Proposal New Cache Coherence Protocol to Optimize CPU Time through Simulation Caches

Dr. Luma Fayeq Jalil  
Department of Computer Sciences, University of Technology /Baghdad  
Dr. Maha Abdulkareem H. Al-Rawi  
Department of Computer Sciences, University of Technology /Baghdad  
Abeer Diaa Al-Nakshabandi  
Distribution office At Ministry of Electricity/Baghdad  
Email:abeerdiaaphd@gmail.com  

Received on:11/5/2016 & Accepted on:20/10/2016

ABSTRACT

The cache coherence is the most important issue that rapidly affected the performance of a multicore processor as a result of increasing the number of cores on chip multiprocessors and the shared memory program that will be run on these processors. "Snoopy protocols" and "directory based protocols" are two types of protocols that are used to achieve coherence between caches. The main objective of these Protocols is to achieve consistency and validation of the data value in the caches of a multi core processor so that any reading of a memory address via any caches will returns the latest data written to that address.

In this paper, a new protocol has been designed to solve a problem of a cache coherence that combines the two schemes of coherency: snooping and directory depending on the states of MESI protocol. The MESI protocol is a version of the snooping cache protocol which based on four (Modified, Exclusive, Shared, Invalid) states that a block in the cache memory can have. The proposed protocol has the same states of MESI protocol but the difference is in laying the directory inside a shared cache instead of main memory to make the processor more efficient by reducing the gap between fast CPU and slow main memory.

Keywords: Cache coherence problem, snooping protocol, Directory-Based cache Protocols, MESI, Cache Simulator, Dev. C++, Multiprocessor, shared memory.

INTRODUCTION

Shared memory is the hardware part that supported by many modern computer systems and multicore 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].

Two hardware-based protocols are used to solve coherence problem that appeared in a shared memory system of a multicore processor with caches that can store multiple copies of memory blocks simultaneously which are "Snooping protocols" and "Directory-based protocols" [3,4,5].

Memory Hierarchy

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 as in figure (1). Each level speedier, more expensive and smaller, the closer it is to CPU, to feed the CPU with the required data [6].
Protocols for Cache Coherence

Two hardware-based protocols to coherence the caches in multiprocessor systems are used which include:

Snooping Protocol

To solve the problem of cache coherence by snoopy protocol, the central bus is used as a "broadcast medium" which make the transactions on bus visible to all caches [11,12]. As a result the cache controllers of all processors can observe all memory accesses (figure 2) [5,7]:

Figure 2: Snoopy Protocol [13]

These protocols are called "update-based protocols" when updated is performed directly by the cache controllers. Also "invalidation-based protocols" occur when the cache block that match memory block is invalidated and as a result main memory must update next read [3].
Directory Based Protocol

The ability of scaling in directory based schemes is better than snooping because it does not depend upon a shared bus for communication. The directory which can be central or distributed keeps state of all memory block shared between processors and then the cache controller uses point-to-point messages looking up directory instead of observing shared broadcast to get memory block state [3, 10]. (Figure 3). Although the directory-based protocols will likely have to be employed for multi core architectures of the future, there exist a drawbacks that appears in a directory which are: storage overhead, frequent indirections, and are more prone to design bugs [14,15,16].

Figure 3: Directory Based Protocols [6]

Preprocessing steps to proposal protocol

A Preprocessing steps before beginning proposal protocol are illustrated as in Figure 4, Figure 5, Figure6 respectively:
Determine the size of a main memory, for example, from 0 up to 255, which will consider the addresses of that memory.

Use function for convert decimal addresses to binary address.

Use function to calculate tag & index& offset From binary number.

Simulate caches as a subset of a main memory to a three level depending on index and tag and offset of all addresses probability using a direct mapped method. All levels of caches lie in a temporary position before beginning execution of a sample program.

Simulate 4 caches in level1 that contain 8 tag (3-bit), 4 index (2-bit), 8 offset (3-bit) From 8-bit of main memory address (256 byte).

Simulate 4 caches in level2 that contain 4 tag (2-bit), 8 index (3-bit), 8 offset (3-bit) That service as a victim cache, that is only contains data evicted from Level1 cache. If there is no available space in L1 cache.

Construct a directory at Level3 cache without tag that contain all memory addresses within 32 index (5-bit), 8 offset(3-bit) which will be shared among all caches at both level1 and level2. This directory used as a tracker of shared caches and contain last updated of data and states Within name of sharer cores.

