Memory Benchmarking

Design Decisions:

- Strong scaling is performed by having the problem size in our case 1.3Gb fixed memory and varying the number of threads to execute (1, 2, 4 ,8) and number of blocks
- The program allocates memory to both source and target memory using malloc function
- To illustrate **read and write operation** we are using **memcpy**.
- To illustrate **sequential write** we are using **memset.**
- To illustrate **random write** also we are making use of function **memset**.
- The main method divides the memory for thread to be equal, meaning each thread will have same amount of memory to copy from source block to target memory block.
- Also, big memory of 1.3 GB is logically divided into small chunk based on block size, so that we can avoid collision of threads working on same piece of memory block.

1.Performance Results

Throughput Results:

Throughput results is calculated for all three operation that is Sequential Read/Write, Sequential Write and Random Write and the below table illustrates that.

From the tables below, we can observe that as the block size increases, we can observer that the throughput also increases for both sequential and random operations since bigger memory block copies a significantly larger data in a single operation.

a) Sequential Read/Write

Throughput readings for Sequential Read Write operations varying block sizes and varying concurrency.

	8KB	8MB	80MB
Threads - 1	1536.910590	2341.153846	2520.363636
Threads - 2	2536.017045	4634.666667	4432.363636
Threads - 4	2778.932292	5236.272727	5207.363636
Threads - 8	2654.932292	5136.428571	5189.666667

b) Sequential Write

Throughput readings for Sequential Write operations varying block sizes and varying concurrency.

	8KB	8MB	80MB
Threads - 1	1536.399639	4506.727273	3306.000000
Threads - 2	3425.017045	6740.800000	7340.666667
Threads - 4	5689.118750	6922.272727	7522.363636
Threads - 8	5536.084821	6856.461538	7456.666667

c) Random Write

Throughput readings for Random Write operations varying block sizes and varying concurrency.

	8KB	8MB	80MB
Threads - 1	7443.041016	9729.000000	6640.190476
Threads - 2	13827.490625	16668.190476	11532.925926
Threads - 4	13449.028906	15644.909091	14288.333333
Threads - 8	12362.857031	14456.043478	14048.000000

Latency

Latency results are calculated only for an 8Byte memory and varying the number of threads and calculating it for various operations.

In latency we see a different trend, as the block size increases we see increase in latency as each operation involved gets more and more expensive.

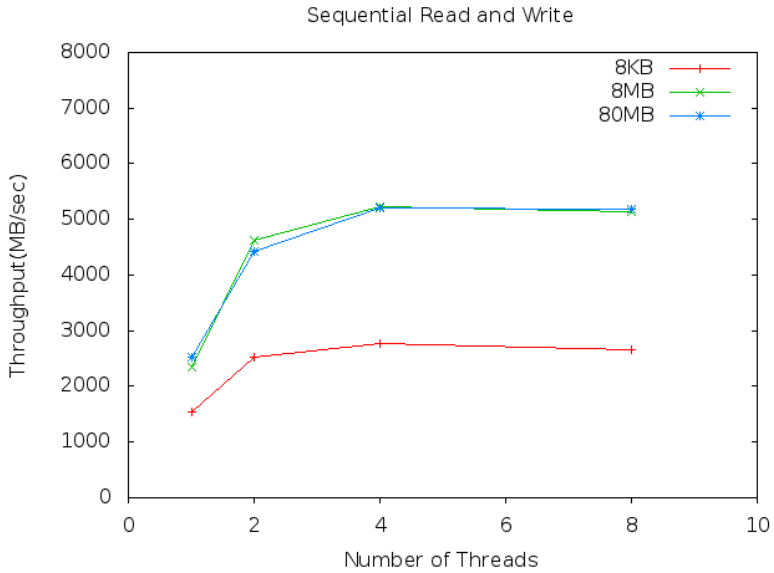
Readings for * Byte block size with varying operations and varying concurrency.

