International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)

http://ijcjournal.org/

RWMSI (Read Exclusive Write Exclusive Modified Shared Invalid) Cache Coherence Protocol

Luma F. Jalil[a]*, Abeer D. Al-Nakshabandi[b]

[a]*Al Rasheed University College, Al-Hussein neighborhood, Baghdad10001, Iraq*
[b]*Distribution office At Ministry of Electricity, Baghdad 10001, Iraq*
[a]*Email: luma.jalil@alrasheedcol.edu.iq*
[b]*Email: abeerdiaaphd@gmail.com*

Abstract

This paper proposes a novel coherence protocol RWMSI (Read exclusive Write exclusive Modified Shared Invalid) that merges "snooping and directory – based coherence protocols "and enhanced them depending on the state of "MESI snooping protocol ". " Coherence" is implemented with "snooping or directory based protocols ". Because of the shared bus the " Snooping protocols " are not scalable , while directory protocols incur directory storage overhead , frequent indirections , and are more prone to design bugs.

Keywords: Computer Architecture; Cache Simulator; MESI; RWMSI; visual C++.

1. Introduction

To improve the efficiency of a processor to work with data, cache memories are used to compensate the latency delay to access data from the main memory. But because of the installation of different caches In different processors in shared memory architecture makes it very difficult to maintain consistency between the cache memories of different processors. For that reason, having a cache coherency protocol is really essential in those kinds of system. There are different coherency protocols for caches to maintain consistency between different caches in a shared memory system. Few of the famous cache coherency protocols are MSI , MESI , MOSI , MOESI . In modern processor architecture, because of the speed difference between the main memory and processor, it might take too many cycles for a processor to access the main memory. As a result this may cause hindrance in performance. So to deal with this problem, faster memories can be installed in the system in order to store the data from the main memory for frequent use by the processor .theses fast memories which are either on chip or off –chip to improve latency and performance, are called cache memories [1].

---

* Corresponding author.

Because of high degree of locality in most programs i.e. if a processor reads or write a memory address (memory location), then there is a high probability that the processor might read or write the same location again very soon. Another feature is that if a processor read or writes a memory location then there is a probability to read or write nearby locations also. To exploit the second behavior, caches may operate by holding a group of neighboring data know as cache (also called cache blocks). Now , if system has many processors and those different processors have different cache memories and the data from the main memory is shared with different caches of different processors then it might give rise to high inconsistency if there is any change in the shared data even in one of the caches . But if the processors only read from the same memory address then there is no problem of inconsistency. for example if one processor tries to update the shared cache line then the copy of that cache line in other processors have to be invalidated in order to make sure that other processors do not read an out – of data value of the modified cache line . This problem is called cache coherency. To address the cache coherency problem, there are many protocols to deal with this [1, 5]. This paper gives basic idea of how to design and implements a simple cache simulator to implement RWMSI cache coherence protocol by using Visual C++ language. In cache simulator it must determine the number of cache levels. The user can indicate the whole cache memory parameters such as the "cache memory capacity" for each cache level, "the cache line size", "associativity", "the replacement policy" , "the number of words per memory access" , "the writing policy" , etc Moreover the user can configure the statistics of the memory access to study the number of cache misses and their type the number of memory access hits, etc. Then we will be reviewing the basic working of RWMSI protocols. The RWMSI protocol is a method to maintain the coherence of the cache memory content in hierarchical memory system. It is based on five possible states of the cache blocks: Read Exclusive, Write Exclusive, Modified, Shared and invalid. Each accessed block is in one of these stage and the transitions among them define the RWMSI protocol. Finally the user can study the changes of memory content and the transitions among the five RWMSI states at the simulation time.

2. Cache Coherency

Cache coherency or " Cache consistency " the synchronization of data in multiple caches such that reading a memory location via any cache will return the most recent data written to that location via any other cache. There are many problem can be accrued when using cache for multiprocessor system [6]:

- It is possible for more than one processor to cache an address at the same time because all the processors share the same address space
- Inconsistency may result and cause incorrect execution if one processor updates the data item without informing the other processor in order to eliminate the problem of cache coherency in the memory system some basic protocols are adapted such as snooping protocol and directory protocol as illustrated in the Following [4,6]:

- Bus – Snooping Protocols

This protocol (Not scalable) is used in "bus – based system" where all the processors observe memory transctions and take proper action to " invalidate" or "update" the local cache " content if needed .

