POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

cache simulator

Laurea Magistrale in

Author: Mahmoud Ahmed Eltaras - omar mohamed eldafrawy

Advisor: Prof. Cristina Silvano

Co-advisor: Tommaso Spagnolo

Academic year: 2024-2025

1. Introduction

Cache memory is a fundamental component in modern computer architectures, designed To increase the performance of a computer and to reduce the average time to access data from main memory. It acts as a high-speed buffer between the CPU and main memory. The effectiveness of a cache is determined by its ability to successfully predict which data the CPU will need, a principle known as locality.

- Temporal Locality: Recently accessed data is likely to be needed again soon.
- Spatial Locality: Data located near recently accessed data is likely to be needed soon.

2. Project Objective

The primary objective of this project was to significantly enhance the performance of a baseline pipeline simulator
Our work involved a systematic modification of this base simulator to implement a modern cache architecture by bridging the critical performance gap between the CPU and main memory.

2.1. Implementation of Cache Architecture

Our first major contribution was the introduction of the cache system. We implemented separate instruction and data caches to handle the distinct access patterns of code and data. The implementation required careful handling of the processor pipeline to manage cache hits and misses, ensuring that the CPU could proceed on a hit and stall all the pipeline correctly on a miss untill the required data is fetched.

2.2. Advanced Features

we explored the cache design space by implementing and analyzing several advanced features to further optimize performance.

2.2.1 Expansion Of Replacement Policies

The initial implementation used a standard least recently used (LRU) policy. To perform a comparative analysis, we implemented two additional policies. First-In, First-Out (FIFO), and Random. Show that an intelligent replacement strategy is critical to maximize the hit rate.
When a cache set is full, a replacement policy decides which block to evict.

- **Least Recently Used (LRU):** Evicts the block that has not been used for the longest time. It generally performs best due to its effective use of temporal locality.
- **First-In, First-Out (FIFO):** Evicts the

oldest block. It is simpler than LRU but can perform poorly.

- **Random:** Evicts a random block. Performance is unpredictable.

2.2.2 Implementation OF Victim Cache

A key innovation in our project was the integration of a victim cache. This is a small, fully associative buffer designed to sit between the cache and main memory. Its purpose is to hold recently evicted blocks from the cache, providing a "second chance" to recover data without the high cost of full-memory access.

3. Experimental Setup

To thoroughly analyze the cache behavior, an ideal experimental setup would involve systematically varying key cache parameters and observing their impact on performance metrics. This typically includes manipulating cache size, block size, associativity, and replacement policies

3.1. Cache Parameters

• **Cache Size:** The total data capacity of the cache. Larger caches generally lead to higher Hit rates but also cause higher costs and potentially longer access times.
• **Block Size (Cache Line Size):** The amount of data transferred between main memory and cache in a single operation. Larger block sizes can exploit spatial locality more effectively.
• **Associativity:** The number of ways a main memory block can be mapped to a cache set. Direct-mapped caches are simple but suffer from conflict misses.
Fully associative caches offer maximum flexibility but are complex and expensive. Set associative caches provide a balance between these extremes, reducing conflict misses without excessive hardware complexity.
• **Replacement Policies:** Cache replacement policies determine which block to evict when a new block needs to be
brought into a full cache set. The choice of policy significantly impacts cache performance.
Hit Rate (HR):In a set-associative cache, when a miss occurs and a new block must be inserted into a set, the insertion policy determines the new block is prioritized relative to the existing blocks in the set.
• **Insertion Policies:** A cache insertion policy is the rule that decides where a new block of data should be placed into the cache *after* a cache miss has occurred.
1. Most Recently Used (MRU) Insertion Policy: When a cache miss occurs, the new block is fetched from memory and placed in the Most Recently Used (MRU) position within its designated set.
2. **Least Recently Used (LRU) Insertion Policy :** When a cache miss occurs, the new block is fetched from memory but is placed in the Least Recently Used (LRU) position within its set.
The functionality of both the MRU and LRU insertion policies is intrinsically dependent on the LRU replacement framework. The concept of an "MRU position" or an "LRU position" only exists because the LRU algorithm maintains a ranked list or recency chain of the blocks within each set.

