UACEE International Journal of Computer Science and its Applications - Volume 2: Issue 1 [ISSN 2250 - 3765]

A Survey of Cache Coherence Protocols in Multiprocessors with Shared Memory

Manoj Kumar
Delhi Technological University
Delhi, India
mkg1109@rediffmail.com

Pooja Arora
Delhi Technological University
Delhi, India
poojaarora014@gmail.com

***Abstract*— Appropriate solution to illustrious Cache Coherence Problem in shared memory multiprocessors system is one of the crucial issue for improving system performance and scalability. In this paper we have surveyed various cache coherence mechanisms in shared memory multiprocessor. Various hardware based and software based protocol have been investigated in depth including recent protocols. We have concluded that hardware based cache coherence protocol are better than software based protocol according to presently available protocols, but hardware based protocol have added the cost to implement them. In comparison analysis of protocols on SMP Cache simulator by varying certain parameters we noticed that the Dragon Protocol is giving the best results in terms of number of hits at great extent. As software based cache coherence protocol are more economical therefore more devotion is needed for software based protocol as they show great promise for future work.**

***Keywords—Shared Memory, Multiprocessors, Cache Cohernce Problem*, Hardware Based Protocol, Software Based Protocol.**

I. INTRODUCTION

These days the speed of processor is increasing exponentially. Multiprocessors are the best type of computer responsible for continuously increasing computing power. Among these multiprocessors with shared memory is the most efficient class of multiprocessors. In 2000, the sales of shared-memory systems with more than eight processors passed $16 billion [1]. In multiprocessors system with shared memory, work load can be divided among these processors therefore, they work faster than uniprocessor. These systems allow the easier development of parallel software and also can increase the system throughput, reliability and they are economical too.

The shared memory multiprocessors suffer with significant problem of accessing shared resources in a shared memory it will result in longer latencies consequently the performance of the system will get affected. With the object of solving the problem of increased access latency due to large number of processors with shared memory, Cache is being used. Every processor has its own private cache, now they can update or access the data comfortably but again it leads to another serious issue i.e. cache coherence problem.

Cache coherence problem arises when multiple processes are trying to access the same data for updating purpose or one processor is trying to modify the data and rest processors are trying to read simultaneously. It may lead to inconsistent state of data at cache of different processors and the main memory. We will discuss the solutions of cache coherence problem in detail.

In shared memory multiprocessors system where we can have multiple copies of same data in the private cache of processor. If all the processors are allowed to independently update the data then it will lead to malfunction. This is the well-known impression of cache coherence problem. We call the cache of the system coherent only if every read operation results in the value which is updated by previous write operation, even by the process at any other processor of that system. To resolve this problem the system must comprise of some mechanisms to maintain the coherent view of memory and assures execution of program with correct version data.

Cache coherence problem has attracted the attention of various universities and companies in last two decades. In fact the researchers had come out with lots of solutions to this problem. This problem is not only forcing the mal-functioning of the program but also impacting the system performance drastically. Efficiency of cache coherence depends on system

II. BACKGROUND

On the basis of write operation Cache Coherence Protocol can be categorized as [2]: 1.Write Update 2.Write invalidate. Difference between these two is that when one processor issues write operation Invalidate protocol modifies the copy of cache and invalidates all other copies of that data block. In case of update protocol it will not only write on that processor's cache which is trying to update but also will forward this change to other existing copies. In 1993, Gee et al. compared invalidate, update, and adaptive protocols for cache coherence in [3][4]. They showed that invalidate protocols were best for vector data, update protocols were best for scalar data, and adaptive protocols were the best on average.

On the basis that how memory is updated we can categorize these protocol as [2] : 1.Write Through Protocol 2.Write Back Protocol. In write through protocol, when processor tries to update the shared data block, it will update in memory too. But in write-back protocol when processor tries to update, the main memory can be updated as:

- When the only valid copy of data block is available in the processor and it replaces that block
- When processor reads it from another processor's cache.

We can classify cache coherence mechanisms as:

1.Software-based solutions: These solutions generally rely on compiler or operating system dealing with coherence problem. Hardware-based solution: This approach can deal with coherence problem at run time.

If we compare these two strategies then we see that though hardware based solutions are expensive as it adds up new hardware cost but it is scalable up to hundreds or thousands of processors [5]. But when it come to software based solution, since it is not adding any hardware so cost is not getting enhanced but its scalable up to 32 processors only. And software based protocol are not capable to deal with coherence problem at run time.

