Name: Brijesh Mavani
CWID: A20406960
University: Illinois Institute of Technology
Course: Cloud Computing (CS 553)
Assignment: PA1

Design:

CPU Benchmarking:

- Programming language C has been used to implement this benchmarking task.
- In this program we are calculating the Giga Char operations, Giga Short operations, Giga Integer operations and Giga Floating point operations.
- Aim of the program is to utilize CPU to full capacity by performing various Arithmetic operations.
- Functions qops, hops, sops, dops are implemented for calculating Giga Char operations, Giga Short operations, Giga Integer operations and Giga Floating point operations respectively.
- Function qops: Total 1trillion operations are performed in an order to calculate the Giga operations. As Char datatype has a limit of 256, Interger data type has been used in iterations in For loop. The separate Forloop function has been created and executed 1st to extract time information for iterating For loop. Once the qops function is executed and timing are extracted, actual time of executing Char operations time is calculated by subtracting For loop time. Later, it calculates the Giga Char operations.
- Function hops: Total 1trillion operations are performed in an order to calculate the Giga operations. As Short datatype has a limit of 65,535, Interger data type has been used in iterations in For loop. The separate Forloop function has been created and executed 1st to extract time information for iterating For loop. Once the hops function is executed and timing are extracted, actual time of executing Short operations time is calculated by subtracting For loop time. Later, it calculates the Giga Short operations.
- Function sops: Total 1trillion operations are performed in an order to calculate the Giga Integer operations (GIOPS).
- Function dops: Total 1trillion operations are performed in an order to calculate the Giga Floating point operations (GFLOPS).
- Each of the functions are executed for 1, 2 and 4 threads.
- Strong scaling has been implemented in multithreading i.e. each thread will work on same workload.
- pthread_barrier_wait is utilized to synchronize all the threads. It has been included at multiple places to make sure only time taken by each function to iterate and perform arithmetic operations and not the variable creation and assignment operations.
- Scripts are created to execute the code and writing the output in a file.

Memory Benchmarking:

- Programming language C has been used to implement this benchmarking task.
- In this program we are calculating the throughput and latency of the main memory i.e. RAM.
- Design includes the different methods, RWS: read+write (e.g. memcpy) with sequential access pattern and RWR: read+write (e.g. memcpy) with random access pattern.
- Each method is implemented for various block sizes of 1B,1KB, 1MB and 10MB.
- For calculating throughput, functions are executed for block sizes 1KB, 1MB and 10MB for total 100GB data.
- Latency calculations are limited for the 100 million operations with 1B as block size.
- memset and memcpy functions are implemented to read and write from RAM.
- Strong scaling has been implemented in multithreading i.e. each thread will work on same workload.
- Each of the functions are executed for 1, 2 and 4 threads.
- pthread_barrier_wait is utilized to synchronize all the threads. It has been included at multiple places to make sure only time taken by each function to perform read and write operations on RAM and not the variable creation and assignment operations.
- Scripts are created to execute the code and writing the output in a file.

Disk Benchmarking:

- Programming language C has been used to implement this benchmarking task.
- In this program we are calculating the throughput, latency and IOPS of the Disk (Hard Disk).
- Design includes the different methods, RS: read with sequential access pattern, WS: write with sequential access pattern, RR: read with random access pattern and WS: write with random access pattern.
- Each method is implemented for various block sizes of 1KB,1MB, 10MB and 100MB.
- For calculating throughput, functions are executed for block sizes 1MB, 10MB and 100MB.for total 10GB data.
- Latency and IOPS calculations are limited for the 1GB of data with 1KB as block size.
- Various system calls such as open, write, read, etc. has been used to implement this benchmark.
- For sequential access for read, pread system call is used so there is no need to update the file pointer for subsequent read. For write operation, append mode has been used to make sure resulting file is of 1 or 10GB.
- For random access, a thread block size has been utilized with rand function to generate a random number and, is used as a location for read or write from file.
- O_SYNC option in open system call and fsync system call has been used to avoid reading from and writing to cache.

- Each of the functions are executed for 1, 2 and 4 threads except when block size is 1KB. In that case, additionally 8,16,32,64 and 128 threads are also executed.
- Strong scaling has been implemented in multithreading i.e. each thread will work on same workload.
- pthread_barrier_wait is utilized to synchronize all the threads. It has been included at multiple places to make sure only time taken by each function to perform read and write operations on Disk and not the variable creation and assignment operations.
- Scripts are created to execute the code and writing the output in a file.

Network Benchmarking:

- Programming language C has been used to implement this benchmarking task.
- In this program we are calculating the throughput and latency of the network.
- Design includes the different protocols, TCP and UDP.
- Each method is implemented for various block sizes of 1B, 1KB and 32KB.
- For calculating throughput, functions are executed for block sizes of 1KB and 32KB for total 100GB data.
- Latency calculations are limited for the 1million of operations with 1B as block size.
- The implementation is for transmitting packets from client to server over the network and back to get RTT of transmission.
- TCP is connection oriented protocol which requires the connection setup and acceptance between client and server before transmitting any data/packets.
- The UDP protocol is connection less protocol in which data/packets are transmitted without any established connection. This may result in data loss.
- Each of the functions are executed for 1, 2, 4 and 8 threads.
- Scripts are created to execute the code and writing the output in a file.

Improvement which can be done:

CPU Benchmarking:

Benchmarking could have been more efficient if implemented more complex instructions such as solving Liner equations. The implementation of AVX instructions would have provided more accurate results. LINPACK also uses such instructions and Matrix Multiplications which results in more FLOPS and IOPS. For future improvement, Linear equations and AVX instructions can be implemented.

Memory Benchmarking:

The memory availability was limited due to shared resource allocated for this experiment. The better understanding on AVX instructions and usage could have been helped to implement this benchmark more efficiently. For future enhancements, along with AVX instructions, handling cache in more efficient way can be implemented.

Disk Benchmarking:

As we were provided shared cluster to run this experiment, resource was not always available to execute the test cases. The limitation of creating files in tmp folder also made this benchmark execution difficult. In future to improve design and achieve better efficiency, we can extend the design to multi-core environment with more memory allocation.

Network Benchmarking:

The network loss was not accounted much in this benchmark. Though TCP handles it implicitly, there were several occasions when package loss was noted at client and server. Also, server was not stable, there were time when either client or server is executing. This was resulting in job time out due to limited allocated time to each job. For future, network loss can be counted to acquire accurate results.

Performance report:

The results of all the test cases and experiments are presented here. All testing has been done on Hyperion cluster.

