Caches

Introduction

1. Basic review of caches from undergraduate computer architecture course

Locality Principle

1. Locality Principle: Things that will happen soon are likely to be close to things that just happened
    - Also used in branch prediction

Locality Quiz

1. Which of these are not good examples of locality?
    - Rained 3 time today -> likely to rain again today
    - Ate dinner at 6 every day last week -> probabaly we will eat dinner around 6 this week
    - It was New Year's Eve yesterday -> Probably it will be New Year's Eve today (not a good example)

Memory References

1. If a processor accessed address X recently...
    - Likely to access X again soon (temporal)
    - Likely to access addresses close to X too (spatial)

Temporal Locality Quiz

1. Consider the following program:
    - int sum = 0;
    - for(int j = 0; j < 1000; j++) { sum = sum + arr[j]; }
2. Which of these memory locations has temporal locality in this code?
    - j (yes)
    - sum (yes)
    - Elements of arr (no)

Spatial Locality Quiz

1. Consider the following program:
    - int sum = 0;
    - for(int j = 0; j < 1000; j++) { sum = sum + arr[j]; }
2. Which of these memory locations has spatial locality in this code?
    - j (no)
    - sum (no)
    - Elements of arr (yes)

Locality and Data Accesses

1. Example: Library - Large amount of data but slow to access
    - Accesses have both temporal and spatial locality
        - Temporal: Look up definition of locality frequently
        - Spatial: Look up some computer architecture definition -> will look up other computer architecture information too
    - A student will...
        - Go to library, find info, go back home (doesn't benefit from locality)
        - Borrow the book (ideal)
        - Take all books and build a library at home (expensive, but does not benefit from locality)

Cache Quiz

1. Main memory is large and slow to access
2. Lots of spatial and temporal locality in programs
3. When accessing a memory location...
    - Go to main memory for every access (no)
    - Have small memory inside processor core, bring accessed data there (yes)
    - Have huge memory next to our processor chip, bring everything there

Cache Lookups

1. Attributes of cache
    - Fast -> small
    - Not everything will fit
    - Access:
        - Cache hit: Found it in the cache (fast)
        - Cache miss: Not in cache -> access slow memory (copy this location to the cache to exploit locality)

Cache Performance

1. Properties of a good cache
    - Average memory access time (AMAT)
        - AMAT = Hit Time + Miss Rate * Miss Penalty
        - Hit time -> Small and fast cache
        - Miss rate -> Large and/or smart cache
        - Miss penalty -> Main memory access time
    - Miss Time = Hit Time + Miss Penality
    - AMAT = (1 - Miss Rate) * Hit Time + Miss Rate * Miss Time

Hit Time Quiz

1. In a well-designed cache..
    - Hit time < Miss time (true)
    - Hit time > Miss penalty (false)
    - Hit time == Miss penalty (false)
    - Miss time > Miss penalty (true)
2. Hit time << Miss time
    - Miss time is roughly equal to the miss penalty

Miss Rate Quiz

1. Which is true for a well-designed cache?
    - Hit rate > Miss rate (true)
    - Hit rate < Miss rate (false)
    - Hit rate == Miss rate (false)
    - Hit rate is almost 1 (true)
    - Miss rate is almost 1 (false)
2. Hit rate = 1 - Miss rate

Cache Size in Real Processors

1. Modern processors typically have several caches
2. L1 Cache: Directly service read/write requests from the processor
    - 16 kB - 64 kB

  - Large enough to get ~90% hit rate
  - Small enough to hit in 1-3 cycles

Cache Organization

1. How to determine a hit or miss?
   - Need a table of some sort that we can quickly index with some bits of the address
     - Cache line contains some bits to tell what data is present in that line
     - Block size: How many bytes are in each entry (line size)
   - A block size of 1 means only 1 byte can be retrieved at a time
     - This complicates LW/SW instructions as they would access 4 cache lines
     - Want block size to be at least as big as the largest single access we can do in the cache (hopefully slightly larger)
     - If there is spatial locality, we want to bring in a chunk of data around the desired memory location
     - Typical block size: 32-128 bytes
     - A block size of 1 kB would mean bringing in a lot of data; if there is no spatial locality, much of this data is never accessed (wasteful)
2. How to determine what to evict?

