Design and Report Document for Benchmarking

1. Processor benchmark:

- I used several functions while designing this benchmark. My benchmark has many functionalities like strong scaling and I used pthread for thread synchronization and to achieve concurrency. I also implement benchmark with AVX instruction.
- I am using strong scaling in this benchmark, I took fixed amount of work and reduce the amount of works per thread as I increase number of threads.

Design of code:

- In the main function where I called method myQPFunc (), myHPFunc(),mySPFunc(),myDPFunc and print the output named CPUOUTPUT.txt in output folder.
- I used pthread for thread synchronization, pthread_create to create thread and pthread_join which waits for thread termination. I specify the loop and call pthread_create and pthread_join function according to number of threads passed.
- I have used AVX instructions for the operation.
- AVX vectors can contain up to 256 bits of data. For myDPFunc () function I used data type __m256d which state 256-bit vector containing 4 doubles.
- AVX vectors can contain up to 256 bits of data. For mySPFunc() function I used data type __m256d which state 256-bit vector containing 8 integer.
- AVX vectors can contain up to 256 bits of data. For myHPFunc () function I used data type __m256d which state 256-bit vector containing 16 short integers.
- AVX vectors can contain up to 256 bits of data. For myQPFunc () function I used data type __m256d which state 256-bit vector containing 32 char.
- I run the loops in this function for run the test for multiple threads and for different operation type (char, short integer, integer, float).

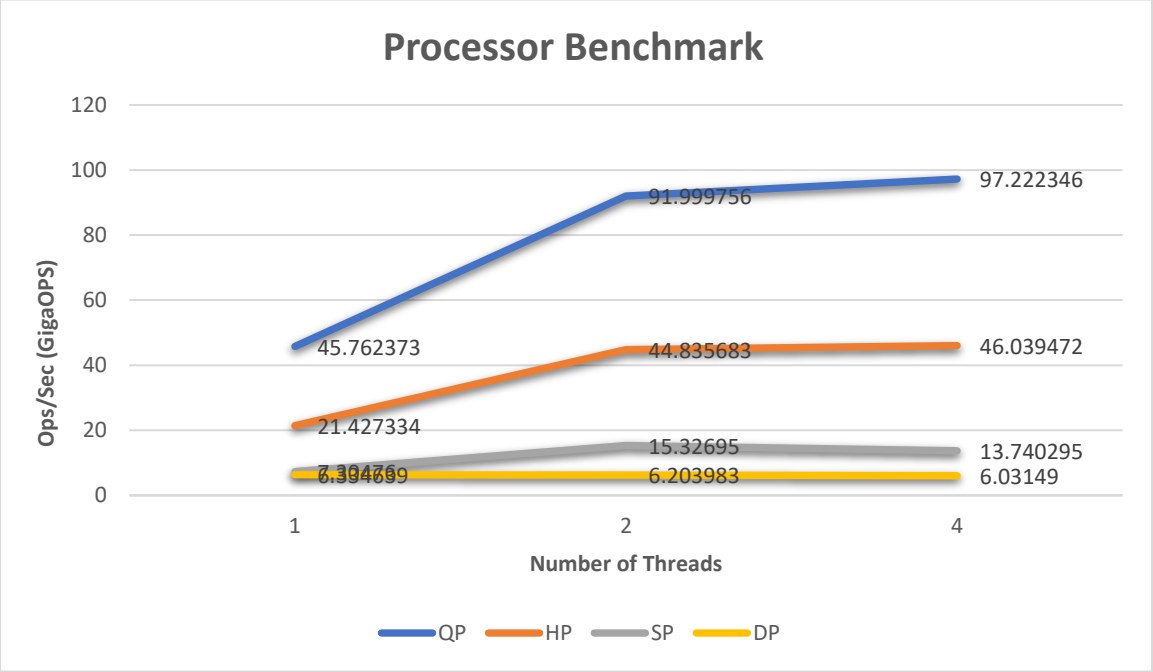
Theoretical Performance for Processor:

Theoretical Performance = CPU Speed * Number of Cores * Instruction per Cycle
So, for QP=883.20, HP =441.60, SP = 220.80, DP= 110.40
For Efficiency = My CPU Bench value / Theoretical CPU Bench value

Benchmark result:

Workload	Concurrency	My CPU Bench Measured Ops/Sec (GigaOPS)	HPL Measured Ops/Sec (GigaOPS)	Theoretical Ops/Sec (GigaOPS)	My CPU Bench Efficiency (%)	HPL Efficiency (%)
QP	1	45.762373	N/A	883.20	5.18	N/A
QP	2	91.999756	N/A	883.20	10.42	N/A
QP	4	97.222346	N/A	883.20	11.01	N/A
HP	1	21.427334	N/A	441.60	4.85	N/A
HP	2	44.835683	N/A	441.60	10.15	N/A
HP	4	46.039472	N/A	441.60	10.43	N/A
SP	1	7.304760	N/A	220.80	3.31	N/A

SP	2	15.326950	N/A	220.80	6.94	N/A
SP	4	13.740295	N/A	220.80	6.22	N/A
DP	1	6.334639	35.34	110.40	5.74	32.01
DP	2	6.203983	61.80	110.40	5.62	55.97
DP	4	6.031490	72.20	110.40	5.46	65.39



Conclusion:

- From graph we can conclude that as we increase the concurrency i.e. number of threads the operations per seconds are increasing. So, there is a performance increase as we increase the number of threads. Also, the operations per seconds are more for QP than SP than HP than DP. So as the precision increases the operations per seconds are decreasing.
- Also, in regards with the comparison of my performance to that achieved by your benchmark, HPL, and the theoretical peak performance HPL is having the performance near to theoretical peak where as my performance is very less than the theoretical peak performance.
- For improvements I would suggest that we should increase the concurrency i.e. we can try the experiment with the increasing the number of threads.

