Studying of Cache Coherence on Shared Memory Multi-processor System

CSCI 6461– Section 11

Name: Yao Xiao

GWid: G36446733

Email: xiaoyao@gwu.edu

3/15/2015

Contents

Abstract

Multi-processor is a trend of development and cache is used in processors in order to raise the access speed. Therefore, cache coherence becomes an important issue for shared memory multi-processor system.

The paper analyzed the cache coherence issue and the characteristics of three typical cache coherence protocols, compared their advantages, limitations and the applicable scopes. The paper also analyzed the challenges for future design of cache coherence protocol.

Keywords: Cache coherence; Multi-processor system.

1. Introduction

With the advance of the system architecture and the rapid development of production technology, the gap between the access speed of the memory and computing speed of processor become more and more significant. So access speed becomes the bottleneck of improving processor performance.

In shared memory multiprocessor, improves the efficiency of data access. Due to the fact that each processor can only get data from its cache, it's possible that different processors get different data from same address. So, it is necessary to apply a protection mechanism for shared memory multiprocessor. And make sure that each cache can always load the latest data from shared memory. With the increasing complexity of multithreading and data exchange mode, it becomes more and more important to provide an effective cache coherence model. And it can influence the efficiency and accuracy of multi-thread program to a great extent.

This paper is going to analyze some typical types of cache coherence protocol and the challenges that designers may face in the future design of cache coherence protocols.

2. Analysis of the cache coherence problem

In single processor, cache coherence means each copy of data in cache should keep consistent with its upper storage. If a line of data in the cache is modified, copies in all the upper storages should also be modified.

In multi-processor, each processor has its own private cache, and all of them share a same memory. Therefore, cache coherence for multi-processor requires not

only the consistent between cache and memory, but also the consistent among the cache.

Coherence problem between cache and main memory can be well solved by using write through or write back strategy. But the cache coherence problem in multi-processor is much more complicated.

In multi-processor, some cache may have a copy of same data. When one processor modified the data in its cache, if the same copy of data on other cache cannot be modified in time, other processors may get outdated data from its cache. Or if a processor is going to read a data from memory which haven't yet finish written by another processor, then it may also get wrong data, and the errors may occur on this program. That is the cache coherence problem.

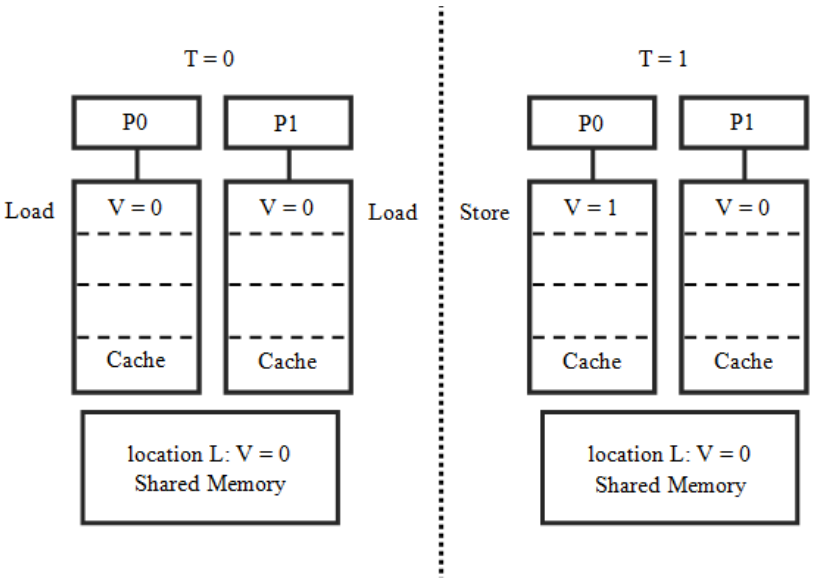

Figure 1: Cache coherence problem in multi-processor system [1]

Assume that there are two processors: P1 and P2 and they will do some load or store operations on a same block of shared memory. On some time T0, both P1 and P2 want to load the data of M1. And both of them will get the same right data from the

memory. So it's all right if there only have read operations. But once they need to write something to the memory, the problem of coherence will appear.

