JUHD / Vol. 3, No.1, March 2017: pp 274-281
DOI: 10.21928/juhd.20170315.13, e-ISSN: 2411-7757, p-ISSN 2411-7765

Cache coherence protocol design using VMSI (Valid Modified Shared Invalid) states

Asst. Prof. Dr. Luma Fayeq Jalil , Asst. Prof. Dr. Maha Abdul kareem .H. Al-Rawi , Abeer Diaa Al-Nakshabandi

***Abstract*— We have proposed in this research the design of a new protocol named VMSI coherence protocol in the cache in order to solve the problem of coherence which is the incompatibility of data between caches that appeared in recent multiprocessors system through the operations of reading and writing. The main purpose of this protocol is to increase processor efficiency by reducing traffic between processor and memory that have been achieved through the removal of the write back to the main memory in the case of reading or writing of shared caches because it depends on existing directory inside that cache which contains all the data that represents a subset of main memory.**

***Index Terms*— Cache coherence problem, snooping protocol, Directory-Based cache Protocols, VMSI, Cache Simulator, Shared memory, Multi processor, Dev. C++.**

I. INTRODUCTION

Shared memory is the hardware part that supported by many modern computer systems and multi core chips. Each of the processor cores in a shared memory system may read and write a single address space [1]. But in designing shared system, one of the most important problems appears which is called coherence problem. The coherence problem results when the caches are laid in recent computers between processor and main memory to solve the contention problem as trying to access a shared memory at the same time which then causes performance degradation[2].

Consistency and validation of the data value are maintained in the caches of a multi core processor such that reading a memory location through any caches will return the most recent data written to that location via any caches through cache coherence protocol that are classified to snoopy protocols or directory based protocols. Coherence is typically implemented within these protocols, but both methods have a drawback: Snooping protocols are not scalable because of the shared bus, while directory protocols incur directory storage overhead, frequent indirections, and are more prone to design bugs [3].

Asst. Proff. Dr. Luma Fayeq Jalil , Department of Information Technology, university of human development, college of science and technology /sulemani, luma.jalil@uhd.edu.iq ,

Asst. Proff. Dr. Maha Abdul kareem .H. Al-Rawi , Department of Computer Sciences, University of Technology/Baghdad, maha_alrawi@yahoo.com,

Head of programmers oldest Abeer Diaa Al-Nakshabandi, Distribution office At Ministry of Electricity/Baghdad, abeerdiaaphd@gmail.com

II. THE REQUIREMENTS OF MEMORY HIERARCHY DESIGNE

The basic idea to overcome the problem of increasing the gap between a fast CPU and a slow RAM is in using a hierarchy of memories: each level speedier, more expensive and smaller, the closer it is to CPU, to feed the CPU with the required data. The following steps are needed in designing a hierarchy of memory [4, 5]:-

A. CACHE ASSOCITIVITY

Where Can a Block Be Placed in a Cache?, Just as bookshelves come in different shapes and sizes, caches can also take on a variety of forms and capacities. But no matter how large or small they are, caches fall into one of three categories: direct mapped, n-way set associative, and fully associative.

Direct mapped: A cache block can only go in one location in the cache. It makes a cache block very easy to find, but it's not very flexible about where to put the blocks. The mapping equals to (Block address) MOD (Number of blocks in cache), two types of association: the cache is a fully associative is if a block can be placed anywhere in the cache. The cache is set associative if a block can be placed in a restricted set of places in the cache.

A set is a collection of blocks in the cache. A block is first mapped onto a set, and then the block can be placed anywhere within that set. The set is usually chosen by bit selection = (Block address) MOD (Number of sets in cache)

If there are n blocks in a set, the placement of the cache is called n-way set associative.

Each of these methods depends on the facts:

2^nMemory locations are grouped into blocks where n is the number of bits used to identify a word within a block and are found at the least significant end of the physical address. The cache is organized into:

1. Index: specifies the cache index (which “row” of the cache we should look in)
2. Offset: specifies which byte within the block we want.
3. Tag: the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to the same location
4. Block Address: Tag + Index

Figure 1 illustrates how mapping is done depending on tag, index and offset that will be gained from a binary

representation of a memory address.