2. Memory benchmark:

- I used C language for memory benchmarking. Three block sizes are considered while testing data that is 1kb,1MB,10MB for throughput and 1 byte of data for calculating the latency.
- I used several functions while designing this benchmark. Our benchmark has many functionalities like I did strong scaling and I used pthread for thread synchronization. I include two basic operations Read+Write sequential access and Read+Write random access.
- I make one struct variable named readwriteinfo which stores the data for each thread like start_block and end_block to maintain starting and ending point for memory reading or writing from buffer char array. Also, readwriteinfoalso uses store info like block size and operation type for each thread.
- I am using strong scaling in this benchmark, I took fixed amount of work and reduce the amount of works per thread as I increase number of threads.

Design of code:

- In the main function where I called method RWR (), RWS (), RWRLat (), RWSLat() and print the output named RAMLATENCYOUTPUT for latency and RAMOUTPUT.txt for throughput and latency in output folder.
- I have used memset() for sequential write access, random write access in which we use memory allocated variable and data like block size etc.
- Also, I have used memcpy() for read+write operation in which we use memory allocated variable and data like block size etc.
- For throughput is calculated in GB/sec using data size of 1 GB and iterating over it 100 times to get the output for 100 GB.
- For latency measured in milliseconds,100 million operations are performed using block size as 1 byte.

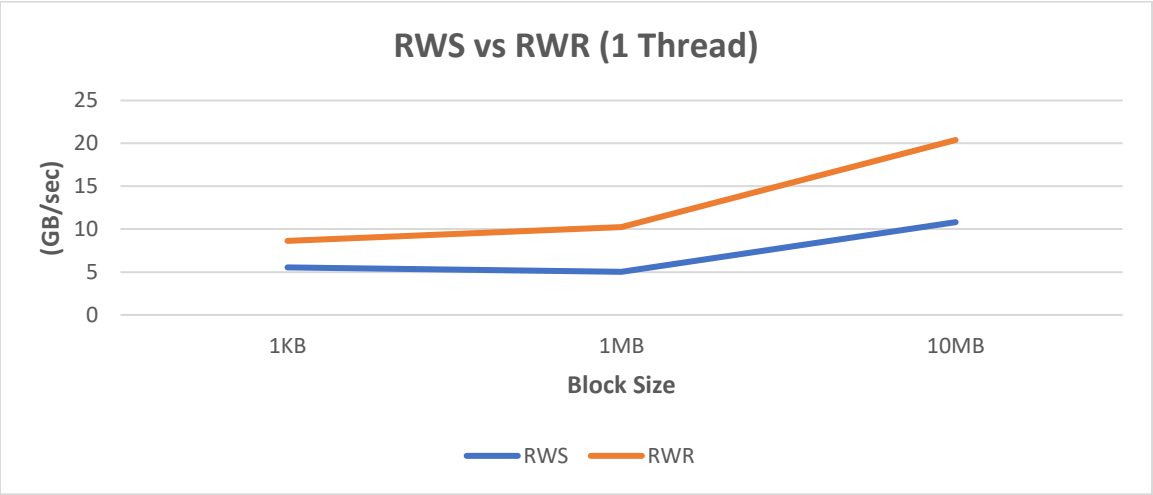
Theoretical Performance for Memory:

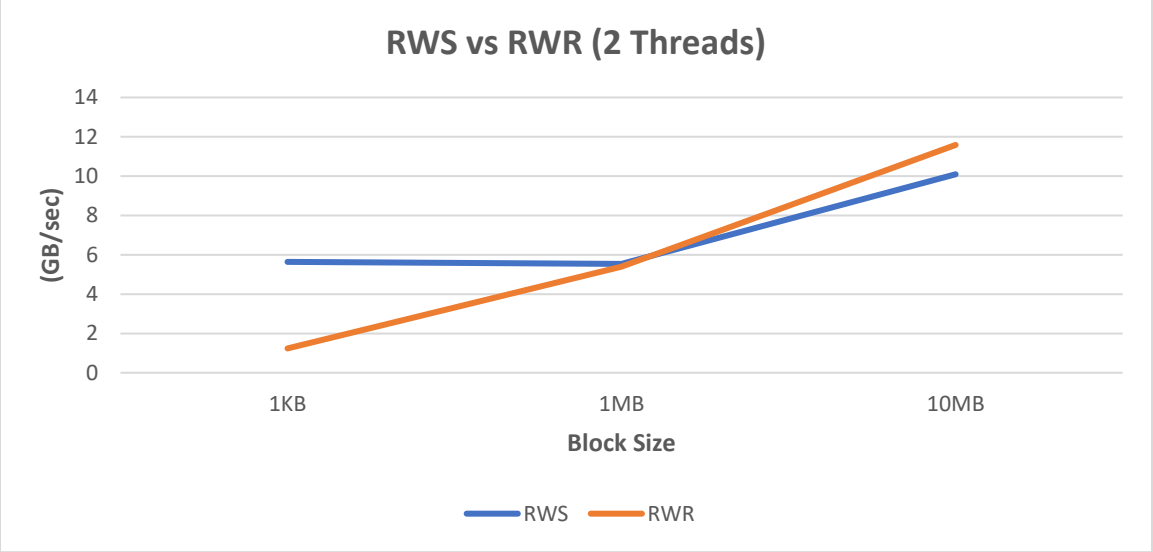
Throughput = Base DRAM clock frequency * lines per clock * memory bus width * number of interfaces.
So, by using the specifications for Xeon E5 processors the theoretical throughput is 68.25.
Efficiency = My RAM Bench Measured Throughput / Theoretical Throughput (GB/sec)
For latency = time for execution / number of instructions

