	10MB	49.0156	351.26	600	8.169267	58.54333	1.020333
WS	4	10MB	58.10763	251.26	600	9.684605	41.87667	1.591361
WS	1	100MB	67.33454	410.469	600	11.22242	68.4115	3.449329
WS	2	100MB	118.4261	562.52	600	19.73768	93.75333	15.78488
WS	4	100MB	88.23094	540.458	600	14.70516	90.07633	3.317427

The below visual aids showcase the performance of the Disk benchmark implemented in the assignment.

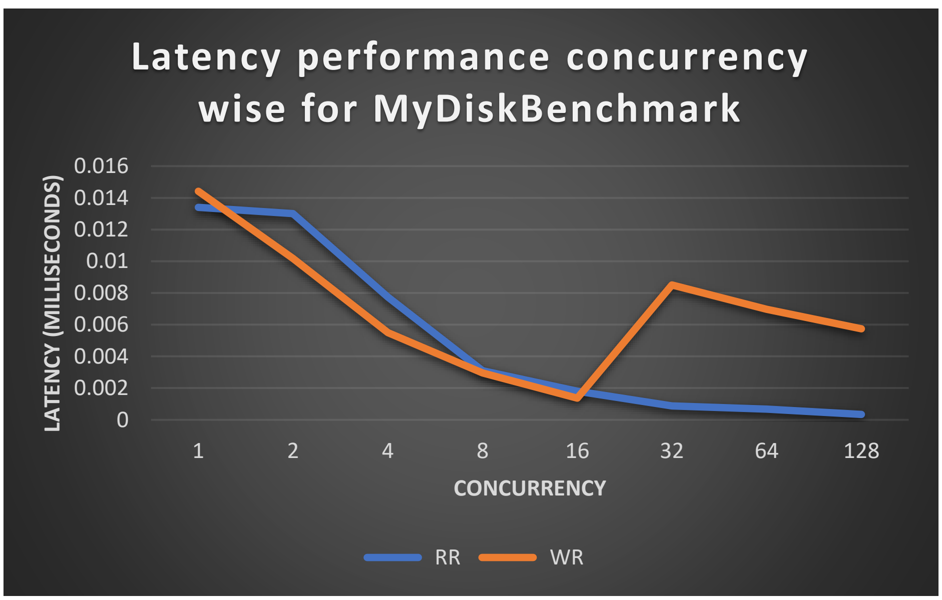
The below graph showcases the performance of the benchmark concurrency wise, implying the obvious that the parallelization of the work for read and write improves the performance as shown by the blocks for group of 4 threads in the below chart.



This is futher enhanced when comparing the efficiency as seen the graph below. This graph also showcase the same result that the increase in the number of threads



The below graph, illustrates the latency performance for the current benchmark for varying number of threads, the data for which was taken from the table mentioned below:



This is calculated using the below formula:

Theoretical value calculation:

$$IOPS = \frac{1}{average\ seek\ time + average\ latency}$$

Here for the specification in Hyperion the calculation using the values is as follows:

$$average\ seek\ time\ (read/write) = \frac{0.0085}{0.0095} ms$$
$$average\ latency = 4.16ms$$

This gives us IOPS as, $IOPS = 1/\,(average\ seek\ time + average\ latency)$

So we have,

$$IOPS = 5782.4991$$

Latency values:

Work-load	Con-currency	Block Size	MyDiskBench Measured Latency(ms)	IOZone Measured Latency(ms)	Theoretical Latency(ms)	MyDiskBench Efficiency(%)	IOZone Efficiency(%)	StandardDeviation
RR	1	1KB	0.0124	0.0217	4.16	0.002981	0.528846	0.001642
	2	1KB	0.0144	0.0273	4.16	0.003462	0.663462	0.000945
	4	1KB	0.00722	0.0347	4.16	0.001736	0.841346	0.000357
	8	1KB	0.00298	0.0264	4.16	0.000716	0.641827	0.0002
	16	1KB	0.00167	0.0067	4.16	0.000401	0.168269	0.00006
	32	1KB	0.00083	0.0357	4.16	0.0002	0.865385	5.77E-05
	64	1KB	0.00068	0.0167	4.16	0.000163	0.408654	5.78E-05
	128	1KB	0.00037	0.0777	4.16	8.89E-05	1.875	0.002136
WR	1	1KB	0.0143	0.1557	4.16	0.003438	3.75	0.021414
	2	1KB	0.01018	0.0377	4.16	0.002447	0.913462	0.00089
	4	1KB	0.004788	0.3637	4.16	0.001151	8.75	0.002906
	8	1KB	0.002934	0.0257	4.16	0.000705	0.625	0.001378
	16	1KB	0.001388	0.0453	4.16	0.000334	1.096154	0.000524
	32	1KB	0.008667	0.1057	4.16	0.002083	2.548077	5.83E-05
	64	1KB	0.006987	0.0553	4.16	0.00168	1.336538	5.09E-05
	128	1KB	0.005454	0.0406	4.16	0.001311	0.983173	0.000151

Apart from the above, the IOPS values were also calculated and are presented as below:

Work-load	Con-currency	Block Size	MyDisk Bench Measured IOPS	IOZone Measured IOPS	Theoretical IOPS	MyDiskBench Efficiency(%)	IOZone Efficiency(%)	Standard Deviation
RR	1	1KB	4466.33	5065.118	5782.499	77.23875	87.59392	11.8781
RR	2	1KB	4627.106	5241.118	5782.499	80.01914	90.63759	7.985152
RR	4	1KB	919.725	3432.645	5782.499	15.90532	59.36265	11.60871

RR	8	1KB	2838.596	3332.578	5782.499	49.08943	57.63214	28.8272
RR	16	1KB	4403.485	4639.175	5782.499	76.15193	80.22785	58.51214
RR	32	1KB	3993.699	4608.389	5782.499	69.06528	79.69545	54.69597
RR	64	1KB	1224.626	2308.116	5782.499	21.17814	39.91555	143.8018
RR	128	1KB	1740.592	3730.642	5782.499	30.10103	64.51609	3.56E-13
WR	1	1KB	331.0262	1330.716	5782.499	5.724622	23.01282	0.659237
WR	2	1KB	602.3134	3302.003	5782.499	10.41614	57.1034	0.120131
WR	4	1KB	1423.922	2223.567	5782.499	24.62468	38.45339	0.715482
WR	8	1KB	2952.676	4310.375	5782.499	51.06229	74.54174	1.726807
WR	16	1KB	3916.505	4716.176	5782.499	67.73031	81.55947	4.810556
WR	32	1KB	793.8901	3163.355	5782.499	13.72919	54.70568	6.000631
WR	64	1KB	1051.233	1989.006	5782.499	18.17956	34.39699	8.233895
WR	128	1KB	1354.593	2430.588	5782.499	23.42573	42.03352	16.19557

IOZONE was the benchmark for benchmarking Disk below are some screenshots of the same:

dkaramchandani@compute-3: /tmp

```
             Alexey Skidanov.

        Run began: Tue Mar 27 13:59:06 2018

        File size set to 10485760 kB
        Record Size 1024 kB
        SYNC Mode.
        Include fsync in write timing
        O_DIRECT feature enabled
        Command line used: /usr/bin/iozone -s 10g -r 1m -o -l2 -u2 -i 0 -i 1 -i 2 -e -I -F -f /tmp/iozonetmp1_d.txt /tmp/iozonetmp2_d.txt
        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.
        Min process = 2
        Max process = 2
        Throughput test with 2 processes
        Each process writes a 10485760 kByte file in 1024 kByte records

        Children see throughput for  2 initial writers  =  284463.17 kB/sec
        Parent sees throughput for  2 initial writers   =  284421.49 kB/sec
        Min throughput per process                      =  142207.39 kB/sec
        Max throughput per process                      =  142255.78 kB/sec
        Avg throughput per process                      =  142231.59 kB/sec
        Min xfer                                        = 10482688.00 kB

        Children see throughput for  2 rewriters        =  310301.44 kB/sec
        Parent sees throughput for  2 rewriters         =  310283.04 kB/sec
        Min throughput per process                      =  155014.59 kB/sec
        Max throughput per process                      =  155286.84 kB/sec
        Avg throughput per process                      =  155150.72 kB/sec
        Min xfer                                        = 10468352.00 kB

        Children see throughput for  2 readers          =  590382.09 kB/sec
        Parent sees throughput for  2 readers           =  590318.54 kB/sec
        Min throughput per process                      =  292360.03 kB/sec
        Max throughput per process                      =  298022.06 kB/sec
        Avg throughput per process                      =  295191.05 kB/sec
        Min xfer                                        = 10286080.00 kB

        Children see throughput for 2 re-readers        =  590659.00 kB/sec
        Parent sees throughput for 2 re-readers         =  590642.40 kB/sec
        Min throughput per process                      =  293761.91 kB/sec
```

dkaramchandani@compute-3: /tmp

```
        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.
        Min process = 2
        Max process = 2
        Throughput test with 2 processes
        Each process writes a 10485760 kByte file in 1024 kByte records

        Children see throughput for  2 initial writers  =  284463.17 kB/sec
        Parent sees throughput for  2 initial writers   =  284421.49 kB/sec
        Min throughput per process                      =  142207.39 kB/sec
        Max throughput per process                      =  142255.78 kB/sec
        Avg throughput per process                      =  142231.59 kB/sec
        Min xfer                                        = 10482688.00 kB

        Children see throughput for  2 rewriters        =  310301.44 kB/sec
        Parent sees throughput for  2 rewriters         =  310283.04 kB/sec
        Min throughput per process                      =  155014.59 kB/sec
        Max throughput per process                      =  155286.84 kB/sec
        Avg throughput per process                      =  155150.72 kB/sec
        Min xfer                                        = 10468352.00 kB

        Children see throughput for  2 readers          =  590382.09 kB/sec
        Parent sees throughput for  2 readers           =  590318.54 kB/sec
        Min throughput per process                      =  292360.03 kB/sec
        Max throughput per process                      =  298022.06 kB/sec
        Avg throughput per process                      =  295191.05 kB/sec
        Min xfer                                        = 10286080.00 kB

        Children see throughput for 2 re-readers        =  590659.00 kB/sec
        Parent sees throughput for 2 re-readers         =  590642.40 kB/sec
        Min throughput per process                      =  293761.91 kB/sec
        Max throughput per process                      =  296897.09 kB/sec
        Avg throughput per process                      =  295329.50 kB/sec
        Min xfer                                        = 10375168.00 kB

        Children see throughput for 2 random readers    =  506618.88 kB/sec
        Parent sees throughput for 2 random readers     =  506590.59 kB/sec
        Min throughput per process                      =  253298.06 kB/sec
        Max throughput per process                      =  253320.81 kB/sec
        Avg throughput per process                      =  253309.44 kB/sec
        Min xfer                                        = 10484736.00 kB
```