III. HARDWARE BASED PROTOCOL

A. *Snoopy Protocol*

Snoopy Cache Coherence Protocol is primarily suited for multiprocessors system with shared memory that has bus with global interconnect, as the shared bus provide very inexpensive and fast broadcast to exchange coherence information among processors. It strictly maintains consistent view of data as any update done by the processor is immediately visible to all other processors of that system. But the shared bus becomes bottleneck for large number of processors. Though it can be resolved by increasing the bandwidth of the bus but consequently it will increase the memory delay. Therefore this protocol can be scaled up to 32 processors only[6].

B. *Directory Based Protocol*

In Directory Based Protocol the global system-wide status information relevant for coherence maintenance is stored in some kind of directory [7]. The responsibility of coherence is predominately delegated to centralized directory controller. On individual request from local cache controller, the centralized controller checks the directory and issue necessary command for transfer of data between caches or cache and memory. It also keeps the information about status, so that any local action which can impact the global state of block must be acknowledged to the central controller.

There is also a private cache, which keeps local state information about cached block. We can organize this global directory as [8]:

- Full Map Directory: In this all the cache can have a copy of every data block, i.e. each directory entry has P pointers where P is a number of processors of that multiprocessor system. The first protocol of this class was developed in IBM 3081[9].
- Limited Directory: This scheme reduces the size of directory by having limited number of pointers for each entry in the directory without any concern with number of processors. The organization of limited directory scheme is described in [10]
- Chained Directory: Chained directory imitates the full map directory scheme by distributing the directory among caches. This scheme does not restrict the number of copies of shared data block. It actually keeps the track of shared data block by maintaining a chain of directory pointers and it does not use broadcast too that mean it does not introduce any increase in the traffic

These days many commercial multiprocessor systems implement directory-based coherence including the new SGI Origin which can have 1,024 processors in a maximal configuration.[11] Many versions of directory schemes have been proposed and many machines with hardware cache coherence have been built [12] [13] [14] [15].

C. *Hybrid Cache Coherence Protocol*

As we know that different data block present different access behavior, for this we require Cache Coherence Protocol which is capable of applying more than one protocol, is known as Hybrid Cache Coherence Protocol. This protocol has potentially enhanced the performance of multiprocessor system. It uses two basic protocol viz. invalidate protocol and update protocol[16][17].

In Hybrid Cache Coherence Protocol we have decision function as quintessence, which selects the appropriate protocol prior to or during execution of the program. The decision function is classified as:1. Online Decision function. 2. Offline Decision Function. In Dynamic or Adaptive Hybrid Cache Coherence protocol, the shared data block residing in specific cache might get updated during the execution of an application. Hybrid Cache Coherence Protocol is also known as Competitive Update Protocol. The performance of this protocol is not good when we have migratory data i.e. data which is read or modified by multiple processors.

D. Lock Based Protocol

This Lock Based Cache Coherence Protocol is improvement of Directory based protocol presented in [18]. This is more promised towards scalability than directory based scheme by implementing scope consistency. The scope consistency is a compromise between lazy release consistency and entry consistency [18]. In this mechanism we do not have directory. All the memory coherence actions are taken through reading and writing to and from lock, which takes care of shared memory. When lock gets released it sends all the write notices to the home of the lock and all the modified memory lines. On acquisition of lock, processor knows from the home of the lock that which lines have been modified and can also access those modifications. This mechanism is more scalable as no directory is required but this scheme is slow as processor has to wait until lock is released and for all the writes to be transmitted and acknowledged.

IV. Software Based Protocol

A. *MSI Protocol*

This is basic protocol for write-back cache. It has three states, used for write-back cache to determine the valid data block which is not modified (dirty blocks). These states are:

- Modified: This is also known as dirty state. This cache has the only valid copy of data blocks, even main memory has incoherent copy of that shared data block.
- Shared: This means it is consistent copy of data.
- Invalid: This means that it is inconsistent copy of shared data block. In this protocol before write operation, all other copies of data shared data block must be invalidated.

