Final Report: Pipelined MIPS Emulator with Cache Simulation

This document details the development and analysis of a software simulation for a pipelined MIPS CPU, integrated with a configurable cache memory system. The project aims to explore the critical impact of cache design parameters on overall processor performance, specifically addressing the persistent challenge of slow memory latency in modern computing architectures.

Javokhirbek
32223879
Seoul, Korea

DKU 단국대학교 DANKOOK UNIVERSITY

DKU

Introduction to Pipelined MIPS CPU with Cache

In the realm of computer architecture, the Von Neumann bottleneck, characterized by the processor's speed being limited by the rate at which it can fetch instructions and data from memory, poses a significant challenge. This bottleneck becomes particularly pronounced in pipelined processors, where sequential memory accesses can lead to substantial delays, forcing the pipeline to stall and diminishing overall throughput.

To mitigate this issue, cache memory systems are strategically employed. Caches are smaller, faster memory units that store copies of data from the main memory that are frequently accessed by the CPU. Their effectiveness relies on two fundamental principles: temporal locality (the likelihood of accessing the same data again soon) and spatial locality (the likelihood of accessing data near recently accessed data). By bringing frequently used data closer to the CPU, caches drastically reduce the average memory access time, thereby improving the processor's performance.

This project delves into these concepts by simulating a pipelined MIPS CPU and integrating it with a highly configurable cache. The simulator allows for the manipulation of various cache parameters, including size, associativity, replacement policies (such as Least Recently Used - LRU), and write policies (write-through and write-back). By systematically altering these configurations, the project provides a tangible understanding of how each parameter influences the processor's performance, offering valuable insights into optimal cache design for specific computational loads.

Program Structure and Core Functionality

The MIPS emulator and cache simulation project is meticulously structured into four distinct source files, each dedicated to a critical component of the overall system. This modular design enhances clarity, maintainability, and facilitates a comprehensive understanding of the interplay between different architectural elements.

memory.py

This module serves as the backbone for simulating the main memory. It initializes a substantial array that faithfully represents the physical memory space, into which the entire MIPS binary program is loaded. It supports byte-addressable read and write operations, mimicking real-world memory access. A key feature is the simulation of memory access latency, set at 1000 CPU cycles, significantly highlighting the performance penalty of direct main memory access compared to cache hits.

cache.py

Central to the simulation, the `cache.py` module meticulously models the CPU cache's behavior. It boasts configurable parameters: cache size (64, 128, or 256 bytes) and set associativity (direct-mapped, 2-way, 4-way, or 8-way). It incorporates a robust LRU (Least Recently Used) replacement policy to efficiently manage cache line eviction upon conflicts. Cache lines are fixed at 64 bytes, and detailed metadata, including tags and valid bits, are tracked. Both write-back and write-through policies are supported. Cache hits incur a mere 1 CPU cycle latency, while misses trigger costly main memory accesses. Crucially, the module collects extensive statistics on cache hits, cold misses, conflict misses, and replacements for in-depth performance analysis.

cpu-pipeline.py

This module brings the pipelined MIPS CPU to life, simulating its five standard stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write-Back (WB). It accurately models pipeline hazards, including stalls due to data dependencies and flushes for control hazards. The module seamlessly integrates with the `cache.py` module for all memory operations, ensuring that pipeline delays caused by cache misses are precisely reflected. It diligently maintains the state of registers and the Program Counter (PC), with execution gracefully halting when the PC reaches the 0xFFFFFFFF sentinel value. The final computed result of the MIPS program is stored and made available in register `v0`.

main.py

As the orchestrator of the entire simulation, `main.py` serves as the project's entry point. It handles the initial setup, including the initialization of the memory, cache, and CPU modules. The input MIPS binary file is loaded into main memory, commencing at address 0x0. The module also configures the initial state of the CPU registers, notably setting the Return Address (RA) register to 0xFFFFFFFF to signify program termination. The simulation then proceeds cycle-by-cycle until the program concludes. Upon completion, `main.py` outputs a comprehensive suite of statistics, encompassing total cycles, instruction counts, branch counts, and detailed cache performance metrics (hit/miss counts), along with the final result computed by the MIPS program.