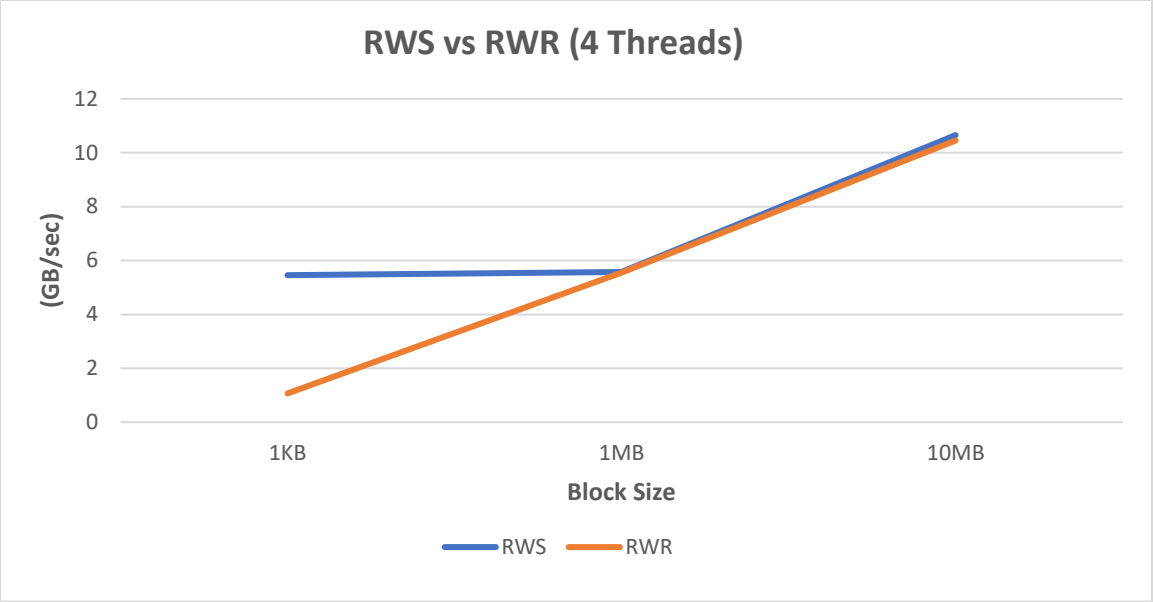
Benchmark results: Throughput

Work-load	Con-currency	Block Size	My RAM Bench Measured Throughput (GB/sec)	pmbw Measured Throughput (GB/sec)	Theoretical Throughput (GB/sec)	My RAM Bench Efficiency (%)	pmbw Efficiency (%)
RWS	1	1KB	5.530976	14.90	68.25	8.10	21.83
RWS	1	1MB	5.026109	19.71	68.25	7.36	28.88
RWS	1	10MB	10.814869	17.90	68.25	15.85	26.23
RWS	2	1KB	5.646446	22.53	68.25	8.27	33.01
RWS	2	1MB	5.534883	20.14	68.25	8.11	29.51
RWS	2	10MB	10.093359	37.83	68.25	14.79	55.43

RWS	4	1KB	5.457641	27.51	68.25	8.00	40.31
RWS	4	1MB	5.571923	40.50	68.25	8.16	59.34
RWS	4	10MB	10.657001	34.58	68.25	15.61	50.67
RWR	1	1KB	3.100642	5.3	68.25	4.54	7.77
RWR	1	1MB	5.197486	1.20	68.25	7.62	1.76
RWR	1	10MB	9.575159	1.01	68.25	14.03	1.48
RWR	2	1KB	1.236636	8.60	68.25	8.10	21.83
RWR	2	1MB	5.390845	1.10	68.25	7.36	28.88
RWR	2	10MB	11.585157	0.79	68.25	15.85	26.23
RWR	4	1KB	1.067270	20.50	68.25	8.27	33.01
RWR	4	1MB	5.555159	3.11	68.25	8.11	29.51
RWR	4	10MB	10.455595	1.01	68.25	14.79	55.43





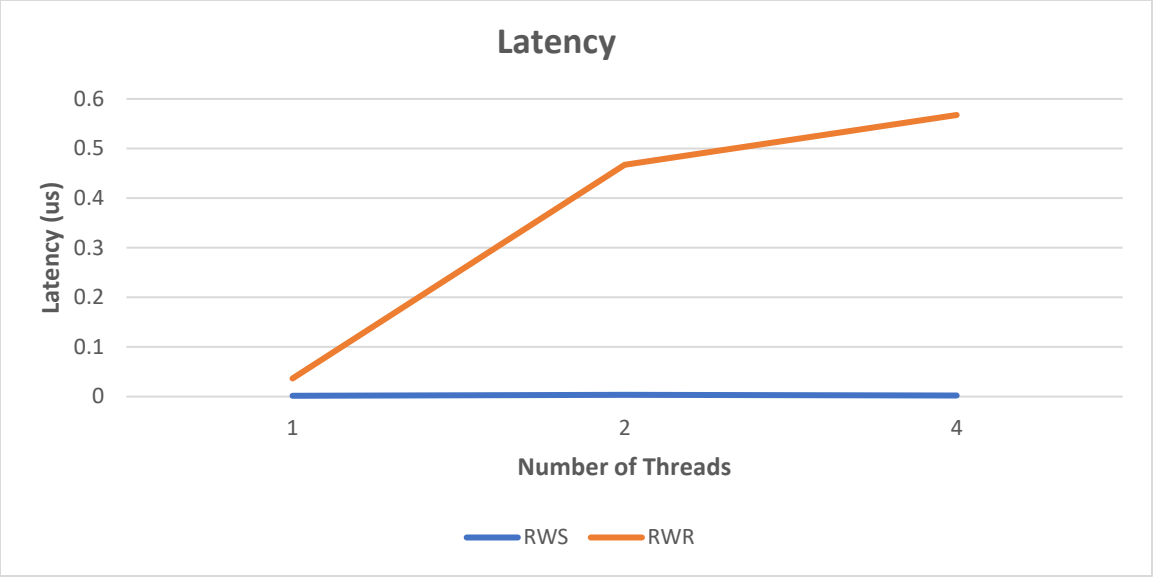


Conclusion:

- From the graph we can see that throughput is increasing as the block size is increasing. So, for obtaining higher throughput we should use the more block size. Also, if we compare the throughput with respect to number of threads then throughput value is increased as number of threads increased. So, for we could obtain the higher throughput with increase in the concurrency.
- Also, in regards with the comparison of My RAM Bench Measured to that achieved by your benchmark, pmbw, and the theoretical performance pmbw is having the performance half to theoretical peak whereas actual performance is very less than the theoretical peak performance.
- For improvements I would suggest that we should increase the block size to get more throughput. Also, we can increase the concurrency i.e. we can try the experiment with the increasing the number of threads for higher throughput.