B. *MESI Protocol*

It is also known as Illinois protocol, due to its development in university of Illinois at Urbana-Chanpaign [19]. This protocol is very renowned, supports write-back cache. It is better than MSI protocol as for every write operation there are two transitions, even when that data block is not shared then too. In the first transition it gets the memory block in shared state and in second transition causes write it also changes the state of that data block to shared state from modified state. It adds a new state to MSI protocol i.e. Exclusive state which reduces the traffic because of write operation of shared data block.

C. *MOSI Protocol*

MOSI is also an extension of basic MSI protocol. One new state has been added to it i.e. Owned state. When the cache line is in owned state has the most recent and correct copy of data. This new state: (i) is like shared state of data. (ii) it is also like modified state as main memory can have the stale copy of the data. At a time only one cache can be in owned state and all other cache hold the data in shared state. After writing, it changes to shared state by modifying the main memory.

D. MOESI Protocol

MOESI protocol encompasses all of the possible states used in other protocols. It has five states. The Owned state represents the data which is modified and shared. This avoids the need to write modified data back to main memory before sharing it.

E. Dragon Protocol

Dragon protocol was first time proposed by researchers at Xerox PARC for their dragon multiprocessor system [19]. It consists of four states:

- Exclusive Clean (Exclusive): It's like exclusive state of cache. In this case maim memory is up to date.
- Shared Clean: More than one cache can have this data block but it may or may not be consistent with main memory.
- Shared Modified: More than one cache can hold this data block, but main memory does not have the recent copy of that data The responsibility of updating this data block has been delegated to cache. At a time only one cache is in this state.
- Modified: It is like modified state of MSI protocol, which can modify shared data block and at this time main memory has stale copy of that data block which is updated by cache.

It does not have any explicit invalid state like MOSI, since it is an update-based protocol. It keeps the cache up-to-date, therefore we can use the data present in this cache at any time if the tag match is successful.

V. CSC(Coherent with Shared Cache) Protocol

This is the most recent cache coherence protocol, introduced in 2010 [20]. In this processor has three cache memories: 1.Current processor's private cache (local cache), 2.Remote processor's private cache (remote cache), 3. SC-cache. The SC-cache (Shared Coherence cache) is a small capacity cache, placed between private cache and the bus. This protocol is a combination of write-through and write-back mechanisms. This includes four states: 1.PI (Private Invalidate) 2. PD(Private Dirty) 3. PE (Private Exclusive) 4. SS (Share Shared).

The first three states are for local and remote cache but the fourth state exists in SC-cache only. In CSC protocol processor first access the local cache, if miss occurs then it searches at remote cache and if again miss occurs then local cache controller broadcast this request in the bus. The simulation results showed that as compared to Dragon protocol and MESI protocol, CSC protocol has reduced the number of times the write back to main memory and number of times the read operation and also total execution time is also reduced by nearly 10% [20]. It is better to employ CSC protocol with SC-cache then traditional protocol.

VI. MECSIF Protocol

MECSIF is recently developed hybrid cache coherence protocol which takes advantage of both directory based and snoopy protocol. This protocol introduced a small volume directory—DCache, which has overcome the problem of the shortcoming of undifferentiated broadcasting in snoopy protocol. [21]. It has seven states given as following: 1. E: Exclusive, 2. PC: Primary Clean, 3. SC: Slave Clean, 4. M: Modified, 5. S: Shared, 6. F: Forwarding, 7. I: Invalid. Simulation results show that the MECSIF protocol extent improves the efficiency of processor data access comparing with MESI protocol [21].

VII. Related Work

We have used SMP Cache simulator, by varying some parameters and keeping certain parameters fix we have drawn various results. As shown in table1. The Dragon Protocol is giving the best result in terms of increased number of hits and by reducing number of misses at great extent. The best results are shown in table 1. On whole we can conclude that if we want the best performance than SMP Cache simulator says that the selection of dragon protocol will be the best decision.

TABLE I. COMAPRISON ANALYSIS.

Sr. No.		**Cache Miss Ratio**		
	Set Associative	***MSI***	***MESI***	***DRAGON***
1	2- Way	0.8794	0.1418	0.0095
2	4-Way	0.1387	0.1337	0.0313

VIII. Conclusion

This survey tries to give a comprehensive overview of hardware and software-based solution to cache coherence problem in shared memory processor. Both approaches perform well but their selection depends on the type of access pattern of shared data block and also number of processors we want to connect. Cache coherence significantly impacts the performance of the processor. The performance includes latency, bandwidth and protocol overhead.