- Directory Schemes :

This protocol is used in “scalable cache-coherent distributed memory multiprocessor systems “ where cache directories are used to keep a record on where copies of cache blockes are reside .

3. MESI Protocol

The MESI protocol (Papamarcos & patel 1984) is a version of the snooping cache protocol. It is widely used cache coherence and memory coherence protocol. The MESI protocol is based on the four state that a block in the cache memory can have . these four state are the abbreviations for MESI, The cache line can be in one of four state , this one can takes 2 bits the state are illustrated as following [2,4] :

- “Modified “: the cache contains a copy which differs from that in memory but there are no other copies the cache line has been modified is different from main memory and is dirty; with respect to system memory.
- “Exclusive“: cache line is the same as main memory (means is the only cached copy, but is “clean”).
- “Shared “: same as main memory but copies may exist in other caches.
- “Invalid”: means that this line of cache is invalid (unused).

When a cache block changes its status from M, it first updates the main memory. The permitted state of a given cache lines are as follows in Table 1. “for any given pair of cache “.

Table 1: The permitted state of a given cache lines

	M	**E**	**S**	**I**
M	×	×	×	✓
E	×	×	×	✓
S	×	×	✓	✓
I	✓	✓	✓	✓

Table 2 below explains this in a more detailed way the state and their characteristics [2,7].

Table 2: The way the state and their characteristics.

	Modified	**Exclusive**	**Shared**	**Invalid**
Line valid	Yes	Yes	Yes	No
Copy in memory	Has to be updated	Valid	Valid	
Othe copies inother caches	No	No	Maybe	Maybe
A write on this line	Access the BUS	Access the BUS	Access the BUS and Update the cache	Direct access to the BUS

Figure 1 below illustrates the MESI Write back invalidation.

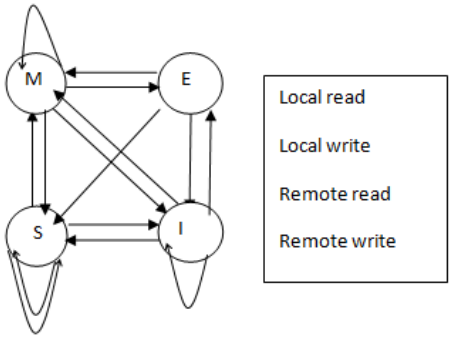

Figure 1: The MESI Write back invalidation.

Figure 2 illustrates MESI state diagram, S: Shared Signal, Processor-initiated, Bus-snooper-intiated, Flush*: Flush for data supplier; no action for other sharers:

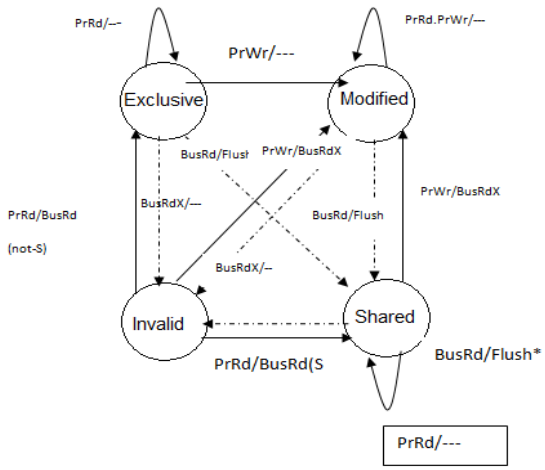

Figure 2: MESI state diagram, S: Shared Signal, Processor-initiated, Bus-snooper-intiated, Flush*: Flush for data supplier; no action for other sharers.

Note: "MESI state diagram". PrRd = processor Read – Read request from processer, PrWr = processer Write – Write request from processor, BusRd = Bus – Read request from the bus without intent to modify. The "S" denotes that the shared signal was asserted by another cache. BusRdx= Bus Read Exclusive – Read request from the bus with intent to modify. The transitions are labeled "action observed\action performed " [3].