Benchmark results: Latency

Work-load	Con-currency	Block Size	MyRAMBench Measured Latency (us)	pmbw Measured Latency (us)	Theoretical Latency (us)	MyRAMBench Efficiency (%)	pmbw Efficiency (%)
RWS	1	1B	0.001602	0.038	0.014	11.44	-271.43
RWS	2	1B	0.003579	0.040	0.014	25.56	-285.71
RWS	4	1B	0.001901	0.042	0.014	13.58	-300.00
RWR	1	1B	0.036514	0.028	0.014	-260.81	-200.00
RWR	2	1B	0.467021	0.037	0.014	-3335.86	-264.29
RWR	4	1B	0.567548	0.050	0.014	-4053.91	-357.14



Conclusion:

- From the graph we can see that latency for RWS is very less and for RWR latency is increasing as number of threads are increased.
- Also, in regards with the comparison of MyRAMBench Measured Latency to that achieved by your benchmark, pmbw, and the theoretical performance pmbw is having the latency is very less than to theoretical peak whereas actual latency is much more less than the theoretical peak performance.
- For improvements I would suggest that we should perform the operations to find some number for concurrency to get the minimum latency.

3. Disk Benchmarking

- I used C language for Disk benchmarking the program. Three block sizes are considered while testing data that is 1MB, 10MB, 100MB and calculated for 1, 2, 4 Threads.
- I used several functions while designing this benchmark. My benchmark has many functionalities like I did strong scaling and we used pthread for thread synchronization. I include four operations
    - RS: read with sequential access pattern
    - WS: write with sequential access pattern
    - RR: read with random access pattern
    - WS: write with random access pattern
- I took some value defined in the program while initializing variables like fixed work size, Number of Threads, Block sizes.
- I make one struct variable named readwriteinfo which stores the data for each thread like start_block and end_block to maintain starting and ending point for memory reading or writing from buffer char array. Also, readwriteinfo also uses store info like block size and operation type for each thread.
- I am using strong scaling in this benchmark, I took fixed amount of work and reduce the amount of works per thread as we increase number of threads.

Design of code

- In the main function where I called method myRRFunc (), myRSFunc(),myWRFunc(),myWSFunc for throughput myRRFuncLat(), myWRFuncLat() for latency.Output is printed in DISKOUTPUT.txt and DISKLATENCYOUTPUT.txt for throughput and latency respectively in output folder.
- I called all this functions for multiple block sizes and multiple threads values.I have used pthread for thread synchronization, pthread_create to create thread and pthread join which waits for thread termination.
- I used fseek() function to Position the file pointer in anticipation of the next read or write .I also measure 2 more things Throughput in Megabyte per sec and latency in micro second.
- I used lseek() for sequential write access, random write access in which I use memory allocated buffer variable and data like block size etc.
- I also used read () and write() function to read and write the data from the file.

Theoretical Performance for Memory:

Throughput RS for 1 thread = 540 for 2 threads = 1080 for 4 threads = 2160

WS for 1 thread = 410 for 2 threads = 820 for 4 threads = 1640

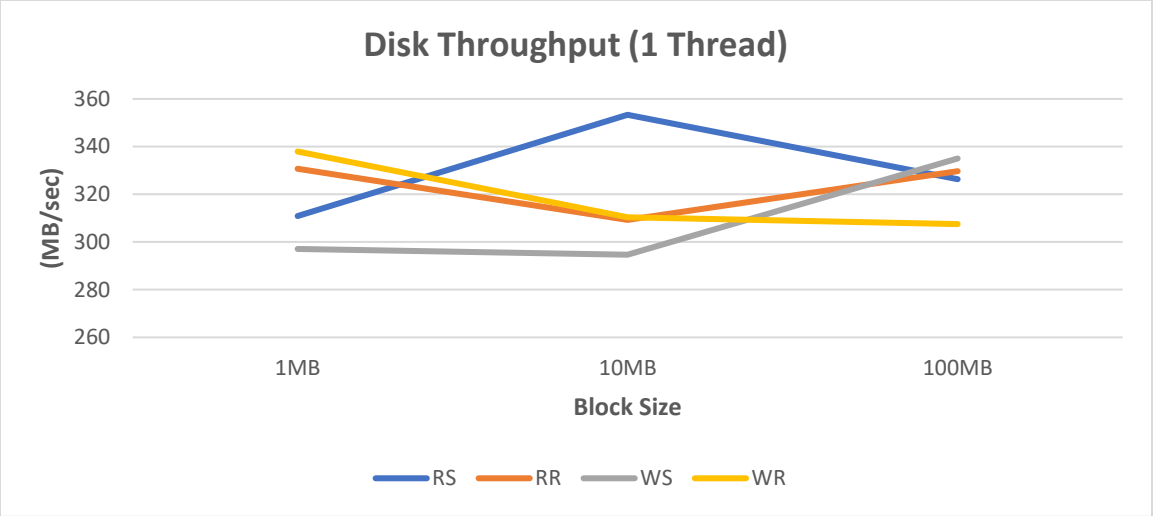
RR for 1 thread = 373 for 2 threads = 744 for 4 threads = 1488

WR for 1 thread = 172 for 2 threads = 344 for 4 threads = 688 (all from the chameleon specification site)

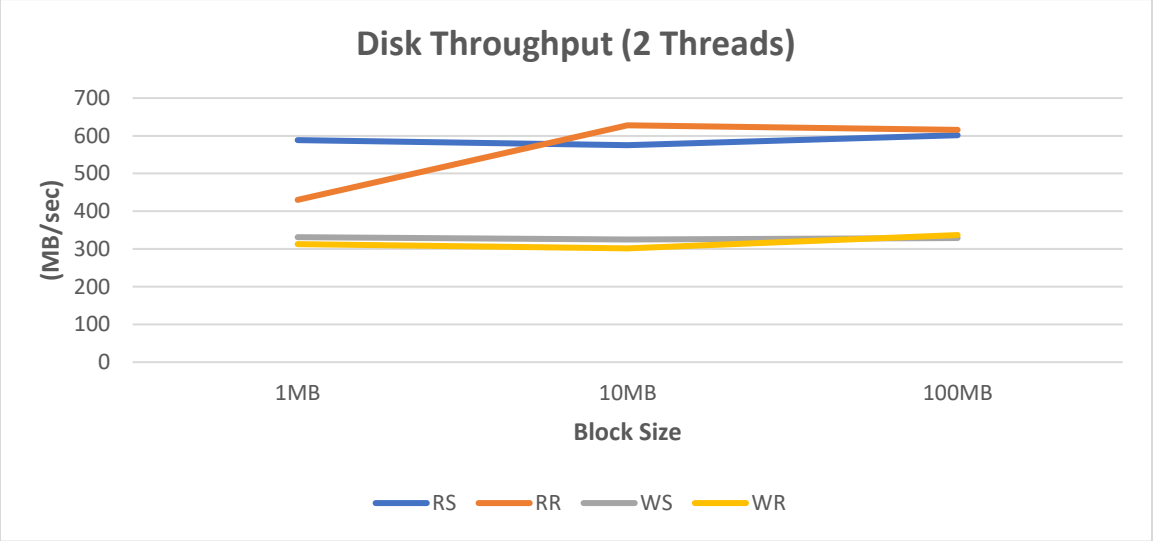
Latency = time for reading/ writing the 1kb data on to the disk.

Disk throughput

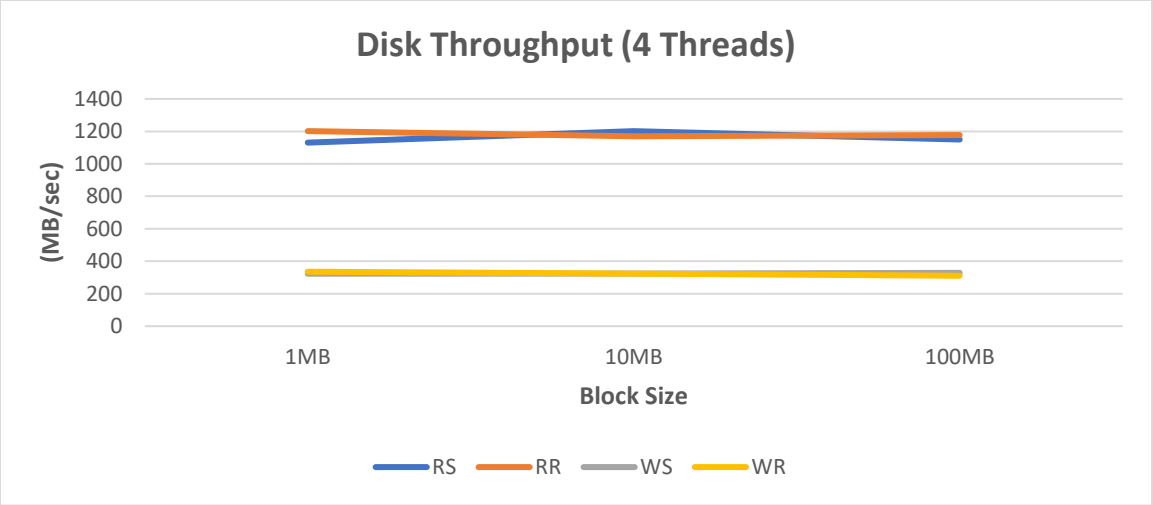
Work-load	Con-currency	Block Size	MyDiskBench Measured Throughput (MB/sec)	IOZone Measured Throughput (MB/sec)	Theoretical Throughput (MB/sec)	MyDiskBench Efficiency (%)	IOZone Efficiency (%)
RS	1	1MB	310.878155	335	540	57.57	62.04
RS	1	10MB	353.334568	327	540	65.43	60.56
RS	1	100MB	326.308143	324	540	60.43	60.00
RS	2	1MB	588.216693	681	1080	54.46	63.06
RS	2	10MB	575.121005	301	1080	53.25	27.87
RS	2	100MB	601.566677	698	1080	55.70	64.63
RS	4	1MB	1130.671876	1280	2160	52.35	59.26
RS	4	10MB	1201.346834	1298	2160	55.62	60.09
RS	4	100MB	1150.582821	1285	2160	53.27	59.49
WS	1	1MB	297.138938	331	410	72.47	80.73
WS	1	10MB	294.661773	295	410	71.87	71.95
WS	1	100MB	335.012521	335	410	81.71	81.71
WS	2	1MB	331.262372	375	820	40.40	45.73
WS	2	10MB	325.154441	462	820	39.65	56.34
WS	2	100MB	329.425158	528	820	40.17	64.39
WS	4	1MB	323.203107	349	1640	19.71	21.28
WS	4	10MB	324.680788	364	1640	19.80	22.20
WS	4	100MB	328.514576	385	1640	20.03	23.48
RR	1	1MB	330.704152	192	372	88.90	51.61
RR	1	10MB	309.266608	313	372	83.14	84.14
RR	1	100MB	329.695274	142	372	88.63	38.17
RR	2	1MB	430.176328	448	744	57.82	60.22
RR	2	10MB	627.757881	695	744	84.38	93.41
RR	2	100MB	616.045505	689	744	82.80	92.61
RR	4	1MB	1201.519068	688	1488	80.75	46.24
RR	4	10MB	1169.886215	1201	1488	78.62	80.71
RR	4	100MB	1177.160633	1345	1488	79.11	90.39
WR	1	1MB	337.942741	303	172	196.48	176.16
WR	1	10MB	310.329	339	172	180.42	197.09
WR	1	100MB	307.50	329	172	178.78	191.28
WR	2	1MB	312.657464	512	344	90.89	148.84
WR	2	10MB	301.57	495	344	87.67	143.90
WR	2	100MB	336.957	420	344	97.95	122.09
WR	4	1MB	335.651510	551	688	48.79	80.09
WR	4	10MB	324.43	395	688	47.16	57.41
WR	4	100MB	310.57	406	688	45.14	59.01



- WE can see that the RS performance is increasing as the block size is increasing and for RR,WS,WR as we increase the block size the throughput is decreasing.



- We can see that the throughput for RR is increasing for as block size is increasing and for rest RS,WR its decreasing where as for WS it is almost constant.