As shown in the figure above, at the time T = 1, P1 want to write data to M2. But before the data reach the memory, processor P2 is about to read this block of data from memory. So at that time, caches of P1 and P2 have different storage views. That means P2 may get a data different from P1, although they correspond to the same memory block. Or if P2 have this block of data in its cache, it may use the old data in cache, without noticing the data have changed. So the key point of the problem is write operation.

What we need to do is avoiding the usage of invalid data. To meet this requirement, we can find a naive solution. The key point in this problem is that there are multiple caches, rather than multiple processors. So the problem can be solved by sharing the cache, which means there has only one cache and all processors need to share it. In a particular cycle, only one of these processors can luckily get the cache to do some memory operation. That solution seems solved the problem of coherence between caches, but it's too slow. Because the processors will spend so much time to wait for their turn to use the cache.

To solve this problem, we should first find out that when will these issues occur.

Usually, there are three causes of incoherence: sharing of the writable data, process migration and I/O transmission.

(1) Incoherence of shared data

For example, there are two or more processors P1, P2… Before the data updated,

two processors have a shared data X. When the processor P1 modifies data X to Y, the situation, according to the two different strategies of cache write operation, will be changed as follows.

If the write through strategy is used, when processor write some data to its cache, it will change the corresponding data in memory as well. By using the write through strategy, data in P1 and memory are changed to Y, but the copy in P2 is remains X. So at this time, P1 and P2 have the cache values of X and Y respectively, which obviously not consistent.

If the write back strategy is used, when data in the cache is modified, it will not update the data for memory immediately. Instead, it will update the lower level when a block falls out of the cache. If we apply write back strategy, when P1 write its cache, data in the memory and P2 are not being changed. So they are also not consistent in write back strategy.

(2) Incoherence of process migration

If the write back strategy is used, cache coherence problem will appear when the processor P1, which includes the shared data X, migrates to processor P2. And if the write through strategy is used, the problem will also appear when P2 migrates to P1 according to this example. [2]

(3) Incoherence of I/O transmission

I/O operations which skip the cache can also cause coherence problem. When the I/O processor write a new data Y into memory, and bypass the cache which uses write through strategy, the coherence problem will come out between cache and memory. If

data are retrieved without passing the cache, the problem will appear in write-back cache. For example, the DMA controller operate (read or write) the main memory directly, if some caches have the copy of this part of memory, the coherence problem will be generated between cache and memory.

In fact, it seems not possible to let all these caches always correspondence with each other. What we need is a protocol that all the processors will obey, and use that to get the result of coherence.

3. Analysis of typical coherence models

There are many different coherence models. This paper will analyze three widely used models. They are snooping protocol, directory-based protocol and token coherence protocol.

3.1. Snooping protocol

Snooping protocol is a commonly used method to maintain cache coherence. The basic idea of "snooping" is that all memory transactions occur in a shared bus which is visible to all the processors. The caches are independent with each other, but the memory is a shared resource. All memory access need arbitrate, which means in any particular instruction cycle, only one cache can read or write the memory. [3]

The idea of "snooping protocol" is that the cache not only interacts with the bus when they need to do some memory transaction, but continuously snoops on bus traffic and keeps track of what other caches are doing. When a cache wants to read from or write to memory for its processor or core, all the other caches will notice that and use this message to keep themselves corresponding. So once a processor does

write operation to the memory, other processors will know that and mark invalid for the copy of data in their caches.

In write-through Caches, the things are not complicated. Because once write operation take place, it will be write to the memory directly and “published” through the bus. But if write-back strategy is used, this doesn’t work. That is because data may be written to the memory at a long time after the data written to the cache. And during this period of time, other caches may also write to the same location, which leads to conflict. Therefore, in write-back method, it is not enough to just publish the message of write memory operation. Instead, it should tell other processors before start changing its cache. And that is the basic idea of the solution which commonly called MESI protocol.

3.1.1. MESI protocol

MESI protocol is a classic snooping protocol. Many companies use this protocol or some its derived protocols. “MESI” are the initials of the four different states in a cache line: Modified, Exclusive, Shared and Invalid. Each cache in the multi-processor system can be in one of these four states. The meanings of these four states are as follows.

Modified cache line is dirty line. It has been locally modified by its processor. If some line of cache becomes modified state, only its owner can use this data. All the same copy of data in other caches must be in the in invalid state and can’t be used anymore. Besides, if a modified cache line becomes invalid, it should be written back to the memory, which just same as the dirty state in write-back cache.

