2 Cache Performance Analysis [80 points]

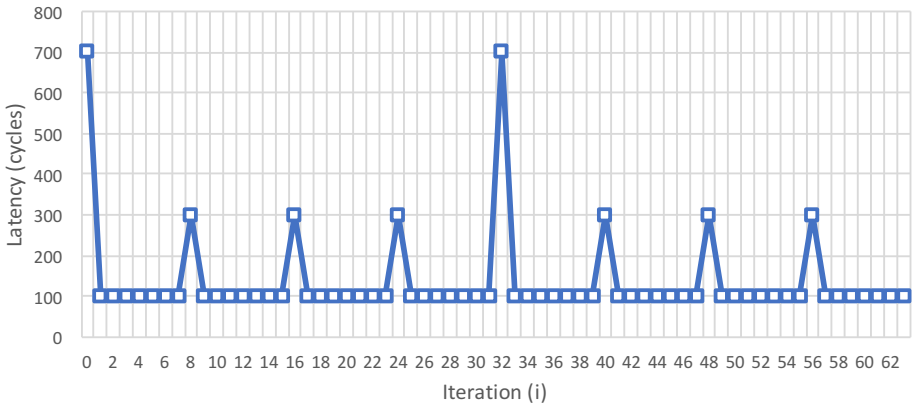
We are going to microbenchmark the cache hierarchy of a computer with the following two codes. The array `data` contains 32-bit unsigned integer values. For simplicity, we consider that accesses to the array `latency` bypass all caches (i.e., `latency` is *not* cached). `timer()` returns a timestamp in cycles.

```
(1) j = 0;
    for (i=0; i<size; i+=stride){
       start = timer();
       d = data[i];
       stop = timer();
       latency[j++] = stop - start;
    }

(2) for (i=0; i<size1; i+=stride1){
       d = data[i];
    }
    j = 0;
    for (i=0; i<size2; i+=stride2){
       start = timer();
       d = data[i];
       stop = timer();
       latency[j++] = stop - start;
    }
```

The cache hierarchy has two levels. L1 is a 4kB set associative cache.

(a) [15 points] When we run code (1), we obtain the latency values in the following chart for the first 64 reads to the array `data` (in the first 64 iterations of the loop) with `stride` equal to 1. What are the cache block sizes in L1 and L2?



L1 cache block size is 32 bytes, and L2 cache block size is 128 bytes.

Explanation:
The highest latency values (700 cycles) correspond to L2 cache misses. The latency of 300 cycles correspond to L1 cache misses that hit the L2 cache. The lowest latency (100 cycles) corresponds to L1 cache hits. We find L1 misses every 8 32-bit elements, and L2 misses every 32 32-bit elements. Thus, L1 cache blocks are of size 32 bytes, while L2 cache blocks are 128 bytes.

(b) [20 points] Using code (2) with `stride1` = `stride2` = 32, `size1` = 1056, and `size2` = 1024, we observe `latency[0]` = 300 cycles. However, if `size1` = 1024, `latency[0]` = 100 cycles. What is the maximum number of ways in L1? (Note: The replacement policy can be either FIFO or LRU).

The maximum number of ways is 32.

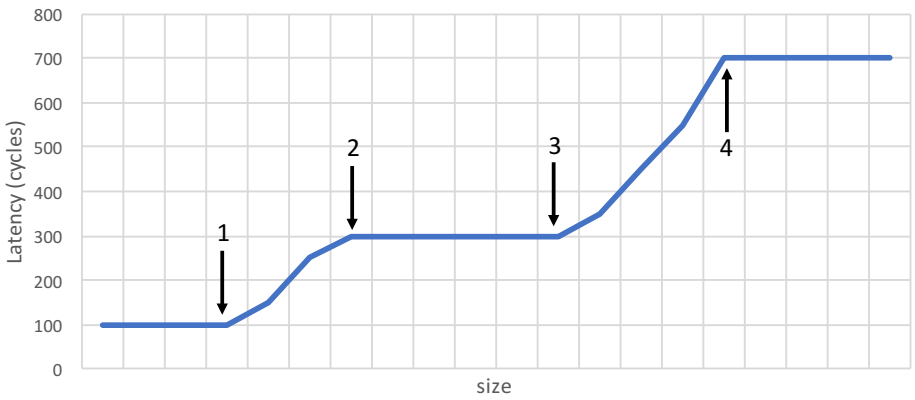
Explanation:
In L1 there are a total of 128 cache blocks. If the accessed space is 1056 elements, 33 cache blocks are read. In case of having more than 32 ways, there would be a hit in the second loop. However, with 32 or less ways, the cache block that contains `data[0]` is replaced in the last iteration of the first loop.

(c) [20 points] We want to find out the exact replacement policy, assuming that the associativity is the maximum obtained in part (b). We first run code (2) with `stride1` = 32, `size1` = 1024, `stride2` = 64, and `size2` = 1056. Then (after resetting `j`), we run code (1) with `stride` = 32 and `size` = 1024. We observe `latency[1]` = 100 cycles. What is the replacement policy? Explain. (Hint: The replacement policy can be either FIFO or LRU. You need to find the correct one and explain).

It is FIFO.

Explanation:
In the second loop of code (2), the last cache block that is accessed (`data[1024]`) replaces the cache block that contains `data[0]`, if the replacement policy is FIFO. If the replacement policy is LRU, the LRU cache block (the one that contains `data[32]`). Because we observe that `latency[1]` is 100 cycles, and it corresponds to the access to `data[32]`, we conclude the replacement policy is FIFO.

(d) [25 points] Now we carry out two consecutive runs of code (1) for different values of `size`. In the first run, `stride` is equal to 1. In the second run, `stride` is equal to 16. We ignore the latency results of the first run, and average the latency results of the second run. We obtain the following graph. What do the four parts shown with the arrows represent?



Before arrow 1:

The entire array fits in L1. In arrow 1 the size of the array is the same as the size of L1 (4kB).

Between arrow 1 and arrow 2:

Some accesses in the second run hit in L1 and other accesses hit in L2.

Between arrow 2 and arrow 3:

All accesses in the second run hit in L2.

Between arrow 3 and arrow 4:

Still some accesses in the second run hit in L2. Arrow 3 marks the size of the L2 cache.

After arrow 4:

The accesses of the second run always miss in L2, and it is necessary to access main memory.

Explain as needed (if you need more):