Cache Coherence for Embedded Multi-core System Architectures: A Survey and Challenges

M. Thillai Rani, R. Rajkumar, K. P. Sai Pradeep, M. Jaishree, and S. TamilSelvan

Abstract Cache coherency refers to the ability of multiprocessor system cores to share the same memory structure while maintaining their separate instruction caches. Cache coherency is used in coherence protocols to maintain data consistency between cache memory in multiprocessor systems. All cores have the same design, share same main memory (MM) and have their own cache memory. Whenever a core requests a block of data from MM for its cache, it needs a protocol to broadcast the status of blocks in MM and cores. Various hardware and software-based cache coherent mechanisms including contemporary protocols, have been thoroughly explored. This survey focuses on analyzing the different cache coherence techniques used in SoC devices. With a variety of cache coherence techniques to choose from, the best strategy is determined by a number of factors such as latency, scalability and so on.

Keywords Cache coherence · Protocols · Main memory · Coherence mechanism and SoC

M. Thillai Rani
Department of ECE, Sri Krishna College of Technology, Coimbatore, Tamilnadu, India
e-mail: thillairani.m@skct.edu.in

R. Rajkumar (✉)
Department of ECE, Vel Tech Rangarajan Dr.Sangunthala R&D Institute of Science and Technology, Avadi, Tamilnadu, India
e-mail: rajkumarramasami@gmail.com

K. P. Sai Pradeep
Department of ECE, Dr. N.G.P Institute of Technology, Coimbatore, Tamilnadu, India
e-mail: saipradeep@drngpit.ac.in

M. Jaishree
Department of ECE, Sri Ramakrishna Engineering College, Coimbatore, Tamilnadu, India
e-mail: jaishree.m@srec.ac.in

S. TamilSelvan
Department of CSE, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Science, Chennai, India

P. P. Joby et al. (eds.), *IoT Based Control Networks and Intelligent Systems*,
Lecture Notes in Networks and Systems 528,
https://doi.org/10.1007/978-981-19-5845-8_49

1 Introduction

Cache coherence techniques have a huge influence on the performance of a centralized and distributed shared memory of multi-core systems architecture [1]. Correct execution happens when any one of the two cores perform an instruction to read a value from variable "a." Core 2 must see the changed value if core 1 conducts a store instruction that alters variable "a," and core 2 then performs a load instruction from that variable. As a result, the new value must be transmitted to core 2's copy of variable "a." This is known as the cache coherence problem shown in Fig. 1.

Consider two systems interacting in a shared-memory paradigm. They may navigate a shared address-based namespace domain, perform direct read and write operations on places inside the region, and hence transfer data through different addresses.

In a single processor context, read and write operations are primarily used for interprocessor communication. As the shared-memory model is a simple and straightforward expansion of the single processor, numerous devices need be combined for calculations to generate correct results by ensuring cache coherence [2]. This cooperation is in addition to the SoC basic memory access transaction flow. These new transactions must also be handled differently.

An action that necessitates the invalidation of particular information from cache for example shown in Fig. 2, should broadcast to all other caches in the system. When read or write exchange with no coherency have D2D communication topologies, enforcing cache coherence for the similar communications results in difficult many-to-many communication topologies. This methodology demands the collection of specific coherence responses transaction by following multi-casting of the combined coherence output. All of these actions must be supported efficiently by SoC coherency connectivity [3]. The objective of this study is to explore cache coherence protocols and their challenges for multi-core systems.

