International Multidisciplinary Research Journal
ISSN: 2424-7073
Available Online at https://imrjournal.info/
Volume:1; December 2023; Page No.22-28
DOI: https://doi.org/10.47722/imrj.2001.18

Research Article

A THREE-STEP TEACHING MODEL FOR MANY-CORE CACHE MEMORY DESIGN

[1]Yul Chu, [2]Yan Bai, and [3]John J. Lee

[1]University of Texas Rio Grande Valley, 1201 W. University Dr., Edinburg, TX 78539, U.S.A.
[2]University of Washington, 1900 Commerce St, Tacoma, WA 98402, U.S.A.
[3]Indiana University-Purdue University Indianapolis, 420 University Blvd., Indiananpolis, IN 46202, U.S.A.

ARTICLE INFO

Article History:

Received 22.07.2023
Accepted 15.11.2023
Published 05.12.2023.

Keywords:

manycore, cache memory, cache coherency

ABSTRACT

The shift in processor design towards manycore architectures necessitates effective management of data in cache memories, posing challenges related to cache coherence and parallel computing. This paper proposes a three-step teaching model to meet the demand for designing cache memory for many-core systems. The steps include: 1) acquiring a comprehension of an open-source simulator like FM-SIM (Flexible Multicore Cache Simulator), 2) learning the process of collecting trace files using Pin Tools, and 3) designing and integrating a new cache memory and/or cache coherence protocol into FM-SIM to assess their efficacy. The proposed model has demonstrated its capability to enable students to make significant progress in exploring and researching modern cache memory design, particularly in the context of manycore architectures.

1. Introduction

Cache memory, a small and high-speed memory type, is employed to store frequently accessed data and instructions, aiming to accelerate processing times. Positioned as an intermediary between the main memory (RAM) and the CPU, cache memory significantly contributes to enhancing the overall performance of computer systems. Its role involves reducing memory access latency and optimizing the efficiency of data retrieval from the main memory to the CPU [1].

In light of technological trends, the escalating importance of many-core cache memory design in the industry emphasizes the increasing demand for processors with multiple cores to achieve enhanced computational power through parallelism. Many-core architectures are becoming prevalent in various applications, ranging from high-performance computing to embedded systems. However, efficiently managing data in cache memories among many cores, while maintaining cache coherence poses substantial challenges [2-3].

Current educational methodologies may not adequately prepare students for these challenges. Conventional curricula often concentrate on single-core or dual-core architectures, potentially limiting exposure to the complexities of many-core cache memory design. The industry's transition towards manycore processors necessitates an educational paradigm shift to equip students with the necessary knowledge and skills to address these complexities.

To bridge this educational gap, methodologies should incorporate comprehensive modules on many-core cache memory design. This includes hands-on experience with simulators like FM-SIM [4] and

Pin Tools [5], exposing students to real-world challenges in data management and cache coherence in many-core environments. Furthermore, students should be encouraged to explore the integration of new cache memory and coherence protocols, fostering a deeper understanding of the unique design considerations associated with many-core architectures.

In summary, the industry's increasing reliance on many-core architectures underscores the critical importance of many-core cache memory design. Educational methodologies must evolve to provide a more comprehensive and practical understanding of many-core cache memory design, ensuring graduates are well-equipped to contribute effectively to the evolving landscape of processor design in the industry.

1.1 Related works for Current Educational Methodologies and Open-Source Simulators

Pedagogical approaches in teaching cache memory mainly involve two methodologies: 1) lecture-based learning and 2) project-based learning. Lecture-based learning follows the traditional classroom teaching model, where instructors verbally convey information using tools like whiteboards and projectors [6]. While offering tailored learning materials and control over content delivery, lecture-based learning has limitations when delving into the complexities of cache memory, requiring specialized knowledge from instructors.

In contrast, project-based learning aims to deepen comprehension by fostering higher-order thinking and problem-solving skills [7]. It involves research and cache memory design based on literature reviews, motivation, and problem definition. Successful project-based learning requires essential tools: a simulator (software architecture) and benchmark programs facilitating the design and evaluation of new many-core cache memory systems.

