CS-553 CLOUD COMPUTING

PROGRAMMING ASSIGNMNET -1

BENCHMARKING

By

Teja Maripuri (A20376276)

Abhishek Vijhani (A20377670)

Design Document

We have designed 4 benchmarks:

1. **CPU Benchmarking (Abhishek Vijhani)**
2. **Memory Benchmarking (Teja Maripuri)**
3. **Disk Benchmarking (Teja Maripuri)**
4. **Network Benchmarking (Abhishek Vijhani)**

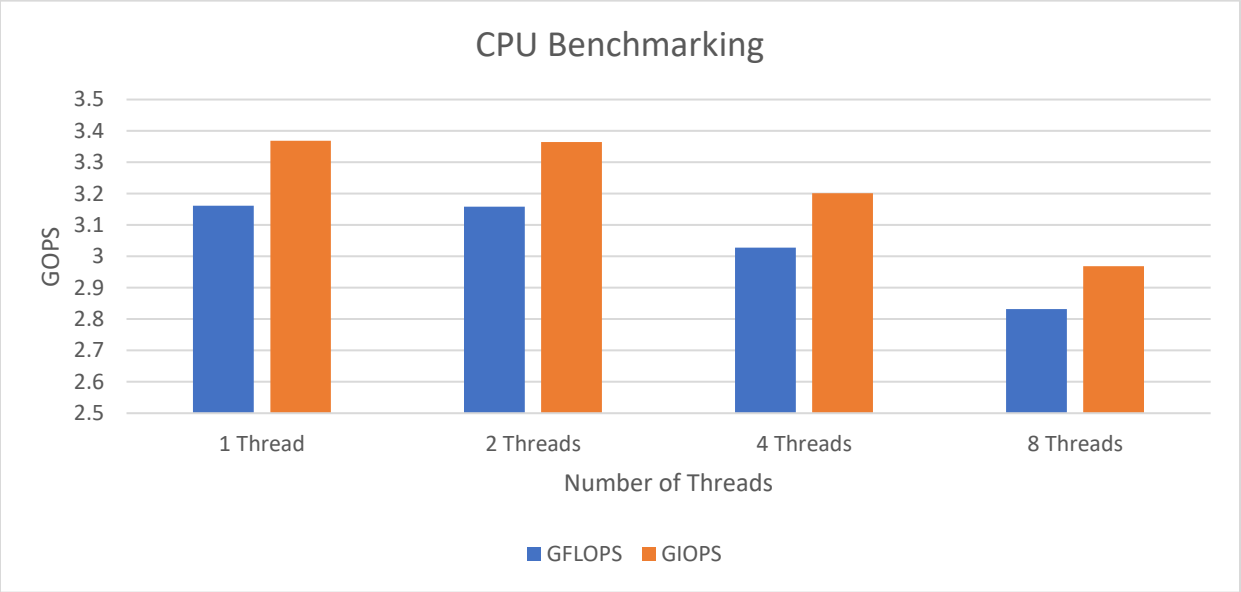
Note: All the following Experiments have been performed on the Baremetal Instance because we are not able to get the floating IP for the open stack KVM.

CPU Benchmarking

- We have written the program in C Language.
- We have implemented the multithread and parallel processing concept in C where the total work is divided among the respected number of threads which is entered by the user to calculate the FLOPS and IOPS one after the other.
- Strong Scaling has been implemented.
- We have also implemented AVX instructions to improve the number of operations per second by calculating the multiple operations in each cycle.

Results

Number of Threads	GFLOPS	GIOPS
1	3.161939	3.368602
2	3.158064	3.364204
4	3.028179	3.201218
8	2.831847	2.968871



Analysis/ Theoretical

The theoretical limit for the CPU Intel Xeon e5-2670 v3 used in Baremetal Instance is 883 GFLOPS. By performing Linpack benchmark, using huge problem size we were able to achieve 626.5 GFLOPS which means we achieved 70% efficiency using Linpack . The peak GFLOPS which we achieved in our benchmarking is 3.5 which is far less when we compare to the theoretical or linpack benchmarking. In the benchmarking which we have performed, we have taken simple AVX instructions but for the linpack the more complex AVX instruction have been taken into processing for achieving better GFLOPS. We can also observe that the GFLOPS and GIOPS have decreased while the threads increased but this could be changed by using complex AVX instructions.

Improvements/Extensions

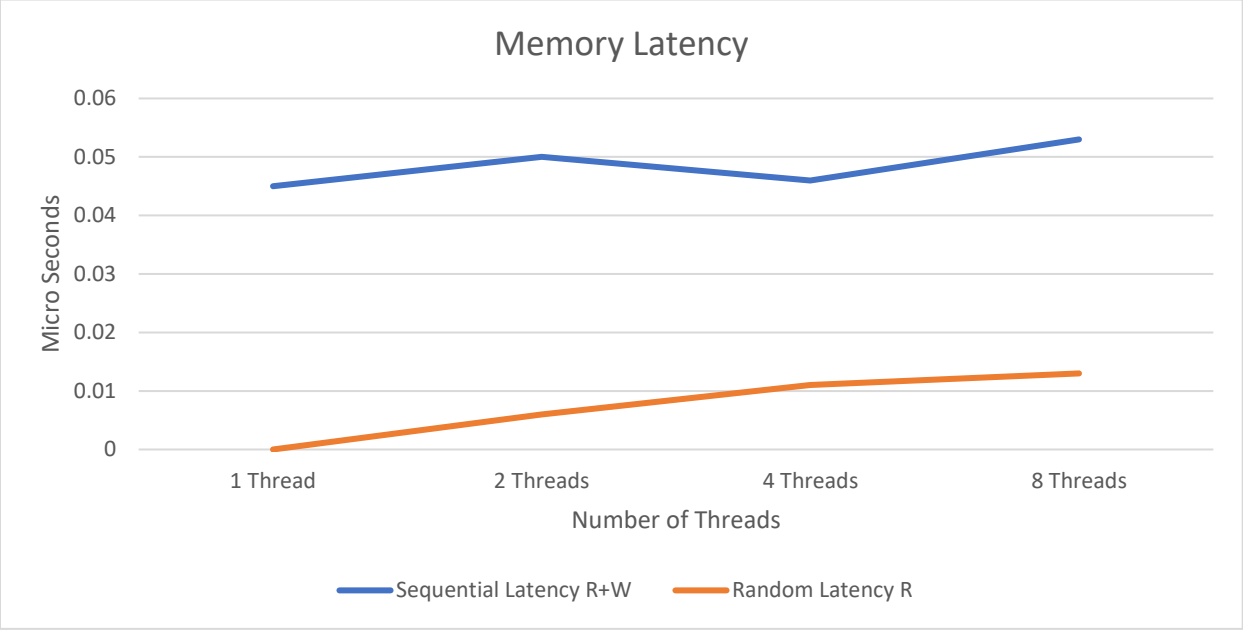
We can progressively increase number of AVX operations per cycle and their complexity to the point where the increase of number of operation per second is stagnated. Thus, achieving the peak performance of the CPU Benchmarking.

Memory Benchmarking

- We have written the program in C Language
- We have taken four different blocks sizes (8B, 8KB, 8MB, 80MB)
- We have implemented the multithread and parallel processing concept in C where the total work is divided among the respected number of threads on various block sizes.
- We will be calculating the throughput by measuring the total data processed per second with respect to the block size and we are calculating the latency using the 8B block.
- Random Access pointers would get block size data from the random parts of an array and Sequential Access pointers would work with each block size successively.
- Strong Scaling has been implemented.

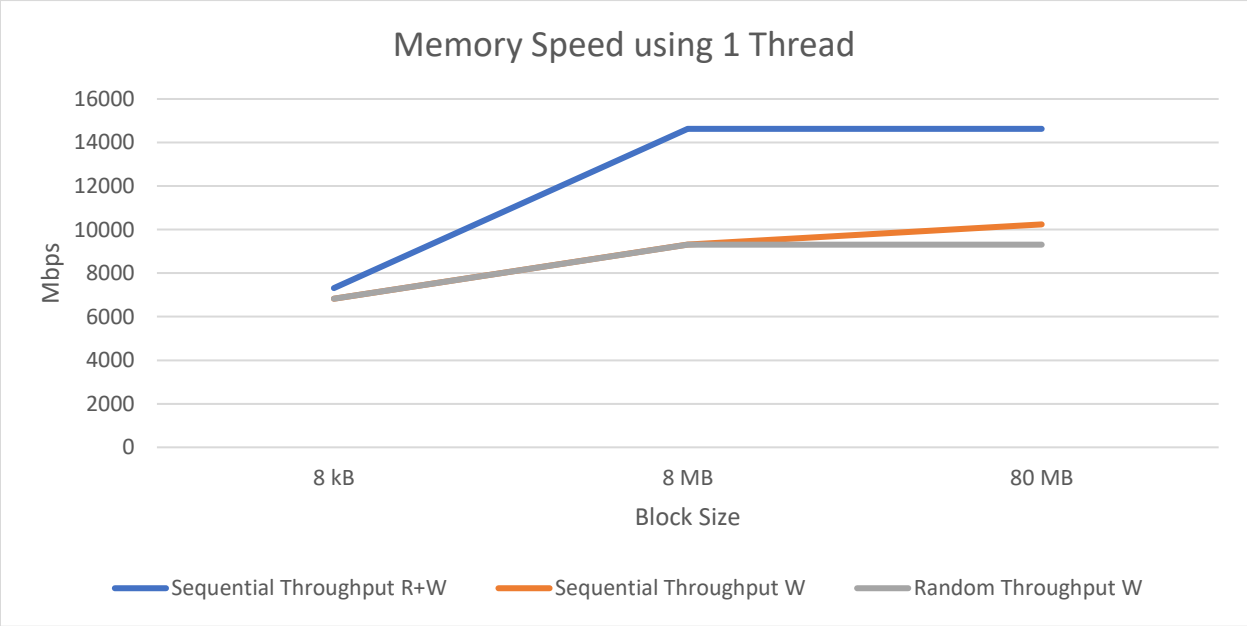
Results

Number of Threads	Sequential Latency R+W	Random Latency R
1	0.045	0
2	0.05	0.006
4	0.046	0.011
8	0.053	0.013



