Cache Coherence Protocols in Distributed Systems

Abstract

Distributed systems performance is affected significantly by cache coherence protocols due to their role in data consistency maintaining. Also, cache coherent protocols have a great task for keeping the interconnection of caches in a multiprocessor environment. Moreover, the overall performance of distributed shared memory multiprocessor system is influenced by the used cache coherence protocol type. The major challenge of shared memory devices is to maintain the cache coherently. Therefore, in past years many contributions have been presented to address the cache issues and to improve the performance of distributed systems. This paper reviews in a systematic way a number of methods used for the cache-coherent protocols in a distributed system.

Keywords: *Cache coherence protocol, Distributed System, Multiprocessor system, Shared memory multiprocessor system.*

I. Introduction

The separated design of distributed systems from the core network makes the worst-case assumptions [1]. However, in deploying them through distributor applications today in data centres, it is possible to participate in the design of distributed systems with their own network layer but it offers essential benefits [2], [3]. The displaying operating systems problem in distributed systems suggests new solutions to current problems [4], [5]. We can alleviate the problem by maintaining a table of trusted contact canters in each operating system and making all IPI processors verify sources [6], [7]. Distributed systems permit several clients from accessing a common computing source thus delivers resource sharing [8]–[10]. Air traffic control, online railway reservation systems, and internet banking are examples of such distributed computing [11]–[13].

One of the most commonly used parallel programming models is shared memory due to the advantages of global address space [14], [15]. The major challenge of shared memory devices is to maintain the cache coherent, typically addressed by the hardware cache coherent protocols [16]. coherent failure occurs when updating the local node cache copy and revoking all shared copies to keep the data coherent [17], [18].

The most commonly used cache coherence protocols are SI, MI, MSI, MESI, MOSI, and MOESI. The first one where releasing storage is not allowed by the cores. The second protocol (MI) considered the simplest one in use to maintain cache coherence in MC / MP systems. The third one the MSI protocol is the simplest of protocols based on deactivation. MESI which supports both write cache 1 and 2. MOESI protocol where it reduces the number of bus messages [19], [20].

The main objective of this article is to review the last researches on the cache coherence protocols in distributed system. In addition, to systematically summarize the numerous works which have achieved in the last few years ago. Section 2 provide details explanation of the cache coherence protocols. Section 3 gives a literature review about the cache coherence protocol in a distributed system and presents a comparative analysis of the surveyed methods. Section 4 is a discussion of the surveyed methods in the cache coherence protocols of a distributed system. The conclusion of the study is presented in section 5.

II. Cache Coherence Protocols

A cache is a small sized and high-speed memory that caches data from some of the frequently used addresses in the main memory. There are two categories of the cache coherence systems (directory-based and snoopy-based). The directory-based schemas maintain a central directory to store the memory block sharing state. In snoopy-based schemas to maintain consistent data, the request broadcasting and activity monitoring of the memory bus is done by the cache controller [16].

A cache coherent protocol is one way to maintain the interconnection of caches in a multiprocessor environment using hardwar [18]. Cache consistency issues cause another type of "cache error" coherence loss plus binding, conflict, and capacity errors. The coherence failure occurs in case of updating the local node cache copy and canceling all shared copies to maintain data consistency. The most common types of cache coherent protocols are SI, MI, MSI, MESI, MOSI, MEOSI, and MESIF [20].

A. *Shared-Invalid (SI)*

The SI is a parsed version of the MSI protocol, where releasing storage is not allowed by the cores. The system that has n cores, the correct general state allows the system to store cache blocks in any m centers in I and block the cache in other n - m centers in the S state. The entire space of the SI protocol state with cores n is hypercube1 dimensions [20].

B. *Modified-Invalid (MI)*

The MI protocol considered the simplest one in use to maintain cache coherence in MC / MP systems. This protocol uses two conditions: Modify (M) and Invalid (I). It has the least hardware complexity among all coherence protocols. Reading/writing in one node invalidates the data in another node if the same data exists [17].

C. *Modified -Shared-Invalid (MSI)*

The MSI protocol is the simplest of protocols based on deactivation. It consists of only 3 cases modified (M), shared (S), and invalid (I). Fig. 1 illustrates the MSI protocol diagram. The invalid case is where the local cache copy does not contain a valid copy. The shared state means that the local cache and other caches may contain a valid copy in relation to the main memory [19].

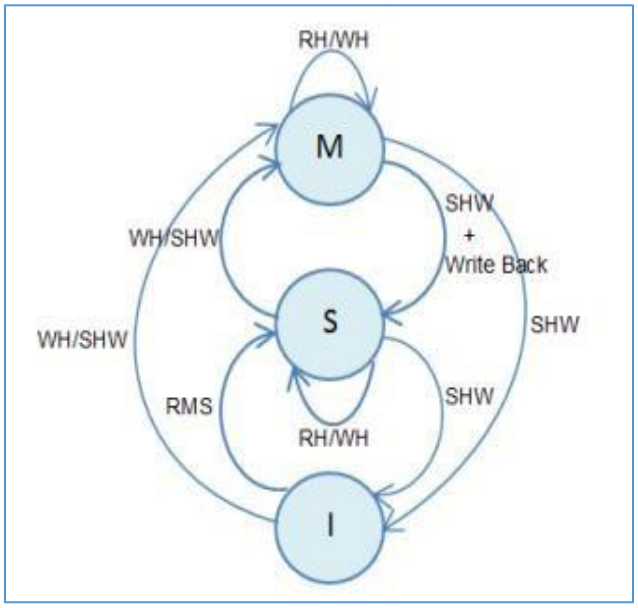

Fig 1: State diagram of the MSI protocol [21]

D. *Modified-Exclusive-Shared-Invalid (MESI)*

The MESI protocol considered one of the most commonly used cache coherence protocols, which supports both write cache 1 and 2 [19]. This protocol is an advance of the MSI protocol that has added a special case (E) to reduce the number of mobile carriers sending the incorrect fix to the rate. The exclusive case is where the local cache contains a valid copy [16]. Fig. 2 illustrates the state transition of the MESI protocol.

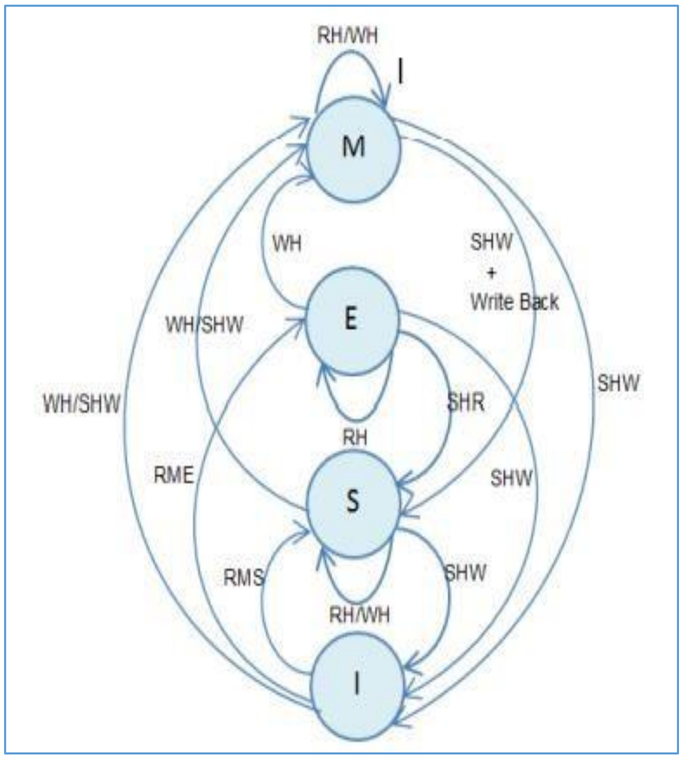

Fig 2: State diagram of the MESI protocol [22]

E. *Modified-Owned-Shared-Invalid (MOSI)*

In the MOSI, the transport node transfers the converted data to a read requestor without rewriting it. In the requesting node and the responding node, the cache line state is updated to S and O respectively. The status O permits the dirty copy to be shared between various nodes. It is omitted via combining the states of the MESI and MOSI in the Modified-Owned-Exclusive-Shared-Invalid protocol (MOESI) [23].

