COL216 Assignment 3: L1 Cache Simulation with MESI Coherence

EntryNum1, EntryNum2

April 2025

Abstract

In this assignment, we implemennted L1 cache simulator for a quad-core processor system, adhering to the MESI cache coherence protocol. The simulator models write-back, write-allocate L1 data caches with LRU replacement, processing 32-bit memory addresses and 4-byte word accesses.

1 Implementation

The L1 cache simulator, implemented in C++, models a quad-core processor system where each core has its own L1 data cache, backed by main memory. It uses the MESI protocol for cache coherence, a write-back and write-allocate policy, and LRU replacement.

1.1 Code Organization and File Structure

- `main.cpp`:

  Program entry point, responsible for parsing command-line arguments and initializing the simulator.

- `cache.h`, `cache.cpp`:
  - `CacheLine` struct: Stores MESI state (`'M'`, `'E'`, `'S'`, `'I'`), tag, and LRU timestamp (`last_used_cycle`).
  - `Cache` class: Manages a 2D vector of cache lines (`num_sets` × `associativity`), tracks statistics (`misses`, `hits`, `evictions`, `writebacks`, `invalidations`, `data_traffic`), and handles read/write accesses and snooping.
  - Methods: `access` (read/write), `snoop`, `find_line`, `find_line_to_replace`, `handle_write_back`, `force_update_line`, address parsing (`get_set_index`, `get_tag`).

- `processor.h`, `processor.cpp`:
  - `Processor` class: Contains a `Cache`, trace file stream, current instruction (`current_op`, `current_addr`), and state (`is_stalled`, `stall_cycles`, `total_cycles`, `idle_cycles`, `reads`, `writes`).
  - Methods: `execute_cycle`, `process_instruction`, `read_next_instruction`, `snoop_request`, `invalidate_line`.

- `bus.h`, `bus.cpp`:
  - `Message` struct: Stores communication details (`core_id`, `address`, `new_state`).
  - `Bus` class: Maintains a list of processors, tracks bus state (`free_time`), and manages transactions (`total_transactions`, `total_traffic`).

  - Methods: `read` (BusRd), `invalidate` (BusRdX), `is_busy`.

- `simulate.h`, `simulate.cpp`:
  - `Simulator` class: Manages four `Processor` instances and a `Bus`, coordinates execution, and outputs results.
  - Methods: `run` (main simulation loop), `print_results`.

- `utils.h`, `utils.cpp`:
  - `Wrong_Parameters`: Prints usage help.
  - `print_simulation_parameters`: Outputs cache configuration.
  - `extract_bits`: Extracts specific bits from an address (unused but included for extensibility).

- `Makefile`:
  - `all`: Compiles sources into `L1simulate`.
  - `run`: Runs the simulator with default parameters (`app=test`, $s = 5$, $E = 2$, $b = 5$).
  - `clean`: Removes object files and executable.

1.2 Key Classes and Data Structures

- **CacheLine**: Represents a cache line with:
  - `state`: MESI state ('M', 'E', 'S', 'I').
  - `tag`: Cache line tag for address mapping.
  - `last_used_cycle`: Tracks LRU status.

- **Cache**: Manages a single L1 cache with:
  - `cache_lines`: 2D vector (`num_sets` × `associativity`) of `CacheLine`.
  - Parameters: `set_bits` (s), `associativity` (E), `block_bits` (b).
  - Statistics: `misses`, `hits`, `evictions`, `writebacks`, `invalidations`, `data_traffic`.
  - Methods: `access`, `snoop`, `update_line_state`, `force_update_line`, `get_set_index`, `get_tag`.

- **Processor**: Represents a core with:
  - `cache`: An instance of `Cache`.
  - `trace_file`: Reads memory references (R/W, address).
  - State: `is_stalled`, `stall_cycles`, `total_cycles`, `idle_cycles`, `reads`, `writes`, `has_instruction`, `has_new_instruction`.
  - Methods: `execute_cycle`, `process_instruction`, `read_next_instruction`, `snoop_request`, `invalidate_line`.

- **Bus**: Models the shared bus with:
  - `processors`: Vector of `Processor` instances.
  - `message`: Struct for inter-cache communication (`core_id`, `address`, `new_state`).
  - Statistics: `total_transactions`, `total_traffic`, `free_time`.
  - Methods: `read`, `invalidate`, `is_busy`.

- **Simulator**: Orchestrates the simulation with:
  - `processors`: Four `Processor` instances.
  - `bus`: Shared `Bus` instance.
  - Methods: `run`, `print_results`.

