International Journal of Information Systems and Computer Technologies (IJISCT) Vol. 1(2) – July 2022
ISSN: 2791-3635(Print), 2791-3643(Online)

Cache Coherence issues and Solution: A Review

Khalid Hamad Alkhamisi[*1]

Department of Computer Science and Artificial Intelligence, University of Jeddah, Saudi Arabia

Corresponding Author: Khalid Hamad Alkhamisi[*] (kalkhamisi.stu@uj.edu.sa)

ABSTRACT
The computer systems are extensively being used in the today's era and the advancement in the computer systems has led to the evolution of various technologies. As the efficiency of the computer systems is to be enhanced, the usage of cache memory becomes a must. The cache memory problems occur when multiprocessor systems are used. When multiple processors share a common memory pool, the problem of cache coherence occurs. Cache coherence is a state in which the cache memories of the processors must stay in coherence with each other. The data should be updated in all the cache systems. Various protocols are used to ensure the cache coherence. An extensive literature review has been conducted in this study to identify the cache coherence issues. This study delineates the cache memory architecture, cache coherence and the issues that arise due to non-consistent data in the cache of multiprocessor systems. Moreover, this study also delineates the protocols of the cache coherence and also describes in detail the MISE protocol. This study will provide an in-depth review of cache, cache coherence and the issues associated with cache coherence along with the solution. This study will contribute by providing significant knowledge about the cache issues and may also lead to the solution of the problems that arise due to cache coherence issues. This study may act as a guide for identification of cache coherence issues.

Keywords: Cache memory, cache coherence,

1. INTRODUCTION

The advancement in the computer systems has escalated the need of a high speed memory. Moore's law for the doubling of the doubling of components per circuit seems farfetched and this effect has gone for the whole computer architecture as the computer capabilities are improving day by day. With the continuous evolution and improvement, the gap between the memory speed and the speed between the instruction retrieval has been extended [1].

A computer system comprises of various components that include memory unit as well which is mainly used for the storage of data and information. In consideration of the memory hierarchy of the computer systems with respect to cost, speed and size, a memory element called cache occurs between the main memory and the processor that improves the memory access time. Cache is small sized highspeed memory of the computer that is integrated into the Central Processing Unit (CPU) or may also be used separately as chip on the motherboard connected with bus to the CPU. The memory hierarchy is depicted in Figure 1. The main purpose of the cache memory is to store the data which is repeatedly being used by the system instead of fetching data and instructions from main memory for the execution of next program. When the data is once brough into the cache memory, the data is then fetched from the cache and thus speeding up the computing process which in turn enhances the efficiency of the computer system [2].

The continuous improvement of the processors and the performance means that memory caching is more important to maintain the advancements of the computer systems. It is mainly due to the fact that data and instruction fetching takes more time than the actual computing operation. Speed is the actual essence in the process and this has exclusively made memory a key element in the success of computer systems. The more data that can be saved in the cache memory, the more is the speed of the processor [3].

The distance of the cache from the main memory affects the performance of the system largely, it works on the locality of reference which is the tendency of the process to access the same set of the memory locations for reuse of recently

used instructions over short time period. With the increase in the processing speed of the processor, the equivalent increase in the cache size is required which will ensure that the data is timely synchronized between the processor and the main memory [4].

