Lab 1: Simulating and Exploring Cache Behavior

Introduction

In this lab, I extended a timing simulator for a five-stage pipelined MIPS processor with separate L1 instruction and data caches. I wrote a benchmarking framework to analyze the effects of various cache design parameters and replacement policies. To test the cache's effectiveness in representative and meaningful situations, I extended the provided set of test cases.

Cache Implementation Details

Cache Data-Structure The instruction cache (I-Cache) and data cache (D-Cache) are based on a generic implementation of a set-associative cache in `src/cache.c` and `src/cache.h`. Since the timing simulator doesn't need to model the full data-path details, the cache data structure only tracks the most necessary metadata. This includes an array of the cache's sets, the blocks inside the sets and the related metadata for correctly handling cached data depending on the replacement policy. A cache access function checks if the memory address in its argument is a hit or a miss, updates the internal data structures and returns the incurred number of cycles to stall the calling stage. The data to be managed is handled directly using the provided `mem_read_32` and `mem_write_32` functions.

Replacement Policies I implemented three different replacement policies, which are activated with compiler flags.

1. LRU (Least Recently Used): Maintains a recency counter for each block. On insertion, evicts the block with the highest (oldest) counter value. On access, resets the accessed block's counter and increments all others.
2. Random: Selects a victim block uniformly at random from the set. This provides a simple baseline to compare other policies to.
3. RRIP (Re-Reference Interval Prediction): Implements a scan-resistant policy using 2-bit counters. New blocks are inserted with a distant re-reference prediction (value 2), and blocks are promoted on hits. This protects the cache from streaming access patterns that pollute LRU caches (Jaleel et al. 2010).

`Pipe_State` Integration The pipeline data-structure `Pipe_State` is extended with a I-Cache and D-Cache `struct`. Two stall cycle counters are added to track the remaining number of cycles. The cache organization, miss timings, dirty evictions and assumptions about the I-Cache / D-Cache are set as specified in the task requirements. Pipeline flushes to stages waiting on memory requests do not cancel the memory requests, the stall cycle counters are not reset.

Instruction Cache The I-Cache is used in the fetch stage of the pipeline. If the cache access determines a stall is necessary, the stage's stall cycle counter is set and the stage returns early. In later cycles, the stage only progresses once the stall cycle counter has been decremented till zero. A quirk of this mechanism is that the program counter `PC` can't be incremented during a `syscall` if the fetch stage is stalled. To ensure the program terminates with the correct `PC` value if the fetch stage is stalled, it is manually incremented in the `syscall` handler.

Data Cache The D-Cache is used in the mem stage of the pipeline and features the same stalling mechanism as the fetch stage. Due to the given specification, that dirty blocks *"are written immediately into main memory in the same cycle as when the new block is inserted into the cache"*, stores don't stall the pipeline beyond the initial read at the beginning of the stage.

Verification I verified correctness by comparing the register dumps with the baseline sim. I validated the cycle counts by checking the miss penalties (as multiples of 50) for the individually activated I-Cache and D-Cache on simple input programs.

Experimental Methodology

Benchmark Suite

As suggested in the task description, cache performance is measured using the `IPC`.

Input Tests

Parameter Sweep

To explore the cache behavior, I parametrized the sim implementation to support runtime configuration of the caches using a custom `bench` command defined in `shell.c`. A modified version of the `run.py` file generates the parameter sweep and runs the simulators in parallel. The sweep generates all physically valid combinations of the parameters for both I-Cache and D-Cache. All powers of two in the specified ranges are considered.

- Cache capacity : [`1KB`, `2KB`, ..., `1024KB`]
- Block size: [`4B`, `8B` ..., `512B`]
- Associativity: [1, 2, ..., 16]
- Replacement policy (LRU, Random, RRIP)

Results and Discussion

Effect of Cache Size

Effect of Block Size

Effect of Associativity

Replacement Policy Comparison

Discussion of Observations

Appendix

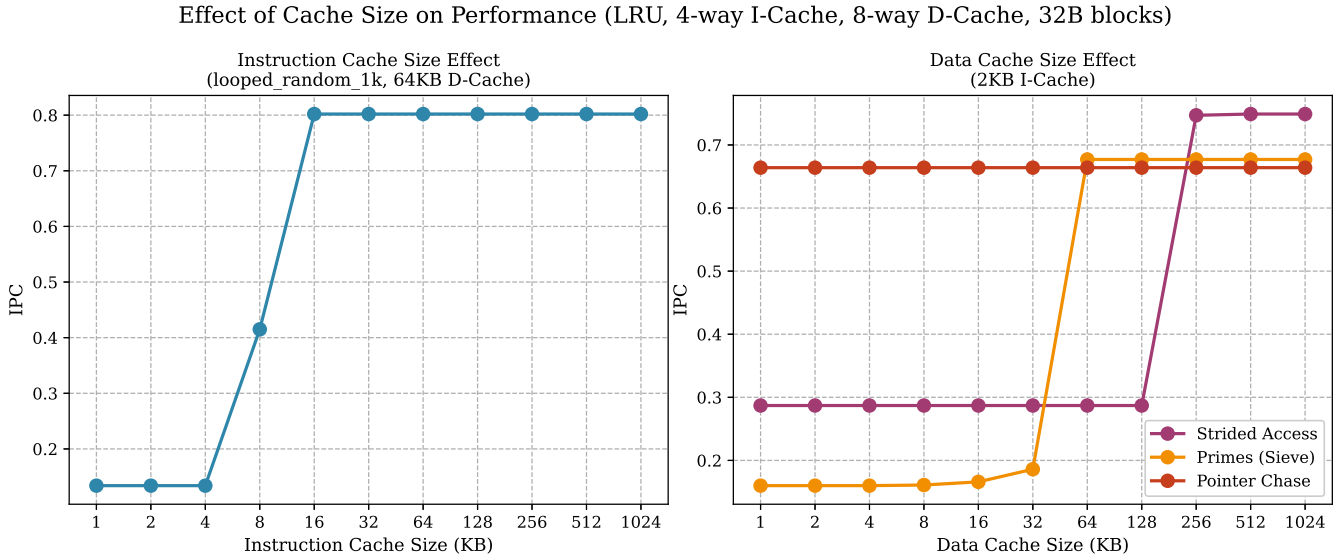

Figure 1: Cache Size Effect

Citations

Jaleel, Aamer, Kevin B. Theobald, Simon C. Steely, and Joel Emer. 2010. "High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)." *SIGARCH Comput. Archit. News* (New York, NY, USA) 38 (3): 60–71. https://doi.org/10.1145/1816038.1815971.