1.3 Execution Flow

The simulator operates in discrete cycles, processing memory references from four trace files (`appX_procY.trace`) in a lock-step manner. The flow is as follows:

1. **Program Startup** (`main.cpp`):
   - Parse command-line arguments using `getopt` (`-t`, `-s`, `-E`, `-b`, `-o`, `-h`).
   - Validate inputs; if invalid, call `Wrong_Parameters` and exit.
   - Create a `Simulator` instance with parsed parameters.
2. **Simulator Initialization** (`simulate.cpp`):
   - Instantiate four `Processor` objects, each with a unique ID (0–3), a trace file, and a `Cache` initialized with s, E, b.
   - Add processors to the `Bus`.
   - Each `Processor` opens its trace file and reads the first instruction (`read_next_instruction`).
3. **Simulation Loop** (`Simulator::run`):
   - Maintain a `global_cycle` counter starting at 1.
   - Continue until all processors have no instructions (`has_instruction = false`).
   - For each cycle:
     - Iterate over processors, calling `Processor::execute_cycle`, which returns a status code indicating whether an instruction was processed, the processor is stalled, or has no more instructions.
     - If no instructions were processed and the bus is busy (`free_time > 0`), add `free_time` to `idle_cycles` and `total_cycles` of active processors, then clear `free_time`.
     - If `bus.free_time = 0`, process any pending `bus.message` by updating the target cache line state via `Cache::force_update_line`.
     - Decrement `bus.free_time` and increment `global_cycle`.
4. **Processor Execution** (`processor.cpp`):
   - `execute_cycle`:
     - If `is_stalled`, decrement `stall_cycles`, increment `idle_cycles` if applicable, and return stalled status.
     - If `has_instruction = false`, return no instructions status.
     - Otherwise, call `process_instruction` and return success or waiting status.
   - `process_instruction`:
     - Determine operation type (`current_op = 'W'` for write, else read).
     - Call `Cache::access` with address, operation type, processor ID, bus, cycle, and `has_new_instruction` flag.

- If `access` returns `true`, read the next instruction; otherwise, set `is_stalled` and `stall_cycles`.

- `read_next_instruction`: Parse the next trace line into `current_op` and `current_addr`, increment `reads` or `writes`, and set `has_instruction` and `has_new_instruction`.

5. **Cache Operations** (`cache.cpp`):

   - **Read Access** (`access_read`):
     - Compute set index and tag from the address.
     - Check for a hit (`find_line`). If hit (state $\neq$ 'I'), update `last_used_cycle` and return `true`.
     - On miss:
       * Increment `misses` if `has_new_instruction`.
       * If bus is busy (`Bus::is_busy`), return `false`.
       * Select an LRU line (`find_line_to_replace`). If replacing a dirty line ('M'), schedule a writeback (100 cycles, `handle_write_back`).
       * Issue `Bus::read` (BusRd) to snoop other caches.
       * If found in another cache:
         · 'M': Transfer data (2N cycles, N = words per block), write back to memory (100 cycles), set local state to 'S'.
         · 'E' or 'S': Transfer data (2N cycles), set local state to 'S'.
       * If not found, fetch from memory (100 cycles), set state to 'E'.
       * Update `data_traffic` (block size bytes) and `stalls`.
   - **Write Access** (`access_write`):
     - On hit:
       * 'M': Return `true`.
       * 'E': Set state to 'M', return `true`.
       * 'S': If bus is free, issue `Bus::invalidate`, set state to 'M', return `true`.
     - On miss:
       * Increment `misses` if `has_new_instruction`.
       * If bus is busy, return `false`.
       * Select LRU line. If 'M', schedule writeback (100 cycles).
       * Issue `Bus::read` (BusRdX). If found in 'M', write back to memory (100 cycles). Invalidate other caches (`Bus::invalidate`).
       * Fetch data from memory (100 cycles), set state to 'E' or 'M' based on context.
       * Update `data_traffic` and `stalls`.
   - **Snooping** (`snoop`):
     - Respond to `Bus::read` or `Bus::invalidate`.
     - For read: 'M' $\rightarrow$ 'S', 'E' $\rightarrow$ 'S', 'S' remains 'S', return state and add 2N cycles for transfer.
     - For write: Invalidate line ('M', 'E', 'S' $\rightarrow$ 'I'), return original state.
   - Helper Functions: `find_line`, `find_line_to_replace`, `get_set_index`, `get_tag`, `get_address_from_`

6. **Bus Operations** (`bus.cpp`):

   - `read`: Snoop all processors except the requester. Return the first non-'I' state, setting `free_time` (2N for cache-to-cache, 100 for memory).

- `invalidate`: Invalidate lines in all other processors, incrementing `invalidations`.
- `is_busy`: Check if `free_time > 0`.

7. **Output** (`simulate.cpp`):
   - `print_results`: Output per-core statistics (instructions, reads, writes, cycles, idle cycles, miss rate, evictions, writebacks, invalidations, data traffic), bus summary (transactions, traffic), and maximum execution time across cores.

