Cache Coherency Mechanisms in RISC-V Multicore Architectures

Cache Coherency Mechanisms in RISC-V Multicore Architectures

By

Kim Ho Yeap and Wei Kun Tan

Cambridge
Scholars
Publishing

Cache Coherency Mechanisms in RISC-V Multicore Architectures

By Kim Ho Yeap and Wei Kun Tan

This book first published 2025

Cambridge Scholars Publishing

Lady Stephenson Library, Newcastle upon Tyne, NE6 2PA, UK

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN: 978-1-0364-4480-8
ISBN (Ebook): 978-1-0364-4481-5

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

In multicore systems, cache coherency is a dynamic and multifaceted area that plays a significant role in ensuring data consistency in parallel computing environments. In this book, a comprehensive exploration is performed to explore the complexities of cache coherency, offering detailed insights into each of the snoop-based protocol and directory-based protocol. Under these two categories, several specific protocols such as MSI, MESI, MOSI, MOESI, Write-Once protocols, etc. are studied, which will uncover each of their advantages and potential trade-offs, considering various factors in real-world implementation. These factors focus on the delicate balance needed to optimise scalability, system performance, and energy efficiency.

In the Instruction Set Architecture (ISA) domain, Reduced Instruction Set Computer -Five (RISC-V) architecture is chosen due to its open-source nature, scalability, efficiency, performance, extensible and modular design.

The project in this book aims to design and develop a scalable cache coherency fabric in a RISC-V multicores system. The cache coherency fabric is a module in the design utilised to maintain the caches' memory consistency across all the systems' multiple cores. The cache coherency fabric model is built up from several modules, specifically the First-In-Frist-Out (FIFO) buffer, Directory, and MESI controller, to simulate cache requests from two of the RISC-V cores. Several test cases are planned to cover all the possible scenarios to validate the system's functionality. Testbenches are then simulated to generate waveforms and results. The fabric and testbench are designed using the System Verilog Hardware Description Language (HDL) and simulated using ModelSim software.

To summarise, the investigation performed in this book consolidates an extensive amount of helpful information about the RISC-V cache coherency multicore systems, which contributes to the current discussion of the effective application of RISC-V multicore systems.

DISCLAIMER

In the course of composing the content of this book, the authors made use of ChatGPT to enhance its quality, clarity, grammar, and overall language proficiency. After utilizing this tool/service, the authors carefully examined and edited the content to ensure its logical coherence and appropriateness.

CHAPTER 1

INTRODUCTION

1.1 Overview

In recent years, the rapid advancement and widespread adoption of artificial intelligence (AI) technologies have significantly increased demand within the server industry. As AI continues to push the boundaries of what is possible, data centers are compelled to acquire and deploy vast numbers of data processing units to handle the burgeoning computational requirements. This growing need has driven industry players to invest heavily in innovative solutions within the domain of multicore processors, where the ability to perform high-speed load and store operations is critical for efficiently processing large datasets.

However, a considerable challenge lies in the prevalent use of x86 architecture, a Complex Instruction Set Computing (CISC) architecture first introduced in 1978, which remains dominant in the design of data processing units provided by industry players. The CISC architecture's inherent complexities in pipeline design necessitate a higher transistor count within the dielet, leading to reduced power efficiency when compared to processors based on the Reduced Instruction Set Computing (RISC) architecture, such as RISC-V. This inefficiency becomes particularly pronounced in the context of modern computing, where power consumption and efficiency are paramount.

Moore's Law, which predicts the doubling of transistors on integrated circuits approximately every two years, has held true over the decades, resulting in modern microprocessors boasting more than ten billion transistors (Yeap, Isa, and Loh, 2020). This exponential increase in transistor count has driven significant gains in computational power, particularly in personal computers, where the number of compute cores has steadily risen, allowing users to run multiple applications simultaneously. The benefits of multicore designs extend beyond personal computers, with smartphones and other devices also embracing these architectures to achieve enhanced performance and power efficiency, particularly through the adoption of RISC-V architecture.

As multicore systems become increasingly essential for achieving higher computing performance, scalability, and efficiency, they introduce new challenges, particularly in maintaining data consistency across multiple cores. This underscores the critical importance of effective cache coherency mechanisms in multicore environments. Cache coherency is vital for ensuring the integrity of shared data and enabling seamless collaboration among cores, even as the number of cores in a system continues to grow.

In this landscape, understanding the RISC-V architecture is crucial, as it offers a unique and modular design philosophy that is well-suited to the demands of modern multicore systems. RISC-V's open architecture has garnered significant attention across various computing domains, making it an important area of study for those interested in the intricacies of cache management in multicore processors. This chapter will provide a comprehensive overview of the key features, instruction set characteristics, and architectural components of RISC-V that influence cache management, setting the stage for an in-depth exploration of cache coherency within RISC-V multicore systems.