Exclusive cache lines have some similarities with shared lines. It also has the copy of the data of memory. The different is that once a cache gets a cache line with exclusive state, other processors cannot hold it at the same time. That means if this copy of data already held by some other caches, these lines have to be in the invalid state.

Shared lines are the cache lines that keep same copy of data with main memory. Lines under this state can only be read by the processor, but cannot be written to. Different caches can have the contents of same memory location in their shared lines.

Invalid cache lines are the data that already outdated or not present in the cache. Once a cache line is marked as invalid, this line of data is regarded as not exist in the cache.

For single processor, there are similar concepts with modified, shared and invalid states in write back cache. But exclusive is a new state for multi-processor. Exclusive state is used to help the processor do write operation. To ensure the coherence of the caches, if one processor wants to write some data, it should inform other caches before changing the data. MESI protocol requires a modified or exclusive state for processor to do a write operation, because only in these two states can processor get the exclusive access. If a processor does not exclusively owned a cache line, when it want to write, it should send a request to the bus and let all the other processors invalid their copies of this line. After that the processor can begin its writing operation. And the processor knows that the only valid copy of that cache line is in its cache, so no conflict can occur from this operation.

If there has another processor want s to read this cache line, we can know that immediately, because we are snooping at the bus. And then the exclusive or modified cache must return to the shared state. If it was in modified state, it also needs to write the data to main memory.

The MESI protocol is a state machine which can not only respond to the requests from the local processor, but also send the messages to the bus. Figure 2 below is the state transition diagram of MESI protocol.

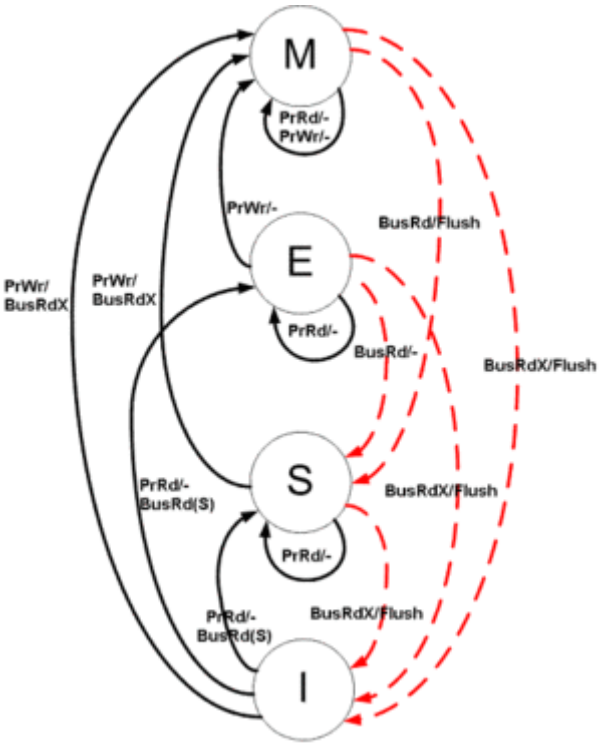

Figure 2: State transition diagram for MESI protocol
Source: http://en.wikipedia.org/wiki/MESI_protocol

In single processor system, if the processor wants to read a cache line, it does not need to consider about the memory. In a multi-processor system, however, read operation can involve other processors and even memory access. Write a cache line need more steps. Before writing data, the processor first needs the exclusive ownership of the cache line and the copy of its data.

After writing back all the modified cache lines, all the data copies of each cache will have same values with the memory. In addition, if the data of some memory location has an exclusive copy in a cache, then it can't be present in any other processor's cache, or at least in invalid state.

For any given pair of caches, the permitted states of a given cache line are shown in the table below:

Table 1: Permitted states for MESI protocol

	M	E	S	I
M	✗	✗	✗	✓
E	✗	✗	✗	✓
S	✗	✗	✓	✓
I	✓	✓	✓	✓

Source: http://en.wikipedia.org/w/index.php?title=MESI_protocol&redirect=no

In fact, MESI protocol just like the write back strategy with a additional exclusive state. This protocol can keep the coherence of caches in multi-processor system effectively, and does not weaken the original memory model.

3.1.2. MOESI protocol

MOESI protocol is a variation of MESI protocol. It has all the four states in MESI protocol and a new 'O' state which means Owned. The shared state is slightly changed in this protocol.

