IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-5 E-ISSN: 2347-2693

Comparative Study and Performance Analysis of Cache Coherence Protocols

S. Kumar[1*], K. Gupta[2]

[1]Dept. of Computer Science and Engineering, Rajkiya Engineering College, Kannauj, India
[2]Dept. of Computer Science, GLS University, Ahmedabad, India

[*]*Corresponding Author: swadheshkumar@gmail.com*

Online Available at: www.ijcseonline.org

Received: 24/Apr/2017, Revised: 30/Apr/2017, Accepted: 22/May/2017, Published: 30/May/2017

***Abstract*—** Cache memory is a small less access time semiconductor memory that sits between the processor and memory in the memory hierarchy to bridge the speed mismatch between processor and main memory. Multiprocessor System contains multiple processors working simultaneously and share memory. Multiprocessors are most widely used in computational devices due to their reliability and throughput. In multiprocessor system maintaining data consistency is an important parameter to be maintained because different processors communicate and share data. In multiprocessors caching plays a vital role because cache Coherence is a problem that should be handled very carefully. In this paper we have studied various Cache Coherence Protocols and simulate their behavior on various platforms on the basis of miss rate.

***Keywords*—** MSI, MESI, DRAGON

I. INTRODUCTION

Multiprocessor Systems have better performance as compared to single processor systems because these are tightly coupled systems which can process multiple jobs simultaneously without interfering jobs running on other parallel processors. Processors in Multiprocessor Systems can communicate each other due to shared address space. In Multiprocessor Systems each processor has a separate cache, so there is identical cache entry exists in other processors of Multiprocessor Systems due to shared address space[1]. Fig. 1 shows the Shared memory multiprocessor System.

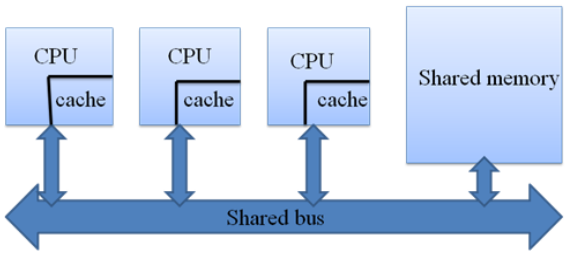

Figure.1 Shared memory multiprocessor system

In Multiprocessor Systems sharing of data does not create any problem during memory read operation but there may be a problem during write operation because when a processor of Multiprocessor Systems writes a value to a location that is being shared, the changed value must be updated to all other caches of different processors using it otherwise caches of different processors hold different data for the same location which is called cache coherence problem [2].

Suppose three processors P1, P2, P3 of a Multiprocessor System are sharing a memory space [3]. Here if Processor P1 wants to read a value at a location X from shared memory then it caches its value in to P1. If Processor P2 also wants to read a value at location X from shared memory then it caches its value in to P2 as shown in Figure 2.

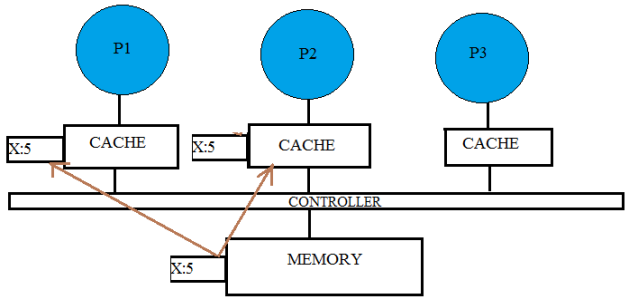

Figure.2 Processor P1and P2 reading from shared memory

Now if the processor P1 wants to write a value at location X in shared memory. This is shown in Fig.3. Now if processor P3 performs read operation on location X. In p1 at location X, value stored is 10 and in p2 at location X, the data stored is 5, therefore data inconsistency arises when we perform write operation to a shared address location, Now it may create a problem because here the situation also depends on writing strategy if it is write through then content or value at location X in shared memory will be 10 and if it is write back strategy then value in shared memory will be 5. This is shown in Fig.4.