1.4 Key Assumptions

- **Address Parsing**: 32-bit addresses, zero-padded if shorter (e.g., `0x817b08` → `0x00817b08`).
- **Cache Blocking**: Cores halt on misses, but snooping continues.
- **Bus Arbitration**: Sequential processing resolves ties arbitrarily.
- **Timing**: Cache hit = 1 cycle, memory access = 100 cycles, cache-to-cache transfer = 2N cycles (N = words per block), writeback = 100 cycles.
- **Initial State**: All cache lines start in 'I' (invalid).
- **Data Traffic**: Counts block transfers (memory or cache-to-cache) in bytes.
- **Snooping Overhead**: Snooping incurs no cycles unless data is transferred.

1.5 MESI Protocol Implementation

The MESI protocol is implemented to maintain cache coherence across the four processor cores:

- **Modified (M)**: Line is present only in the current cache and is dirty (has been written to).
- **Exclusive (E)**: Line is present only in the current cache and is clean (matches memory).
- **Shared (S)**: Line may be present in other caches and is clean.
- **Invalid (I)**: Line is invalid (not present or invalidated).

State transitions occur based on processor requests (PrRd, PrWr) and bus-side requests (BusRd, BusRdX):

1.6 Cache Operation Details

- **Read Hit**: No state change, 1 cycle. States 'M', 'E', 'S' are valid.
- **Read Miss**:
  - Find LRU line. If 'M', write back to memory (100 cycles).
  - If bus is busy, stall core.
  - Snoop other caches:
    - 'M': Copy data (2N cycles), write back (100 cycles), set both to 'S'.
    - 'E': Copy data (2N cycles), set both to 'S'.
    - 'S': Copy data (2N cycles), set local to 'S'.
  - in cache to cache copy both caches get stalled

  - If not found, fetch from memory (100 cycles), set to 'E'.

- **Write Hit**:
  - 'M': No State change, 1 cycle.
  - 'E': Set to 'M', 1 cycle.
  - 'S': If bus free, invalidate other caches, set to 'M', 1 cycle.

- **Write Miss**:
  - Find LRU line. If 'M', write back (100 cycles).
  - If bus busy, stall core.
  - Snoop other caches:
    * 'M': Write back (100 cycles), invalidate, fetch to local cache (100 cycles), set to 'M'.
    * 'E'/'S': Invalidate others, fetch from memory (100 cycles), set to 'E' or 'M'.
  - If not found, fetch from memory (100 cycles), set to 'M'.

2 Simulation

In our assignment we use round robin method to decide the order in which each processor gets evaluated within a cycle and this is deterministic so everytime we run the simulator with given parameters, we get the exact same output.

2.1 app1 output

- **Cache Configuration:**
  - Set Index Bits: 6
  - Associativity: 2
  - Block Bits: 5
  - Block Size (Bytes): 32
  - Number of Sets: 64
  - Cache Size (KB per core): 4
  - MESI Protocol: Enabled
  - Write Policy: Write-back, Write-allocate
  - Replacement Policy: LRU
  - Bus: Central snooping bus

- **Core 0 Statistics:**
  - Total Instructions: 2,497,349
  - Total Reads: 1,489,888
  - Total Writes: 1,007,461
  - Total Execution Cycles: 2,533,084
  - Idle Cycles: 11,099,739
  - Cache Misses: 35,713
  - Cache Miss Rate: 1.43%

  - Cache Evictions: 35,527
  - Writebacks: 7,157
  - Bus Invalidations: 10
  - Data Traffic (Bytes): 33,649,536

- **Core 1 Statistics:**
  - Total Instructions: 2,490,468
  - Total Reads: 1,485,857
  - Total Writes: 1,004,611
  - Total Execution Cycles: 2,525,680
  - Idle Cycles: 10,662,640
  - Cache Misses: 35,197
  - Cache Miss Rate: 1.41%
  - Cache Evictions: 35,013
  - Writebacks: 6,772
  - Bus Invalidations: 21
  - Data Traffic (Bytes): 33,521,568

- **Core 2 Statistics:**
  - Total Instructions: 2,509,057
  - Total Reads: 1,492,629
  - Total Writes: 1,016,428
  - Total Execution Cycles: 2,549,751
  - Idle Cycles: 11,813,577
  - Cache Misses: 40,687
  - Cache Miss Rate: 1.62%
  - Cache Evictions: 40,472
  - Writebacks: 10,446
  - Bus Invalidations: 46
  - Data Traffic (Bytes): 34,260,352

- **Core 3 Statistics:**
  - Total Instructions: 2,503,127
  - Total Reads: 1,493,736
  - Total Writes: 1,009,391
  - Total Execution Cycles: 2,539,027
  - Idle Cycles: 11,033,396
  - Cache Misses: 35,877
  - Cache Miss Rate: 1.43%
  - Cache Evictions: 35,708
  - Writebacks: 7,000
  - Bus Invalidations: 10