For simulators, project-based learning can leverage open-source cache memory simulators like Gem5 [8] or Multi2Sim [9]. However, these simulators have historically been challenging for students due to their complexity. There is a clear need for an easily accessible and flexible simulator, along with benchmark programs, to facilitate the design and implementation of many-core cache memory systems.

In computer science and computer engineering, term projects play a crucial role in project-based learning, allowing students to apply theoretical and experimental knowledge to address complex design issues related to cache memory. Providing concise and effective guidelines for many-core cache memory design, empowering students to successfully complete term projects, remains a significant challenge. This paper addresses this challenge by introducing a meticulously designed three-step graduate-level term project as part of project-based learning, aimed at enhancing students' knowledge and technological proficiency in the field of many-core cache memory design.

The remainder of this paper are organized as follows: Section 2 delineates the three-step teaching model for designing many-core cache memory and sample outcomes. Section 3 delves into the evaluation process and discussion. Finally, Section 4 serves as the conclusion and future work of the paper.

2. Three-step teaching model for designing many-core cache memory and sample outcomes.

In recent times, processor design has shifted towards architectures featuring many-core and GPU (Graphics Processing Unit), boasting tens to hundreds or even more cores to enhance computational power through parallelism [2]. However, efficiently managing data in cache memories across numerous cores while maintaining cache coherence remains a challenging task. As many-core technology continues to advance, the integration of many-core cache memory design term projects, alongside cache coherency protocols, presents a new challenge in the realm of graduate-level cache memory design [3].

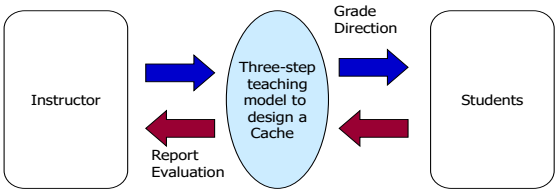

Figure 1. The overview of the three-step term project.

To address these challenges in both research and education, this section introduces a

comprehensive three-step term project. Figure 1 provides an overview of the three-step teaching model, encompassing the following aspects: 1) The instructor furnishes a detailed procedure outlining the three steps for cache memory design and grading guidance, and 2) students meticulously follow each procedure and subsequently report their results for evaluation. Through this three-step term project, students gain valuable experience in designing a novel cache memory using FM-SIM [4] and Pin Tools [5]. Upon completion of the project, students possess a clear understanding of how to engage in research with well-defined concepts and procedures.

The three-step term project is composed of two laboratory (lab) steps, Step 1 and Step 2, and one project step, Step 3. Here are the specifics for each of these steps:

Step 1: Lab for Coding Simulator (FM-SIM) and Sample Outcomes

The primary goal of step 1 is to develop proficiency in using the FM-SIM, a simulator developed by Rano Mal and Yul Chu [4]. FM-SIM serves as a crucial tool for designing and evaluating many-core cache memories and is implemented in C++.

The three-step term project is composed of two laboratory (lab) steps, Step 1 and Step 2, and one project step, Step 3. Here are the specifics for each of these steps:

Step 1: Lab for Coding Simulator (FM-SIM) and Sample Outcomes

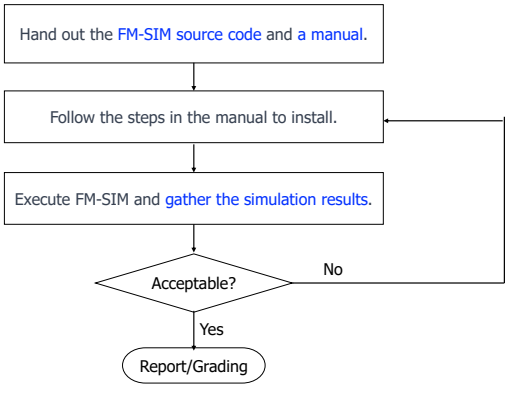

Figure 2. Lab for coding FM-SIM.