Table 3: MESI protocol corresponding to example of figure 3.

Event	**In p1'S cache L=invalid**	**In p2's cache L=invalid**
P1 writes 10 to B1 (write miss)	L←B1=10 (Modified)	L=invalid
P1 reads B1 (read hit)	LB←10 (Modified) **B1 is written back**	L=invalid
P2 reads B1 (read miss)	LB←10 (Shared)	LB←=10 (Shared)
P2 writes 20 to B1 (write hit)	L=invalid	LB←=20 (Modified)
P2 writes 40 to B2 (write miss)	L=invalid	**B1 is written back** L←B2=40 (Modified)
0P1 reads B1 (read miss)	L←B1=20 (Shared)	L←B2=40(Modified)
P1 writes 30 to B1 (write hit)	L←B1=30 (Modified)	L←B2=40(Modified)
P2 writes 50 to B1 (write miss)	L=invalid	**B2 is written back** L←B1=50(Modified)
P1 reads B1 (read miss)	LB←50 (Shared)	**B2 is written back** L←B1=50(Shared)
P2 reads B2 (read hit)	LB←50 (Shared)	L←B2=40(Modified)
P1 writes 60to B2 (write miss)	L←B2=60 (Modified)	L=invalid

Figure 3 (a, b, c) below a, b and c states are in one example, Table 3 of MESI protocol of the example as shown below

- Assumes that blocks B1 and B2 map to same cache location L.
- Initially neither B1 nor B2 is cached
- Block size = one word

Example of MESI protocol

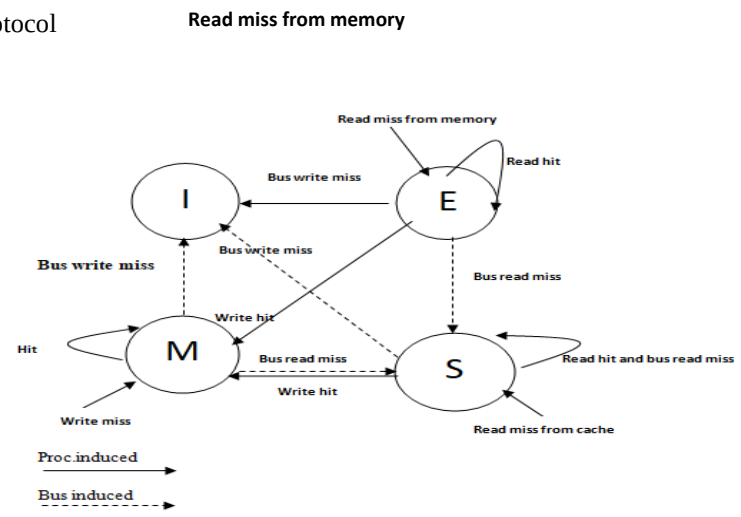

Figure 3: (a) MESI protocol (P2 reads A (A only in memory))

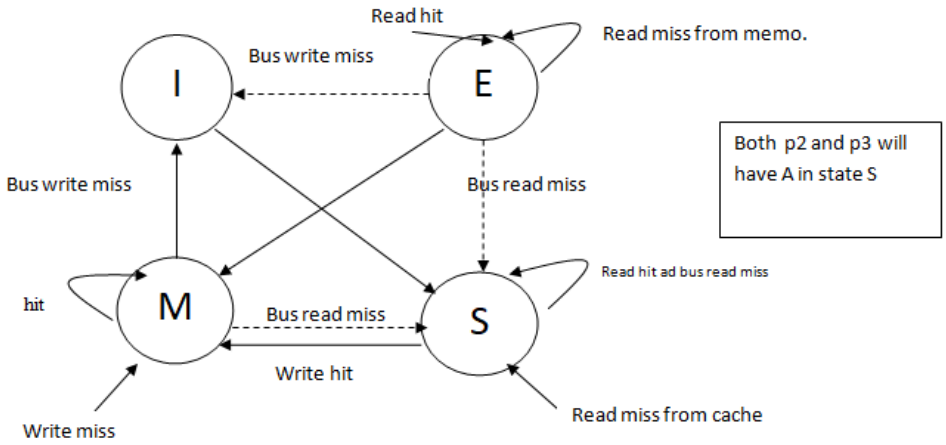

Figure 3: (b) MESI protocol (P3 reads A (A comes from P2))