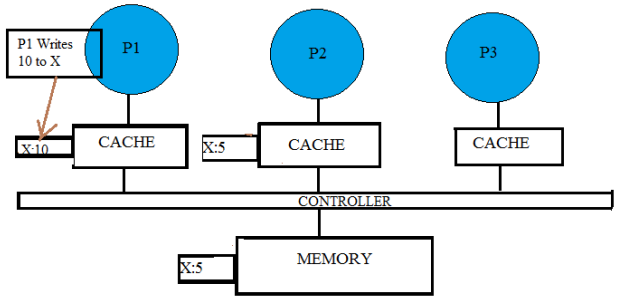

Figure.3 Processor P1 writes at location X

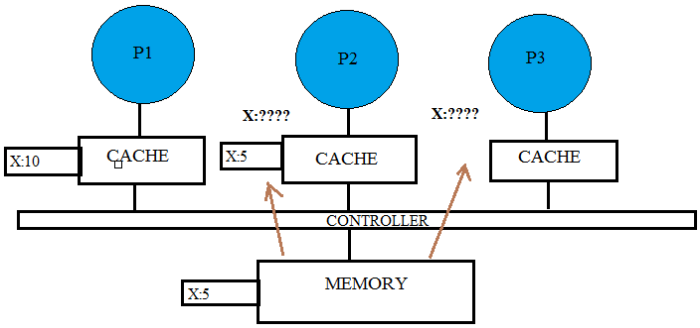

Figure.4 Data inconsistency in multiprocessor systems

II. CACHE COHERENCE PROTOCOLS

There are two methods to solve cache coherence problem one is Software Solution and other is Hardware Solution. In Software Solution, the detection of cache coherence problem is transferred from run time to compile time, and the overall design complexity is also transferred from hardware to software. One of the Software Approach is Compiler Based Cache Coherence Method, which perform an analysis on the code and discover various data which may become unsafe for caching. The main disadvantage of Software Solutions is inefficient cache utilization.

Hardware solutions for cache coherence problems provide dynamic recognition at run time of inconsistency conditions. In Hardware Solutions the problem is treated when inconsistency arises. Hardware Solutions leads effective cache utilization and improved performances over a software Based Solution. Hardware Solutions can be categories in to two types: Snoopy Protocol and Directory Protocol

A. Snoopy Protocols

The responsibility for maintaining cache coherence is distributed among all of the cache controllers. In Snoopy Protocols a cache must recognize when its Data or line is shared with caches of other Processors[4]. If any update or write action is performed on shared cache then it must be informed to other caches also by using a broadcast mechanism using Bus. There are mainly two basic types of snoopy protocols

1) Write Invalidate:

In write this protocol, there can be multiple read actions but only one write action at a time. Here, if cache need to perform a write action to a line it first generate a notice which invalidates that line in the other caches, making the line exclusive to the writing cache. If the line is exclusive then the owner processor can locally writes until some other processor needed the same line [5].

2) Write Update or Write Broadcast:

In this protocol, there can be multiple readers as well as multiple writers. If a processor updates a shared line then updated line is distributed to all other processors and caches containing that line can update it.

There are various types of Cache Coherence Protocols as:

B. MSI Protocol:

MSI Protocol is a three state write back invalidation technique. It marks the cache line in Shared (S), Invalid (I) and Modified (M) state. When cache line is not present then it is marked as Invalid. When cache line is clean and shared by more than one processor then it is marked as Shared. When the cache line is dirty and a processor has exclusive ownership of cache line then it is marked as Modified state[6]. BusRdx responsible for making others to invalidate I state. If it is present in M state in another cache, it will flush. A BusRdx, even if it causes a cache hit in S state, is promoted to M (upgrade) state.

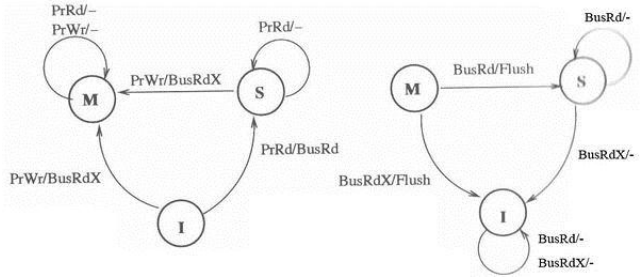

Figure.5 State Diagram of MSI Protocol

From the state transition diagram of MSI, we observe that there is transition to state S from state M when a BusRd is observed for that block. The content of the block is flushed to the bus before going to S state. It would look more appropriate to move to I state thus giving up the block entirely in certain cases. This choice of moving to S or I reflects the designer's assertion that the original processor is more likely to continue reading the block than the new processor to write to the block.

C. MESI Protocol:

The disadvantage of MSI is that each read-write sequence incurs two bus transactions. MESI protocol solves this by introducing a new Exclusive state to differentiate between a cache line stored in multiple caches and a line stored in a single cache. MESI coherence protocol marks each cache line in of the Modified, Exclusive, Shared, or Invalid state.

- Invalid: When cache line is not present then it is marked as Invalid
- Exclusive: The cache line is clean and is owned by this processor only
- Modified: When the cache line is dirty and a processor has exclusive ownership of cache line then it is marked as Modified state
- Shared: When cache line is clean and shared by more than one processor then it is marked as Shared.