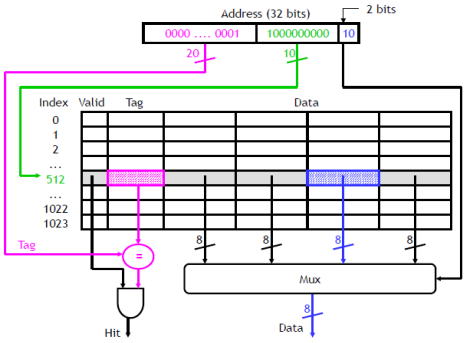

Fig. 1: Cache Example with 1,024 blocks of 4 bytes each, And 32-bit memory addresses

B. REPLACING POLOCIES [6]

Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced. The replacing policy depends upon the type of cache.

For direct mapping there is only one possible line for any particular block, and no need for replacement algorithm.

For fully associative and set-associative caches a replacement algorithm is needed because the block may go in several positions (at different indexes), and, as a result, there are different possibilities to choose a block that will be replaced. The most used policies for replacement are:

a) Random: this technique is very simple; one block is selected at random and replaced.

b) LRU (Least Recently Used): in this approach accesses to the cache are recorded; the block that will be replaced is the one that has been unused (un accessed) for the longest period of time.

c) Least Frequently Used - replace block which has had fewest hits

d) FIFO (First In First Out): Replace that block in the set that has been in the cache longest.

FIFO is easily implemented as a round-robin or circular buffer technique.

C. WRITE POLICIES5

The coherence is maintained among all the caches and global memory through four groups that are [2]:

Write-update and write-through;

Write-update and write-back;

Write-invalidate and write-through; and

Write-invalidate and write-back.

Caching data which is only to be read is easy, since the copy in the cache and memory will be equivalent. Writing in cache is more difficult because the copy in the cache and memory must be kept consistent. How ? There are two main strategies.

Write through: ensuring that main memory is always valid by do the following: all write operations are made to main memory as well as to the cache, .

Write back: A cache memory write isn't directly written into main memory unless another cache needs that cache line. The most common write-back protocol is MESI, and will be discussed next [7].

The Difference between Write Update and Write invalidate 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[6].

III. CLASSES OF CACHE COHERENCE PROTOCOL

Two cache coherence protocols hardware based are used, snoopy protocols and directory based protocols [3, 8].

a) The technique of bus snooping relies on the property that on such systems all memory accesses are performed via the central bus, i.e., the bus is used as broadcast medium. Thus, all memory accesses can be observed by the cache controllers of all processors. When the cache controller observes a write into a memory location that is currently held in the local cache, it updates the value in the cache by copying the new value from the bus into the cache. Thus, the local caches always contain the most recently written values of memory locations [9].

b) Directory Based Protocols Here the locations of all cached copies of every block of shared data must be taken into account and store those copies carefully in these cache locations, and those locations can be centralized or distributed and are called a directories. There is a directory entry that contains a number of pointers for each block of data. This number is to mention the locations of block copies [10].

IV. MESI WRITE-BACK INVALIDATION PROTOCOL

MESI (Modified-Exclusive-Shared-Invalid) stands for the state of each cache line at any time. It is based upon the data ownership model, where only one cache can have dirty (modify) data. When data is written, the cache of modifying line informs other caches of the fact; the data is not transmitted by itself. A cache line in each cache can be in one of the following states as in figure (2): [7,11, 12, 13, 14, 15]:-

Modified: one processor (owner) has data, but it is dirty; must respond to any read/write request

Exclusive: one processor has data and it is clean; no need to inform others about further changes

Shared: cached in more than one processor and memory is up-to-date

Invalid: The block has been invalidated (possibly on the request of someone else)."