F. *Modified-Owned-Exclusive-Shared-Invalid (MOESI)*

The additional extension of the MESI protocol is the MOESI protocol where it minimizes the number of bus messages sent for an invalid transition to a rate while still allowing multiple participants. This protocol has another state called Owned that has a valid copy of the local cache [19]. Fig. 3 illustrates the transition diagram of the MOESI case.

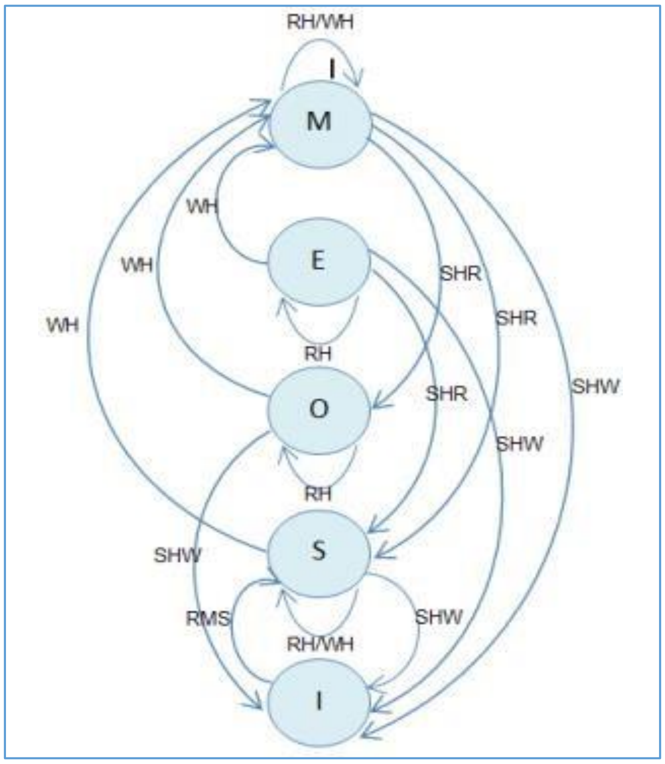

Fig 3: State diagram of the MOESI protocol [22]

III. Literature Review

Bernard et. al. [24] proposed a native protocol for network cache correlation (IN-CC) with linear scalability. Moreover, expand the protocol with several improvements: mechanisms for blocking both message routing and resource reservation cycles, supporting MOESI instances and accelerating data transfers by dividing the network into two parallel segments. Cache integration in shared memory greatly simplifies its programmability but faces scalability and cost problems. Finally, they recognize the implementation of the promote protocol on FPGA for the purpose of performance measurements and validation.

Quang and Do [25] focused on exploring MOESI, a well-defined cache coherence protocol common in CMP. The cause of CMP performance is strongly influenced by the extent to which data is fetched from the memory system. Our experience is based on the Splash-2 standard, which is widely used in every publication related to CMP design. The results of the experiment show that by rearranging the range of addresses of memory banks, the ratio of access to L2 can be improved to 13.5%.

Almakdi et al. [26] implemented snoopy and directory protocols and measured the entry rate, mandatory error rate, amplitude loss rate and coherent strength for each one. Solved the problem of the specific time value of data time based on processor instructions executed each clock cycle with corresponding memory access. Additionally, they explained how each map is affected by block size and cache size. The objectives are to explore the characteristics of consistency mechanisms that include custom caches by searching for two common types of mechanisms.

Team et al. [27] implemented a robust MESI protocol designed specifically for Verilog directory-based caching protocol. The increasing complexity of the coherence protocol and network on a chip has become a major challenge for pre-silicon verification. Adding Inferno to the verification process can reduce the time and effort that verification engineers need to detect and locate potential design errors. Also, summarize user experience with Inferno and offer suggestions to both Inferno users and developers.

Aghaei et al. [28] evaluated the latency and power of a multiprocessor system with several cache coherence protocols. They used the GEM5 simulation for implementing their system. The access latency increased because of the large number of processors with shared memory. Moreover, five injection rates (0.1, 0.2, 0.3, 0.4 and 0.5) were depended for traffic generation. The experimental based on measuring the characters of power, latency, and empirical result. The experimental result presented that the maximum power consumption and latency was performed by MOESI-CMP.

