CS 61C
Fall 2022

# Cache Performance

Discussion 10

## 1 Pre-Check

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 In a write-back cache, writes are generally slower than a write-through cache.

False. Writes are generally faster in a write-back cache since they only write to cache, whereas writes in a write-through cache write to both cache and main memory.

1.2 In a fully associative cache, the number of index bits is always zero.

True. Since any block can go anywhere in a fully associative cache, there is no need for an index field that determines where each block could go to.

1.3 For the same cache size and block size, a 4-way set associative cache will have fewer index bits than a direct-mapped cache.

True. A direct mapped cache needs to index every line of the cache, whereas a 4-way set associative cache needs to index every set of 4 lines. The 4-way set associative cache will have 2 fewer index bits than the direct-mapped cache.

1.4 Using N-way set associative caches can help balance between hardware complexity and cache performance.

True. As N goes up, there will generally be less conflict misses, but more intricate comparator circuits are needed for the tag. Since the value of N is adjustable, N-way set associative caches helps balance these two aspects.

1.5 Increasing cache size (while keeping other parameters constant) for a fully associative cache with LRU replacement will never increase miss rate (for the same memory access pattern).

True. Since we always keep the most recently used blocks in cache, the misses on a cache with larger size is a subset of the misses on the same cache with smaller size.

Interestingly, this is not true if the replacement policy was FIFO! This is called Bélády's anomaly and is out of scope for this class. CS 162 will talk more about this...

1.6 Adding another level of caches will never increase AMAT.

False. In the extreme case of a miss rate of 100%, adding a new level of caches will increase AMAT by the hit time of those caches.

1.7 The miss penalty for the last level of caches is equal to the access time of main memory.

True. If we miss on the last level of caches, we must go to main memory to find the data we need.

## 2 Cache Associativity

In the previous discussion, we primarily focuses 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 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` | | |
| `0b1100 1000` | | |

Since our cache is 2-way set associative, there are 2 blocks in a set. Given the cache size and the block size, we have 32 / 8 = 4 blocks. Thus, there are 4 / 2 = 2 sets in our cache. We need $\log_2(2) = 1$ bit to differentiate the 2 sets, so we have 1 index bit. Our block size of 8 B means we have $\log_2(8) = 3$ offset bits, and that the rest of our bits are our tag bits. Therefore, our TIO breakdown means bits 0, 1, and 2 are our offset bits, the only index bit is bit 3, and bits 4-7 being the tag bits.

```
0b0000 0100    Tag 0000, Index 0, Offset 100 - M, Compulsory
0b0000 0101    Tag 0000, Index 0, Offset 101 - H
0b0110 1000    Tag 0110, Index 1, Offset 000 - M, Compulsory
0b1100 1000    Tag 1100, Index 1, Offset 000 - M, Compulsory
0b0110 1000    Tag 0110, Index 1, Offset 000 - H
0b1101 1101    Tag 1101, Index 1, Offset 101 - R, Compulsory
0b0100 0101    Tag 0100, Index 0, Offset 101 - M, Compulsory
0b0000 0100    Tag 0000, Index 0, Offset 100 - H
0b1100 1000    Tag 1100, Index 1, Offset 000 - R, Capacity
```

2.2 What is the hit rate of our above accesses?

$\frac{3 \text{ hits}}{9 \text{ accesses}} = \frac{1}{3}$ hit rate

## 3 Writes

When it comes to writing data to cache memory, there are multiple write policies to consider that offer different options when building our system. Some of them you might encounter are:

1. **Write-through**: In this policy, when we have a write we write to both the cache and the memory. This is the case for every write, so the main memory always has the updated data. This is simple to implement, but writing to main memory every single time is slow.
2. **Write-back**: On a write, the data is only updated/written in the cache. The main memory only receives the data upon eviction. This means the cache has more up to date data most of the time. While this is faster as there is less accesses to main memory, it is harder to implement as we have to include more overhead, such as dirty bits and so on.
3. **Write-around**: Data is only written to main memory, and whenever we do so we automatically invalidate the old data in the cache.

Another thing to consider is what we do when we want to write to memory that is not in the cache, or a write miss. For that, we have 2 possible policies:

1. **Write-allocate**: On a write miss, we pull the block you missed on into the cache
2. **No write-allocate**: On a write miss, you do not pull the block you missed on into the cache. Only memory is updated. On a read cache miss, we still pull the data into the cache.