A few frameworks are normally progressive, including frameworks contained various multicore chips. Inside every cores, there is an intra-chip convention as well as a convention across the chips. Coherency issues may be fulfilled by the intra-chip convention don't interface with the between chip convention; just when a solicitation

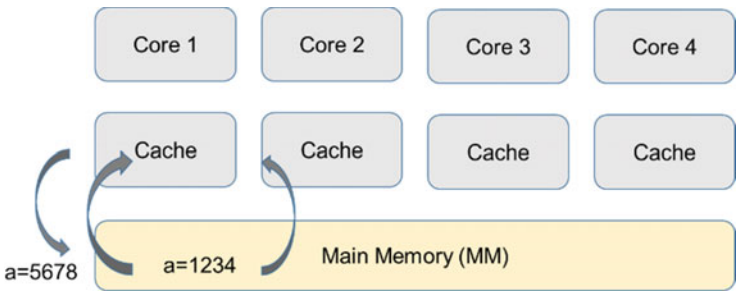

Fig. 1 Cache coherency addressed in multi-core system architecture

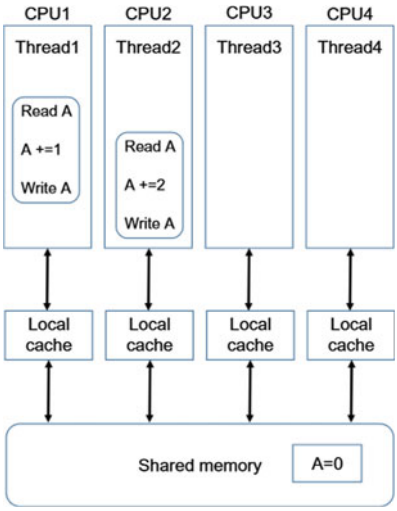

Fig. 2 Cache coherency addressed in shared memory architecture

can't be fulfilled by one more hub on the chip requests gets elevated between the chip convention. This decision of convention at each core is generally autonomous of the decision to next core. For instance, an intra-chip sneaking around convention can be made viable with off-chip index convention. Each chip needs a solitary registry regulator which believes that whole chip are connected with single hub in the registry convention. The off-chip registry convention could be indistinguishable from any registry conventions introduced in following section with index process normally addressed in a coarse design.

Another conceivable progressive framework is index conventions for inter-chip and intra-chip conventions. These conventions will be something very similar and unique when compared with each other. A benefit of progressive conventions for various leveled frameworks is that empowering plan of basic, possibly non-adaptable intra-chip plan for production. Once it is processed, it is useful to plan a solitary convention that scales to the biggest conceivable amount of centers which can exist in a framework. Such a convention is probably going to be needless excess in huge majority of frameworks contained a single chip.

This paper is organized as follows. Section 2 begins with overview of cache coherence mechanisms. Section 3 explains the protocols to maintain cache coherence. Section 4 investigates the challenges in coherencies followed with concluding remarks in Section 5.

2 Cache Coherence Mechanisms

The underlying interconnection system is enhanced by cache coherence, which introduces a specific group of procedures and network traffic topologies. With the intention of maintaining coherency over MM and caches, it may be necessary to exchange some information to various entities in the system. The organization of cache coherence mechanisms is shown in Fig. 3.

For hardware-based approaches, because of the instruction count is unaffected by hardware, it's tempting to use it to judge processor performance. Many a computer designer has been stymied by such oblique performance indicators [4]. The temptation for evaluating memory hierarchy performance is to focus on miss rate because it is also unaffected by hardware speed. As we'll see, the miss rate is just as deceptive as the instruction count. The average memory access time (T_avg) is a strong metric of memory hierarchy efficiency given by,

$$T_{avg} = T_H + T_M.T_P \tag{1}$$

where, T_H, T_M *and* T_P are hit time, miss rate and miss penalty rate respectively.

Cache coherence mechanisms based on compilers analyze the code to identify which information are potentially dangerous for caching [5]. Snooping protocol and directory-based protocol are the two most used cache coherence techniques. Only a bus-based system can utilize the snooping protocol, which employs a number of states to identify whether there is necessity to update cache entries and control over write process to blocks. The directory-based protocol is scalable to multiple processors

