Minor Project End Sem Presentation

Title:

Cache Simulator

Presented by:

Atin Anant, 500093013, CCVT,B3
Piklu De, 500096448,CCVT,B6
Shubham Jaiswal, 500096554,B6
Ashmit Panwar, 500093054, B3

Guided by:
Ms. Shahina Anwarul
Assistant Professor (SS)
Systemics Cluster
School of Computer Science

Content

1. Introduction
2. Technical Background of Project
3. Problem Statement
4. Tech Stack
5. Objectives
6. Methodology
7. Flow Chart
8. Justification of Objectives

1. Introduction

WHAT IS CACHE MEMORY?

Cache memory is a small but high-speed type of computer memory that acts as a bridge between the CPU and the slower main memory (RAM). Its purpose is to store frequently accessed data and instructions, enabling faster retrieval for the CPU. By keeping a copy of frequently used data close to the processor, cache memory reduces the time the CPU spends waiting for data from the main memory, thereby enhancing overall system performance. Modern computers have multiple cache levels (L1, L2, L3) to efficiently manage data access at different speeds and capacities.

USE CASE OF CACHE MEMORY

Cache memory is a small but high-speed type of computer memory that acts as a bridge between the CPU and the slower main memory (RAM). Its purpose is to store frequently accessed data and instructions, enabling faster retrieval for the CPU. By keeping a copy of frequently used data close to the processor, cache memory reduces the time the CPU spends waiting for data from the main memory, thereby enhancing overall system performance. Modern computers have multiple cache levels (L1, L2, L3) to efficiently manage data access at different speeds and capacities.

Introduction (contd.)

How Cache Memory Works:

Cache memory operates as a vital component within a computer system, facilitating rapid data access for the Cache memory operates as a vital component within a computer system, facilitating rapid data access for the CPU. Here's a step-by-step explanation of how cache memory works:

1. **Data Hierarchy:** Computers have multiple levels of memory, including registers, cache, main memory (RAM), and storage devices (e.g., hard drives). Cache memory is positioned between the CPU and RAM.

2. **Data Retrieval:** When the CPU requires data or instructions, it first checks the cache memory (L1, L2, or L3 cache levels) for a copy of the needed information.

3. **Cache Hit:** If the required data is present in the cache, it's called a cache hit. The CPU can access the data almost instantly, significantly reducing latency.

4. **Cache Miss:** If the data isn't in the cache (cache miss), the CPU must retrieve it from the slower main memory (RAM).

5. **Cache Coherency:** In multi-core or multi-processor systems, cache coherence protocols ensure that all caches have the same view of memory to prevent data inconsistencies.

6. **Data Replacement:** Cache management algorithms, such as LRU (Least Recently Used) or FIFO (First-In-First-Out), determine which data gets evicted from the cache when new data needs space.

7. **Performance Boost:** Cache memory purpose is to store frequently accessed data, which results in improved system performance by reducing the CPU's wait time for data from RAM or storage devices.

8. **Size vs. Speed Trade-off:** Cache memory sizes and speeds are carefully balanced to optimize performance while controlling cost. Smaller and faster caches store critical data, while larger but slower caches accommodate more data.

9. **Dynamic Nature:** Cache contents continuously change as the CPU fetches and stores data, ensuring that the most frequently used data remains readily available.

2. Technical Background Of Project

Configurable Cache Parameters:

Enable users to customize cache size, associativity, and block size for versatile architecture simulations.

Trace File Handling:

Implement a reliable mechanism for interpreting memory access trace files, facilitating real-world workload simulations.

Performance Metrics Calculation:

Compute hit rate, miss rate,= for comprehensive performance evaluation.

Error Handling Mechanisms:

Incorporate robust error handling to ensure a stable and reliable user experience.

Statistical Information:

Provide detailed statistics on cache hits, misses, and evictions for in-depth analysis.