- Data Traffic (Bytes): 33,704,736

- **Overall Bus Summary:**
  - Total Bus Transactions: 145,861
  - Total Bus Traffic (Bytes): 4,927,040
  - Maximum Execution Time: 2,549,751 cycles

2.2 app2 output

- **Simulation Parameters:**
  - Trace Prefix: `./assignment3`$_t races/app2 SetIndexBits : 6$
  - Associativity: 2
  - Block Bits: 5
  - Block Size (Bytes): 32
  - Number of Sets: 64
  - Cache Size (KB per core): 4
  - MESI Protocol: Enabled
  - Write Policy: Write-back, Write-allocate
  - Replacement Policy: LRU
  - Bus: Central snooping bus

- **Core 0 Statistics:**
  - Total Instructions: 3,270,132
  - Total Reads: 2,380,720
  - Total Writes: 889,412
  - Total Execution Cycles: 3,454,830
  - Idle Cycles: 45,890,505
  - Cache Misses: 184,339
  - Cache Miss Rate: 5.64%
  - Cache Evictions: 183,610
  - Writebacks: 26,249
  - Bus Invalidations: 426
  - Data Traffic (Bytes): 35,376,704

- **Core 1 Statistics:**
  - Total Instructions: 3,287,252
  - Total Reads: 2,388,005
  - Total Writes: 899,247
  - Total Execution Cycles: 3,471,272
  - Idle Cycles: 51,520,766
  - Cache Misses: 183,964
  - Cache Miss Rate: 5.60%
  - Cache Evictions: 183,745

  - Writebacks: 27,582
  - Bus Invalidations: 74
  - Data Traffic (Bytes): 35,770,784

- **Core 2 Statistics:**
  - Total Instructions: 117,698
  - Total Reads: 74,523
  - Total Writes: 43,175
  - Total Execution Cycles: 125,959
  - Idle Cycles: 2,940,263
  - Cache Misses: 8,255
  - Cache Miss Rate: 7.01%
  - Cache Evictions: 8,114
  - Writebacks: 2,803
  - Bus Invalidations: 13
  - Data Traffic (Bytes): 1,776,000

- **Core 3 Statistics:**
  - Total Instructions: 3,324,919
  - Total Reads: 2,416,052
  - Total Writes: 908,867
  - Total Execution Cycles: 3,510,350
  - Idle Cycles: 45,775,994
  - Cache Misses: 185,021
  - Cache Miss Rate: 5.56%
  - Cache Evictions: 184,466
  - Writebacks: 26,084
  - Bus Invalidations: 319
  - Data Traffic (Bytes): 35,990,240

- **Overall Bus Summary:**
  - Total Bus Transactions: 557,435
  - Total Bus Traffic (Bytes): 18,601,824
  - Maximum Execution Time: 3,510,350 cycles

2.3 graphs

next we vary the paramters and check the total number of cycles the simulator runs

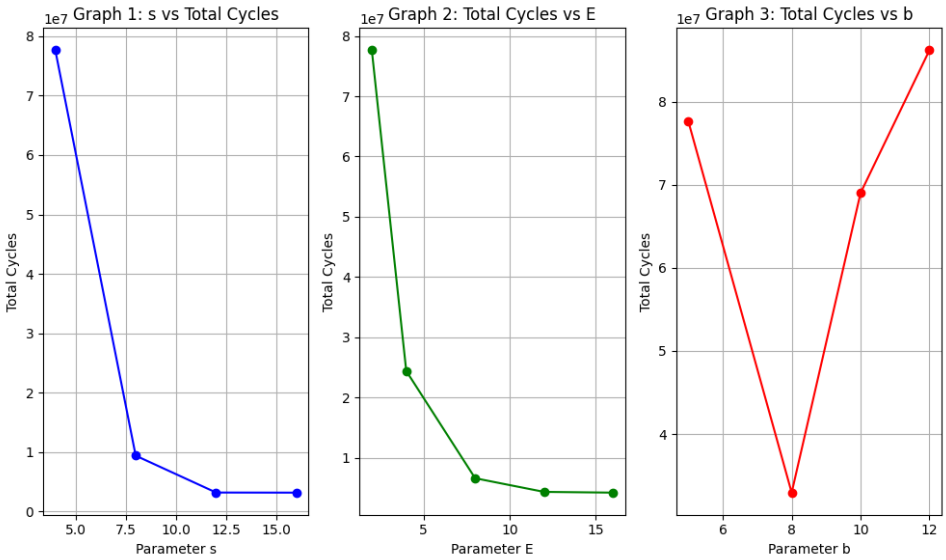

Figure 1: Graph showing total cycles vs. parameter