**TANIA MAINA.**

**SCT212-0179/2021.**

**BCT2408: COMPUTER ARCHTECTURE.**

**E1:**

**Cache parameters:**

* Cache size: 16 KB = 16 × 1024 = 16,384 bytes
* Line size: 64 bytes
* Associativity: Direct-mapped
* Addressing: Physical

Compute:

Number of cache lines = Cache size / Line size = 16,384 / 64 = 256 lines

*Data layout and access pattern*

Each array element (X[i] and Y[i]) = 4 bytes

Arrays X and Y have 4096 elements → 4096 × 4 = 16,384 bytes per array

Total data involved: X (16 KB) + Y (16 KB) = 32 KB

Arrays are consecutively allocated in physical memory, so:

If X starts at address A, Y starts at A + 16,384

**Access pattern in the loop:**

*Each loop iteration does:*

Load X[i] → lw f2, 0(r1)

Load Y[i] → lw f4, 0(r2)

Store X[i] → sw 0(r1), f2

Therefore, each iteration accesses:

2 loads (X[i] and Y[i])

1 store (X[i])

The loop runs for 4096 iterations, so total accesses:

Loads: 4096 (X) + 4096 (Y) = 8192

Stores: 4096 (X)

Total accesses: 8192 + 4096 = 12,288 memory accesses

**Cache analysis**

**1. Compulsory (Cold) Misses**

First time accessing a cache line = compulsory miss

Each cache line is 64 bytes → 64 / 4 = 16 elements per line

X has 4096 elements → 4096 / 16 = 256 cache lines

Y has 4096 elements → also 256 cache lines

Total cold misses: 256 (X) + 256 (Y) = 512

Because this is a direct-mapped cache and X and Y are contiguously allocated, they might conflict.

**2. Conflict Misses**

**Key point:**

Cache has 256 lines.

Each line is identified by its index = (address / 64) % 256

Since X and Y are 16 KB apart = exactly one cache size apart, they map to the same cache lines.

That means:

X[0] and Y[0] → both map to cache line 0

X[1] and Y[1] → both map to cache line 1

...

X[4095] and Y[4095] → map to line 255 (modulo 256)

So each X[i] and Y[i] access evicts the other.

Thus, in every iteration:

Load X[i] → miss (load into line n)

Load Y[i] → miss (evicts X[i] from line n)

Store X[i] → miss again (X[i] was just evicted by Y[i])

So, every single access is a miss.

Total:

Loads: 8192 misses (4096 X + 4096 Y)

Stores: 4096 misses (all stores miss too)

Total misses = 12,288

BUT:

Cold misses: only the first time for each cache line (512 total)

Conflict misses: the rest → 12,288 - 512 = 11,776 conflict misses

**3. Capacity misses**

Cache is large enough to hold either X or Y alone.

Total data = 32 KB, cache = 16 KB → Not large enough to hold both arrays.

**Miss Rate**

Total accesses = 12,288

Total misses = 12,288

Miss rate = 12,288 / 12,288 = 1.0 = 100%

This code causes 100% data cache misses due to:

* Direct-mapped cache (1-to-1 mapping)
* Arrays X and Y conflict in cache because they map to the same cache lines.
* Each access evicts the previous, causing every access to miss.

b)

**Software optimization strategy: Loop Interchange + Padding**

Since the issue is that X[i] and Y[i] map to the same cache lines, the simplest software fix is to adjust memory access pattern to prevent mapping conflicts.

Solution 1: Array Padding

Insert padding between arrays X and Y in memory so that they no longer map to the same cache lines. The original problem arises because:

X and Y are allocated back-to-back in physical memory.

They are 16 KB apart (equal to cache size).

So address\_of\_Y[i] ≡ address\_of\_X[i] mod 16KB → they map to the same cache line.

Padding by one cache line (64 bytes) breaks this alignment.