Figure 2 illustrates the installation and execution of FM-SIM. During step 1, students delve deep into FM-SIM's intricacies by closely examining its source code. The process is outlined as follows:

1. The instructor provides students with the FM-SIM source code, a comprehensive manual, and sample trace files.

2. Students meticulously follow the instructions outlined in the manual to install FM-SIM on their laptops or desktop computers.

3. Students execute FM-SIM and gather simulation results for many-core cache memories, which are pre-programmed in FM-SIM. They accomplish this by employing provided sample trace files.

4. If the simulation results align with expected outcomes, students submit their lab reports for grading.

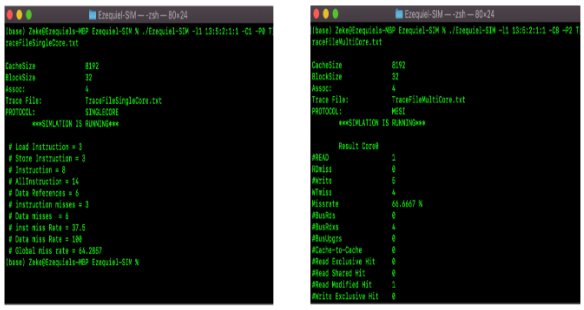

Figure 3. Single-core and Four-core sample outcomes.

Figure 3 provides sample outcomes collected during Step 1, utilizing the provided sample trace files. The left side displays single-core simulation results, revealing cache miss rates for a Level 1, 2-way, and 8KB set-associative cache using a provided trace file. On the right side, you can observe the 4-core simulation results, displaying cache miss rates for a 4-core configuration, Level 1, 2-way, and 8KB set-associative cache, also employing a provided trace file.

Step 2: Lab for Generating Trace Files Using Pin Tools and Sample Outcomes

The primary purpose of step 2 is to instruct students on collecting trace files using Pin Tools, an open-source dynamic instrumentation tool provided by Intel [5].

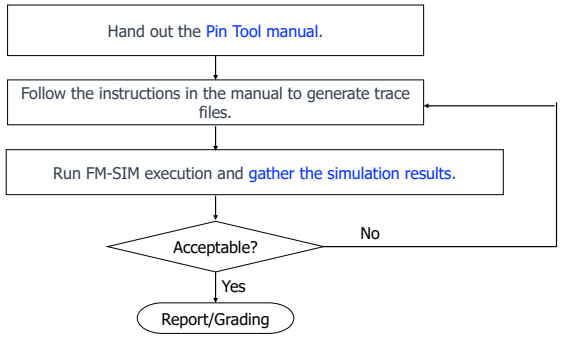

Figure 4. Lab for generating trace files using Pin Tool.

Figure 4 provides a comprehensive guide on generating and collecting trace files. During Step 2, students acquire proficiency in using Pin Tools by carefully studying the provided manual.

Figure 4 outlines the following procedures:

1.The instructor supplies the Pin Tools manual.

2.Students meticulously follow the manual's instructions to generate trace files.

3.Students execute FM-SIM using the collected trace files and gather simulation results for conventional cache memories.

4.If the simulation results align with expectations, students submit their lab reports for grading.

Figure 5 illustrates sample outcomes of FFT trace file results generated during Step 2. The left side showcases single-core FFT trace files, while the right side displays eight-core FFT trace files [11].

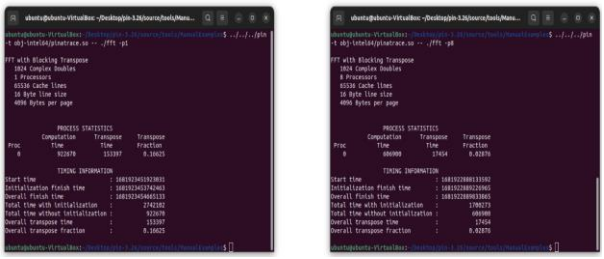

Figure 5. Generating Single-core FFT Trace files and Eight-core sample outcomes.