3. Problem Statement

In the realm of computer architecture and system optimization, understanding cache behavior is crucial for achieving efficient and high-performance computing systems. Cache simulators are essential tools for evaluating and optimizing cache memory designs. However, there is a need for a new cache simulator project to address specific challenges and requirements.

4. Technology Stack

Version Control

- Github

Development

- **Backend -** Java
- **Frontend -** Basic Frontend Tools, Java fx
- **Java Version -** Java 18
- **IDE –** Intellij idea

5.Objectives

Design and implement a cache simulator that can simulate the behaviour of a cache memory system.

Sub-objectives

- The simulator should use algorithms like LRU, FIFO, or random for cache line replacement
- The simulator should be able to take as input the cache size, block size, associativity, and replacement policy.
- It should also be able to read a trace of memory accesses and simulate the behaviour of the cache for each access.
- The simulator should output statistics such as the number of cache hits, misses, and evictions. Additionally, the simulator should be able to visualize the state of the cache at any point in time.
- Assess the impact of cache optimizations on overall system performance.

6. Methodology

Parse Command Line Arguments:

•The program starts by reading command line arguments, which include cache size, associativity, replacement policy, write-back policy, and an input file containing memory access traces.

Create Cache Object:

- An instance of the Cache class is created, initializing cache data structures and metadata based on the provided cache size and associativity.
- Cache Parameters: Cache Size: Determines the number of cache lines.
- Block Size: Specifies the size of cache blocks.
- Associativity: Sets the level of associativity (direct-mapped, set-associative, fully associative).
- Replacement Policy: Algorithms like LRU, FIFO, or random for cache line replacement.

Data Structures:

- Arrays: Used for cache lines, tag arrays, data arrays, and other metadata storage.
- Linked Lists: For implementing cache replacement policies like LRU (Least Recently Used) or FIFO (First-In-First-Out).
- Hash Tables: Useful for quick lookups in associative caches.

Methodology (contd.)

Memory Access Traces:

Memory traces obtained from real applications or benchmarks are essential for simulating cache behavior.

Simulation Algorithms:

- Direct-Mapped Cache Simulation: Uses simple indexing to find cache lines.
- Set-Associative Cache Simulation: Utilizes indexing and tag matching for associativity.
- Fully Associative Cache Simulation: Requires a more complex search to find cache lines.

Replacement Policies:

LRU (Least Recently Used): Evicts the least recently used cache line.

FIFO (First-In-First-Out): Evicts the oldest cache line.

Random Replacement:

Randomly selects a cache line for eviction.

Prefetching Algorithms:

Implement prefetching strategies to improve cache performance.