dkaramchandani@compute-3: /tmp

```
        Min throughput per process                      =  142207.39 kB/sec
        Max throughput per process                      =  142255.78 kB/sec
        Avg throughput per process                      =  142231.59 kB/sec
        Min xfer                                        = 10482688.00 kB

        Children see throughput for  2 rewriters        =  310301.44 kB/sec
        Parent sees throughput for  2 rewriters         =  310283.04 kB/sec
        Min throughput per process                      =  155014.59 kB/sec
        Max throughput per process                      =  155286.84 kB/sec
        Avg throughput per process                      =  155150.72 kB/sec
        Min xfer                                        = 10468352.00 kB

        Children see throughput for  2 readers          =  590382.09 kB/sec
        Parent sees throughput for  2 readers           =  590318.54 kB/sec
        Min throughput per process                      =  292360.03 kB/sec
        Max throughput per process                      =  298022.06 kB/sec
        Avg throughput per process                      =  295191.05 kB/sec
        Min xfer                                        = 10286080.00 kB

        Children see throughput for 2 re-readers        =  590659.00 kB/sec
        Parent sees throughput for 2 re-readers         =  590642.40 kB/sec
        Min throughput per process                      =  293761.91 kB/sec
        Max throughput per process                      =  296897.09 kB/sec
        Avg throughput per process                      =  295329.50 kB/sec
        Min xfer                                        = 10375168.00 kB

        Children see throughput for 2 random readers    =  506618.88 kB/sec
        Parent sees throughput for 2 random readers     =  506590.59 kB/sec
        Min throughput per process                      =  253298.06 kB/sec
        Max throughput per process                      =  253320.81 kB/sec
        Avg throughput per process                      =  253309.44 kB/sec
        Min xfer                                        = 10484736.00 kB

        Children see throughput for 2 random writers    =  283166.94 kB/sec
        Parent sees throughput for 2 random writers     =  283140.80 kB/sec
        Min throughput per process                      =  141574.30 kB/sec
        Max throughput per process                      =  141592.64 kB/sec
        Avg throughput per process                      =  141583.47 kB/sec
        Min xfer                                        = 10484736.00 kB


iozone test complete.
dkaramchandani@compute-3:/tmp$
```

Network:

I wasn't able to fully implement the socket programming bits for TCP and UDP protocols mainly due to the unstable cluster performance all over the past couple of days. The situation was really bad when the supporting cluster (Prometheus in this case) also failed to curb the growing outages faced by the students.

So, I haven't attached my network codes but I did get the benchmark values for TCP and UDP using Iper3 some of the screenshots for the same are as below:

Iperf3 screenshots:

1 thread

```
dkaramchandani@hyperionides:~$ iperf3 -c 192.168.9.67 -P 1
Connecting to host 192.168.9.67, port 5201
[  4] local 192.168.27.155 port 58320 connected to 192.168.9.67 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   165 MBytes  1.39 Gbits/sec   41    732 KBytes
[  4]   1.00-2.00   sec   175 MBytes  1.47 Gbits/sec   12    728 KBytes
[  4]   2.00-3.00   sec   176 MBytes  1.48 Gbits/sec    8    707 KBytes
[  4]   3.00-4.00   sec   176 MBytes  1.48 Gbits/sec    9    686 KBytes
[  4]   4.00-5.00   sec   175 MBytes  1.47 Gbits/sec   10    666 KBytes
[  4]   5.00-6.00   sec   176 MBytes  1.48 Gbits/sec    7    655 KBytes
[  4]   6.00-7.00   sec   178 MBytes  1.49 Gbits/sec   13    648 KBytes
[  4]   7.00-8.00   sec   176 MBytes  1.48 Gbits/sec    2    635 KBytes
[  4]   8.00-9.00   sec   175 MBytes  1.47 Gbits/sec   29    615 KBytes
[  4]   9.00-10.00  sec   175 MBytes  1.47 Gbits/sec    8    605 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  1.71 GBytes  1.47 Gbits/sec  139             sender
[  4]   0.00-10.00  sec  1.70 GBytes  1.46 Gbits/sec                  receiver

iperf Done.
```