1.2 Motivational Background

The increasing popularity of multi-core systems has resulted in a wide range of product options for both individual consumers and the industrial market. The advancement of Artificial Intelligence (AI) and Large Language models has created a higher demand for compute workstations capable of processing big data. Even consumer computing devices now come equipped with Neural Processing Units, allowing users to experience artificial intelligence firsthand. However, the need for faster load and store operations to handle AI workloads requires an upgrade in cache components to match the data processing capabilities.

Scalability is crucial in system development and manufacturing environments as it allows for flexibility and reduces the time required for similar projects or the development of different product variations within the same project. Power consumption is another important factor that influences user experience and choices. Complex Instruction Set Architecture (CISC) processors offer greater processing power but consume more power. On the other hand, Reduced Instruction Set Architecture (RISC) processors provide a more energy-efficient alternative with slightly lower computing performance.

In this context, a scalable cache coherency fabric on a Multi-RISC V core system emerges as a solution to enhance data processing in multicore systems. This fabric ensures cache coherency between multiple RISC V cores, addressing the need for efficient communication and management of cache coherency.

The ever-evolving field of computer architecture necessitates meeting the increasing demands for improved performance, scalability, and efficiency, which has led to the widespread adoption of multicore systems. The AI domain, in particular, has experienced significant growth, resulting in a higher demand for AI applications. As a result, the server industry is actively seeking innovative solutions to accommodate the expanding needs of AI development.

The historical backdrop highlights the prevalent utilization of x86 architecture in data processing units, particularly in personal computers, and the challenges presented by the CISC architecture. This historical context establishes the framework for comprehending the driving force behind the development of computing architectures. Embedded within this progression is Moore's Law, which guides the doubling of transistors in integrated circuits every two years. As a result, contemporary microprocessors now boast an approximate transistor count of 82 billion, showcasing the relentless pace of technological advancement.

The proliferation of multicore systems has emerged as a significant response to the pursuit of enhanced computing performance and scalability. By executing tasks in parallel across multiple cores, these systems promise increased computational power, thereby providing a means to meet the demands of modern applications, including those involving AI workloads. However, this shift introduces a fundamental challenge: the need for effective cache coherency mechanisms.

Within the paradigm of multicore systems, each core maintaining its cache introduces the potential for inconsistencies when accessing shared data. The juxtaposition of parallelism and data consistency underscores the critical role of cache coherency in reconciling conflicting requirements. Effective cache coherency mechanisms are pivotal in ensuring coherent and reliable execution of parallelized tasks, addressing the complexities introduced by the inherent parallel nature of multicore architectures.

To delve into the intricacies of cache coherency within a multicore environment, it becomes imperative to have a foundational understanding of the RISC-V architecture. RISC-V, an open-source and standardized instruction set architecture, stands out for its open and modular design philosophy, gaining prominence across diverse computing domains. The architecture's modular nature allows for flexibility in design, making it an appealing choice for various applications, including those that demand efficient cache management in multicore systems.

Understanding the fundamental aspects of RISC-V, such as its key features, instruction set characteristics, and architectural components, is crucial in order to delve into the intricacies of cache coherency. The open nature of RISC-V, which is not bound by proprietary constraints, promotes innovation and adaptability, making it an attractive option for addressing contemporary computing challenges.

This comprehensive understanding of RISC-V serves as a backdrop for comprehending the evolution of computing architectures, the increasing demands for data processing driven by artificial intelligence, the impact of Moore's Law on transistor count, the emergence of multicore systems, and the significant role played by cache coherency within this complex landscape. Subsequent sections of this chapter will build upon this foundation, exploring cache coherency protocols, architectural considerations, real-world implementations, and the broader implications for system performance, energy consumption, and security aspects within RISC-V multicore systems.

1.3 Significance of Cache Coherency in RISC-V Systems

In the current computing landscape, the application of multicore systems has emerged to be an essential part of supporting scalability and improving computing performance for data processing and power efficiency. However, using multicore systems poses challenges, such as ensuring data consistency across multiple cores in a system. To overcome the challenges, an effective cache coherency mechanism is required to communicate and manage the data. This book will comprehensively explore the foundation and working principle of cache coherency in the RISC-V multicore systems.