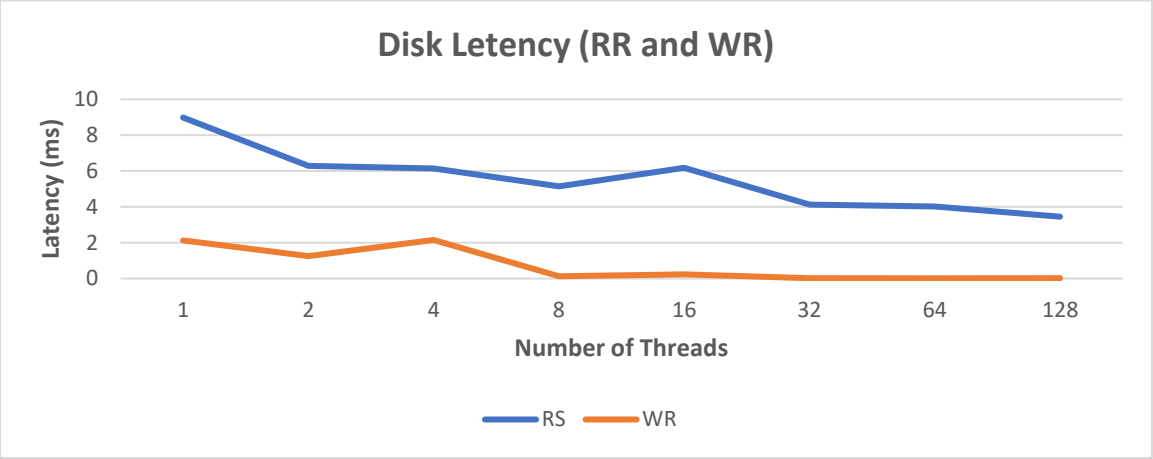
Conclusion:

- From the graphs we can see that throughput is increasing number of threads increases. So, for we could obtain the higher throughput with increase in the concurrency.
- Also, in regards with the comparison of My RAM Bench Measured to that achieved by your benchmark, IOZone , and the theoretical performance IOZone is having the performance almost near to theoretical peak whereas actual performance somewhat less than the theoretical peak performance.
- For improvements I would suggest that we should increase the block size to get more throughput. Also, we can increase the concurrency i.e. we can try the experiment with the increasing the number of threads for higher throughput.

Disk Latency (measured in ms)

Work-load	Con-currency	Block Size	MyDiskBench Measured Latency (ms)	IOZone Measured Latency (ms)	Theoretical Latency (ms)	MyDiskBench Efficiency (%)	IOZone Efficiency (%)
RR	1	1KB	8.979	5.914	0.5	5.57	8.45
RR	2	1KB	6.288	4.125	0.25	3.98	6.06
RR	4	1KB	6.143	5.001	0.125	2.03	2.50
RR	8	1KB	5.1374	3.159	0.0625	1.22	1.98
RR	16	1KB	6.174	4.125	0.03125	0.51	0.76
RR	32	1KB	4.128	3.698	0.0156	0.38	0.42
RR	64	1KB	4.012	3.489	0.0078	0.19	0.22
RR	128	1KB	3.453	2.697	0.0039	0.11	0.14
WR	1	1KB	2.131	1.258	0.5	23.46	39.75
WR	2	1KB	1.246	1.159	0.25	20.06	21.57
WR	4	1KB	2.154	0.975	0.125	5.80	12.82
WR	8	1KB	0.125	0.84	0.0625	50.00	7.44

WR	16	1KB	0.234	0.726	0.03125	13.35	4.30
WR	32	1KB	0.019	0.209	0.0156	82.11	7.46
WR	64	1KB	0.017	0.009	0.0078	45.88	86.67
WR	128	1KB	0.02	0.005	0.0039	19.50	78.00

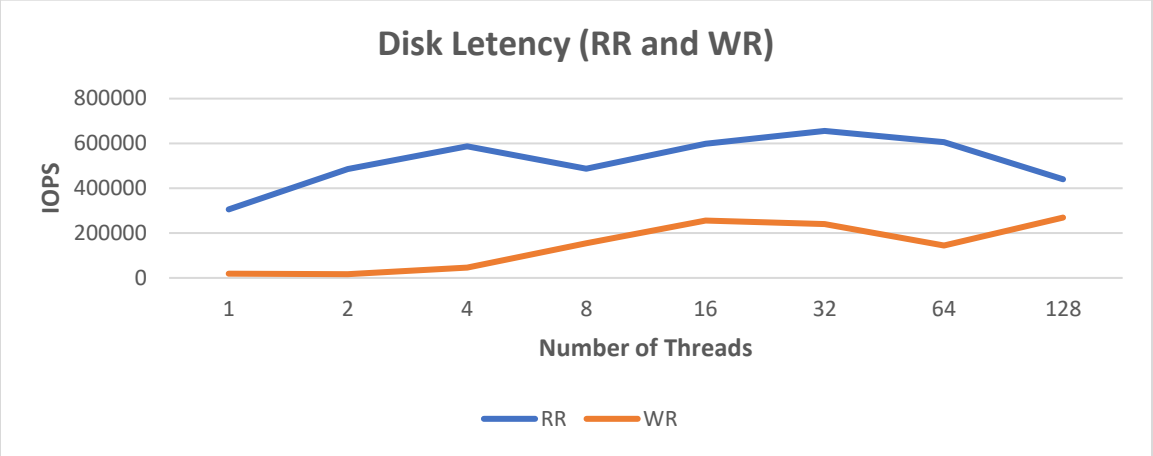


- From the graph we can see that latency for RR is higher than WR. Also, for RR latency is decreasing as number of threads increases and for WR beyond 8 threads latency is almost zero.
- Also, in regards with the comparison of Actual Measured Latency to that achieved by your benchmark, IOZone benchmark , and the theoretical performance IOZone is having the latency is very less than to theoretical peak whereas actual latency is much more less than the theoretical peak performance.
- For improvements I would suggest that we should perform the more operations to find some number for concurrency to get the minimum latency.