CPU Benchmarking:

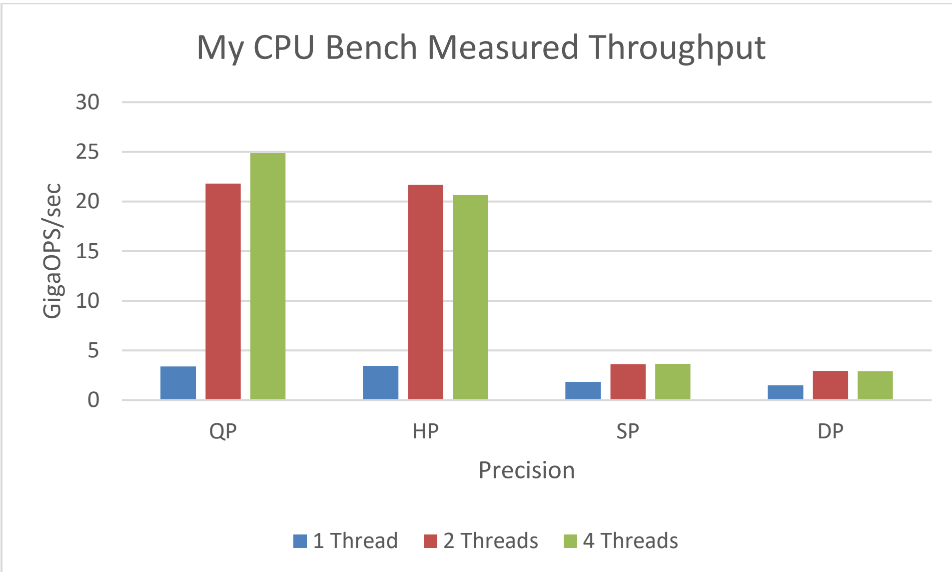
Model Name	CPU	Cache	CPU Cores
Intel(R) Xeon(R) CPU E5-2670 v3	2.30GHz	256 KB	1

All test cases were carried for three times and average benchmark value was calculated which is shown in below table. The standard deviation of all three runs is also calculated and present in table.

Workload	Concurrency	MyCPUBench Measured Ops/Sec (GigaOPS)	HPL Measured Ops/Sec (GigaOPS)	Theoretical Ops/Sec (GigaOPS)	MyCPUBench Efficiency (%)	HPL Efficiency (%)	Standard Deviation
QP	1	3.402684	N/A	588.8	0.606563	N/A	0.162719
QP	2	21.81213	N/A	588.8	4.354811	N/A	3.591331
QP	4	24.8573	N/A	588.8	4.24594	N/A	1.534984
HP	1	3.449062	N/A	294.4	1.163271	N/A	0.0635
HP	2	21.68376	N/A	294.4	8.087506	N/A	1.853281
HP	4	20.65276	N/A	294.4	6.532216	N/A	1.839364
SP	1	1.844318	N/A	147.2	1.284217	N/A	0.065036
SP	2	3.603588	N/A	147.2	2.506829	N/A	0.106707
SP	4	3.641645	N/A	147.2	2.434948	N/A	0.068509
DP	1	1.490745	38.1844	73.6	2.09353	51.88098	0.051015

DP	2	2.929472	73.4327	73.6	4.117276	99.77269	0.107182
DP	4	2.898467	70.1092	73.6	3.984461	95.25707	0.038219

Based on above result, few graphs have been created to organize observed information for easy interpretations.



Above graph shows the throughtput achieved by my code for various precisions. As we can see from graph, multithreading achieves the more throughput compared to single threaded execution. Also, the throughput achieved for quarter and half precisions is more than single precision and double precision operations.

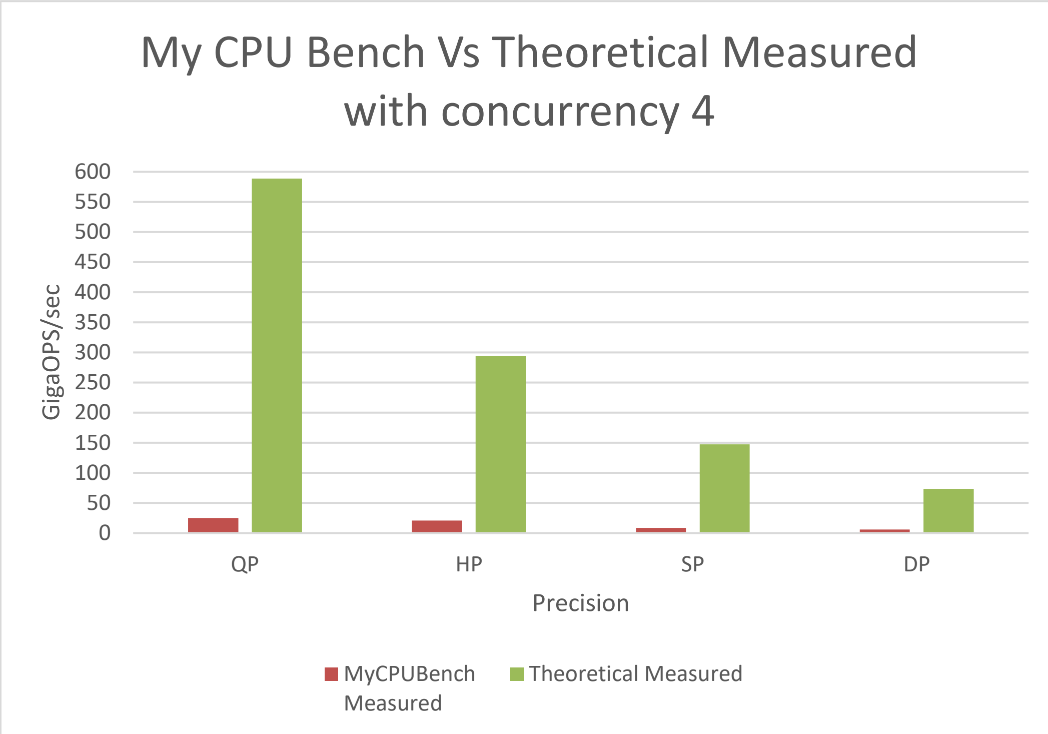
Through put has been calculated by formula:

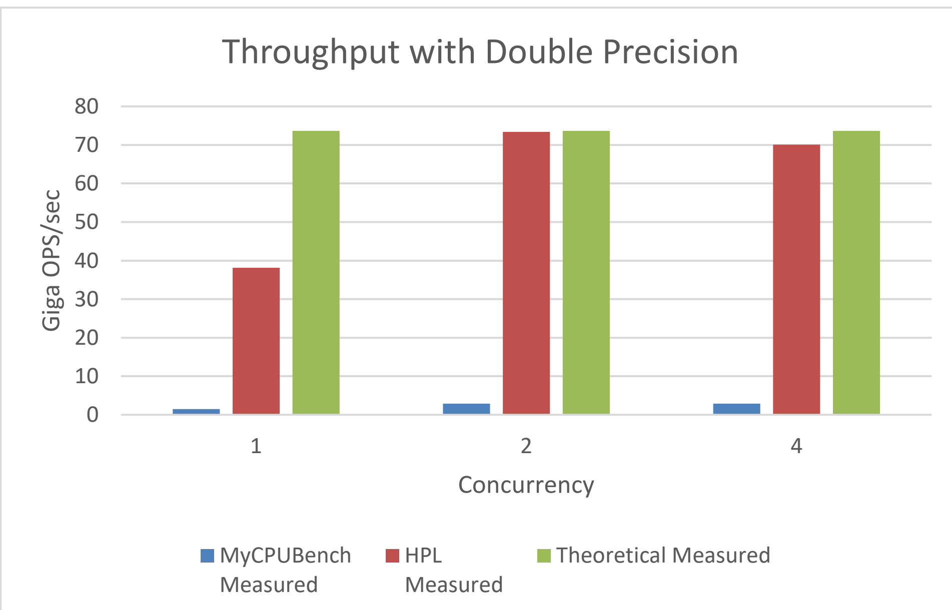
$$\mathrm{FLOPS} = \mathrm{sockets} \times \frac{\mathrm{cores}}{\mathrm{socket}} \times \frac{\mathrm{cycles}}{\mathrm{second}} \times \frac{\mathrm{FLOPs}}{\mathrm{cycle}}$$

Where, sockets =2, cores/socket = 1, cycles/second=2.3, FLOPs/cycle = 16,32,64,128 for DP,SP,HP and QP respectively.

Ref: https://en.wikipedia.org/wiki/FLOPS

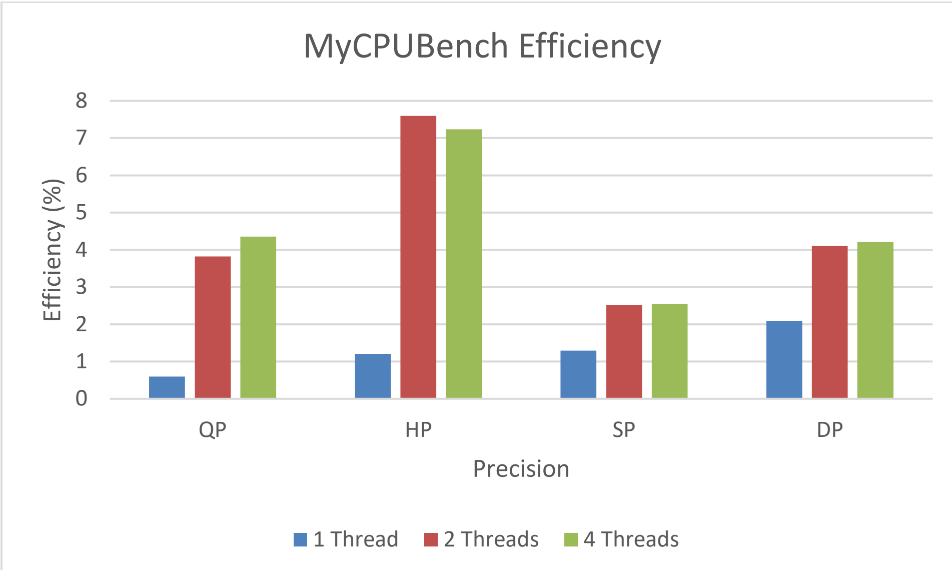
Below graph indicates that the throughput achieved by my code with multithreading with 4 threads is way too low compared to theoretical value. This is due to the same reason explained in improvement section above. The implementation of normal arithmetic operations doesn't push CPU to its limit to achieve high throughput which can be achieved if Matrix Multiplication or linear equations were solved. Through put can also be achieved by implementing AVX and FMA instructions.





We also executed the LINPACK which is a standard benchmark program to evaluate CPU performance. The results can be visualized in above graph. As we can see that LINPACK is providing throughput closer to the actual theoretical value of the CPU.

It is always better to provide the result in percent format to easy grasping and understanding the performance. Due to same, efficiency was calculated and plotted in bar chart below:



Memory Benchmarking:

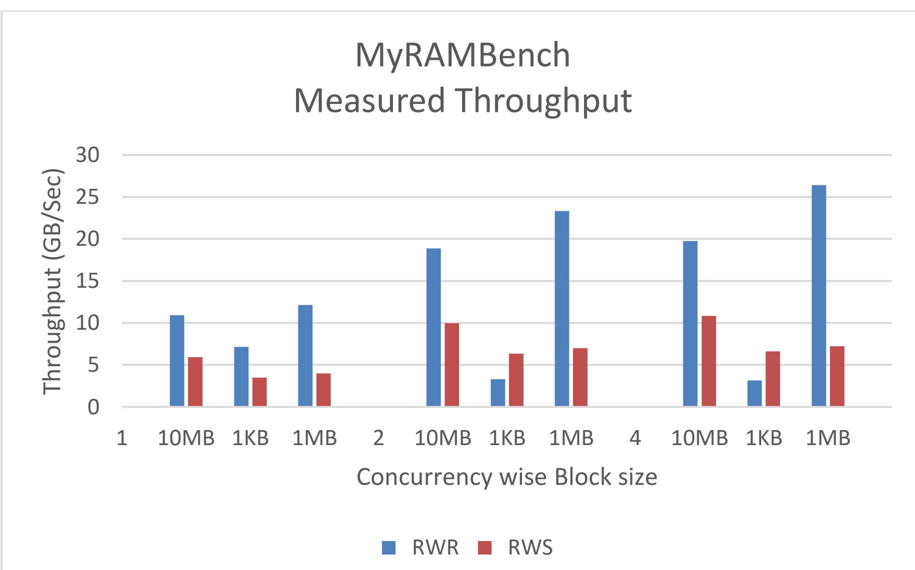
Specification: 8GiB DIMM Synchronous 2400 MHz (0.4 ns, 64bit)

Like CPU benchmarking all test cases were executed thrice and average and standard deviation has been calculated.

