15213 - Lecture 12 - POGIL Activity (Caching)

Introduction

In this activity you will learn about caches. Some of this material is based on ideas from peerinstruction4cs.org.

Model 1: Caches

As you proceed through college, you will collect more and more notes and textbooks. You can store them in three places: your backpack, your dorm room, and your parents' house (or other home away from CMU).

1. When you are in class today, which place is the most convenient? Where is the least convenient?
2. Which place can hold the most notes and books?
3. When you leave your dorm room, how do you pick what books and notes to take with you?
4. At the end of the semester, how might you change your storage of your notes and textbooks?

Model 2: Lookup

When a program tries to access memory, before the request goes onto the bus, it is passed through one or more caches. Each cache must quickly answer the question: does it have the data?

1. If the `array` below starts at (32-bit) address `0x00 00 FF 00`, what is the address of the last element?

   ```
   int array[20];
   ```

2. Considering the four bytes of a 32-bit address, which address byte relates most to spatial locality?
3. A cache is very similar to a hash table. As we have learned (in Sec 5.14 among other places), we want hash values to be across a large range. Caches use a very simple hashing algorithm of selecting a subset of the address bits. Discuss which range(s) you might use.
4. (advanced) Which part of the address often indicates whether the access is to the stack, heap, or global variables?

Unlike a hash table, caches do not resize, rehash, or otherwise move elements. Caches chain elements to a limited extent if they hash to the same location in the cache.

Model 3: Hardware

We can describe a cache using three numbers: S, E, B. The following figure shows how the first two relate to the cache organization.

1. The hashing bits from the previous model are actually the set index bits.
   If there are S sets, how many bits are required for this purpose?

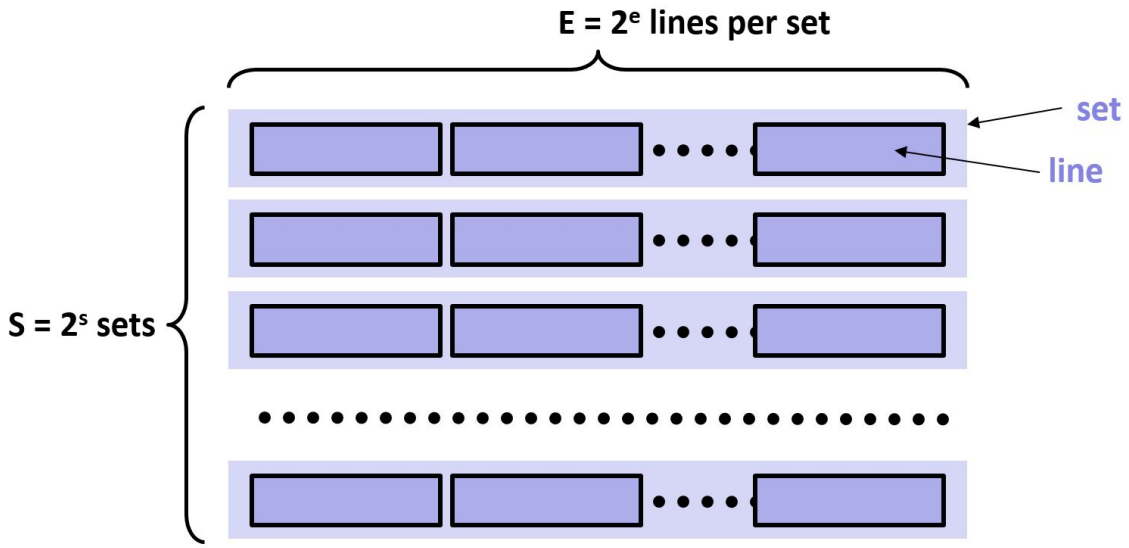

Figure 1: Cache

2. Cache lines contain blocks of data. The spatial locality bits are the block offset, which select the byte(s) from the block to return. How many bits are required for this purpose?
3. On selecting a cache set using the index bits, the cache must search (simultaneously as this is hardware) all of its lines to see whether any match the address. Each check is done using the remaining bits of the address. These bits are the tag. If m is the number of address bits, how many bits are in the tag (t)?