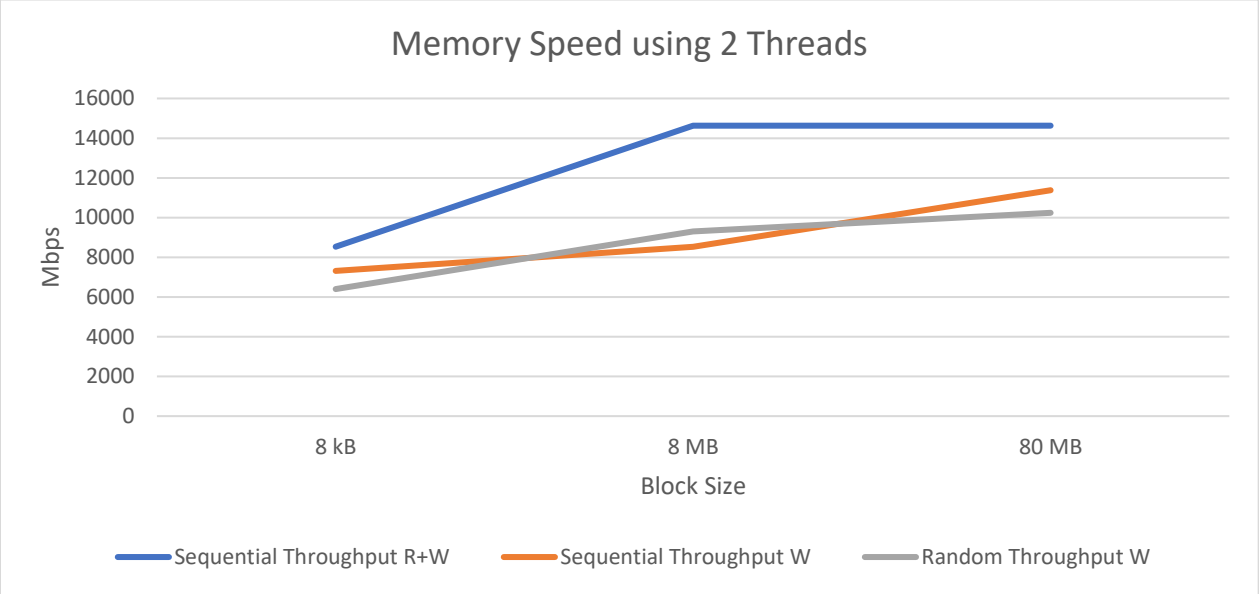
1 Thread

Block Sizes	Sequential Throughput R+W	Sequential Throughput W	Random Throughput W
8 kB	7314.29	6826.67	6826.67
8 MB	14628.57	9309.09	9309.09
80 MB	14628.57	10240	9309.09



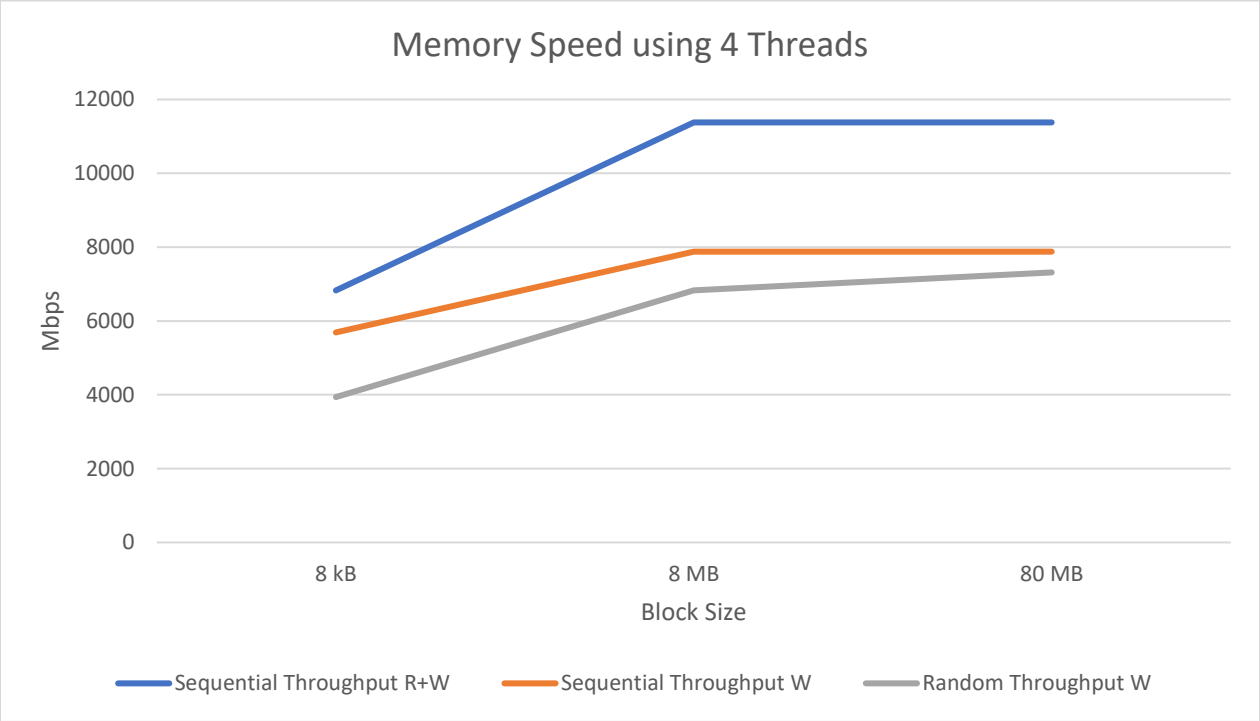
2 Threads

Block Sizes	Sequential Throughput R+W	Sequential Throughput W	Random Throughput W
8 kB	8533.33	7314.29	6400
8 MB	14628.57	8533.33	9309.09
80 MB	14628.57	11377.78	10240



