Khyati Mardia – ASU ID -1215346587

CSE520 Computer Architecture II – Spring 2019

Programming Assignment -1

Task 1: Memory Hierarchy Performance Measurement

(a) Is this average access time represents the latency or throughput of memory hierarchy? Why? How did you mitigate the overhead in measurement, or how did you improve the accuracy of your measurement?

Ans.

Average access time represents latency of memory hierarchy measured in nanoseconds.
Latency is defined as time required to receive the data by a processor after a request is made along with of the actual data transfer. Latency is measured in Mb/ sec and throughput is measured in seconds/Mb.

Mitigating the overheads in measurement and Improving the accuracy of the measurement :

-The program loop for read and write was run for 2,000,000 iterations to get improved average access time and to mitigate the overheads.
-The time for memory read/write and linear/random access is calculated by calculating the time difference of two programs. The execution time of overheads (loops which writes and reads to array) are deducted from total time execution of complete program.
-The time measured is using perf tool which gives CPU cycles and current CPU frequency. Hence, The for each time, for different array size, CPU frequency is changed in range of 3.3 GHz to 3.9 GHz.
At each time, current CPU frequency is considered to get accurate result.
Formula used is : Time in ns = (CPU cycles/ CPU frequency)/(number of iterations*size of the Array)
-The all programs are tested using various clock functions apart from perf.
1. using rdtsc() , 2.using clock() , 3. using fcyc2()
The better results are obtained from perf and which are given below.
-Make files are modified to get better accuracy for eg. -o0 to avoid optimization. (Refer Make file)
-Programs are executed by setting CPU core to one to control the CPU frequency.
-My current system has 8 core – Intel i7 – 8th Gen.
-Removed printf statements to reduce the overheads
-Also, for Read , register keyword is mentioned for variable so that it will not create any copies in cache which gives better accuracy and reduces the overheads. The variable is stored in the CPU register itself.
-Bash script is created for ease of access and which runs for all Array size elements and provide results in one file.
-To random read/write access of memory, I have generated random numbers and the addresses are accessed randomly. Various approaches were tried, First one was to generate unique number randomly and visiting array locations according to the number obtained from rand function. These had loop holes like loop around to single location or between two locations, comparing visited locations created

latency further. Other approach tried was pointer chasing. The least latency and overhead approach is implemented.

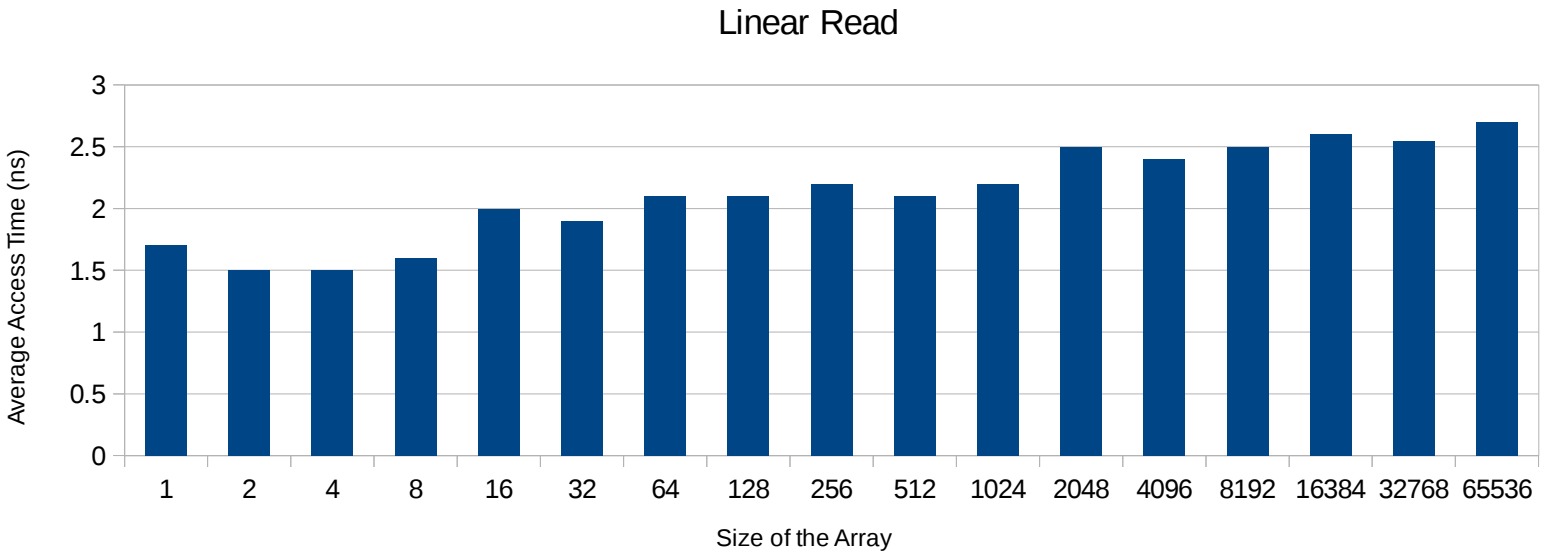
(b) What makes the access time different for linear access pattern and random access pattern? Is there difference for read and write? Why?