Block Size Quiz

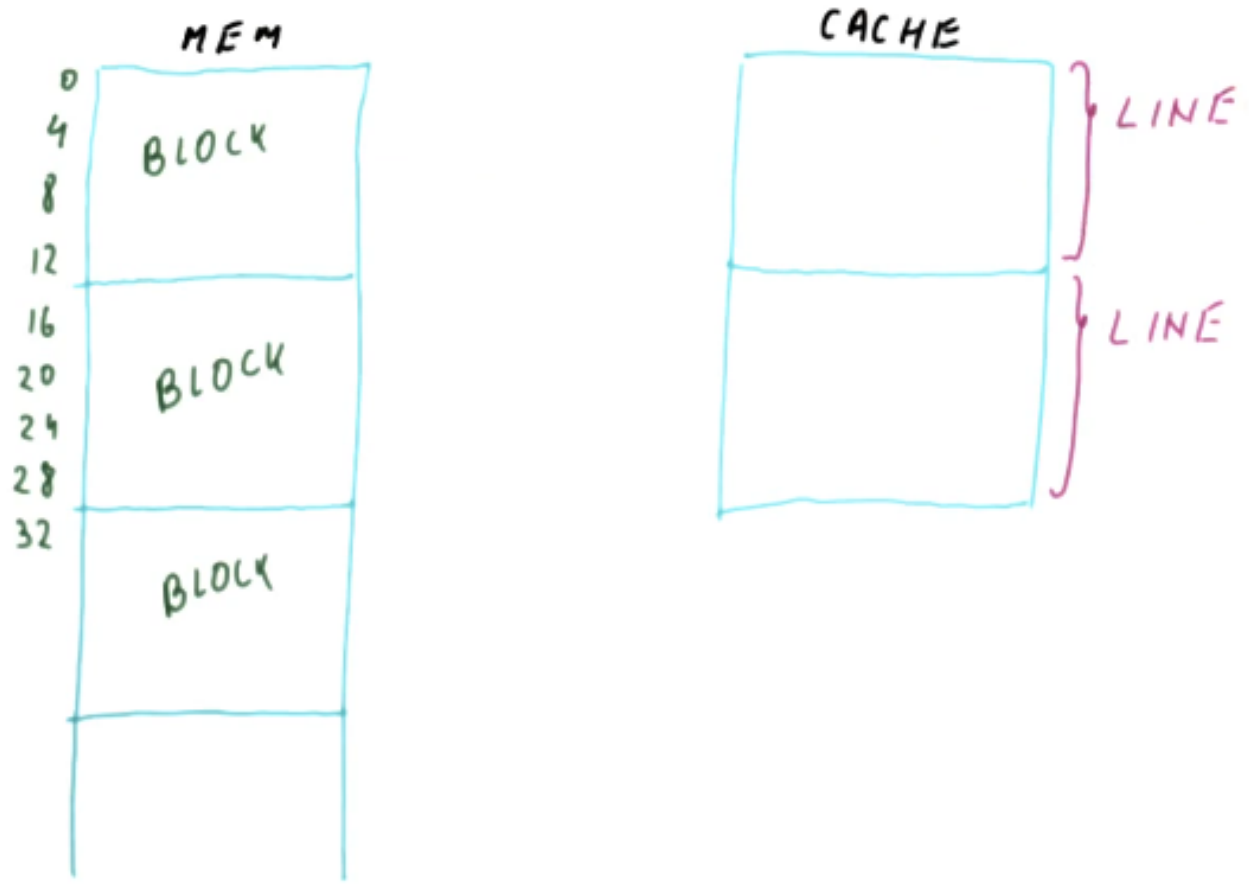
1. 32 kB cache with 64 byte block size
2. Program accesses variables x1, x2, . . . , xN
   - Lots of temporal locality
   - No spatial locality
3. What is the largest N that still results in a high hit rate?
   - N = 2 ^ 15 / 2 ^ 6 = 2 ^ 9 = 512