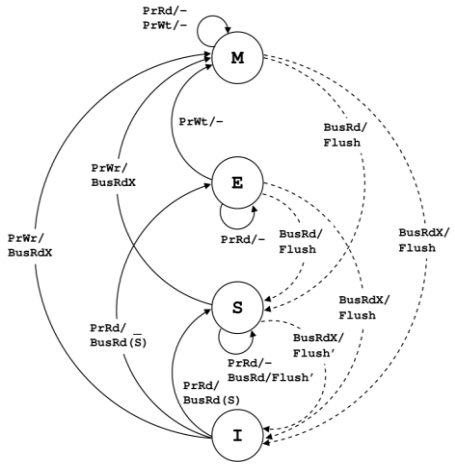

Fig.2: MESI cache coherence protocol state diagram [16]

V. METHODOLOGY, PREPROCESSING STEPS

In order to implement the proposed protocol there are several steps that must be completed which are:

Initially the following parameters must be determined before using the proposed protocol:-

size of a main memory, cache memory capacity, cache line size, associativity, replacement policy, number of words per memory access, number of cache levels, number of processors in level1, location of a directory. After definition of these parameters, the address of a main memory is converted to a binary number by using conversion function, then other function is used to gain tag, index and offset from a binary addresses to be use in a simulation of caches generally. All these steps are illustrated in figure 3.

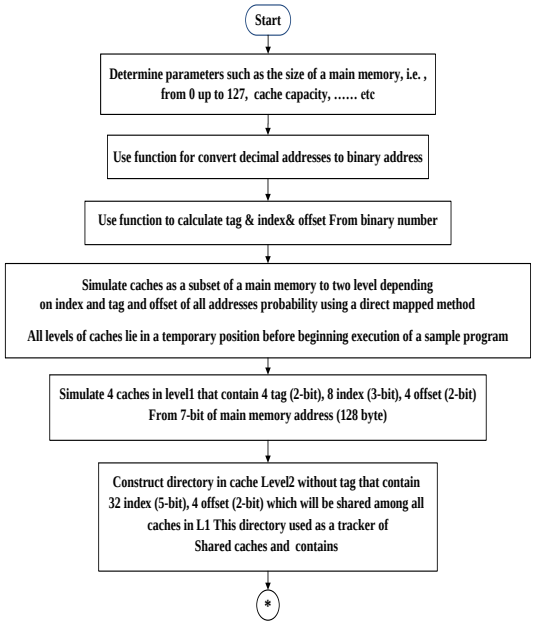

Fig. 3: preprocessing of the proposed protocol

VI. SIMULATION PROCESS

After preprocessing steps, a check functions are called to get the location of an input address and the processor job and the processor type and then tag, index and offset are obtained for used in simulation each address of a specific input program which has been clarified by Figure 4.

