Contents

1 Introduction

In computing, a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it. The term 'benchmark' is also mostly utilized for the purposes of elaborately designed benchmarking programs themselves. Benchmarking is usually associated with assessing performance characteristics of computer hardware. Benchmarks provide a method of comparing the performance of various subsystems across different chip/system architectures.

In this assignment, we aim to gauge the performance metrics of the memory and disk of a computer. Though the aim is to test the efficacy of the intended system on a virtual test bed, the following experiments could be run on any computer, personal or otherwise, to estimate its average effectualness and usability. The specifications of the computer are :

- Machine ID: 00c04eeb-ad2d-4c2d-aabf-a2fdbda5a752
- Number of CPUs: 2
- Processor: Intel Xeon at 2.3 Ghz(Base), 3.1 Ghz(Max Turbo)
- Number of cores: 24
- Number of threads: 48
- Memory: 128 GB

2 Memory

To measure the capabilities of the provided memory, a simple C program was written to perform three basic operations:

1. Sequential read and write
2. Sequential write
3. Random write

Each operation was launched with varying levels of concurrency (one, two, four and eight threads) and on four different block sizes: 8 Bytes, 8 Kilobytes, 8 Megabytes and 80 Megabytes in this case. The number of operations were manipulated so as to be able to work within the limit of a Gigabyte and half, and the metrics measured were throughput and latency. A well known benchmarking tool called Stream was also used to make comparisons with and draw conclusions from. It is to be noted that, ideally, a high throughput and low latency are considered desirable.

2.1 Program Design

The basic idea of the program is to perform either read+write, sequential write or random write using different threads and to measure the monotonic time taken for all the operations to finish.

Before processing any operations, source and destination char arrays are first allocated a fixed size of memory. For read+write, memcpy() is used to read from the source and put into the destination. For sequential writes, the destination is accessed in order of memory and is set to a constant value using memset(). For random writes, the same thing is essentially done but by accessing random portions of the destination's memory with the help of a random number generator (rand()). All these operations and passed on to POSIX threads that are created and joined at the requisite portion of the program. The parameters of the run are altered as per required by redefining its struct parameters and the time taken for all the operations to finish is calculated by utilizing the clock_gettime() function.

2.2 Theoretical bandwidth

The formula for calculation of the theoretical bandwidth of memory in MBps is as follows:

$$\frac{Memory\ clock \times Bus\ Width}{8} \times DDR\ type\ multiplier$$

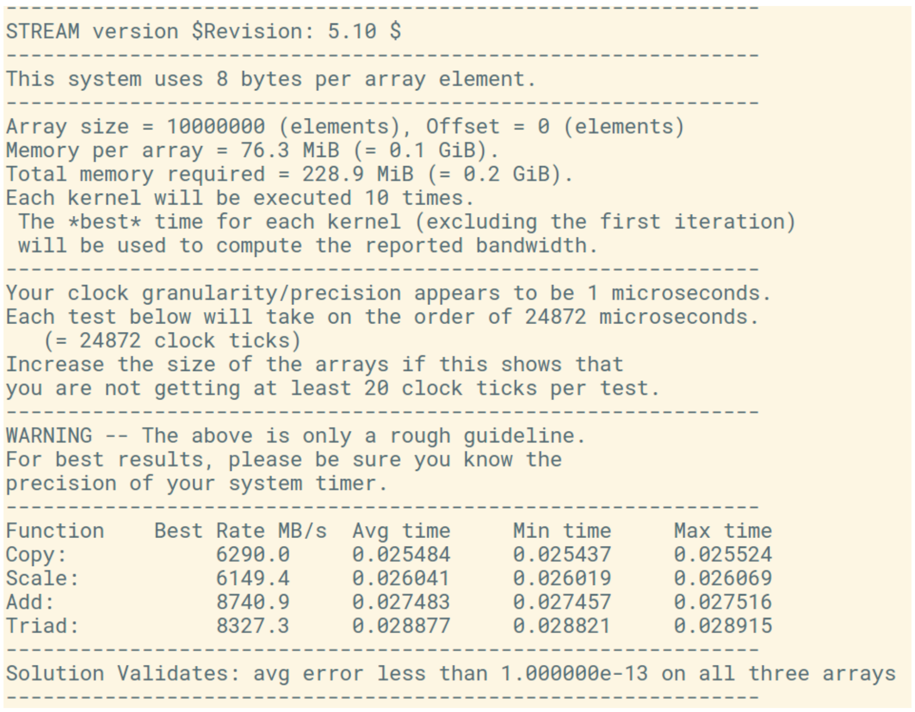
It is given that the memory is of type DDR4 with a clock speed of 2133 Mhz and a bus width of 64 bits. The type multiplier in this case would be 2. Hence, the bandwidth would be $\frac{2133 \times 64}{8} \times 2 =$ **34.1 GB/s**