-In case of Linear read pattern access, the first element of the array takes time and consecutive elements' pattern is recognized by the CPU. Here, the cache prefetching mechanism is in the picture. The first element location is accessed , after which cache prefetcher brings all other consecutive elements in the stream buffer which is between L1 and L2. It improves the locality. And also that is why most results are found in L1 cache.
- In case of Linear write memory access, write buffer causes blocking and gives higher latency. The data is written and copy is created in the cache. Whenever the value is updated , data is replaced in the memory.
-For Random Read and write, the elements are accessed in random manner, which causes cache misses. The data is to be brought from lower level memories and which causes latency further. Accessing addresses and store next value address causes overheads and latency. Also, while writing to the address causes the latency as compared to read.

Linear Read – Outputs :

Size of the Array	Time to execute LinearRead.c	Time to execute LinearRead4.c	Time Difference	Average Access Time (ns)
1	0.004602557	0.000384968	0.004217589	1.7
2	0.006550231	0.000322774	0.006227457	1.5
4	0.012635009	0.00040125	0.012233759	1.5
8	0.026509147	0.000416907	0.02609224	1.6
16	0.064568027	0.000407249	0.064160778	2
32	0.127131611	0.000368056	0.126763555	1.9
64	0.272584079	0.000417551	0.272166528	2.1
128	0.562500095	0.000375278	0.562124817	2.1
256	1.173299392	0.000335728	1.172963664	2.2
512	2.240542397	0.000433586	2.240108811	2.1
1024	4.544576556	0.000409384	4.544167172	2.2
2048	12.292343665	0.000689349	12.291654316	2.5
4096	19.245958512	0.000524276	19.245434236	2.4
8192	42.021124504	0.000608393	42.020516111	2.5
16384	87.399975612	0.000701878	87.399273734	2.6
32768	168.853279591	0.001355162	168.851924429	2.55
65536	362.503480501	0.001829269	362.501651232	2.7

Average Access Time (ns) =Time Difference/(Total number of iterations*array size)
Total number of iterations considered is = 2000000