In the domain of multicore processors, cache coherency plays an essential role in maintaining the integrity of shared data and ensuring seamless data transmission between the cores. The respective caches attached to each core introduce several challenges in ensuring data consistency, even though parallelism can enhance computational power. A foundational knowledge of the RISC-V architecture is required to understand cache coherency in a multicore environment. RISC-V has garnered popularity in various computing sectors due to its open and modular design architecture. This book explores the RISC-V architectural components, essential features, and characteristics of the instruction set that would affect cache management, laying the groundwork for a detailed understanding of a multicore system.

Subsequently, a structured examination of cache coherency protocols and architectural considerations will be performed to discuss the complexities and implementation of cache coherency in RISC-V multicore systems. This would provide detailed information on the trade-offs, design choices, and impact of cache coherency on the efficiency of the system performance. Multiple cache coherency protocols such as MSI, MESI, MOSI, MOESI, and Write-Once protocols are studied to gain insight into trade-offs, advantages, and challenges. Eventually, this affects the responsiveness and efficiency of data access operation in a RISC-V multicore system. Furthermore, the thesis will analyse the interconnections between cache coherency and parallelism, discussing how these components improve performance in multicore systems. Hence, several strategies and approaches that leverage the strength of RISC-V multicore architectures are implemented. This will serve as a helpful reference for industry players, university researchers, and system developers who want to understand the complex RISC-V multicore cache coherency system.

1.4 Problem Statement

The recent rise in popularity of multicore systems contributed to the introduction of increased product choices in both the customer and industrial markets. Workstation processors capable of computing large amounts of data are commonly used for AI models and Large Language Models (LLMs). These models require significant computational power to perform model training that utilises datasets that could be up to hundreds of thousands of data. Recently, in the year 2024, Samsung introduced "Galaxy AI," which uses generative AI technology on a consumer-grade product. This indicates a trend for more computational power in future products.

However, a notable challenge arises when the devices require high computing power to support high-speed load and store operations. A single-core processor might need more computing power, and the power efficiency of a CICS-based processor is poor due to the pipeline complexity and higher number of transistors required. Hence, an innovative solution is needed to enhance the capability of cache components so that they will match the computing power required for AI workloads. The innovation of cache coherency in multicore systems can significantly improve the computing power of a multicore system and enhance load balance to spread out the workloads evenly to all the cores. This would significantly improve the system's performance by reducing the time taken to process data.

The next challenge is that the multicore system needs to support the scalability of multiple RISC-V cores while maintaining the cache coherency throughout all the scaled cores. Scalability is important because it allows the system to adjust the number of cores during the development stage flexibly. Hence, the project development time and cost can be reduced. Furthermore, power consumption is another challenge that would affect the user experience and workload of the device. As the number of cores increases, the device's power consumption significantly increases.

On the other hand, developing a scalable cache coherency fabric on a RISC-V multicore system will face several challenges. The first challenge is to design the interconnection between the cores that supports parallelism for the data communication of the cache request operation. Next, a cache coherency fabric, including a FIFO buffer module, a Directory module, and a cache coherency protocol module, needs to be designed. After that, multiple test cases need to be planned to cover all the possible scenarios the system will hit, verifying the design functionality.

1.5 Aims and Objectives

The project presented in this book focuses on the design and implementation of a sophisticated cache coherency fabric within a RISC-V multicore system, leveraging the capabilities of the System Verilog hardware description language (HDL). The cache coherency fabric plays a critical role in maintaining data consistency across multiple caches in the system by ensuring that any shared data remains uniform and up-to-date, even when accessed simultaneously by different cores.

To achieve this, a series of modules are meticulously developed and integrated into the system architecture. The process begins with the creation of a FIFO (First-In, First-Out) Buffer module, which is essential for managing the orderly storage and retrieval of input data. This module ensures that data flows smoothly through the system without conflicts or loss, maintaining the integrity of the data throughout the process.

Following this, a Directory module is designed to serve as a centralized index, which efficiently maps memory addresses to the corresponding cache entries. This module is crucial for quick and accurate data retrieval, as it directs requests to the correct locations within the system, thereby optimizing the performance and speed of cache operations.

To maintain coherence across the entire multicore system, a MESI (Modified, Exclusive, Shared, Invalid) controller is deployed. This controller implements the widely adopted MESI protocol, which is fundamental for managing the states of cached data. By tracking and controlling the state of each cache line, the MESI controller ensures that any changes made in one cache are appropriately propagated or invalidated across other caches, thus preventing data inconsistencies and enhancing overall system reliability.

In conjunction with these modules, a comprehensive testbench is developed to rigorously verify the functionality of the cache coherency fabric. The testbench simulates a variety of scenarios and edge cases to ensure that the system behaves as expected under different conditions, validating the correctness, efficiency, and robustness of the design.

