Cache Simulator: Implementation and Testing Report

November 12, 2024

1 Introduction

This report documents the development and testing of a cache simulator intended to explore the behavior of cache memory under various configurations. The simulator is designed to work with RISC-V ISA and allows for testing different cache parameters, including cache size, block size, associativity, replacement policies, and write policies. Verification was performed using Risc-V tools to generate and analyze memory access traces.

2 Data Structures and Cache Models

2.1 Direct-Mapped Cache

The direct-mapped cache structure is implemented as:

```
map<string, tuple<string, string, string, vector<string>>> direct_mapped;
```

where each 'string' key is an index in the cache. The tuple for each entry contains:

- **Valid Bit:** Shows if data in this entry is valid.
- **Dirty Bit:** Indicates if the data has been modified.
- **Data in Memory:** Contains the data being cached.

2.2 Fully Associative Cache

For the fully associative cache:

```
vector<tuple<string, string, string, vector<string>, int>> fully_associative;
```

Each vector element is a tuple consisting of:

- **Valid Bit:** Validity status of the cache entry.
- **Dirty Bit:** Modification status of the data.
- **Data in Cache:** Cached data.
- **Tag:** Uniquely identifies the block in the cache.
- **Timestamp:** Last access time, used for LRU replacement policy.

2.3 Set-Associative Cache

The set-associative cache data structure is:

```
map<string, vector<tuple<string, string, string, vector<string>, int>>> set_associative;
```

where each key is an index for a set, and each set contains tuples:

- **Valid Bit, Dirty Bit, Data in Cache, Tag, Timestamp:** Function as in fully associative cache but limited to each set.

This model implements k-way associativity with per-set replacement.

3 Function Definitions and Key Algorithms

3.1 Direct-Mapped Cache Initialization

The direct-mapped cache initialization function:

```
tuple<bool, vector<string>, tuple<string, string, string, vector<string>>> direct_mapped_cache_initia
```

accepts parameters:

- **address:** Memory address to be accessed.
- **block_size, cache_size, allocate:** Cache configuration settings.

Returns a tuple:

- **Hit/Miss:** Boolean for cache hit/miss.
- **Data:** Cached data.
- **Evicted Block:** Details if an eviction occurs.

3.2 Fully Associative Cache Initialization

The fully associative cache initialization function:

```
tuple<bool, vector<string>, ll, tuple<string, string, string, vector<string>, int>> fully_associative
```

has parameters:

- **address, block_size, cache_size, allocate:** Cache settings and memory address.

Returns:

- **Hit/Miss, Data, Evicted Block Timestamp, Evicted Block Details**

3.3 Set-Associative Cache Initialization

The set-associative cache function:

```
tuple<bool, vector<string>, ll, tuple<string, string, string, vector<string>, int>> set_associative_i
```

accepts:

- **address, block_size, cache_size, associative, allocate:** Cache settings for k-way associativity.

Returns:

- **Hit/Miss, Data, Evicted Block Timestamp, Evicted Block Details**

4 Implementation Design

4.1 Cache Simulator Configuration

The simulator's configuration parameters:

- **Cache Size, Block Size, Associativity, Replacement Policy, Write Policy**

define the behavior of the simulator, enabling it to handle different cache setups.

4.2 Memory Address Decoding

The address is split into *tag*, *index*, and *offset* based on the configuration, allowing the simulator to identify cache lines effectively.

4.3 Replacement and Write Policies

Implemented policies:

- **Replacement:** FIFO, LRU, Random
- **Write:** Write-Back, Write-Through

FIFO and LRU use counters, while Write-Back marks dirty entries for delayed memory writes.

5 Testing Strategy

5.1 Test Cases

Test cases were generated to validate each cache configuration:

- **Direct Mapped:** Sequential and random access patterns were tested to observe eviction behavior.
- **Set-Associative:** Configurations with 2-way and 4-way associativity were tested using both FIFO and LRU replacement policies to verify correct set allocation and eviction.
- **Fully Associative:** The cache was tested using random and LRU replacement policies to ensure correct handling of associative mapping.

5.2 Verification Ripes

- **Ripes Simulator:** Additionally, generated test cases were cross-verified using the Ripes RISC-V standard simulator. This provided a secondary reference for standard RISC-V cache behavior and validation of our simulation results.

6 Results and Observations

Key insights included:

- **Higher Associativity:** Improved hit rates for sequential patterns.
- **Write-Back Efficiency:** Write-Back with allocation yielded better hit rates.
- **FIFO vs. LRU:** LRU outperformed FIFO in reducing misses for repeated accesses.

7 Conclusion

The cache simulator achieved accurate modeling of cache behaviors, validated through Spike and DineroIV. Future work includes multi-level cache hierarchy and performance optimization for large cache sizes.