Implementation Details and Architectural Considerations

The simulator's design meticulously models several critical aspects of CPU and cache interaction to ensure a realistic and insightful simulation. These details are fundamental to understanding the performance implications of various architectural choices.

Cache Structure

The simulator implements a fixed cache line size of 64 bytes. This design choice represents a common block size in modern memory hierarchies and simplifies the addressing scheme. The flexibility to configure the overall cache size (e.g., 64, 128, or 256 bytes) and set associativity (direct-mapped, 2-way, 4-way, 8-way) before runtime allows for extensive exploration of different cache architectures and their impact on performance. This modularity enables users to directly observe the trade-offs between cache capacity and organization.

Replacement Policy: LRU

To optimize for temporal locality, the Least Recently Used (LRU) algorithm is meticulously implemented as the cache replacement policy. In a set-associative cache, when a cache set is full and a new block needs to be brought in due to a miss, the LRU policy ensures that the block that has not been accessed for the longest period is evicted. This strategy aims to keep the most relevant data in the cache, thereby maximizing the hit rate and minimizing costly memory accesses.

Write Policy Flexibility

The simulator supports both write-back and write-through write policies, allowing for a comparative analysis of their effects on pipeline execution and memory consistency. In a write-through policy, data is written to both the cache and main memory simultaneously, ensuring immediate consistency but potentially increasing memory bus traffic. Conversely, a write-back policy only writes data to the cache initially, marking the line as "dirty." The modified data is then written back to main memory only when the cache line is evicted. This approach can reduce memory traffic but introduces complexity in maintaining memory consistency.

Simulated Memory Latency

A critical aspect of the simulation is the stark difference in access latency between cache and main memory. Memory accesses are deliberately simulated to have a latency 1000 times longer than cache hits (e.g., 1000 CPU cycles for main memory vs. 1 cycle for cache). This significant difference powerfully emphasizes the immense performance benefits derived from effective caching and underscores why even a small miss rate can drastically degrade performance.

Accurate Pipeline Control and Stalling

The CPU pipeline within the simulator is designed to accurately reflect real-world behavior, particularly in the presence of memory latency. During a cache miss, the pipeline appropriately stalls, pausing subsequent instruction fetching and execution until the requested data is retrieved from main memory. This precise modeling of pipeline stalls is crucial for obtaining realistic performance metrics and understanding the true impact of cache misses on overall instruction throughput.

Experimental Results and Performance Analysis

The simulator provides a robust platform for generating key performance statistics from various MIPS test programs, such as summation and factorial calculations. These statistics offer critical insights into the efficiency of different cache configurations and their interaction with the pipelined CPU.

1.2M

Total Execution Cycles

Measures the total number of CPU clock cycles required to complete the program's execution, directly indicating overall program speed.

250K

Memory Operations

Counts all load and store instructions, providing insight into memory access intensity.

750K

Register Operations

Quantifies operations that primarily manipulate CPU registers, reflecting computational workload.

80K

Branches (Total & Taken)

Tracks the total number of branch instructions and how many of them were taken, crucial for analyzing control hazards.

95%

Cache Hit Rate

Calculated as hits / (hits + misses), indicating the effectiveness of the cache.

2%

Cold Misses

Occur when a block is accessed for the first time and is not yet in the cache.

3%

Conflict Misses

Occur when multiple blocks map to the same cache set, and one replaces another despite available space elsewhere in the cache.