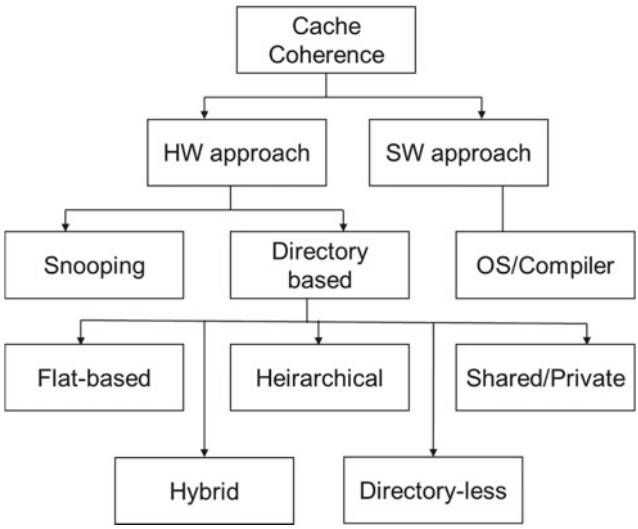

Fig. 3 Classiifcation of cache coherence

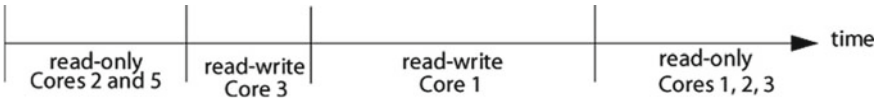

Fig. 4 Partitioning memory locations into iterations or epochs

or cores since it may be used on any network. Snooping, on the other hand, is not scalable [6].

In this architecture, a directory is utilized to keep track of which memory addresses are shared across several caches and which are reserved for one core's cache alone. Snooping, on the other hand, is not scalable. In directory-based method, a directory is utilized to keep track of which memory addresses are shared across several caches also may reserve for single cache. The directory is aware when a block needs to be updated or invalidated [5].

One method is to utilise a cache coherence technique that is based on invalidation. This method tackles the cache coherence problem by requiring that whenever a core asks to write to a cache block, the core must invalidate (delete) the block's copy from any other core's cache that has the block. The requesting core now owns the lone copy of the cache block and has complete control over its contents. When any other core tries to read the block later, it will encounter a cache miss and will have to retrieve the updated data from the core that modified it. Figure 4 demonstrates the iterative partitioning of memory locations.

Shared memory is supported in hardware by many processors and multi-core processors as well. Each of the processor cores in a shared memory system may read and write to a single shared address space. The memory consistency models specify the architectural and observable behaviour of a shared memory machine's memory system. Read (load) and write (store) operations can affect memory defined by reliability characterizations. Numerous mechanisms implement cache coherency methods to assure that multiple copies of cached data are maintained at present as part of providing a memory consistency model.

2.1 Distributed Memory Architecture

Data must be explicitly transferred between jobs in the distributed memory architecture shown in Fig. 5. Synchronous send-receive semantics can be used to accomplish task synchronization. The receiving job is paused until the data for transmission is ready. Asynchronous message passing is also possible. Send-receive semantics are employed in this technique, and the receiving process checks or is told when data is ready without blocking. This allows processing and communication to overlap, resulting in considerable speed increases.

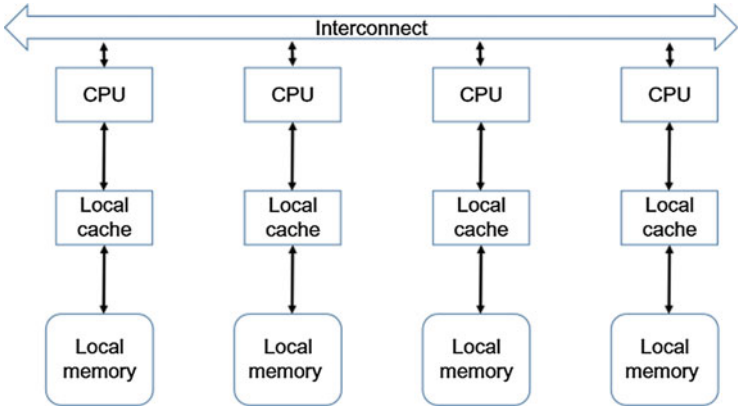

Fig. 5 Organization of cache coherence

In comparison to shared memory, the distributed memory architecture generally results in a greater communication cost (mostly in message creation and tear-down, as well as explicit data copying). As a result, message forwarding for both functions should be optimized. The fact that memory is estimated based on the number of processors is another significant benefit of the distributed paradigm. As a result, there will be an increase in the number of processors.

As a result, as the number of processors rises, so does the size of memory. Another benefit is that each CPU now has direct access to its own memory, free of interference from other cores and the expense of maintaining cache coherency. It is easier to employ commodity, off-the-shelf CPUs and networking using this method. The main downside of this design is that the programmer is now in charge of all data transmission aspects. Existing data structures based on a global memory layout may also be more challenging to translate to a distributed memory architecture.

2.2 Symmetric Multiprocessing Architecture (SMP)

Two or more identical processors are combined to particular, shared main memory in symmetric multiprocessing (SMP) systems. All I/O devices, such as UARTs and Ethernet, are accessible to SMP systems. Any processor on an SMP system may work on any job, irrespective of location where data for each operation is stored in memory. The system's tasks should not be running on two or more processors at the same time. Figure 6 gives the multicore SMP based architectures. Many ways have been proposed to improve the scalability of directories for multicore SMP. However, they often decrease directory memory cost by reducing coherence information that results in extra unneeded coherence messages and, as a result, energy wasted, performance deterioration, and lack of scalability.

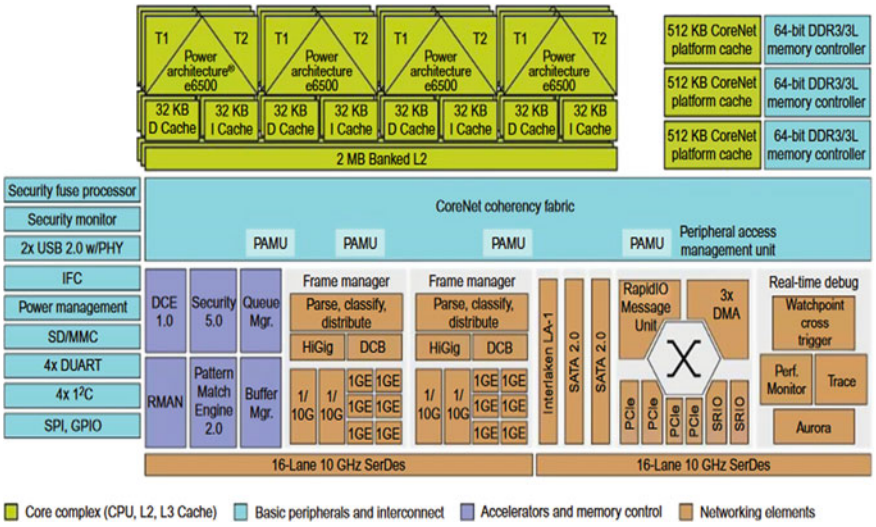

Fig. 6 Detailed structure for multicore architectures

2.3 Asymmetric Multiprocessing Architecture (AMP)

All CPUs are not handled equally in an AMP systems shown in Fig. 7. First and foremost, the CPUs do not have to be identical; alternative core architectures and instruction sets can be used. Certain processors in AMP systems can be dedicated to I/O activities with certain peripherals. This sort of CPU specialisation has the potential to improve system performance.

2.4 Maintaining Cache Coherence

The preceding section's coherence invariants give some insight into how coherence protocols function. The great majority of coherence protocols, known as "invalidate protocols," are built with these invariants in mind. If a core reads a memory location, it transmits information to cores to determine present value of that particular memory location by ensuring that no other cores having cached read–write replicas of that same memory location. With these messages, any active read–write epoch iteration is terminated, by initialising read-only operation. When the particular core decides to write an information to any memory locations, it transmits a message to other cores to get the memory location's present value. It does not have a definite read-only cached copy, and to ensure that no other cores have read-only or read–write cached copies of the memory location.

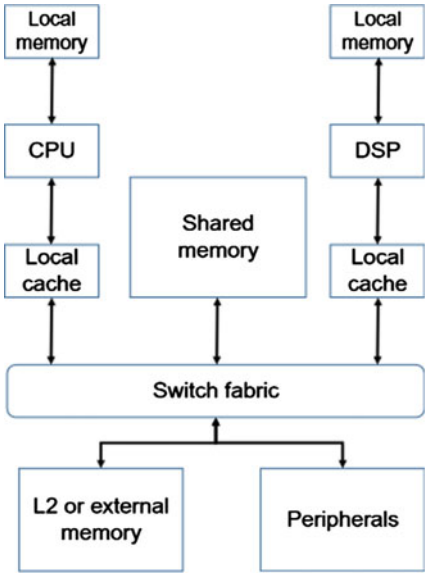

Fig. 7 Structure of asymmetric multiprocessing architecture

2.5 Directory Based Cache Coherence

This is a cache coherence protocol system that does not employ the broadcasting technique, therefore it must keep all of the cached information of every block in the shared data, whether it is centralized or dispersed across several processors [7]. Every memory block has a directory entry, which is the name of the structure that keeps all of the information about the shared block's various locations. Depending on the current state of the directory and transactions, the directory-based cache will take any action. The directory-based protocol must also be disseminated across the nodes. Each nodes in interconnection networks has blocks of local memory that are always associated with local cache and directory in the cache coherent non-uniform memory access. Distributing the directory-based protocol have benefits of reducing bandwidth difficulties and potential bottlenecks. Cache coherence techniques based on directories offer the ability to grow shared memory multiprocessors to huge numbers of processors. An important advantage of directory based protocols is the effective scaling over snoopy protocols. The significant characteristic of directory protocols is the ability to exploit arbitrary point-to-point interconnects.

2.6 Snooping Cache Coherence

Snooping method is used for write-invalidate and write-update protocols. Each cache listens in on bus transactions to observe what other processors are doing in memory,

and thus requires the employment of a broadcast media in the machine. One of the major advantages of snoopy protocols is the average cache-cache miss latency is very low. The bandwidth required to broadcast messages to all processors is limited by cache coherence overhead and performance of shared buses. Another issue with snoopy protocols is their inefficiency in terms of power consumption. Each cache controller has the ability to "snoop" on the network in order to detect and respond to these broadcasted notifications. Snoopy protocols are particularly suited to a bus-based multiprocessor because the shared bus provides a direct method for broadcasting [8].

3 Cache Coherence Protocols for Multi-core System Architectures

3.1 MSI Protocol

The MSI protocol detects when a cache line has been modified (M), indicating that the cache block has been changed and the data in the cache varies from data in backup repository [9]. The modified block cache is in control over updating underlying store and read data, but it will not share or send any information externally.

A block that has not been updated and is read-only in at least one cache is referred to as shared (S). Without first updating the underlying store, the cache can get the data. Invalid (I) state indicates that the block is invalid and unavailable, so it is invalidated by bus request in the present cache, requiring it to be retrieved from memory. The connection between the cache and the backup store keeps the states S, M, and I active.

3.2 MESI Protocol

MESI protocol is also termed as Illinois protocol as it was developed in University of Illinois [13–16]. This is well-known protocol that incorporates a write-back cache. Even though MESI is the extension of MSI protocol, two transitions for each write operation is processed and there is no sharing in data blocks. In the first epoch, the memory block is put into shared state, and in the second epoch, the status of data blocks are changed from modified to shared state. It includes a new Exclusive (E) state to the MSI protocol, which saves bandwidth use by writing to a shared data block.

3.3 MOSI Protocol

MOSI is a variation of MSI protocols. The additional state is called as Owned state [11] which is accurate and has the present copy of data is obtainable as the cache line is in this state. The owned state is related to the shared data memory and same as changed state since main memory may have an expired backup of the information. Only one cache may be in the owned state at a time, with all other caches holding the data in the shared state. It converts into shared state after writing by updating the MM.

3.4 MOESI Protocol

MOESI protocol [12] is classified into five states. The data update and data sharing is denoted by Owned (O) state shown in Fig. 8. This eliminates the requirement for changed data to be written back to main memory before being shared.

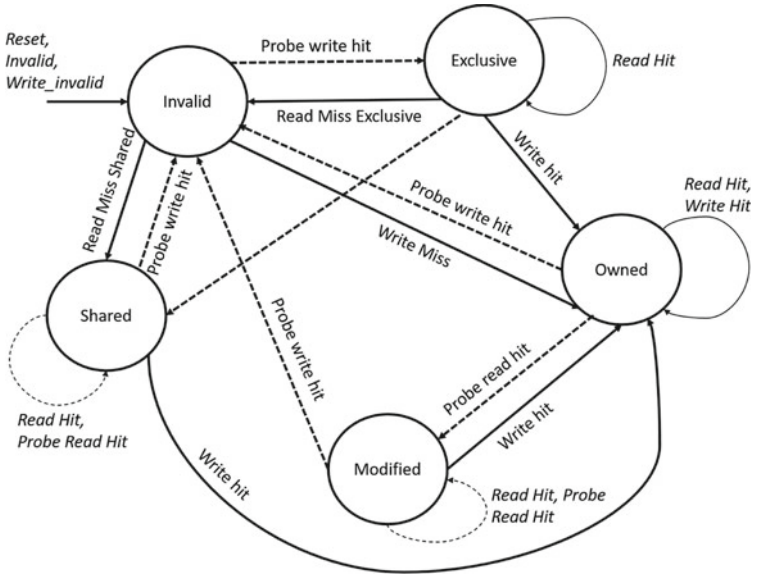

Fig. 8 MOESI protocol

3.5 *Firefly Protocol*

Firefly algorithm involves in three states such as Valid-Exclusive, Shared and Dirty. The cache block is valid, clear, and unique in that it only exists in one cache. The cache block is legitimate, clean, and may be found in several caches [16]. The block is the sole duplicate of the memory, and it is dirty, meaning that its value has changed since it was brought from MM. The only condition that causes a write-back is the replacement of block in the cache.

3.6 *Dragon Protocol*

When a write operation to a cache block is followed by many read operation by processor, update-based protocols like the Dragon protocol [17] perform well because the updated cache block is quickly accessible over every processors. It involves in four states which are as follows. The present processor was the first to fetch the cache block, and no other processor has accessed it subsequently. The cache block is very certainly present in many processor caches.

The protocol keeps states Exclusive (E) and Shared clean (Sc) distinct to prevent read–write operations on non-shared cache blocks from causing bus transactions and so slowing down the execution. In shared modified (SM), the block persists in many processor caches, with the present processor being the latest to alter it. As a result, the present processor is referred to as the block's owner. Unlike invalidation protocols, the block only has to be updated in the processor, not the MM. When a cache block is evicted, the processor is responsible for updating the main memory. In single-threaded applications, this is a regular event. Finally, modify (M) the value from MM.

3.7 *MECSIF Protocol*

MECSIF is a custom established hybrid cache coherence method that employs both directory and snoopy protocols [18]. It involves in seven states including clean (C) and forward (F) states. If a copy of a data block has never been updated and at least one cache contains a copy of the data block, the processor will only respond to a remote processor request with a copy of the data block in its current state.

4 Challenges in Cache Coherency

Though the protocol discussion appears to be straightforward, the implementation is fraught with difficulties. The protocol's main flaw is that it assumes operations are atomic—that is, that an operation may be performed without any intervening actions. For example, the given protocol implies that write misses may be detected, the bus acquired, and a response received in a single atomic operation. This is not the case in reality. Similarly, read misses would not be atomic if we employed a switch, as many current multiprocessors do. Non-atomic actions increase the likelihood that the protocol may deadlock, or reach a point where it will be unable to proceed.

Any centralized resource in the system can become a bottleneck as the number of processors in a multiprocessor increases, or as each processor's memory demands increase. The bus and memory form a bottleneck in the simple case of a bus-based multiprocessor. As a result, scalability becomes a problem. Designers have employed numerous buses as well as interconnection networks, such as crossbars or tiny point-to-point networks, to boost the communication bandwidth between CPUs and memory. The memory system can be divided into many physical banks in such architectures to increase the effective memory bandwidth while maintaining a consistent memory access time. Both hardware and software is always seen as integrated and interdependent in multicore systems.

With growing complications, the resulting logical steps involved in shifting and transforming parallel hardware and software via plenty of programming models, as well as programmable processors that specify operating on multiprocessor systems-on-chips (MPSoCs) [19]. As of now, design of multicore processors is getting higher difficulties, forcing the manufacturers move ahead to block-level design, which separate the design of sub-system blocks from design of SoC platform also keeping power problems in mind. The technological aspects has enhanced the electronics hardware part to enable for verification and automated synthesis of gates from gate designs to transistors levels and eventually to register-transfer level (RTL) design [20, 21].

Abstraction, in combination with automation, translations, and validation in contradiction of the resulting lowest level of abstractions have traditionally been the response to escalating complexity, resulting in increased silicon real estate in prior decades. Both hardware and software must be viewed as integrated in multicore designs. Even the best hardware multicore architectures in industry will perform badly if only inadequate group of programmers can program to accomplish tasks. The most imaginative algorithm, on the other hand, will not run as planned if the underlying hardware design lacks sufficient computation, storage, and communication resources. The following are the common problems addressed in cache coherency mechanisms:

1) Detecting certain sharing patterns in order to improve coherence. We may use modified adaptability coherence mechanisms to increase system performance by exploiting application-level asymmetrical behaviours, such as application access patterns at application-level granularity.

2) Workload heterogeneity is taken into account while designing coherence protocols. We can change the operating system to lead coherence protocol activities and transition across protocols.
3) Adapting area-effective directory architecture to assure scalability of on-chip directory storage.For these goals, we can use hierarchical and tag-less design alternatives, for example.
4) Optimizing data placement strategy in a multi-core cache system to reduce distant cache visits by bringing private data closer to the "home core" and reducing data migration operations in the coherence protocol.

5 Conclusion

The different coherence maintenance strategies exploited in multi-cores are investigated in this survey. Cache coherence is expected to be phased away soon, according to few researchers, because it raises hardware costs by storing more state, transmitting more messages, and verifying that everything is correct.

However, we envisage that coherence will endure popular since the software cost of dealing with incoherence is always high and is recognized by few established design engineers rather than developers who should deal with it. This work aids researchers in comprehending coherence processes and investigating the implementation issues provided by the rapidly expanding number of cores. Despite significant progress in this field, it remains a very active study topic. Many research areas exist, such as protocol correctness verification, performance assessment, comparison, directory size, and protocol overhead minimization that has to be looked at in the future.

Acknowledgements The authors would like to thank reviewers for their valuable comments and suggestions.

References

1. Ros A, Acacio ME, Garcia JM (2010) A direct coherence protocol for many-core chip multiprocessors. IEEE Trans Parallel Distrib Syst 21(12):1779–1792. https://doi.org/10.1109/TPDS.2010.43
2. Joshi AD, Ramasubramanian N (2015) Comparison of significant issues in multicore cache coherence. In: 2015 international conference on green computing and internet of things (ICGCIoT), pp 108–112. https://doi.org/10.1109/ICGCIoT.2015.7380439
3. Al-Waisi Z, Agyeman MO (2017) An overview of on-chip cache coherence protocols. In: 2017 intelligent systems conference (IntelliSys), pp 304–309. https://doi.org/10.1109/IntelliSys.2017.8324309
4. Jang YJ, Ro WW (2009) Evaluation of cache coherence protocols on multi-core systems with linear workloads. In: 2009 ISECS international colloquium on computing, communication, control, and management, pp 342–345. https://doi.org/10.1109/CCCM.2009.5267596