Figure 4: preprocessing steps of a proposed protocol.
Direct Mapped Method

It is the simplest technique which maps each block of main memory into only one possible cache line [16]. The mapping is expressed using equation 1

\[ i = j \mod L \quad \text{where} \]

\[ i = \text{cache line number} \]
\[ j = \text{main memory block number} \]
\[ L = \text{number of lines in the cache} \]

Cache block = address \ mod \ number of words in cache line \quad \text{------- (2)}

Total number of memory blocks = number of main memory addresses \ div \ number of words in cache line \quad \text{------- (3)}
Figure 5: Simulation process using Direct Mapped Method
In the case of not existence the address
Transition State Diagram of proposal protocol

A cache line in each cache of proposal protocol can be in one of the following states as in Figure 7:-

**Modified**: the data owned by one processor, but it is dirty; must respond to any read/write request

**Exclusive**: the data owned by one processor and it is clean; no need to inform others about further changes

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

**Invalid**: The block has been invalidated (possibly on the request of someone else)
Figure 7: The state diagram of proposal protocol within shared directory at L3

The abbreviate symbols of these buses are 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
Measuring Cache Performance

Time CPU lapses in the implementation of the program as well as in waiting inside the memory, so CPU time is calculated as in the following equations [6, 7]:

\[
\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory stall clock cycles}) \times \text{clock cycle time} \quad - \quad (4)
\]

\[
\text{CPU time} = IC \times \left( \frac{\text{CPI}_{\text{Execution}}}{\text{Inst.}} + \frac{\text{Mem Access}}{\text{Inst.}} \times \text{Miss rate} \times \text{Miss penalty} \right) \times \text{cycle time} \quad - \quad (5)
\]

\[
\text{CPU time} = IC \times \left( \frac{\text{ALUOPS}}{\text{Inst.}} \times \text{CPI}_{\text{ALUOPS}} + \frac{\text{Mem Access}}{\text{Instruction}} \times \text{AMAT} \right) \times \text{cycle time} \quad - \quad (6)
\]

\[
\text{AMAT} = L1 \text{Hit time} \times L1 \text{Hit rate} + L1 \text{Miss penalty} \times L1 \text{miss rate} \quad - \quad (7)
\]

\[
\text{L1 Miss penalty} = \text{Access time of L2} = L2 \text{ Hit time} \times L2 \text{ Hit rate} + L2 \text{ Miss penalty} \times L2 \text{ Miss rate} \quad - \quad (8)
\]

\[
\text{L2 Miss penalty} = \text{Access time of L3} = L3 \text{ Hit time} \times L3 \text{ Hit rate} + L3 \text{ Miss penalty} \times L3 \text{ Miss rate} \quad - \quad (9)
\]

\[
\text{L3 Miss penalty} = \text{Access time of Main Memory} \quad - \quad (10)
\]

\[
\text{AMAT} = L1 \text{ Hit time} \times L1 \text{ Hit rate} + (L2 \text{ Hit time} \times L2 \text{ Hit rate} + (L3 \text{ Hit time} \times L3 \text{ Hit rate} + \text{Access time of Main Memory} \times L3 \text{ Miss rate}) \times L2 \text{ Miss rate}) \times L1 \text{ Miss rate} \quad - \quad (11)
\]

Where

- \( \text{IC} = \text{Instruction Counter} \)
- \( \text{CPI} = \text{Clock cycle Per Instruction} \)
- \( \text{AMAT} = \text{Average Memory Access Time} \)

- \( \text{Hit} \) -- the referenced information is in the cache.
- \( \text{Miss} \) -- the referenced information is not in cache, and must be read from MM
- \( \text{Hit time} \) -- is how long it takes data to be sent from the cache to the processor. This is usually fast, on the order of 1-5 clock cycles at Level1, of 10-20 clock cycles at Level2, of 30-40 clock cycles at Level3, of 50-100 clock cycles at main memory.
- \( \text{Miss penalty} \) -- is the time to copy data from main memory to the cache. This often requires dozens of clock cycles (at least).
- \( \text{Hit ratio} \) -- percentage of time the data is found in the higher cache.
- \( \text{Miss ratio} \) -- is the percentage of misses and equal (100 - hit ratio).

The Experiment Result Using DEV C++ Language

Binary Representation

Binary representation is one of a necessary preprocessing steps used to convert decimal addresses to binary address in order to obtain tag and index and offset of each binary address so as to facilitate the work of a mapping algorithm. In a proposed protocol, Main Memory has 8-bit to represent the address. So, the addresses of memory have 256 addresses as in table 1.
Table 1: Binary Representation of a Memory Addresses as Tag and Index and Offset

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Binary representation</th>
<th>Decimal number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tag</td>
<td>index</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>255</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Cache Simulation using Direct Mapped Function
The mapping of a memory addresses into L1 caches using direct mapped function are as follow:

- Number of cache block at level1 obtains from equation 3
  - Total number of memory blocks = 256 / 8 = 32 block