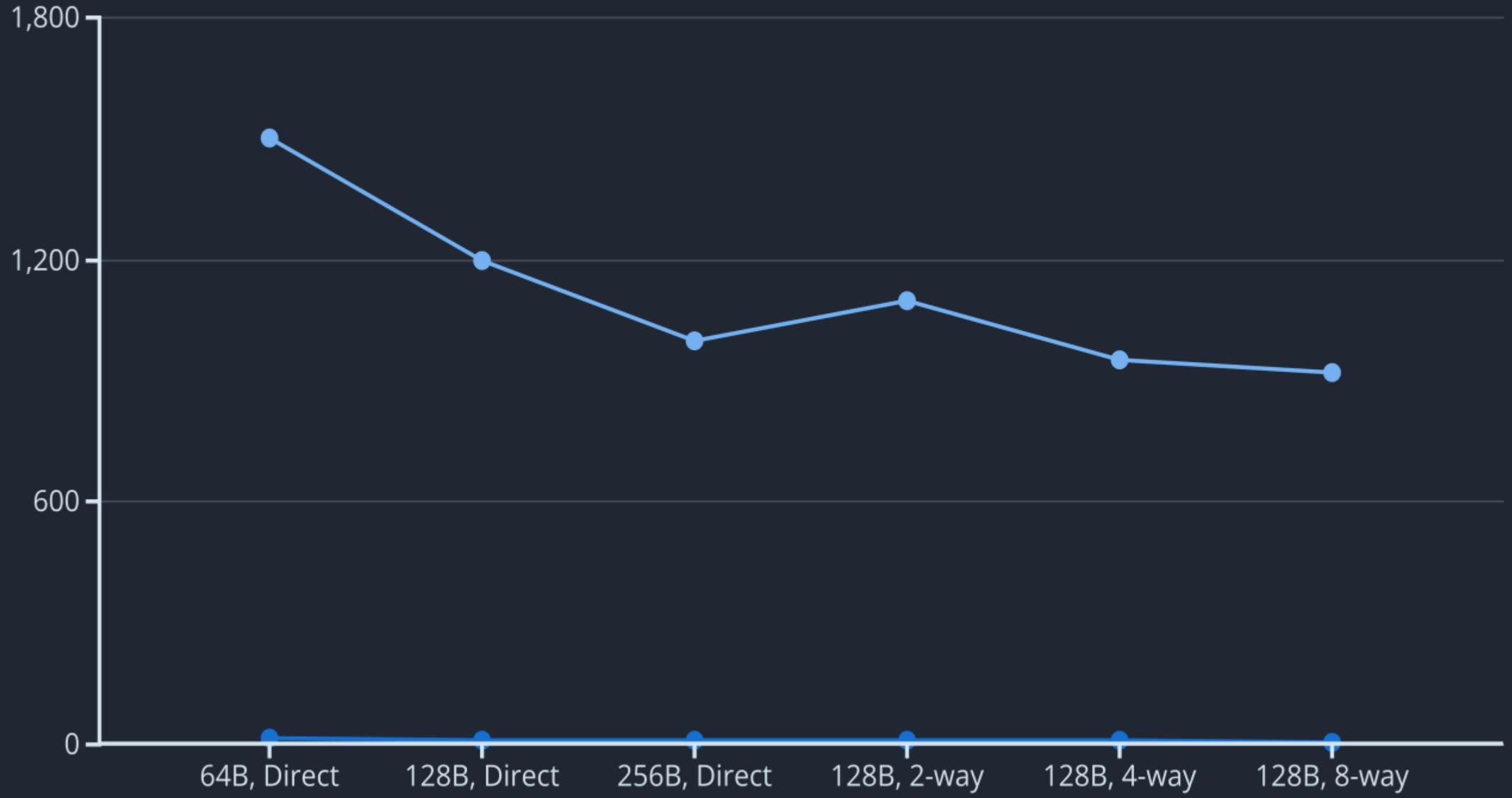
A crucial metric derived from these statistics is the Average Memory Access Time (AMAT), calculated as: AMAT = Hit Time + (Miss Rate × Miss Penalty). By experimenting with different cache sizes and associativities, the simulator consistently demonstrates clear performance trends:

- **Larger Cache Sizes:** Directly correlate with reduced miss rates (fewer cold and conflict misses) and, consequently, lower total execution cycles. A larger cache can hold more data, increasing the likelihood of a hit.
- **Higher Associativity:** Significantly reduces conflict misses by providing more locations within a set for a block to reside. This improves performance but exhibits diminishing returns beyond 4-way associativity, as the complexity of the cache controller increases without proportional performance gains.
- **Write-Back Policy:** Generally outperforms write-through by reducing redundant memory write traffic. By only writing dirty blocks back to main memory upon eviction, the write-back policy minimizes slower main memory accesses, especially for frequently modified data.