Disk Latency (measured in IOPS)

Work-load	Con-currency	Block Size	MyDiskBench Measured IOPS	IOZone Measured IOPS	Theoretical IOPS	MyDiskBench Efficiency (%)	IOZone Efficiency (%)
RR	1	1KB	305687	358952	384500	79.50	93.36
RR	2	1KB	486353	657894	769000	63.24	85.55
RR	4	1KB	586768	1456329	1538000	38.15	94.69
RR	8	1KB	487806	1789654	3076000	15.86	58.18
RR	16	1KB	598801	1645987	6152000	9.73	26.76
RR	32	1KB	655772	1923654	12304000	5.33	15.63
RR	64	1KB	606415	2158964	24608000	2.46	8.77
RR	128	1KB	439839	2314589	49216000	0.89	4.70
WR	1	1KB	18256	65145	163200	11.19	39.92
WR	2	1KB	16617	74895	326400	5.09	22.95

WR	4	1KB	46468	631587	652800	7.12	96.75
WR	8	1KB	154224	812593	1305600	11.81	62.24
WR	16	1KB	255791	795412	2611200	9.80	30.46
WR	32	1KB	240810	745698	5222400	4.61	14.28
WR	64	1KB	144056	845621	10444800	1.38	8.10
WR	128	1KB	269215	851479	20889600	1.29	4.08



- From the graph we can see that latency for RR is higher than WR.
- Also, in regards with the comparison of Actual Measured Latency to that achieved by your benchmark, IOZone benchmark , and the theoretical performance IOZone is having the latency is almost equal to theoretical peak whereas actual latency is less than the theoretical peak performance.
- For improvements I would suggest that we should perform the more operations to find some number for concurrency to get the minimum latency.

3. Network Benchmarking

- I used C language for Disk benchmarking the program. I have used separate files for Client and Server of TCP and UDP socket. Three block sizes are considered while testing data that is 1KB, 32KB and calculated for 1, 2, 4,8 Threads.
- I used several functions while designing this benchmark. My benchmark has many functionalities like I did strong scaling and we used pthread for thread synchronization. I include four operations
- I took some value defined in the program while initializing variables like fixed work size, Number of Threads, Block sizes.
- I make one struct variable named th_info_c which stores the data for each thread like start_block and end_block to maintain starting and ending point for memory reading or writing from buffer char array. Also, th_info_c also uses store info like block size and operation type for each thread.
- I am using strong scaling in this benchmark, I took fixed amount of work and reduce the amount of works per thread as we increase number of threads.

Design of Code:

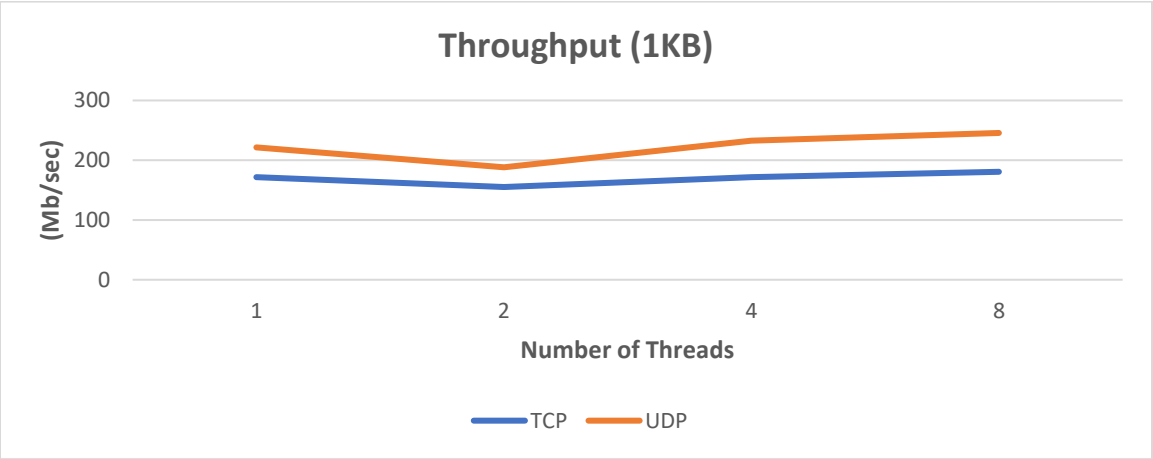
- For MyNETBench-TCP-Client class, in the main function where I called method client_tcp() for throughput and latency. Output is printed in NETWORKOUTPUT.txt and NETWORKLATENCYOUTPUT.txt for throughput and latency respectively in output folder.
- For MyNETBench-TCP-Server class, in the main function where I called method server_tcp () for connecting with the different client sockets.
- For MyNETBench-UDP-Client class, in the main function where I called method udp_client () for throughput and latency. Output is printed in NETWORKOUTPUT.txt and NETWORKLATENCYOUTPUT.txt for throughput and latency respectively in output folder.
- For MyNETBench-UDP-Server class, in the main function where I called method server_tcp () for connecting with the different client sockets.
- I called all this functions for multiple block sizes and multiple threads values.I have used pthread for thread synchronization, pthread_create to create thread and pthread join which waits for thread termination.

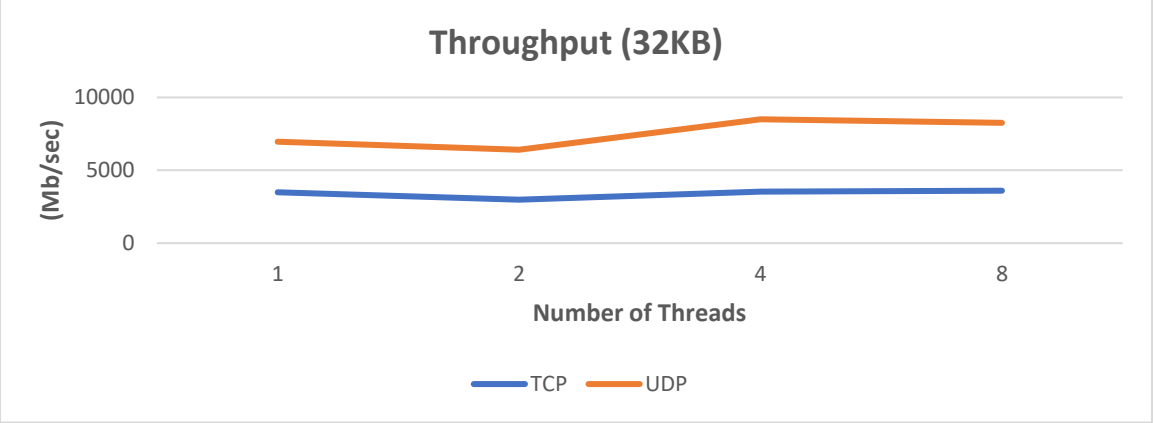
- **Theoretical Performance for Memory:**
  Theoretical Throughput = 10000 Mb/Sec (as per the specs from piazza)
  Theoretical Latency = 0.0005 ms