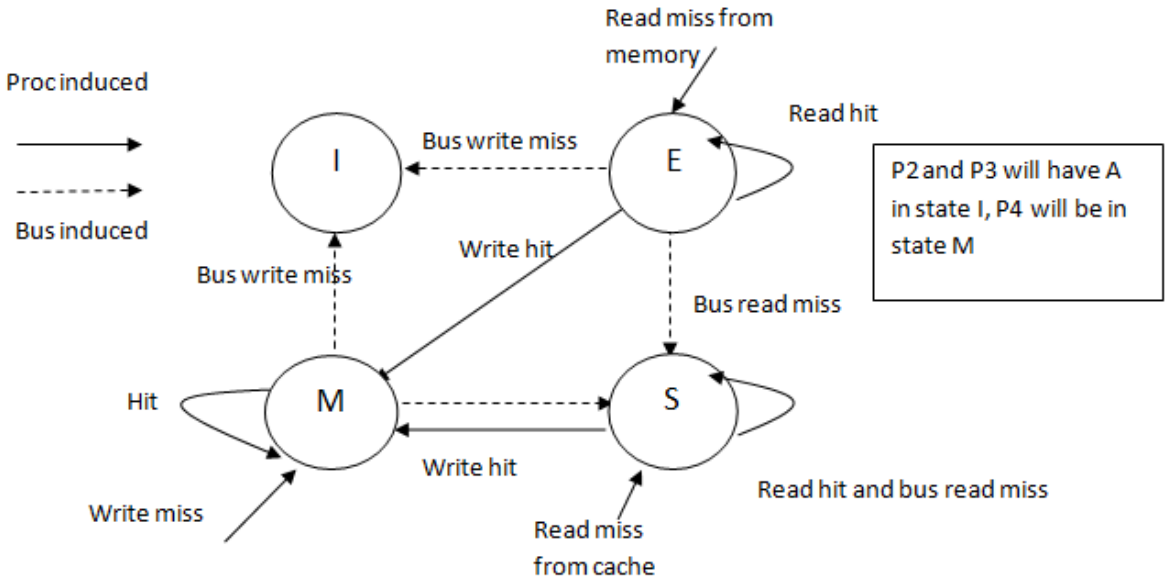

Figure 3: (c) MISE protocol (P4 writes A (A comes from P2))

4. RWMSI proposal protocol

4.1. RWMSI levels

The proposed project deals with a multi processor that has two cache levels:

1- The first level is private L1 and supposes we distributed this cache in this level between three cores
2- The second Level is shared global L2 cache within a three core in cache L1

4.2. RWMSI states

In this protocol each line has one of the five states as in figure 4 below:

Modified

It is similar to modified state in MESI protocol the different that the processor Read–Read request from processor does not occur in this state. When the processer request read the state translate to the Read exclusive state. The idea in this protocol that the read request isolated from write request.

Read Exclusive

The cache line is present only in the current cache but is clean ; it matches main memory . it may be changed to the shared state at any time , in response to a read request .

Write Exclusive

The cache line is present only in the current cache and appears at the first time of writhing, when the writing repeated locally it goes to the modified state. The cache line matches main memory. It may be changed to the shared state at any time in response to a read request remotely.

