Panamerican Mathematical Journal
ISSN: 1064-9735
Vol 35 No. 1s (2025)

Cache Coherence Protocols in Multiprocessor Systems: A Survey and Comparative Analysis

Mukesh Kashyap[1], Sanjay Kumar[2], Swati Jain[3]

[1]Research Scholar, S.o.S. in Computer Science & IT, Pt. Ravishankar Shukla University, Raipur, Chhattisgarh, India,

[2] Professor, S.o.S. in Computer Science & IT, Pt. Ravishankar Shukla University, Raipur, Chhattisgarh, India

[3] Assistant Professor, Govt. J. Yoganandam Chhattisgarh College, Raipur, Chhattisgarh, India,

Article History:
Received- *23-07-2025*
Revised- *03 -09- 2025*
Accepted- *16-09- 2025*

Abstract - Cache coherence is a fundamental requirement in shared memory multiprocessor systems where multiple processors access and modify shared data concurrently. With the increasing adoption of multicore architecture, efficient coherence mechanisms have become essential to ensure data consistency and high system performance. This paper presents a survey and comparative analysis of widely used cache coherence protocols in multiprocessor systems. The study reviews the fundamental concepts of cache coherence and categorizes coherence mechanisms based on their architectural approaches. Classical protocols including MSI, MESI, MOESI, and MESIF are examined with respect to their state models, operational mechanisms, and performance characteristics. A comparative analysis is presented to highlight the differences among these protocols in terms of scalability, communication overhead, and implementation complexity. In addition, the paper discusses key design challenges such as coherence traffic, latency, and energy consumption, and reviews emerging research directions including adaptive coherence techniques, machine-learning-assisted optimization, and coherence support for heterogeneous computing systems. The insights provided in this survey help researchers and system designers understand the strengths and limitations of existing coherence mechanisms and support the development of scalable and efficient coherence solutions for future multiprocessor architectures.

Keywords: Multiprocessor Systems; Multicore Architecture; Coherence Protocols; Shared Memory Systems; Cache Consistency.

1 INTRODUCTION

The rapid growth of multicore and multiprocessor architectures has significantly improved the computational capabilities of modern computing systems. These architectures integrate multiple processing cores on a single chip, enabling parallel execution of applications and improving system throughput. Most modern multicore systems adopt a shared memory programming model in which multiple processors communicate through a common memory space. Although this model simplifies programming, it introduces a critical challenge known as the cache coherence problem, where multiple processors may simultaneously access and modify shared data stored in their private caches. Ensuring that all processors observe a consistent view of shared data is

essential for maintaining program correctness and reliable system operation [1-3].

To reduce the latency gap between the processor and main memory, modern computer architectures employ cache memories that store frequently accessed data close to the processor. While caches significantly improve system performance, they also introduce data inconsistency issues when multiple processors maintain copies of the same memory block in their local caches. When one processor modifies a data block, the other cached copies may become stale, resulting in incorrect program behaviour. Cache coherence protocols are therefore required to maintain consistency among the multiple cached copies of shared data and ensure that all processors observe the most recent value of a memory location [4-6].

Cache coherence protocols define a set of rules that coordinate communication among cache controllers and main memory to maintain consistency. These protocols manage the states of cache blocks and control how data is shared, updated, or invalidated across processors. Early coherence mechanisms primarily relied on snooping-based protocols, in which all caches monitor a shared communication bus and observe memory transactions to maintain coherence. Snooping protocols are efficient in small scale multiprocessor systems because they allow caches to react quickly to memory operations. However, the broadcast nature of snooping communication leads to scalability limitations as the number of processors increases, resulting in increased coherence traffic and bus contention [7-9].

To address these scalability challenges, directory-based coherence protocols were introduced. In directory- based systems, a centralized or distributed directory maintains information about which processors currently hold copies of each memory block. Instead of broadcasting coherence messages to all processors, communication occurs only between the processors involved in a memory transaction. This approach significantly reduces unnecessary coherence traffic and improves scalability for large scale multiprocessor systems [10-11]. Over the years, several widely adopted coherence protocols have been developed, including MSI, MESI, MOESI, and MESIF, which introduce additional states to improve performance and reduce memory access overhead. These protocols optimize cache to cache communication and minimize unnecessary write-backs to main memory, thereby enhancing system efficiency [6], [32].

Despite these advancements, cache coherence continues to present several challenges in modern computing systems. As processor core counts increase and heterogeneous architectures such as CPU-GPU systems become more common, maintaining coherence efficiently becomes increasingly complex. High coherence traffic can lead to increased network congestion, latency, and energy consumption in many-core systems. Consequently, recent research has focused on developing scalable and energy efficient coherence techniques, including adaptive coherence protocols, hybrid snooping-directory approaches, and machine learning based coherence optimization methods [12-15].

Given the increasing complexity of modern multicore systems and the wide variety of coherence mechanisms proposed in the literature, a comprehensive understanding of existing cache

coherence protocols is essential. This paper presents a systematic survey and comparative analysis of cache coherence protocols used in multiprocessor systems. The study reviews classical coherence protocols, analyses their architectural mechanisms, and evaluates their advantages and limitations in terms of scalability, coherence traffic, and implementation complexity. Furthermore, emerging research directions and recent developments in coherence optimization are also discussed to provide insights into future trends in multiprocessor system design.

The remainder of this paper is organized as follows. Section 2 presents the background of cache memory and discusses the cache coherence problem in multiprocessor systems. Section 3 introduces the taxonomy of cache coherence protocols. Section 4 reviews classical coherence protocols including MSI, MESI, MOESI, and MESIF. Section 5 provides a comparative analysis of these protocols. Section 6 discusses the major design challenges in cache coherence mechanisms. Section 7 highlights emerging trends in coherence optimization, while Section 8 outlines research gaps and future directions. Finally, Section 9 concludes the paper.

2 BACKGROUND OF CACHE COHERENCE IN MULTIPROCESSOR SYSTEMS

Multiprocessor and multicore systems rely on hierarchical memory organizations to reduce memory access latency and improve overall system performance. In such systems, each processor core typically contains a private cache that stores frequently accessed data closer to the processor. This hierarchical design exploits spatial and temporal locality of reference, allowing processors to access data much faster than if they relied solely on main memory. However, the presence of multiple private caches introduces challenges related to maintaining a consistent view of shared data among processors. As a result, cache coherence mechanisms are required to ensure correct data sharing in shared-memory multiprocessor architectures [16-17].

2.1 Memory Hierarchy and Cache Memory

Modern computer architectures employ a multilevel memory hierarchy consisting of registers, cache memory, main memory, and secondary storage. Cache memory acts as an intermediate layer between the processor and main memory and is typically implemented using high speed static random access memory (SRAM). The primary objective of cache memory is to reduce the latency of memory accesses by storing frequently used instructions and data close to the processor [17]. The hierarchical organization of memory in a multiprocessor system is illustrated in Fig. 1.

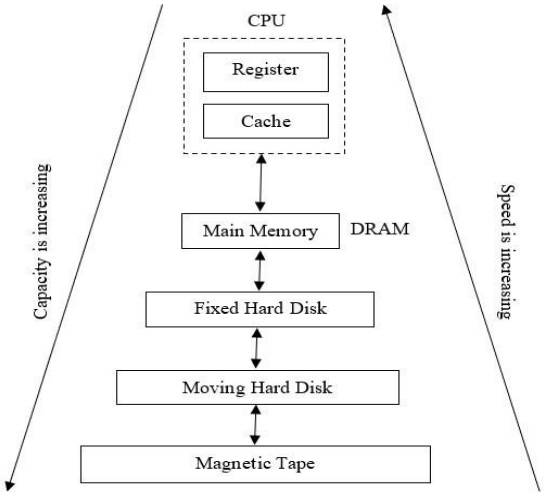

Fig.1. Memory Hierarchy

In contemporary multicore processors, caches are commonly organized into multiple levels such as L1, L2, and sometimes L3 caches. The L1 cache is usually private to each core and provides the fastest access, while the lower- level caches may be shared among multiple cores to facilitate efficient data sharing. When a processor requests data, the system first checks the cache hierarchy before accessing the main memory. If the requested data is found in the cache, the event is referred to as a cache hit, whereas the absence of the data in the cache results in a cache miss, requiring the data to be fetched from lower levels of the memory hierarchy [1]. Although caches significantly enhance performance, they also create potential consistency issues when multiple processors maintain copies of the same memory block.

2.2 Cache Coherence Problem in Multiprocessor Systems

The cache coherence problem occurs when multiple processors store copies of the same memory block in their local caches and one processor modifies its copy. Without a proper coherence mechanism, other processors may continue to read outdated data from their caches, leading to incorrect program behaviour. Therefore, maintaining a consistent view of shared data across all caches is essential for the correctness of parallel programs in shared- memory systems [4].

For example, consider two processors that hold a cached copy of a shared variable in their respective caches. If processor P1 modifies the variable stored in its cache, the corresponding value in the cache of processor P2 may become outdated if no coherence mechanism is applied. As a result, processor P2 may continue to read an obsolete value from its cache while processor P1 operates on the updated value, leading to inconsistent system behaviour. Cache coherence protocols address this issue by coordinating communication among cache controllers and defining rules for updating, invalidating, or sharing cache blocks among processors. These protocols track the state of cache lines and ensure that all processors observe the most recent value of shared data during program execution [2], as illustrated in Fig.2.

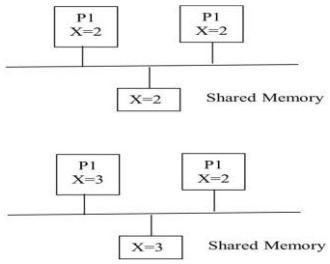

Fig. 2. Example illustrating the cache coherence problem in a multiprocessor system.

2.3 Cache Write Policies

Cache write policies determine how modifications made to cached data are propagated to the main memory, thereby influencing both memory consistency and system performance in multiprocessor architectures. Among the commonly used policies, write-through and write-back represent two fundamental approaches for handling cache updates. In the write-through policy, every write operation performed on a cache block is immediately propagated to the main memory. This ensures that the main memory always contains the most recent value of the data, simplifying consistency management across processors. However, the frequent updates to main memory can significantly increase memory traffic, which may degrade overall system performance, particularly in systems with multiple processors [18].

In contrast, the write-back policy updates only the cache when a processor modifies a data block, while the corresponding main memory location is updated later when the cache block is replaced. The modified block is marked using a dirty bit to indicate that the cache holds a more recent copy of the data than the main memory. Thisapproach reduces memory traffic and improves overall system efficiency, but it also introduces additional complexity because other processors must not access stale data. Therefore, effective cache coherence mechanisms are required to maintain data consistency in systems employing write-back caches, as illustrated in Fig. 3.

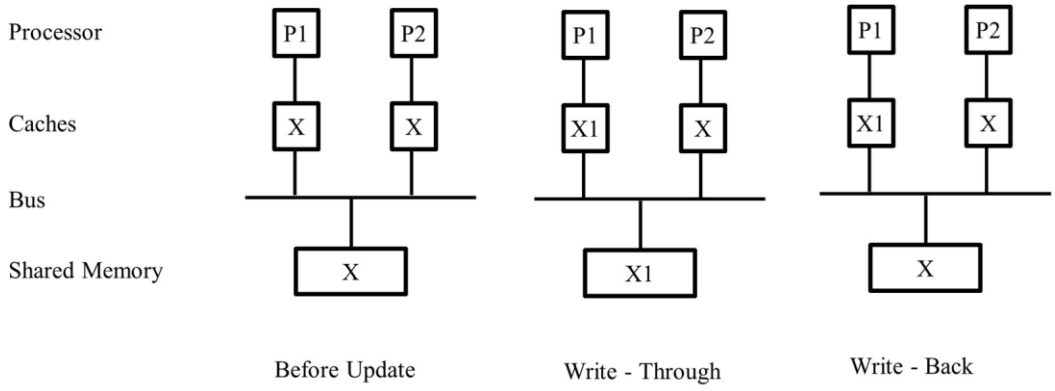

Fig. 3. Illustration of write-through and write-back cache write policies in a multiprocessor system

3 TAXONOMY OF CACHE COHERENCE PROTOCOLS

Cache coherence protocols are designed to maintain consistency among multiple cached copies of shared data in multiprocessor systems. Over the years, several coherence mechanisms have been proposed to manage communication among caches and ensure that processors observe a consistent view of memory. These mechanisms can generally be categorized into snooping-based protocols, directory-based protocols, and hybrid coherence approaches based on how coherence information is tracked and propagated among processors [2], [4]. The classification of these coherence mechanisms is illustrated on Fig. 4. Each approach has different characteristics in terms of scalability, communication overhead, and implementation complexity.

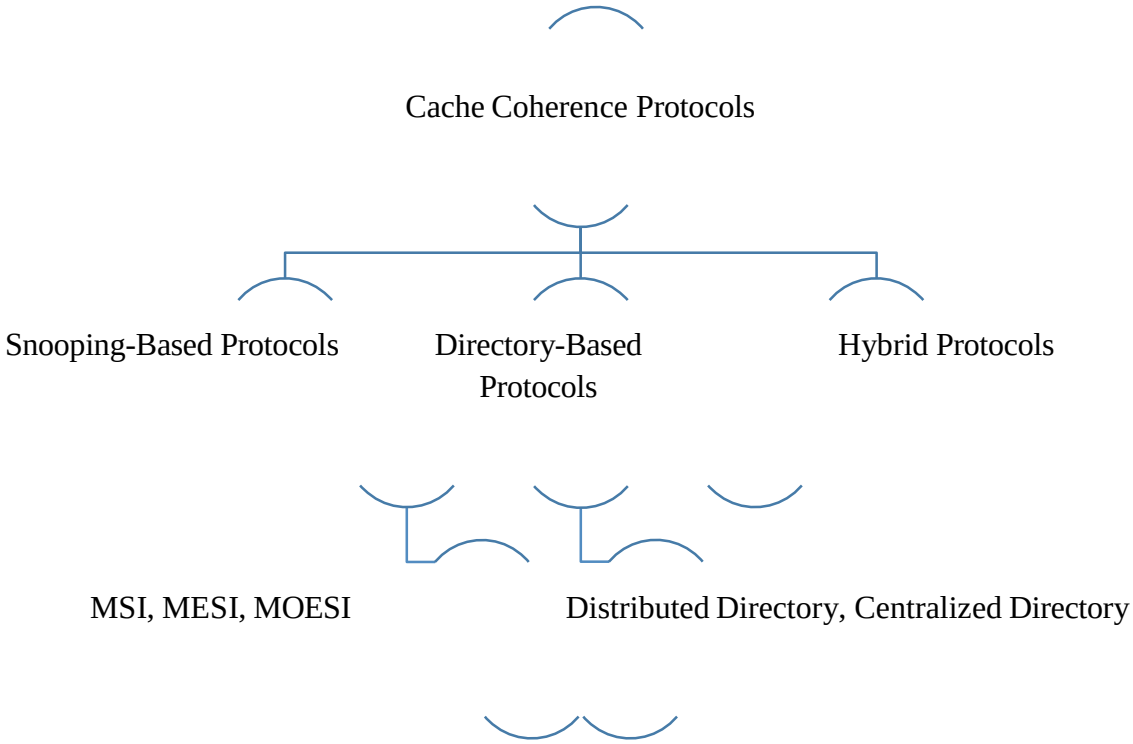

Fig. 4. Taxonomy of Cache Coherence Protocols in Multiprocessor Systems

3.1 Snooping Based Coherence Protocols

Snooping based coherence protocols rely on a shared interconnection medium, typically a bus, where all cache controllers monitor or “snoop” on memory transactions occurring on the bus. When a processor performs a read or write operation, the request is broadcast to all caches connected to the bus. Each cache controller observes the transaction and updates its cache state accordingly to maintain coherence. For example, if a processor writes to a

cache block that is shared by other processors, the snooping mechanism ensures that the corresponding copies in other caches are either invalidated or updated [7], [19-21].

Snooping protocols are relatively simple to implement and provide low latency coherence management because all processors can observe memory transactions simultaneously. As a result, they are commonly used in small scale multiprocessor systems where the number of processors is limited. However, the broadcast-based communication model generates significant coherence traffic and bus contention as the number of processors increases. This limitation restricts the scalability of snooping protocols in large multicore and manycore systems [37].

3.2 Directory Based Coherence Protocols

To overcome the scalability limitations of snooping protocols, directory-based coherence protocols were introduced. In directory-based systems, a directory structure is used to maintain information about which processors currently hold copies of each memory block. Instead of broadcasting coherence messages to all processors, communication occurs only between the processors involved in a particular memory transaction. The directory maintains a record of sharers and coordinates coherence actions such as invalidation or update messages when a processor modifies a shared cache block [10-11], [19], [22-24].

Directory-based protocols significantly reduce unnecessary communication by replacing broadcast transactions with targeted point-to-point messages. This design improves scalability and makes directory-based coherence suitable for large scale multiprocessor systems and distributed shared memory architectures. However, the directory structure requires additional storage and management logic, which increases hardware complexity and introduces additional latency in certain cases [10], [25].

3.3 Hybrid Coherence Protocols

Hybrid coherence protocols combine features of both snooping and directory-based approaches to balance scalability and performance. These protocols typically use snooping mechanisms within small processor clusters while employing directory-based coordination across clusters. By combining both techniques, hybrid protocols aim to reduce coherence traffic while maintaining fast communication within local processor groups[26-30]. Hybrid coherence approaches are increasingly relevant in modern manycore processors and heterogeneous architectures where processors are organized into multiple clusters connected through high-speed interconnect networks. In such systems, hybrid protocols provide a flexible solution that adapts to different communication patterns and reduces the overhead associated with purely broadcast-based or purely directory-based mechanisms [13-14].

The taxonomy of cache coherence protocols highlights the trade-offs between simplicity, scalability, and communication overhead. Snooping protocols offer simplicity and low latency for small systems, while directory- based protocols provide better scalability for large multiprocessor architectures. Hybrid approaches attempt to combine the advantages of both mechanisms to meet the demands of modern multicore and heterogeneous computing systems.

4 CLASSICAL CACHE COHERENCE PROTOCOLS

Over the years, several cache coherence protocols have been proposed to maintain consistency among cached data in multiprocessor systems. These protocols regulate the states of cache blocks and define how processors communicate with each other during memory read and write operations. Classical coherence protocols typically operate using a finite state machine model, where each cache line can transition between different states depending on processor requests and coherence messages. Among the most widely studied and implemented protocols are MSI, MESI, MOESI, and MESIF, each introducing additional states to improve system performance and reduce coherence traffic [4], [31-33]. These protocols form the foundation of many modern multiprocessor architectures.

4.1 MSI Protocol

The MSI protocol is one of the earliest and simplest cache coherence protocols used in shared-memory multiprocessor systems. It defines three possible states for each cache block: Modified (M), Shared (S), and Invalid

(I). In the Modified state, the cache block has been updated by the processor and differs from the corresponding value in main memory. In this state, the cache holds the only valid copy of the data and must write the updated value back to memory before another processor can access it. In the Shared state, multiple caches may hold copies of the same data block, and the value is consistent with the main memory. The Invalid state indicates that the cache block does not contain valid data and must fetch the block from memory or another cache when accessed [34-35].

Although the MSI protocol is simple to implement, it has certain limitations. For example, a processor must invalidate other shared copies before modifying a cache block, which can lead to additional coherence traffic and performance overhead. As a result, more advanced protocols were later introduced to reduce unnecessary memory transactions and improve system efficiency [4].

4.2 MESI Protocol

The MESI protocol extends the MSI protocol by introducing an additional state called Exclusive (E). The four states of the MESI protocol are Modified, Exclusive, Shared, and Invalid. The Exclusive state indicates that a cache block is present only in a single cache and is consistent with the main memory. This allows the processor to modify the block without generating invalidation messages, thereby reducing unnecessary coherence traffic [4], [36].

The MESI protocol improves system performance by minimizing memory access operations and enabling efficient cache-to-cache communication. It is widely implemented in many commercial processors because it provides a good balance between hardware complexity and performance optimization. However, MESI still requires memory intervention in certain cases where multiple processors request shared data simultaneously [31].

4.3 MOESI Protocol

The MOESI protocol further enhances the MESI protocol by introducing an additional Owner (O) state. The five states in this protocol are Modified, Owner, Exclusive, Shared, and Invalid. The Owner state allows a cache that holds the modified version of a block to supply data directly to other requesting caches without writing the block back to main memory. This capability significantly reduces memory bandwidth consumption and improves overall system performance [37-38]. By enabling direct cache-to-cache data transfer, the MOESI protocol reduces the number of write-back operations and decreases coherence traffic in multiprocessor systems. As a result, MOESI is commonly used in high performance multicore processors and advanced shared-memory architectures [4].

4.4 MESIF Protocol

The MESIF protocol is another extension of the MESI protocol and introduces a Forward (F) state to optimize data sharing among processors. The protocol consists of five states: Modified, Exclusive, Shared, Invalid, and Forward. In this scheme, when multiple caches hold a shared copy of a block, only one cache is designated as the forwarder. This cache is responsible for supplying the requested data to other processors, thereby reducing redundant responses and improving communication efficiency [31], [39]. The MESIF protocol is particularly useful in systems with point-to-point interconnect networks, where minimizing communication overhead is essential for maintaining high performance. By designating a single forwarding cache, the protocol ensures efficient data delivery while reducing unnecessary memory accesses and coherence traffic [31].

These classical coherence protocols illustrate the evolution of cache coherence mechanisms in multiprocessor systems. Each successive protocol introduces additional states and optimizations to improve performance, reduce communication overhead, and enhance scalability. Understanding these protocols provides valuable insight into the design of modern coherence mechanisms used in multicore and manycore architectures.

5 COMPARATIVE ANALYSIS OF CACHE COHERENCE PROTOCOLS

Cache coherence protocols play a crucial role in maintaining data consistency in multiprocessor systems. Over time, several coherence protocols have been developed with the objective of improving system performance, reducing coherence traffic, and enhancing scalability. Classical protocols such as MSI, MESI, MOESI, and MESIF differ primarily in the number of cache states they employ, and the mechanisms used to manage data sharing and modification among processors. These differences significantly influence communication overhead, memory bandwidth utilization, and implementation complexity in multiprocessor architectures [1], [4].

The MSI protocol represents the simplest coherence mechanism, but it generates relatively higher coherence traffic due to frequent invalidation and write-back operations. The MESI protocol improves upon MSI by introducing an Exclusive state that allows a processor to modify a cache block without generating additional coherence messages when the block is not shared.

This optimization reduces unnecessary bus transactions and improves overall performance [1], [35].

The MOESI protocol further enhances coherence efficiency by introducing an Owner state that allows modified cache blocks to be supplied directly to requesting caches without first writing the data back to main memory. This capability reduces memory bandwidth consumption and enables efficient cache-to-cache communication. Similarly, the MESIF protocol introduces a Forward state that designates a specific cache to respond to read requests, thereby avoiding multiple simultaneous responses from different caches and reducing communication overhead [31], [37].

Table 1 summarizes the key characteristics of commonly used cache coherence protocols in terms of the number of states, scalability, communication overhead, and implementation complexity.

Table 1: Comparative Analysis of Classical Cache Coherence Protocols

Protocol	**States**	**Key Feature**	**Communication Overhead**	**Scalability**	**Implementation Complexity**
MSI	M, S, I	Basic invalidation protocol	High	Low	Low
MESI	M, E, S, I	Exclusive state reduces memory access	Moderate	Medium	Moderate
MOESI	M, O, E, S, I	Owner state enables cache- to-cache transfer	Low	High	High
MESIF	M, E, S, I, F	Forward state optimizes data response	Low	High	High

For visualization purposes, the qualitative metrics such as communication overhead, scalability, and implementation complexity are represented using a relative scale ranging from 1 (low) to 5 (high), and the comparative characteristics of classical cache coherence protocols are illustrated in Fig. 5.

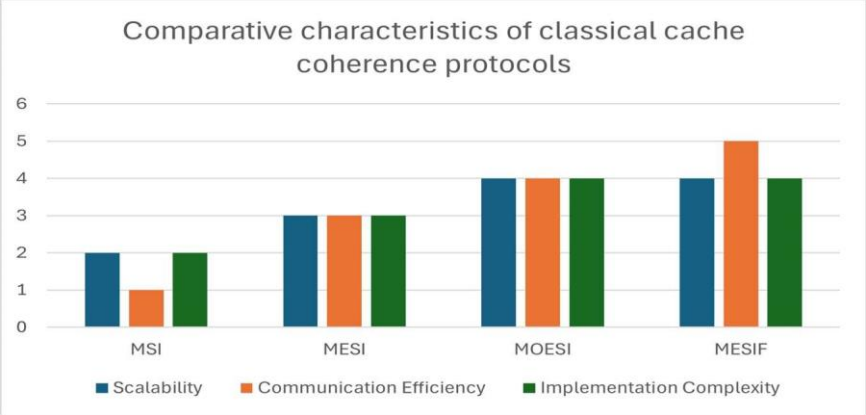

Fig. 5. Comparative characteristics of classical cache coherence protocols

From the comparison, it can be observed that the evolution of coherence protocols primarily focuses on reducing unnecessary memory transactions and improving data sharing efficiency among processors. Protocols with additional states, such as MOESI and MESIF, provide improved performance and scalability by enabling direct cache-to-cache data transfer and minimizing coherence traffic. However, these protocols also introduce increased hardware complexity due to the additional state transitions and control logic required in the cache controller [38].

In large scale multicore systems, scalability becomes a critical factor when selecting an appropriate coherence protocol. Snooping-based implementations of MSI and MESI protocols are suitable for small multiprocessor systems due to their simplicity and low latency. In contrast, protocols such as MOESI and MESIF are better suited for modern multicore processors where efficient data sharing and reduced communication overhead are essential for maintaining high system performance [2], [4].

The comparative analysis highlights the trade-offs between simplicity, performance, and scalability in cache coherence protocol design. While simpler protocols offer ease of implementation, advanced protocols provide better performance and scalability by incorporating additional optimization states. Understanding these trade-offs is essential for designing next generation coherence mechanisms in manycore and heterogeneous computing environments.

6 DESIGN CHALLENGES IN CACHE COHERENCE PROTOCOLS

Although cache coherence protocols play a vital role in maintaining data consistency in shared-memory multiprocessor systems, their design becomes increasingly challenging as the number of processing cores grows. Modern multicore and manycore systems introduce significant complexity in managing coherence traffic, memory latency, and energy consumption. As a result, designing scalable and efficient coherence mechanisms remains an important research challenge in computer architecture [2], [5], [40].

6.1 Scalability Challenges

One of the primary challenges in cache coherence protocol design is scalability. Early coherence mechanisms such as snooping-based protocols rely on broadcast communication over a shared bus, where all cache controllers monitor memory transactions to maintain consistency. While this approach works efficiently for small multiprocessor systems, it becomes inefficient as the number of processors increases. The broadcast nature of snooping communication leads to increased bus contention and coherence traffic, limiting the scalability of these protocols in large multicore systems [7], [21]. Directory-based coherence protocols were introduced to address this issue by replacing broadcast communication with targeted point-to-point messaging. Although directory-based protocols improve scalability, they introduce additional storage overhead for maintaining directory entries and increase the complexity of coherence management in large systems [10], [41-42].

6.2 Communication Overhead

Cache coherence protocols generate many control messages to maintain consistency among caches. These messages include invalidation requests, acknowledgments, data transfers, and state transition signals. As the number of processors increases, the volume of coherence traffic also increases significantly, which can saturate the interconnection network and degrade system performance. Excessive coherence traffic can also lead to increased memory latency and reduced throughput in high performance multiprocessor systems [5], [43]. Optimizing communication overhead is therefore an important objective in coherence protocol design. Techniques such as cache-to-cache data transfer, selective invalidation, and coherence filtering have been proposed to reduce unnecessary communication among processors [38].

6.3 Latency and Synchronization Issues

Another important challenge in coherence protocols is the latency associated with coherence operations. When a processor requests access to a cache block that is currently owned or modified by another processor, the requesting

processor must wait until the coherence protocol resolves the ownership and provides the correct data. This process may involve multiple coherence messages and state transitions, which can increase memory access latency and stall processor pipelines [44]. Synchronization operations in parallel programs can further exacerbate these delays because multiple processors may attempt to access shared data simultaneously. Efficient coherence mechanisms must therefore minimize latency while maintaining correct data consistency across caches.

6.4 Power and Energy Consumption

Energy efficiency has become an important concern in modern multicore systems. Coherence protocols contribute significantly to power consumption due to the large number of coherence messages and memory accesses generated during program execution. Increased coherence traffic leads to higher energy usage in the interconnection network, cache controllers, and memory

subsystems. As a result, researchers have explored several energy-efficient coherence techniques such as adaptive coherence protocols, traffic filtering mechanisms, and energy aware cache management strategies [13], [35]. Reducing coherence-related power consumption is particularly important in many-core processors and large-scale computing systems, where energy efficiency directly impacts system reliability and operational cost.

These challenges highlight the need for scalable and efficient coherence mechanisms capable of supporting modern multicore architectures. Addressing these issues remains an active area of research in computer architecture and parallel computing.

7 EMERGING TRENDS IN CACHE COHERENCE

The rapid evolution of multicore and many-core processors has led to the development of advanced coherence mechanisms that aim to improve scalability, reduce communication overhead, and enhance energy efficiency. Traditional coherence protocols such as MSI, MESI, and MOESI were designed primarily for small to medium-scale multiprocessor systems. However, modern computing platforms, including many-core processors and heterogeneous architectures, require more adaptive and scalable coherence techniques. As a result, recent research has focused on developing innovative approaches such as adaptive coherence protocols, machine learning based coherence optimization, and coherence mechanisms for heterogeneous computing systems [13-15].

7.1 Adaptive Cache Coherence Protocols

Adaptive coherence protocols dynamically adjust their behaviour based on the runtime characteristics of applications and memory access patterns. Instead of using a fixed coherence mechanism, these protocols can switch between different coherence strategies to improve system performance and reduce communication overhead. For example, adaptive protocols may employ directory-based coherence for large-scale communication while using snooping mechanisms within smaller processor clusters. This flexibility enables the protocol to balance scalability and performance depending on the workload characteristics [13], [45-50].

Adaptive coherence approaches also attempt to optimize coherence traffic by detecting sharing patterns such as migratory sharing and producer consumer communication. By identifying these patterns, the protocol can apply specialized optimization techniques that reduce unnecessary coherence messages and improve data access efficiency.

7.2 Machine Learning Based Coherence Optimization

Machine learning techniques have recently been explored as a promising approach for optimizing cache coherence mechanisms. In these approaches, predictive models analyze memory access patterns and coherence behaviour to anticipate future coherence events. By predicting whether a cache block will be shared, modified, or invalidated, the system can proactively adjust coherence actions and reduce unnecessary communication among processors. Machine learning based coherence techniques can also be used to filter redundant coherence

messages and dynamically optimize cache configurations. These methods have shown potential for improving energy

efficiency and reducing coherence traffic in many-core architectures. Although still an emerging research area, machine learning assisted coherence mechanisms are expected to play a significant role in the design of future multicore processors [12], [51].

7.3 Coherence in Heterogeneous Architectures

Modern computing systems increasingly integrate heterogeneous processing units such as CPUs, GPUs, and specialized accelerators within a single system. These heterogeneous architectures introduce additional challenges in maintaining coherence because different processing units may have distinct memory hierarchies and access patterns. Ensuring efficient data sharing among these components requires advanced coherence mechanisms that can operate across diverse hardware components [14], [52-53]. Recent research has proposed heterogeneous coherence frameworks that support unified memory access across CPUs and GPUs [15], [54]. These mechanisms aim to maintain data consistency while minimizing communication overhead and synchronization delays. Efficient coherence support for heterogeneous architectures is becoming increasingly important as heterogeneous computing platforms continue to gain prominence in high performance computing and data intensive applications[55].

These emerging trends indicate that future cache coherence mechanisms will likely incorporate adaptive strategies, predictive models, and cross architecture coherence management to address the growing complexity of modern computing systems.

8 RESEARCH GAPS AND FUTURE DIRECTIONS

Although significant progress has been made in the design of cache coherence protocols, several challenges and limitations remain in modern multiprocessor systems. Classical coherence protocols such as MSI, MESI, and MOESI have been widely adopted in shared memory architectures; however, these protocols were originally designed for systems with a relatively small number of processors. As the scale and complexity of modern computing systems continue to grow, several research gaps have emerged that require further investigation.

One major limitation of traditional coherence protocols is their limited scalability in many-core architectures. As the number of processing cores increases, the amount of coherence traffic generated by read and write operations also grows significantly. This increase in communication overhead can lead to network congestion and reduced system performance. Although directory-based coherence protocols help mitigate this issue by reducing broadcast communication, they introduce additional storage overhead and management complexity for maintaining directory information [10].

Another important research gap lies in the efficient management of coherence traffic in large-scale multicore systems. Even with advanced protocols such as MOESI and MESIF, coherence communication may still generate unnecessary invalidation messages and data transfers among

processors. Reducing redundant coherence traffic through intelligent filtering techniques and adaptive communication mechanisms remains an open research problem in modern computer architectures [38].

Energy efficiency is also a growing concern in modern processors. As coherence protocols generate many control messages and memory accesses, they contribute significantly to the overall power consumption of the system. Designing energy efficient coherence mechanisms that minimize communication overhead while maintaining data consistency is therefore an important research direction [35], [56].

In addition, the emergence of heterogeneous computing architectures introduces new challenges for coherence protocol design. Modern systems often integrate CPUs, GPUs, and specialized accelerators within the same platform. These components may use different memory hierarchies and communication mechanisms, making it difficult to maintain efficient coherence across the entire system. Developing scalable coherence frameworks that support heterogeneous architectures is therefore an important area for future research [14].

Future research in cache coherence protocols is expected to focus on several promising directions. These include the development of traffic-optimized coherence protocols, adaptive coherence mechanisms that dynamically adjust protocol behaviour based on workload characteristics, and machine learning assisted coherence techniques

that predict memory access patterns and reduce unnecessary communication. Such approaches have the potential to significantly improve the scalability, performance, and energy efficiency of next generation multiprocessor systems.

9 CONCLUSION

This paper presented a survey and comparative analysis of cache coherence protocols used in shared memory multiprocessor systems. Classical protocols such as MSI, MESI, MOESI, and MESIF were examined to highlight their design principles, advantages, and limitations in terms of scalability, communication overhead, and implementation complexity. The analysis indicates that while advanced protocols improve cache-to-cache communication and reduce memory traffic, maintaining efficient coherence remains challenging as processor core counts increase. Furthermore, emerging approaches such as adaptive coherence mechanisms, heterogeneous coherence frameworks, and machine learning assisted optimization show promising directions for improving the scalability and efficiency of future multicore systems.

REFERENCES

[1] P. Stenstrom, “A survey of cache coherence schemes for multiprocessors,” *Computer (Long. Beach. Calif).*, vol. 23, no. 6, pp. 12–24, Jun. 1990, doi: 10.1109/2.55497.

[2] L. Han, Jianfeng An, D. Gao, X. Fan, X. Ren, and T. Yao, “A survey on cache coherence for tiled many- core processor,” in *2012 IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC 2012)*, IEEE, Aug. 2012, pp. 114–

118. doi: 10.1109/ICSPCC.2012.6335721.

[3] T. Varsha and K. Sanjay, "Perspective Study and Analysis of Parallel Architecture," *Int. J. Comput. Appl.*, vol. 148, no. 14, pp. 22–25, 2016, doi: 10.5120/ijca2016911285.

[4] Z. Al-Waisi and M. O. Agyeman, "An overview of on-chip cache coherence protocols," in *2017 Intelligent Systems Conference (IntelliSys)*, London, UK: IEEE, Sep. 2017, pp. 304–309. doi: 10.1109/IntelliSys.2017.8324309.

[5] N. Parvathy, B. R. Upadhyay, and T. S. B. Sudarshan, "Cache coherence: A walkthrough of mechanisms and challenges," in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, Chennai, India: IEEE, Mar. 2016, pp. 2251–2256. doi: 10.1109/ICEEOT.2016.7755093.

[6] D. Ramtake, N. Singh, S. Kumar, and V. K. Patle, "Cache Associativity Analysis of Multicore Systems," in *2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*, IEEE, Mar. 2020, pp. 1–4. doi: 10.1109/ICCSEA49143.2020.9132884.

[7] B. Hashemi, "Simulation and Evaluation Snoopy Cache Coherence Protocols with Update Strategy in Shared Memory Multiprocessor Systems," in *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications Workshops*, Busan, Korea (South): IEEE, May 2011, pp. 256–259. doi: 10.1109/ISPAW.2011.68.

[8] R. Lawrence, "A Survey of Cache Coherence Mechanisms in Shared Memory Multiprocessors," 1998.

[9] V. Nagarajan, D. J. Sorin, M. D. Hill, and D. A. Wood, *A Primer on Memory Consistency and Cache Coherence*, vol. 6, no. 3. in Synthesis Lectures on Computer Architecture, vol. 6. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-031-01764-3.

[10] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," in *[1988] The 15th Annual International Symposium on Computer Architecture. Conference Proceedings*, Honolulu, HI, USA: IEEE Comput. Soc. Press, 1988, pp. 280–289. doi: 10.1109/ISCA.1988.5238.

[11] H. Arora, R. Mukherjee, A. Bej, and H. Adak, "Directory based cache coherence modeller in multiprocessors: Medium insight," in *2014 International Conference on Advances in Computing,Communications and Informatics (ICACCI)*, Delhi, India: IEEE, Sep. 2014, pp. 2611–2617. doi: 10.1109/ICACCI.2014.6968444.

[12] S. Charles, A. Ahmed, U. Y. Ogras, and P. Mishra, "Efficient Cache Reconfiguration Using Machine Learning in NoC-Based Many-Core CMPs," *ACM Transact. Des. Autom. Electron. Syst.*, vol. 24, no. 6, pp. 1–23, Nov. 2019, doi: 10.1145/3350422.

[13] N. Chaturvedi, P. Sharma, and S. Gurunarayanan, "An adaptive coherence protocol with adaptive cache for multi-core architectures," in *2013 International Conference on*

Advanced Electronic Systems (ICAES), Pilani, India: IEEE, Sep. 2013, pp. 197–201. doi: 10.1109/ICAES.2013.6659391.

[14] I. Singh, A. Shriraman, W. W. L. Fung, M. O'Connor, and T. M. Aamodt, "Cache Coherence for GPU Architectures," *IEEE Micro*, vol. 34, no. 3, pp. 69–79, May 2014, doi: 10.1109/MM.2014.4.

[15] D. K. Keshar, S. Kumar, and V. K. Patle, "A Comparative Study of GPU Computing Techniques : A Review," *International Journal of Computer Sciences and Engineering*, vol. 7, no. 3, pp. 178–181, Feb. 2019.

[16] M. T. Banday and M. Khan, "A study of recent advances in cache memories," in *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, Mysore, India: IEEE, Nov. 2014, pp. 398–

403. doi: 10.1109/IC3I.2014.7019786.

[17] K. Hwang, *Advanced Computer Architecture*, Third., vol. 626. in Communications in Computer and Information Science, vol. 626. Singapore: Springer Singapore, 2016. doi: 10.1007/978-981-10-2209-8.

[18] N. P. Jouppi, "Cache write policies and performance," *ACM SIGARCH Computer Architecture News*, vol. 21, no. 2, pp. 191–201, May 1993, doi: 10.1145/173682.165154.

[19] S. Soomro, S. Al-Hothali, K. Tanvir, and R. Tuli, "Snoopy and Directory Based Cache Coherence Protocols:

A Critical Analysis," 2010. [Online]. Available: https://www.researchgate.net/publication/259821213

[20] S. Miyoshi, T. Sasaki, Y. Fukazawa, and T. Kondo, "An Architectural Framework of Snoopy Interconnection for Heterogeneous Cache Systems," in *2015 Third International Symposium on Computing and Networking (CANDAR)*, Sapporo, Japan: IEEE, Dec. 2015, pp. 561–565. doi: 10.1109/CANDAR.2015.44.

[21] M. Tomasevic and V. Milutinovic, "A simulation study of snoopy cache coherence protocols," in *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, IEEE, 1992, pp. 427–436 vol.1. doi: 10.1109/HICSS.1992.183192.

[22] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, "The directory-based cache coherence protocol for the DASH multiprocessor," *ACM SIGARCH Computer Architecture News*, vol. 18, no. 2SI, pp. 148–159, Jun. 1990, doi: 10.1145/325096.325132.

[23] Z. Wu, M. Bekmyrza, N. Kapre, and H. Patel, "Ditty: Directory-based Cache Coherence for Multicore Safety-critical Systems," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium: IEEE, Apr. 2023, pp. 1–6. doi: 10.23919/DATE56975.2023.10136986.

[24] R. E. Ahmed and M. K. Dhodhi, "Directory-based cache coherence protocol for power-

aware chip- multiprocessors," in *2011 24th Canadian Conference on Electrical and Computer Engineering(CCECE)*, Niagara Falls, ON, Canada: IEEE, May 2011, pp. 001036–001039. doi: 10.1109/CCECE.2011.6030618.

[25] J. Archibald and J.-L. Baer, "Cache coherence protocols: evaluation using a multiprocessor simulation model," *ACM Transactions on Computer Systems*, vol. 4, no. 4, pp. 273–298, Sep. 1986, doi: 10.1145/6513.6514.

[26] J. H. Henry, "HYBRID COHERENCE FOR SCALABLE MULTICORE ARCHITECTURES," 2010.Accessed: Mar. 16, 2026. [Online]. Available: https://api.semanticscholar.org/CorpusID:60041698

[27] S. H. Gade, M. Sinha, M. Kumar, and S. Deb, "Scalable Hybrid Cache Coherence Using Emerging Links for Chiplet Architectures," in *2022 35th International Conference on VLSI Design and 2022 21st International Conference on Embedded Systems (VLSID)*, IEEE, Feb. 2022, pp. 92–97. doi: 10.1109/VLSID2022.2022.00029.

[28] H. Chtioui, R. Ben Atitallah, S. Niar, J.-L. Dekeyser, and M. Abid, "A Dynamic Hybrid Cache Coherency Protocol for Shared-Memory MPSoC," in *2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, Patras, Greece: IEEE, Aug. 2009, pp. 3–10. doi: 10.1109/DSD.2009.220.

[29] M. Gemieux, M. Li, Y. Savaria, J.-P. David, and G. Zhu, "A Hybrid Architecture With Low Latency Interfaces Enabling Dynamic Cache Management," *IEEE Access*, vol. 6, pp. 62826–62839, 2018, doi: 10.1109/ACCESS.2018.2876597.

[30] R. Wang, L. Yu, and W. Zhuang, "Design of an efficient hybrid cache coherence protocol on chiplet architecture," in *Third International Conference on Algorithms, Microchips, and Network Applications (AMNA 2024)*, J. Lu and R. Davidrajuh, Eds., SPIE, Jun. 2024, p. 29. doi: 10.1117/12.3031958.

[31] B. Qian and L. Yan, "The research of the inclusive cache used in multi-core processor," in *2008 International Conference on Electronic Packaging Technology & High Density Packaging*, IEEE, Jul. 2008,pp. 1–4. doi: 10.1109/ICEPT.2008.4606981.

[32] S. Almakdi, A. Alazeb, and M. Alshehri, "Cache Coherence Mechanisms," *International Journal of Engineering and Innovative Technology*, vol. 4, no. 7, pp. 158–167, Jan. 2015.

[33] M. R. Marty, "Cache Coherence Techniques for Multicore Processors," University of Wisconsin-Madison, 2008.

[34] N. B. Mallya, G. Patil, and B. Raveendran, "Simulation based Performance Study of Cache Coherence Protocols," in *2015 IEEE International Symposium on Nanoelectronic and Information Systems*, Indore, India: IEEE, Dec. 2015, pp. 125–130. doi: 10.1109/iNIS.2015.52.

[35] A. Saraswat, K. Abhishek, H. K. Azad, and S. Shitharth, "MSI-A: An Energy Efficient

Approximated Cache Coherence Protocol," *IEEE Access*, vol. 11, pp. 48123–48135, 2023, doi: 10.1109/ACCESS.2023.3273219.

[36] A. Patel and K. Ghose, "Energy-efficient MESI cache coherence with pro-active snoop filtering for multicore microprocessors," in *Proceeding of the thirteenth international symposium on Low power electronics and design - ISLPED '08*, New York, New York, USA: ACM Press, 2008, p. 247. doi: 10.1145/1393921.1393988.

[37] H. Altwaijry and D. S. Alzahrani, "Improved-MOESI Cache Coherence Protocol," *Arab. J. Sci. Eng.*, vol. 39, no. 4, pp. 2739–2748, Apr. 2014, doi: 10.1007/s13369-013-0787-7.

[38] H. Shukur, S. Zeebaree, R. Zebari, O. Ahmed, L. Haji, and D. Abdulqader, "Cache Coherence Protocols in Distributed Systems," *Journal of Applied Science and Technology Trends*, vol. 1, no. 2, pp. 92–97, Jun. 2020, doi: 10.38094/jastt1329.

[39] A. W. Hay, "MESIF Cache Coherence Protocol," University of Auckland, 2012.

[40] R. Wang, J. Wang, and W. G. Aref, "Cache Coherence Over Disaggregated Memory," in *Proceedings of the VLDB Endowment*, VLDB Endowment, 2025, pp. 2978–2991. doi: 10.14778/3746405.3746422.

[41] S. Demetriades and S. Cho, "Stash directory: A scalable directory for many-core coherence," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, Feb. 2014, pp. 177–188. doi: 10.1109/HPCA.2014.6835928.

[42] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal, "Directory-based cache coherence in large-scale multiprocessors," *Computer (Long. Beach. Calif).*, vol. 23, no. 6, pp. 49–58, Jun. 1990, doi: 10.1109/2.55500.

[43] S. Kaxiras and G. Keramidas, "SARC Coherence: Scaling Directory Cache Coherence in Performance and Power," *IEEE Micro*, vol. 30, no. 5, pp. 54–65, Sep. 2010, doi: 10.1109/MM.2010.82.

[44] B. Boothe and A. Ranade, "Improved multithreading techniques for hiding communication latency in multiprocessors," in *Proceedings of the 19th annual international symposium on Computer architecture - ISCA '92*, New York, New York, USA: ACM Press, 1992, pp. 214–223. doi: 10.1145/139669.139729.

[45] A. L. Cox and R. J. Fowler, "Adaptive Cache Coherency For Detecting Migratory Shared Data," in *Proceedings of the 20th Annual International Symposium on Computer Architecture*, CA, USA: IEEE Comput. Soc. Press, 1993, pp. 98–108. doi: 10.1109/ISCA.1993.698549.

[46] L. G. Menezo, V. Puente, and J. A. Gregorio, "An adaptive cache coherence protocol: Trading storage for traffic," *J. Parallel Distrib. Comput.*, vol. 102, pp. 163–174, 2017, doi: 10.1016/j.jpdc.2016.12.020.

[47] P. Stenstrom, M. Brorsson, and L. Sandberg, "An Adaptive Cache Coherence Protocol

Optimized For Migratory Sharing," in *Proceedings of the 20th Annual International Symposium on Computer Architecture*, IEEE Comput. Soc. Press, 1993, pp. 120–130. doi: 10.1109/ISCA.1993.698551.

[48] Won-Kee Hong, Nam-Hee Kim, and Shin-Dug Kim, "Design and performance evaluation of an adaptive cache coherence protocol," in *Proceedings 1998 International Conference on Parallel and Distributed Systems (Cat. No.98TB100250)*, Tainan, Taiwan: IEEE Comput. Soc, 1998, pp. 33–40. doi: 10.1109/ICPADS.1998.741017.

[49] V. Uma and R. Marimuthu, "D-wash – A dynamic workload aware adaptive cache coherance protocol for multi-core processor system," *Microelectronics J.*, vol. 132, p. 105675, Feb. 2023, doi: 10.1016/j.mejo.2022.105675.

[50] A. Srinivasan, M. Amidzadeh, J. Zhang, and O. Tirkkonen, "Adaptive Cache Policy Optimization Through Deep Reinforcement Learning in Dynamic Cellular Networks," *Intelligent and Converged Networks*, vol. 5, no. 2, pp. 81–99, Jun. 2024, doi: 10.23919/ICN.2024.0007.

[51] A. Ahmed, Y. Huang, and P. Mishra, "Cache Reconfiguration Using Machine Learning for Vulnerability- aware Energy Optimization," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 2, pp. 1–24, Mar. 2019, doi: 10.1145/3309762.

[52] S. Kumar, "Mathematical Modelling and Simulation of a Buffered Fault Tolerant Double Tree Network," in *15th International Conference on Advanced Computing and Communications (ADCOM 2007)*, IEEE, Dec. 2007, pp. 422–433. doi: 10.1109/ADCOM.2007.62.

[53] S. Pei, M.-S. Kim, J.-L. Gaudiot, and N. Xiong, "Fusion Coherence: Scalable Cache Coherence for Heterogeneous Kilo-Core System," in *CCIS*, vol. 451, Springer, 2014, pp. 1–15. doi: 10.1007/978-3-662- 44491-7_1.

[54] N. Agarwal, D. Nellans, E. Ebrahimi, T. F. Wenisch, J. Danskin, and S. W. Keckler, "Selective GPU caches to eliminate CPU-GPU HW cache coherence," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, Mar. 2016, pp. 494–506. doi: 10.1109/HPCA.2016.7446089.

[55] N. Oswald, V. Nagarajan, D. J. Sorin, V. Gavrielatos, T. X. Olausson, and R. Carr, "HeteroGen: Automatic Synthesis of Heterogeneous Cache Coherence Protocols," *IEEE Micro*, vol. 43, no. 4, pp. 62–70, Jul. 2023, doi: 10.1109/MM.2023.3274993.

[56] R. Ahmed, "Energy-Aware Cache Coherence Protocol for Chip-Multiprocessors," in *2006 Canadian Conference on Electrical and Computer Engineering*, IEEE, 2006, pp. 82–85. doi: 10.1109/CCECE.2006.277390.