	Sequential Read/Write	Sequential Write	Latency
Threads - 1	0.103162	0.103162	0.123345
Threads - 2	0.103162	0.097431	0.248471
Threads - 4	0.103162	0.108893	0.321390
Threads - 8	0.097431	0.114799	0.256942

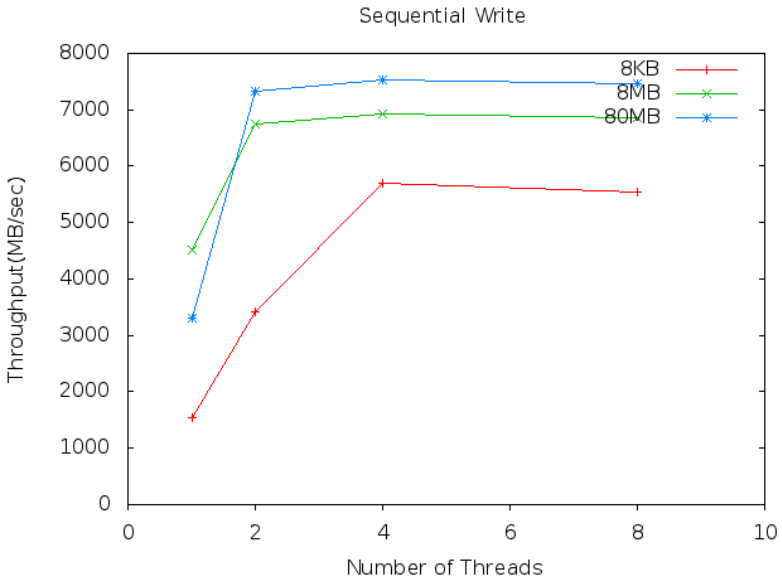
2.Performance Graph

Throughput

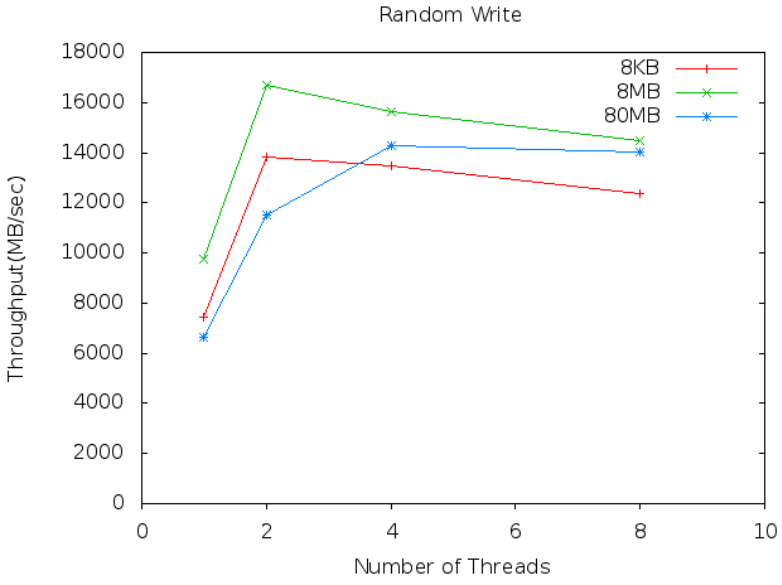
Throughput is calculated for 8KB,8MB,80MB by varying number of threads and operations.



Above graph is for plotting throughput(MB/sec) vs number of threads by varying different memory blocks. We can observe that throughput increases as we increase the number of threads and peaks at 4 threads.



Above graph is for plotting throughput(MB/sec) vs number of threads by varying different memory blocks. We can observe that throughput increases as we increase the number of threads and peaks at 8 threads. For 8 KB the throughput is low as there memory block is too small to calculate without overheads that are involved.



Above graph is for plotting throughput(MB/sec) vs number of threads by varying different memory blocks. We can observe that throughput increases as we increase the number of threads and peaks at 2 threads. Also, we can see that as we increase the threads the throughput is reducing.

Latency

Latency results are calculated only for an 8Byte memory and varying the number of threads and calculating it for various operations.