5. Kaushik AM, Hassan M, Patel H (2021) Designing predictable cache coherence protocols for multi-core real-time systems. IEEE Trans Comput 70(12):2098–2111. https://doi.org/10.1109/TC.2020.3037747
6. Tomasevic M, Milutinovic V (1992) A simulation study of snoopy cache coherence protocols. In: Proceedings of the twenty-fifth Hawaii international conference on system sciences, vol 1, pp 427–436. https://doi.org/10.1109/HICSS.1992.183192
7. Mittal S, Nitin (2014) A new approach to directory based solution for cache coherence problem. In: 2014 3rd international conference on eco-friendly computing and communication systems, pp 9–13. https://doi.org/10.1109/Eco-friendly.2014.77
8. Bhardwaj K, Havasi M, Yao Y, Brooks DM, Lobato JMH, Wei G (2019) Determining optimal coherency interface for many-accelerator SoCs using Bayesian optimization. IEEE Comput Archit Lett 18(2):119–123. https://doi.org/10.1109/LCA.2019.2910521
9. Fuchsen R (2010) How to address certification for multi-core based IMA platforms: current status and potential solutions. In: 29th digital avionics systems conference, pp 5.E.3-1–5.E.3-11. https://doi.org/10.1109/DASC.2010.5655461
10. Patel, Ghose K (2008) Energy-efficient MESI cache coherence with pro-active snoop filtering for multicore microprocessors. In: Proceeding of the 13th international symposium on low power electronics and design (ISLPED 2008), pp 247–252. https://doi.org/10.1145/1393921.1393988
11. Yang Q, Bhuyan LN, Liu B (1989) Analysis and comparison of cache coherence protocols for a packet-switched multiprocessor. IEEE Trans Comput 38(8):1143–1153. https://doi.org/10.1109/12.30868
12. Li S, Guo D (2017) Cache coherence scheme for HCS-based CMP and its system reliability analysis. IEEE Access 5:7205–7215. https://doi.org/10.1109/ACCESS.2017.2701406
13. Martin MMK, Hill MD, Wood D (2003) Token coherence: decoupling performance and correctness. In: 30th annual international symposium on computer architecture, 2003. Proceedings, San Diego, CA, USA, pp 182–193
14. Sun S, An H, Chen J (2014) Cache coherence method for improving multi-threaded applications on multicore systems. In: 2014 6th international conference on multimedia, computer graphics and broadcasting, Haikou, pp 47–50
15. Ahmed RE, Dhodhi MK (2011) Directory-based cache coherence protocol for power-aware chip multiprocessors. In: 2011 24th Canadian conference on electrical and computer engineering (CCECE), Niagara Falls, ON, pp 1036–1039
16. Kaur DP, Sulochana V (2018) Design and implementation of cache coherence protocol for high-speed multiprocessor system. In: 2018 2nd IEEE international conference on power electronics, intelligent control and energy systems (ICPEICES), Delhi, India, pp 1097–1102
17. Lametti S (2010) Cache coherence techniques, A Technical report
18. Li J et al (2011) A new kind of hybrid cache coherence protocol for multiprocessor with D-cache. In: 2011 international conference on future computer science and education, pp 641–645. https://doi.org/10.1109/ICFCSE.2011.160
19. Babu P, Parthasarathy E (2021) Reconfigurable FPGA architectures: a survey and applications. J Inst Eng Ser (B) 143–156
20. Durai PM (2019) Enhanced network performance and mobility management of IoT multi networks. J Trends Comput Sci Smart Technol (TCSST) 1(02):95–105
21. Krishnaraj N, Smys S (2019) A review of multi homing and its associated research areas along with internet of things (IOT). IRO J Sustain Wirel Syst 1(1):69–76