CSC3050 Project 3: Cache Simulation

April 12, 2024

1 Introduction

Cache is an important component of a CPU system that has a significant impact on computer performance by reducing memory access times. The focus of this project is to simulate the cache in the RISC-V architecture to give you hands-on experience with the cache system and its role in improving system performance.

2 Overview

This project is divided into three main parts:

1. **Single-Level Cache Simulation:** In this part, you are required to implement a cache simulator that enables the single-level cache simulation. After that, you are required to simulate different cache conditions for single-level cache and compare how different cache conditions affects the cache performance.
2. **Multi-level Cache Simulation:** In this part, based on the single-level cache simulator, you are required to further implement the multi-level cache simulation. You are required to further exam how multi-level cache can improve the performance.
3. **Integration with CPU Simulation:** In this part, you are required to integrate the cache simulator with CPU simulator that simulates a pipeline CPU with multi-level cache. You will also do some performance comparison between CPU with cache and CPU without cache.

3 Single-Level Cache Simulation

- **Implementation Requirements:** You are required to implement a *Cache* class for simulating a single-level cache (The code from [1] can serve as a reference for your implementation). You can directly use the *MemoryManager* class implemented in [1] to integrate your cache with memory. The file structure and description you may used are shown in Table 1.

  The simulated cache should be able to perform some parameter tuning such as cache size, block size and associativity level. Besides that, you are required to simulate

file name	Discription
include/Cache.h	Statement of the Cache class.
include/MemoryManager.h	Statement of the MemoryManager class.
src/Cache.cpp	Implementation of Cache class.
src/MemoryManager.cpp	Implementation of the MemoryManager class.
src/MainSinCache.cpp	Main entrance of the single-level cache simulator.
src/MainMulCache.cpp	Main entrance of the multi-level cache simulator.

Table 1: File structure and description of single-level and multi-level cache simulation.

Write Back and Write Allocate policies using the LRU replacement algorithm in your simulation. Finally, some performance data (e.g. miss rate of the cache and total access latency) needs to be saved in a CSV file.

- **Performance Evaluation:** For single-level cache simulation, you should compare the system performance for different parameter combinations as given in Table. 2. After

Parameter	Values
Cache Size	4KB to 1MB, incremented by 4X.
Block Size	32Bytes to 256Bytes incremented by 2X.
Associativity	2 to 32 incremented by 2X
Write Back	True or False.
Write Allocate	True of False.

Table 2: Parameters used in single-level cache simulation.

that, you are required to do some analysis of the results of the simulation (We will provide you with two test traces (trace1.trace and trace2.trace) and you can choose one of them for testing and analysis). This includes, but is not limited to

  - Analyzing the trend of Miss Rate with Block Size under different cache sizes,
  - Analyzing the change of Associativity with Miss Rate under different cache sizes,
  - Analyzing the amount of cache misses per thousand instructions under different cache sizes.

You should provide graphical or tabular data and conduct the analysis based on the data mentioned above. You should put your results and analysis in your report.

4 Multi-Level Cache Simulation

- **Implementation Requirements:** You are required to further adjust the *Cache* class you have already implemented for single-level cache to enable the multi-level cache simulation. Your implementation is required to support two different types of multi-level cache: inclusive cache and exclusive cache (The referenced code in [1] is implemented as the inclusive cache). Besides that, Your implementation is also required to support a victim cache (The referenced code in [1] does not implement the victim cache). Specifically, the victim cache is set to the size of 8 lines of data, and the L1 cache is direct mapped (which means the associativity is one).

- **Performance Evaluation:** For multi-level cache simulation, you should conduct the following comparisons based on one of the test traces:
  - Comparing the performance between the single-level cache with the inclusive three-level cache whose parameters are given in Table 3 and Table 4, respectively.
  - Comparing the performance between the inclusive and exclusive three-level cache (without the victim cache) whose whose parameters are given in Table 4.
  - Comparing the performance between the three-level inclusive cache with and without victim cache.

  Also, graphical or tabular data are required and you should put the above comparisons and analysis in your report.

Level	**Capacity**	**Associativity**	**Block Size**	**Write Policy**	**Hit Latency**
L1	16 KB	1 way	64 Bytes	Write Back	1 CPU Cycle

Table 3: Default cache setting used in single-level cache simulation.

Level	**Capacity**	**Associativity**	**Block Size**	**Write Policy**	**Hit Latency**
L1	16 KB	1 way	64 Bytes	Write Back	1 CPU Cycle
L2	128 KB	8 ways	64 Bytes	Write Back	8 CPU Cycle
L3	2 MB	16 ways	64 Bytes	Write Back	20 CPU Cycle

Table 4: Default cache setting used in multi-level cache simulation.

5 Integration with CPU Simulator

- **Implementation Requirements:** You are required to integrate multi-level cache with the pipeline CPU. You can directly use the CPU simulator in [1] to simplify your work. In this part you need to simulate a inclusive three-level cache without victim cache. The cache parameters are set according to Table 4.
- **Performance Evaluation:** You should run some RISC-V programs (We will provide some reference programs from [1]) to compare and analyze the CPI of each RISC-V program with and without Cache. An example is given in Table 5. You should put your data and analysis on your report.

File name	CPI without Cache	CPI with Cache
helloworld	1.3333	1.6286

Table 5: Example of the CPI comparison.

6 Submission

For this project, you must use C/C++ to implement the cache simulator. If you use python, you will get a 0 score. You need to submit the following files:

- src/*: include all source code files
- include/*, include all header files
- CMakelists.txt, the cmake file for your project
- project-report.pdf: a detailed introduction to your project. The specific things that need to be included are as follows:
  - The implementation details of your simulator (how to implement victim cache and exclusive cache etc.)
  - Performance analysis and discussion mentioned above.

Please compress all files into a single zip file and submit it to the BlackBoard. The file name should be your student ID, like xxxxxxxxx.zip.

7 Grading Details

The overall score will be calculated as follows:

- Single-level cache simulation: 20%
- Multi-level cache simulation: 30%
  - Implementing inclusive multi-level cache simulation without victim cache: 10%
  - Implementing exclusive multi-level cache: 10%
  - Implementing victim cache: 10%
- Integrating multi-level cache with CPU: 10%
- Report: 40%
  - Performance analysis for single-level cache simulation: 10%.
  - Performance analysis for multi-level cache simulation: 20%.
  - Performance analysis for integration with CPU: 10%.

- About the reference code: To reduce the difficulty and complexity of implementation, we encourage you to refer to existing code like [1]. This project is also designed based on [1]. However, if you simply submit the code from the reference [1] or only do simple tasks like adding comments, we consider that you haven't put much effort into this project, and your grade will be directly marked as zero.

References

[1] Hao He, "RISCV-Simulator," https://github.com/hehao98/RISCV-Simulator, 2019.