Step 3: Project for Cache Memory Design and Sample Outcomes

The primary objective of Step 3 is to design a cache memory and port its code into the FM-SIM. Figure 6 provides a detailed explanation of how to design a cache memory and integrate its code into the FM-SIM. During this step, students acquire an in-depth understanding of cache memory design and the process of code integration.

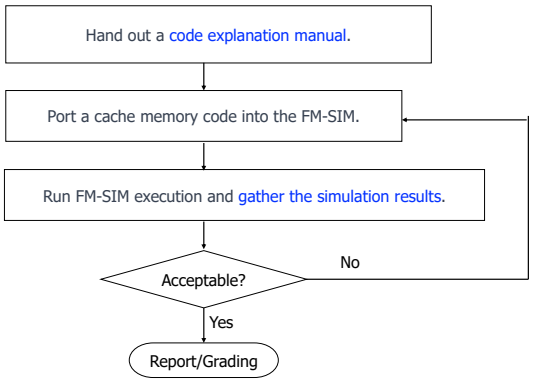

Figure 6. Project for designing a cache memory.

Figure 6 delineates the following procedures:

1. The instructor provides a comprehensive code explanation manual, demonstrating how to design a cache memory using C++.
2. Students diligently follow the manual's instructions to integrate their designed cache memory code into the FM-SIM.
3. Students execute FM-SIM using the collected trace files and gather simulation results specific to their designed cache memory.
4. If the simulation results align with expectations, students submit their project reports for grading.

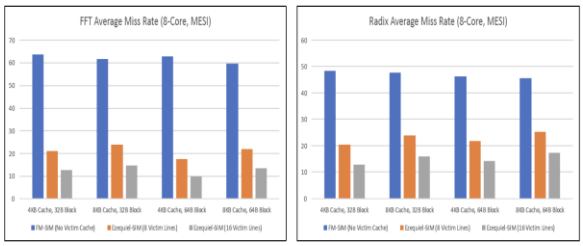

Figure 7. Eight-core FFT results (left) and Radix results (right).

Figure 7 presents sample outcomes of 8-core victim cache (4KB and 8KB cache) miss rates compared to 8-core direct-map cache (4KB and 8KB cache) using FFT and Radix trace files generated during Step 2 [11]. The left side employs the eight-core FFT trace files, while the right side utilizes Radix trace files. The results indicate that victim cache exhibits significantly lower cache miss rates compared to direct-map caches.

3. Evaluation process and discussion

In this section, we will outline the cache memory evaluation process and discuss the significance of benchmark programs, simulators, and Pin Tools in assessing cache memory. We will also delve into students' performance and feedback on the process.

3.1 Evaluation process for a Newly Designed Cache Memory

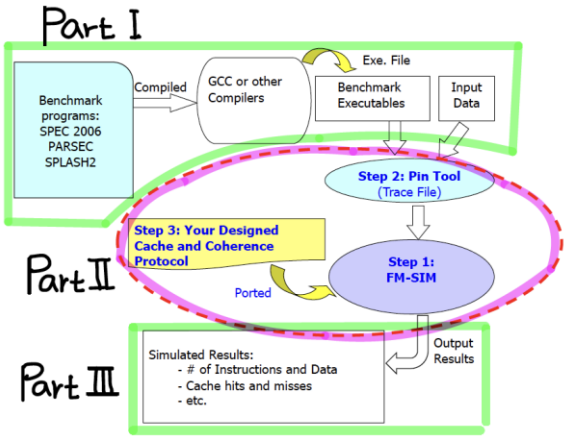

Figure 8. Cache memory evaluation process

Figure 8 illustrates the evaluation process for a newly designed cache memory, comprising three key parts, such as Part I, II, and III.