Despite of considerable advancement in this discipline it's still very active research area. There exist many research topics like verification of protocol correctness, performance evaluation, comparison, size of the directory, minimization of protocol overhead. And more, which are needed to digged in the future.

IX. References

[1] Alan E. Charlesworth. The Sun Fireplane System Interconnect. IEEE Micro, 22(1):36{45, January 2002.

[2] P. Stenstorm, "A survey of cache coherence Schemes for Multiprocessors", IEEE Computer, Vol. 23, No. 6, June 1990, pp. 12-24

[3] Ramon Lawrence "A Survey of Cache Coherence Mechanisms in Shared Memory Multiprocessors" University of Manitoba, May 1998

[4] Jeffrey G. Gee and Alan Jay Smith. Absolute and comparative performance of cache consistency algorithms. Technical Report CSD-93-753, University of California, Berkeley, 1993

[5] M. Tomasevic and V. Milutinovic, " Hardware Solutions for Cache Coherence ins Shared-Memory Multiprocessors Systems", The Cache Coherence Problem in Shared-Memory Multiprocessors: Hardware Solutions, M. Tomasevic and V. Milutinovic, Ed., IEEE Computer Society Press, Los Alamitos, California, 1993, pp. 57-67

[6] R.Lawrence, "A survey of cache coherence mechanisms in shared memory", University of Manitoba May 1998

[7] M. Tomasevic and V. Milutinovic, " A Survey of Hardware Solutions for Maintenance of Cache Coherence in Shared Memory Multiprocessors" IEEE 1993

[8] Agrawal A., Gupta A., " Temporal, Processor, and Spatial Locality in Multiprocessor Memory References", Technical Report MIT/LCS/TM-397, June1989

[9] Tang761 Tang C., "Cache System Design in the Tightly Coupled Multiprocessor System," Proceedings of the National Computer Confmce, 1976, pp. 749-753.

[10] Aganwal A., Simoni R., Hennessy J., Hmwitz U,"An Evaluation of Directory Schemes for Cache Coherence," Proceedings of the 16th ISCA, 1989, pp. 280-289.

[11] James Laudon and Daniel Lenoski, "The SGI Origin: A ccNUMA highly scalable server", In Proceedings of the 24th Annual International Symposim on Computer Architecture, pages 241-251,1997

[12] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta, and John L. Hennessy, "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor", In Proceedings of the 17th Annual International Symposium on Computer Architecture, pages 148{159, Seattle, Washington, June 1990.

[13] David Chaiken, John Kubiatowics, and Anant Agarwal. LimitLESS Directories, "A Scalable Cache Coherence Scheme. In Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating System", volume 26, pages 224{234, Santa Clara, California, April 1991.

[14] Mark Heinrich, Vijayaraghavan Soundararajan, John L. Hennessy, and Anoop Gupta, "A Quantitative Analysis of the Performance and Scalability of Distributed Shared Memory", IEEE Transactions on Computers, 48(2):205{217, February 1999.

[15] Richard Simoni and Mark Horowitz, "Dynamic Pointer Allocation for Scalable Cache Coherence Directories", In Proceedings of the International Symposium on Shared Memory Multiprocessing, pages 72{81, Tokyo, Japan, April 1991.

[16] Fredrik Dahlgren, "Boosting the performance of hybrid snooping cache protocols", In Proceedings of the 22th Annual International Symposim on Computer Architecture, pages 60–69, 1995

[17] Jack E. Ve e n stra and Robert J. Fowler, "The prospects for on-line hybrid coherency protocols on bus-based multiprocessors", Technical Report TR490, University of Rochester, Computer Science Department, March 1994.

[18] W. Hu, W. Shi, and Z. Tang. A lock-based cache control protocol for scope consistency. *Journal of Computer Science and Technology*, 13(2), March 1998.

[19] Silvia Lametti, "Cache Coherence Techniques", A Technical Report, December 2010

[20] J. Li, W. Liu, P. Jiao, "A new kind of cache coherence protocol with SC-cache for multiprocessor", IEEE 2010

[21]] J. Li, P. Yang, N. Ding ,H. Guan, J. Zhang, C. Men,"A New Kind of Hybrid Cache Coherence Protocol for Multiprocessors with D-Cache", IEEE 2011