Computer Architecture – tutorial 5

# Context, Objectives and Organization

The goal of the quantitative exercise in this tutorial is to explore qualitatively and quantitatively some hardware and software optimizations to improve cache performance.

# E1: CAR September 2003 exam P2, groups of 2 – 35 min

*Problem*

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

1. **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?**

Cache Parameters:

* Line size = 64 B → 16 elements/line (each element = 4 B).
* Cache size = 16 KB → 256 lines (16 KB/64 B).

Memory Layout:

* X and Y are contiguous. Assuming X starts at address 0:
  + X: Addresses 0x0000–0x3FFF (16 KB).
  + Y: Addresses 0x4000–0x7FFF (16 KB).

Conflict Misses:

* Direct-mapped cache maps addresses to lines via:

Line index = mod 256

* X[i] and Y[i] map to the same cache line (since 0x4000 mod 256 = 0x000 mod 256 = 0)

Miss Breakdown:

1. Compulsory Misses:

First access to each line of X and Y:

* + - X: 16 KB / 64 B = 256 misses.
    - Y: 256 misses.

Total: 256 + 256 = 512 misses.

1. Capacity Misses:

None (each array fits entirely in cache).

1. Conflict Misses:

Every access to Y[i] evicts the line of X[i] (and vice versa).

Total: 4096×2 = 8192 misses (each X[i] and Y[i] reloaded every iteration).

Total Misses:

512 (compulsory) + 8192 (conflict) = 8704 misses.

Miss Rate:

=

1. **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?**

**Solution**: **Loop Blocking (Tiling)**

Process data in smaller blocks that fit in cache.

Block size = cache size / 2 = 8 KB → 128 lines.

Code:

***for (int b = 0; b < 4096; b += 2048) { // Process 2048 elements (8 KB) per block***

***for (int i = b; i < b + 2048; i++) {***

***X[i] = X[i] \* Y[i] + C;***

***}***

***}***

Cache miss’s breakdown;

1. **Compulsory Misses**: Unchanged (512).
2. **Conflict Misses**:

Each block fits in cache without conflicts.

Only compulsory misses per block reload.

Total: 512×2=1024512×2=1024 misses (2 blocks).

Total misses: 512+1024 = 1536

Miss rate:

1. **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?**

Solution:

Increase Associativity (2-way Set-Associative Cache)

Allows X[i] and Y[i] to coexist in cache.

Miss Breakdown:

Compulsory Misses: Unchanged (512).

Conflict Misses: Eliminated (no evictions).

Total Misses: 512.

Miss rate:

Therefore;

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Baseline** | **Software (Blocking)** | **Hardware (2-way)** |
| Total Misses | 8704 | 1536 | 512 |
| Miss rate (%) | 70.8 | 12.5 | 4.2 |