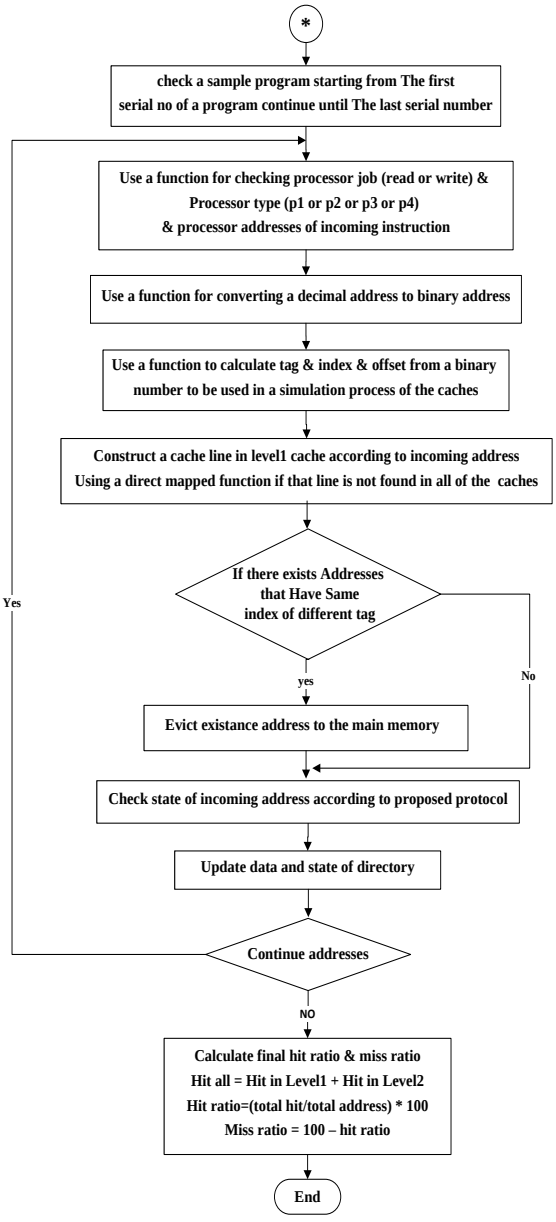

Fig. 4: Simulation process using a direct mapped method

VII. VMSI CACHE COHERENCE PROTOCOL

1) VMSI levels: The proposed project deals with a multi core processor that has two cache levels:

-The first level is private Level1 and supposes this level has four cores.

-The second Level is shared global Level2 cache that contain a directory as a central share point to the four core in cache Level1

2) Cache Organization using a Direct Mapped method

At the beginning work in this research, the four caches in level 1 and the cache in level 2 are simulated by using a direct mapped method take advantage of spatial locality as follow:-

Suppose memory address = 128[7-bit to represent main memory address].

The cache in level1, 2-bit to represent tag [tag = 4], 3-bit to represent index [cache = 8 block], 2-bit to represent offset [block = 4 byte], The shared cache in level2 5-bit to represent index [cache = 32 block], 2-bit to represent offset [block = 4 byte]. The shared cache in level2 has not a tag and all the addresses of a main memory are subset to that cache in order to reduce write back that present in MESI cache coherence protocol.

VMSI state: In this protocol each cache line has one of the four states as follow: Valid (Read at any time/Write at first time that occur exclusively). 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 writes to the directory inside cache level 2 and writes through to the main memory only in the case of replacement between different tags of the same index at level1 cache to prevent losing address. Also the read request enters this state and become exclusively that will be isolated from modified state.

Modified (Write only – previously write that occur exclusively). It is similar to modified state in MESI protocol but the different that the Processor Read - Read request from processor does not occur in this state. When the processor request read the state translate to the valid state. Also when invalidate this state or shared this state in case of a remote write or remote read, there is not needed to a write back to the main memory.

Shared marks that this cache line may be stored in other caches and is clean; it matches the directory in cache level2. The line may be changed to the Invalid state at any time.

Invalid marks that this cache line is not valid (unused), i.e. no processor has it.

3) VMSI Transition State Diagram

This protocol as shown in figure 5 has been use bus snooping protocol that appear when each state translate to other state, the abbreviation symbols of these buses are written as follow: Bus transaction:

Invalidate = Broadcast Invalidate, 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. In VMSI protocol we put the shared directory of the type full map in the cache level2 which represent as dependent central point to caches at level 1 and also put a simple directory in the memory that will be used only in the case of eviction tag from cache level1 in order to be as a tracker for eviction tag and prevent data of these tag to become lost.

VIII. THE EXPERIMENT RESULT USING DEV C++ LANGUAG

A. BINARY REPRESENTATION

Binary representation is one of a necessary preprocessing step that it is used to gain tag and index and offset of each decimal input address so as to facilitate the work of a mapping. Table (1), list 32 decimal addresses are entered in order to simulate caches at level1 and level2..

B. Cache simulation

The caches are simulated by using a direct mapped method to an input addresses that listed in Table (1) , and shown through Tables (2–4). From table (2) that shown the simulation of cache at level1: The hit and miss ratios at level1 of caches that are calculated on the previous set of addresses entered are as follow:-

Total Hit in cache level1=15 Total Address=32

Hit Ratio of cache level1 (total Hit/total Address) * 100 = 46.875%. Table (3) contain the eviction of a cache line that occur at simulation of cache in level1. From table (4)that shown the simulation of cache at level2: The hit and miss ratios at shared cache in level2 that are calculated on the previous set of addresses entered are as follow:-

Total Hit in cache level2 =20 Total Address=32, Hit Ratio of cache level2 (total Hit/total Address) * 100 = 62.5%

In simulation cache at level2 there is no need to for eviction line of the cache.