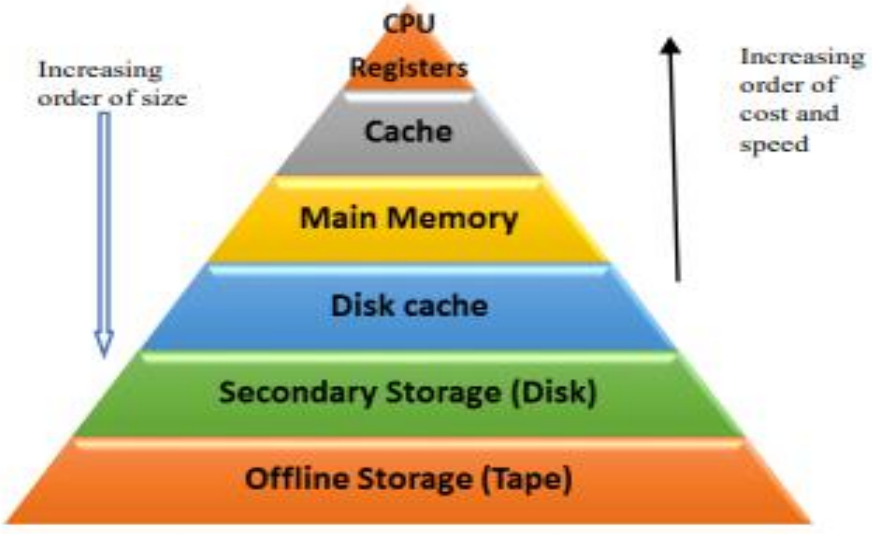


Figure 1: Memory Hierarchy

The major issue in the large sized cache memory usage is the cost. Cache memory comprises of highspeed SRAM instead of slower DRAM (Dynamic Random Access Memory). SRAM is more expensive and is unaffordable to make a system comprising of just SRAM. The cache memory can be sorted in levels which identify that how closely the memory is located to the processor. The levels are L1, L2, L3, Level 1 cache are very fast but are quite small and are used in processor chips, L2 is more spacious than L1 and can be on CPU or separate chip. L3 cache is not common but however it can double the speed of RAM. When an instruction is references, then it is elevated to higher level cache. Figure 2 shows the architecture of cache memory and also shows its function cycle.

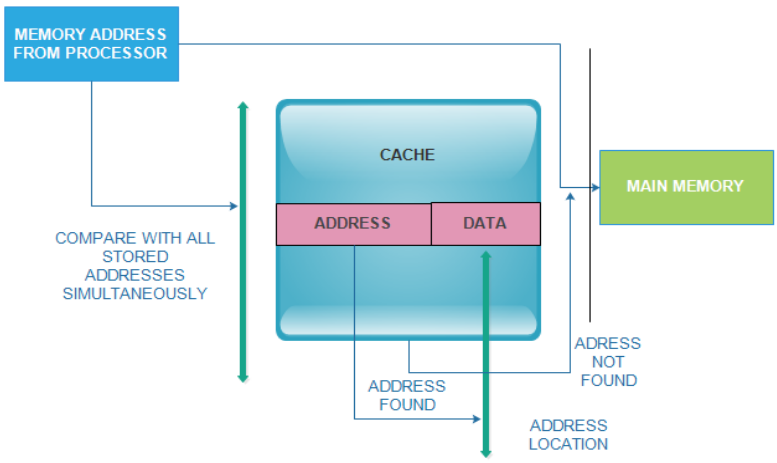


Figure 2: Cache Memory Design

1.1 Cache Coherence

With technological advancement, distributed shared memory systems or multiprocessors are now widely used and have designed in a way to have more than one level of cache that are embedded in processors. The multiple copies of the program instructions and the operands simultaneously exist in every processor and the main memory as well. This arrangement assists in improving the performance of the system as the system will instantly use the instructions that are available in the cache of processor which is currently in the active state, this arises a bottleneck problem which is termed as cache coherence problem. This problem arises when the update information in cache is not reflected in other caches that are holding the same information [5]. This data disparity among cache occurs due to the write back policy- it is an approach deployed by the memory to do the write operation on the cache memory on [6]. The cache coherence is a design consideration considered in the design of the memory architecture and it certifies the consistency of data that is stored in cache memory. This design consideration prevents the over writing and maintains uniform data state for each cached data block.

Various software and hardware techniques have been deployed for managing the cache coherence problems. The software approach is done by the drivers which flushes away the dirty data and also invalidates old data by enabling the sharing with other processors. This effects the bus bandwidth, processor cycles and the power resulting in high cost of cache maintenance and also led to the inefficient utilization of cache, specifically for sharing rates between caches. The software solution relies on the operating system or the compiler that are dealing with the cache coherence and dealing with the cache coherence problems at runtime becomes unachievable. The hardware approach is named as the cache coherence protocols and it turns out to be the best possible solution to cater the data inconsistency. The dynamic recognition is provided to the potentially inconsistency situations at the runtime [6, 7]. As the inconsistency problem is addressed when it occurs, thus providing the effective usage of cache memory and also improves the performance of the system.