2 threads

```
dkaramchandani@hyperionides:~$ iperf3 -c 192.168.9.67 -P 2
Connecting to host 192.168.9.67, port 5201
[  4] local 192.168.27.155 port 58614 connected to 192.168.9.67 port 5201
[  6] local 192.168.27.155 port 58616 connected to 192.168.9.67 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   132 MBytes  1.11 Gbits/sec   50    583 KBytes
[  6]   0.00-1.00   sec   102 MBytes   856 Mbits/sec   10    288 KBytes
[SUM]   0.00-1.00   sec   234 MBytes  1.97 Gbits/sec   60
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   1.00-2.00   sec   142 MBytes  1.19 Gbits/sec   34    460 KBytes
[  6]   1.00-2.00   sec   100 MBytes   840 Mbits/sec   18    355 KBytes
[SUM]   1.00-2.00   sec   242 MBytes  2.03 Gbits/sec   52
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   2.00-3.00   sec   128 MBytes  1.07 Gbits/sec   28    406 KBytes
[  6]   2.00-3.00   sec   120 MBytes  1.01 Gbits/sec   15    335 KBytes
[SUM]   2.00-3.00   sec   247 MBytes  2.08 Gbits/sec   43
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   3.00-4.00   sec   120 MBytes  1.01 Gbits/sec   27    379 KBytes
[  6]   3.00-4.00   sec   120 MBytes  1.01 Gbits/sec   12    472 KBytes
[SUM]   3.00-4.00   sec   240 MBytes  2.01 Gbits/sec   39
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   4.00-5.00   sec   118 MBytes   986 Mbits/sec   26    352 KBytes
[  6]   4.00-5.00   sec   141 MBytes  1.18 Gbits/sec   49    533 KBytes
[SUM]   4.00-5.00   sec   258 MBytes  2.17 Gbits/sec   75
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   5.00-6.00   sec   108 MBytes   902 Mbits/sec    4    385 KBytes
[  6]   5.00-6.00   sec   132 MBytes  1.11 Gbits/sec   38    462 KBytes
[SUM]   5.00-6.00   sec   239 MBytes  2.01 Gbits/sec   42
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   6.00-7.00   sec   110 MBytes   923 Mbits/sec   38    373 KBytes
[  6]   6.00-7.00   sec   138 MBytes  1.16 Gbits/sec   12    508 KBytes
[SUM]   6.00-7.00   sec   248 MBytes  2.08 Gbits/sec   50
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   7.00-8.00   sec   125 MBytes  1.05 Gbits/sec    7    475 KBytes
[  6]   7.00-8.00   sec   123 MBytes  1.03 Gbits/sec   20    451 KBytes
[SUM]   7.00-8.00   sec   248 MBytes  2.08 Gbits/sec   27
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   8.00-9.00   sec   112 MBytes   944 Mbits/sec   53    308 KBytes
[  6]   8.00-9.00   sec   133 MBytes  1.12 Gbits/sec   17    390 KBytes
[SUM]   8.00-9.00   sec   246 MBytes  2.06 Gbits/sec   70
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   9.00-10.00  sec   116 MBytes   975 Mbits/sec    5    413 KBytes
[  6]   9.00-10.00  sec   134 MBytes  1.13 Gbits/sec   39    462 KBytes
[SUM]   9.00-10.00  sec   251 MBytes  2.10 Gbits/sec   44
```