Owned state means the cache line has the most recent copy of the data. This copy of data is also in some other processors' caches. It can be used by other processors, but only the processor with owned state is responsible to update the correct value in memory before it gets changed. If a copy of data is present in several caches, only one

of these caches can be in the owned state, and others are in shared state. [4]

In MOESI protocol, cache line in Shared state may not have same data with memory. If none of the other caches who have this copy of memory is in the owned state, then this cache has the same data with memory. If there has an "owned" cache, the data of this cache is not coherence with the memory.

The state transition diagram of MOESI protocol is shown in the figure below:

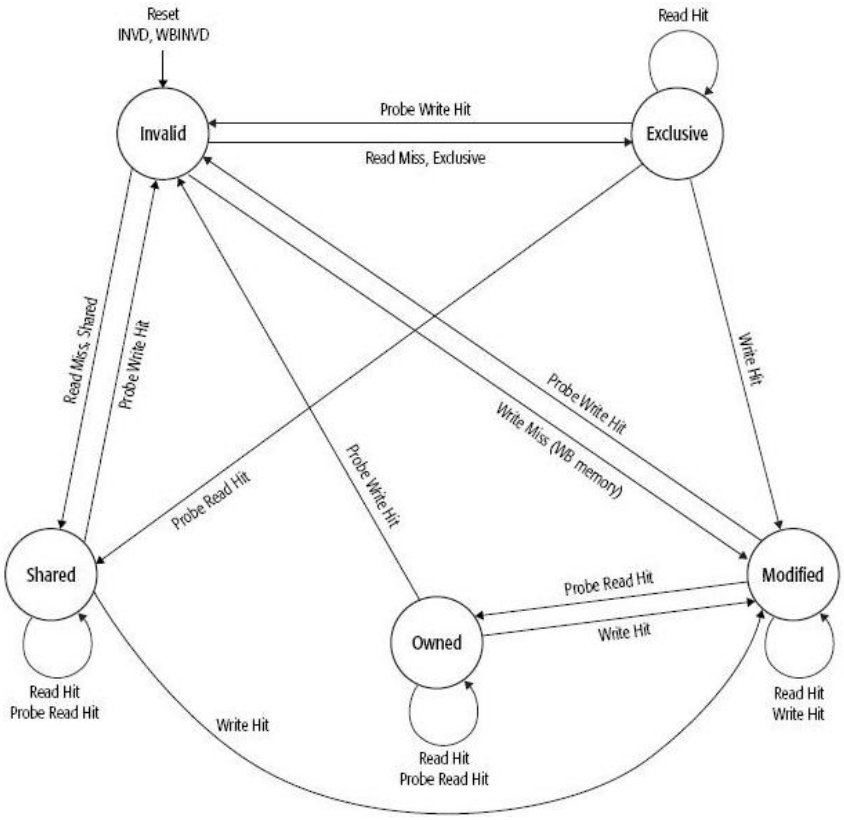

Figure 3: State transition diagram for MOESI protocol

Source: http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_506_Spring_2013/8a_an

The primary difference between MOESI and MESI protocol is that in MOESI protocol, shared data in caches can be different with the main memory. In a multi-processor system, if a processor is seeking the data access, it will firstly try to find the latest data in its cache. If read miss, it will try to get the data from other caches. If the data is not in any cache, it will finally fetch it from main memory. This

kind of share strategy also called dirty sharing.

For any given pair of caches, the permitted states of a given cache line is as follows:

Table 2: Permitted states for MOESI protocol

	M	O	E	S	I
M	✗	✗	✗	✗	✓
O	✗	✗	✗	✓	✓
E	✗	✗	✗	✗	✓
S	✗	✓	✗	✓	✓
I	✓	✓	✓	✓	✓

Source: http://en.wikipedia.org/wiki/MOESI_protocol

In some situation, MOESI protocol can remarkably improve the utilization of Cache. Owned state makes the processor does not need to write the memory directly, but send them to other caches. And other caches can use the most recent data without waiting for the data written to the memory which requires a long time. This protocol is widely used by processor companies like AMD and RMI.

3.1.3. MESIF protocol

MESIF is another variation of MESI protocol. It was developed by Intel for cache coherent non-uniform memory architectures (ccNUMA). This protocol is used more in distributed multi-processor system based on network, and therefore, more complex than MOESI protocol.