3.1.1 Part I: Compilation of Benchmark Programs

This stage encompasses the compilation of benchmark programs, actual application programs, to produce executable files. The benchmark programs employed in this context include SPEC 2006, PARSEC, and SPLASH2.

3.1.2 Part II: Trace File Collection and Many-Core Cache Simulation

In this stage, trace files are collected using the generated execution files as inputs to Pin Tools. Subsequently, the C++ code for the designed cache memory is integrated into the simulator (FM-SIM), and the simulation is executed using the trace files. Part II is the linchpin of this evaluation process, where students or researchers are actively engaged in critical tasks such as trace file collection, code porting for the cache memory into FM-SIM, and simulator execution. Given that Part II can often be complex and challenging for students, we have developed and presented a three-step teaching model in this paper.

3.1.3 Part III: Gathering Simulation Outcomes

This stage involves gathering the outcomes of the many-core cache memory simulations, including metrics like cache miss rates and execution time.

3.2 Discussion of Benchmark Programs, Simulators, Pin Tools, and Student Performance

3.2.1 What are benchmark programs?

Benchmark programs designed for simulating cache memory act as standardized sets of tests or challenges. Their purpose is to assess the performance of a computer's cache memory in various scenarios, mimicking real-world activities like working with spreadsheets, databases, or playing games. These tests play a crucial role for researchers and engineers, allowing them to evaluate how well the computer's cache memory performs under different conditions. Essentially, it's akin to assigning different tasks to the computer to understand how effectively it can handle diverse challenges.

Prominent examples of well-established benchmark programs for cache memory simulation include SPEC CPU benchmarks, PARSEC, and SPLASH2, particularly for many-core cache memory implementations [10-11]. These benchmarks cover various applications and memory access patterns, facilitating a thorough assessment of cache memory systems. Researchers often rely on these benchmarks for performance analysis and cache hierarchy optimization within computer architecture research.

3.2.2 Simulators and Pin Tools

A cache memory simulator is specialized software, or a program tailored to model and simulate cache memory system behaviour within a computer's architectural framework. These simulators are indispensable for research in computer architecture, enabling performance analysis and optimization of cache memory. They offer a platform for experimenting with different cache configurations, policies, and parameters without the need for physical hardware, making cache memory design and analysis cost-effective and flexible.

Prominent cache memory simulators used in computer architecture research include Gem5, Multi2Sim, and SimpleScalar, all of which are open source. However, these simulators are often complex and challenging to grasp, necessitating a more accessible solution. To address this, we introduced FM-SIM for students to implement and evaluate newly designed many-core cache memory systems.

While the Pin Tools itself is not typically employed to directly implement cache memory simulation, it plays a crucial role when used in conjunction with cache simulation tools or frameworks. Its primary function is to collect trace files representing memory access patterns generated by real applications or workloads. These trace files are subsequently input into cache simulators for thorough evaluation and analysis of cache performance. Pin Tools is especially vital for many-core cache memory implementation, as it aids in converting selected benchmark programs into trace files compatible with FM-SIM.

3.2.3 Students' Performance and Feedback

As for students' performance, before the introduction of this three-step term project, most teams employing traditional open-source simulators like SimpleScalar [12] or Model2Sim encountered difficulties in completing their designs due to the complexities of these simulation programs. However, since 2020, students have experienced smooth implementations of their low-power cache memory designs using FM-SIM and Pin Tools. This positive change in student outcomes can be attributed to the accessibility and user-friendliness of the newly introduced tools.

Students' feedback on the term project has been overwhelmingly positive, with many expressing enthusiasms for the three-step approach. Some comments include: 1) "Great class and learned a lot! Although the workload with assignments, labs, projects, and presentations was significant. Time was insufficient," 2) "I loved this class. The projects were challenging but helped me gain a deeper understanding of the course material," 3) "The project should have been made available earlier," 4) "Another excellent course! It would be beneficial for the department to offer specialized programs for graduate students.

As a result, we consider the three-step teaching model highly successful, effectively conveying the concept of many-core cache memory systems to all students involved in the project.