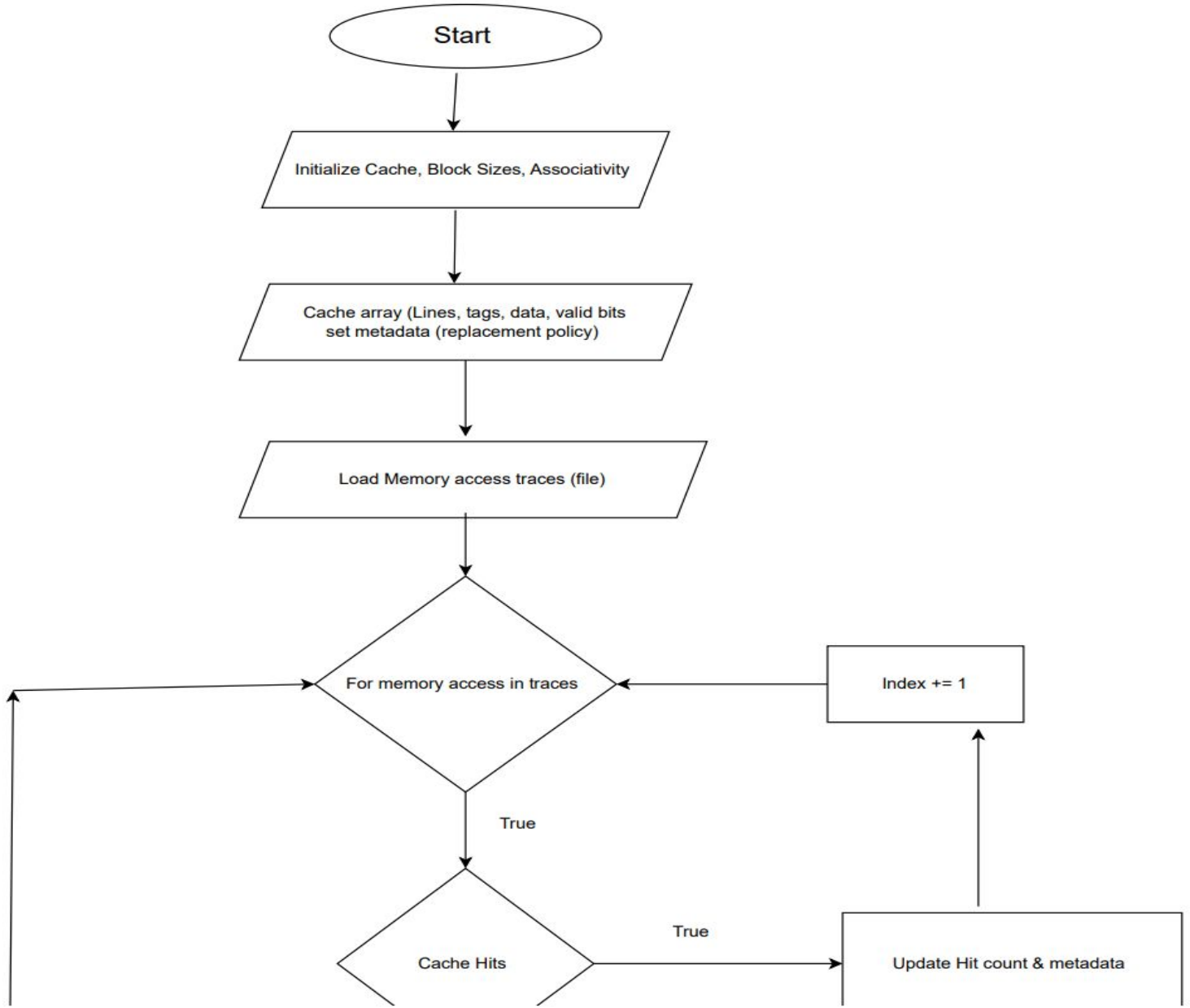
Write Policies:

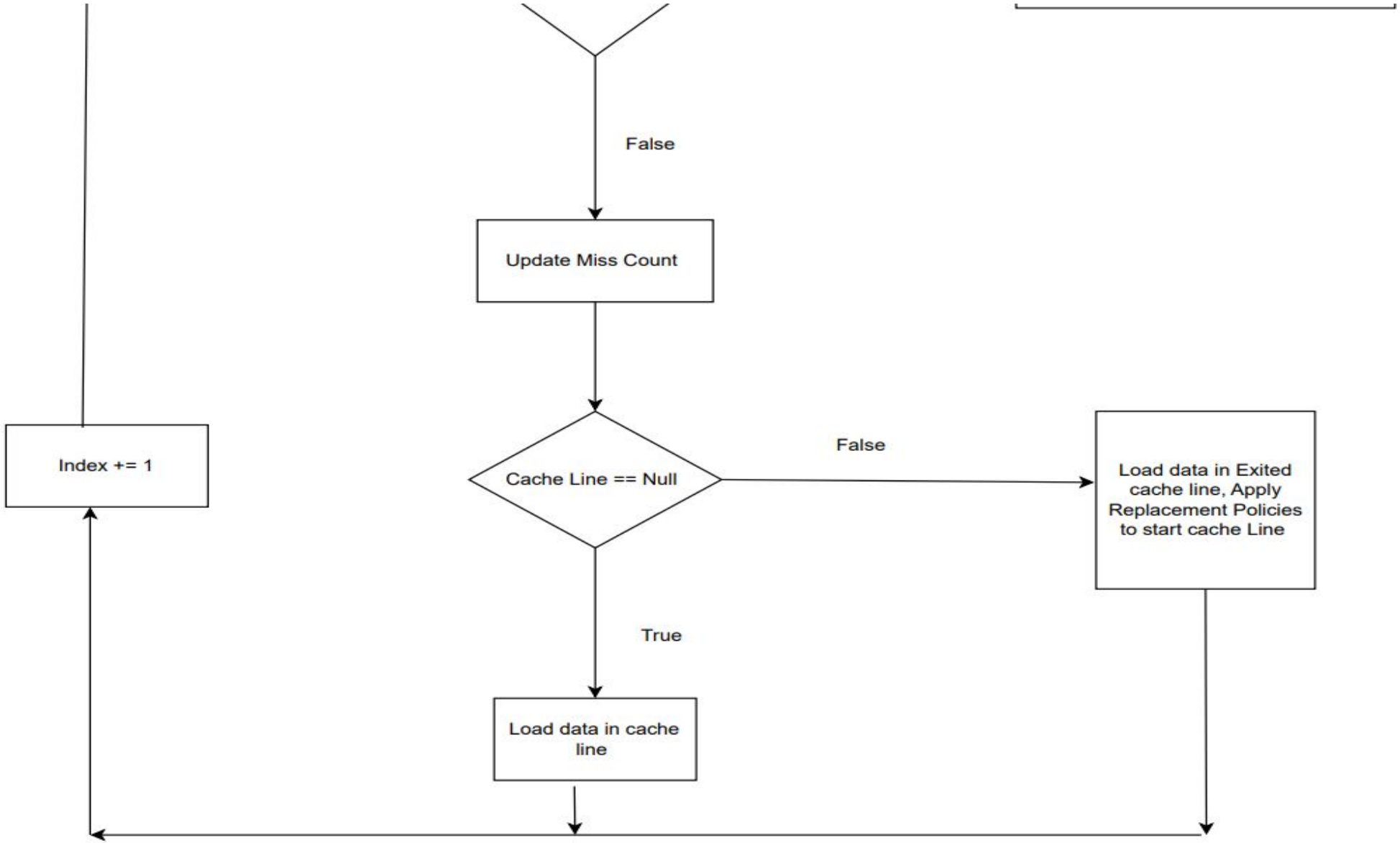
Choose between write-through and write-back policies.

Performance Metrics:

- Hit Rate: The percentage of cache accesses that result in hits.
- Miss Rate: The percentage of cache accesses that result in misses.

10. Flow Chart




False
Update Miss Count
Index += 1
Cache Line == Null
False
Load data in Exited cache line, Apply Replacement Policies to start cache Line
True
Load data in cache line

11. Justification Of Objectives

1. Objective: Create an Efficient Cache Simulator

Justification:

- A cache simulator is essential for evaluating the performance of cache memory systems. By creating an efficient cache simulator, we enable users to model and understand how different cache configurations impact overall system performance.
- This objective addresses the need for a tool that assists in the design and analysis of cache architectures, contributing to the improvement of memory management in computer systems.

2. Objective: Support Both Command-Line Interface (CLI) and Graphical User Interface (GUI)

Justification:

- Providing both CLI and GUI interfaces caters to users with varying preferences and levels of expertise. Advanced users who are comfortable with command-line interactions can use the CLI, while a GUI appeals to users who prefer a more user-friendly experience or may not be familiar with command-line operations.
- This objective ensures broader usability and accessibility, making the cache simulator accessible to a wider audience.

3. Objective: Implement Input Validation for Security and Robustness

Justification:

- Input validation is crucial for preventing potential security vulnerabilities, such as injection attacks or unintended file access. By implementing robust input validation, we enhance the software's reliability and mitigate the risk of security breaches.
- This objective addresses the need for a secure and resilient cache simulator, ensuring that users cannot compromise the application through malicious input.

Thank You