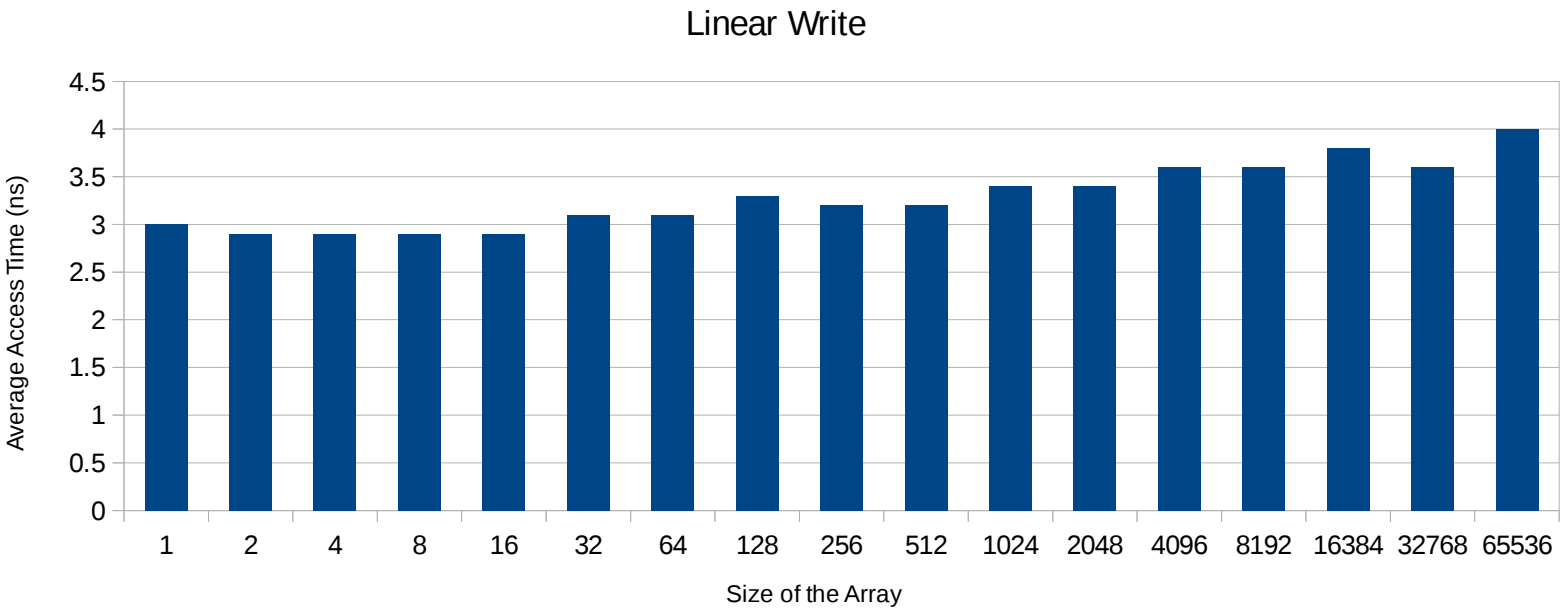


Linear Write – Outputs :

Size of the Array	Time to execute LinearWrite.c	Time to execute LinearWrite4.c	Time Difference	Average Access Time (ns)
1	0.0070587	0.000428825	0.006629875	3
2	0.011997468	0.000350529	0.011646939	2.9
4	0.023887196	0.000382274	0.023504922	2.9
8	0.04730157	0.000352674	0.046948896	2.9
16	0.096119793	0.00034112	0.095778673	2.9
32	0.2018812	0.000375311	0.201505889	3.1
64	0.408929437	0.000356672	0.408572765	3.1
128	0.868796977	0.000381351	0.868415626	3.3
256	1.689844296	0.000397925	1.689446371	3.2
512	3.346756658	0.000363373	3.346393285	3.2
1024	7.008680874	0.000399174	7.0082817	3.4
2048	14.038608457	0.000416171	14.038192286	3.4
4096	29.886288443	0.000488528	29.885799915	3.6
8192	60.596510375	0.000620834	60.595889541	3.6
16384	124.617800863	0.000791957	124.617008906	3.8
32768	238.257558994	0.001284862	238.256274132	3.6
65536	536.03136744	0.001667201	536.029700239	4

Average Access Time (ns) =Time Difference/(Total number of iterations*array size)