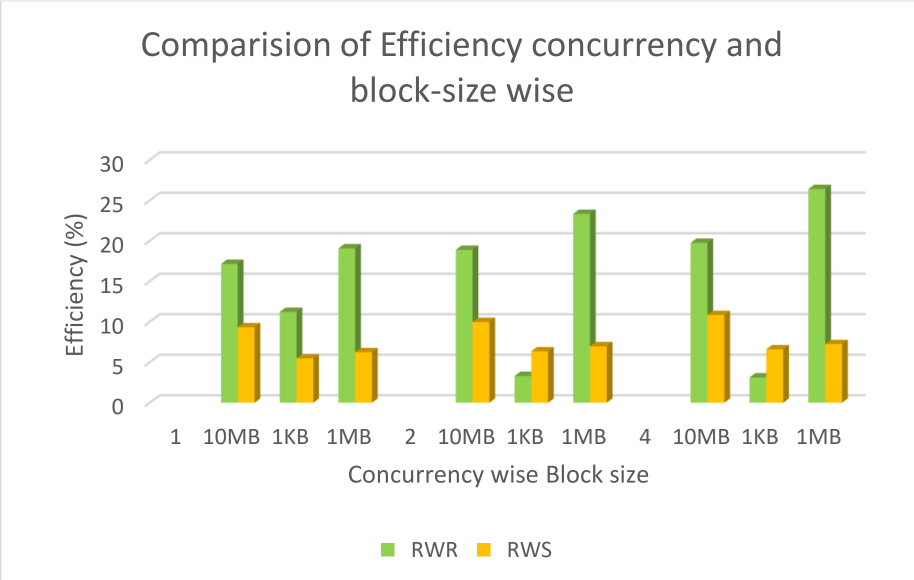
Workload	Concurrency	BlockSize	MyRAMBench Measured Throughput (GB/sec)	pmbw Measured Throughput (GB/sec)	Theoretical Throughput (GB/sec)	MyRAMBench Efficiency (%)	pmbw Efficiency (%)	Standard Deviation
RWS	1	1KB	3.480392	16.7557	63.56835	5.475039	26.35862	0.625841
RWS	2	1KB	6.357561	16.1617	63.56835	10.00114	25.42405	0.977229
RWS	4	1KB	6.60822	14.8422	63.56835	10.39546	23.34833	0.956674
RWS	1	1MB	3.973819	27.8342	63.56835	6.251255	43.78624	0.647799
RWS	2	1MB	6.980057	29.4591	63.56835	10.9804	46.34246	1.233654
RWS	4	1MB	7.242063	28.4649	63.56835	11.39256	44.77842	1.045204
RWS	1	10MB	5.931106	34.5599	63.56835	9.330281	54.36658	1.606689
RWS	2	10MB	9.97151	34.2737	63.56835	15.68628	53.9163	1.973847

RWS	4	10MB	10.83333	30.6455	63.56835	17.04202	48.20866	1.443376
RWR	1	1KB	7.130703	4.84358	63.56835	11.21738	7.619477	0.45808
RWR	2	1KB	3.311688	0.45295	63.56835	5.209649	0.712545	0.271212
RWR	4	1KB	3.133044	0.25561	63.56835	4.928622	0.402107	0.434419
RWR	1	1MB	12.12315	9.2621	63.56835	19.07165	14.57029	0.88592
RWR	2	1MB	23.33333	1.18606	63.56835	36.7059	1.865809	2.886751
RWR	4	1MB	26.40921	0.58467	63.56835	41.54459	0.919752	0.469511
RWR	1	10MB	10.90067	23.1729	63.56835	17.14796	36.45352	1.71426
RWR	2	10MB	18.88889	2.86845	63.56835	29.7143	4.512389	1.924501
RWR	4	10MB	19.75224	0.68336	63.56835	31.07244	1.075005	0.98329

The theoretical value has been calculated with formula:
(Clock frequency) * (lines per clock) * (Memory Bus Width) * (Number of Interfaces)
Where clock freq = 2133, lines per clock is 2, Memory Bus width being 64 and Number of Interfaces are 2. This gives the theoretical through put of memory as 63.568354.

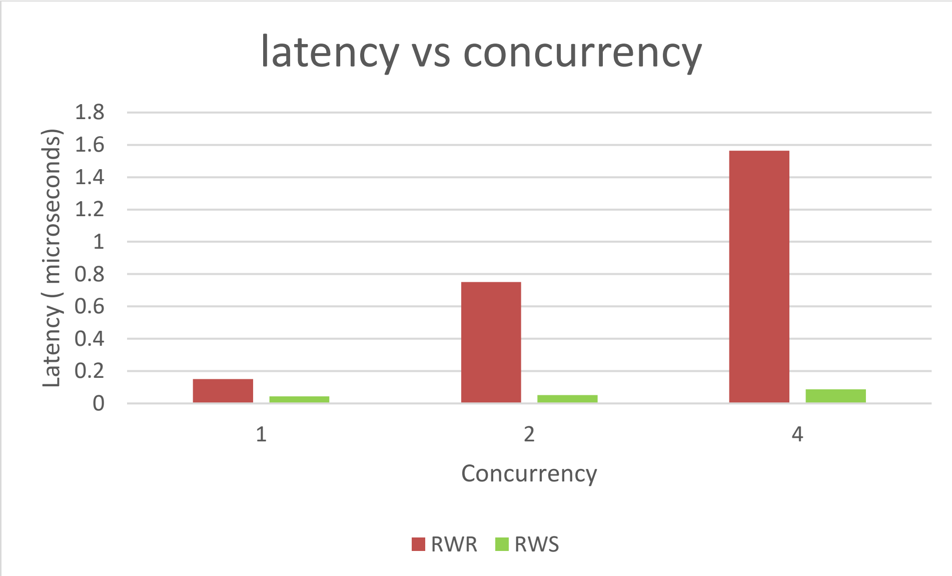


We can see that random-access pattern for read and write providing us more throughput then sequential access. Also, as thread count is increased there is an increase in through put. The efficiency of my code in terms of throughput is calculated and compared with different block sizes and access pattern. As we can see that read write random access is providing me more efficiency.