We noted from previous results that whenever we increase the size of cache memory, it is more likely the presence of addresses and thus increasing the value of hit ratio.

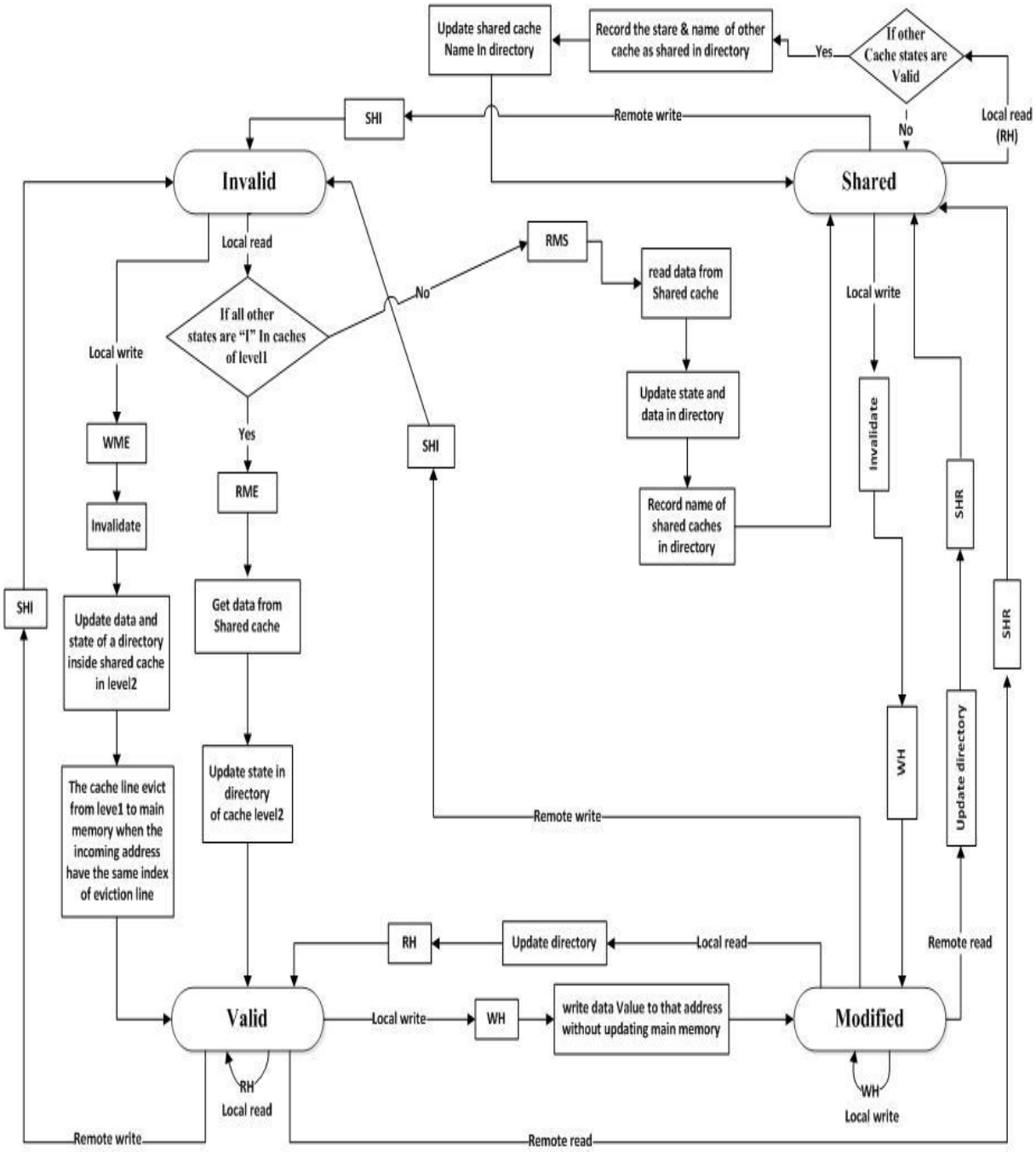

Fig. 5: VMSI cache coherence protocol state diagram.