2.3 Stream benchmark

The obtained result from the benchmarking tool provides us a maximum bandwidth of **6290.0 MBps** as shown in the screenshot below. The result can also be read by accessing stream_result.txt in the submission folder.

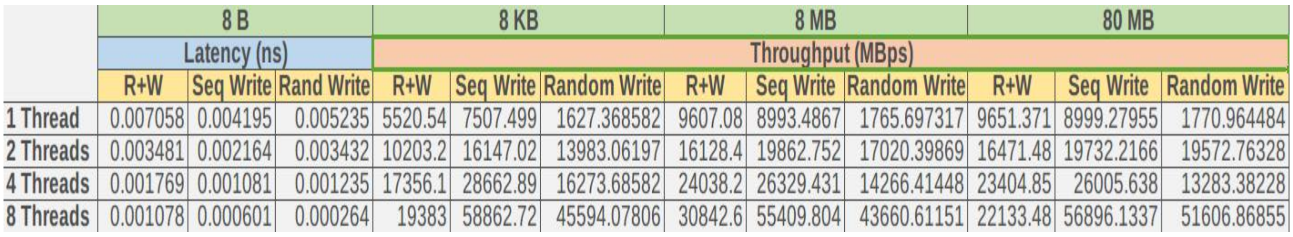
2.4 Results

The numbers shown below indicate that sequential write operations are much faster than random write operations. It can also be concluded that the throughput increases as the number of threads increase for bigger and smaller block sizes alike. For achieving peak performance, throughput and latency wise, the optimal number of concurrency appears to be **4**. Also, the attained bandwidth for read and write operation seems to be close to the maximum copy bandwidth of Stream, give or take a few hundred MBs per second. Average results of three runs of the experiment with a standard deviation of around 3% are shown below. Pictorial graphs indicating throughput/latency per thread based on strong scaling have also been provided.



```
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
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.
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 24872 microseconds.
   (= 24872 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           6290.0     0.025484     0.025437     0.025524
Scale:          6149.4     0.026041     0.026019     0.026069
Add:            8740.9     0.027483     0.027457     0.027516
Triad:          8327.3     0.028877     0.028821     0.028915
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

Figure 1: Result of the Stream benchmark



	8 B			8 KB			8 MB			80 MB		
	Latency (ns)			Throughput (MBps)								