3.2. Performance Metrices

• **Hits:**Number of memory accesses that find the data in the upper level
• **Misses:**Number of memory accesses not finding the data in the upper level
• **Instruction Per Cycle (IPC):**The average number of instructions the CPU successfully executes in a single clock cycle.
IPC = Total Number of Instructions Executed / Total Number of Clock Cycles Taken

3.3. Benchmarks

They are not just random tests; they are diagnostic tools designed to stress and expose different aspects of the cache design.
• **Sequential Access Benchmark:** A sequential access benchmark is a program that reads or writes to memory addresses in a contiguous, linear order. Primary Purpose: To Test for Spatial Locality. This benchmark is the ultimate test for how well cache system exploits spatial locality. When the CPU requests data from `address[i]`, a well-designed cache will fetch a block of data that includes not only `address[i]` but also `address[i+1]`, `address[i+2]`, etc.
• **Random Access Benchmark:** A random

access benchmark is a program that reads or writes to memory addresses that are scattered unpredictably across the memory space. There is no predictable pattern to the accesses. Primary Purpose: To Stress-Test the Cache Under Worst-Case Conditions , This benchmark is designed to simulate workloads with poor or no locality. It effectively tests how your cache behaves when its predictive capabilities are rendered useless.

4. Expected Results

● **Impact Of Cache Size:**Increasing the cache size generally leads to a higher hit rate and a lower miss rate. This is because a larger cache can store more data, increasing the probability that a requested piece of data will be found in the cache.

	1KB	**2KB**	**4KB**
IHits	101126	101126	101126
IMiss	5	5	5
DHits	256	256	256
DMiss	256	256	128
IPC	0.592	0.592	0.627
Cycles	114176	114176	107776

Table 1: CacheBenchmark.x

● **Impact Of Block Size:**Increasing the block size can improve performance by exploiting spatial locality. A larger block size brings more of this spatially local data into the cache with a single fetch. This reduces the number of cache misses.

	16B	**32B**	**64B**
Hits	30	30	30
Misses	8	4	2
IPC	0.021	0.025	0.028
Cycles	1230	1030	930

Table 2: sequential access.x

● **Associativity:**Associativity directly influences the flexibility of block placement within the cache. Higher
associativity generally leads to lower misses, particularly by reducing conflict misses

	4-way	**8-way**	**16-way**
Hits	16	16	16
Misses	16	8	8
IPC	0.025	0.041	0.041
Cycles	1030	630	630

Table 3: sequential access.x

● **Impact Of Replacement Policies**
LRU is generally considered the most effective policy for programs
exhibiting temporal locality, as it aims to keep the most frequently and recently used data in the cache. This directly translates to a lower misses.
FIFO Compared to LRU, FIFO is simpler to implement as it doesn't require tracking access recency. However, its performance is generally inferior to LRU.
Random its performance is highly unpredictable and generally the
worst among the common policies. It does not leverage any form of locality
● **Impact Of Victim Cache** The integration of a victim cache enhanced cache performance by mitigating conflict misses.

	LRU	FIFO	Random	VictimCache
Hits	MRU:16,LRU:16	16	..	Dcache Hits:16 /Vcache Hits:0
Misses	MRU:16,LRU:13	16	..	Dcache Misses :0 / Vcache Miss:8
IPC	MRU:0.025,LRU:0.030	0.025	0.030 /0.028 /..	0.041
Cycles	MRU:1030,LRU:880	1030	880/930/..	630

Table 4: sequential access.x

5. Conclusion

we have successfully designed and integrated a cache memory system, providing a clear and practical demonstration of fundamental computer architecture principles. The journey from a simple, non-cached pipeline to one with a split cache, a victim cache, and configurable management policies has offered deep insight into the challenges and rewards of processor optimization. Our benchmark analysis confirmed that each layer of complexity contributes to achieve our goal of minimizing memory latency. this project has met all its objectives, it also opens avenues for future exploration. The current framework could be extended to include a multi-level L2 cache, more sophisticated prefetching mechanisms. the simulator stands as a powerful and effective tool for studying the intricate and critical relationship between processor and memory.