VCU Electrical and Computer Engineering
College of Engineering

LAB REPORT

EGRE 426 – Computer Organization & Design

Lab 5: Cache Simulator

Due Date: December 5, 2023

Author: Nathan Reyes

On my honor, I have neither given nor received unauthorized aid on this assignment, and I pledge that I am in compliance with the VCU Honor System.

VCU Electrical and Computer Engineering
College of Engineering

1 Introduction

The advancement of computer technology has seen significant improvements in processing speed, which often outpaces the speed of memory access. This disparity has necessitated the use of cache memory—a smaller, faster type of volatile computer memory that provides high-speed data access to a processor and stores frequently used computer programs, applications, and data.

Cache memory plays a critical role in enhancing the data retrieval processes by reducing the latency to access data from the main memory. A cache simulator, such as the one developed for this lab, is a tool that models the behavior of a cache to study its effectiveness in various configurations and access patterns. This report outlines the development of a cache simulator designed to emulate the set-associative caching mechanism with Least Recently Used (LRU) and random replacement policies, as discussed in class.

2 Background

Caching is a key concept in computer architecture, crucial for bridging the speed gap between the central processing unit (CPU) and the main memory. A cache's efficiency is governed by its architecture, which includes the block size, number of blocks, and its associativity level. In a direct-mapped cache architecture, each block of main memory maps to exactly one cache line. Conversely, a set-associative cache allows each memory block to map to any one of several lines in a cache set, based on a replacement policy when the set is full.

The effectiveness of a cache can be quantified through its hit and miss rates—the proportion of memory accesses that are found in the cache versus those that are not. The hit rate is crucial for performance, as it reflects the percentage of accesses that were quickly served by the cache without needing to reach slower main memory. The simulation of such metrics and their dependencies on cache configurations is the primary objective of our cache simulator.

This background sets the stage for understanding the design decisions and implementation details that will be presented in the subsequent sections of this report.

3 Design

The design of the cache simulator is based on a modular approach, encapsulating distinct functionalities within specific classes and functions. The simulator's architecture mirrors that of a real-world cache, including components such as cache lines, sets, and a configuration module that sets the parameters for simulation.

At the core of the design is the principle of set-associativity, which is implemented to allow flexibility in the cache's behavior, ranging from direct-mapped (single-line sets) to fully associative (single set with all lines). This design enables the simulator to model a wide range of caching strategies simply by adjusting the configuration parameters.

Furthermore, the simulator incorporates two replacement policies: Least Recently Used (LRU) and Random. The LRU policy is implemented using a counter to track the usage of cache lines, while the Random policy selects a cache line to replace at random, providing a comparison between deterministic and non-deterministic cache behaviors.

4 Implementation

The implementation of the cache simulator is done in Python, providing an accessible and high-level framework to model the caching logic. The project is divided into three main components: the `Cache` class, the `CacheConfig` class, and the `main` module.

The `Cache` class is the heart of the simulator, where the set-associative logic is executed. It contains methods for reading data from the cache (`read_address`), determining whether a cache hit or miss occurs, and handling cache misses with the appropriate replacement policy. The cache lines within each set are represented by instances of a nested `CacheLine` class, which stores the tag, data, and last used counter for the LRU policy.

The `CacheConfig` class serves as a container for the cache's configuration parameters, such as block size, number of blocks, associativity, and replacement policy. These parameters are passed as command-line arguments and parsed within the `main` module, demonstrating how the simulator can be dynamically configured for different simulation scenarios.

The `main` module orchestrates the simulation process. It parses command-line arguments using the `argparse` library to set up the cache configuration and initiates the cache simulation by reading a sequence of byte addresses from an input file. The `main` module also calculates and displays the hit/miss rates, providing immediate feedback on the cache's performance.

By separating concerns across these components, the cache simulator not only achieves its functional requirements but also adheres to software engineering best practices, promoting readability, maintainability, and extensibility of the codebase.

5 Testing

Comprehensive testing is imperative to validate the reliability and accuracy of the cache simulator. The test suite includes a series of unit tests that cover various functionalities of the caching mechanism, including the correct calculation of set indices and tags, adherence to the Least Recently Used (LRU) policy, and the handling of cache hits and misses.

5.1 Unit Tests

A suite of unit tests, implemented using Python's `unittest` framework, is provided to systematically test the individual components of the cache simulator. Here is a summary of the key test cases:

- **Set Index and Tag Calculation:** Tests whether the `get_set_index_and_tag` method correctly computes the set index and tag based on given memory addresses.

- **LRU Policy:** Ensures that the cache evicts the least recently used cache line when a miss occurs and a replacement is needed.
- **Cache Hit and Miss:** Confirms that the cache correctly identifies hits and misses.
- **Read Operations:** Simulates read operations to check for correct cache behavior on initial and subsequent accesses.
- **Cache Configuration:** Verifies that the cache behaves as expected with different configurations, such as varying block sizes and associativities.