	R+W	Seq Write	Rand Write	R+W	Seq Write	Random Write	R+W	Seq Write	Random Write	R+W	Seq Write	Random Write
1 Thread	0.007058	0.004195	0.005235	5520.54	7507.499	1627.368582	9607.08	8993.4867	1765.697317	9651.371	8999.27955	1770.964484
2 Threads	0.003481	0.002164	0.003432	10203.2	16147.02	13983.06197	16128.4	19862.752	17020.39869	16471.48	19732.2166	19572.76328
4 Threads	0.001769	0.001081	0.001235	17356.1	28662.89	16273.68582	24038.2	26329.431	14266.41448	23404.85	26005.638	13283.38228
8 Threads	0.001078	0.000601	0.000264	19383	58862.72	45594.07806	30842.6	55409.804	43660.61151	22133.48	56896.1337	51606.86855

Figure 2: Average results of three runs of experiments

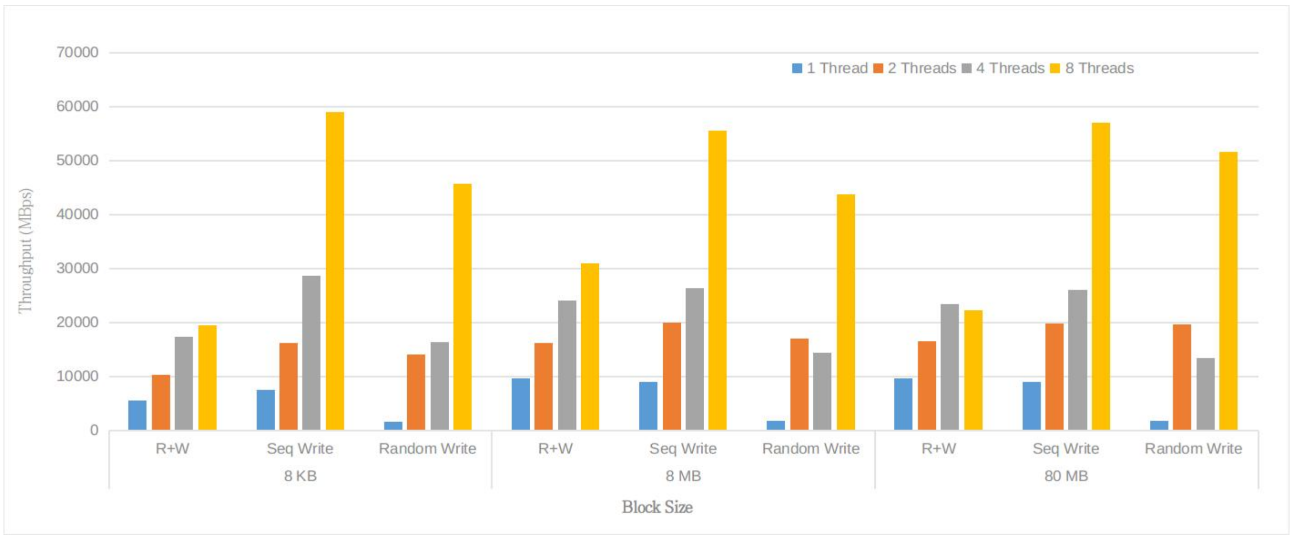


Figure 3: Throughput benchmark

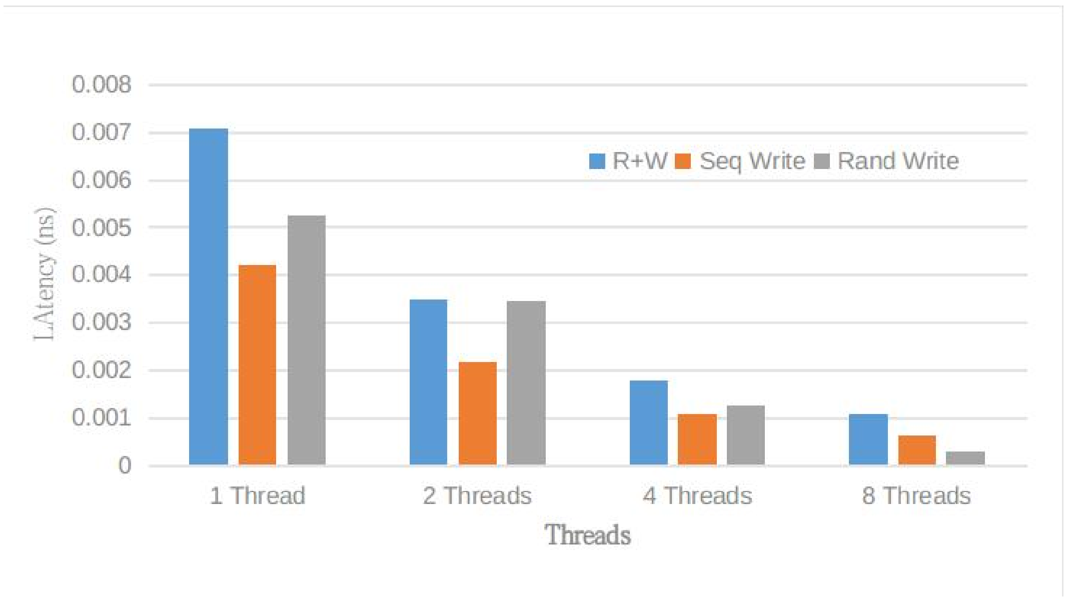


Figure 4: Latency benchmark

3 Disk

This experiment is very similar to the memory evaluation above in the sense that the requirements (throughput and latency), the variations in concurrency and block sizes are almost the same. Though the space allocated to the input/output files in this case is 10GB (It was 1.5GB in the memory experiment), the block sizes (8B, 8KB, 8MB, 80MB) and the number of threads (1, 2, 4, 8) remain exactly identical. The major distinction however is quite obviously in the mode of operations:

1. Sequential read and write
2. Sequential read
3. Random read

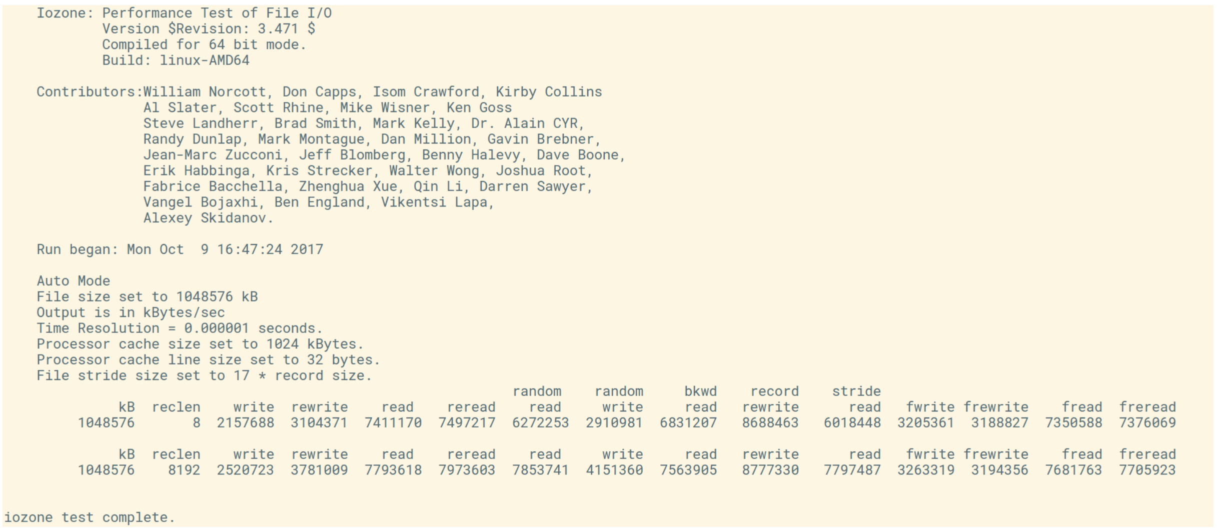
The theoretical performance of the provided disk, as advertised by the manufacturer is **6Gbps**. It is also important to note that the standard benchmarking tool used in this case is called **IOzone**, which, like Stream, is freely available on the Internet and can be used to measure the performance/speed of the computer's disk.

3.1 Program Design