In MESI Protocol a line that is fetched, receives E (Exclusive), or S (Shared) state depending on whether it exists in other processors in the system. A cache line gets the M(Modified) state when a processor writes to it; if the line is not in Exclusive or Modified state prior to writing it, the cache sends a Bus Upgrade (BusUpgr) signal or as the Intel manuals term it, "Read-For-Ownership (RFO) request" that ensures that the line exists in the cache and is in the I state in all other processors on the bus (if any). A table is shown below to summarize the MESI protocol.

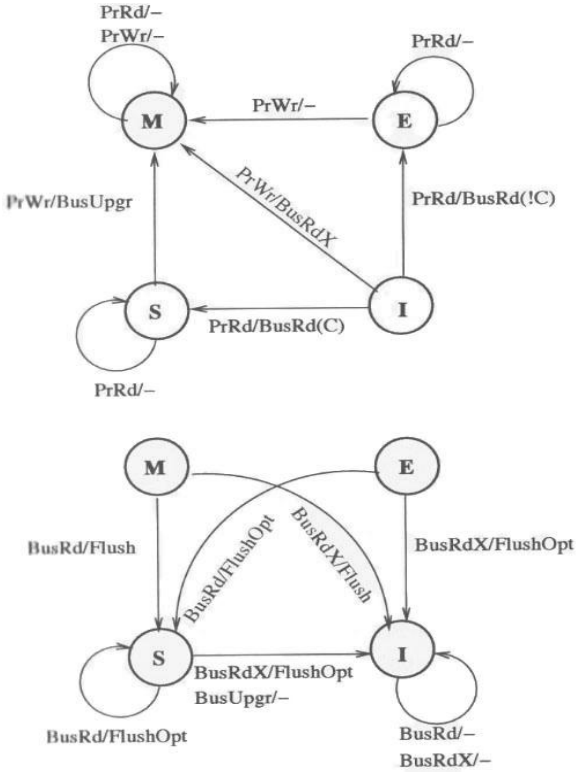

Figure.6 State Diagram of MESI Protocol

TABLE I. STATE TRANSITION TABLE FOR MESI PROTOCOL

State of Cache Line:	Modified	Exclusive	Shared	Invalid
Valid cache line	Yes	Yes	Yes	No
Memory copy is	out of date	Valid	Valid	
Is Copies also exist in other processors caches	No	No	Maybe	Maybe
Write to this line	does not go to bus	does not go to bus	goes to bus and updates cache	goes directly to bus

D. Dragon Protocol:

Dragon is a cache coherence protocol having four states. Dragon differentiates between Shared Modified state (Sm) and Shared Clean (Sc) states. Figure 7 represents the state transition diagram for Dragon Protocol.

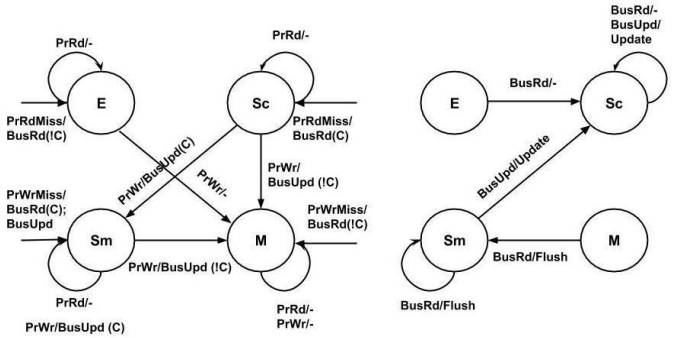

Figure.7 State Diagram of DRAGON Protocol

III. EVALUATION METHODOLOGY

We have used SMP3.0, a trace driven simulator for performance analysis of some mostly cache replacement policies [7]. The Trace driven simulator is a cost effective way of performance evaluation of computer system design, specially for cache memory design, TLB, and paging system. In this paper, we have used various Benchmarks for the performance analysis such as: FFT64, SIMPLE64, SPEECH64 and WEATHER64. Here, we have done simulations of various Cache Coherence Protocols on the Basis of Miss Rate.