4 threads

dkaramchandani@hyperionides: ~

```
[SUM]   7.00-8.00   sec   311 MBytes  2.61 Gbits/sec  635
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   8.00-9.00   sec  40.8 MBytes   342 Mbits/sec   89    158 KBytes
[  6]   8.00-9.00   sec  41.2 MBytes   346 Mbits/sec  130   96.2 KBytes
[  8]   8.00-9.00   sec  33.8 MBytes   284 Mbits/sec   89    103 KBytes
[ 10]   8.00-9.00   sec  41.8 MBytes   351 Mbits/sec   34    165 KBytes
[ 12]   8.00-9.00   sec  32.5 MBytes   273 Mbits/sec   51    129 KBytes
[ 14]   8.00-9.00   sec  48.5 MBytes   407 Mbits/sec   55    158 KBytes
[ 16]   8.00-9.00   sec  30.9 MBytes   260 Mbits/sec   73    102 KBytes
[ 18]   8.00-9.00   sec  45.6 MBytes   383 Mbits/sec   59    161 KBytes
[SUM]   8.00-9.00   sec   315 MBytes  2.64 Gbits/sec  580
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   9.00-10.00  sec  45.9 MBytes   385 Mbits/sec   39    240 KBytes
[  6]   9.00-10.00  sec  39.5 MBytes   331 Mbits/sec  119    102 KBytes
[  8]   9.00-10.00  sec  27.5 MBytes   231 Mbits/sec  183    174 KBytes
[ 10]   9.00-10.00  sec  28.8 MBytes   241 Mbits/sec   80    123 KBytes
[ 12]   9.00-10.00  sec  28.8 MBytes   241 Mbits/sec  102   84.8 KBytes
[ 14]   9.00-10.00  sec  44.6 MBytes   374 Mbits/sec   87    102 KBytes
[ 16]   9.00-10.00  sec  35.7 MBytes   299 Mbits/sec   63    148 KBytes
[ 18]   9.00-10.00  sec  54.3 MBytes   456 Mbits/sec   43    136 KBytes
[SUM]   9.00-10.00  sec   305 MBytes  2.56 Gbits/sec  716
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec   413 MBytes   347 Mbits/sec  684             sender
[  4]   0.00-10.00  sec   412 MBytes   346 Mbits/sec                  receiver
[  6]   0.00-10.00  sec   376 MBytes   315 Mbits/sec  877             sender
[  6]   0.00-10.00  sec   375 MBytes   314 Mbits/sec                  receiver
[  8]   0.00-10.00  sec   388 MBytes   326 Mbits/sec  993             sender
[  8]   0.00-10.00  sec   387 MBytes   325 Mbits/sec                  receiver
[ 10]   0.00-10.00  sec   380 MBytes   319 Mbits/sec  816             sender
[ 10]   0.00-10.00  sec   379 MBytes   318 Mbits/sec                  receiver
[ 12]   0.00-10.00  sec   361 MBytes   303 Mbits/sec  988             sender
[ 12]   0.00-10.00  sec   360 MBytes   302 Mbits/sec                  receiver
[ 14]   0.00-10.00  sec   371 MBytes   311 Mbits/sec  1019             sender
[ 14]   0.00-10.00  sec   370 MBytes   310 Mbits/sec                  receiver
[ 16]   0.00-10.00  sec   363 MBytes   305 Mbits/sec  835             sender
[ 16]   0.00-10.00  sec   363 MBytes   304 Mbits/sec                  receiver
[ 18]   0.00-10.00  sec   421 MBytes   354 Mbits/sec  668             sender
[ 18]   0.00-10.00  sec   420 MBytes   352 Mbits/sec                  receiver
[SUM]   0.00-10.00  sec  3.00 GBytes  2.58 Gbits/sec  6880             sender
[SUM]   0.00-10.00  sec  2.99 GBytes  2.57 Gbits/sec                  receiver

iperf Done.
dkaramchandani@hyperionides:~$
```

```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec   417 MBytes   350 Mbits/sec  719             sender
[  4]   0.00-10.00  sec   415 MBytes   348 Mbits/sec                  receiver
[  6]   0.00-10.00  sec   388 MBytes   326 Mbits/sec  853             sender
[  6]   0.00-10.00  sec   387 MBytes   325 Mbits/sec                  receiver
[  8]   0.00-10.00  sec   337 MBytes   283 Mbits/sec  1151             sender
[  8]   0.00-10.00  sec   336 MBytes   282 Mbits/sec                  receiver
[ 10]   0.00-10.00  sec   453 MBytes   380 Mbits/sec  495             sender
[ 10]   0.00-10.00  sec   452 MBytes   379 Mbits/sec                  receiver
[ 12]   0.00-10.00  sec   315 MBytes   265 Mbits/sec  1167             sender
[ 12]   0.00-10.00  sec   314 MBytes   264 Mbits/sec                  receiver
[ 14]   0.00-10.00  sec   409 MBytes   343 Mbits/sec  773             sender
[ 14]   0.00-10.00  sec   408 MBytes   342 Mbits/sec                  receiver
[ 16]   0.00-10.00  sec   379 MBytes   318 Mbits/sec  823             sender
[ 16]   0.00-10.00  sec   378 MBytes   317 Mbits/sec                  receiver
[ 18]   0.00-10.00  sec   326 MBytes   273 Mbits/sec  979             sender
[ 18]   0.00-10.00  sec   325 MBytes   272 Mbits/sec                  receiver
[SUM]   0.00-10.00  sec  2.95 GBytes  2.54 Gbits/sec  6960             sender
[SUM]   0.00-10.00  sec  2.94 GBytes  2.53 Gbits/sec                  receiver

iperf Done.
dkaramchandani@hyperionides:~$
```