In Non-uniform memory access system, shared storages are physically distributed. So the access storage time a processor need is different according to the physical location. Obviously, the speed of local storage access is much faster than global

shared storage or remote access.

The name of this protocol also denotes the initials of cache line states. The M, E S and I state are same as MESI protocol. F is a new state in this protocol which means "Forward". F state is a specialized form of the Shared state. In ccNUMA system, there may have several caches that have a same copy of data. Only one of these caches can in forward state, and others should in shared state. Cache line with forward state must have same data with memory. [5]

The state transition diagram of MESIF protocol is shown in the figure below:

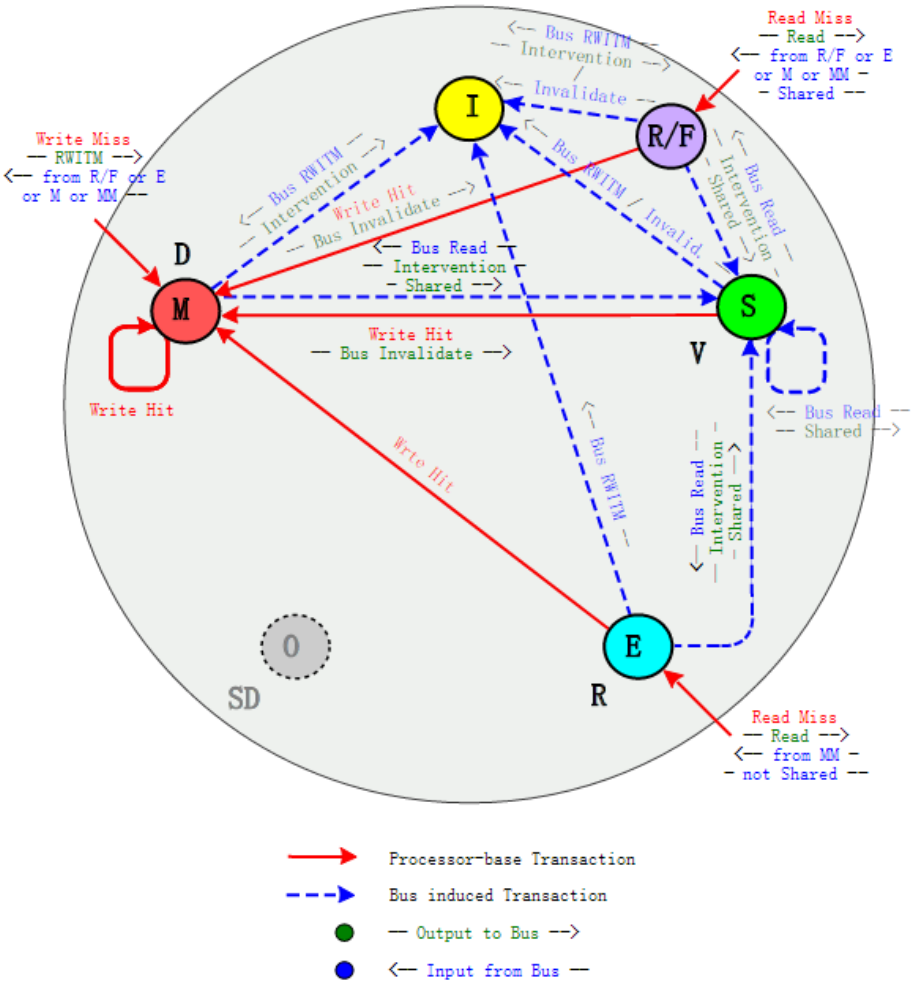

Figure 4: State transition diagram for MESIF protocol

Source: http://commons.wikimedia.org/wiki/Image:MERSI-MESIF_Protocol_-_State_Transaction_Diagram.svg?uselang=it

If a copy of data is held by an F state cache line, then only this forward cache line can response the request of this copy data. Cache lines in shared state can't response the request in that time.

If the data of F state cache line is copied by other processor, the forward state will transfer to the new copy. The original one will change to shared state. If the data of an F state cache line is changed, the system should firstly set all the other copies of this data to invalid state, and set this copy to modified state.

Sometimes a cache may discard a line with F state, so it is possible that all the copies of some data are in S state, but no cache has a copy in the F state. In this case, it can't get the improvement brought by F state. To minimize the chance that cache lines with forward state be discarded, the most recent requestor of a line will be set to forward state.