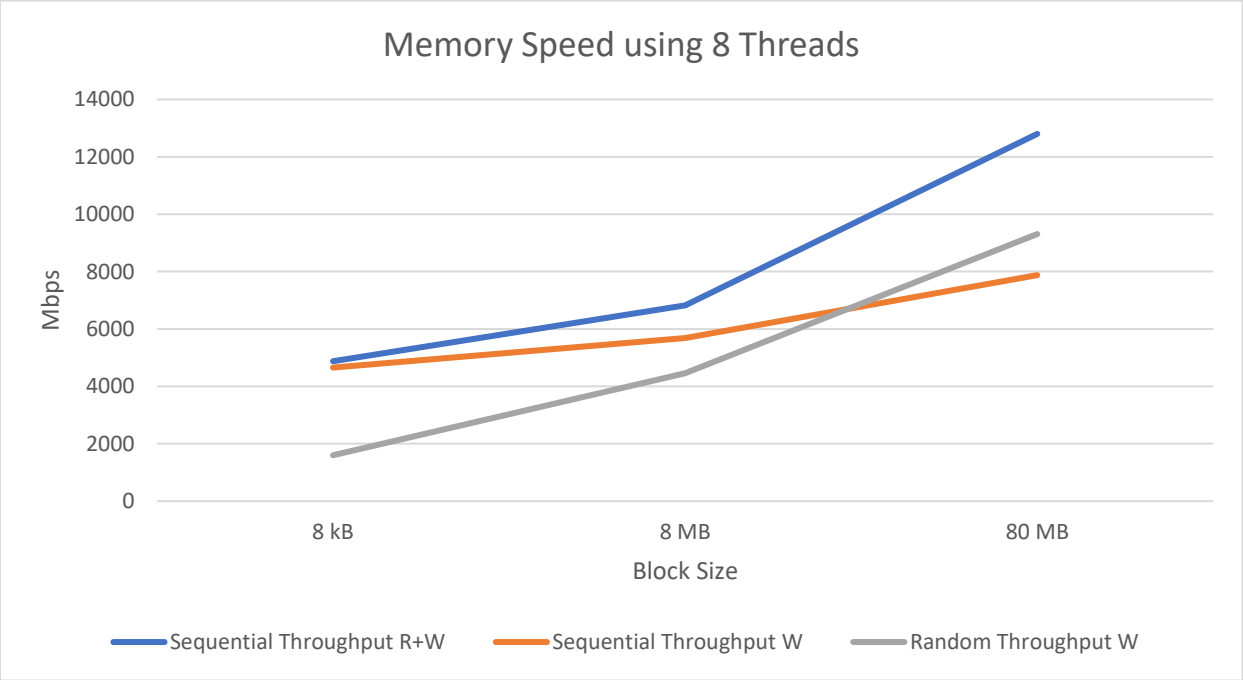
4 Threads

Block Sizes	Sequential Throughput R+W	Sequential Throughput W	Random Throughput W
8 kB	6826.67	5688.89	3938.46
8 MB	11377.78	7876.92	6826.67
80 MB	11377.78	7876.92	7314.29



8 Threads

Block Sizes	Sequential Throughput R+W	Sequential Throughput W	Random Throughput W
8 kB	4876.19	4654.55	1600
8 MB	6826.67	5688.89	4452.17
80 MB	12800	7876.92	9309.09



Analysis/ Theoretical

In the above graphs we can see that the latency has increased and throughput decreased with more threads which means 1/2 thread(s) was best. The throughput has increased with increasing size of block size for same number of thread which actually explains transferring few large chunks of data is better than a lot of small blocks. The theoretical performance limit for the memory is nearly 59 GB/s and the Stream benchmarking achieved is close 6 Mbps to 14 Mbps and we achieved similar results using our application. Even though we have less memory speed when compared to theoretical but a lot of processes are running which could affect the benchmarking readings.

