CS 61C Summer 2023 — Caches — Discussion 11

1 Precheck

This section is designed as a conceptual check for you to determine if you conceptually understand and have any misconceptions about this topic. Please answer true/false to the following questions, and include an explanation:

1.1 We cannot use a 1KB cache in a 32-bit system because it's too small and cannot contain all possible addresses.

1.2 If a piece of data is both in the cache and in memory, reading it from cache is faster than reading from memory.

1.3 Caches see an immediate improvement in memory access time at program execution.

1.4 Increasing cache size by adding more blocks always improves (increases) hit rate for all programs.

1.5 Decreasing block size to increase the **number** of blocks held by the cache improves the program speed for all programs.

2 Cache Associativity

When working with caches, we have to be able to break down the memory addresses we work with to understand where they fit into our caches. There are three fields:

- **T**ag: Used to distinguish different blocks that use the same index. Number of bits: (# of bits in memory address) - Index Bits - Offset Bits

- **I**ndex: The set that this piece of memory will be placed in. Number of bits: $\log_2$(# of indices)
- **O**ffset: The location of the byte in the block. Number of bits: $\log_2$(size of block)

In order to evaluate cache performance and hit rate, especially with determining how effective our current cache structure is, it is useful to analyze the misses that do occur, and adjust accordingly. Below, we categorize cache misses into three types:

- **Compulsory**: A miss that must occur when you bring in a certain block for a first time, hence "compulsory". Reduce compulsory misses by having longer cache lines (bigger blocks), which bring in the surrounding addresses along with our requested data. Can also pre-fetch blocks beforehand using a hardware prefetcher (a special circuit that tries to guess the next few blocks that you will want).
- **Conflict**: Occurs if the block was fetched before, but evicted while the cache was not full. Increasing the associativity or improving the replacement policy would remove the miss. Note that Conflict misses tend to occur with inefficient cache usage, compared to Capacity misses.
- **Capacity**: Occurs if the block was fetched before, but evicted while the cache was full. Capacity misses are independent of the associativity of your cache. The only way to remove the miss is to increase the cache capacity.

Last discussion, we focused on Direct-Mapped caches, in which blocks map to specifically one slot in our cache. This is good for quick replacement and finding out block, but not good for spatial efficiency!

This is where we bring associativity into the matter. We define associativity as the number of slots a block can potentially map to in our cache. Thus, a Fully-Associative cache has the most associativity, meaning every block can go anywhere in the cache. Our Direct-Mapped cache, on the other hand, has the least, being only 1-way set associative.

For an N-way associative cache, the following are true:

$$\text{N} * \#\text{ sets} = \#\text{ blocks}, \quad \text{Index bits} = \log_2(\#\text{ sets})$$

2.1 Here's some practice involving a 2-way set associative cache. This time we have an 8-bit address space, 8 B blocks, and a cache size of 32 B. Classify each of the following accesses as a cache hit (H), cache miss (M) or cache miss with replacement (R). For any misses, list out which type of miss it is (Compulsory, Conflict, or Capacity). Assume that we have an LRU replacement policy (in general, this is not always the case).

Address	T/I/O	Hit, Miss, Replace
`0b0000 0100`		
`0b0000 0101`		
`0b0110 1000`		
`0b1100 1000`		
`0b0110 1000`		
`0b1101 1101`		
`0b0100 0101`		
`0b0000 0100`		
`0b0011 0000`		
`0b1100 1011`		
`0b0100 0010`		

2.2 What is the hit rate of our above accesses?

3 AMAT

Recall that AMAT stands for Average Memory Access Time. The main formula for it is:

$$\text{AMAT} = \text{Hit Time} + \text{Miss Rate} * \text{Miss Penalty}$$

In a multi-level cache structure, we can separate miss rates into two types that we consider for each level.

- **Global:** Calculated as the number of accesses that missed at that level divided by the total number of accesses *to the cache system.*
- **Local:** Calculated as the number of accesses that missed at that level divided by the total number of accesses *to that cache level.*

3.1 In a 2-level cache system, after 100 total accesses to the cache system, we find that the L2$ (L2 cache) ended up missing 20 times. What is the global miss rate of L2$?

3.2 Given the system from the previous subpart, if L1$ had a local miss rate of 50%, what is the local miss rate of L2$?

Suppose your system consists of:

1. An L1$ that has a hit time of 2 cycles and has a local miss rate of 20%
2. An L2$ that has a hit time of 15 cycles and has a global miss rate of 5%
3. Main memory where accesses take 100 cycles

3.3 What is the local miss rate of L2$?

3.4 What is the AMAT of the system?

3.5 Suppose we want to reduce the AMAT of the system to 8 cycles or lower by adding in a L3$. If the L3$ has a local miss rate of 30%, what is the largest hit time that the L3$ can have?

4 Analysis

Given the follow chunk of code, analyze the hit rate given that we have a byte-addressed computer with a total memory of **1 MiB**. It also features a **16 KiB** Direct-Mapped cache with **1 KiB** blocks. Assume that your cache begins cold.

```
#define NUM_INTS 8192   // 2^13
int A[NUM_INTS];        // A lives at 0x10000
int i, total = 0;
for (i = 0; i < NUM_INTS; i += 128) {
    A[i] = i;           // Line 1
}
for (i = 0; i < NUM_INTS; i += 128) {
    total += A[i];      // Line 2
}
```

4.1 How many bits make up a memory address on this computer?

4.2 What is the T:I:O breakdown?

4.3 Calculate the cache hit rate for the line marked Line 1:

4.4 Calculate the cache hit rate for the line marked Line 2: