European Journal of Computer Science and Information Technology,13(37),25-35, 2025
Print ISSN: 2054-0957 (Print)
Online ISSN: 2054-0965 (Online)
Website: https://www.eajournals.org/
Publication of the European Centre for Research Training and Development -UK

Demystifying Cache Coherency in Modern Multiprocessor Systems

Sruthi Somarouthu
University of Texas at Austin, USA

doi: https://doi.org/10.37745/ejcsit.2013/vol13n372535 Published June 07, 2025

Citation: Somarouthu S. (2025) Demystifying Cache Coherency in Modern Multiprocessor Systems, *European Journal of Computer Science and Information Technology*,13(37),25-35

Abstract: *Cache coherency remains a fundamental architectural challenge in modern multi-core processors, balancing data consistency with performance. This article examines the intricate mechanics of cache coherence protocols, from the basic principles of memory hierarchy to advanced implementations like MESIF, MOESI and token coherence. The exploration begins with the core problem of maintaining consistent data views across distributed caches, continues through implementation mechanisms, including snooping and directory-based approaches, and addresses critical performance considerations such as coherency traffic, latency penalties, and false sharing. The discussion extends to cutting-edge protocol extensions that optimize for specific access patterns in contemporary computing environments, providing insights for digital hardware architects seeking to maximize multi-core efficiency.*

Keywords: Cache coherency, multi-core processors, memory hierarchy, false sharing, snooping, token coherence

INTRODUCTION

In the realm of computer architecture, few concepts are as fundamental to the performance of modern multi-core processors as cache coherency. As we continue to pack more processing cores into our CPUs, the efficient management of shared data becomes increasingly critical. This article explores the intricacies of cache coherence protocols, their implementation, and their impact on system performance. The evolution of multi-core processors has necessitated sophisticated cache coherency mechanisms. Research by Amit Joshi et al., demonstrates that in interconnected multi-core systems, coherence traffic can consume a large portion of total on-chip network bandwidth and significantly impact overall system performance [1]. Their analysis of directory-based and snooping protocols across varied workloads reveals that directory-based protocols like MESI show much lower latency compared to basic snooping approaches when core counts exceed eight.

Cache coherency performance varies dramatically based on workload characteristics. When testing with SPLASH-2 and PARSEC benchmark suites, researchers observed that applications with frequent write-sharing patterns, such as Ocean and Canneal, experienced coherence miss rates of 14.2% and 12.7%,

respectively, while those with primarily read-sharing patterns showed rates below 5% [1]. These differences translate directly to execution time, with high coherence miss rates correlating to 22-35% longer completion times.

Cache coherence also increases power consumption due to frequent inter-core communication and increased memory traffic. Notably, the power overhead increases non-linearly with the core count. This kind of growth in energy costs underscores why optimization of coherence protocols remains a critical focus in processor design. Modern architectural innovations like non-uniform cache architectures (NUCA) and chiplet-based designs further complicate coherency management, with cross-chipset coherence operations in AMD's EPYC processors incurring up to 3.2x higher latency than same-chipset operations according to empirical measurements.

The Cache Coherence problem

Modern processor architectures employ a hierarchical memory system to bridge the speed gap between fast CPUs and relatively slow main memory. Each processor core typically has its own private cache, providing rapid access to frequently used data. While this arrangement significantly improves performance for single-core applications, it introduces a challenging problem in multi-core environments: maintaining data consistency across caches in multiple cores.

The memory hierarchy design follows fundamental principles of locality to optimize system performance. Temporal locality (recently accessed data is likely to be accessed again) and spatial locality (nearby data is likely to be accessed soon) guide cache design decisions. According to computer architecture principles, an ideal memory system would provide single-cycle access times with unlimited capacity, but physical and economic constraints make this impossible. Instead, hierarchical designs deliver a practical compromise, with modern systems commonly implementing 2-4 cache levels with gradually increasing capacity and latency. A typical L1 cache offers access times of 1-3 cycles with capacities of 32-64 KB per core, while main memory access might require 150-300 cycles but provides gigabytes of storage [2]. This approach delivers an average memory access time approaching that of the fastest level while providing the capacity of the largest level.

Consider the following scenario that exemplifies one of the cache coherency challenges:
Core 1 and Core 2 both read the same memory address A, caching its value (say, 5). Core 1 updates the value at address A to 10 in its local cache. Core 2, unaware of this change, continues using the stale value 5 from its cache.

Research by Molka et al. quantifies the performance implications of memory hierarchy. Their detailed benchmarking of Intel Sandy Bridge and AMD Bulldozer architectures reveals that cache coherency traffic significantly impacts multi-core scaling. On a dual-socket Sandy Bridge system, inter-core communication latency varies dramatically depending on the core distance: 36.4ns for cores sharing an L3 cache segment, 65.5ns for cores on the same socket using different L3 segments, and up to 137ns for cores on different

sockets [3]. This non-uniform access pattern creates severe performance penalties when data-sharing patterns don't align with hardware topology. Their measurements show that memory bandwidth becomes saturated at just 53.3% of the theoretical maximum when coherency traffic is high, compared to 81.7% when cores operate independently [3].

Without proper synchronization mechanisms, the memory hierarchy situation leads to a cache coherence problem, where different cores have inconsistent views of memory. The resulting race conditions can cause unpredictable program behavior, subtle bugs, and system instability. The impact on real-world applications is substantial, with Molka's benchmarks demonstrating that coherency-related overhead can reduce application performance by 12-38% in memory-intensive parallel workloads, particularly when memory accesses cross NUMA boundaries [3]. In contrast, computation-intensive workloads with minimal shared data experience coherency overhead below 5%.

Modern processor designs implement increasingly sophisticated coherency protocols to mitigate these issues. Memory hierarchy designers face the ongoing challenge of balancing coherency maintenance costs against the performance benefits of private caches, particularly as core counts continue to increase. Solutions include snoop-based protocols in which each cache tracks the communication bus, adjusting its state as needed, directory-based protocols that track cache line states centrally, reducing unnecessary broadcasts, and "inclusive" cache designs where higher-level caches contain all data in lower-level caches, simplifying coherency logic at the cost of effective capacity [4].

Table 1: Cache Coherency Latency Across Different Core Topologies [2, 3]

Core Communication Scenario	**Latency (nanoseconds)**	**Relative Latency (x L1 access)**
L1 Cache Access (baseline)	4.0	1.0
Cores sharing L3 cache segment	36.4	9.1
Cores on the same socket, different L3 segments	65.5	16.4
Cores on different sockets	137.0	34.3

Cache Coherence Protocols

To address the challenges, processor designers implement cache coherency protocols—systematic approaches to ensuring that all caches maintain a consistent view of shared memory. These protocols dictate

how caches communicate with each other when data is read or modified. According to Martin et al., despite predictions that increasing core counts would make coherence impractical, their analysis proves that cache coherence remains viable and cost-effective even as processors scale to hundreds of cores. They demonstrate that directory-based coherence protocols require storage overhead that grows as the square root of the number of cores (approximately 2% of total cache size for 512 cores), not quadratically as previously feared [4].

The MESI Protocol

One of the most widely implemented cache coherency protocols is MESI, named after the four possible states a cache line can have:

1. **Modified (M)**: The cache line has been modified and differs from main memory. This cache has the only valid copy of the data.
2. **Exclusive (E)**: The cache line is unmodified and matches the main memory. No other cache has a copy of this line.
3. **Shared (S)**: The cache line is unmodified and matches the main memory. Other caches may have copies of this line.
4. **Invalid (I)**: The cache line is invalid and should not be used.

When a core needs to read or write data, it must follow specific state transition rules based on the current state of the cache line and the operation being performed. According to Nagarajan et al., the MESI protocol leverages the Exclusive state to optimize performance by enabling silent transitions from read to write operations without requiring bus activity — a key difference from the older MSI protocol. This design reduces unnecessary coherence traffic, particularly in scenarios where data is not shared but written by a single processor after a read. These transitions directly impact system performance [5]. Martin et al. explain that modern protocols implement optimizations such as clean-owner requests (allowing a processor to forward clean data without memory controller involvement) and coarse-grain tracking (monitoring coherence at a region level rather than cache-line level), which significantly reduce coherence traffic in scientific and commercial workloads [4].