Workload	Concurrency	BlockSize	MyRAMBench Measured Latency (us)	pmbw Measured Latency (us)	Theoretical Latency (us)	MyRAMBench Efficiency (%)	pmbw Efficiency (%)	Standard Deviation
RWS	1	1B	0.0071773	0.006319	0.01406	51.0478663	44.9431	0.00029666
RWS	2	1B	0.009313	0.003148	0.01406	66.2375533	22.38976	0.00018745
RWS	4	1B	0.0098353	0.001874	0.01406	69.9525605	13.32859	0.0001677
RWR	1	1B	0.136594	0.143937	0.01406	971.507824	1023.734	0.0107543
RWR	2	1B	0.248353	0.067617	0.01406	1766.3798	480.9175	0.0268849
RWR	4	1B	0.307336	0.033241	0.01406	2185.88905	236.4225	0.0238849

Below diagram indicates the performance of the RAM in terms of latency when no. of threads is increased. The latency stays almost same for sequential access but increases with thread count for random access. This can happen as random access can increase the seek time for subsequent read/write operations.



The PMBW benchmark has been executed on memory to compare the performance of it with theoretical value. Benchmark has been executed for both throughput and latency. The result and % efficiency has been provided in the table earlier.

Disk Benchmarking:

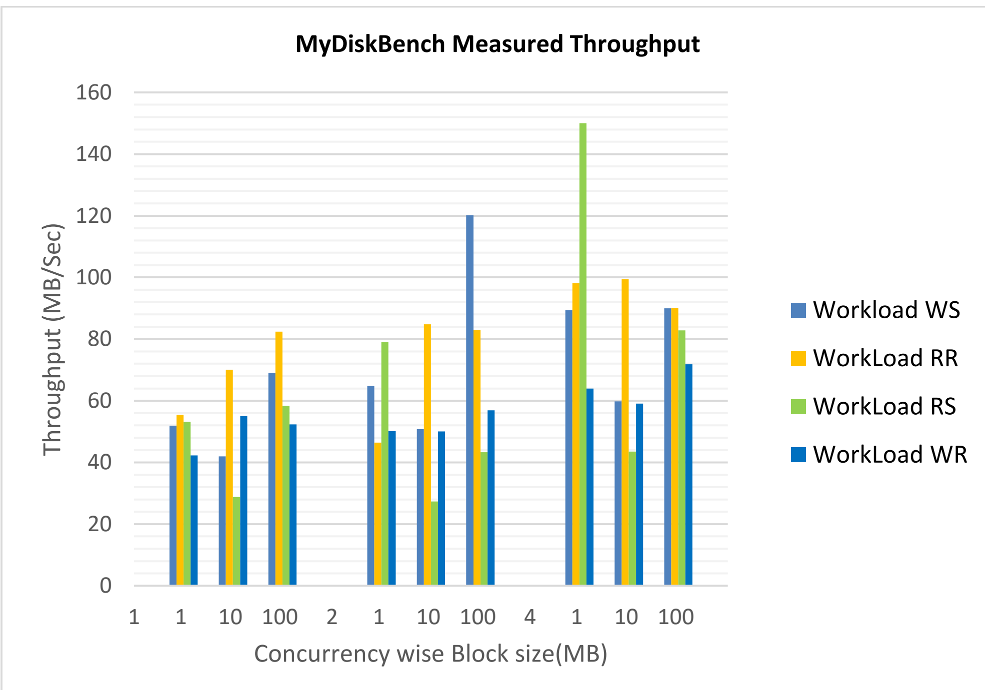
The disk benchmarking has been executed three times and average and standard deviations are calculated as below:

Workload	Concurrency	Block Size	MyDiskBench Measured Throughput (MB/sec)	IOZone Measured Throughput (MB/sec)	Theoretical Throughput (MB/sec)	MyDiskBench Efficiency (%)	IOZone Efficiency (%)	Standard Deviation
RR	1	1MB	55.4761903	313.18	600	9.246032	52.19667	13.57474
RR	2	1MB	46.4444443	353.18	600	7.740741	58.86333	39.41378
RR	4	1MB	98.2222223	303.18	600	16.37037	50.53	19.70689
RR	1	10MB	70.1111111	526.36	600	11.68519	87.72667	9.853445
RR	2	10MB	84.7777777	406.36	600	14.12963	67.72667	43.88933
RR	4	10MB	99.4578786	566.36	600	16.57631	94.39333	344.8706
RR	1	100MB	82.3903743	452.72	600	13.73173	75.45333	189.692

RR	2	100MB	82.888889	332.72	600	13.81481	55.45333	344.8706
RR	4	100MB	90.050505	532.72	600	15.00842	88.78667	44.78838
RS	1	1MB	53.1462817	310.52	600	8.857714	51.75333	3.742875
RS	2	1MB	79.0478133	325.52	600	13.17464	54.25333	15.98345
RS	4	1MB	150.056227	301.52	600	25.00937	50.25333	34.39568
RS	1	10MB	28.834042	321.04	600	4.805674	53.50667	21.52195
RS	2	10MB	27.3944	501.04	600	4.565733	83.50667	2.321074
RS	4	10MB	43.5521623	561.04	600	7.258694	93.50667	13.38196
RS	1	100MB	58.3392023	442.08	600	9.7232	73.68	13.89876
RS	2	100MB	43.2850241	302.08	600	7.214171	50.34667	213.602
RS	4	100MB	82.7768393	302.08	600	13.79614	50.34667	12.28252
WR	1	1MB	42.2673927	284.25	600	7.044565	47.375	0.850862
WR	2	1MB	50.1192487	219.25	600	8.353208	36.54167	0.61584
WR	4	1MB	63.910731	184.25	600	10.65179	30.70833	5.318532
WR	1	10MB	55.0478043	598.5	600	9.174634	99.75	6.333656
WR	2	10MB	50.0312443	468.5	600	8.338541	78.08333	6.333656
WR	4	10MB	59.0896523	588.5	600	9.848275	98.08333	6.333656
WR	1	100MB	52.3751237	426	600	8.729187	71	1.817117
WR	2	100MB	56.9466907	367	600	9.491115	61.16667	8.857561
WR	4	100MB	71.8187857	483	600	11.9698	80.5	23.97478
WS	1	1MB	51.9362277	185.63	600	8.656038	30.93833	1.850086
WS	2	1MB	64.7588123	195.63	600	10.79314	32.605	4.718009
WS	4	1MB	89.331139	105.63	600	14.88852	17.605	1.969421

WS	1	10MB	41.978678	371.26	600	6.996446	61.87667	3.981788
WS	2	10MB	50.735601	351.26	600	8.455934	58.54333	1.787333
WS	4	10MB	59.8276273	251.26	600	9.971271	41.87667	2.358361
WS	1	100MB	69.0545383	410.469	600	11.50909	68.4115	4.216329
WS	2	100MB	120.146088	562.52	600	20.02435	93.75333	16.55188
WS	4	100MB	89.9509377	540.458	600	14.99182	90.07633	4.084427

Throughput bar graph has been plotted as below.