1.2 Cache Coherence Implementation

There are two categories of the cache coherence systems namely snoopy based and directory-based schemas. In directory-based schema, a central directory is maintained which stores the sharing state of the memory block [8]. Whereas in snoopy based schema for consistency maintenance, request broadcasting and the activities of the memory bus are monitored by cache controller.

1.2.1 Snoopy Protocol

Bus snooping protocol is given in Figure 3, it requires the adoption of the bus based communication medium in machine which is used to broadcast the messages to the other connected processors [9, 10]. By this concept each cache maintains the cache coherence as it can listen to the interconnected bus for the updates or can write on the shared data. Each processor monitors the activities which are carried out in the bus.

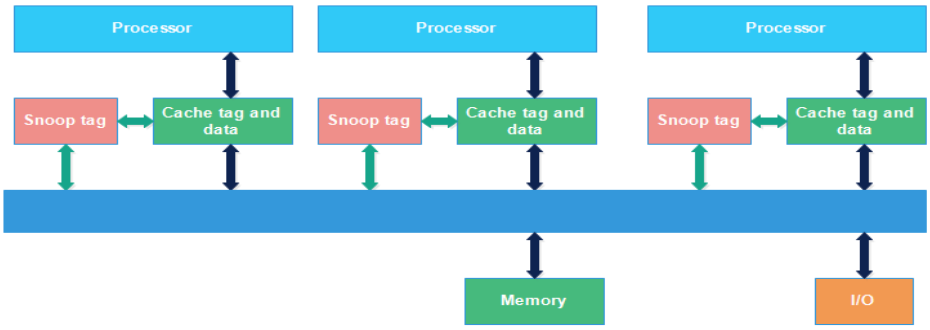


Figure 3: Snoopy based Protocol

The Snoopy protocol is inexpensive, scalable and efficient. For read instructions all the caches check whether there is any copy of the requested block and if it is yes then data is to be supplied. Similar is the case for the write instructions, all caches check to see if they have copy of data, if yes then the local copy is invalidated and is updated with the new value. A snooping protocol makes sure that invalid data is not being used by any core.

1.2.2 Directory based protocol

It keeps track of all the changes in every block of the shared data in the directory. It functions like a look up table and is used by the processors to identify the data consistency and coherence [8]. A centralized directory maintains the information about the block states in all other caches as illustrated in Figure 4.

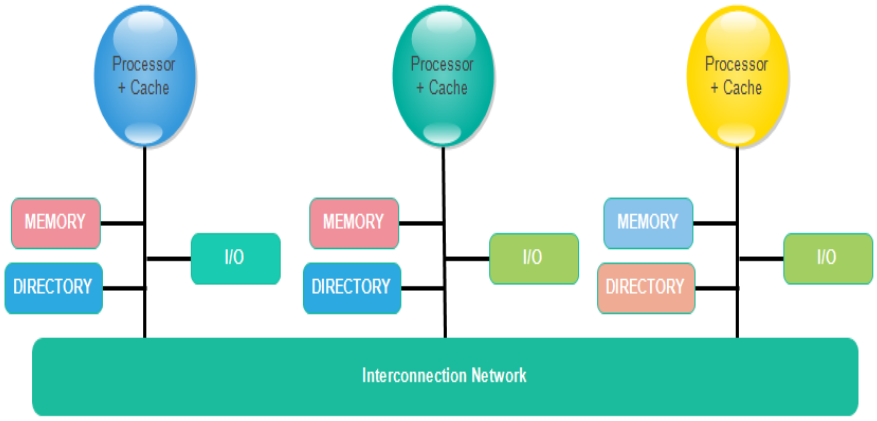


Figure 4: Directory based protocol

1.3 Cache States

In both snoopy based and the cache coherence based every block of the cache is controlled by using three states named as uncached/ invalid, shared/clean and exclusive/dirty.

a) *Uncached/Invalid:* It means that the block is invalid in all caches, no cache processor possess this block.
b) *Shared/Clean:* The cache block is with one or more processor and is up to date.
c) *Exclusive/ Dirty:* The block is updated for only one processor and memory is out dated.