Cache Block Start Address

1. How do we determine where a cache block can start?
   - Anywhwere: 64B block -> 0..63, 1..64, 2..65
     - This complicates access because a single address could map to several different locations in the cache
     - Another complication is that blocks can overlap; when writing, need to determine all of the locations that the memory location references and update them
   - Aligned: Align addresses to block size (0..63, 64..127, . . .)
     - Can use some bits of the address to index into the cache

Blocks in Cache and Memory

1. Space in memory is referred to as a block
2. Space in cache is referred to as a line
3. Blocks get mapped to lines



Blocks in Cache and Memory

Cache Line Sizes Quiz

1. Which of these are not good line sizes in a 2kB cache?
   - 1 B (not good)
   - 32 B (good)
   - 48 B (not good)
   - 64 B (good)
   - 1 kB (not good)
2. Explanations:
   - 1B does not exploit spatial locality and word accesses will go to multiple cache locations
   - 32B exploits spatial locality and is a power of 2
   - 48B is not a power of 2 (more complicated to divide by 48)
   - 64B is good for the same reason as 32B
   - 1 kB is not good because only 2 will fit in the cache

Block Offset and Block Number

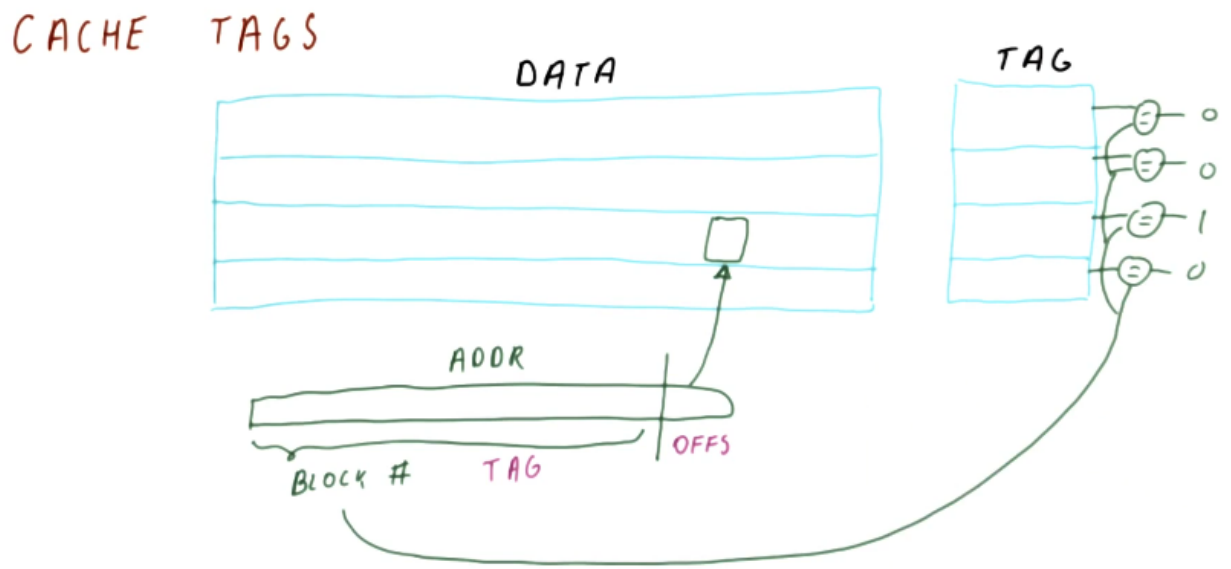
1. Address is 32 bits, block size is 16 bits
   - 4 bits needed to determine the block offset
   - Remaining bits determine the block index (block number)

Block Number Quiz

1. Consider a processor with 16-bit addresses and a cache with 32B block size
2. Address is 1111 0000 1010 0101
   - Block number is 1111 0000 101
   - Block offset is 00101