In scaled out graph in fig 1, and from zoomed in graph in fig -2, it is observed that for Random write operation, latency is huge and we also can see from the graph that for both sequential Read/Write operation and Sequential Write operation, latency increases when the number of threads are increased.

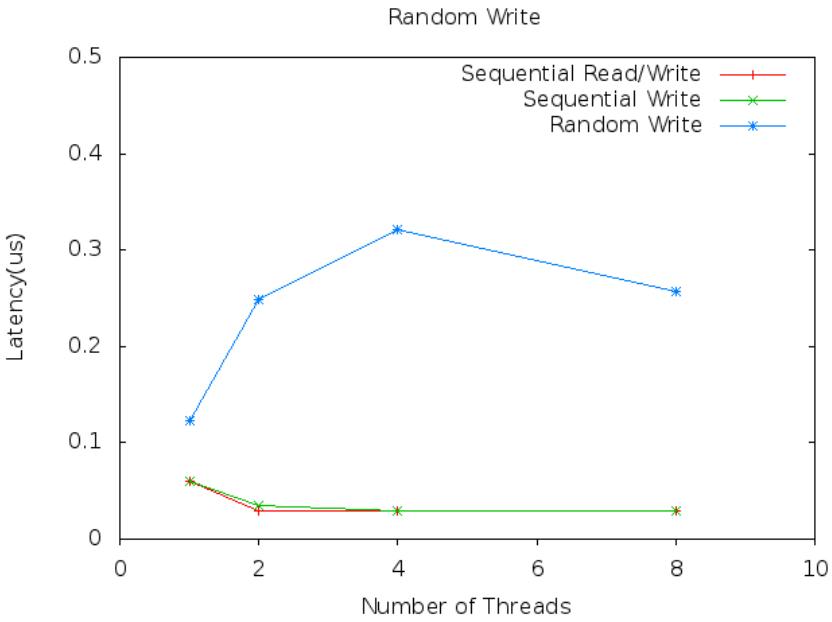


Fig 1- Scaled out graph for latency

3.Theoretical Peak Performance

Intel(r) Xeon (r) CPU E5-2670 v3 (formerly Haswell)
Max Memory Size (dependent on memory type) - 768 GB
Memory Types - DDR4 3.2Ghz
Max # of Memory Channels - 4
Max Memory Bandwidth - 68 GB/s
Instruction Set - 64 bit

Theoretical Maximum Memory Bandwidth is calculated using formula:
= RAM clock frequency * Memory Type(DDR4) * 64 bit Instruction Set
= 3.2 * 2 * 8(bytes)
= **51 GB/sec**

From the intel specification of the processor, and though the below link https://ark.intel.com/products/81709/Intel-Xeon-Processor-E5-2670-v3-30M-Cache-2_30-GHz we can see that the **Maximum bandwidth is 68 GB/s**
This number is higher than the one we have obtained in the result, since we run on KVM instances and the hardware performance of the host machine is much higher than the once given to each KVM machine.

4.Stream Benchmark

Below is the output obtained from stream benchmark. Based on the documentation by stream, it performs operations considering 8 bytes block size. **The best result obtained from stream for copy operation is 6098.8 MB/sec** which is nearer to what I obtained for 8KB of data.

```
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 1
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 33256 microseconds.
   (= 33256 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           6098.8     0.028200     0.026235     0.030877
Scale:          5879.3     0.029207     0.027214     0.031715
Add:            8536.6     0.030117     0.028114     0.032621
Triad:          8319.4     0.031016     0.028848     0.032983
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
```

5. Vary the number of threads and find the optimal number of concurrency to get the best performance

After varying the number of threads from the above graphs we can infer that throughput is the best and latency is lowest when the number of threads concurrently executing is 4.If we decrease the number of threads to 1 or 2 or if we increase the number of threads to 8, the throughput number is lower and latency is bit higher.

This result is consistent in all the operations such as Sequential Read/Write , Sequential Write and Random Write. Therefore, we can come to a conclusion that 4 threads are better to give the best performance.