Kehagias and Raptis [23] purposed to teach and learn cache cohesion in advanced computer engineering courses. The problem of maintaining data consistency between memory and all caches is known as a cache cohesion problem. This work is a continuation of the previous desktop application for the MESI cache cohesion simulator. Simulation provides an interactive communication with the students; it is implemented in Visual Studio IDE and Unity Engine by using scripts in C#.

Kehagias [29] developed two simulation tools that are supported to the learning and teaching of the MESI cache coherence protocol and dynamic scheduling using the Tomasulo algorithm. Furthermore, introduced this simulation and assess its impact on the learning process. This simulation will evaluate its effect on the learning process. Results are presented in terms of quality and quantity.

Smelt [30] illustrated Intel x86 CPU networking from bus to point-to-point interfaces, was increasingly evident. The underlying causes of energy efficiency and their impact on the development of microspheres. The researcher examined the performance, power, and space scalability on some processors by using the Snooper x86 PC architecture emulator. The scalability between the bus topology and loops (NoC) and different networks (NoC) in Haswell's simulated architecture is compared by Snooper results, which comprise the power and region model by McPAT.

Eltaras et al. [31] proposed a novel scheme control to enhance the performance of a cache coherent network based on the adaptive routing algorithms. They aimed to reuse virtual channels among short and long packet. Moreover, the authors split the long packets into several chunks and allocated virtual channels to chunk and the entire packet. The experiment results illustrated that the performance of the routing algorithms increased and outperformed routing function in terms of flexibility. Besides, the presented scheme achieved better performance compared to other works regarding the number of resources.

Prabhu et al. [32] suggested work encompass achieving multiple channels by using an independent 2D network topology based on the ACE protocol specification. The objective of the proposed work is to design a multi-core processor linking system after selecting the appropriate routing algorithm, flow control protocol, and precise router architecture.

Deb et al. [33] proposed an energy-efficient caching strategy for blocks bringing ECAP. An effective prefetching strategy may generate more traffic, which increases energy consumption and network congestion. It uses a lower set of near-tile cache that runs light applications such as memories of the default tile cache that runs high applications in order to place the prefetching blocks. The ECAP minimizes the link power in NoC, the router, and AMAT by 27%, 14.42%, and 23.54%, respectively compared to conventional pre-level technology.

Kaur and Sulochana [22] designed and implemented a dual core system that consisted of two caches and main memory. The cache coherence system based on the write invalidate approaches and on the MSI, MESI and MOESI protocols. The results indicated that the MOESI protocol obtained better performance in terms of memory latency, power consumption, gate count and execution time.

Jianhua Li et al. [34] proposed thread-progress-aware coherence protocol system. They used the thread progress information as hints for coherence adaption. Also, they depended on the thread critically to dynamically adapting threads coherence. Moreover, they used Simics simulator for performing their system experiment.

The results shown that the proposed system enhanced the execution time and energy squandering. Furthermore, they the proposed system could execute well especially when the chip bandwidth is limited.

Mencagli et al. [35] proposed new hardware mechanism for reducing the interactions between caches generated by the cache coherence protocol. They implemented their mechanism in a run time support able system in order to depressing individual cache contention so as to decrease the active latency of inter thread cooperation primitives. They aimed to enable fine grained parallelism by optimizing the overhead of communication through coordinating cache coherence protocol. The proposed system attained good speedup in fine grained parallelism.

Sun et al. [36] designed a system to enhance performance of the shared memory. They used selective write shared transformation strategy in order to remove coherence traffic and misses. Moreover, the proposed technique was implemented and evaluated by parallel benchmark called NAS. The experimental results of the designed strategy outperformed the write invalidate transition by improving 21% of speedup.

J. Wang and D. Wang [37] proposed a model for network on chip based energy used for cache coherence protocol. They aimed to reduce the energy of low activated protocol in order to reduce the traffic energy and latency by using task mapping algorithm. They used three benchmarks for comparing their strategy with five protocol of the application layer. The results indicated that the proposed model could decrease energy by 20%. A comparative analysis of all previous mentioned methods is shown in Table I.

