Recitation 5

09/27/2024

Logistics

- Lab 1 done
- HW 3 out now, due next Thursday, October 3
- Lab 2 releases Monday, September 30
    - Due Monday after fall break (3 weeks, plenty of time)

Outline

1. HW 2 Review
2. Lab 2 Overview
3. Lab 1 Postmortem

HW 2 Review

Question 1

Recall the RISC-V 5 stage pipeline **WITHOUT DATA FORWARDING** from lecture 5. Examine the following program run by the machine. Assume that without a stall, each instruction only stays in a stage for one cycle.

```
add x1, x0, x0
add x3, x1, x2
add x4, x1, x3
add x3, x3, x4
addi x1, x1, 0x1
addi x2, x2, 0x2
```

Q1

```
add x1, x0, x0
add x3, x1, x2
add x4, x1, x3
add x3, x3, x4
addi x1, x1, 0x1
addi x2, x2, 0x2
```

6 stall cycles
16 total cycles
6 instructions

IPC = 6/16 = 0.375

	Fetch	Decode	Execute	Memory	Writeback
t1	add0				
t2	add1	add0			
t3	add2	add1	add0		
t4	add2	add1	STALL	add0	
t5	add2	add1	STALL	STALL	add0
t6	add3	add2	add1	STALL	STALL
t7	add3	add2	STALL	add1	STALL
t8	add3	add2	STALL	STALL	add1
t9	addi0	add3	add2	STALL	STALL
t10	addi0	add3	STALL	add2	STALL
t11	addi0	add3	STALL	STALL	add2
t12	addi1	addi0	add3	STALL	STALL
t13		addi1	addi0	add3	STALL
t14			addi1	addi0	add3
t15				addi1	addi0
t16					addi1

Q3

Suppose the following chunk of code is run extremely often when compiled in -O0 (no compiler tricks)

```
int foo(i) {
    int x = 0;
    if (i % 4 == 0)
        x += bar(i + 1);
    if (i % 2 == 0)
        x += bar(i);
    return x;
}
```

Explain why a simple 2-bit BHT branch predictor might not be as accurate as other branch predictors we talked about in lecture.

Lab 2 Overview

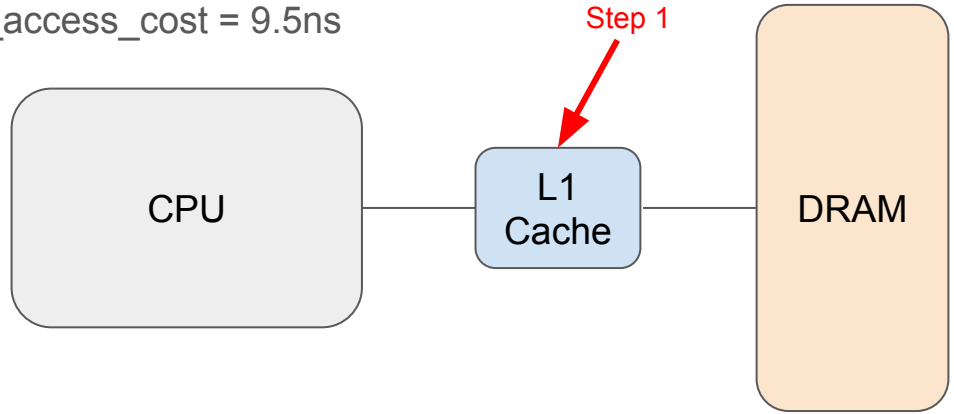
Step 1: Implement a Set-Associative Cache

- Simulate a set-associative cache, which takes in memory loads/stores from Pin, and updates the cache state after each memory access.
- The cache should have three configurables parameters:
  - Cache size, Block size, and Associativity
- You will sweep across reasonable values for these parameters.
- For each cache configuration, you must profile the SPEC workloads to get
  - total accesses, hit rate, and miss rate
- You will have to implement the required file I/O in the pintool yourself

Step 1: Implement a Set-Associative Cache Cont.