The summary of the directory based and the snoopy based protocols is illustrated in Table 1.

Table 1: Directory based Verses Snoopy protocols

	Directory based protocols	**Snoopy Protocols**
Advantages	• Better scaling than the snoopy protocols • Less bandwidth-as the messages are not broadcasted	• It has low average miss latency • It has higher bandwidth and is thus faster
Disadvantages	• It has longer latency. • The interconnected traversal and the directory access is on critical path of cache to cache miss. • It also involves the storage and the manipulation of the directory state.	• It dissipates more power which makes it inefficient. • Every request is required to be sent to all other nodes in the system, which means that if the system size increases, along with it the bandwidth and the bus size should also be enhanced.

A cache coherence protocol is a way to maintain the interconnection between caches in the multiprocessor system by deploying hardware [11]. The issue of cache consistency causes cache error which produces the coherence loss with conflict, capacity errors and binding. The coherence failure occurs when the cache of the local node copies and then cancels the shared copies to maintain data consistency. The most common cache coherence protocols are listed below [12].

1.3.1 Shared-Invalid (SI)

It is parsed version of the MSI protocol and in this the cores are not allowed to release storage. A system having n cores, the correct general state allows system for storing cache block in m centers in I, and blocking the cache in other n-m centers in the S state. The entire space of the SI protocol state having n cores is hypercube.

1.3.2 Modified-Invalid (MI)

The MI protocol is the simplest one and is used to maintain the cache coherence. This protocol has two conditions modify (M) and Invalid (I). This protocol has least complex hardware among all other protocols. The read/write operation invalidates the data in the other node if the same data exists.

1.3.3 Modifies-Exclusive -Shared- Invalid

The MESI protocol was firstly used in the Pentium processors. In the MESI protocol the data is represented by block in cache and the cache will be in any state from modified, shared, invalid, and exclusive [13]. The cache controller will make snooping to ensure that no block has old data and will update the block containing old data that is from invalid to shared, shared to exclusive and so on. The MESI protocol is considered as most commonly used cache coherence protocol and it supports write cache 1 and cache 2. It is an advanced form of MSI protocol and has additional case E which reduces the count mobile carriers that send incorrect fix to rate. In the exclusive case the local cache contains the valid copy [14]. The Figure 5 illustrates the state diagram of MESI protocol.

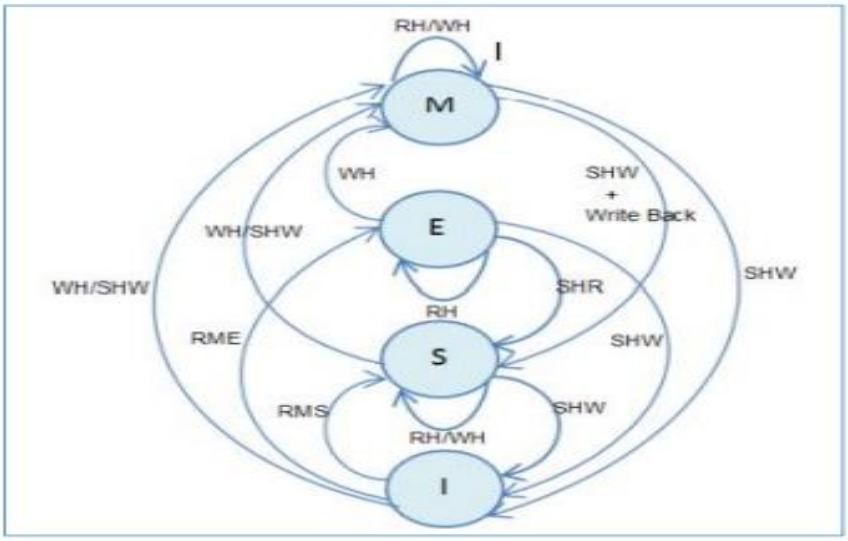


Figure 5: MESI Protocol

1.3.4 Modified-Owned-Exclusive-Shared-Invalid (MOESI)

MOESI has four states that denote a cache line. The owned state acts as an optimization for the MESI protocol. The exclusive assists in saving bus request, when sequential process is running as only on one processor, all the accesses would be exclusive. It minimizes the bus requests for invalid transition to a specific rate, still permitting multiple participants. This protocol has an additional state which contains valid copy of the local cache.