Through these carefully crafted components and verification processes, the project aims to deliver a reliable and scalable cache coherency fabric that meets the demands of modern multicore processing environments.

CHAPTER 2

CACHE COHERENCY IN MULTICORE SYSTEMS

2.1 Memory Hierarchy in Multicore Systems

The emergence of various pioneering designs and proposals in multicore frameworks has contributed to the advancement of RISC-V multicore systems. These proposals address critical components such as cache coherency and system interconnection. Balkin et al. (2016), Petrisko et al. (2020), and Emil et al. (2022) provide distinct insight into memory hierarchy optimization, which includes the difficulties and solutions of cache coherency in a multicore system.

Balkin et al. (2016) presented an open-source 64-bit RISC-V multicore system called Open Piton that exhibits the Ariane RISC-V core with a write-through L1.5 cache. To ensure that the communication in the system is efficient, Balkin's design uses a creative way to sync the cores with a 2D mesh cache coherency method. This method effectively handles memory consistency and cache coherency by using spatial coordinates and chip identifications to improve the capabilities of data processing. Additionally, the proposed solution is highly adaptable and flexible due to its modular structure of the design. This design helps RISC-V developer to instantiate multicore systems that can be custom-made to their needs, and it can support the number of cores of up to 64 cores. The flexibility of the Open Piton framework allows scalability and supports different research requirements, which helps to explore a variety of computing scenarios in the domain of multicore systems. The modular structure of the Open Piton design is a beneficial resource for researchers to experiment with their innovative design proposals, which could boast more different developments of multicore systems in the future that could push the limit of the architecture. Figure 2.1 shows the datapath of the Open Piton memory hierarchy. The cache hierarchy of the OpenPiton framework consists of L1, L1.5, and L2 cache. The L1 cache uses very high bandwidth to write to the subsequent level of the cache. This increases the chances of congestion in the multicore system interface. Also, the compliance of the cache communication interface between the L1 and L2 cache needs to be considered since both caches use different interface protocols. Therefore, Balkin proposed an additional L1.5 cache to handle these two issues. The L2 cache is a shared distributed write-back cache for all tiles in the system.

Petrisko et al. (2020) introduced a Linux-capable, open-source 64-bit RISC-V multicore system called as BlackParrot. To optimise the performance and enhance the functionality of BlackParrot, Petrisko's design emphasised the cache coherence and the accelerator chips. This design is built up of core tiles, L2 extension tiles, and coherent and streaming accelerator tiles. Figure 2.2 shows the BlackParrot multicore system. Core tiles act as the backbone of the design that implements the processor, directory, and the L2 cache. The L2 extension tiles improve the capabilities of the cache in the system by allowing how much the L2 cache can be modified. Furthermore, the coherent accelerator tiles are used to improve the system and strengthen the parallel processing capacity by speeding up access to the cache coherent memory system. The streaming accelerator or the input-output (I/O) tiles help to interface the memory system flexibly through a common non-L1 cache. The combination of these four tiles in the BlackParrot design significantly improves the efficiency and performance. BlackParrot stands out as a practical and robust design that is suitable for real-world applications. The tape-out of GlobalFoundaries's 12 nm FinFET has proven that this design is a reliable and industry-ready solution. In short, BlackParrot is the product of innovation and practice that provides a versatile and proven open-source solution for developers looking to leverage the advantages of RISC-V multicore systems alongside Linux operating systems. Figure 2.2 shows the BlackParrot multicore system designed in the Petrisko proposal.

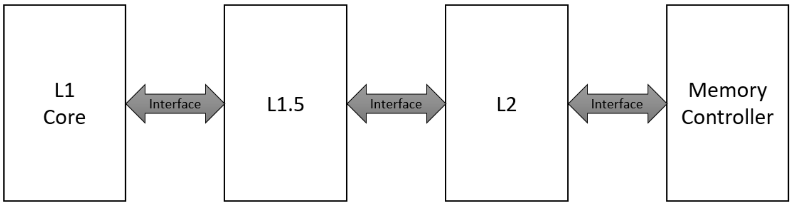

Fig. 2-1 The Datapath of the Open Piton Memory Hierarchy

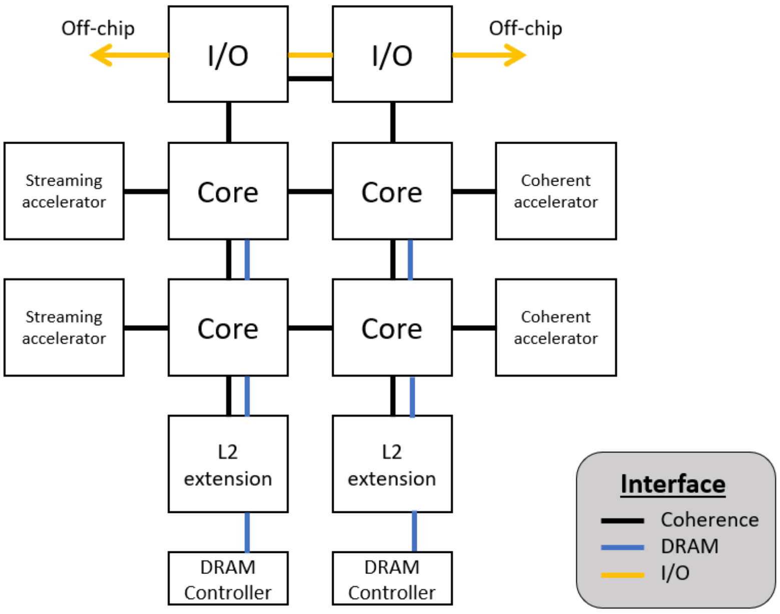

Fig. 2-2 BlackParrot Multicore System

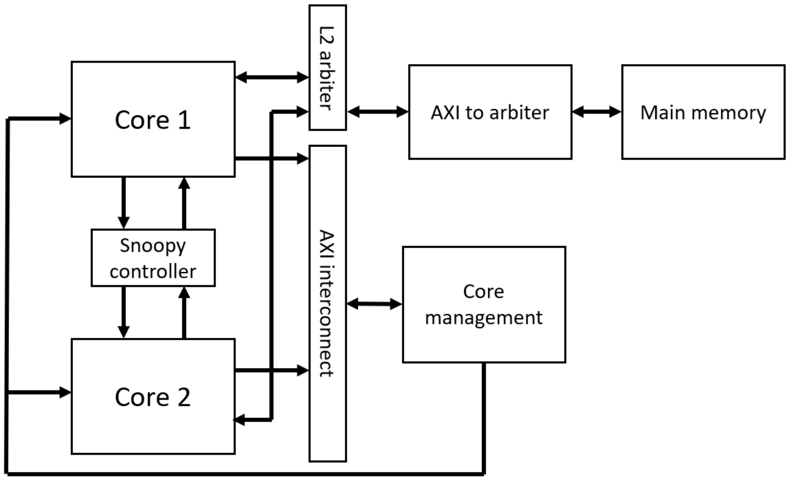

Fig. 2-3 Top Design of the Dual-Taiga Core

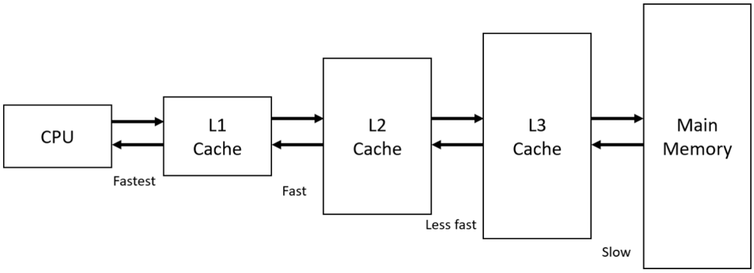

Fig. 2-4 Block Diagram of the Memory Hierarchy

More recently, Emil et al. (2022) put forward a design that uses an open-source single RvCore processor, also known as Taiga. This type of multicore RISC-V processor has some differences in the modular interconnection that restricts the maximum supported number of cores of up to two. By taking into account the cache coherency, memory synchronization, and interface interconnection of the design, Emil incorporated two Taiga cores based on the snoopy cache coherence architecture. The main memory and caches of the proposed design are intended to have enhanced data coherency. The synchronisation between all the working cores is accomplished by a hardware customised peripheral unit that was developed in this design. In term of bus interface protocol, an on-chip Advanced eXtensible Interface (AXI) was used on the dual-port of the main memory. This helps to increase the reliability and performance of bus communication. Furthermore, Emil's design integrated a core management system to have extensive control over the activation and deactivation of the core in the system. Figure 2.3 shows the top design of the dual-Taiga core. L2_arbiter is used to store all requested operations from the core and transmit the operation to the main memory. AXI-to-arbiter acts as a converter block in the system to transform the request operation interface of the L2 arbiter to an AXI standard interface and vice versa.

In the multicore system domain, optimising memory hierarchy is an extremely significant topic because it ensures that data access is efficient, latency is less, and coherency between multiple caches remains consistent in the system. The memory hierarchy can be broken down into various cache levels, each performing its own specific task to ensure data access is fast and efficient.

By referring to the memory hierarchy block diagram in Figure 2.4, L1 Cache is the first level cache in the hierarchy that is placed in closest proximity to the central processing unit (CPU). It is the smallest in memory size but has the fastest read and write speed. L1 cache operates at high speed and is able to facilitate rapid access to frequently utilised instructions and data. This helps to significantly reduce the access time to the instruction and data and, in turn, acts as a catalyst for the high-speed execution of computational tasks. The next level of the cache hierarchy is the L2 Cache. It is located right outside of the CPU, having a memory specification that is balanced between the latency and memory size. At the L2 cache level, it enhances data access processes by providing a larger storage size to often utilised data so that the efficiency of the system will be optimised. At the same time, this hierarchy of memory is able to handle the trade-off between memory access speed and storage size. The last level of the cache hierarchy is the L3 Cache. L3 Cache functions as a shared resource accessible to all the computing cores. The introduction of this level of cache requires higher latency to access the memory but, at the same time, addresses the essential requirement for common data access. The design of the L3 cache ensures the balance between memory access speed and the increase of inevitable latency attributed to shared resources.

Besides storing data in different cache hierarchy levels, as discussed earlier, memory hierarchy also involves the organisation of how the data is accessed. The advantages of using cache hierarchy in data organisation are to manage complex cache coherency in multicore systems and to minimise data transmission delay, leading to an efficient system. As technology continues to advance, new technology, such as fine-tuning, can be used on the memory hierarchy in multicore systems to enhance performance and reduce the latency of data transmission. In multicore systems, cache coherency plays a vital role in ensuring that the data across all the caches in the system is consistent. When data in one of the caches get modified, other caches are able to synchronise the changes rapidly. This would significantly reduce the possibility of data discrepancies in multicore systems through seamless core communication and precise data synchronisation. The cache coherency process is used to ensure that the system is able to manage challenges emerging from multicore design, fostering efficient data transmission among interconnected cores and improving the performance of the overall system.

In the domain of RISC-V multicore systems, a feasible solution to effectively manage the cache coherency between numerous RISC-V cores is to adopt a scalable cache coherency fabric. In addition to addressing the interface connection of all the cores and managing cache coherency in the system, this fabric also validates functional correctness, consequently contributing to the system's overall robustness. This method helps improve the reliability of the system, thus fortifying its capacity to handle different types of tasks and maintain the stability of the system's operation. The complexities associated with cache coherency and memory hierarchy mechanisms are pivotal to determining the scalability and performance of the multicore design. A fine-tuned memory hierarchy, together with efficient cache coherency, will significantly boost the performance of the system. This can be observed from the accelerated data access, lower latency, and better overall operational efficiency.

The efficient management of cache coherency also contributed to the scalability of the system, especially when the system requires the synchronisation of the cache for a large number of cores. When the design becomes more complex, such as in a situation where a large number of cores require the cache to be coherent, the system can implement a scalable cache coherency fabric to effectively manage this large and complex design to perform seamless operation, reduced latency, and enhanced performance even when the design faces expanding computational demands. In short, this technique reinforces the foundation of the system by enabling it to evolve in the current dynamic and very demanding computing conditions.

2.2 Cache Coherency Protocols

Cache coherency is a fundamental aspect in the ever-evolving domain of multicore systems within computer architecture. As computing systems increasingly adopt multicore designs to enhance performance and scalability, the importance of maintaining data consistency across multiple caches becomes more pronounced. In multicore environments, each core typically has its own cache, which temporarily stores copies of data from the main memory to reduce latency and improve processing speed. However, this setup introduces the potential for incoherent data—a situation where different caches hold different versions of the same memory block, leading to inconsistencies and errors in data processing.

The issue of incoherent data arises when multiple cores (referred to as clients) maintain their own caches of a shared memory resource. For example, consider a scenario where two clients have cached instances of a specific memory block after reading from the shared memory. If one client subsequently modifies this memory block, the other client may continue to hold an outdated copy of that data in its cache. This divergence between the cached data and the actual data in memory can result in incorrect program behavior, as the second client may operate on stale information.

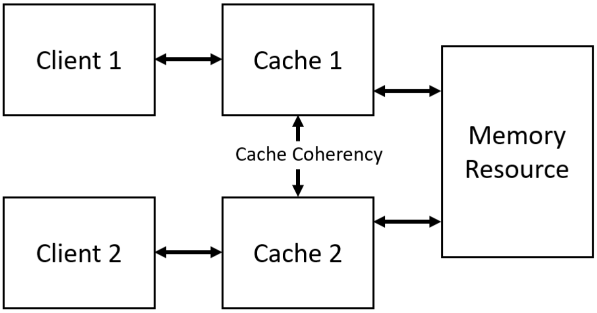

Fig. 2-5 Block Diagram of Two Caches Sharing a Resource

Figure 2.5 illustrates this problem with a block diagram showing multiple caches sharing a single memory resource. In the diagram, if the upper client modifies a memory block that both it and the lower client have cached, the lower client will continue to reference its old, now incorrect, version of the data unless a mechanism is in place to update or invalidate this stale cache entry.