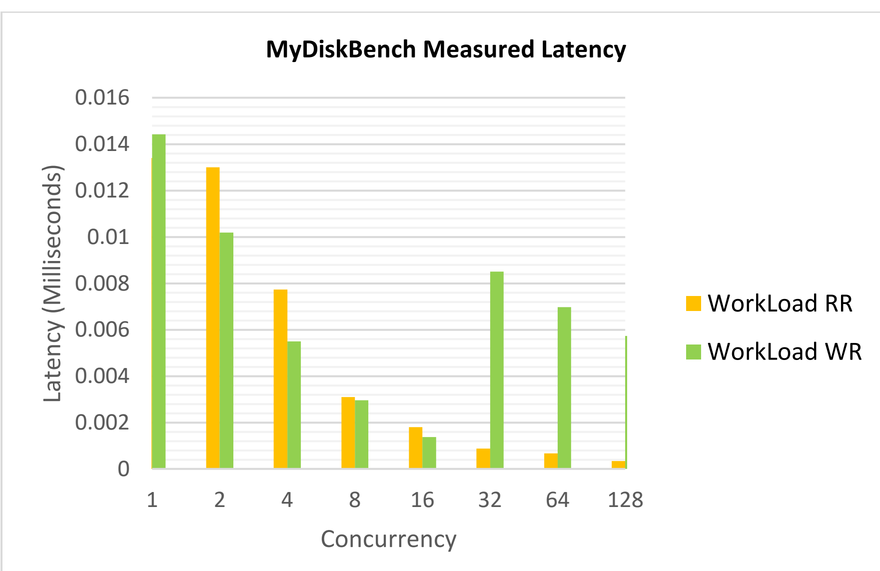


Latency performance can be seen in below table:

Workload	Concurrency	BlockSize	MyDiskBench Measured Latency (ms)	IOZone Measured Latency (ms)	Theoretical Latency (ms)	MyDiskBench Efficiency (%)	IOZone Efficiency (%)	StandardDeviation
RR	1	1KB	0.013400000	0.022	4.16	0.032211538	0.528846	0.002042058
	2	1KB	0.013000000	0.0276	4.16	0.03125	0.663462	0.001345362

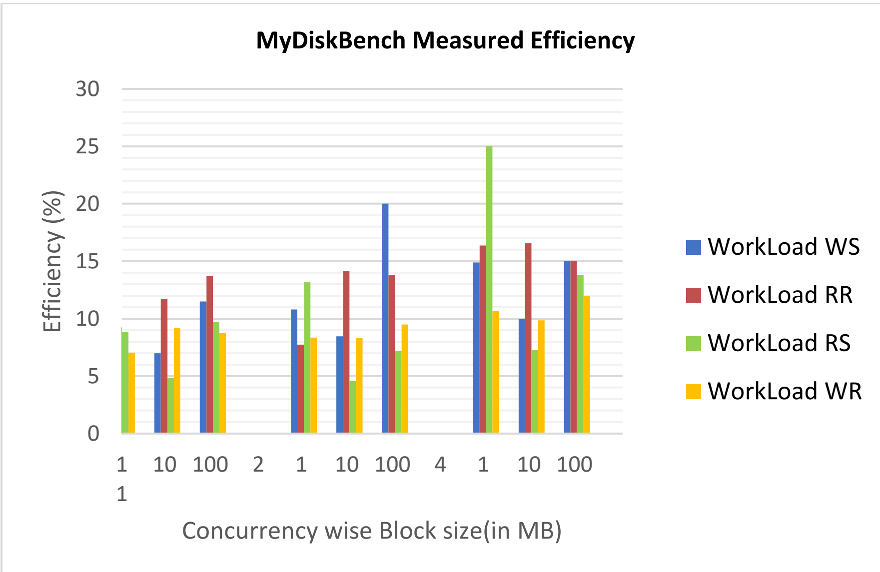
	4	1KB	0.007733333	0.035	4.16	0.018589744	0.841346	0.000757188
	8	1KB	0.003100000	0.0267	4.16	0.007451923	0.641827	0.000300000
	16	1KB	0.001800000	0.007	4.16	0.004326923	0.168269	0.000200000
	32	1KB	0.000874275	0.036	4.16	0.000232372	0.865385	0.000057735
	64	1KB	0.000675838	0.017	4.16	0.000160256	0.408654	0.000057735
	128	1KB	0.000340000	0.078	4.16	0.008173077	1.875	0.002535744
WR	1	1KB	0.014416667	0.156	4.16	0.346554487	3.75	0.021813834
	2	1KB	0.010186667	0.038	4.16	0.244871795	0.913462	0.001289703
	4	1KB	0.005500000	0.364	4.16	0.132211538	8.75	0.003306055
	8	1KB	0.002960000	0.026	4.16	0.071153846	0.625	0.001777639
	16	1KB	0.001366667	0.0456	4.16	0.032852564	1.096154	0.000923760
	32	1KB	0.008500000	0.106	4.16	0.020432692	2.548077	0.000458258
	64	1KB	0.006966667	0.0556	4.16	0.016746795	1.336538	0.000450925
	128	1KB	0.005733333	0.0409	4.16	0.013782051	0.983173	0.000550757

Latency with respect to the number of threads and random read write pattern has been plotted. One can notice that as number of threads increases the latency decreases.



Name: Brijesh Mavani CWID: A20406960 Assignment: 1

Disk efficiency of throughput has been plotted in below graph:

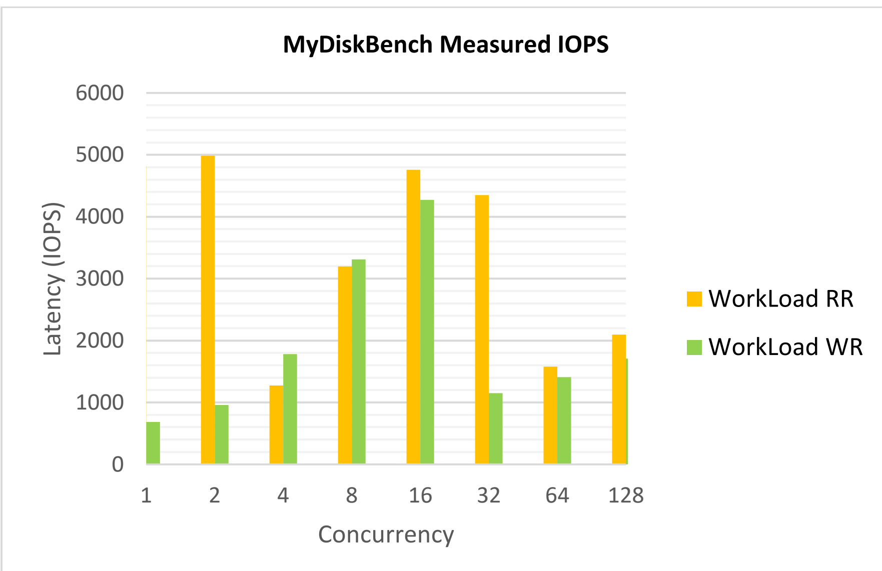


IOPS details are provided in below table:

Workload	Concurrency	BlockSize	MyDiskBench Measured IOPS	IOZone Measured IOPS	Theoretical IOPS	MyDiskBench Efficiency (%)	IOZone Efficiency (%)	Standard Deviation
RR	1	1KB	4822.889785	5321.987561	5782.4991	83.404938	92.0361157	12.2180961
RR	2	1KB	4983.665943	5497.987561	5782.4991	86.1853302	95.0797824	8.325151721
RR	4	1KB	1276.284976	36895.14572	5782.4991	22.0715119	638.048448	11.94870544
RR	8	1KB	3195.155825	3589.447851	5782.4991	55.2556217	62.0743348	29.1672035
RR	16	1KB	4760.044739	4896.044767	5782.4991	82.3181234	84.6700481	58.8521369
RR	32	1KB	4350.259394	4865.258694	5782.4991	75.2314755	84.1376473	55.0359669
RR	64	1KB	1581.186032	2564.986032	5782.4991	27.3443369	44.357742	144.1418181
RR	128	1KB	2097.152	3987.512415	5782.4991	36.2672257	68.9582886	3.56448E-13
WR	1	1KB	687.5862213	1587.586221	5782.4991	11.8908142	27.4550189	0.999236558

WR	2	1KB	958.8734217	3558.873422	5782.4991	16.5823359	61.5455941	0.121031293
WR	4	1KB	1780.481961	2480.436961	5782.4991	30.7908731	42.8955875	1.055482213
WR	8	1KB	3309.236347	4567.245347	5782.4991	57.2284801	78.9839353	2.066807157
WR	16	1KB	4273.064618	4973.045618	5782.4991	73.896503	86.001667	5.150556157
WR	32	1KB	1150.450133	3420.225486	5782.4991	19.8953794	59.1478776	6.340631364
WR	64	1KB	1407.792813	2245.875691	5782.4991	24.3457507	38.8391879	8.573895255
WR	128	1KB	1711.152574	2687.457874	5782.4991	29.5919212	46.4757163	16.53557076

IOPS performance graph for various number of threads are provided below:



Theoretical value calculation:
IOPS = 1/(average seek time)+(average latency)

average seek time (read/write)= 0.0085/0.0095ms
average latency=4.16ms
IOPS = 78.98894(Read) and 73.20644(Write). Total: 5782.4991

Ref for calculating Theoretical values:

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

https://www.cnet.com/products/seagate-constellation-2-st9250610ns-hard-drive-250-gb-sata-6gb-s/specs/

IOZONE tool has been also executed to compare our results. The resulted values are shown in different format in above tables.

```
bmavani@compute-2: /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 102400 kByte records

        Children see throughput for  2 initial writers  =  305749.34 kB/sec
        Parent sees throughput for  2 initial writers   =  303826.78 kB/sec
        Min throughput per process                      =  151205.28 kB/sec
        Max throughput per process                      =  154544.06 kB/sec
        Avg throughput per process                      =  152874.67 kB/sec
        Min xfer                                        = 10240000.00 kB

        Children see throughput for  2 rewriters        =  308558.73 kB/sec
        Parent sees throughput for  2 rewriters         =  308251.73 kB/sec
        Min throughput per process                      =  152602.97 kB/sec
        Max throughput per process                      =  155955.77 kB/sec
        Avg throughput per process                      =  154279.37 kB/sec
        Min xfer                                        = 10240000.00 kB

        Children see throughput for  2 readers          = 1096008.94 kB/sec
        Parent sees throughput for  2 readers           = 1095796.07 kB/sec
        Min throughput per process                      =  545237.62 kB/sec
        Max throughput per process                      =  550771.31 kB/sec
        Avg throughput per process                      =  548004.47 kB/sec
        Min xfer                                        = 10342400.00 kB

        Children see throughput for 2 re-readers        = 1093827.31 kB/sec
        Parent sees throughput for 2 re-readers         = 1093409.19 kB/sec
        Min throughput per process                      =  541323.56 kB/sec
        Max throughput per process                      =  552503.75 kB/sec
        Avg throughput per process                      =  546913.66 kB/sec
        Min xfer                                        = 10240000.00 kB

        Children see throughput for 2 random readers    = 1088108.94 kB/sec
        Parent sees throughput for 2 random readers     = 1085513.23 kB/sec
        Min throughput per process                      =  534693.75 kB/sec
        Max throughput per process                      =  553415.19 kB/sec
        Avg throughput per process                      =  544054.47 kB/sec
        Min xfer                                        = 10137600.00 kB
```