The table below summarizes the MESIF cache states:

Table 3: Cache states for MESIF protocol

State	Clean/ Dirty	May Write?	May Forward?	May Transition To?
M – Modified	Dirty	Yes	Yes	-
E – Exclusive	Clean	Yes	Yes	MSIF
S – Shared	Clean	No	No	I
I – Invalid	-	No	No	-
F – Forward	Clean	No	Yes	SI

Source: http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_506_Spring_2013/8a_an#MESIF_Protocol

The idea of this design is to reduce the data congestion. When we applying the MESI protocol in a multi-processor system, if a cache line request received by multiple caches with shared state, this request will be serviced inefficiently. That is

because all the sharing caches will respond for the request and cause data congestion. So F state in MESIF protocol is used to solve this issue.

3.2. Directory-based protocol

Directory-based protocol is another typical cache coherence protocol for shared memory multi-processor. Different from snooping bus protocol, directory-based protocol is used more in large-scale systems. [6]

Directory-based protocol has three states for data: Shared, Invalid and Exclusive. These kinds of states have similar meaning with those in snooping protocol, but the objects of the states are data blocks in memory rather than cache lines.

1) Shared state means at least one processor has the data cached, and all the copies have same contents with shared memory.
2) Invalid state means the data is not cached by any processor, and therefore the memory is up-to-date for this data.
3) Exclusive state means only one processor has the ownership of this data, and it’s not consistent with the shared memory.

In directory-based system, all of the shared data are placed in a directory which is used to maintain the coherence among caches. The directory will save and trace the states of all the caches, and it can tell the cache how to respond the requests.

The state transition diagram of Directory-based protocol is shown in the figure below:

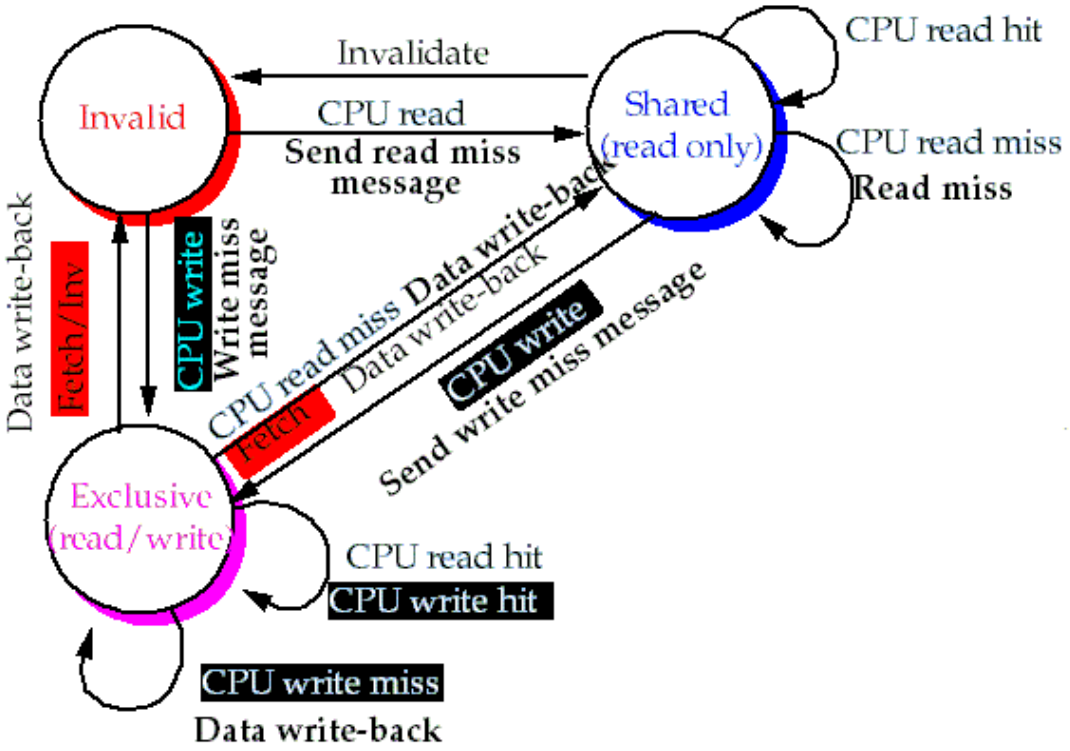

Figure 5: State transition diagram for Directory-based protocol

Source: http://www.ece.unm.edu/~jimp/611/slides/chap8_3.html

If a processor read miss at some address, it should first check the directory to learn the state of the data. And then the directory will forward the request to the data's owner cache (if in S or E state) or the shared memory (if in I state). After that, the requester will become a member of the processors who shared this data.