This is where cache coherency protocols come into play. These protocols are essential for managing and synchronizing data across multiple caches, ensuring that all clients see a consistent view of the shared memory. By enforcing rules and mechanisms that detect and resolve discrepancies between caches, these protocols maintain the integrity of the system's operations.

2.2.1 Snoop-Based Protocols

Snoop-based protocols represent a class of cache coherence mechanisms utilised in multicore systems to manage shared data coherence across multiple caches. This methodology propagates data on the state changes of a particular cache to all other caches in the system using a broadcasting mechanism. A cache will broadcast a "snoop" signal to all other caches in the system when a data block gets modified, facilitating the synchronisation of their own copies. This would ensure that all caches receive a signal indicating a state modification via the broadcasting-based method.

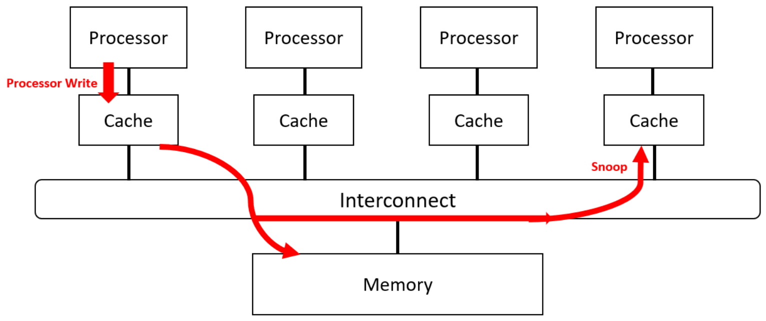

Fig. 2-6 The Structure of Snoop-Based Protocols

The working principle of snoop-based protocols is to ensure coherent data across all caches in the system, avoiding data incoherency when multiple caches store the same block of data. A cache will initiate a process to cross-check the state of its block of data when there is a write or read operation on a specific data block. Concurrently, it sends a "snoop" signal toward any other caches that exist in the system to cross-check the state of their copies. Through the snooping mechanism, if it detects another cache holding a copy of the data in an altered state or modified state, then it shows that there are changes to the data stored. Hence, a corrective response needs to be undertaken on the cache receiving this snoop to manage data consistency. Several corrective responses might happen depending on the specific protocol to be used in the system, which will be discussed in the subsequent part of this topic. Some corrective responses include invalidating the cache existing data copy and updating the cache with the modified version of data or implementing other alternative measures to synchronise the cache's state with the modified data. The structure of the Snoop-Based protocols is shown in Figure 2.6.

The benefits of snoop-based protocols are simple to implement and require low latency to broadcast the changes in the cache's state. The simple working principle of this protocol is well-suited for small multicore systems, where the advantages in real-time coordination within caches exceed the disadvantage of high traffic volume caused by a high frequency of snoop broadcasts to all the caches within the system in small multicore systems. In contrast, snoop-based protocols could potentially have a high volume of bus traffic in large multicore systems. All the caches in the large system require to be notified of any changes in this broadcasting working principle of snoop-based protocols, causing communication bottlenecks. In short, the specification and workload demand of the multicore system design can directly influence the efficiency of snoop-based protocols.

2.2.2 Directory-Based Protocols

Directory-based protocols represent another common class of cache coherence mechanisms used to manage data consistency in multicore systems. This methodology employs a centralized directory to monitor and control the data blocks across the shared caches within the system. The directory maintains information about the location of each cache and the state of each data block.

Within the directory, each block of shared data can be registered in various states. Commonly defined states include shared, modified, and invalid. A shared state indicates that the data block exists in more than one cache, with all copies holding consistent data. A modified state means that the data block exists in a cache and has been altered. An invalid state signifies that the data block is either absent from any cache in the system or present but marked as invalid. To read or write a specific data block, a cache must initiate communication with the centralized directory. The directory provides information regarding the status of the requested data block and its presence in other caches. When the directory receives a write operation, it limits the number of caches permitted to modify the data at a given time. If multiple caches hold copies of the data, the directory must ensure data coherence is maintained. Possible actions include invalidating the data copy in other caches or updating it with the latest data, depending on the protocol in use. If the data block is in a modified state in another cache, the directory must coordinate necessary measures to ensure data consistency across all caches.