Table 4: Contains the new state as a result of a request and a processor job on a previous state that illustrated in RWMSI transition state diagram

Seq	Previous state	Request	Processor job	New state
0	I	RMS	Local read	S
1	I	RME	Local read	R(Invalidate the rest core)
2	I	WME	Local write	W(Invalidate the rest core)
3	R	RH	Local read	R
4	R	RH	Remote read	S
5	R	SHI	Remote write	I
6	R	WH	Local write	W(Invalidate the rest core)
7	S	RH	Local or remote read	S
8	S	SHI	Remote write	I
9	S	WH	Local write	W (Invalidate the rest core)
10	W	RH	Local read	R
11	W	SHR	Remote read	S
12	W	SHI	Remote write	I
13	W	WH	Local write	M
14	M	WH	Local write	M
15	M	SHR	Remote read	S
16	M	SHI	Remote write	I
17	M	RH	Local write	R

Shared

Indicate that this cache line may be stored in other cache of the machine and is clean; it matches the main memory. The line may be discarded (changed to the invalid state) at any time.

Invalid

Indicate that this cache line is invalid (unused) The following table 4 contains the new state as a result of a request and a processor job on a previous state that illustrated in RWMSI transition state diagram (figure 4 in subsection 4.3)

4.3. RWMSI transition state diagram

Figure 4 illustrates RWMSI [Read exclusive Write exclusive Modified Shared Invalid] cache coherence protocol. This protocol has been used bus snooping protocol that appear when each state translate to other state, the abbreviate symbols of these buses are follow:

Bus transaction:

Commit = Write cache line back to memory

Invalidate = Broadcast invalidate

Read = Read cache line from memory

Events:

RH = Read Hit

RMS = Read Miss, Shared

RME = Read Miss, Exclusive

WH = Write Hit

WM = Write Miss

WME = Write Miss, Exclusive

SHR = Snoop Hit on Read

SHI = Snoop Hit on Invalidate

Here, table 5 illustrates the discussion of Theoretical example of RWMSI (the same of example 2 in MESI)

- Assumes that blocks B1 and B2 map to same cache location L
- Initially neither B1 nor B2 is cached