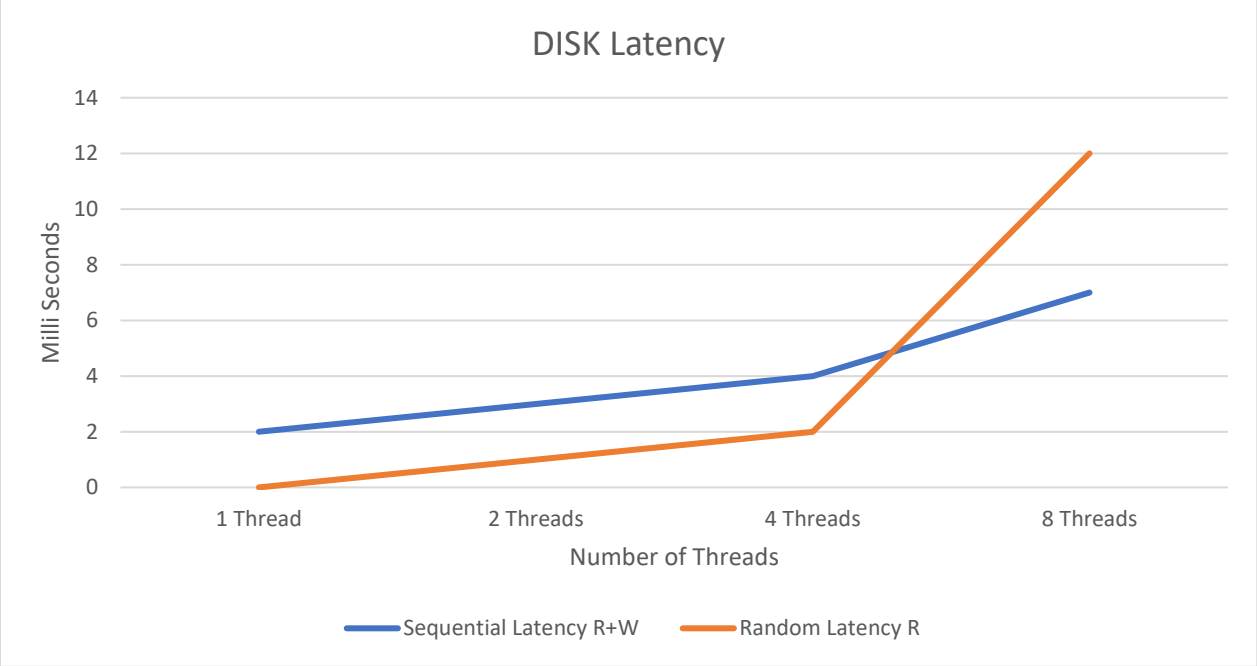
Improvements/Extensions

Due to the small virtual machine size, we are bound to take only one gigabyte of memory space and to perform experiments accordingly. If we have larger virtual machines, further extensive experiments can be possible.

Disk Benchmarking

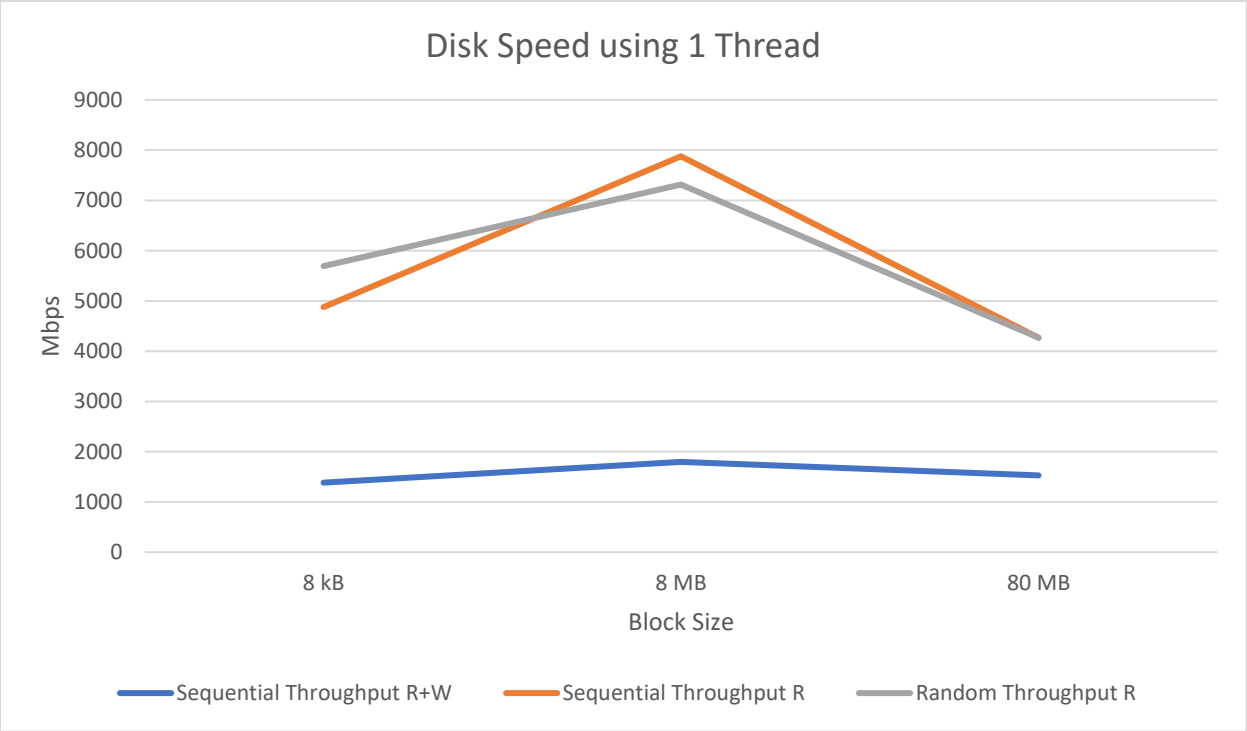
- We have written the code in C language.
- In this benchmarking we have created 1 Gigabyte of file using the write function.
- Then we are using the read+write function for reading from the original file and writing that file to a new file.
- We have implemented the multithread and parallel processing concept in C where the total work is divided among the respected number of threads on various block sizes.
- Our Sequential Access contains two methods read & write and write. Whereas, Random Access contains only read method.
- We will be calculating the throughput by measuring the total data processed per second with respect to the block size and we are calculating the latency using the 8B block.
- Strong Scaling has been implemented.