3.1 Considering the above information, lets consider a direct mapped, no write-allocate write-through cache with a capacity of 8B and a block size of 4B. Lets also assume that the memory addresses are 8 bits each. Assuming the cache is completely empty in the beginning, we make memory accesses to the following locations:

- 0x6A, Write
- 0x85, Read
- 0x6B, Read

- 0x87, Read
- 0x68, Write

With the above memory access pattern and the given cache configuration, how many times do we access the main memory?

Short Answer: 4
Long Answer:

- The first cache access to the location 0x6A is a miss, as the cache is initially empty. As the cache is no write allocate, on this cache miss we just write to the main memory only, so this is our first main memory access, and the cache is still empty.
- Then on the second cache access, we have a read miss - for this one we go to main memory, and actually bring the line in the cache, which occupies the cache line with index 1. 2nd main memory access.
- Third one is again a read miss, so the same happens and the line with the index 0 gets filled out with the memory address this time. Third main memory access.
- This one is actually a cache hit - tag is 0b1 0000 (which was put in the cache in step 2) so no main memory access, read hit.
- This one is a write hit, but because our cache is write through we actually write in the main memory as well, so 4th main memory access.

3.2 Lets say for the same cache size, memory accesses but the only difference is that we have a no write-allocate write-back cache instead. How many memory accesses to the main memory do we have in this case?

Short Answer: 3
Long Answer:

- The first cache access it the same as above, 1st main memory access.
- Again, same as above, 2nd main memory access.
- Same as above, 3rd main memory access.
- Same as above, read hit (no main memory access)
- This one is a write hit, but this time as we have a write back cache, we do not go to main memory - we write the new data on the cache, and turn the dirty bit on the relevant line to 1.

3.3 For one last optimization, we decide to use a write-allocate cache instead. So, now we have a write-allocate, write-back cache. How many times do we access the memory now?

Short Answer: 2
Long Answer:

- This is again a write miss, but because our cache is write-allocate now, we actually bring the data in from the main memory into the cache, and the line with index 0 gets filled. 1st main memory access.

- This one is the same as the above 2 examples, so cache line with index 1 gets filled and we have a read miss. 2nd main memory access.
- Because the first write actually filled the 0th index line with the relevant data, this 3rd memory access actually becomes a read hit, as there is data on the cache now. No main memory access!
- Same as above, read hit (no main memory access).
- Same as 2.2, as the cache is write-back, this write hit is taken on the cache itself, and the line on the cache gets changed - no main memory access.

3.4 We discovered in our previous subparts that one set of write policies leads to less main memory accesses than the others. Out of the policies covered in class, is this the most optimal set of write policies for a cache? If so, explain why. If they are not, what trade-offs come as a result of these policies?

Our write-allocate, write-back cache seems to have the best performance here. In practice, write-back policies have been used much than write-through, though some architectures still employ specialized write-through caches. However, some trade-offs to a write-back cache include having to deal with overhead especially when dealing with multiple local caches as a result of multiple CPUs (this is different from multi-level cache structures). In a write-back cache, where the caches may hold updated data while main memory doesn't, 'cache coherency' must be maintained between caches so that each write is propagated across the system.

## 4 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*.

4.1 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\$?

$\frac{20}{100} = 20\%$

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

$\frac{20}{50\% * 100} = \frac{20}{50} = 40\%$. We know that L2\$ is accessed when L1\$ misses, so if L1\$ misses 50% of the time, that means we access L2\$ 50 times, of which we ended up having 20 misses in 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

4.3 What is the local miss rate of L2$?

The number of accesses to the L2$ is the number of misses in L1$, so we divide the global miss rate of L2$ with the miss rate of L1$.

L2$ Local miss rate $= \frac{\text{Misses In L2\$}}{\text{Accesses in L2\$}} = \frac{\text{Misses in L2\$}}{\text{Total Accesses}} / \frac{\text{Misses in L1\$}}{\text{Total Accesses}} =$
$\frac{\text{Global Miss Rate}}{\text{L1\$ Miss Rate}} = \frac{5\%}{20\%} = 0.25 = 25\%$

4.4 What is the AMAT of the system?

AMAT = 2 + 20% x (15 + 25% x 100) = 10 cycles, as the Miss Penalty of the L1$ is the 'local' AMAT of the L2$.

Using global rates of each level, alternatively, AMAT = 2 + 20% x 15 + 5% x 100 = 10 cycles (using global miss rates)

4.5 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?

Let $H$ = hit time of the cache. Extending the AMAT equation so that the Miss Penalty of the L2$ is the 'local' AMAT of the L3$, we can write:
$2 + 20\% * (15 + 25\% * (H + 30\% * 100)) \leq 8$
Solving for H, we find that $H \leq 30$. So the largest hit time is 30 cycles.