Block size = one word

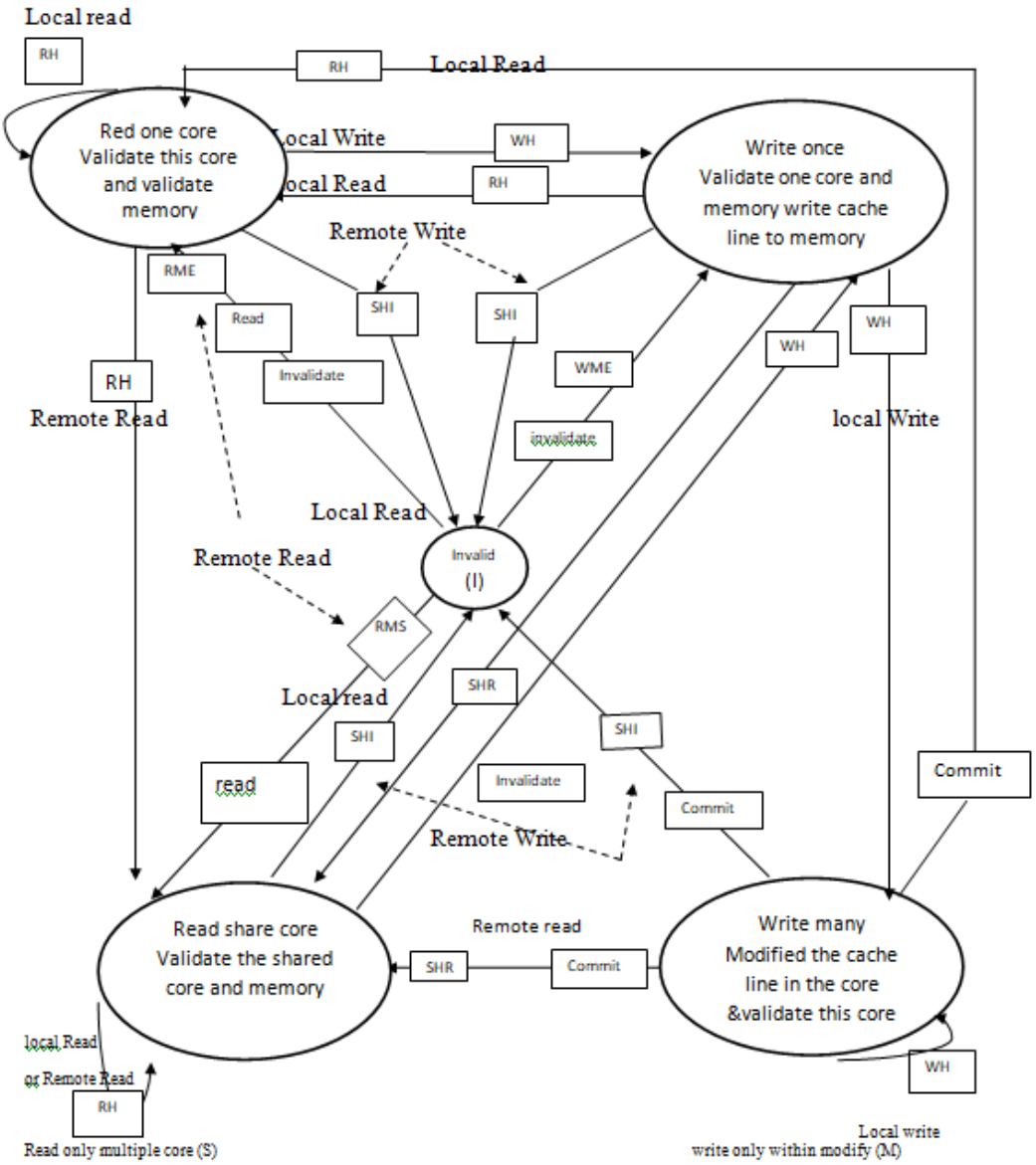

Figure 4: RWMSI [Read exclusive Write exclusive Modified Shared Invalid] cache coherence protocol.

In this proposed protocol the directory cache coherence protocol used as follow:

a) The directory is located inside the cache level 2 that will be shared between all cores in cache level 1
b) The number of bit in a directory is 16 bit distributed as the following :
c) 1 bit to represent as a valid or not, 7 bit to represent main memory address [memory address = 128] , 3 bit to represent tag , 2 bit to represent index [cache = 4 block] , 2 bit to represent offset [block = 4

byte] the directory contains 3 bit to represent the presence of a cores in the case of a shared state , as shown in table 6.

Table 5: Illustrates the discussion of Theoretical example of RWMSI (the same of example 2 in MESI)

Event	In p1's cache L=invalid	In p2's cache L=invalid
P1 writes 10 to B1 (write miss)	L←B1 = 10 (W) B1 is written through	L = invalid
P1 reads B1 (read hit)	L←B1 = 10 (R)	L = invalid
P2 reads B1 (read miss)	L←B1 = 10 (Shared)	L←B1 = 10 (Shared)
P2 writes 20 to B2 (write hit)	L = invalid	L←B1 = 20 (W) B1 is written through
P2 writes 40 to B2 (write miss)	L = invalid	L←B2 = 40 (W) B2 is written through
P1 reads B1 (read miss)	L←B1 = 20 (Shared)	L←B2 = 40 (W)
P1 writes 30 to B1 (write hit)	L←B1 = 30 (W) B1 is written through	L←B2 = 40 (W)
P2 writes 50 to B1(write miss)	L = invalid	L←B1 = 50 (M) B1 is written back
P1 reads B1 (read miss)	L←B1 = 50 (Shared)	L←B1 = 50 (Shared)
P2 reads B2 (read hit)	L←B1 = 50 (Shared)	L←B2 = 40 (R)
P1 write 60 to B2 (write miss)	L←B2 = 60 (W) B2 is written through	L = invalid

Table 6: The directory cache coherence protocol of the proposed protocol

Decimal number	Binary representation			Shared core
0	0	0	0	None of them
1	0	0	1	P1
2	0	1	0	P2
3	0	1	1	P3
4	1	0	0	P1,P2
5	1	0	1	P1,P3
6	1	1	0	P2,P3
7	1	1	1	P1,P2,P3

Finally the RWMSI state are 18 states in a transition diagram , so the number of bits that will be selected are 5 bit as shown in table 7 Notes: Each core in a proposed must have also 16 bit for only locally representing.