As cores increase, maintaining data consistency across caches is crucial to avoid performance bottlenecks and data inconsistencies. Furthermore, with the growing complexity of parallel applications and the need for high-performance computing, Martin et al. emphasize that the future of chip design relies heavily on effective and scalable cache coherence solutions, ensuring that these mechanisms will remain a core component in processor design for the foreseeable future [4].

Implementation Mechanisms

Cache coherence protocols are typically implemented through two primary mechanisms, each with distinct performance characteristics and scalability profiles.

Snooping-Based Coherence

In snooping protocols, all cache controllers monitor (or "snoop") the system bus for memory operations. When a cache controller observes an operation on a memory address it has cached, it takes appropriate action based on the coherency protocol. Sorin, Hill, and Wood's comprehensive analysis reveals that snooping protocols rely on atomic broadcast communication, where each coherence transaction must be serialized and observed by all caches simultaneously [6]. Their formal verification methodology demonstrates that without this total ordering of coherence events, race conditions can occur in multi-threaded memory accesses, potentially leading to data inconsistency.

For instance, if Core 1 writes to address A, its cache controller broadcasts this information on the bus. Core 2, snooping the bus, sees this operation and invalidates its own copy of address A if it has one. The performance implications of this approach are significant; Sorin et al. explain that in typical implementations, each broadcast requires the transmission of 3-4 control messages, with each coherence event generating additional on-chip network traffic [6]. Their detailed protocol specifications illustrate that implementation complexity grows with the square of processor count, as each snooping controller must maintain a state regarding all other controllers in the system.

Snooping works well for smaller systems with a single shared bus but becomes less efficient as the number of cores increases due to bus contention. According to formal analysis methods developed by Sorin's team, scalability limitations emerge when system-wide coherence bandwidth exceeds approximately 60-70% of available bus bandwidth, typically occurring at 8-16 cores in conventional architectures [6].

Directory-Based Coherence

For larger multi-core systems, directory-based protocols provide better scalability. These protocols maintain a directory that tracks which caches have copies of each memory block. The Stanford DASH multiprocessor introduced one of the first scalable directory-based cache coherence protocols. Lenoski et al. detail how DASH implements a distributed directory structure that maintains memory coherence across multiple processors organized in clusters [7]. Their directory protocol maintains a bit vector for each memory block, tracking which clusters have cached copies, allowing for targeted invalidations and updates. When a processor requests a memory block, the directory determines the appropriate action based on the current state of the block and the requesting processor's cache state.

Performance measurements from DASH demonstrate that directory-based coherence successfully manages the latency-bandwidth tradeoff in large multiprocessor systems. The overhead introduced by the network and directory access is comparable to the CPU overhead for initiating a single bus transaction. Transactions within a single cluster in DASH require approximately 20 processor cycles, while inter-cluster transfers take about 60 cycles [7]. Despite these latencies, their detailed benchmarking shows that DASH achieves near-linear speedup for many applications, with coherence-related traffic growing proportionally to the number of processors rather than quadratically as in broadcast-based systems.

DASH's directory protocol employs selective invalidation to maintain coherence while minimizing unnecessary network traffic. When a processor modifies a shared cache line, invalidation messages are sent only to processors actually caching the line rather than broadcasting to all processors [7]. Their comparative analysis shows this targeted approach reduces network traffic by 30-70% compared to broadcast-based protocols. This efficiency allows DASH to maintain high performance even as system size scales up, handling large working sets without saturating the interconnection network.

Table 2: Scalability Metrics for Cache Coherency Implementation Mechanisms [6, 7]

Metric	Snooping-Based Protocol	Directory-Based Protocol
Bus Bandwidth Usage	O(N)	O(1) to O(log N)
Scalability (Max # of Cores)	~4–8 cores	32+ cores
Traffic per Memory Access	~N messages (broadcast)	~1–log N messages (targeted)
Storage Overhead	~0 (no directory)	Square root of number of cores
Coherency Overhead (Ops/sec)	Increases linearly with N	Grows with active sharers (≈ O(k), k < N)
Power Consumption	High (scales with N)	Lower (scales with sharer set size)
Fault Tolerance	1 (single point of failure: bus)	3–5 (redundant dir possible, scalable net)
Complexity	simpler, centralized	complex, requires directory controller logic

Performance Implications

Cache coherency mechanisms, while essential for correctness, come with performance costs that must be carefully managed in modern multi-core systems.

Coherency Traffic

Coherency traffic is a critical performance factor in cache coherence protocols used in shared-memory multiprocessor systems. As processors exchange messages to maintain a consistent view of memory, coherence protocols like MESI, MOESI, and other directory-based schemes generate substantial interconnect traffic. This includes invalidations, updates, and acknowledgments, which can congest the memory subsystem and increase latency. The problem intensifies with higher core counts, resulting in reduced scalability and performance. Research by Molka et al. demonstrates that coherence traffic can dominate chip communication, emphasizing the need for scalable coherence mechanisms [8].

Latency Penalties

When a core needs to access data that another core has modified, it must wait for coherency operations to complete, introducing latency. Molka et al.'s comprehensive benchmarking of Intel's Haswell-EP architecture quantifies these penalties precisely. Their cache coherence test reveals that accessing data modified by another core on the same CPU socket incurs a latency of approximately 50 nanoseconds compared to just 1.6 nanoseconds for local L1 cache access—a 31× increase [8]. For cross-socket coherence operations, this penalty jumps to about 95 nanoseconds. Their SPEC benchmarks demonstrate that these coherence latencies reduce effective memory bandwidth significantly when multiple cores simultaneously access shared data structures. Particularly notable is their finding that the cache coherence protocol causes bandwidth to scale sub-linearly, achieving only 68% of the theoretical peak with all cores active.

False Sharing

When different cores modify different variables that happen to reside on the same cache line, the cache line may bounce back and forth between caches, causing severe performance degradation. Rui Zhang et al. demonstrate that false sharing remains a persistent performance bottleneck in parallel applications despite decades of research [9]. Their comprehensive study across contemporary architectures highlights the significant performance and energy consumption penalties caused by false sharing, which vary depending on sharing patterns and memory access frequencies. To address this, they propose Neat, a low-complexity, efficient cache coherence protocol. Neat minimizes the impact of false sharing by intelligently managing cache coherence operations, reducing unnecessary invalidations and improving overall system performance in multi-core environments. This approach enhances the efficiency of cache coherence without adding significant complexity to the system.

Software Optimization Techniques

Software developers can optimize for cache coherency issues through several effective techniques. Padding involves inserting unused bytes between variables to prevent them from sharing the same cache line, minimizing false sharing. Data alignment techniques, like aligning data structures to cache line boundaries, help reduce cache contention. Thread-local storage avoids unnecessary sharing by allocating separate data for each thread, preventing false sharing altogether. Restructuring memory access patterns, such as switching from array-of-structures (AoS) to structure-of-arrays (SoA), can also improve memory locality

and reduce contention. Additionally, minimizing synchronization overhead by using efficient mechanisms like fine-grained locks or lock-free data structures further reduces cache coherency issues.

Beyond MESI: Advanced Coherency Protocols

While MESI is a foundational protocol, modern processors often implement extended variants such as MESIF (adding a "Forward" state) or MOESI (adding an "Owned" state). These extensions help optimize specific scenarios common in multiprocessor workloads, such as read-sharing or efficient handling of modified data.

