BCT 2408 Computer Architecture-II: Lab 5 Solution

Name: Jeff Gicharu
Reg No: SCT212-0053/2021

This document provides the solutions to the problems presented in Lab 5, focusing on analyzing cache performance for a given code loop and exploring hardware/software optimizations.

E1: Cache Miss Analysis and Optimization (groups of 2 - 35 min)

Problem

Consider a computer system with a first-level data cache with the following characteristics:

- size: 16 KBytes
- associativity: direct-mapped
- line size: 64 Bytes
- 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?

(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?

(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?

- **Analysis Setup:**
  - Cache Size = 16 KB = 16384 Bytes
  - Block Size = 64 Bytes
  - Associativity = 1 (Direct Mapped)
  - Number of Cache Lines = Cache Size / Block Size = 16384 / 64 = 256 lines
  - Element Size = 4 Bytes
  - Elements per Block = Block Size / Element Size = 64 / 4 = 16 elements
  - Array Size (X or Y) = 4096 elements * 4 Bytes/element = 16384 Bytes (Exactly the cache size)
  - Total Iterations = 4096
  - Memory References per Iteration = 3 (lw X, lw Y, sw X)
  - Total Memory References = 4096 iterations * 3 refs/iteration = 12288 references
  - **Key Conflict:** Since X and Y are 16KB each and allocated consecutively, and the cache is 16KB direct-mapped, Address(Y[i]) maps to the *same cache line index* as Address(X[i]). Index(Y[i]) = Index(X[i]).
- **Answer (a): Misses and Miss Rate (Naive Code)**
  Let's trace the accesses within an iteration i:
  1. lw X[i]: Accesses cache line for X[i].
     - If i mod 16 == 0 (first access to this block): **Compulsory Miss**. Fetches X block.
     - If i mod 16 != 0: Hit (assuming block is still there).
  2. lw Y[i]: Accesses cache line for Y[i]. This maps to the *same line* as X[i].
     - This access *always* conflicts with the X block. It evicts the X block.
     - If i mod 16 == 0: **Compulsory Miss** (for Y's block). Fetches Y block.
     - If i mod 16 != 0: **Conflict Miss**. Fetches Y block.
  3. sw X[i]: Accesses cache line for X[i]. This maps to the *same line*.

        - The lw Y[i] just evicted the X block.
        - This access *always* results in a **Conflict Miss**. Fetches X block again.

Miss Count per Iteration:

- If i mod 16 == 0: Miss (X load) + Miss (Y load) + Miss (X store) = **3 misses**.
- If i mod 16 != 0: Hit (X load) + Miss (Y load) + Miss (X store) = **2 misses**.

Total Misses Calculation:

- Number of iterations with i mod 16 == 0 = 4096 / 16 = 256.
- Number of iterations with i mod 16 != 0 = 4096 - 256 = 3840.
- Total Misses = (256 iterations * 3 misses/iter) + (3840 iterations * 2 misses/iter)
- Total Misses = 768 + 7680 = **8448 misses**.

Miss Breakdown (3 Cs):

- **Compulsory Misses:** First access to each block of X and each block of Y.
    - Blocks in X = 4096 elements / 16 elem/block = 256 blocks.
    - Blocks in Y = 4096 elements / 16 elem/block = 256 blocks.
    - Total Compulsory = 256 (for X loads) + 256 (for Y loads) = **512 misses**.
- **Capacity Misses:** 0. The working set (one block of X, one block of Y at a time) fits easily within the 16KB cache. Misses are not due to lack of overall space.
- **Conflict Misses:** All remaining misses are due to X and Y mapping to the same cache lines.
    - Conflict Misses = Total Misses - Compulsory Misses = 8448 - 512 = **7936 misses**.

Data Cache Miss Rate:

- Miss Rate = Total Misses / Total Memory References
- Miss Rate = 8448 / 12288 = **0.6875 (or 68.75%)**.

- **Answer (b): Software Solution**
  Solution: Modify memory allocation. Add padding between arrays X and Y, or ensure their base addresses are such that Index(X[i]) != Index(Y[i]). For example, place array Y starting at Address(X[0]) + 16KB + 64B (offset by at least one cache line). Alternatively, merge the arrays into an array of structures: struct { float x; float y; } Z[4096];. We'll analyze the padding/offsetting approach as it's simpler to implement post-allocation.
  Miss Analysis (with Padding/Offsetting):
    - Now lw Y[i] does not conflict with lw X[i] or sw X[i].
    1. lw X[i]: Miss only on i mod 16 == 0 (**Compulsory Miss**). Hit otherwise.

2. lw Y[i]: Miss only on i mod 16 == 0 (**Compulsory Miss**). Hit otherwise.
3. sw X[i]: Always **Hit** (The X block loaded by lw X[i] remains in the cache as lw Y[i] maps elsewhere).

Total Misses Calculation:

- Misses only occur on the first load of each block of X and Y.
- Total Misses = Compulsory Misses for X + Compulsory Misses for Y
- Total Misses = 256 + 256 = **512 misses**.

Miss Breakdown (3 Cs):

- **Compulsory Misses: 512 misses**.
- **Capacity Misses: 0 misses**.
- **Conflict Misses: 0 misses**. (The software change eliminated them).

Data Cache Miss Rate:

- Miss Rate = Total Misses / Total Memory References
- Miss Rate = 512 / 12288 ≈ **0.0417 (or 4.17%)**.

- **Answer (c): Hardware Solution**
  Solution: Change the cache associativity. Increase it from direct-mapped (1-way) to **2-way set-associative**. Keep the size (16KB) and line size (64B) the same.
  New Cache Parameters:
  - Associativity = 2
  - Number of Sets = Cache Size / (Block Size * Associativity) = 16384 / (64 * 2) = 128 sets.

  Miss Analysis (2-way Set-Associative):
  - X[i] and Y[i] still map to the *same set index*, but now they can reside in different *ways* within that set without evicting each other.

  1. lw X[i]:
     - If i mod 16 == 0: **Compulsory Miss**. Fetches X block (e.g., into Way 0).
     - If i mod 16 != 0: Hit.
  2. lw Y[i]:
     - If i mod 16 == 0: **Compulsory Miss**. Fetches Y block (e.g., into Way 1).
     - If i mod 16 != 0: Hit.
  3. sw X[i]: Always **Hit** (The X block loaded by lw X[i] remains in its way, not evicted by lw Y[i]).

  Total Misses Calculation:
  - Misses only occur on the first load of each block of X and Y.
  - Total Misses = Compulsory Misses for X + Compulsory Misses for Y

- Total Misses = 256 + 256 = **512 misses**.

Miss Breakdown (3 Cs):

- **Compulsory Misses: 512 misses**.
- **Capacity Misses: 0 misses**.
- **Conflict Misses: 0 misses**. (The hardware change eliminated them).

Data Cache Miss Rate:

- Miss Rate = Total Misses / Total Memory References
- Miss Rate = 512 / 12288 ≈ **0.0417 (or 4.17%)**.