Caches Workout

Discussion 07 - 25 November 2019
CS 250P - Computer Systems Architecture
TA - Aftab Hussain

Problem 1. The L1 cache is of relatively small size (e.g. ~32kB). Can you explain how the L1 cache could still have a high hit rate (or low MPKI - misses per kilo instructions) ?

The cache access patterns are predictable. This is because of two properties of cache memory access, temporal locality and spatial locality. Temporal locality refers to the property that the same block of data is likely to be accessed after a given unit of time (number of cycles). Spatial locality is the property of the likelihood of accessing memory in a contiguous (or sequential) pattern, over a period of time (or cycles). Thus, even if the L1 cache is small, the likelihood of it containing the data needed by a process over a given period of time can be high because of these properties.

Problem 2. Why does it take more time to access the L2 cache than it takes to access the L1 cache?

The L2 cache is larger than the L1 cache, which means, an access would need signals to transmit across longer wires across the structure, and thus take more time.

Problem 3. Say a processor executes a 1000 instructions, of which 200 instructions are loads and stores. Say 5% of these 200 instructions miss the L1 cache. What is the MPKI of the L1 cache?

5/100 x 200 = 10 instructions are missed. So MPKI is 10. In other words, for every 1000 instructions in the processor, 10 of them would be misses.

Problem 4. Explain inclusive, exclusive, and non-inclusive cache hierarchies.

They are described respectively as follows,
> There are duplicate data across different levels of cache. For instance, everything in L1 cache can also be found in L2 cache.
> There are no duplicate data across different levels of cache.
> There may or may not be data across cache across different levels.

Problem 5. Let's say in any given 64-byte cache, we want to store data as 8-byte words.
How many such words can we store in this cache?
How many bits do we need to uniquely identify (and thereby access) each of these words?

How many bits do we need to access a byte within each word?

For 8-byte words, we have,
64/8 = 8 words.
We need 3 (2^3 = 8) bits to uniquely address each of these words.
Each word has 8 bytes, so we would again need 3 more bits to access each byte within a word.

Problem 6. Let's say in any given 64-byte cache, we want to store data as 16-byte words.
How many such words can we store in this cache?
How many bits do we need to uniquely identify (and thereby access) each of these words?
How many bits do we need to access a byte within each word?

For 16-byte words, we have,
64/16 = 4 words.
We need 2 bits to uniquely address each of these words.
Each word has 16 bytes, so we would need 4 more bits to access each byte within a word.

Problem 7. *Cache layout terminology.* 40-bit addresses are used to access data in the cache. Say our 64-byte cache can store 8-byte words. (The cache is direct mapped, single set associative).

(a) What is each entry of the cache, which holds an 8-byte word called? Write all the terms that may be used interchangeably.
Row/set/cache line/block

The last 6 bits of the 40 bit address are used to access data in the cache. Say these last 6 bits are, 101011

(b) What do the first 3 bits of these 6 bits identify? What are these bits called?
The first 3 bits access row/entry/set number 5 in the cache. These bits are called the index bits.

(c) What do the last 3 bits identify? What are they called?
These bits are the offset bits. They identify a byte within a word (in this case byte number 3 in row number 6).

Problem 8. Following on from the previous problem, there may be multiple 40-bit addresses that have the same last 6 bits. I.e., the first 34-bits of those addresses may be different. How do we distinguish between those addresses? What are those bits called?

Link to solution.

Problem 9. What is a direct-mapped cache?

Link to solution.

Problem 10. Explain a cache design structure that allows us to refer to multiple entries in the cache, using a single 40-bit address. Mention what cache performance metric is affected by such a design and how.

The cache design structure is called the n-way set associative cache (e.g. the 2-way set-associative cache). Explanation.

Problem 11. A set is the same as a cache line in the design in Problem 7. In Problem 10, we use a different cache design. How does the definition of a set change in these two designs?

In a cache design with single set associativity,
[---------]
[---------]
[---------]
// Each row has 1 set, and correspond to 1 cache line.

In a 4-way set associative cache,
[---------][---------][---------][---------]
[---------][---------][---------][---------]
[---------][---------][---------][---------]
[---------][---------][---------][---------]
//each row corresponds to a set, each set can have 4 different blocks of data or 4 cache lines.

Problem 12. Let us say you have a 32kB, 4-way set-associative data cache array. The size of each cache line is 32 bytes.

(a) How many sets do we have?
Each cache line or block is of 32 bytes. Since it is 4-way, each set will have 4 of these blocks. So 1 set has 4x32 = 128 bytes. So the number of sets would be total cache size (32kB) divided by 128, which is 256. The cache layout or data array is shown as below.

[---32 bytes---][---32 bytes---][---32 bytes---][---32 bytes---]
[---32 bytes---][---32 bytes---][---32 bytes---][---32 bytes---]
[---32 bytes---][---32 bytes---][---32 bytes---][---32 bytes---]
..
.. (256 entries/sets in total)
..
[---32 bytes---][---32 bytes---][---32 bytes---][---32 bytes---]

Problem 10 tells us about three segments of the address bits. Assume we use 40-bit addresses.

(b) How many bits are used for each of these segments for the cache design in this problem?
The three segments are tag, index, and offset bits.

Offset bits
Since there are 32 bytes in each block, we need **5** bits to access each byte in each block.
Index bits
The index bits are used to access a set. Since there are 256 sets, we need **8** bits.
Tag bits
The rest are tag bits (40-13=**27** bits).

The first segment of those bits are used to access what is known as a tag array.

(c) How big is the tag array?
The tag array mirrors the data array (shown above). Each block in the data array will have a corresponding tag in the tag array. Each tag is 27 bits. Since there are 4 ways and therefore 4 blocks in each of the 256 sets, our tag array size is 27 bits x 4 x 256 = 27 kilobits of data (3.375 kilobytes of data)

Problem 13. Give an example where a write-allocate policy for caches would be useful.

Explanation of difference between write-allocate policy and no-write allocate policy, and an example where the former policy would be useful.

Problem 14. Explain the performance differences between a serial tag/data access strategy and a parallel tag/data access strategy.

Solution.

Problem 15. What is a victim cache?

Solution.