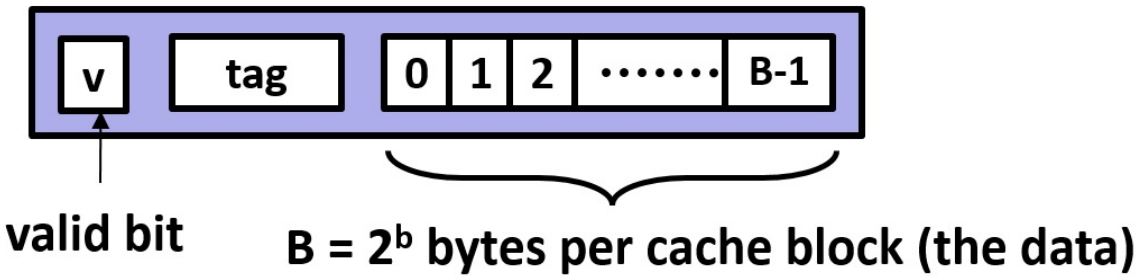

Figure 2: Line

Cache lines also contain a valid bit to store whether this line is valid.

Sets in caches will have one or more lines. If there is only one line, that cache is termed 'direct mapped'. If there is only one set, then the cache is termed 'fully associative'. Otherwise, the cache is considered E-way associative.

4. Associativity has trade-offs. Remember that hardware costs space and energy. Between direct mapped and fully associative, select the one that:
   - Requires the most hardware
   - Can determine whether a cache line is present fastest
   - Is more likely to have the data present
5. There are three basic types of cache misses. We also rate caches for their miss rates, which are *misses/accesses*. When an address is read from for the very first time, where is it?

6. The program accesses `array[0]`, `array[1]`, `array[2]`, and `array[3]` (see Model 2). At minimum, how large should B be for there to be (ideally) only 1 miss?

Model 4: Replacement

Although not all college students do this (take for example one of my roommates!), let's consider hanging your shirts in a closet. Imagine that your closet has limited space, but that you also have a box under your bed to store additional clothes.

1. Your closet is full and you have received a new shirt. How do you make room in your closet for this shirt?

- Pick randomly
- Box up the newest shirt
- Box up your oldest shirt
- Box up a shirt you haven't worn in a while
- I use a different policy to replace my shirts (seasonal, etc)

2. For the previous question, what additional information would you need to:

- Make the decision
- Determine whether the decision is best

When a cache access misses, the new address and its associated line is added to the appropriate set. Caches do not "rehash" addresses on a collision, instead the cache must select a line to replace in that set. Similarly, every cache line also maintains additional data in order to determine which cache line to replace.

3. Caches seek to exploit temporal and spatial locality to mitigate the memory wall (i.e., the vast latency difference between a CPU cycle and a memory access). Consider the following access stream going to a set, where E = 2. Using the common least recently used (LRU) replacement policy, mark which accesses will hit and which will miss:

   ```
   A, B, A, C, B, C, A
   ```

4. Could a different replacement algorithm have done better? If so, how?

5. When a cache line is replaced, it is termed evicted. What should the cache do with the evicted line?

Model 5: Writing to Cache Lines

1. When a processor writes to a memory location, from where does the data come?

2. When a processor writes to a cache line present, the access hits in the cache. However, there is the question of what to do with the written data. Should the cache immediately update memory? Is there locality in the writes? How did we handle this code before?

   ```
   for (int i = 1; i < (1000 * 1000); i++) a[0] += a[i];
   ```

If the cache caches the written values, then the cache is write back (WB). If it immediately updates memory, then it is write through (WT).

3. In a write back cache, what must the cache do in when a cache line is evicted? Does the cache line need additional information?

4. Now imagine writing to a location that is not already in cache. Do you think the cache should allocate a cache line to store the new data, or should the write skip the cache and go directly to memory? Why or why not?

The choice described in the previous question is termed write allocate (WA) vs. write no allocate (WNA). However, most architectures make both write design choices together, with the common pairings being WBWA and WTWNA.

5. (advanced) When might it be advantageous to use each of these?