TABLE 1
Binary representation within tag and index and offset of an input addresses.

The representation of an input addresses			cache at level1			Shared cache at level2		
Seq	Decimal Number	Binary number	Tag	Index	Offset	Tag	Index	Offset
0	44	0101100	1	3	0	0	11	0
1	0	0000000	0	0	0	0	0	0
2	1	0000001	0	0	1	0	0	1
3	4	0000100	0	1	0	0	1	0
4	5	0000101	0	1	1	0	1	1
5	8	0001000	0	2	0	0	2	0
6	22	0010110	0	5	2	0	5	2
7	12	0001100	0	3	0	0	3	0
8	2	0000010	0	0	2	0	0	2
9	16	0010000	0	4	0	0	4	0
10	39	0100111	1	1	3	0	9	3
11	18	0010010	0	4	2	0	4	2
12	20	0010100	0	5	0	0	5	0
13	23	0010111	0	5	3	0	5	3
14	33	0100001	1	0	1	0	8	1
15	25	0011001	0	6	1	0	6	1
16	26	0011010	0	6	2	0	6	2
17	27	0011011	0	6	3	0	6	3
18	30	0011110	0	7	2	0	7	2
19	31	0011111	0	7	3	0	7	3
20	0	0000000	0	0	0	0	0	0
21	1	0000001	0	0	1	0	0	1
22	2	0000010	0	0	2	0	0	2
23	45	0101101	1	3	1	0	11	1
24	4	0000100	0	1	0	0	1	0
25	5	0000101	0	1	1	0	1	1
26	43	0101011	1	2	3	0	10	3
27	7	0000111	0	1	3	0	1	3
28	8	0001000	0	2	0	0	2	0
29	9	0001001	0	2	1	0	2	1
30	33	0100001	1	0	1	0	8	1
31	11	0001011	0	2	3	0	2	3

TABLE 2
Cache Simulation at level1 using direct mapping method of an input addresses that listed in table (A.1)

Simulation of caches in level1				
	Offset			
index	**0**	**1**	**2**	**3**
0	32	33	34	35
1	4	5	6	7
2	8	9	10	11
3	44	45	46	47
4	16	17	18	19
5	20	21	22	23
6	24	25	26	27
7	28	29	30	31

TABLE 3
The eviction of a cache lines from level1 cache

The evicted cache lines from level1				
	Offset			
index	**0**	**1**	**2**	**3**
0	0	1	2	3
1	36	37	38	39
2	40	41	42	43
3	12	13	14	15

TABLE 4
Cache Simulation at level2 using direct mapping method of an input addresses that listed in table 1.

Simulation of shared caches in level2				
	Offset			
index	**0**	**1**	**2**	**3**
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19
5	20	21	22	23
6	24	25	26	27
7	28	29	30	31
8	32	33	34	35
9	36	37	38	39
10	40	41	42	43
11	44	45	46	47

TABLE 5

The results of the implementation of the proposed protocol on the sample program

Seq	Program instructions	Cache Line	Cache State	Line value	Hit	Miss	Processor Job
1	P1 writes 10 to B1	0 1 2 3	I II V	0 0 0 10	0	1	write miss
2	P2 reads B1	0 1 2 3	I II S	0 0 0 10	0	1	read miss
3	P2 writes 20 to B1	0 1 2 3	I II M	0 0 0 20	1	2	write hit
4	P2 writes 40 to B2	44 45 46 47	I V I I	0 40 0 0	1	3	write miss
5	P1 reads B1	0 1 2 3	I II S	0 0 0 20	1	4	read miss
6	P1 writes 30 to B1	0 1 2 3	I II M	0 0 0 30	2	4	write hit
7	P3 writes 77 to B1	0 1 2 3	I II V	0 0 0 77	2	5	write miss
8	P2 writes 50 to B1	0 1 2 3	I II V	0 0 0 50	2	6	write miss
9	P1 reads B1	0 1 2 3	I II S	0 0 0 50	2	7	read miss
10	P2 reads B2	44 45 46 47	I V I I	0 40 0 0	3	7	read hit
11	P3 reads B2	44 45 46 47	I S I I	0 40 0 0	3	8	read miss
12	P1 writes 70 to B2	44 45 46 47	I V I I	0 70 0 0	3	9	write miss
13	P3 writes 88 to B1	0 1 2 3	I II V	0 0 0 88	3	10	write miss
14	P2 reads B1	0 1 2 3	I II S	0 0 0 88	3	11	read miss
15	P4 reads B1	0 1 2 3	I II S	0 0 0 88	3	12	read miss
16	P4 writes 93 to B2	44 45 46 47	I V I I	0 93 0 0	3	13	write miss
17	P3 reads B2	44 45 46 47	I S I I	0 93 0 0	3	14	read miss
18	P2 reads B2	44 45 46 47	I S I I	0 93 0 0	3	15	read miss
19	P1 reads B2	44 45 46 47	I S I I	0 93 0 0	3	16	read miss
20	P4 writes 11 to B2	44 45 46 47	I M I I	0 11 0 0	4	16	write hit