Table 7: The RWMSI state (18 states)

Decimal number	Binary representation					Previous state	New state
0	0	0	0	0	0	I	S
1	0	0	0	0	1	I	R
2	0	0	0	1	0	I	W
3	0	0	0	1	1	R	R
4	0	0	1	0	0	R	S
5	0	0	1	0	1	R	I
6	0	0	1	1	0	R	W
7	0	0	1	1	1	S	S
8	0	1	0	0	0	S	I
9	0	1	0	0	1	S	W
10	0	1	0	1	0	W	R
11	0	1	0	1	1	W	S
12	0	1	1	0	0	W	I
13	0	1	1	0	1	W	M
14	0	1	1	1	0	M	M
15	0	1	1	1	1	M	S
16	1	0	0	0	0	M	I
17	1	0	0	0	1	M	R

5. The comparison between MESI protocol and RWMSI protocol

The following points show the important comparison between the proposal protocol and MESI protocol.

First point:

The processor requests enter modified state in MESI are :

a- Any of the local writing whether occur – time or more
b- Also in the case of local reading comes after writing

Every time downgrade from “M“ state to “S“ needs data to be written back to Memory in all of the previous request , which is considered one of the Disadvantage of this protocol .

The processor request enter modified state in RWMSI are in only one case:

- When local writing occur more than once

- While in the other cases:

If local write comes ones it enter W state If local read comes it enter R state Therefore in this protocol the write back to memory occur lesser in this protocol than in MESI when converting from "M"state to "S"

Second point

In RWMSI, the write – through policy has been used more than write – back

Write through: The information is written to both the block in the cache and Can always discard cached data – most up – to –date data is in memory Cache control bit: only a valid bit WT always combined with write buffers so that don't wait for lower level memory

Write back: The information is written only to the block in the cache . the modified cache block is written to main memory only when it is replaced Cache just discard cached data – may have to write it back to memory Cache control bits both valid and dirty bits

Pros and cons of each:

- WT : read misses do not need to write back evicted line contents
- WB: no writes of repeated writes

Other RWMSI Advantages:

Write- through: Memory (or other processors) always have latest data

Simpler management of cache

Write-back: Much lower bandwidth, since data often overwritten multiple times

Better tolerance to long- latency memory.

6. Conclusion and future works

In this paper we propose RWMSI protocol that it is used to achieve coherency. The cache coherence protocol is one of the major factors influencing the performance of multi-core computer systems. The coherence protocol must be selected based on the chip architecture and the performance that the system wants to achieve. RWMSI protocol is an extension for MESI protocol, which minimizes a write back to main memory by separating a reading state from writing state and a multi write from single write. In future work we tried also to find other state that minimizes accessing to main memory depending on MESI protocol states.

References

[1]. Chang ,Y. “Cache memory protocols”, wiley encyclopedia of electrical and electronics engineering, 1999.

[2]. Jimenez, F.J., Gomez, J., Mesons, A. , Hertzog, E., Benavides, J.I., Y. Sanches,F.J. “Teaching the cache memory coherence with the MESI protocol simulator” , p.2 ,Spain, 2006.

[3]. Lametti, S. “cache coherence techniques” , master program in computer science and networking, p.6-11 , technical report, December 2010.

[4]. Lorenzo del Castillo, J.A. “performance counter-based strategies to improve data locality on multiprocess system: re-ordering and page migration techniques“, PHD dissertation, university of Santiago de compostela, p.23-28, 2011.

[5]. Magnusson, P. S. “Efficient instruction cache simulation and execution profiling with a threaded-code interpreter, Swedish institute of computer science, BOX 1263, S-164,28, KISTA, SWEDIN.

[6]. Marty, M. R. “Cache coherence techniques for multicore processors”, PHD thesis, university of Wisconsin-madison, 2008 .

[7]. SUH, T., Lee, H. S. & Blough, D. M. “Integration cache coherence protocols for heterogeneous multiprocessors systems”, Part 2. IEEE Micro, 24(5), 70-78. https://doi.org/10.1109/MM.2004.50

.