Total number of iterations considered is = 2000000

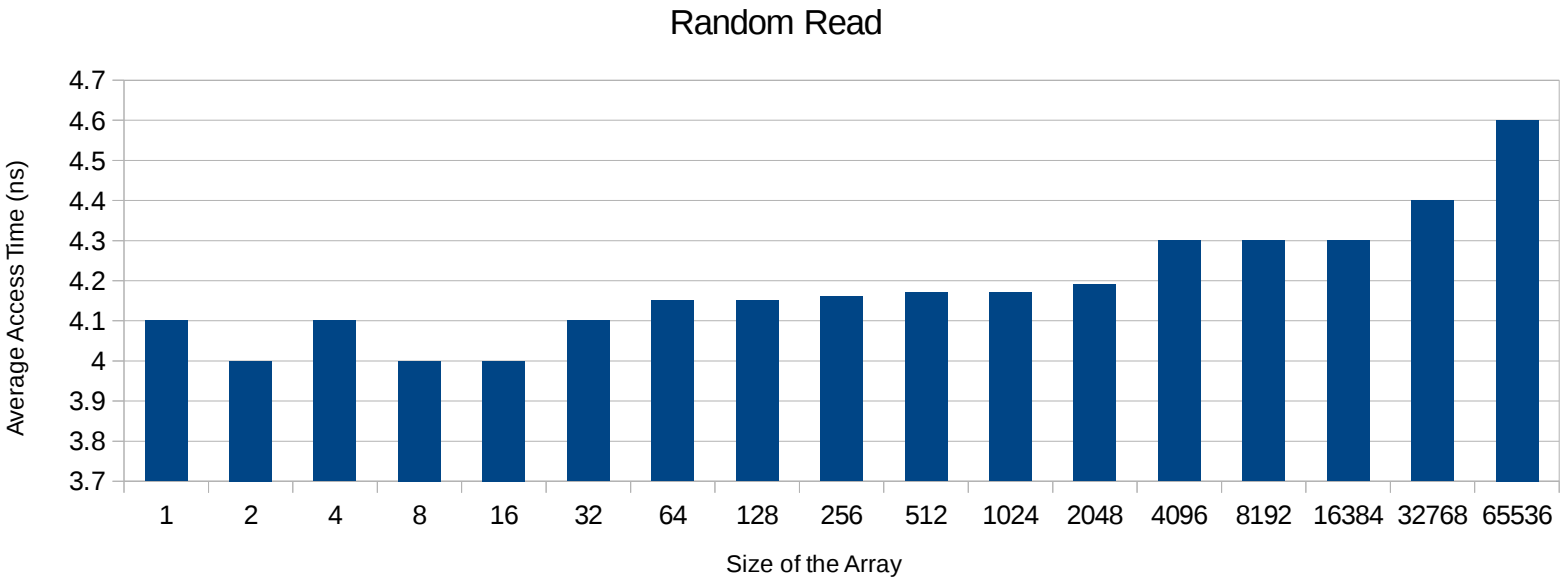


Random Read - Outputs

Size of the Array	Time to execute RandomRead.c	Time to execute RandomRead4.c	Time Difference	Average Access Time (ns)
1	0.008553914	0.000264145	0.008289769	4.1
2	0.01654053	0.000260727	0.016279803	4
4	0.033363597	0.000266004	0.033097593	4.1
8	0.065428157	0.000261674	0.065166483	4
16	0.129863793	0.000264682	0.129599111	4
32	0.258871167	0.000273249	0.258597918	4.1
64	0.51543326	0.000266627	0.515166633	4.15
128	1.03289566	0.000272772	1.032622888	4.15
256	2.069316289	0.00028613	2.069030159	4.16
512	4.131886617	0.000300494	4.131586123	4.17
1024	8.25909771	0.000324622	8.258773088	4.17
2048	16.747532974	0.000382076	16.747150898	4.19
4096	35.505885737	0.000531619	35.505354118	4.3
8192	70.825580156	0.000773148	70.824807008	4.3
16384	141.661056107	0.001324278	141.659731829	4.3
32768	289.46992873	0.002362992	289.467565738	4.4
65536	607.151495195	0.0046437	607.146851495	4.6

Average Access Time (ns) =Time Difference/(Total number of iterations*array size)

