Lab 5

SCT212-0051/2021 MAKUMI BEDAN NGUGI

EXAMPLE 1

Problem

Consider a computer system with a first-level data cache with the following characteristics: size: 16KBytes; associativity: direct-mapped; line size: 64Bytes; addressing: physical. The system has a separate instruction cache and you can ignore instruction misses in this problem. This system is used to run the following code:

```
for (i=0; i<4096; i++)
X[i] = X[i] * Y[i] + C
```

Assume that both X and Y have 4096 elements, each consisting of 4 bytes (single precision floating point). These arrays are allocated consecutively in physical memory. The assembly code generated by a naive compiler is the following:

```
loop: lw f2, 0(r1) # load X[i]
lw f4, 0(r2) # load Y[i]
multd f2, f2, f4 # perform the multiplication
addd f2, f2, f0 # add C (in f0)
sw 0(r1), f2 # store the new value of X[i]
addi r1, r1, 4 # update address of X
addi r2, r2, 4 # update address of Y
addi r3, r3, 1 # increment loop counter
bne r3, 4096, loop # branch back if not done
```

a. How many data cache misses will this code generate? Breakdown your answer into the three types of misses. What is the data cache miss rate?

Problem Setup Analysis:

- **Cache:** 16KB, Direct-Mapped, 64B line size.
    - Number of blocks/sets = 16384 / 64 = 256 sets (indices 0-255).
    - Index bits = log2(256) = 8.
    - Offset bits = log2(64) = 6.
- **Data:** Arrays X and Y, 4096 elements * 4 Bytes/element = 16384 Bytes (16KB) each.
- **Memory Layout:** X and Y are consecutive. If X starts at AddrX, Y starts at AddrY = AddrX + 16384.
- **Access Pattern per Iteration:** Read X[i], Read Y[i], Write X[i].
- **Elements per Cache Block:** 64 Bytes / 4 Bytes/element = 16 elements.

Address Mapping:

Since the cache size is 16KB (16384 Bytes) and Y starts exactly 16384 bytes after X:

- Addr(Y[i]) = Addr(X[i]) + 16384
- The cache index is determined by bits 6 through 13 of the address (8 index bits, 6 offset bits).
- Adding 16384 (which is 2^14) to an address effectively flips bit 14, leaving bits 6-13 (the index bits for *this* cache) unchanged.
- **Therefore, X[i] and Y[i] always map to the exact same cache set (index).** Because it's direct-mapped, they map to the same cache line.

a. Cache Misses and Miss Rate (Original Code)

1. **lw f2, 0(r1) (Read X[i]):**
   - *Compulsory Miss:* The first time an element within a specific 64B block of X is accessed, it causes a miss. The block is fetched. This happens once for each of the 256 blocks of X.
   - *Conflict Hit/Miss:* If the cache line currently holds the Y block data (due to the previous iteration's Y access), this read of X will be a **conflict miss**, evicting the Y block. If it holds the correct X block (from the previous iteration's write), it's a hit.
   - *Hit:* If accessing an element within an X block that is already in the cache (and wasn't just evicted by Y).
2. **lw f4, 0(r2) (Read Y[i]):**
   - *Compulsory Miss:* The first time an element within a specific 64B block of Y is accessed, it causes a miss.
   - *Conflict Miss:* Since X[i] and Y[i] map to the same line, and the lw X[i] likely just brought the X block in (or it was already there), this access to Y[i] will **always conflict** with the X block. It forces the Y block to be loaded, **evicting the X block**. This will be a miss (either compulsory or conflict).
3. **sw 0(r1), f2 (Write X[i]):**
   - *Conflict Miss:* The preceding lw Y[i] just loaded the Y block into the cache line, evicting the X block. Therefore, this write attempt to X[i] will **always find that the required X block is not present**. It causes a **conflict miss**. The X block must be fetched again (assuming Write Allocate), evicting the Y block.

Counting the Misses:

- **Iteration i=0 (First element of a block):**

  - Read X[0]: Compulsory Miss (fetches X block 0)
  - Read Y[0]: Compulsory Miss (Y block 0 maps to same line, evicts X block 0, fetches Y block 0)
  - Write X[0]: Conflict Miss (X block 0 needed, but Y block 0 is present, evicts Y block 0, fetches X block 0)
  - *Misses = 3*
- **Iteration i=1 to 15 (Remaining elements in the block):**
  - Read X[1]: Hit (X block 0 is currently in cache)
  - Read Y[1]: Conflict Miss (Y block 0 maps to same line, evicts X block 0, fetches Y block 0)
  - Write X[1]: Conflict Miss (X block 0 needed, but Y block 0 is present, evicts Y block 0, fetches X block 0)
  - *Misses = 2*
- **Total Misses per Block (16 iterations):** 3 (for first iter) + 2 * 15 (for next 15 iters) = 3 + 30 = 33 misses.
- **Number of Blocks:** The loop processes 4096 elements / 16 elements/block = 256 blocks.
- **Total Data Cache Misses:** 33 misses/block * 256 blocks = **8448 misses**.

Breakdown:

- **Compulsory:** First read of each X block (256) + First read of each Y block (256) = **512 misses**.
- **Capacity:** 0 misses. The entire dataset (32KB) doesn't fit, but the access pattern causes conflicts before capacity limits are the primary driver.
- **Conflict:** Total Misses - Compulsory Misses = 8448 - 512 = **7936 misses**.

Data Cache Miss Rate:

- Total Data Accesses = 3 (Read X, Read Y, Write X) * 4096 iterations = 12288 accesses.
- Miss Rate = Total Misses / Total Data Accesses = 8448 / 12288 = **0.6875 or 68.75%**.