TABLE I. A COMPARATIVE ANALYSIS OF ALL METHODS

Ref.	Year	Problem	Significant Results	Network Topology
[24]	2013	Cache integration in shared memory faces scalability and cost problems.	Supporting MOESI instances and accelerating data transfers.	MESH
[34]	2013	Thread progress is affected by cache memory	The proposed protocol outperformed directory protocol	MESH
[36]	2014	Shared data writing latency affects the system performance	The proposed system reduced latency by 21%	RING
[25]	2015	The cause of CMP performance is strongly influenced by the extent to which data is fetched from the memory system.	The address range is displayed by rearrange L2 improved up to 13, 5 %.	RING
[26]	2015	Time value of data time based on processor instructions executed each clock cycle with corresponding memory access.	Display how each scheme affected by cache size and block size.	BUS
[27]	2015	The increasing complexity of the coherence protocol and network on a chip has become a major challenge for pre-silicon verification.	Effectively reduce engineers' verification efforts and times.	MESH
[28]	2016	Increased access latency because of the large number of processors with shared memory.	Power consumption and maximum latency.	MESH
[23]	2017	Maintaining data consistency between memory and all caches.	Give a clear picture of the reading and writing application being implemented.	BUS
[35]	2017	Communication overhead among caches of parallelism	Enhance speed of parallelism computation	MESH
[30]	2018	Energy efficiency impact on the development of microspheres.	Include the power and region model by McPAT.	MESH
[29]	2018	This simulation will evaluate its effect on the learning process.	Presented in terms of quality and quantity.	BUS
[22]	2018	Inconsistency of data between shared memory and caches led to the cache coherence problem	The performance of MOESI protocol outperformed MSI and MESI	BUS

[37]	2019	Cache coherence protocol consume energy in application layer	The designed model saved energy by 15%	MESH
[31]	2019	Adaptive routing algorithms implementation force a relevant resource increase at both network and protocol level.	The performance of the adaptive routing algorithms increased more than 23%.	MESH
[32]	2019	Uploading and storing data providing resource allocation, managing multiple accesses, and keeping the cache coherency**.**	Designing a multi-core processor linking system after selecting the appropriate flow control protocol.	MESH
[33]	2019	An effective prefetching strategy may generate more traffic which increases energy consumption and network congestion.	The latency of cache miss packets reduces by 25.34%.	MESH

IV. Discussion

Considering the comparison details illustrated in Table 1, there are several semaphores can be observed. The comparison focused on scalability and cost problems facing cache integration. Cache memory effects on thread progress. System performance affected by shared data writing latency. Data fetching from/to memory effects on CMP performance. Data time based on processor instructions executed each clock cycle. The major challenge for pre-silicon verification is increasing complexity of the coherence protocol and network on a chip. Some researches depended the uploading and storing data providing resource allocation, managing multiple accesses, and keeping the cache coherency. This is can be done based on designing a multi-core processor linking system. This section deals with providing a discussion of a number of methods that use the cache coherent protocol in a distributed system. Hardware cache coherence systems are classified into two categories which are directory-based and snoopy-based. The directory-based schemas maintain a central directory to store the memory block sharing state. The most common types of cache coherent protocols are SI, MI, MSI, MESI, MOSI, MEOSI, and MESIF. The best one is MOESI because this protocol minimizes number of bus messages sent for an invalid transition to a rate while still allowing multiple participants.

V. Conclusion

A cache coherent protocol is one way to interconnection of the caches in a multiprocessor. The major challenge of shared memory devices is to maintain the cache coherently. In this paper, we presented a number of methods used for the cache-coherent protocol in a distributed system and which type of protocol used for network topology and provided the type of cache-coherent protocol. It can be concluded that Cache coherence protocol consume energy in application layer and the designed model can saves energy by 15%. Also, GPU data are used for consumption and data in the shared cache causing extended broadcasting period. However, the other one MESI which supports both write cache. Hoever, effective prefetching strategy may consume latency of cache miss packets reducing by 25.34%.

References

[1] S. R. Zeebaree, L. M. Haji, I. Rashid, R. R. Zebari, O. M. Ahmed, K. Jacksi, & H. M. Shukur, “Multicomputer Multicore System Influence on Maximum Multi-Processes Execution Time,” *TEST Engineering & Management*, vol. 83, no. May/June, pp. 14921–14931, May 2020.