To begin with, this benchmark is written in C as well. The idea of the program is to perform either read+write, sequential read or random read using different threads and to measure the monotonic time taken for all the operations to finish. Before processing any operations, a file of size 1 GB is filled with dummy data to be read from later on and at the same time a char array is initialized with data as well for future operations. For read+write, fwrite() is used to read from the aforementioned char array and write into an empty file. For sequential read,the input file created in the very first step is read into our char array using fread(). Now, for random write, the same thing is essentially done but by accessing random portions of the file with the help of a random number generator (rand()), the function fseek() to offset the stream and fread() together. All these operations and passed on to POSIX threads that are created and joined at the requisite portion of the program. The parameters of the run are altered as per required by redefining its struct parameters and the time taken for all the operations to finish is calculated by utilizing the clock_gettime() function.

3.2 IOzone

The IOzone benchmark was set to run on a fixed file size of one GB, varying the block sizes from 8 KB to 8 MB. Due to a limit on the buffer size of the benchmark, the 80 MB block size was not tested on. After an in-depth investigation of the obtained results in the next section, we will see that the IOzone benchmarks are way higher than ours. A screenshot is attached below for reference. Please note that the results are included as iozone_result.txt in the submission folder as well.

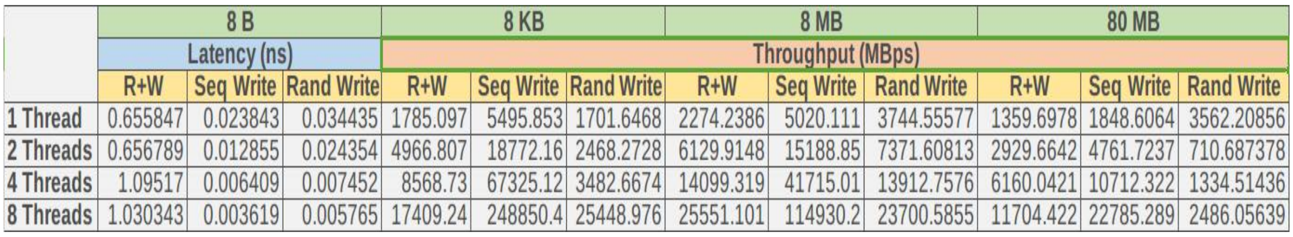