Cache Tags

1. In addition to the data, the cache keeps a separate tag to keep track of what memory location is stored in each cache line
2. Compare the block number from the address to the tag; if identical, that memory location is already present in the cache
3. Block # is not necessarily equal to the tag



Cache Tags

Cache Tag Quiz

1. The tag always...
   - Contains the entire address of the 1st byte (false)
   - Contains at least one bit from the block offset (false)
   - Contains at least one bit from the block number (true)
   - Contains some bits from block offset and some from block number (false)

Valid Bit

1. Need a way of designating whether an entry is legitimate
   - Any possible tag value might match an address
2. Add another bit of state to the cache
   - Valid bit: 1 if the entry is populated
   - Initialize to zero when the processor is started
3. HIT = (TAG == BLOCK #) & V

Types of Caches

1. Fully associative: Any block can be in any line
2. Set-associative: N lines where a block can be
   - N is typically > 1 but smaller than total number of lines in the cache
   - N is typically small relative to the number of lines in the cache
     - Often 2, 4, or 8
3. Direct-mapped: A block can go into 1 line

Direct Mapped Cache

1. For any given block address, there is only one place in the cache where that address can be
2. Least significant bits of block address are still block offset
3. Next bits are index into the cache
4. Remaining bits are the tag
   - Tag does not include index bits
   - Tag needs to identify what data is in the cache; we've already determined which line this address corresponds to with the index bits
     - Storing index bits in the tag is redundant (all entries must have the same index bits)

Upside and Downside of Direct Mapped Cache

1. Upsides:
   - Only need to look in one place (faster, cheaper, more energy efficient)
     - Fewer comparisons result in less hardware
2. Downsides:
   - Block MUST go in one place
     - If A and B map to the same location, we may continually invalidate that line as we swap between them (conflicts)

Direct Mapped Cache Quiz 1

1. 16 kB direct-mapped cache, 256-byte blocks
2. Which of these conflict with 0x12345678?
   - 0x12345677 (no)
   - 0x11335577 (no)
   - 0x11115678 (yes)
   - 0x12341666 (yes)
3. Explanation:
   - 256-byte blocks -> lowest 8 bits are block offset
   - 16 kB / 256 -> 6 bits for index
   - Index of 0x12345678 is 010110
   - First is not a conflict because they map to different blocks (same tag)

Direct Mapped Cache Quiz 2

1. Processor produces the following sequences of accesses:
   - 0x3F1F
   - 0x3F2F
   - 0x3F2E
   - 0x3E1F
2. What does the cache contain after these accesses? Assume a cache line is 32B and there are 8 such lines.
   - Lowest 5 bits are block offset
   - Next 3 bits are index

Line	Contents
0	0x3E1F
1	0x3F2F,0x3F2E
2	
3	
4	
5	
6	
7	

Set Associative Caches

1. N-way set-associative: A block can be in one of N lines
2. Cache is divided into sets
   - Use some bits of the cache's address to determine which set it belongs to
3. 2-way set associative means there are two blocks in each set, not that there are two sets
4. A 2-way set-associative cache with 8 cache lines would have $8/2 = 4$ sets

Offset - Index - Tag for Set Associative

1. Least significant bits are still block offset
2. Next bits are index -> tell us which set to go to
3. Rest of the bits are still the tag
4. When we get a tag, we need to check every way in the set

2 Way Set Associative Quiz

1. Processor produces the following sequences of accesses:
   - 0xF303
   - 0xF503
   - 0xF563
   - 0xEF63
2. What does the cache contain after these accesses? Assume a cache line is 32B and there are 8 such lines in a 2-way set-associative cache (4 sets).