[2] D. R. K. Ports, J. Li, V. Liu, N. K. Sharma, and A. Krishnamurthy, “Designing Distributed Systems Using Approximate Synchrony in Data Center Networks,” 2015, pp. 43–57, Accessed: Feb. 27, 2020. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/ports.

[3] R. R. Zebari, S. R. Zeebaree, and K. Jacksi, “Impact Analysis of HTTP and SYN Flood DDoS Attacks on Apache 2 and IIS 10.0 Web Servers,” in 2018 International Conference on Advanced Science and Engineering (ICOASE), 2018, pp. 156–161.

[4] O. H. Jader, S. R. Zeebaree, and R. R. Zebari, “A State Of Art Survey For Web Server Performance Measurement And Load Balancing Mechanisms,” INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH, vol. 8, no. 12, pp. 535–543, Dec. 2019.

[5] S. R. Zeebaree, R. R. Zebari, K. Jacksi, and D. A. Hasan, “Security Approaches For Integrated Enterprise Systems Performance: A Review,” International Journal of Scientific & Technology Research, vol. 8, no. 12, Dec. 2019.

[6] G. Lu, J. Zhan, X. Lin, C. Tan, and L. Wang, “On Horizontal Decomposition of the Operating System,” CoRR abs/1604.01378, 2016.

[7] L. M. Haji, S. R. Zeebaree, K. Jacksi, and D. Q. Zeebaree, “A State of Art Survey for OS Performance Improvement,” Science Journal of University of Zakho, vol. 6, no. 3, pp. 118–123, 2018.

[8] Z. N. Rashid, S. R. Zebari, K. H. Sharif, and K. Jacksi, “Distributed Cloud Computing and Distributed Parallel Computing: A Review,” in 2018 International Conference on Advanced Science and Engineering (ICOASE), 2018, pp. 167–172.

[9] S. R. M. Zeebaree, H. M. Shukur, L. M. Haji, R. R. Zebari, K. Jacksi, and S. M.Abas, “Characteristics and Analysis of Hadoop Distributed Systems,” Technology Reports of Kansai University, vol. 62, no. 4, pp. 1555–1564, Apr. 2020.

[10] S. R. Zeebaree, R. R. Zebari, and K. Jacksi, “Performance analysis of IIS10.0 and Apache2 Cluster-based Web Servers under SYN DDoS Attack,” TEST Engineering & Management, vol. 83, no. March-April 2020, pp. 5854–5863, 2020.

[11] S. Bansal, S. Sharma, and I. Trivedi, “A Detailed Review of Fault-Tolerance Techniques in Distributed System.,” International Journal on Internet & Distributed Computing Systems, vol. 1, no. 1, 2011.

[12] K. Jacksi, “Design and Implementation of Online Submission And Peer Review System: A Case Study Of E-Journal Of University Of Zakho,” International Journal of Scientific & Technology Research, vol. 4, no. 8, pp. 83–85, 2015.

[13] S. R. Zeebaree, K. F. Jacksi, and R. R. Zebari, “Impact analysis of SYN flood DDOS attack on HAPROXY and NLB cluster-base web servers,” Indonesian Journal of Electrical Engineering and Computer Science, vol. 19, no. 1, Art. no. 1, doi: 10.11591/ijeecs.v19.i1.pp. 505 - 512, 2020.

[14] Z. Subhi RM and J. Karwan, “Effects of Processes Forcing on CPU and Total Execution-Time Using Multiprocessor Shared Memory System,” International Journal of Computer Engineering in Research Trends, vol. 2, no. 4, pp. 275-279, 2015.

[15] R. R. Zebari, S. R. Zeebaree, K. Jacksi, and H. M. Shukur, “E-Business Requirements For Flexibility And Implementation Enterprise System: A Review,” International Journal of Scientific & Technology Research, vol. 8, no. 11, pp. 655–660, Nov. 2019.

[16] R. Komuravelli, S. V. Adve, and C.-T. Chou, “Revisiting the complexity of hardware cache coherence and some implications,” ACM Transactions on Architecture and Code Optimization (TACO), vol. 11, no. 4, pp. 1–22, 2014.