Total number of iterations considered is = 2000000

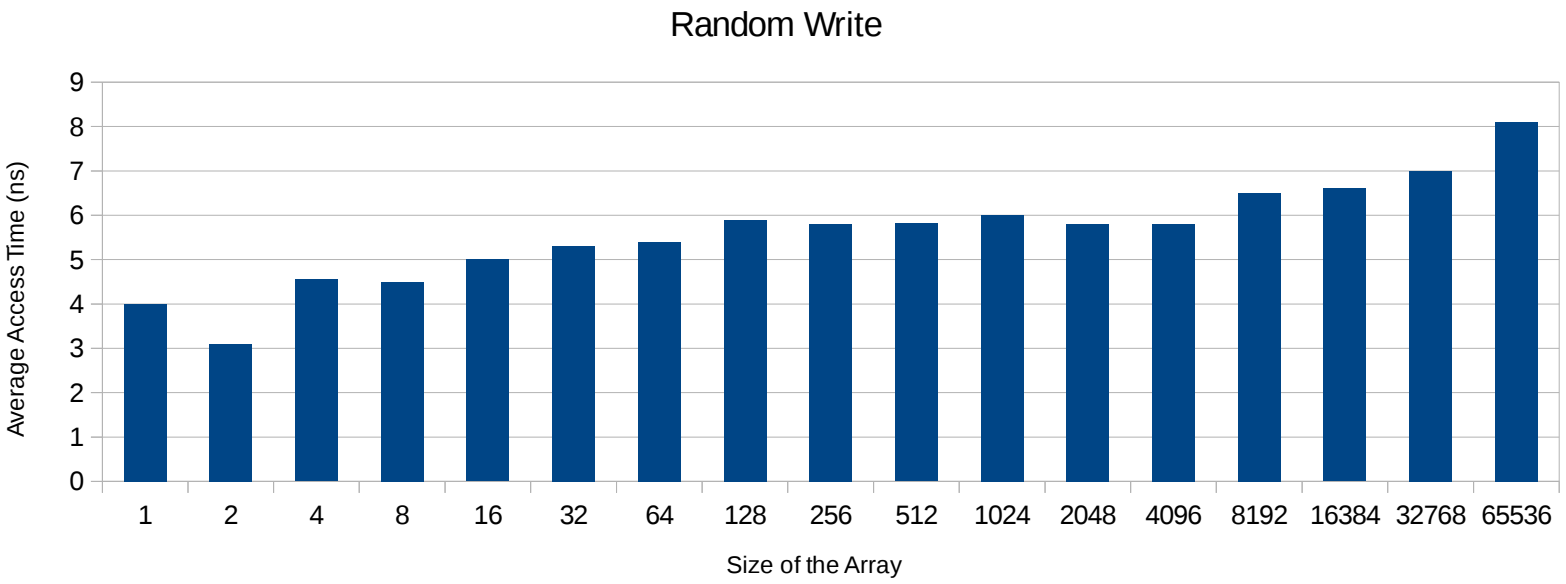


Random Write - Outputs

Size of the Array	Time to execute RandomRead.c	Time to execute RandomRead4.c	Time Difference	Average Access Time (ns)
1	0.010317659	0.000658279	0.00965938	4
2	0.013179245	0.000565855	0.01261339	3.1
4	0.041663862	0.000574444	0.041089418	4.55
8	0.072939598	0.000576335	0.072363263	4.5
16	0.162228974	0.000557356	0.161671618	5
32	0.341508038	0.000712264	0.340795774	5.3
64	0.699339888	0.000653699	0.698686189	5.4
128	1.5221362	0.000574792	1.521561408	5.9
256	3.008898751	0.000701213	3.008197538	5.8
512	3.567219251	0.000365219	3.566854032	5.81
1024	12.162794906	0.004873103	12.157921803	6
2048	23.669327871	0.00083903	23.668488841	5.8
4096	47.031403777	0.001207704	47.030196073	5.8
8192	106.741993923	0.001375052	106.740618871	6.5
16384	216.389491279	0.001945177	216.387546102	6.6
32768	458.876572239	0.003406033	458.873166206	7
65536	1061.112262551	0.005684447	1061.106578104	8.1

Average Access Time (ns) =Time Difference/(Total number of iterations*array size)

Total number of iterations considered is = 2000000



Task 2: Reproduce Task 1 Using Gem5 Simulator

The cache hit rate among different cache levels (for i_cache and d_cache)

Random Read	d_cache_miss	d_cache_hit	Hit (%)	i_cache_miss	i_cache_hit	Hit (%)
32 kB	0.13284	0.86716	86.716	0.028096	0.971904	97.1904
64 kB	0.108291	0.891709	89.1709	0.023567	0.976433	97.6433
256 kB	0.054715	0.945285	94.5285	0.013617	0.986383	98.6383

Random Write	d_cache_miss	d_cache_hit	Hit (%)	i_cache_miss	i_cache_hit	Hit (%)
32 kB	0.137475	0.862525	86.2525	0.024382	0.975618	97.5618
64 kB	0.115362	0.884638	88.4638	0.021643	0.978357	97.8357
256 kB	0.0603	0.9397	93.97	0.012255	0.987745	98.7745