C. VMSI cache coherence protocol on a sample of a program

From Table (1), addresses are converted to binary addresses by using binary conversion function and then hit and miss ratio is calculated on them. But now it will be use VMSI cache coherence Protocol that applies on a sample program example. The sample program and results of a proposed protocol on this example are listed in Table (5). In this table we suppose that the address of B1=3 and the address of B2=45 and initially all states of these addresses are Invalid and the initial value of these addresses in main memory are zero. The names of sharer cores from table(5) are:

At steps 2,5,9 the sharers are: P1 & P2and at steps 11,14 the sharers are: P2 & P3, At step 15,18 the sharers are: P2& P3 & P4and at step 17 the sharers are: P3 & P4

At step19 the sharers are: P1 & P2 & P3 & P4

The cache performance from table (5) is calculated by evaluated:

Hit = 4, miss = 16Hit ratio = (Hit / total address) * 100 = (4 / 20) * 100 = 20

Miss ratio = 100 – hit ratio = 100 – 20 = 80.

D. RESULT AND DISCUSSION

The main difference between MESI and a proposed protocol is that the VMSI method enters main memory in only one case; in the case when cache block replaced with other block of different tag that will be lie in the same cache line in order to maintain the data from losing. As a result the efficiency is increased by reducing a gap between a fast CPU and a slow main memory.

in MESI cache coherence Protocols the directory that keep track of shared data is located in main memory but in a proposed protocol two directory : one in cache level2 that will be act as the directory in memory of MESI protocol and other in memory of only an eviction action.

The data is saved in the directory of cache level 2 instead of main memory in the case of write through and write back. So the disadvantage of a write through and a write back have been reduced, the main disadvantages of write through -every write needs a main memory access as a result increasing memory bandwidth, and the disadvantages of write back - main memory isn't always proportionate with cache and reads that result in replacement may lead writes of dirty blocks to main memory. Only the write request enter modified state in a proposed protocol that the addresses is found, when the write request at a first time then it enters V state, also a read request enter V state also.

In comparisons between MESI & VMSI cache coherence protocol using a previous sample program in table3 is that by applying MESI cache coherence protocol, the number of a write back that will be occur is eight. Whereas when VMSI method was used only directory in shared cache is reached and the main memory has not been accessed. The advantage is to reduce access to the main memory, thereby increasing the efficiency of the processor.

IX CONCLUSION AND FUTURE WORK

In this paper we propose a new protocol called VMSI that it is used to achieve cache 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. VMSI protocol is an extension for MESI protocol, which minimizes a write back to main memory by placing a write state that initially write in a valid state.

In future work we tried to increase the level of caches such that we are using three levels instead of two. Also we increase the number of caches in level1 and increase associativity and also we try to modify one of the states.

REFERENCES

[1] J. SORIN DANIEL & D. HILL MARK & A. WOOD DAVID, "A PRIMER ON MEMORY CONSISTENCY AND CACHE COHERENCE", A PUBLICATION IN THE MORGAN & CLAYPOOL PUBLISHERS SERIES, 2011, PAGE 1.