```
    Iozone: Performance Test of File I/O
            Version $Revision: 3.471 $
            Compiled for 64 bit mode.
            Build: linux-AMD64

    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: Mon Oct  9 16:47:24 2017

    Auto Mode
    File size set to 1048576 kB
    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.
                                                               random   random     bkwd    record     stride
              kB  reclen    write  rewrite    read    reread     read    write     read   rewrite       read   fwrite frewrite    fread  freread
         1048576       8  2157688  3104371 7411170   7497217  6272253  2910981  6831207   8688463    6018448  3205361  3188827  7350588  7376069

              kB  reclen    write  rewrite    read    reread     read    write     read   rewrite       read   fwrite frewrite    fread  freread
         1048576    8192  2520723  3781009 7793618   7973603  7853741  4151360  7563905   8777330    7797487  3263319  3194356  7681763  7705923


iozone test complete.
```

Figure 5: Result of the IOzone benchmark

3.3 Results

Based on the obtained values, it can be confirmed that the disk in this case is a **hard disk drive**. For achieving peak performance, throughput and latency wise, the optimal number of concurrency appears to be either 4. It is important to observe that for R+W, the latency appears to be the highest. It can also be noted that a good percentage of the intended performance is achieved in this experiment. Result metrics and graphs are provided below for further clarity. The standard deviation is around 5%.



	8 B			8 KB			8 MB			80 MB		
	Latency (ns)			Throughput (MBps)								
	R+W	Seq Write	Rand Write	R+W	Seq Write	Rand Write	R+W	Seq Write	Rand Write	R+W	Seq Write	Rand Write
1 Thread	0.655847	0.023843	0.034435	1785.097	5495.853	1701.6468	2274.2386	5020.111	3744.55577	1359.6978	1848.6064	3562.20856
2 Threads	0.656789	0.012855	0.024354	4966.807	18772.16	2468.2728	6129.9148	15188.85	7371.60813	2929.6642	4761.7237	710.687378
4 Threads	1.09517	0.006409	0.007452	8568.73	67325.12	3482.6674	14099.319	41715.01	13912.7576	6160.0421	10712.322	1334.51436
8 Threads	1.030343	0.003619	0.005765	17409.24	248850.4	25448.976	25551.101	114930.2	23700.5855	11704.422	22785.289	2486.05639

Figure 6: Average results of three runs of experiments

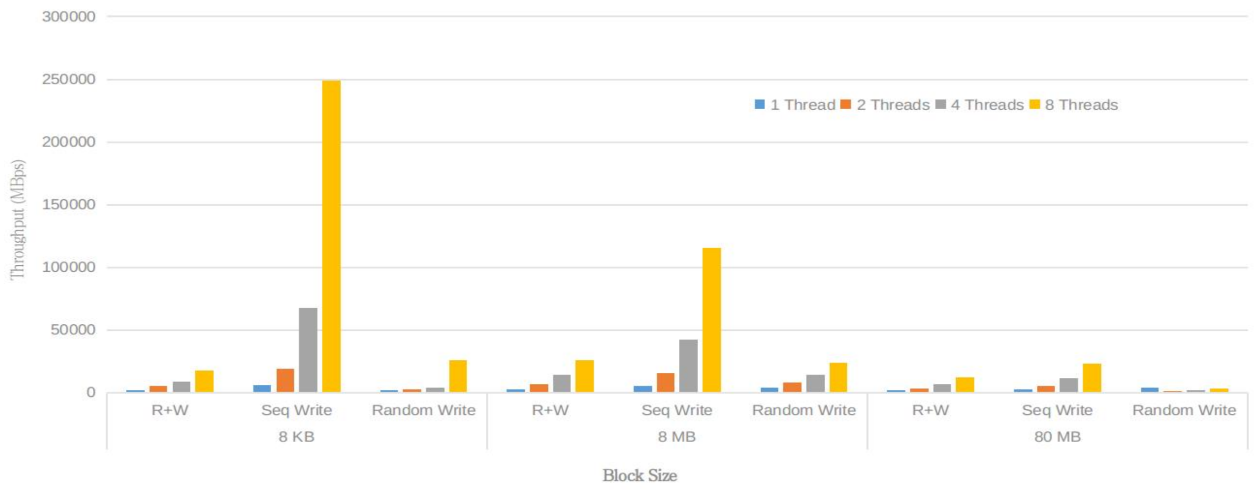


Figure 7: Throughput benchmark

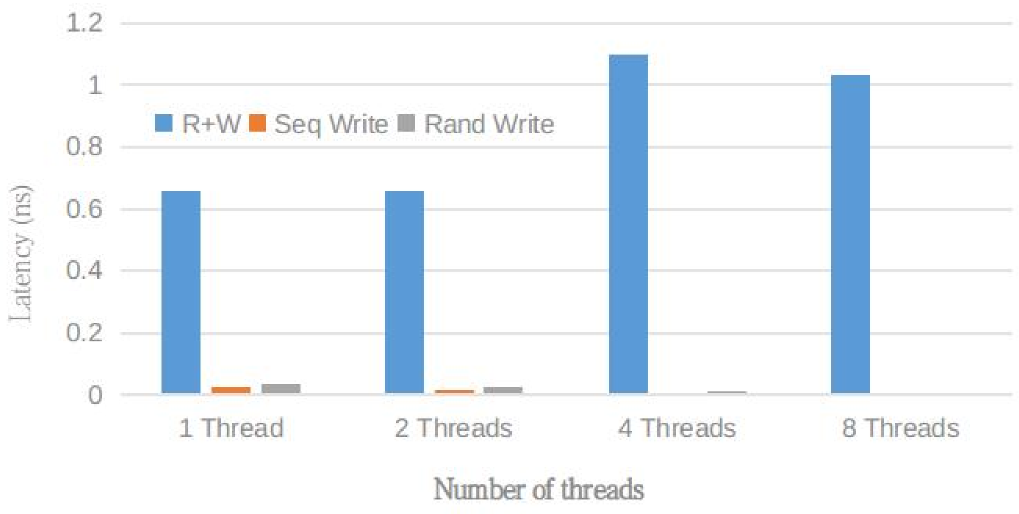


Figure 8: Latency benchmark