IV. SIMULATION SETUP

Simulator configuration for Cache Replacement Policies
Number of Processors = 4
Cache Coherence Protocol = MSI, MESI, DRAGON
Bus Arbitration = LRU
Word Wide (bits) = 16
Blocks in Main Memory = 524288

Block size = 64 bytes
Main Memory size = 32 M Bytes
Blocks in Cache = 256
Cache size = 16 KB
Mapping = Fully Associative
Writing Strategy = Write Back
Replacement Policies =LRU

TABLE II. EXPERIMENTAL RESULT

BENCHMARK	Coherence Protocol		
	MSI	MESI	DRAGON
FFT64	8.824	8.91	0.272
SIMPLE64	22.035	22.035	0.548
SPEECH64	7.367	7.371	0.922
WEATHER64	13.653	13.822	1.103

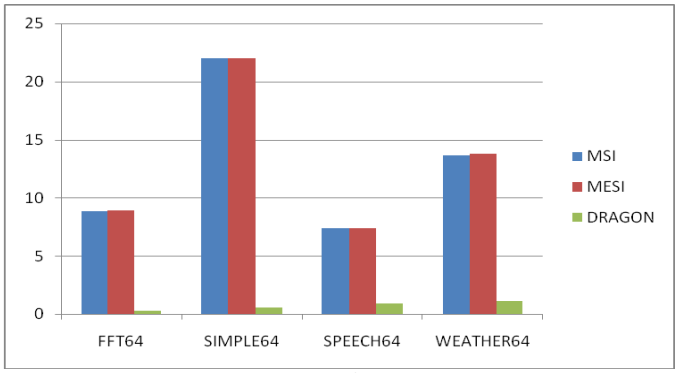

Figure.8 State Diagram of DRAGON Protocol

Table II shows the simulation results of various Cache Coherence Protocols in the form of Miss Rate on different Benchmarks. Here, all the Cache Coherence Protocols have different Miss Rate we have found that DRAGON Cache Coherence Protocols has better performance than other Cache Coherence Protocols across almost all the Benchmarks. DRAGON Cache Coherence Protocols is an update based Cache Coherence Protocol, while MSI and MESI are invalidation based protocols. These two Cache Coherence Protocols are different. The differences between invalidation based protocol and update based protocol are laid on the update condition and write operation. The updates conditions in update based update based protocol are improved to a single word write rather than transfer of full cache block. While in invalidation based Cache Coherence Protocols, on a write operation the cache state of that memory block in all other processor caches is set to invalid, so those processors will have to obtain the block through a miss (a coherence miss)

V. CONCLUSION

The simulation results show that the use of a good cache coherence protocol improves the overall performance of multiprocessor system. More miss rate is resulted due to use of large number of processors for a parallel application. By simulating on SMP Cache Simulator we can see the influence of the cache coherence protocol on miss rate. Here, all the Cache Coherence Protocols have different Miss Rate. We have found that DRAGON Cache Coherence Protocol has better performance than other Cache Coherence Protocols across almost all the Benchmarks. The results and conclusions obtained with these experiments are of general application.

REFERENCES

[1] K.D. Kohle, U.M. Gokhale, D. Pendhari, "*Design of cache controller for multicore systems using parallelization method*", IEEE Proceedings, Vol.86, Issue.5, pp.837-52, 2014.
[2] B. Dubois, *"Effects of cache coherency in multiprocessors"*, IEEE Transactions on computers, Vol.31, Issue.11, pp.1083-1099, 1982.
[3] D.J. Lilja, "*Cache coherence in large scale shared memory multiprocessors*", ACM Computing surveys, Vol.25, No.3, pp. 303-338, 1993.
[4] M. Thapar, B. Delagi, "*Standford distributed-directory protocol*". Computer, Vol.23, Issue.6, pp.78-80, 1990.
[5] R.E. Ahmed, M.K Dhodhi, "*Directory-based cache coherence protocol for power-aware chip-multiprocessors*", Canadian Conference Electrical and Computer Engineering (CCECE), Canada, pp.001036, 001039, 2011.
[6] S. Almakdi, A.W. Alazeb, M. Alshahari, "*Cache coherence mechanisms",* International journal of engineering and innovative technology, Vol. 4, Issue.7, pp.7-13, 2015.

Authors Profile

Mr. Swadhesh Kumar pursed Bachelor of Technology from NDUAT University of Faizabad, India in 2014 and Master of Technology from MMM University of Technology Gorakhpur, India in year 2016. He is currently working as Assistant Professor in Department of Computer Science and Engineering, Rajkiya Engineering College Kannauj, India since August 2016. He has published many technical papers in reputed international journals and conferences including SPRINGER, IEEE and it's also available online. His main research work focuses on Memory Management, Cloud Security, Big Data Analytics.

Mrs. Kreetika Gupta pursed Bachelor of Technology from IET, Alwar, Rajasthan in 2012 and Master of Technology from MMM University of Technology Gorakhpur India in year 2016. She is currently working as Visiting Lecturer in Department of Computer Science of GLS University, Ahmedabad Gujarat. She has published many technical papers in reputed international journals and conferences and it's also available online. Her main research work focuses on Wireless Sensor Network, Memory Management, Network Security, Big Data Analytics.