Similarly for UDP as well, I managed to secure the below screenshots :

UDP :

1 thread

```
dkaramchandani@hyperionides:~$ iperf3 -c 192.168.9.67 -P 1 -u
Connecting to host 192.168.9.67, port 5201
[  4] local 192.168.27.155 port 57471 connected to 192.168.9.67 port 5201
[ ID] Interval           Transfer     Bandwidth       Total Datagrams
[  4]   0.00-1.00   sec   120 KBytes   983 Kbits/sec  15
[  4]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  16
[  4]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  16
[  4]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec  16
[  4]   4.00-5.00   sec   128 KBytes  1.05 Mbits/sec  16
[  4]   5.00-6.00   sec   128 KBytes  1.05 Mbits/sec  16
[  4]   6.00-7.00   sec   128 KBytes  1.05 Mbits/sec  16
[  4]   7.00-8.00   sec   128 KBytes  1.05 Mbits/sec  16
[  4]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[  4]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  4]   0.00-10.00  sec  1.24 MBytes  1.04 Mbits/sec  0.274 ms  0/159 (0%)
[  4] Sent 159 datagrams

iperf Done.
```

2 threads

```
dkaramchandani@hyperionides:~$ iperf3 -c 192.168.9.67 -P 2 -u
Connecting to host 192.168.9.67, port 5201
[  4] local 192.168.27.155 port 57346 connected to 192.168.9.67 port 5201
[  6] local 192.168.27.155 port 60596 connected to 192.168.9.67 port 5201
[ ID] Interval           Transfer     Bandwidth       Total Datagrams
[  4]   0.00-1.00   sec   120 KBytes   982 Kbits/sec  15
[  6]   0.00-1.00   sec   120 KBytes   982 Kbits/sec  15
[SUM]   0.00-1.00   sec   240 KBytes  1.96 Mbits/sec  30
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   1.00-2.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   2.00-3.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   3.00-4.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   4.00-5.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   4.00-5.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   4.00-5.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   5.00-6.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   5.00-6.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   5.00-6.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   6.00-7.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   6.00-7.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   6.00-7.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   7.00-8.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   7.00-8.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   7.00-8.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   8.00-9.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[  6]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   9.00-10.00  sec   256 KBytes  2.10 Mbits/sec  32
```

```
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   5.00-6.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   5.00-6.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   5.00-6.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   6.00-7.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   6.00-7.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   6.00-7.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   7.00-8.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   7.00-8.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   7.00-8.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   8.00-9.00   sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[  6]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   9.00-10.00  sec   256 KBytes  2.10 Mbits/sec  32
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  4]   0.00-10.00  sec  1.24 MBytes  1.04 Mbits/sec  0.198 ms  0/159 (0%)
[  4] Sent 159 datagrams
[  6]   0.00-10.00  sec  1.24 MBytes  1.04 Mbits/sec  0.197 ms  0/159 (0%)
[  6] Sent 159 datagrams
[SUM]   0.00-10.00  sec  2.48 MBytes  2.08 Mbits/sec  0.197 ms  0/318 (0%)

iperf Done.
```