MESIF Protocol

Intel's MESIF protocol represents a significant evolution in coherence design, particularly for snoop-based multiprocessor systems with distributed caches. As detailed by Goodman and Hum, MESIF addresses a critical inefficiency in the basic MESI protocol by designating a single cache as the responder for shared data through the Forward state [10]. This optimization eliminates redundant responses when multiple caches hold the same data in the Shared state. Their analysis demonstrates that MESIF significantly reduces both bandwidth consumption and latency for read requests to shared data by ensuring only one cache responds while others remain silent. This approach is especially beneficial in workloads with high read-sharing patterns common in server applications. The protocol maintains the memory system's conceptual simplicity while improving scalability in systems with complex interconnection networks.

MOESI Protocol

AMD's MOESI protocol takes a different approach by adding an "Owned" state, which allows a cache line to be both modified and shared without writing back to the main memory. The Owned state elegantly addresses a common performance bottleneck in multiprocessor systems. According to Amit et al., this additional state reduces the need for write-backs to memory when multiple processors are reading data that one processor has modified [1]. Their analysis shows that MOESI creates a clear performance advantage for producer-consumer patterns by eliminating redundant memory traffic. By allowing direct cache-to-cache transfers of modified data without memory updates, the protocol effectively reduces memory bandwidth pressure and latency for shared writable data. This optimization proves particularly beneficial in workloads with high data sharing characteristics.

Token Coherence

Token Coherence represents an innovative approach to cache coherency that decouples performance from correctness concerns. As proposed by Martin et al., this protocol assigns a fixed number of tokens to each memory block, requiring a processor to collect all tokens to write and at least one token to read [11]. This elegant scheme naturally prevents race conditions without complex ordering constraints. Token Coherence achieves high performance through fast, speculative requests that operate without sequencing overhead, while maintaining correctness through a separate persistent request mechanism that resolves potential starvation. Their evaluation demonstrates substantial performance improvements, showing 15-28%

speedup over snooping protocols and 17-54% speedup over directory protocols by enabling direct cache-to-cache transfers without indirection through home nodes.

Table 3: Comparison of Cache Coherence Protocol Features and Characteristics [1, 10, 11]

Feature	MESI	MESIF	MOESI	Token Coherence
Shared State Behavior	Memory supplies data	Forward cache supplies data	Owned cache can supply	Any cache with tokens can supply data
Extra State Purpose	N/A	**F**: Designated forwarder	**O**: Modified but shared	Tokens manage ownership and validity
Scalability	Poor beyond 8–16 cores	Improved read handling	Slightly better	High scalability
Complexity	Low	Moderate	Moderate	High (needs token tracking, ordering)
Broadcasting	Yes	Reduced via F state	Yes	No (point-to-point messages)
Used In	Intel CPUs (legacy), basic systems	Intel QuickPath Interconnect (QPI)	AMD Opteron, some ARM cores	Research prototypes, academic systems
Memory Writeback	Required before sharing	Same as MESI with forwarder assist	Can skip writeback using Owned	Often avoided if tokens stay in caches
Main Strength	Simplicity	Efficient Shared reads	Reduced writebacks	High scalability and flexibility
Main Weakness	Not scalable	Still relies on broadcast	More complex state machine	Message overhead, complex protocol

Hybrid and Adaptive Approaches

Traditional coherence protocols like MESI use fixed strategies—typically write-invalidate or write-update—for maintaining cache consistency. However, these static approaches can lead to suboptimal performance depending on workload characteristics. **Hybrid** and **Adaptive protocols**, like the one

proposed by Chtioui et al., combine multiple coherence strategies, dynamically selecting the most efficient one based on runtime behavior. Their Dynamic Hybrid Cache Coherency Protocol for Multi-Processor System-on-Chip (MPSoC) architectures switches between update and invalidate schemes depending on memory access patterns, using the update protocol for frequent cache updates and switching to invalidate when updates are less frequent. This dynamic adaptation reduces cache misses, energy consumption, and execution time, thereby improving system performance [12]. Similarly, Stenstrom et al. proposed an adaptive cache coherence protocol optimized for migratory sharing, further improving cache efficiency in systems with frequent data transfers [13]. This protocol specifically targets workloads where data is repeatedly transferred between processors, minimizing unnecessary invalidations and reducing memory latency.