Line	Contents
0	0xF303
1	0xF503
2	
3	
4	
5	
6	0xF563
7	0xEF63

Fully Associative Cache

1. Any block can map to any line
2. Least significant bits are still block offset
3. A fully associative cache has no index bits; all remaining bits are the tag

Direct Mapped and Fully Associative

1. Direct-mapped = 1-way set associative
2. Fully associative = N-way set associative (N = number of lines)
3. Breaking down the address:
   - Offset = log2(block size)
   - Index = log2(number of sets)
   - Tag = Remaining bits
   - Determine this in order: offset -> index -> tag

Cache Replacement

1. Cases where replacement is needed:
   - Set is full
   - Miss -> Need to put new block in set
2. Which block do we kick out?
   - Random
   - FIFO
   - Least recently used
     - Preferred policy, but difficult/expensive to determine absolutely
3. LRU is typically approximated
   - Not most recently used (NMRU) - Pick randomly from any block that wasn't accessed most recently

Implementing LRU

1. Keep an LRU counter for each block in a cache
   - Additional state (data, tag, valid, LRU counter)
   - For 4-way set associative cache, 0 is LRU, 3 is MRU
   - When a block is accessed, set its counter to 3 and decrement all others
   - If a block is accessed and it isn't the LRU or MRU, set its counter to 3
     - However, we can't simply decrement all other counters
     - Instead, only decrement the blocks whose counters were greater than the accessed block's counter
2. For an N-way set associative cache, we need N log2(N)-bit counters
   - Cost is relatively high
3. Need to change N counters on each access (even hits)
   - Additional energy required

LRU Quiz

1. 8-way set associative cache with accesses A, B, A, D, K
   - 0 is initial state
   - 1 = A, 2 = B, 3 = A, 4 = D, 5 = K

0	Count	1	Count	2	Count	3	Count	4	Count	5	Count
A	7	A	7	A	6	A	7	A	6	A	5
B	3	B	3	B	7	B	6	B	5	B	4
C	2	C	2	C	2	C	2	C	2	C	1
D	6	D	6	D	5	D	5	D	7	D	6
E	5	E	5	E	4	E	4	E	4	E	3
F	1	F	1	F	1	F	1	F	1	F	0
G	4	G	4	G	3	G	3	G	3	G	2
H	0	H	0	H	0	H	0	H	0	K	7

Write Policy

1. Allocate policy: Do we insert blocks we write?
   - Write-allocate: Brings the block we write into the cache
   - No-write-allocate: Don't bring the block we write into the cache
2. Most modern caches are write-allocate because there is some locality between reads and writes
3. Do we write just to cache or also to memory?
   - Write-through: Update memory immediately
   - Write-back: Write to cache, write to memory when replaced
4. Most modern caches are write-back because it delays accessing memory unnecessarily
5. If you have a write-back cache, you want to have a write-allocate cache because writes need to go to the cache since they aren't going to memory

Write Back Caches

1. If a block has been written since bringing in from memory, we must replace it when we write to memory
2. If a block hasn't been written since bringing in from memory, there is no need to write that block back to memory
3. Dirty bit: Every block has a bit to designate if it has been updated
   - 0: Block is clean (not written since last brought from memory)
   - 1: Block is dirty (need to write back on replacement)

Write Back Cache Example

1. A processor performs the following sequence of accesses. Assume A, B, C map to different sets in the cache, but E maps to A and F maps to B
   - Write A
   - Read A
   - Read B
   - Read C
   - Write C
   - Read E
2. The final state of the cache is:
   - Dirty bit is set whenever the instruction is a write
   - When E overwrites A, we must write the data in A back to memory