8 threads

```
dkaramchandani@hyperionides:~$ iperf3 -c 192.168.9.67 -P 8 -u
Connecting to host 192.168.9.67, port 5201
[  4] local 192.168.27.155 port 41043 connected to 192.168.9.67 port 5201
[  6] local 192.168.27.155 port 51942 connected to 192.168.9.67 port 5201
[  8] local 192.168.27.155 port 38680 connected to 192.168.9.67 port 5201
[ 10] local 192.168.27.155 port 52004 connected to 192.168.9.67 port 5201
[ 12] local 192.168.27.155 port 55070 connected to 192.168.9.67 port 5201
[ 14] local 192.168.27.155 port 49561 connected to 192.168.9.67 port 5201
[ 16] local 192.168.27.155 port 51946 connected to 192.168.9.67 port 5201
[ 18] local 192.168.27.155 port 49231 connected to 192.168.9.67 port 5201
[ ID] Interval           Transfer     Bandwidth       Total Datagrams
[  4]   0.00-1.00   sec   120 KBytes   983 Kbits/sec  15
[  6]   0.00-1.00   sec   120 KBytes   983 Kbits/sec  15
[  8]   0.00-1.00   sec   120 KBytes   983 Kbits/sec  15
[ 10]   0.00-1.00   sec   120 KBytes   983 Kbits/sec  15
[ 12]   0.00-1.00   sec   120 KBytes   983 Kbits/sec  15
[ 14]   0.00-1.00   sec   120 KBytes   983 Kbits/sec  15
[ 16]   0.00-1.00   sec   120 KBytes   983 Kbits/sec  15
[ 18]   0.00-1.00   sec   120 KBytes   983 Kbits/sec  15
[SUM]   0.00-1.00   sec   960 KBytes  7.86 Mbits/sec  120
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  16
[  8]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 10]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 12]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 14]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 16]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 18]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   1.00-2.00   sec  1.00 MBytes  8.39 Mbits/sec  128
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  16
[  8]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 10]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 12]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 14]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 16]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 18]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   2.00-3.00   sec  1.00 MBytes  8.39 Mbits/sec  128
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec  16
[  8]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec  16
```

```
[ 18]   7.00-8.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   7.00-8.00   sec  1.00 MBytes  8.39 Mbits/sec  128
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[  6]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[  8]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 10]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 12]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 14]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 16]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[ 18]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   8.00-9.00   sec  1.00 MBytes  8.39 Mbits/sec  128
- - - - - - - - - - - - - - - - - - - - - - - - -
[  4]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[  6]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[  8]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[ 10]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[ 12]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[ 14]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[ 16]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[ 18]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec  16
[SUM]   9.00-10.00  sec  1.00 MBytes  8.39 Mbits/sec  128
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Jitter    Lost/Total Datagrams
[  4]   0.00-10.00  sec  1.24 MBytes  1.04 Mbits/sec  1.245 ms  0/159 (0%)
[  4] Sent 159 datagrams
[  6]   0.00-10.00  sec  1.24 MBytes  1.04 Mbits/sec  1.099 ms  0/159 (0%)
[  6] Sent 159 datagrams
[  8]   0.00-10.00  sec  1.24 MBytes  1.04 Mbits/sec  0.949 ms  0/159 (0%)
[  8] Sent 159 datagrams
[ 10]   0.00-10.00  sec  1.24 MBytes  1.04 Mbits/sec  0.963 ms  0/159 (0%)
[ 10] Sent 159 datagrams
[ 12]   0.00-10.00  sec  1.24 MBytes  1.04 Mbits/sec  0.982 ms  0/159 (0%)
[ 12] Sent 159 datagrams
[ 14]   0.00-10.00  sec  1.24 MBytes  1.04 Mbits/sec  0.898 ms  0/159 (0%)
[ 14] Sent 159 datagrams
[ 16]   0.00-10.00  sec  1.24 MBytes  1.04 Mbits/sec  0.907 ms  0/159 (0%)
[ 16] Sent 159 datagrams
[ 18]   0.00-10.00  sec  1.24 MBytes  1.04 Mbits/sec  0.791 ms  0/159 (0%)
[ 18] Sent 159 datagrams
[SUM]   0.00-10.00  sec  9.94 MBytes  8.34 Mbits/sec  0.979 ms  0/1272 (0%)

iperf Done.
```