CONCLUSION

Cache coherency continues to evolve as processors advance toward higher core counts and more complex memory hierarchies. The progression from basic MESI protocols to sophisticated variants like MESIF, MOESI and token coherence reflects the ongoing effort to balance consistency requirements with performance demands. While coherency mechanisms inevitably introduce overhead, thoughtful hardware design coupled with software optimization techniques can substantially mitigate these costs. The future of cache coherency likely lies in adaptive, hybrid approaches that dynamically adjust to workload characteristics, potentially incorporating predictive elements to transform coherence from reactive to proactive. As computing becomes increasingly parallel, understanding and optimizing cache coherency remains essential for extracting maximum performance from modern architectures while maintaining the programming simplicity that coherent shared memory provides.

REFERENCES

1. Amit Joshi et al., "Performance Analysis of Cache Coherence Protocols for Multi-core Architectures: A System Attribute Perspective," ResearchGate, pp. 631-635, 2016. [Online]. Available: https://www.researchgate.net/publication/310360949_Performance_Analysis_of_Cache_Coherence_Protocols_for_Multi-core_Architectures_A_System_Attribute_Perspective
2. Sciencedirect, "Main Memory Access," Computer Science, 2025. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/main-memory-access
3. Daniel Molka, Daniel Hackenberg, Robert Schöne, "Main memory and cache performance of intel sandy bridge and AMD bulldozer," MSPC '14: Proceedings of the workshop on Memory Systems Performance and Correctness, 2014. [Online]. Available: https://dl.acm.org/doi/10.1145/2618128.2618129
4. Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin, "Why on-chip cache coherence is here to stay," *Communications of the ACM*, 2012. [Online]. Available: https://dl.acm.org/doi/10.1145/2209249.2209269

5. Vijay Nagarajan et al., "A Primer on Memory Consistency and Cache Coherence, Second Edition": Synthesis Lectures on Computer Architecture (SLCA), 2020. [Online]. Available: https://link.springer.com/book/10.1007/978-3-031-01764-3
6. D.J. Sorin et al., "Specifying and verifying a broadcast and a multicast snooping cache coherence protocol,"IEEE Transactions on Parallel and Distributed Systems, 2002. [Online]. Available: https://ieeexplore.ieee.org/document/1011412
7. Daniel Lenoski, James Laudon, Kourosh Gharachorloo, "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor," *Computer Systems Laboratory Stanford University, CA 94305*, 1990. [Online]. Available: https://people.eecs.berkeley.edu/~kubitron/cs258/handouts/papers/p148-lenoski.pdf
8. Daniel Molka et al., "Cache Coherence Protocol and Memory Performance of the Intel Haswell-EP Architecture," *2015 44th International Conference on Parallel Processing*, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7349629
9. Rui Zhang et al., "Neat: Low-Complexity, Efficient On-Chip Cache Coherence,"arXiv:2107.05453v2, 2021. [Online]. Available: https://arxiv.org/pdf/2107.05453
10. J.R. Goodman H.H.J. Hum, "MESIF: A Two-Hop Cache Coherency Protocol for Point-to-Point Interconnects (2009)," 32nd International Symposium on Computer Architecture (ISCA'10), 2009. [Online]. Available: https://www.cs.auckland.ac.nz/~goodman/TechnicalReports/MESIF-2009.pdf
11. M. R. Martin, M. D. Hill, and D. A. Wood, "Token Coherence: decoupling performance and correctness," 30th Annual International Symposium on Computer Architecture, 2003. Proceedings. 2003. [Online]. Available: https://ieeexplore.ieee.org/document/1206999
12. Hajer Chtioui et al., "A Dynamic Hybrid Cache Coherency Protocol for Shared-Memory MPSoC," 2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, 2009. [Online]. Available: https://ieeexplore.ieee.org/document/5349988
13. Per Stenstrom, Mats Brorsson, and Lars Sandberg, "An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing," Department of Computer Engineering, Lund University, 1993. [Online]. Available: https://people.eecs.berkeley.edu/~kubitron/courses/cs258-S02/handouts/papers/p109-stenstrom.pdf