If the processor write miss at some address, which means it wants to write a non-exclusive data, it should require for the exclusive ownership of this data.

In multi-processor system, a processor may want to write a data that haven't finish writing by other caches. To maintain the order of the requests, the directory will sort the requesters. A processor can operate the data only when it gets the confirm message from prior owner cache.

Other types of messages are shown in the table below:

Table 4: Directory Protocol Messages

Directory Protocol Messages			
Message type	*Source*	*Destination*	*Msg Content*
Read miss	Local cache	Home directory	P, A
– *Processor P reads data at address A; make P a read sharer and arrange to send data back*			
Write miss	Local cache	Home directory	P, A
– *Processor P writes data at address A; make P the exclusive owner and arrange to send data back*			
Invalidate	Home directory	Remote caches	A
– *Invalidate a shared copy at address A.*			
Fetch	Home directory	Remote cache	A
– *Fetch the block at address A and send it to its home directory*			
Fetch/Invalidate	Home directory	Remote cache	A
– *Fetch the block at address A and send it to its home directory; invalidate the block in the cache*			
Data value reply	Home directory	Local cache	Data
– *Return a data value from the home memory (read miss response)*			
Data write-back	Remote cache	Home directory	A, Data
– *Write-back a data value for address A (invalidate response)*			

Source: https://courses.cs.washington.edu/courses/cse471/00au/Lectures/luke_directories.pdf

Directory-based protocol can avoid the message broadcast, and therefore reduce the traffic that generated by coherence protocol. So this protocol performs better than snooping bus protocol on large-scale systems.

3.3. Token coherence protocol

Token coherence protocol is a more recent design which has a lot of differences with snooping and directory-based protocol. It separates the correctness and the performance of the cache coherence. It builds a framework to guarantee the correctness, and a variety of performance strategies can be used based on this framework. The performance policies share some similarities with snooping and directory-based protocol, and used to increase the efficiency of the protocol. The correctness framework is the key part of this protocol, which used to ensure the coherence of caches. [7]

The figure below shows the state transitions of the correctness framework in token protocol:

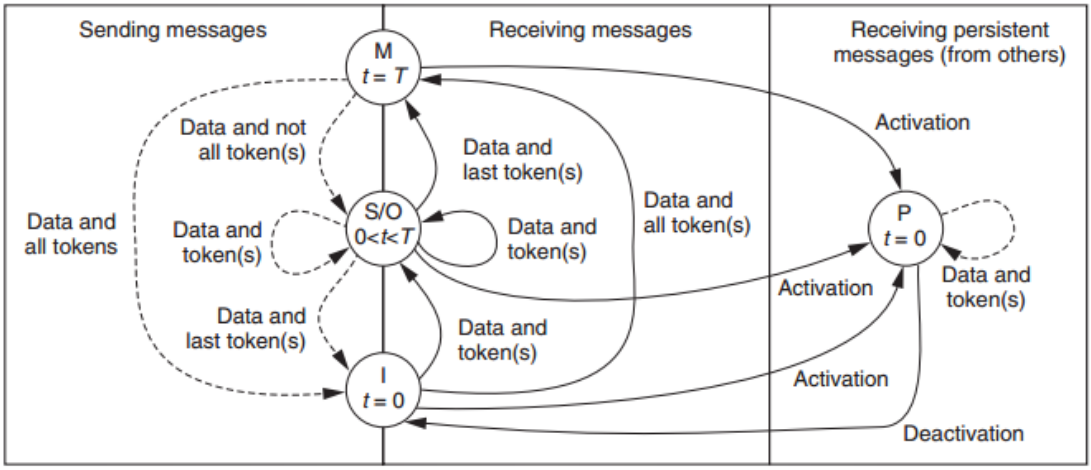

Figure 5: Correctness substrate state transitions for a processor

Source: http://repository.upenn.edu/cgi/viewcontent.cgi?article=1282&context=cis_papers

The correctness framework set a certain number of tokens for each block of data. If a processor wants to read this data, it should have a token of this data. If the processor wants to write the data, it should have all of the tokens of this block of data. This counting mechanism guarantees that only one write operation or some read operations can take place on a particular block of data at a time. And therefore, caches can keep coherence. Moreover, the correctness framework can prevent the requests from waiting too much time. If a request has failed for several times, the system will invoke a persistent request, which always succeeds.