Network throughput:

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	171.785689	1759	10000.000000	1.72	17.59
TCP	1	32KB	3483.508800	5249	10000.000000	34.84	52.49
TCP	2	1KB	155.140482	6895	10000.000000	1.55	68.95
TCP	2	32KB	2982.146414	6925	10000.000000	29.82	69.25

TCP	4	1KB	171.592156	4856	10000.000000	1.72	48.56
TCP	4	32KB	3538.602392	5102	10000.000000	35.39	51.02
TCP	8	1KB	180.615356	4695	10000.000000	1.81	46.95
TCP	8	32KB	3599.688208	6259	10000.000000	36.00	62.59
UDP	1	1KB	221.310855	1985	10000.000000	2.21	19.85
UDP	1	32KB	6957.460123	7985	10000.000000	69.57	79.85
UDP	2	1KB	188.172880	6954	10000.000000	1.88	69.54
UDP	2	32KB	6412.097041	7563	10000.000000	64.12	75.63
UDP	4	1KB	232.521207	2489	10000.000000	2.33	24.89
UDP	4	32KB	8501.662427	9254	10000.000000	85.02	92.54
UDP	8	1KB	245.615418	5478	10000.000000	2.46	54.78
UDP	8	32KB	8259.489059	9856	10000.000000	82.59	98.56



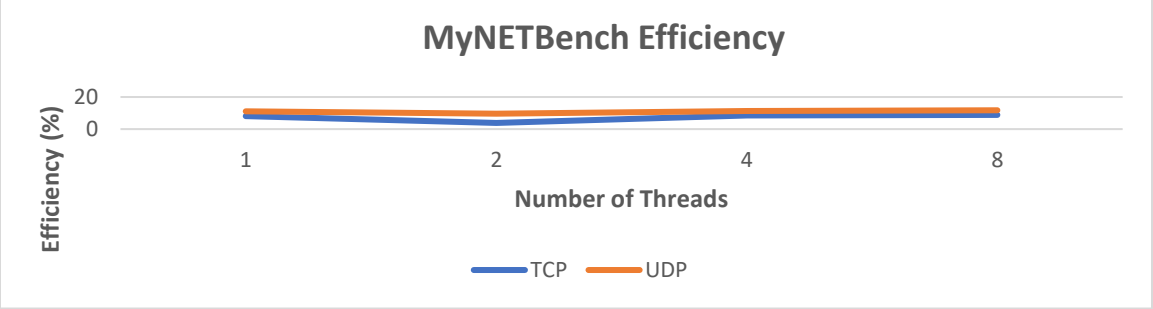


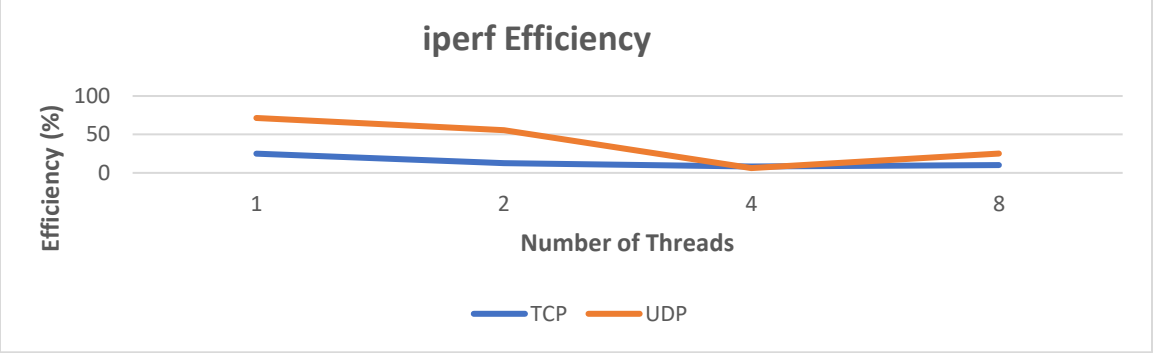
Conclusion:

- From the graphs we can see that of throughput of UDP is higher than throughput of TCP. Also, for both TCP and UDP the throughput value is increasing as number of threads increases.
- Also, in regards with the block size the throughput of both TCP and UDP is increased.

Network Latency:

Proto-col	Con-currency	Message Size	MyNETBench Measured Latency (ms)	ping Measured Latency (ms)	Theoretical Latency (ms)	MyNETBench Efficiency (%)	iperf Efficiency (%)
TCP	1	1B	0.006153	0.002	0.0005	8.13	25.00
TCP	2	1B	0.013015	0.004	0.0005	3.84	12.50
TCP	4	1B	0.005951	0.006	0.0005	8.40	8.33
TCP	8	1B	0.005653	0.005	0.0005	8.84	10.00
UDP	1	1B	0.004543	0.0007	0.0005	11.01	71.43
UDP	2	1B	0.005205	0.0009	0.0005	9.61	55.56
UDP	4	1B	0.004419	0.008	0.0005	11.31	6.25
UDP	8	1B	0.004264	0.002	0.0005	11.73	25.00





Conclusion:

- From the graphs we can see that of efficiency of UDP is higher than throughput of TCP. Also, for both TCP and UDP the efficiency value is decreasing as number of threads increases.
- Also, in regards with Actual efficiency, iperf efficiency and theoretical efficiency theoretical efficiency is higher than both Actual efficiency, iperf efficiency.