William Shin 884819645 wshi593

CompSys 304 A03

1.

A)

i) Direct mapped cache

$q = 64\ bits,\ p = 40,\ 512\ blocks\ and\ block\ size = 128\ bytes$

$b = log_2(\frac{q}{8}) = 3$

2^m *is the block size in words.* 2^{m+b} *is the block size in bytes*

$2^{m+b} = 128 = 2^7$

$2^{m+3} = 2^7$

$m + 3 = 7$

$m = 4$

2^n *is the cache size in number of blocks*

$2^n = 512 = 2^9$

$n = 9$

$t = p - n - m - b$

$t = 40 - 9 - 4 - 3$

$t = 24$

ii) 2-way set associative cache

There are 256 *sets with* 2 *blocks in each set*

Similar to i), $b = 3,\ m = 4$

Since there are 256 *sets*, $2^n = 256 = 2^8$

$n = 8$

$t = p - n - m - b$

$t = 40 - 8 - 4 - 3$

$t = 25$

B)

$AMAT = hit\ time + (miss\ rate * miss\ penalty)$

$AMAT = 60\% * (4 + 2\% * 100) + 40\% * (4 + 15\% * 100)$

$AMAT = 11.2\ cycles$

C)

$AMAT\ of\ L2 + MM$

$AMAT = 30 + 40\% * 100$

$AMAT = 70$

$AMAT\ of\ L1\ \&\ L2 + MM$

$AMAT = 60\% * (4 + 2\% * 70) + 40\% * (4 + 15\% * 70)$

$AMAT = 9.04\ cycles$

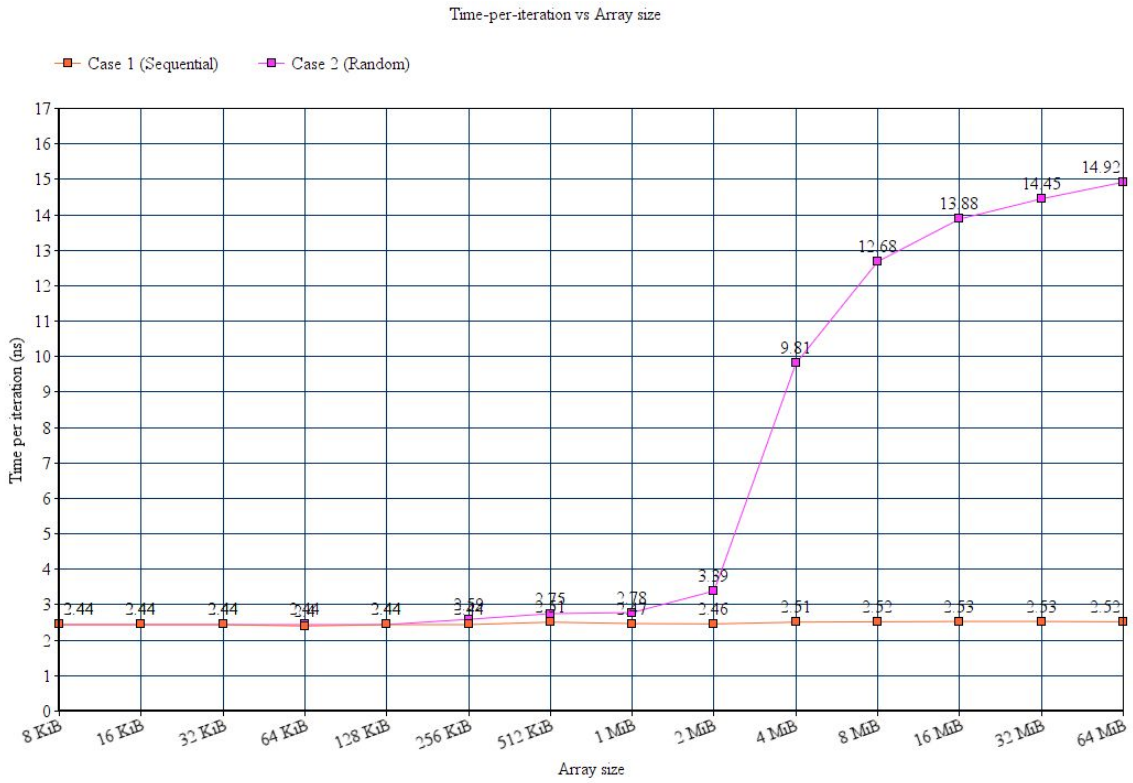
William Shin 884819645 wshi593

2.
Processor: Intel Core i5-4590

Level 1 Cache size: 128KiB
Level 2 Cache size: 1 MiB
Level 3 Cache size: 6 MiB

N	Size of *a*	Time per iteration/ns case 1 (sequential)	Time per iteration/ns case 2 (random)
2048	8 KiB	2.44	2.44
4096	16 KiB	2.44	2.44
8192	32 KiB	2.44	2.44
16384	64 KiB	2.40	2.44
32768	128 KiB	2.44	2.44
65536	256 KiB	2.44	2.59
131072	512 KiB	2.51	2.75
262144	1 MiB	2.47	2.78
524288	2 MiB	2.46	3.39
1048576	4 MiB	2.51	9.81
2097152	8 MiB	2.52	12.68
4194304	16 MiB	2.53	13.88
8388608	32 MiB	2.53	14.45
16777216	64 MiB	2.52	14.92

William Shin 884819645 wshi593



In the first case, the time per iteration remains relatively uniform as the program is able to take advantage of spatial locality by accessing contiguous memory locations. When a cache miss occurs, a block containing the next sequence of values will be loaded into the cache and the next memory accesses can directly use the values in the cache block. The miss rate is relatively uniform as the array size increases because the access pattern is also uniform. Therefore, this results in a relatively uniform time per iteration.

In the second case, the program achieves differing results. It is able to achieve a relatively uniform time per iteration, similar to the first case (sequential implementation), as long as the array size does not exceed the size of my machine's L1 cache. This is because the whole array is able to be stored in the L1 cache, and the average miss rate is similar to the first case, as misses earlier in the execution of the program will result in values being loaded into the cache so that there are fewer misses later in execution. However, when the array size becomes greater than the L1 cache size, the miss rate increases as data that is needed later in execution is removed from L1 cache and placed on the L2 cache. This results in the program having to access L2 cache, which increases the AMAT and average time per iteration. Similarly, when the array size exceeds the L2 cache size (1 MiB) and L3 cache size (6MiB), we see a performance hit and an increase in average time per iteration.