Number of Threads	Sequential Latency R+W	Random Latency R
1	2	0
2	3	1
4	4	2
8	7	12



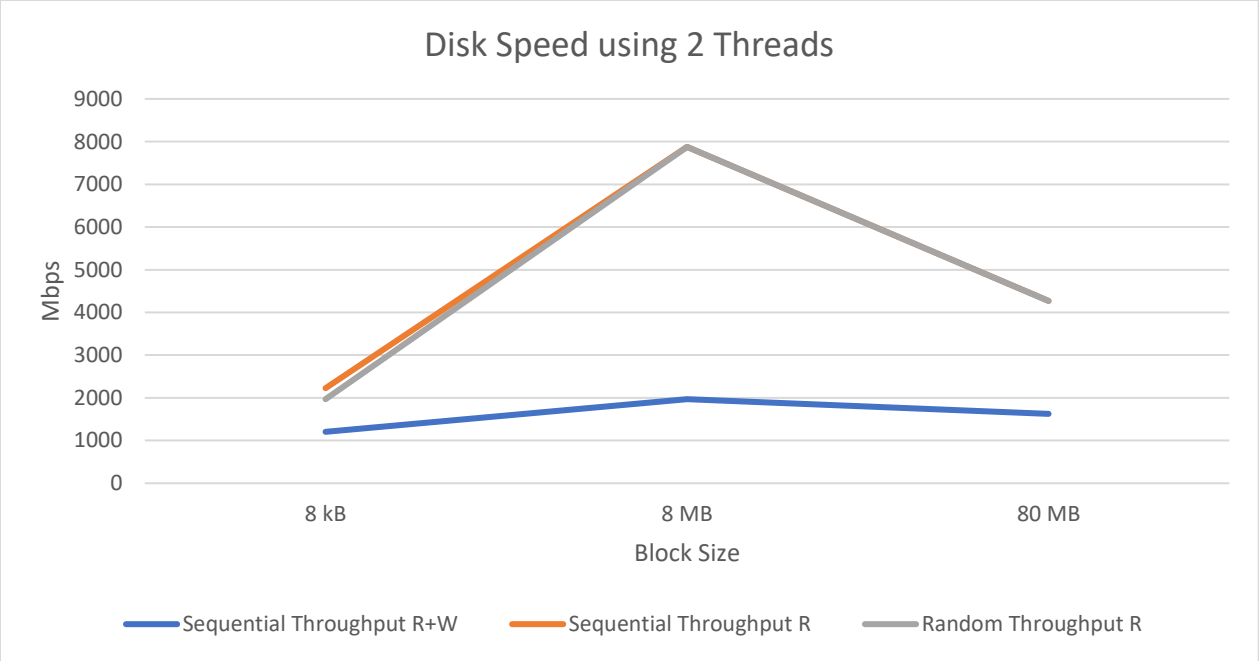
1 Thread

Block Sizes	Sequential Throughput R+W	Sequential Throughput R	Random Throughput R
8 kB	1383.78	4876.19	5688.89
8 MB	1796.49	7876.92	7314.29
80 MB	1528.36	4266.67	4266.67



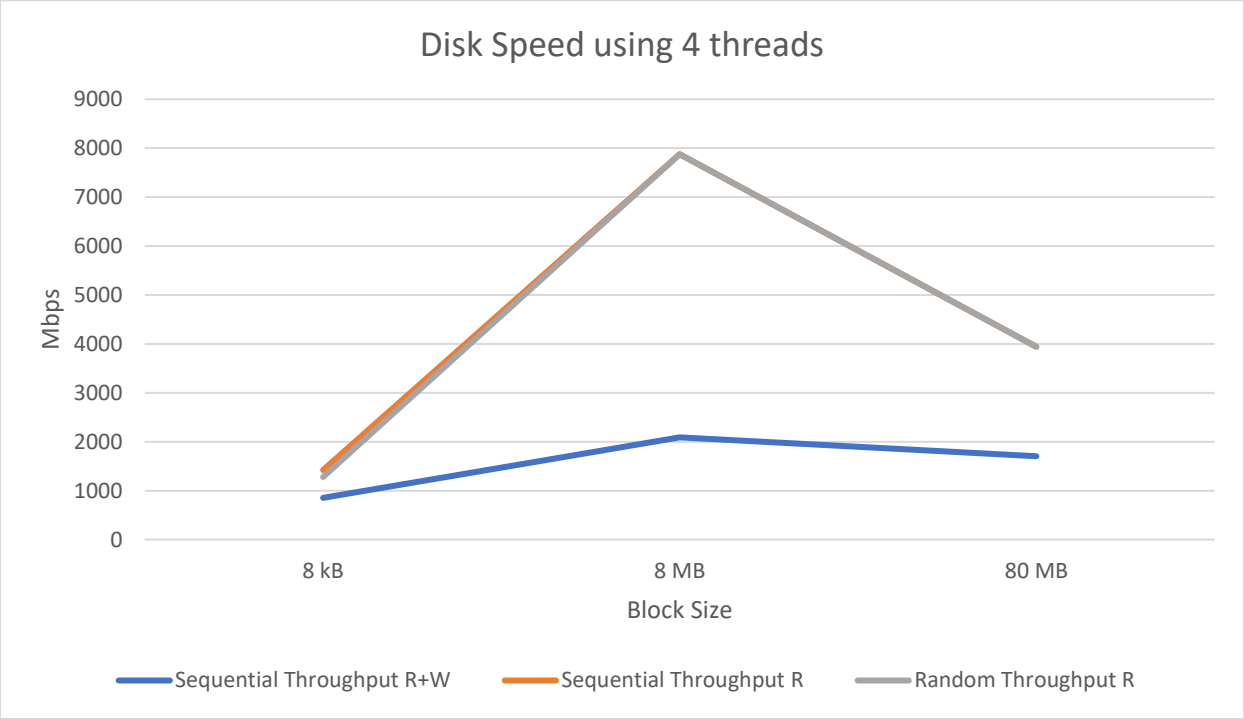
2 Threads

Block Size	Sequential Throughput R+W	Sequential Throughput R	Random Throughput R
8 kB	1204.71	2226.09	1969.23
8 MB	1969.23	7876.92	7876.92
80 MB	1625.4	4266.67	4266.67