- Initially cache block are determined from equation 2 as follows
  - i.e. the cache block 20 contain the cache line from address 160 to address 167 and obtained by dividing these address to 8
  - i.e. the cache block 1 contain the cache line from address 8 to address 15 and obtained by dividing these address to 8
  - i.e. the cache block 18 contain the cache line from address 144 to address 151 and obtained by dividing these address to 8
  - i.e. the cache block 27 contain the cache line from address 216 to address 223 and obtained by dividing these address to 8

- Then the direct mapped function as in equation 1 is applied to these blocks to obtain index that will be used in simulation of caches at level 1 as follow:

- Cache block 0, 4, 8, 12, 16, 20, 24, 28 are mapped to index 0
- Cache block 1, 5, 9, 13, 17, 21, 25, 29 are mapped to index 1
- Cache block 2, 6, 10, 14, 18, 22, 26, 30 are mapped to index 2
- Cache block 3, 7, 11, 15, 19, 23, 27, 31 are mapped to index 3

These steps are repeated to simulate caches at level 2 and level 3 but different is in number of cache line and tag that they are specified initially

Proposal protocol Results
Before applying the proposal protocol, a binary function is used to convert addresses of input sample program to binary address and then other functions are used to obtain tag, index and offset from a binary address. Table 2 list binary addresses that are used in a sample program exist in table 3.

Table 2: Binary Representation of Input Addresses Using Proposal Protocol

<table>
<thead>
<tr>
<th>Seq</th>
<th>binary no</th>
<th>address</th>
<th>cache level1</th>
<th>cache level2</th>
<th>cache level3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>tag</td>
<td>index</td>
<td>offset</td>
</tr>
<tr>
<td>1</td>
<td>010000000</td>
<td>64</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>01000100</td>
<td>68</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>01000110</td>
<td>70</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

921
The results of applying a proposed protocol on a sample program are listed in table 7. Initially all states of input sample program are invalid and also all the values equal to zero.

### Table 3: The Results of a Proposed Protocol on a Sample Program

<table>
<thead>
<tr>
<th>Seq</th>
<th>core name</th>
<th>core request</th>
<th>data</th>
<th>address</th>
<th>state</th>
<th>value</th>
<th>cache line</th>
<th>Event</th>
<th>Bus operation</th>
<th>Sharer Cores</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P1</td>
<td>writes</td>
<td>20</td>
<td>64</td>
<td>M</td>
<td>20</td>
<td>WME</td>
<td>invalidate</td>
<td>P1 &amp; P2 at L1</td>
<td>I M</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>P3</td>
<td>writes</td>
<td>7</td>
<td>68</td>
<td>M</td>
<td>7</td>
<td>WME</td>
<td>invalidate</td>
<td>P1 &amp; P2 at L1</td>
<td>I M</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>P2</td>
<td>reads</td>
<td>64</td>
<td>S</td>
<td>20</td>
<td>RMS</td>
<td>update directory</td>
<td>P1 &amp; P2 at L1</td>
<td>I S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>P2</td>
<td>writes</td>
<td>77</td>
<td>70</td>
<td>M</td>
<td>77</td>
<td>WME</td>
<td>invalidate</td>
<td>I M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>P4</td>
<td>reads</td>
<td>68</td>
<td>S</td>
<td>7</td>
<td>RMS</td>
<td>update directory</td>
<td>P3 &amp; P4 at L1</td>
<td>I S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>P1</td>
<td>reads</td>
<td>70</td>
<td>S</td>
<td>77</td>
<td>RMS</td>
<td>update directory</td>
<td>P1 &amp; P2 at L1</td>
<td>I S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>P2</td>
<td>writes</td>
<td>45</td>
<td>68</td>
<td>S</td>
<td>45</td>
<td>WME</td>
<td>invalidate</td>
<td>I M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>P2</td>
<td>reads</td>
<td>70</td>
<td>S</td>
<td>77</td>
<td>RH</td>
<td></td>
<td></td>
<td>P1 &amp; P2 at L1</td>
<td>I S</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>P4</td>
<td>writes</td>
<td>99</td>
<td>64</td>
<td>M</td>
<td>99</td>
<td>WME</td>
<td>Invalidate</td>
<td>I M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>P1</td>
<td>writes</td>
<td>35</td>
<td>64</td>
<td>M</td>
<td>35</td>
<td>WME</td>
<td>Invalidate &amp; Update directory</td>
<td>I M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>P3</td>
<td>reads</td>
<td>64</td>
<td>S</td>
<td>35</td>
<td>RMS</td>
<td>update directory</td>
<td>P1 &amp; P3 at L1</td>
<td>I S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>P1</td>
<td>writes</td>
<td>80</td>
<td>70</td>
<td>M</td>
<td>80</td>
<td>WH</td>
<td>invalidate</td>
<td>S M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>P3</td>
<td>reads</td>
<td>68</td>
<td>S</td>
<td>45</td>
<td>RMS</td>
<td>update directory</td>
<td>P2 &amp; P3 at L1</td>
<td>I S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>P2</td>
<td>reads</td>
<td>70</td>
<td>S</td>
<td>80</td>
<td>RMS</td>
<td>update directory</td>
<td>P1 &amp; P2 at L1</td>
<td>I S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>P1</td>
<td>writes</td>
<td>54</td>
<td>70</td>
<td>M</td>
<td>54</td>
<td>WH</td>
<td>invalidate</td>
<td>S M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>P2</td>
<td>reads</td>
<td>68</td>
<td>S</td>
<td>45</td>
<td>RH</td>
<td></td>
<td></td>
<td>P2 &amp; P3 at L1</td>
<td>S S</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>P4</td>
<td>reads</td>
<td>70</td>
<td>S</td>
<td>54</td>
<td>RMS</td>
<td>update directory</td>
<td>P1 &amp; P4 at L1</td>
<td>I S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>P1</td>
<td>writes</td>
<td>33</td>
<td>70</td>
<td>M</td>
<td>33</td>
<td>WME</td>
<td>invalidate</td>
<td>I M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>P3</td>
<td>reads</td>
<td>70</td>
<td>S</td>
<td>33</td>
<td>RMS</td>
<td>update directory</td>
<td>P1 &amp; P3 at L1</td>
<td>I S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>P3</td>
<td>reads</td>
<td>64</td>
<td>S</td>
<td>35</td>
<td>RH</td>
<td></td>
<td></td>
<td>P1 &amp; P3 at L1</td>
<td>S S</td>
<td></td>
</tr>
</tbody>
</table>