V	Tag	D	Data
1	E	0	E
1	F	0	F
1	C	1	C

Write Back Cache Quiz

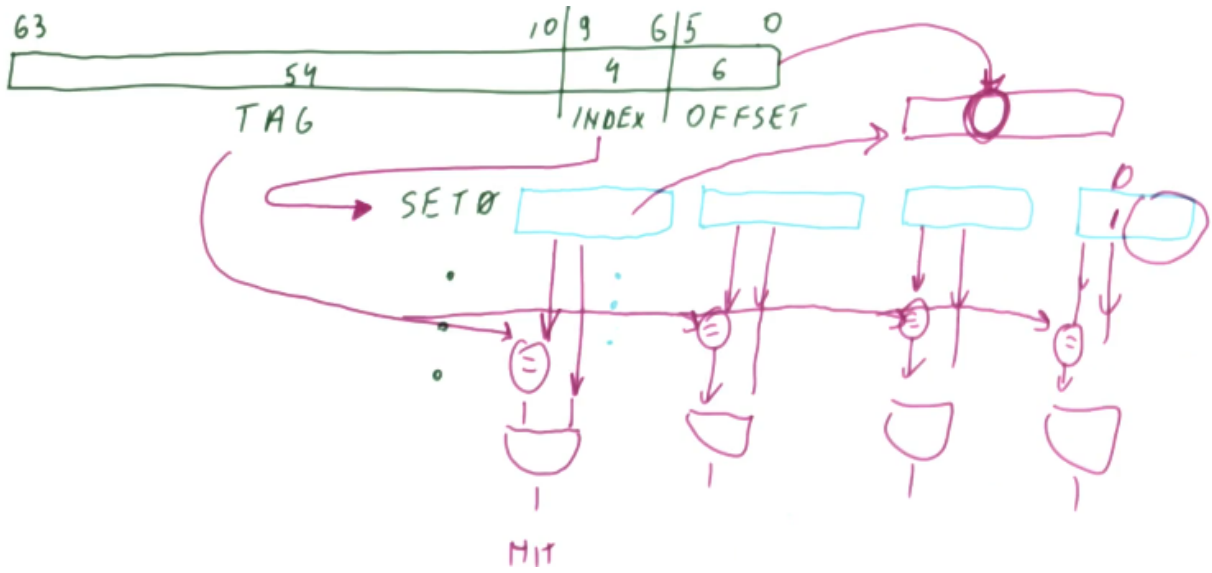
1. A processor performs the following sequence of accesses. Assume A, B, C, D all map to the same set in the cache
   - 1: Read A
   - 2: Read B
   - 3: Write B
   - 4: Read C
   - 5: Read D
   - 6: Write D
2. The final state of the cache is:
   - We had 4 misses
   - We had 1 write back

Access	V	D	Tag	Miss?	Write back?
0	0	1	A		
1	1	0	A	X	
2	1	0	B	X	
3	1	1	B		
4	1	0	C	X	X
5	1	0	D	X	
6	1	1	D		

Cache Summary Part 1

1. 4kB, 4-way set associative with 64-byte line, write-back, write-allocate
2. 64-bit address
   - 6 bits for offset
   - Number of blocks = 2 ^ 12 / 2 ^ 6 = 2 ^ 6
   - Number of sets = Number of blocks / Number of blocks in set = 2 ^ 6 / 4 = 16
   - 4 bits for index
   - 54 remaining bits are the tag
3. Our cache has the following:
   - Valid bit
   - Dirty bit (since it's write back)
   - Tag (54 bits)
   - 2-bit LRU Counter since the cache is set associative
   - Each cache line has 58 bits in addition to the data

Cache Summary Part 2



Cache Summary

Cache Summary Quiz 1

1. 256B, 2-way set associative cache with 32B line, write-back, write-allocate
2. 32-bit address

- 5 bits for offset (log2(32))
- Number of blocks = 2 ^ 8 / 2 ^ 5 = 2 ^ 3
- Number of sets = 2 ^ 3 / 2 / 4
- 2 bits for index

3. Answer the following questions:
   - Tag bits are 7 - 31
   - Index bits are 5 - 6
   - Offset bits are 0 - 4

Cache Summary Quiz 2

1. Consider the following sequence of accesses:
   - 1. LW 0xBCDE0000
   - 2. LW 0xCDEF0000
   - 3. SW 0xBCDE0000
   - 4. SW 0xCDEF0004
   - 5. SW 0xBCDE0000
2. How many cache misses are there?
   - 1. Miss
   - 2. Miss
   - 3. Hit
   - 4. Hit
   - 5. Hit
3. How many blocks are written back to memory?
   - 1. Not dirty
   - 2. Not dirty
   - 3. Dirty, no write back
   - 4. Dirty, no write back
   - 5. Dirty, no write back

Conclusion

1. Covered the concerns and choices when designing caches as well as how caches function