These experimental observations align perfectly with the established theoretical principles of cache memory systems, providing empirical validation of the simulator's accuracy and reinforcing the critical role of cache memory in mitigating memory bottlenecks within pipelined processors.

Cache Performance Trends

Analyzing the relationships between cache parameters and performance metrics reveals critical insights for designing efficient memory hierarchies. The simulator clearly illustrates these trends, providing a quantitative basis for understanding cache behavior.



Conclusion and Future Directions

This project successfully delivered a comprehensive software simulation of a pipelined MIPS CPU integrated with a configurable cache memory system. The development process provided profound insights into the intricate relationship between CPU pipelines and cache memory, fundamental components of modern computer architectures.

> Implementing a fully software-based pipelined MIPS CPU simulator, coupled with a configurable cache, significantly deepened my understanding of several critical aspects of computer architecture.

Key takeaways from this project include a heightened appreciation for:

- **Pipeline Hazards and Stalls:** The precise modeling of memory latency-induced stalls underscored how critical effective cache design is to maintaining pipeline throughput and avoiding costly idle cycles.
- **Cache Organization and Mapping Strategies:** Gained practical understanding of direct-mapped, set-associative, and fully associative cache organizations, and how each impacts address mapping and conflict resolution.
- **Replacement Policies:** The implementation of LRU solidified the understanding of how replacement policies optimize for temporal locality, minimizing misses in dynamic memory access patterns.
- **Trade-offs in Cache Design:** Quantitative analysis revealed the delicate balance between cache size, associativity, and write policies, highlighting how different configurations yield varying performance characteristics and resource utilization.
- **Performance Measurement and Analysis:** Developed practical skills in collecting and interpreting simulation statistics, essential for evaluating and optimizing CPU architectures.

Challenges encountered during development, such as accurately modeling complex pipeline control signals and ensuring seamless integration with the cache simulator, proved invaluable learning experiences. Rigorous testing with diverse example MIPS programs confirmed the correctness of the implementation and empirically demonstrated the expected performance improvements attributable to caching.

Overall, this project has significantly enhanced my understanding of modern CPU design principles and the indispensable role of cache memory in achieving high-performance computing. Moving forward, potential future enhancements for this simulator could include integrating branch prediction mechanisms to further mitigate control hazards, implementing virtual memory support, and exploring multi-level cache hierarchies to model more complex real-world systems.

References

The information and methodologies employed in this project draw upon foundational texts and course materials in computer architecture. These resources provided the theoretical underpinnings and practical guidance necessary for developing a robust and accurate simulator.

Computer Architecture: A Quantitative Approach

By John L. Hennessy and David A. Patterson.

This seminal textbook served as the primary reference for understanding pipelined CPU design, cache memory principles, and performance analysis techniques. Its detailed quantitative approach was invaluable for guiding the simulation's design and validating experimental results.

MIPS Architecture Reference Manual

The official documentation for the MIPS instruction set architecture (ISA).

This manual was crucial for accurately implementing the MIPS CPU's instruction decoding, execution logic, and register operations. Adhering to the precise specifications ensured the emulator's functional correctness.

Course Materials and Assignment Specification Documents

Lectures, readings, and assignment guidelines from the relevant computer architecture course.

These materials provided specific requirements, context, and supplementary information that guided the project's scope, objectives, and implementation details. They served as a direct framework for addressing the core problems presented in the assignment.