Modified Code (in C-style pseudocode)

float X[4096];  
float pad[16]; // 16 × 4 bytes = 64 bytes = 1 cache line  
float Y[4096];  
  
for (i = 0; i < 4096; i++) {  
 X[i] = X[i] \* Y[i] + C;  
}

**Cache behavior after padding**

Previously:

* 16 KB = 16,384 bytes
* Line size = 64 bytes
* Direct-mapped cache = 256 cache lines
* Each cache line holds 16 floats (64 bytes / 4)

Now:

X and Y no longer conflict → each maps to different cache lines

X will use lines 0–255

Y will also use lines 0–255, but offset by 1 line due to padding, so no overlap

Therefore, cache lines used by X and Y are now disjoint → no conflict misses

**Miss analysis**

**Array X:**

4096 elements, 16 per cache line → 256 lines

First access to each line = compulsory miss

Subsequent accesses = cache hit

Each X[i] is accessed twice per iteration (load and store) → second access hits

*Therefore:*

Compulsory misses: 256 (on first load of X[i])

Hit on X[i] store (since it's still in cache after load)

**Array Y:**

4096 elements, 16 per line = 256 lines

Each Y[i] is accessed once (load only)

First access to each line = compulsory miss

So: 256 misses

**Total:**

X loads: 4096

X stores: 4096

Y loads: 4096

*But misses:*

X: 256 (on first load per cache line), stores hit

Y: 256 (on first load per line)

Total memory accesses = 12,288 (4096 X loads, 4096 Y loads, 4096 X stores)

Total misses = 512

Miss rate = 512 / 12,288 ≈ 0.0417 = 4.17%

Conclusively with padding:

Misses reduced from 12,288 to 512

Miss rate reduced from 100% to 4.17%

All misses are cold misses; no conflict or capacity misses

c)

Hardware optimization strategy: Use a 2-Way set associative cache. In a direct-mapped cache, each memory block maps to exactly one cache line → easy for multiple addresses (like X[i] and Y[i]) to collide.

In a 2-way set associative cache, each set contains 2 lines, so two blocks that would map to the same index can coexist without eviction.

**New cache configuration:**

* Parameter, Value
* Cache Size, 16 KB (16,384 bytes)
* Line Size, 64 bytes
* Associativity, 2-way set associative
* Number of Sets, 16KB / (64 × 2) = 128
* Addressing, Physical

In the original code, X[i] and Y[i] are 16 KB apart, and both arrays map to the same cache set index in a direct-mapped cache, causing constant eviction. In a 2-way set associative cache, each set holds two lines, so X[i] and Y[i] can share a set without evicting each other.

**Memory access pattern review**

Each iteration accesses:

Load X[i]

Load Y[i]

Store X[i]

Same as before → total 12,288 memory accesses

**Miss analysis with 2-Way Associativity**

Each cache line holds 16 elements:

4096 elements per array / 16 per line = 256 lines per array

Total: 512 unique lines (X + Y)

Cache has:

128 sets × 2 lines per set = 256 total lines

But because we have 2-way associativity, we can store both X[i] and Y[i] in the same set without eviction.

**Result:**

**Compulsory misses:**

256 lines for X → 256 cold misses (first load)

256 lines for Y → 256 cold misses

Total compulsory = 512

**Conflict misses:**

0, because 2-way associativity avoids eviction

**Capacity misses:**

0, because working set fits in cache: 32 KB of data, but only 4 KB accessed at any time due to locality

Reuse:

X[i] is accessed twice (load and store) within a few instructions → data still in cache

Y[i] is accessed only once → still causes one miss per cache line, but no re-use

**Miss Rate**

Total memory accesses = 12,288

Total misses = 512

Miss rate = 512 / 12,288 ≈ 4.17%

Conclusively by replacing direct-mapped cache with 2-way set associative cache

Misses drop from 12,288 (100%) → 512 (4.17%)

All remaining misses are cold misses

Completely eliminates conflict misses