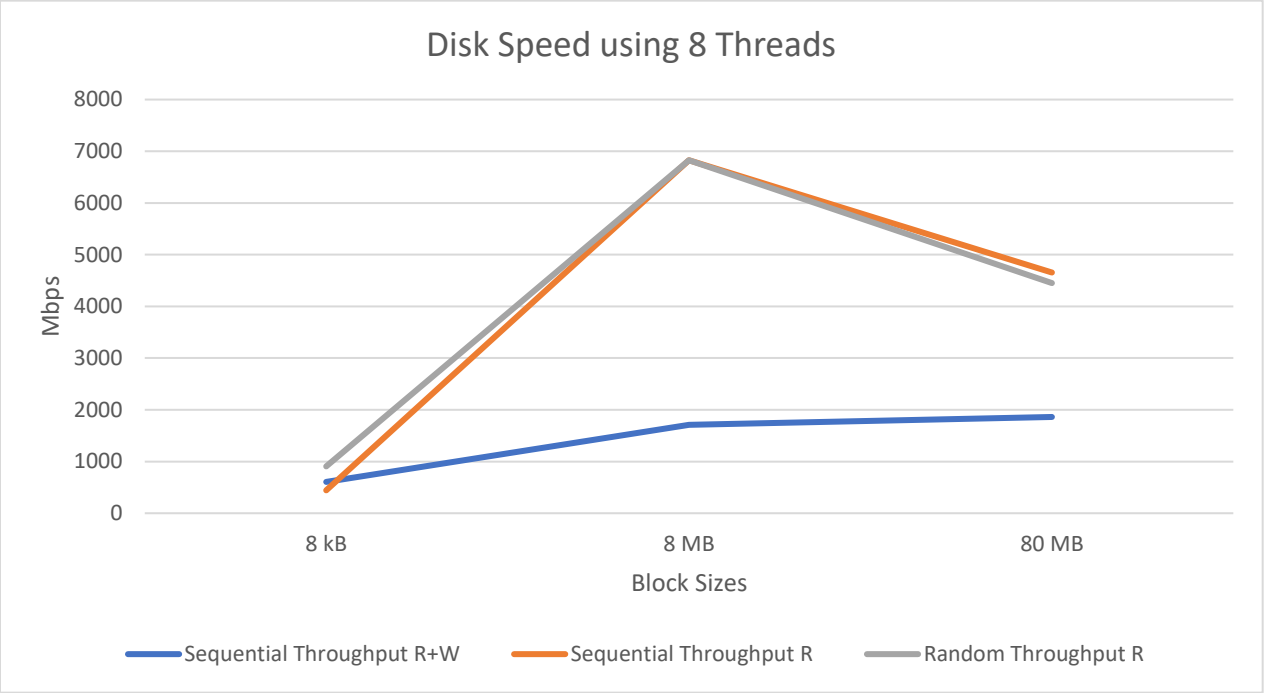
4 Threads

Block Size	Sequential Throughput R+W	Sequential Throughput R	Random Throughput R
8 kB	853.33	1422.22	1280
8 MB	2089.8	7876.92	7876.92
80 MB	1706.67	3938.46	3938.46



8 Threads

Block Sizes	Sequential Throughput R+W	Sequential Throughput R	Random Throughput R
8 kB	605.92	443.29	906.19
8 MB	1706.67	6826.67	6826.67
80 MB	1861.82	4654.55	4452.17



Analysis/ Theoretical

In the above graphs we can see that the latency has increased for every thread increases. The theoretical limit for the HDD in the baremetal instance is 6 GB/s where the IOZone benchmarking revealed to be around 4 GB/s thus we achieved 66% efficiency. In our benchmarking application we have achieved the scores of around 1-2 GB/s. When we see the throughput for various block sizes with same number of threads we can see that peak performance is archived when we performed with 8 MB block size.

Improvements/Extensions

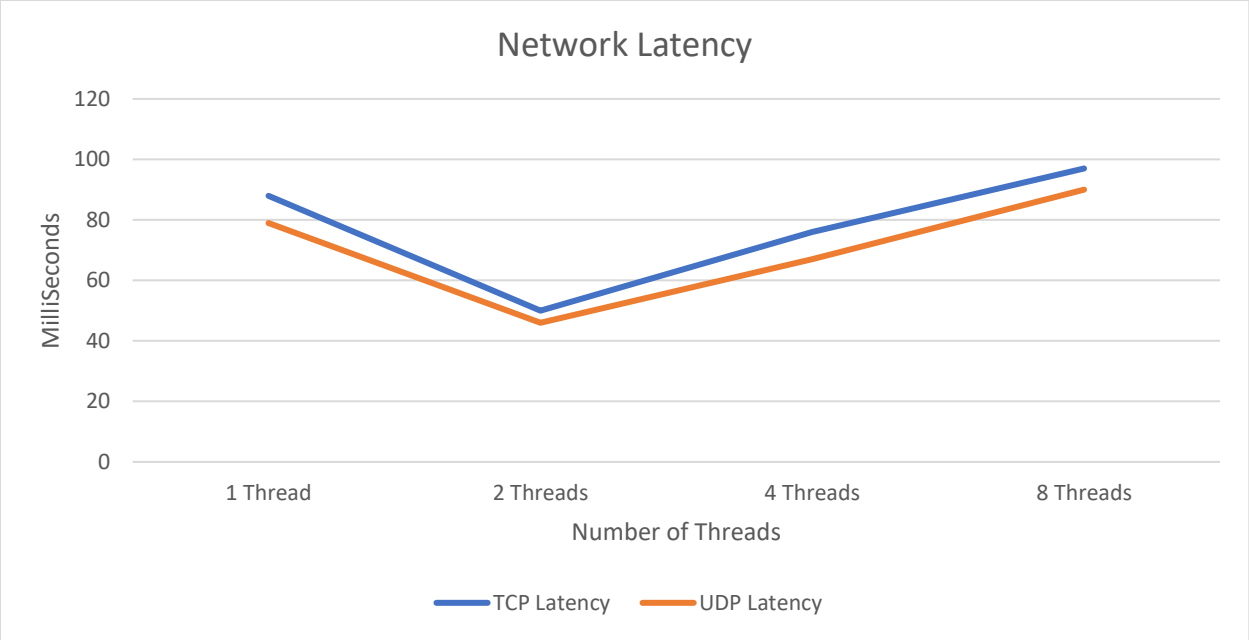
We are limited to the 1 gigabyte of memory due to the small Baremetal Instance thus further efficiency improvement to shunted.