Answer:

- Total Misses: 8448
- Breakdown: Compulsory=512, Capacity=0, Conflict=7936
- Miss Rate: 68.75%

b. Provide a software solution that significantly reduces the number of data cache misses. How many data cache misses will your code generate? Breakdown the cache misses into the three types of misses. What is the data cache miss rate?

Software Solution: Data Padding

- **Idea:** Insert a small amount of unused space (padding) between the allocation of array X and array Y in memory. The goal is to shift the starting address of Y so that Y[i] maps to a *different* cache index than X[i].
- **Implementation:** Allocate X normally. Allocate, say, 64 bytes (or any amount not a multiple of 16KB, preferably related to cache lines) of padding. Then allocate Y.
    - Now Addr(Y[i]) = Addr(X[i]) + 16384 + PaddingSize.
    - If PaddingSize = 64 bytes (1 cache line):
        - Index(Y[i]) = (Addr(X[i])/64 + 16384/64 + 64/64) % 256
        - Index(Y[i]) = (Addr(X[i])/64 + 256 + 1) % 256
        - Index(Y[i]) = (Index(X[i]) + 1) % 256
    - Now X[i] and Y[i] map to *adjacent* cache lines, not the same one. The conflict is eliminated.

Analysis with Padding (e.g., 64 Bytes):

1. **lw f2, 0(r1) (Read X[i]):**
    - Compulsory miss for the first access to each block (256 total).
    - Hits for subsequent accesses within the block.
    - *Crucially, loading Y will not evict this block.*
2. **lw f4, 0(r2) (Read Y[i]):**
    - Compulsory miss for the first access to each block (256 total).
    - Hits for subsequent accesses within the block.
    - *Loading Y does not evict X.*
3. **sw 0(r1), f2 (Write X[i]):**
    - Since the X block was loaded by lw X[i] and *not* evicted by lw Y[i], this write will now be a **hit**.

Counting the Misses (with Padding):

- Misses per block (16 elements): 1 miss for lw X (compulsory) + 1 miss for lw Y (compulsory) = 2 misses.
- Total Misses = 2 misses/block * 256 blocks = **512 misses**.

Breakdown (with Padding):

- **Compulsory:** 512 (256 for X, 256 for Y).

- **Capacity:** 0.
- **Conflict:** 0.

Data Cache Miss Rate (with Padding):

- Total Data Accesses = 12288 accesses.
- Miss Rate = 512 / 12288 ≈ **0.04167 or 4.17%**.

Answer:

- Software Solution: Add padding (e.g., 64 bytes) between the allocation of arrays X and Y in memory to offset their mapping in the cache.
- Total Misses: 512
- Breakdown: Compulsory=512, Capacity=0, Conflict=0
- Miss Rate: approx 4.17%

c. Provide a hardware solution that significantly reduces the number of data cache misses. You are free to alter the cache organization and/or the processor. How many data cache misses will your code generate? Breakdown the cache misses into the three types of misses. What is the data cache miss rate?

Hardware Solution: Increase Associativity

- **Idea:** Allow each memory block to map to more than one possible cache line within a set. This reduces the chance of frequently used blocks evicting each other just because they happen to share the same index.
- **Implementation:** Change the cache from Direct-Mapped (1-way set associative) to **2-Way Set Associative**, keeping the total size (16KB) and line size (64B) the same.
    - Cache Size = 16 KB, Block Size = 64 B => Num Blocks = 256.
    - Associativity = 2-way.
    - Number of Sets = Num Blocks / Associativity = 256 / 2 = 128 sets.
    - Now, two different blocks that map to the same set index can coexist in the cache (one in each "way").

Analysis with 2-Way Set Associativity:

- As determined before, X[i] and Y[i] still map to the *same set index* even with 128 sets. Let this index be S.

1. **lw f2, 0(r1) (Read X[i]):**
    - *Compulsory Miss:* On first access to an X block mapping to set S. Loads the X block into, say, Way 0 of set S. (256 total compulsory X misses).

- *Hit:* On subsequent accesses to elements in that X block, it's found in Way 0.

2. **lw f4, 0(r2) (Read Y[i]):**
    - *Compulsory Miss:* On first access to a Y block mapping to set S. Since Way 0 is occupied by X, the Y block is loaded into Way 1 of set S. The X block in Way 0 is *not* evicted. (256 total compulsory Y misses).
    - *Hit:* On subsequent accesses to elements in that Y block, it's found in Way 1.
3. **sw 0(r1), f2 (Write X[i]):**
    - The X block required for the write is currently residing in Way 0 of set S. The lw Y[i] did not evict it.
    - **Result:** This write is now a **hit**.

Counting the Misses (2-Way Associative):

- Misses per block (16 elements): 1 miss for lw X (compulsory) + 1 miss for lw Y (compulsory) = 2 misses.
- Total Misses = 2 misses/block * 256 blocks = **512 misses**.

Breakdown (2-Way Associative):

- **Compulsory:** 512 (256 for X, 256 for Y).
- **Capacity:** 0.
- **Conflict:** 0.

Data Cache Miss Rate (2-Way Associative):

- Total Data Accesses = 12288 accesses.
- Miss Rate = 512 / 12288 ≈ **0.04167 or 4.17%**.

Answer:

- Hardware Solution: Change the cache associativity from Direct-Mapped to 2-Way Set Associative (keeping total size and line size constant).
- Total Misses: 512
- Breakdown: Compulsory=512, Capacity=0, Conflict=0
- Miss Rate: approx 4.17%