Cache Simulator

Uttam Paharia
Roll Number:CS22BTECH11060

December 31, 2023

Introduction

This report provides an overview of the coding approach and testing strategies employed in the development of the cache simulator program. The cache simulator aims to simulate the behavior of a cache memory system based on the provided configuration file and access traces.

Coding Approach

The coding approach involved several key steps to ensure the correctness and functionality of the cache simulator:

1. **Configuration File Reading:** Implemented a function (`readConfig()`) to read the cache configuration parameters from the file `cache.config`. This included reading cache size, block size, associativity, replacement policy, and write-back policy.
2. **Address Access Simulation:** Developed a function (`accessAddress()`) to simulate the access of memory addresses based on the provided access traces in the file `cache.access`. The function utilizes a cache simulation algorithm with different replacement policies (LRU, FIFO, RANDOM) and write-back policies (WB, WT).
3. **Logging Hits and Misses:** Implemented the logic to identify cache hits and misses during the simulation. The program outputs whether an access results in a hit or miss and displays the corresponding tag.
4. **Dynamic Memory Allocation:** Utilized the C++ `vector` container to dynamically allocate memory for the cache and related data structures. This ensures flexibility in handling different cache configurations.

Testing Strategies

Testing played a crucial role in ensuring the correctness and reliability of the cache simulator. The following testing strategies were employed:

1. **Unit Testing:** Individual functions such as file reading, address simulation, and logarithmic calculations were tested independently to ensure they perform as expected.
2. **Boundary Testing:** The simulator was tested with extreme values of cache size, block size, and associativity to verify the program's robustness.
3. **Policy-specific Testing:** Specific tests were conducted to validate the correctness of each replacement and write-back policy.
4. **Randomized Testing:** Utilized randomized access traces to simulate various scenarios and assess the simulator's adaptability to different workload patterns.

Input and Output

Give input in specified format only all address should be in Hexadecimal with 0x as prefix also for mode give in specified format only like R: 0x.... managing spaces also as this part is hard coded. Give input in specified files only.
Output is given separately in output.txt file.

Compiling and Running Program

To complie the given code give command:
g++ cacheSimulator.cpp
From this we will get an executable file to run that give command
./a.out