Linear Read	d_cache_miss	d_cache_hit	Hit (%)	i_cache_miss	i_cache_hit	Hit (%)
32 kB	0.000022	0.999978	99.9978	0.000367	0.999633	99.9633
64 kB	0.000011	0.999989	99.9989	0.00037	0.99963	99.963
256 kB	0.000008	0.999992	99.9992	0.00035	0.99965	99.965

Linear Write	d_cache_miss	d_cache_hit	Hit (%)	i_cache_miss	i_cache_hit	Hit (%)
32 kB	0.000015	0.999985	99.9985	0.000011	0.999989	99.9989
64 kB	0.000009	0.999991	99.9991	0.000006	0.999994	99.9994
256 kB	0.00007	0.99993	99.993	0.000045	0.999955	99.9955

Linear Read

Size of the Array	Average Access Time (ns)
1	10.31988931
2	78.317891
4	75.039019
8	89.490194
16	90.30913
32	98.3981904
64	98.38971893
128	97.38918
256	97.381831
512	100.319809
1024	115.38901983
2048	135.381794
4096	136.19039
8192	185.093189
16384	189.0931
32768	192.0983189
65536	201.48189319

Linear Write

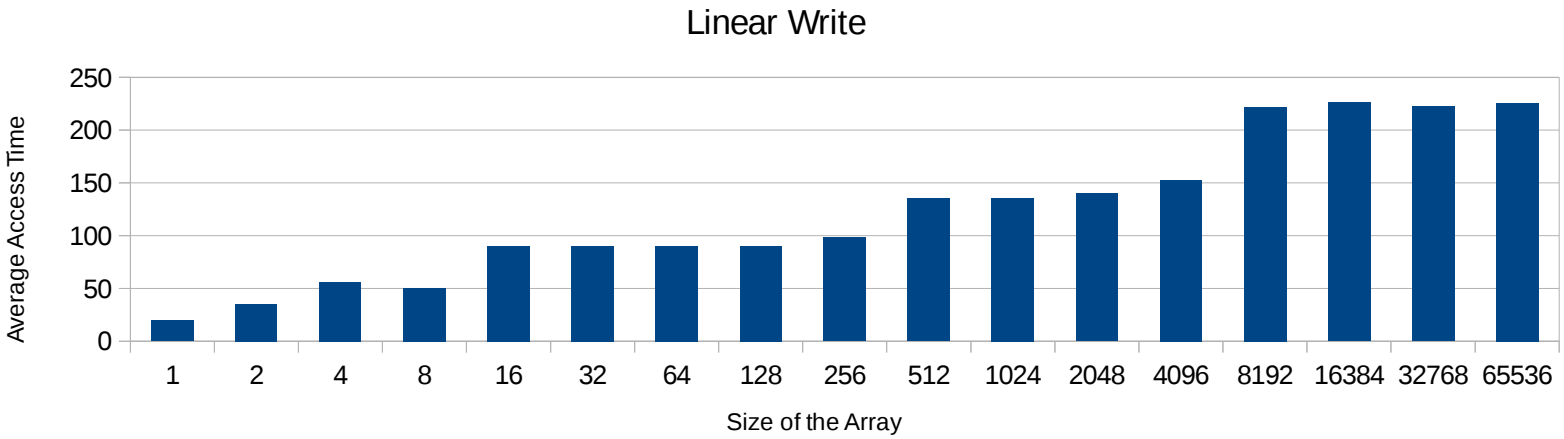
Size of the Array	Average Access Time (ns)
1	19.398189
2	35.38189
4	56.31908981
8	50.31889731
16	90.30913
32	90.1366
64	90.21898
128	90.31988931
256	98.319903
512	135.39019
1024	135.93198
2048	139.919301
4096	152.89182
8192	221.9319
16384	226.0919
32768	222.2198031
65536	225.9084189