4. Conclusion

Given the prevailing technological trends, the growing significance of many-core cache memory design in the industry underscores the rising demand to achieve heightened computational power through parallel processing. However, effectively managing data in cache memories across numerous cores while maintaining cache coherence presents significant challenges. In addition, existing educational approaches may not adequately prepare students for these challenges. In response to such hurdles, we present a comprehensive three-step teaching model designed to furnish students with a clear understanding of engaging in research with well-defined concepts and procedures. This model consists of two laboratory steps, Step 1 and Step 2, along with one project step, Step 3.

Before the implementation of this three-step teaching model, teams using traditional open-source simulators like SimpleScalar or Model2Sim often encountered difficulties in completing their designs due to the intricacies of these simulation programs. However, since 2020, students have reported smoother implementations of their low-power cache memory designs using this model. Feedback from students has been overwhelmingly positive. Consequently, we deem the three-step term project model successful in effectively conveying the concept of many-core cache memory systems to all students involved in the project.

5. Future Work

Despite the notable achievements of the three-step teaching model in many-core cache memory design, there are additional areas for enhancing many-core design. These include: 1) Formulating a cache simulator to create multi-level cache memories, such as L3 or L4; 2) Establishing a user-friendly, web-based cache simulator that allows easy customization of parameters, such as core count, cache levels, and coherence protocols, with a single click; and 3) Assessing power consumption and illustrating thermal effects at the micro-architecture level.

References

[1] David A. Patterson & John L. Hennessy, "Computer organization and design: the hardware/software interface," fourth edition, Morgan-Kaufmann, San Francisco, California, 2012.

[2] Enrique de Lucas, Pedro Marcuello, Joan-Manuel Parcerisa, and Antonio González, "Ultra-Low Power Render-Based Collision Detection for CPU/GPU Systems." In Proc. 48th International Symposium on Microarchitecture (MICRO), 2015.

[3] John L. Hennessy & David A. Patterson, "Computer Architecture: A Quantitative Approach," fifth edition, Morgan-Kaufmann, San Francisco, California, 2012.

[4] Rano Mal, and Yul Chu, "A Flexible Multi-core Functional Cache Simulator (FM-SIM)". In Proceedings of the Summer Simulation Multi-Conference, Article No.: 29, pp 1–12, 2017.

[5] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi and K. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Chicago, IL, USA, 2005.

[6] Sangestani G, Khatiban M, "Comparison of problem-based learning and lecture-based learning in midwifery," Nurse Education Today 33: 791-795, 2013.

[7] Hong JC, Lin CL, Huang HC, "The comparison of problem based learning (PmBL) model and project-based learning (PtBL) model," In International Conference on Engineering Education (ICEE) 2007.

[8] M. Al-Manasia and Z. Chaczko Z, "An Overview of Chip Multi-Processors Simulators Technology," Advances in Intelligent Systems and Computing, vol 366. pp. 877-884, Springer, Cham, 2015.

[9] R. Ubal, et al., "Multi2Sim: A Simulation Framework for CPU-GPU Computing," the 21st IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT), Minneapolis, MN, USA, pp. September 2012.

[10] Standard Performance Evaluation Corporation, "SPEC CPU 2006 [online]," Available: https://www.spec.org/cpu2006.

[11] C. Bienia, S. Kumar, and K. Li, "PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors," In Proceedings of IISWC 2008, pages 47–56, Sept. 2008.

[12] D. Burger, and T. Austin, "The SimpleScalar Tool Set, Version 2.0," University of Wisconsin-Madison Computer Sciences Department Technical Report #1242, June 1997.

How to cite this article:
Chu Yul "et al." (2023) 'A THREE-STEP TEACHING MODEL FOR MANY-CORE CACHE MEMORY DESIGN, *International Multidisciplinary Research Journal*, Volume:1; December 2023; Page 22-28. DOI: https://doi.org/10.47722/imrj.2001.18