[2] EL-REWINI HESHAM & ABD-EL-BARR MOSTAFA, "ADVANCED COMPUTER ARCHITECTURE AND PARALLEL PROCESSING ", PUBLISHED BY JOHN WILEY & SONS, INC., HOBOKEN, NEW JERSEY. PUBLISHED SIMULTANEOUSLY IN CANADA, 2005, PAGES 13, 92, 114, 98.

[3] RAUBER THOMAS &R¨UNGER GUDULA, "PARALLEL PROGRAMMING FOR MULTI CORE AND CLUSTER SYSTEMS ", PUBLISHED IN THE SPRINGER HEIDELBERG DORDRECHT LONDON NEW YORK, , 2007, PAGES 31, 91.

[4] A. PATTERSON DAVID& L. HENNESSY JOHN, "COMPUTER ARCHITECTURE A QUANTITATIVE APPROACH ",MORGAN KAUFMANN IS AN IMPRINT OF ELSEVIER, 2012.

[5] HWANG KAI & A. BRIGGS FAYE, "COMPUTER ARCHITECTURE AND PARALLEL PROCESSING", COPYRIGHT BY MCGRAW-HILL, INC. IN NEW YORK ST. LOUIS SAN FRANCISCO, LONDON, PARIS, 1985.

[6] STALLING WILLIAM, "COMPUTER ORGANIZATION AND ARCHITECTURE DESIGNING FOR PERFORMANCE ", PRINTED IN THE UNITED STATES OF AMERICA BY PEARSON EDUCATION, INC., UPPER SADDLE RIVER, NEW JERSEY, 2010, 07458.

[7] MOYER BRYON, "REAL WORLD MULTI CORE EMBEDDED SYSTEMS", NEWNES IS AN IMPRINT OF ELSEVIER, UNITED STATES OF AMERICA, 2013.

[8] TIWARI ANOOP, " PERFORMANCE COMPARISON OF CACHE COHERENCE PROTOCOL ON MULTI-CORE ARCHITECTURE", DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA ROURKELA, ODISHA, 769008, INDIA,2014.

[9]A. PATTERSON DAVID & L. HENNESSY JOHN, "COMPUTER ORGANIZATION AND DESIGN THE HARDWARE / SOFTWARE INTERFACE ", ELSEVIER INC., 2005.

[10]AL-HOTHALI SAMAHER, SOOMRO SAFEEULLAH, ET.AL.," SNOOPY AND DIRECTORY BASED CACHE COHERENCE PROTOCOLS: A CRITICAL ANALYSIS" ,JOURNAL OF INFORMATION & COMMUNICATION TECHNOLOGY VOL. 4, NO. 1, (SPRING 2010) 01-10.

[11]G. MAYER HERBERT," MESI PROTOCOL FOR MP CACHE COHERENCE", PSU CS STATUS ,2012

[12] SAPARON AZILAH, AND BT RAZLAN FATIN NAJIHAH, " CACHE COHERENCE PROTOCOLS IN MULTI-PROCESSOR", INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND INFORMATION SYSTEMS DUBAI (UAE), (ICSIS'2014) (ICSIS'2014),OCT 17-18, 2014

[13] HANDY JIM, "THE CACHE MEMORY BOOK – 2ND ED.", ACADEMIC PRESS SAN DIEGO NEW YORK BOSTON LONDON SYDNEY TOKYO TORONTO, 1998.

[14] JACOB BRUCE, W. NG SPENCER, T. WANG DAVID, "MEMORY SYSTEM CACHE, DRAM, DISK ",MORGAN KAUFMAN PUBLISHERS IS AN IMPRINT OF ELSEVIER, 2008.

[15] KUBIATOWICZ JOHN, "3+1 CS OF CACHING AND MANY WAYS CACHE OPTIMIZATIONS" , CS252-S07, LECTURE 15 -ELECTRICAL ENGINEERING AND COMPUTER SCIENCES UNIVERSITY OF CALIFORNIA, BERKELEY,2007.

[16] MULLINS ROBERT," CHIP MULTIPROCESSORS (ACS MPHIL)", UNIVERSITY OF CAMBRIDGE COMPUTER LABORATORY", 2011.