The benefit of directory-based protocols lies in their potential to reduce bus traffic compared to snoop-based protocols. The centralized directory acts as a reference point for coordination, eliminating the need for caches to broadcast data changes to every cache in the system. This centralized control architecture is advantageous in systems with multiple caches. However, directory-based protocols have disadvantages, including longer latency for read/write operations, as each operation requires access to the central directory to retrieve data. The directory provides the exact locations of all cached copies of shared data blocks. The structure of directory-based protocols is shown in Figure 2.7.

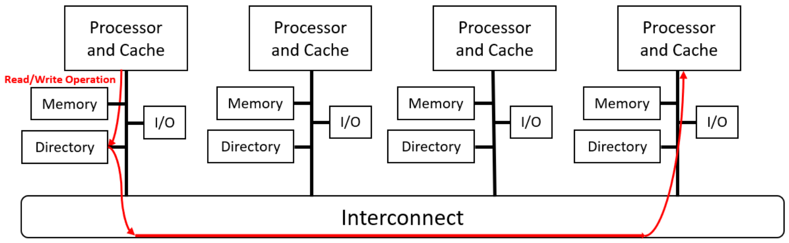

Fig. 2-7 The Structure of Directory-Based Protocols

2.3 Selection Criteria for Protocols in RISC-V Systems

Various factors affect the decision to select between snoop-based protocols and directory-based protocols for RISC-V multicore systems, as this choice directly impacts the performance, behavior, and efficiency of the system. These factors include system size, latency specifications, acceptable levels of bus traffic, and the complexity of implementation.

Firstly, the size of the multicore system is a crucial factor. Snoop-based protocols are well-suited for systems with a smaller number of caches and simpler designs. The direct coordination within a smaller cache ensemble is facilitated by the simplicity of snoop-based protocols. Conversely, directory-based protocols offer greater scalability, making them suitable for larger RISC-V multicore systems with a higher number of caches. The centralized directory can effectively manage the cache coherence across a large number of caches.

Next, latency specifications play a significant role. Snoop-based protocols employ a real-time, reactive methodology that generally requires lower latency to synchronize data across multiple caches. Therefore, they are preferred for performance-intensive applications where rapid data updates are essential. On the other hand, directory-based protocols, while supporting a larger number of caches, may exhibit higher latency. This higher latency can be acceptable in larger RISC-V multicore systems, where the advantages of scalability outweigh the drawbacks of increased latency.

The effect on bus traffic is another critical factor. Snoop-based protocols use a broadcasting method to relay cache state changes to all other caches in the system, resulting in increased bus traffic. While this may be manageable in smaller multicore systems, it can significantly strain bus utilization and cause congestion in larger systems. In contrast, directory-based protocols offer better control over bus traffic. By managing communication through a centralized directory, unnecessary broadcasts are minimized, which is advantageous as the number of caches increases in larger multicore systems.

Moreover, the complexity of implementation and management is an important consideration. Snoop-based protocols typically involve simpler implementation, making them more accessible for smaller multicore systems. The decentralized framework of snoop-based protocols simplifies cache coordination. In contrast, directory-based protocols require managing data coordination through a centralized directory, resulting in higher implementation complexity. However, this complexity is often justified in larger multicore systems, where the benefits of scalability and centralized control outweigh the challenges of implementation.

In conclusion, choosing the most suitable cache coherence protocol between snoop-based and directory-based methods depends on factors such as system size, latency specifications, acceptable bus traffic levels, and implementation complexity. The specific characteristics of the RISC-V multicore system, along with its intended application and workload demands, play a pivotal role in guiding the selection of the appropriate cache coherence protocol.

2.4 Specific Protocols and Their Implementations

The most fundamental function in multicore systems is cache coherence. Cache coherence ensures that shared data remains consistent across all caches in the system. With the increasing popularity of the RISC-V architecture in multicore systems, several factors must be considered to implement cache coherence protocols effectively. As previously discussed, snoop-based protocols and directory-based protocols represent broad categories that explain the principles of achieving coherence at the system level. Therefore, choosing the most suitable protocol involves more than just deciding between snoop-based and directory-based approaches. It is essential to evaluate factors such as scalability, system performance, and energy consumption in the context of specific cache coherence protocols within these two categories.

In the subsequent sections, we will explore specific cache coherence protocols in greater detail. These protocols include MSI, MESI, MOSI, MOESI, and Write Once. By examining these protocols, we will gain valuable insights into the rapidly evolving technology of cache coherence within RISC-V multicore systems. Each protocol has distinct advantages and trade-offs, which ultimately influence the reliability and efficiency of the multicore system.