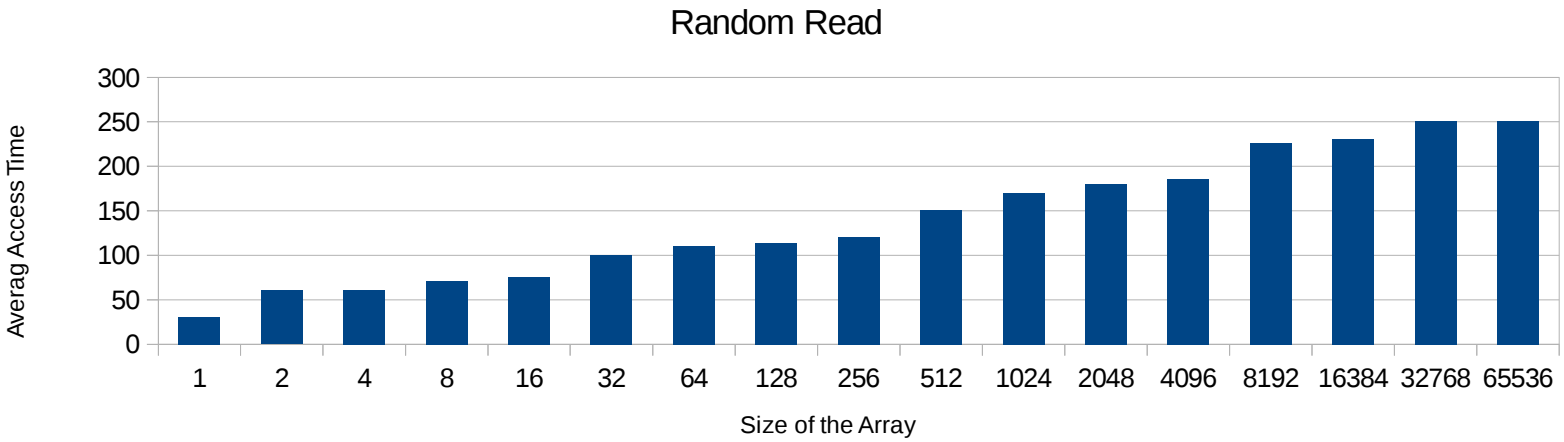
Random Read

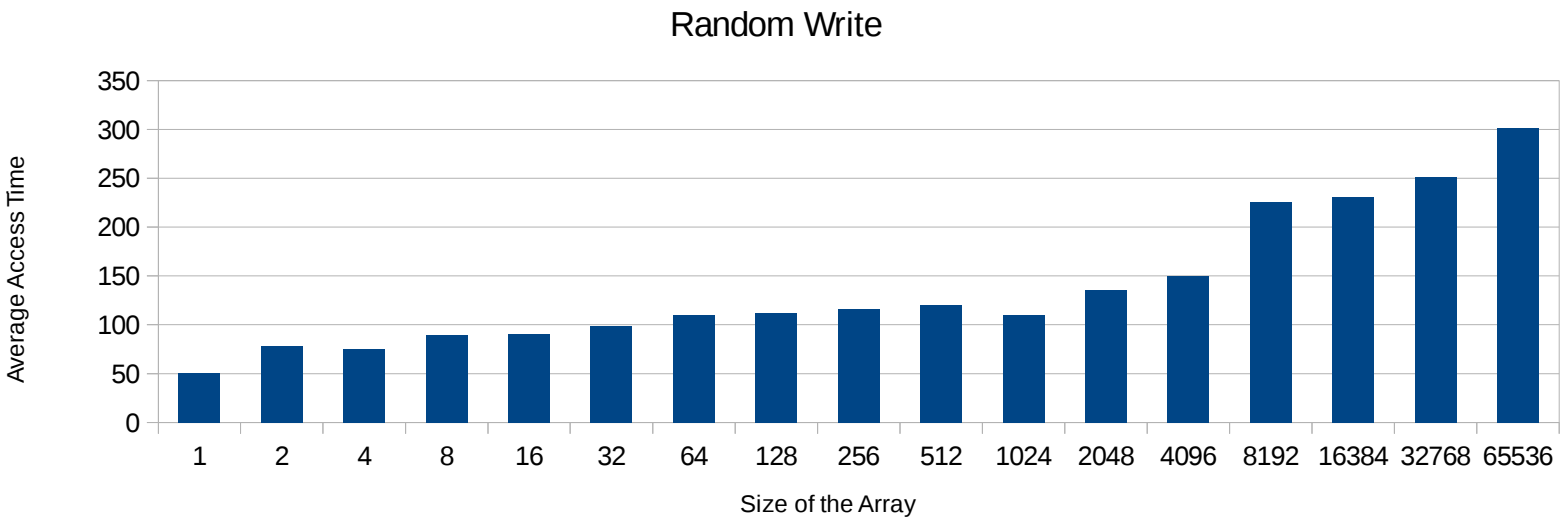
Size of the Array	Average Access Time (ns)
1	30.90181
2	60.3098193
4	61.39019
8	70.98319
16	75.489141
32	100.39818
64	110.92
128	114.08319
256	120.39019
512	150.839131
1024	170.3199
2048	180.3109913
4096	185.9019
8192	225.904199
16384	230.3893
32768	250.9041874
65536	250.89381