Network Benchmarking

- We have written the code in C++ Language.
- We have also used the multithreading concept in C++. (1,2,4,8 Threads)
- This Benchmark is basically divided into two parts: TCP and UDP
- We are first invoking the TCP Client and Server and then UDP Client and Server
- We have first established a TCP Server with a listening socket and then the Client which sends the first message which is used for connection using a new socket on the server.
- After the successful connection, client send buffers filled with the data to find out the throughput and latency.
- We will do same steps for establishing the connection for UDP but it will just listen to the initial socket instead of making a new socket.
- We will be calculating the throughput by measuring the total data processed per second with respect to the block size which is 64Kb.
- Strong Scaling has been implemented.

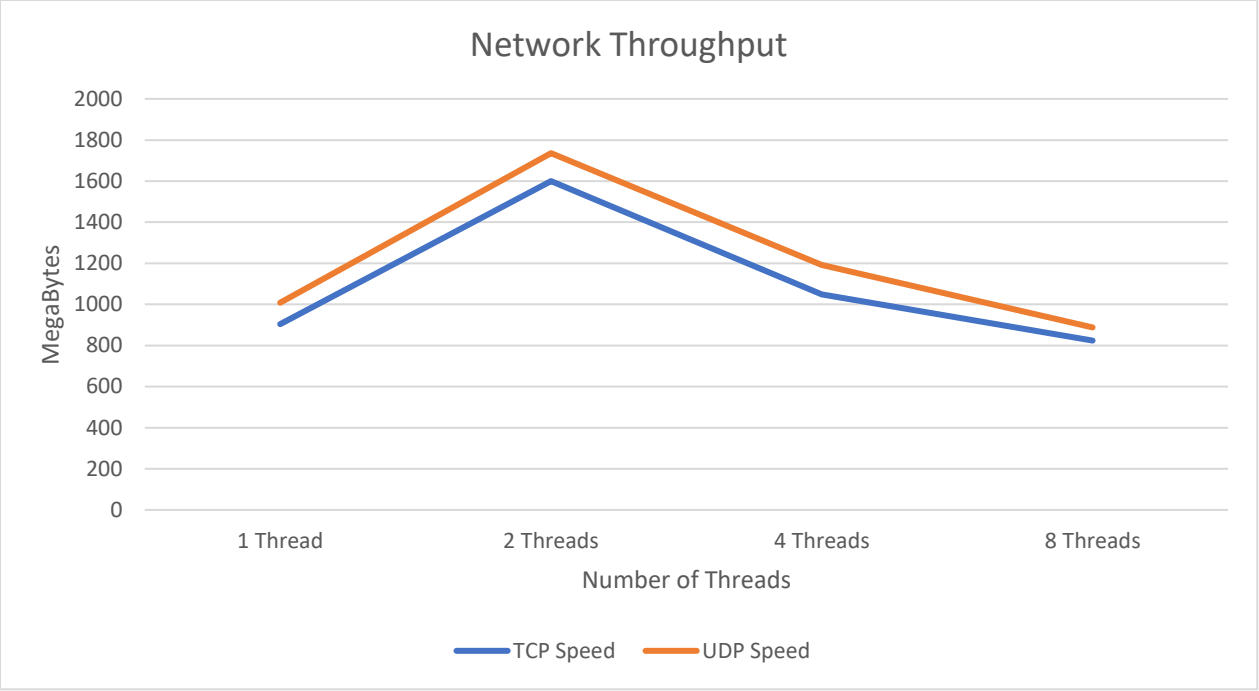
Latency

Number of Threads	TCP Latency	UDP Latency
1	88	79
2	50	46
4	76	67
8	97	90



Throughput

Number of Threads	TCP Speed	UDP Speed
1	904	1008
2	1600	1736
4	1048	1192
8	824	888



Analysis/ Theoretical

The Latency increases with number of threads increased. The theoretical limit for the Network card for the baremetal instance is 10 GB/s and in the Iperf benchmark we have achieved around 5 GB/s and in our network benchmarking program we have achieved nearly 1GB/s. The throughput has decreased when we increase number of threads.

Improvements/Extensions

- We can used unique ports so that we won't have any port collisions.
- To achieve the higher latency we can increase the buffer size and decrease the transfer delay.