These tests are pivotal in ensuring that the cache simulator functions correctly for a variety of scenarios and configurations.

5.2 Edge Case Testing

The robustness of the cache simulator is further validated by edge case testing, which includes:

- Testing with the minimum and maximum number of blocks the simulator is expected to handle.
- Verifying behavior with edge memory addresses, such as 0x0 and the highest possible address in the system.
- Ensuring the simulator gracefully handles invalid inputs and provides clear error messages.

5.3 Test Execution and Coverage

Tests are executed to cover all the functional aspects of the simulator, as well as boundary and edge cases. Each test case includes a descriptive comment explaining its purpose and the expected outcome, which aids in maintaining clarity and intent within the test suite.

6 Results

The execution of the test suite yielded the following results, which confirmed the cache simulator's operational correctness and efficiency.

6.1 Functional Test Results

The functional tests performed as expected, with each test case passing successfully. The cache simulator was able to correctly calculate set indices and tags, identify cache hits and misses, and adhere to the LRU policy.

6.2 Edge Case Test Results

The edge case tests demonstrated the simulator's ability to handle unusual or extreme scenarios without failure. The simulator correctly handled minimum and maximum cache configurations, as well as invalid inputs.

6.3 Performance Metrics

Performance metrics were gathered during the testing phase, highlighting the simulator's efficiency. Even with a large number of reads, the simulator maintained a consistent performance level.

6.4 Output Validation

The output of the cache simulator was validated against expected results for a series of memory addresses as specified in the lab handout. The following table illustrates the results for two specific test cases, as detailed in the README file:

Test Case	Cache Size	Reads	Hits	Hit Rate	Misses	Miss Rate
Block size 16, 256 blocks	4.0k	10000	61	0.61%	9939	99.39%
Block size 256, 256 blocks	64.0k	10000	978	9.78%	9022	90.22%

Table 1: Simulator output for specific test cases.

These outputs confirmed the simulator's performance under different cache configurations, with hit and miss rates corresponding to the expected behavior of the caching algorithm when subjected to the input address sequences.

6.5 Discrepancies and Anomalies

No discrepancies or anomalies were observed in the test results. The simulator's performance matched the expected theoretical outcomes, reinforcing its validity as a tool for educational purposes.

7 Challenges and Learnings

Throughout the development of the cache simulator, several challenges were encountered, providing valuable learning opportunities.

7.1 Challenges

One of the primary challenges was ensuring the accuracy of the cache hit and miss calculations. This required a deep understanding of how different cache configurations impact the cache's behavior. Additionally, implementing the LRU replacement policy presented complexities in efficiently tracking and updating the usage of cache lines.

Debugging edge cases, such as handling zero-sized caches or maximal block sizes, demanded rigorous testing and validation. Ensuring robust error handling and user feedback mechanisms for invalid inputs also posed significant challenges.

7.2 Learnings

Overcoming these challenges led to key learnings, particularly in the areas of software design and testing. The modular design approach proved invaluable in isolating functionality and simplifying the debugging process. Developing a comprehensive suite of unit tests was instrumental in validating the correctness of the simulator and in building confidence in its reliability.

Furthermore, the project underscored the importance of performance considerations in software development. Optimizing the simulator for efficiency was crucial in managing large input sets and maintaining consistent performance.

Overall, the implementation and testing processes reinforced the principles of computer architecture, particularly the role and operation of caches, and the importance of careful design and thorough testing in software development.

8 Conclusion

The cache simulator project successfully achieved its objectives, providing a functional and educational tool that models the behavior of a set-associative cache with various replacement policies. The simulator demonstrated the expected performance characteristics, with hit and miss rates aligning with theoretical predictions.

The simulator's design and implementation facilitated a clear understanding of caching mechanisms and the impact of different configurations on cache performance. The testing phase confirmed the simulator's operational correctness and efficiency, with no significant discrepancies or anomalies observed.

Reflecting on the project, it is evident that the simulator serves as a robust platform for exploring cache memory behavior. It offers a tangible learning experience for students and practitioners interested in computer architecture and memory hierarchies.

8.1 Future Work

Looking forward, there are several opportunities for extending the simulator. These include adding support for more complex replacement policies, simulating multi-level caches, and incorporating a graphical user interface for visualization. Enhancing the simulator with these features would provide an even richer toolset for education and research in the field of computer architecture.

In conclusion, the cache simulator stands as a testament to the effective application of computer science principles and best practices in software development. It offers a solid foundation for further exploration and expansion in the study of caching strategies and their implications on computing performance.