1.3.5 Modified-Owned-Shared-Invalid (MOSI)

In this the transport node sends converted data on the read request without rewriting data. In the requesting and responding node, the cache state is changed to S and O. The O status allows the sharing of dirty copy between various nodes. It can be avoided by combining MESI and MOSI in MOESI [15].

2 Conclusion

The computational systems have taken hold of the world and the activities that are carried out. Multiprocessor systems require that each processor should possess its own cache. It is mandatory for the caches in such design to stay coherent by using different protocols. Cache coherence is interconnection of caches in multiprocessor system. The implementation of protocols is attributed to specific machine and thus there is a question that which protocol will suit best. In this paper a number of methods used in distributed systems for cache coherence protocols are presented along with the state diagram of cache protocols. Moreover, this study discusses in detail the cache, functionality of the cache along with the architecture of the cache.

REFERENCES

[1] D. Sam and M. O. Agyeman, "An overview of design space exploration of cache memory," in *Proceedings of the 2nd International Symposium on Computer Science and Intelligent Control*, 2018, pp. 1-6.

[2] H. Gorjiara, G. H. Xu, and B. Demsky, "Jaaru: Efficiently model checking persistent memory programs," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 415-428.

[3] H.-S. P. Wong and S. J. N. n. Salahuddin, "Memory leads the way to better computing," vol. 10, pp. 191-194, 2015.

[4] I. Arikpo and A. Okoro, "Models for addressing cache coherence problems," 2019.

[5] W. Stallings, *Computer Organization and Architecture: International Edition*: Pearson Higher Ed, 2013.

[6] S. P. Dandamudi, *Fundamentals of computer organization and design* vol. 7: Springer, 2003.

[7] M. Kumar and P. Arora, "A Survey of Cache Coherence Protocols in Multiprocessors with Shared Memory," in *Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering*, 2012, pp. 384-388.

[8] H. Shukur, S. Zeebaree, R. Zebari, O. Ahmed, L. Haji, D. J. J. o. A. S. Abdulqader, *et al.*, "Cache coherence protocols in distributed systems," vol. 1, pp. 92-97, 2020.

[9] S. Al-Hothali, S. Soomro, K. Tanvir, R. J. J. o. I. Tuli, and C. Technology, "Snoopy and directory based cache coherence protocols: A critical analysis," vol. 4, pp. 01-10, 2010.

[10] R. Kaushik, K. Pavan, and S. J. I. J. o. c. e. Meenatchi, "Comparative study on Cache Coherence Protocols," vol. 17, pp. 71-75, 2015.

[11] O. Alzakholi, H. Shukur, R. Zebari, S. Abas, M. J. J. o. A. S. Sadeeq, and T. Trends, "Comparison among cloud technologies and cloud performance," vol. 1, pp. 40-47, 2020.

[12] M. K. Faeq, S. S. J. I. J. o. E. Omran, and C. Engineering, "Cache coherency controller for MESI protocol based on FPGA," vol. 11, p. 1043, 2021.

[13] R. Zhang, S. Biswas, V. Balaji, M. D. Bond, and B. J. a. p. a. Lucia, "Neat: Low-Complexity, Efficient On-Chip Cache Coherence," 2021.

[14] R. Komuravelli, S. V. Adve, C.-T. J. A. T. o. A. Chou, and C. Optimization, "Revisiting the complexity of hardware cache coherence and some implications," vol. 11, pp. 1-22, 2014.

[15] D. Kehagias and I. Raptis, "An Android-based MESI cache coherence simulator," in *The 5th Int. Virtual Conf. on Advanced Scientific Results*, 2017, pp. 194-199.