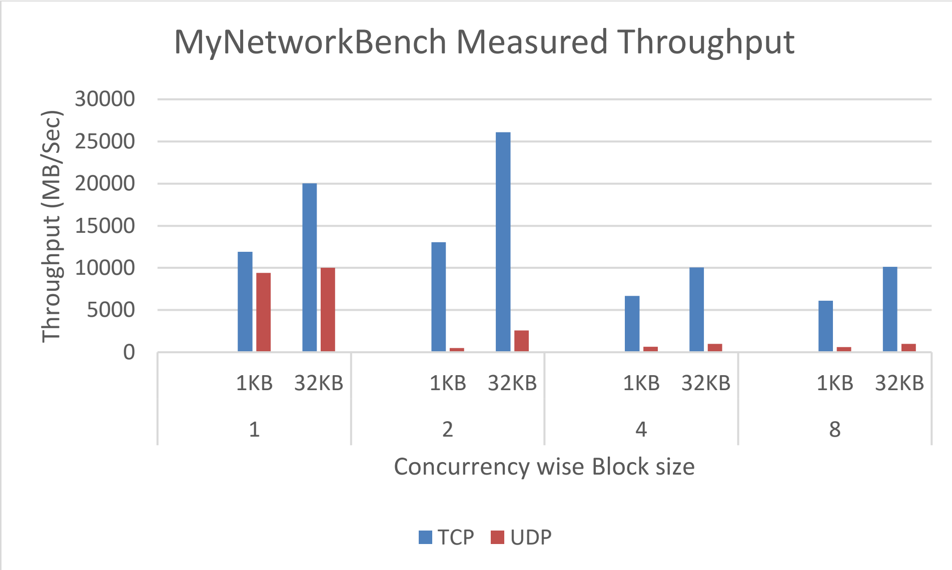
4 threads

```
bmavani@compute-3:~/assignments/benchmarks$ srun iozone -s 10g -r 1m -o -l4 -u4 -i 0 -i 1 -i 2 -e -I -F -f /tmp/iozonetmp1_b.txt /tmp/iozonetmp2_b.txt /tmp/iozonetmp3_b
.txt /tmp/iozonetmp4_b.txt
        Iozone: Performance Test of File I/O
                Version $Revision: 3.471 $
                Compiled for 64 bit mode.
                Build: linux-AMD64

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

        Run began: Tue Mar 27 14:32:11 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 -l4 -u4 -i 0 -i 1 -i 2 -e -I -F -f /tmp/iozonetmp1_b.txt /tmp/iozonetmp2_b.txt /tmp/iozonetmp3_b.txt /tmp/ioz
onetmp4_b.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 = 4
        Max process = 4
        Throughput test with 4 processes
        Each process writes a 10485760 kByte file in 1024 kByte records

Error writing block 0, fd= 3
write: Disk quota exceeded

        Children see throughput for  4 initial writers  =  278709.01 kB/sec
        Parent sees throughput for  4 initial writers   =  173657.12 kB/sec
        Min throughput per process                      =   40873.22 kB/sec
        Max throughput per process                      =  153438.39 kB/sec
```

Network Benchmarking:

The network benchmark evaluation is performed for TCP as well as UDP. Different Packet sizes are transferred ranging from 1B, 1KB, 32KB and the throughput has been calculated. The RTT has been calculated for packet size 1B.

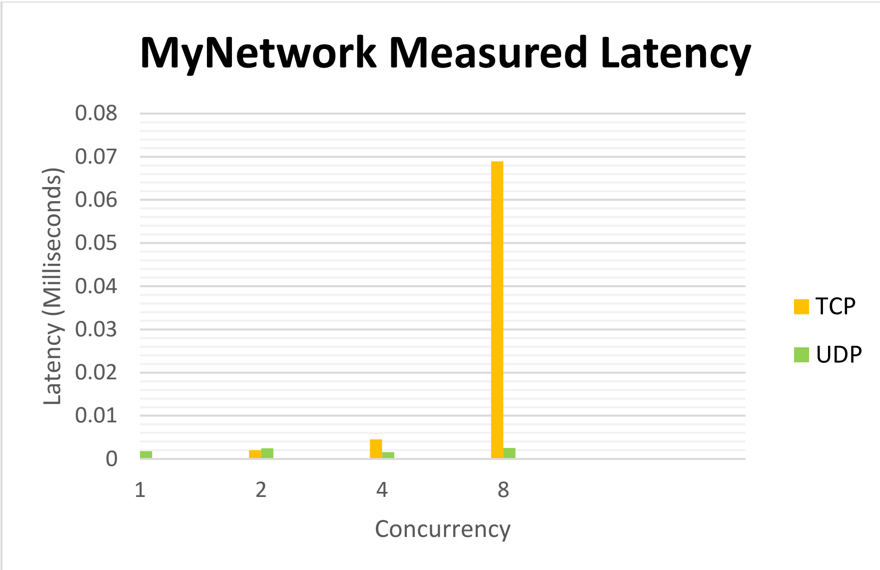
Protocol	Concurrency	Block size	MyNETBench Measured Throughput (Mb/sec)	iperf Measured Throughput (Mb/sec)	Theoretical Throughput (Mb/sec)	MyNETBench Efficiency (%)	iperf Efficiency(%)
TCP	1	1KB	11920.43	1485.25	56000	21.28648	2.652232143
TCP	1	32KB	20020.714	2756.25	56000	35.75128	4.921875
TCP	2	1KB	13045	1706.13	56000	23.29464	3.046660714
TCP	2	32KB	26106.35	2910.12	56000	46.61848	5.196642857
TCP	4	1KB	6669.366	6941.24	56000	11.90958	12.39507143
TCP	4	32KB	10063.88	7970.15	56000	17.97122	14.23241071
TCP	8	1KB	6119.104	5418.12	56000	10.92697	9.675214286
TCP	8	32KB	10114.7	8214.19	56000	18.06196	14.66819643
UDP	1	1KB	9420.43	2941.18	56000	16.8222	5.252107143
UDP	1	32KB	10010.47	3719.14	56000	17.87583	6.641321429
UDP	2	1KB	511.3977	3641.25	56000	0.91321	6.502232143
UDP	2	32KB	2600.635	4648.49	56000	4.643991	8.300875
UDP	4	1KB	656.9366	3751.04	56000	1.173101	6.698285714
UDP	4	32KB	995.3827	5419.27	56000	1.777469	9.677267857
UDP	8	1KB	601.9104	5717.18	56000	1.07484	10.20925
UDP	8	32KB	1001.467	4971.13	56000	1.788334	8.877017857



We can observe that network through put increases with increase in packet size which is being sent over network with TCP or UDP protocol. As UDP is connection less protocol there might be a lot of data loss during the transmission.

Latency performance are given in below table.

Protocol	Concurrency	Message Size	MyNETBench Measured Latency(ms)	ping Measured Latency (ms)	Theoretical Latency (ms)	MyNETBench Efficiency(%)	iperf Efficicency(%)
TCP	1	1B	0.003530165	1.99	0.0007	504.3093109	284285.7143
TCP	2	1B	0.002003535	1.99	0.0007	286.2193439	284285.7143
TCP	4	1B	0.004533607	1.99	0.0007	647.6581161	284285.7143
TCP	8	1B	0.068912551	1.99	0.0007	9844.650163	284285.7143
UDP	1	1B	1.58E-05	1.99	0.0007	2.254942857	284285.7143
UDP	2	1B	2.47E-05	1.99	0.0007	3.522857143	284285.7143
UDP	4	1B	1.55E-05	1.99	0.0007	2.210142857	284285.7143
UDP	8	1B	4.58E-05	1.99	0.0007	6.540742857	284285.7143



Above graph shows the latency factor of both the protocol for various concurrency. Iperf benchmarking was executed and results are shown in above tables.