[17] 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, 2015, pp. 125–130.

[18] O. Alzakholi, L. Haji, H. Shukur, R. Zebari, S. Abas, and M. Sadeeq, “Comparison Among Cloud Technologies and Cloud Performance,” Journal of Applied Science and Technology Trends, vol. 1, no. 2, Apr. 2020, doi: 10.38094/jastt1219.

[19] A. Saparon and F. N. B. Razlan, “Cache Coherence Protocols in Multi-Processor,” in International conference on Computer Science and Information Systems (ICSIS), 2014, pp. 17–18, 2014.

[20] X. Qin and P. Mishra, “Automated generation of directed tests for transition coverage in cache coherence protocols,” in 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012, pp. 3–8.

[21] Z. Al-Waisi and M. O. Agyeman, “An overview of on-chip cache coherence protocols,” in 2017 Intelligent Systems Conference (IntelliSys), 2017, pp. 304–309.

[22] D. P. Kaur and V. Sulochana, “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), 2018, pp. 1097–1102.

[23] D. Kehagias and I. Raptis, “An Android-based MESI Cache Coherence Simulator.” in 2017, International Virtual Conference on Advanced Scientific Result, pp. 194–199.

[24] C. Bernard, H.-N. Nguyen, E. Guthmuller, and Y. Durand, “Design and implementation of an In-Network Cache Coherence protocol,” in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 2013, p. 298.

[25] Quang, Vinh Ngo et al. “Exploring Cache Coherency Design for Chip Multiprocessor using Multi2Sim.” International Journal of Engineering Research and Technology 4 (2015), pp. 775 - 779.

[26] S. Almakdi, A. Alazeb, and M. Alshehri, “Cache coherence mechanisms,” International journal of engineering and innovative technology, vol. 4, pp. 158 - 167, 2015.

[27] F. L. Y. Team, X. K. Z. B. Y. Jiang, and C. Lou, “Robust Cache Coherence Protocol Verification with Inferno.”

[28] Aghaei, Babak, and Negin Zaman-Zadeh. "Evaluation of Cache Coherence Protocols in terms of Power and Latency in Multiprocessors." In 3rd International Conference on Research in Engineering. 2016.

[29] D. Kehagias, “Using two Educational Simulator Tools for Computer Architecture Teaching and Learning Support,” International Journal of Computer Applications, vol. 180, no. 47, pp. 8 - 12, 2018.

[30] D. Smelt, “Modeling many-core processor interconnect scalability for the evolving performance, power and area relation,” 2018.

[31] T. A. Eltaras, W. Fornaciari, D. Zoni, "Partial packet forwarding to improve performance in fully adaptive routing for cache-coherent nocs." In 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2019, pp. 33-40.

[32] S. S. Prabhu, A. A. Kadar, and J. Simon, “Design and Development of Cache Coherent Interconnect based on ACE Protocol Specification.” International Journal of Engineering Research & Technology, vol. 8, pp. 730-734, 2019.

[33] D. Deb, J. Jose, and M. Palesi, “ECAP: energy-efficient caching for prefetch blocks in tiled chip multiprocessors,” IET Computers & Digital Techniques, vol. 13, no. 6, pp. 417–428, 2019.

[34] J. Li, L. Shi, C. J. Xue, and Y. Xu, “Thread progress aware coherence adaption for hybrid cache coherence protocols,” IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 10, pp. 2697–2707, 2013.

[35] G. Mencagli, M. Vanneschi, and S. Lametti, “The home-forwarding mechanism to reduce the cache coherence overhead in next-generation CMPs,” Future Generation Computer Systems, vol. 82, pp. 493–509, 2018.

[36] S. Sun, H. An, and J. Chen, “Cache Coherence Method for Improving Multi-threaded Applications on Multicore Systems,” in 2014 6th International Conference on Multimedia, Computer Graphics and Broadcasting, 2014, pp. 47–50.

[37] J. Wang and D. Wang, “A smart protocol-level task mapping for energy efficient traffic on network-on-chip,” Microprocessors and Microsystems, vol. 65, pp. 69–78, 2019.