### Cache Performance Result

Cache performance can be measured by counting a program execution cycles that include cache Hit time and a memory stall cycles which result from cache misses. Suppose that after depending on the clock speed of the central processor, it takes: 7 ns to access data in L1 cache, 17 ns to access data in L2 cache, 30 ns to access data in L3 cache, 80 ns to access data in Main Memory.

Calculate hit and miss ratio according to both addresses and also to proposed protocol:

- In using proposed protocol:
  
  The proposed protocol in figure 7 are applied on a sample program, then hit and miss ratio results are appears at level1 caches only because the input addresses use only 3 address and all these addresses appear at the same cache line. Hit and miss ratio calculate from table 3 as follow:

  Hit ratio at L1 = (no. of hit in level1/ total no. of address) * 100 = (5/20) * 100 = 25%

  Miss ratio in L1 = 100 - Hit ratio = 100 - 25 = 75%

  Hit and Miss ratio in level2 and Level3 are not exist

922
2- in using addresses
First address equal to miss and all other addresses are equal to hit because the same line contains all addresses and when this line is fetch then the first address take from main memory and the rest addresses appears. So, Hit and miss ratio is calculated as follow:
Hit ratio at L1 = (19/20) * 100 = 95 %
Miss ratio at L1 = 100 – 95 = 5 %
Hit ratio = (hit ratio in protocol + hit ratio in address) / 2 = (25 + 95) / 2 = 60 %
Miss ratio = (miss ratio in protocol + miss ratio in address) / 2 = (75 + 5) / 2 = 40 %
Finally Average Memory Access Time (AMAT) is applied as in equation11

The Comparison between MESI and proposed Protocol
- In MESI cache coherence Protocols the directory that keep track of shared data is located in main memory but in a proposed protocol the directory is located in a shared cache level3. As a result the efficiency is increased by reducing a gap between a fast CPU and a slow main memory.
- The write through and write back has been translated from main memory into level3 shared cache, so the disadvantages of write through in uses more memory bandwidth is reduced, and the disadvantages of write back of making the main memory inconsistent with cache also reduced. The different between MESI and proposed protocol in using sample program that are shown in table 3 are as follow:
Steps 3, 5, 6, 11, 13, 14, 17, 19 are write back addresses of a previous modified state to main memory as a result of a remote read. And step 10 is write back modified address line to main memory as a result of remote write. But in using proposed protocol these steps return to update the directory at level3 instead of access to main memory.

Conclusion and Future Works
A new idea is proposed in this research to achieve cache coherency. The reason behind the development of coherency protocol is that this protocol effectively affects the efficiency of the processor in multi-core computer systems.
In future work the number of caches at level1 and level2 are tried to be increased and also modifying in one of the states and also increase associativity in using mapping algorithm. All these idea are proposed in order to reduce access to main memory.

REFERENCES