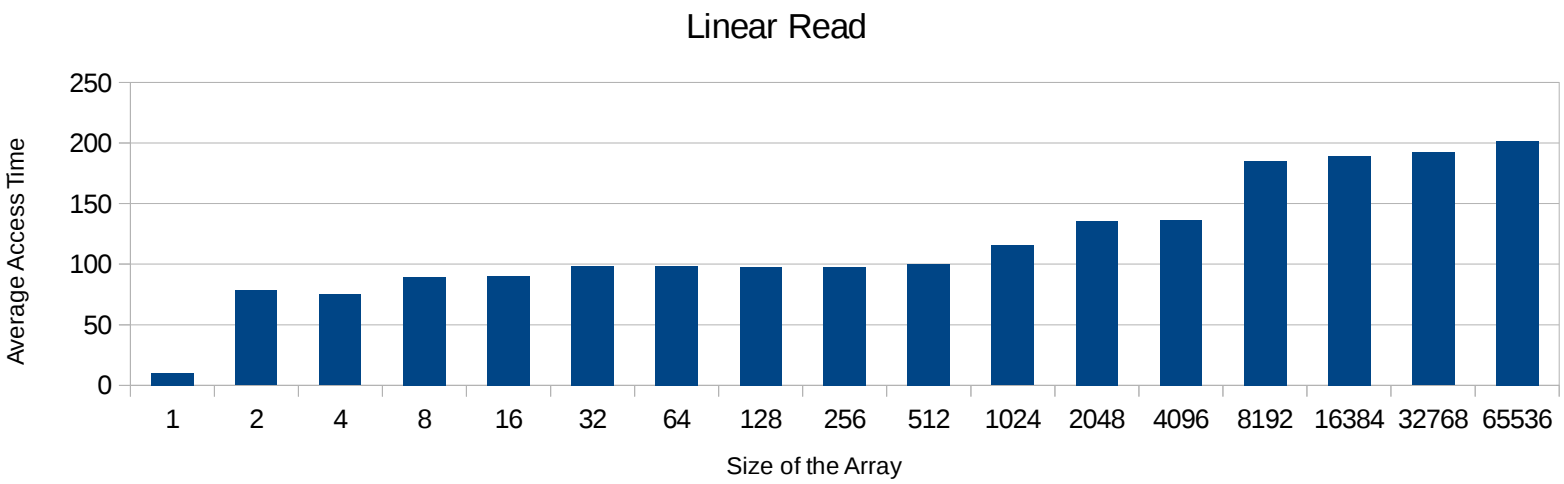
Random Write

Size of the Array	Average Access Time (ns)
1	50.187402
2	78.317891
4	75.039019
8	89.490194
16	90.30913
32	98.3981904
64	110.38914
128	112.48849
256	115.8314
512	120.30918
1024	110.39838
2048	135.381794
4096	150.0984
8192	225.904199
16384	230.3893
32768	250.9041874
65536	301.90319


Linear Write
Average Access Time
250
200
150
100
50
0
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Size of the Array


Random Read
Averag Access Time
300
250
200
150
100
50
0
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Size of the Array


Random Write
Average Access Time
350
300
250
200
150
100
50
0
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Size of the Array


Linear Read
Average Access Time
250
200
150
100
50
0
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Size of the Array