- You will then use Destiny to generate the access/miss latencies for each cache configuration.
- Using this data, you can now calculate the AMAT per cache configuration.
  AMAT (1-level cache) Formula:
  - AMAT = L1_hit_rate * L1_access_cost + L1_miss_rate * (L1_miss_cost + DRAM_access_cost)
  - Use DRAM_access_cost = 9.5ns

Example AMAT calculation in Lecture 7

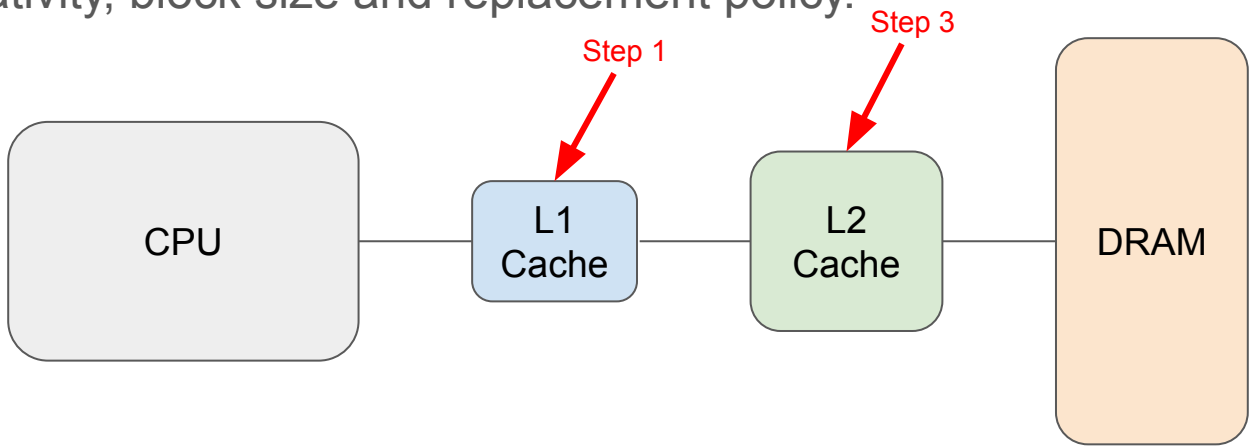


Step 2: Implement cache replacement policies

- Implement the replacement policies from class, and any other replacement policy that you think might improve performance.
  - random, LRU, and bit-PLRU
- Select three different reasonable values for cache size, associativity and block size, and compare the miss rates for each replacement policy.
- Comment on the absence or presence of a trend in the plot

Step 3: Implement a two-level cache hierarchy

- Extend the parametrized set-associative cache structure written in Step 1 to implement a two-level cache hierarchy
- Each level of the cache hierarchy can take a variable value for the cache size, associativity, block size and replacement policy.



Step 4: Design Space Exploration Tool

- Write a design space iteration tool
- Explore the design space created by the different configurations for the cache hierarchy. A configuration can be established as:
  - {(L1 associativity, L1 block size, L1 size, L1 replacement),
    (L2 associativity, L2 block size, L2 size, L2 replacement) }
- Your design space iteration tool should search the space of configuration tuples to find the optimal configuration
- You should evaluate each cache's leakage power, dynamic access power, and access latency using the Destiny memory modeling tool.
- Using these numbers from Destiny, you should report a configuration's performance as the Average Memory Access Time (AMAT)

Step 4: Design Space Exploration Tool Cont.

Again: Example AMAT calculation in Lecture 7

- AMAT (2-level cache) = L1_hit_rate * L1_access_cost + L1_miss_rate * (L1_miss_cost + L2_hit_rate * L2_access_cost + L2_miss_rate * (L2_miss_cost + DRAM_access_cost))
  - Still use DRAM_access_cost = 9.5ns
- Total Storage Budget: Your system has a total budget of 5MB of cache that you can split across the layers of cache in your system (only search within this budget)
- Minimization goal: Your design space iteration tool should minimize area and power if comparable configurations are equal in their performance; always opt for the lower area or lower power configuration.

Step 4: Design Space Exploration Tool Cont.

- Implementability: You must argue in your write-up that the cache hierarchy that you have proposed is implementable using supporting evidence from the Destiny tool.
  - Is the power consumed while doing reads and writes reasonable?
  - Is the tag storage overhead reasonable?
  - Are access latencies reasonable?
  - Your argument should consider that L1 accesses should be only a few cycles, meaning L1 access latency (especially for reads) is critical.

Lab 1 Postmortem

How was Lab 1?

- We’ll try to grade by next week

- Feedback / Any challenges doing the lab?