The performance policy is focuses on higher efficiency. It uses “transient requests” to require for the tokens. Token protocol can apply different performance policies, like snooping or directory and so on. Different performance policies may have different delay, bandwidth cost, storage cost and power consume. Based on the correctness framework, designers can choose an appropriate policy according to the needs of the

specific application. That shows the good expandability of token coherence protocol.

4. Comparison of existing cache coherence protocols

Different protocols have different features, and fit for different applications. This part will analyze the advantages and limitations of three cache coherence protocols mentioned in this paper.

Snooping bus protocol maintain the cache coherence by broadcast the coherence messages. This kind of cache-to-cache communication has very little delay. Other processors can get the message in a short time and respond for it. And this protocol needs a little additional storage space. It works well in small or medium scale multi-processor system. However, will the scale of system goes up, the bus traffic generated by this protocol growth rapidly. Increasing bus competition will lead to higher delay and power consumption. Therefore, snooping bus protocol is difficult to apply to large-scale systems.

Directory-based protocol avoids the message broadcast, and reduces the bus occupation and power consumption. But this protocol uses a directory to save the states of data, which means considerable additional storage consumption. Moreover, in this protocol all the messages between caches should forward by directory. This intermediate layer will increase the delay of requests. So directory-based may not perform as well as snooping protocol in small-scale systems. But in this the costs of the protocol are not increasing significantly with the growth of the system scale. Therefore, directory-based protocol works better than snooping protocol in large-scale multi-processor systems.

Token coherence protocol avoids the dependence of sequential interconnection structure. Delay and power costs may be varied according to its performance policy. But it also needs additional storage space to save tokens. And its complicated structure may increase the complexity of the system. Token coherence protocol is more appropriate for the applications which have special requirements for the properties of the protocol.

In short, snooping bus protocol has lower delay and storage cost, which fits for small or medium scale systems; directory-based protocol has lower bus occupation and power consumption, which fits for large-scale systems; token coherence protocol has better expandability, and fits for the applications which have special requirements.

5. Challenges for future design of cache coherence protocol

Designers always expect the protocol has faster processing speed for cache invalidation, smaller on-chip interconnect bandwidth, smaller area and power consumption in hardware implementation and good expandability. The research for cache coherence mechanism in multi-processor is a trend of development and also faces great challenges.

5.1. Collaborative design of coherence protocol and interconnection structure

The design of cache coherence protocol is strongly related to the on-chip interconnection structure. Firstly, the bandwidth that the interconnection structure can provide will affect the expandability of cache coherence protocol. Secondly, cache coherence protocol may need a sequential interconnection structure to ensure its accuracy. Therefore, an interconnection structure with low delay, high bandwidth and

good expandability is the base of an efficient cache coherence protocol.

Designers want to use sequential interconnection structures like CPU bus to simplify the maintenance of cache coherence mechanism, and also want to use non-sequential structure to provide the better expandability and lower delay communication framework. In fact, in most cases it's not possible to provide both sequential interconnection structure and large bandwidth. For example, CPU bus can make sure that all the consistency messages can be generated and passed in same order. But in non-sequential structure, like Network-on-Chip structure, the design of snooping protocol becomes much more difficult.

Bus and Crossbar structure are good choices for medium-scale multi-processor system, which are used by IBM Cell and Sun Niagara [8]. But their expandability is not good, and therefore not appropriate for large-scale systems. Nevertheless, Network-on-Chip structures represented by Mesh topology structure can provide better expendability. Therefore, the design of cache coherence protocol should associate with the practical interconnection structure.

On the one hand, the different features among interconnection structure, like Bus, Ring, Crossbar and Mesh, makes designers tent to choose different cache coherence protocol. On the other hand, in shared memory multi-processor system, frequently interaction between processors and threads makes on-chip communication more complex. The large amount of message transmission and data exchange requires an interconnection structure which can provide large bandwidth and low delay. The collaborative design of cache coherence protocol and on-chip interconnection

structure will be an important research direction in the future.

With the development of processors, hardware issues, like power consumption, become more and more important. Multi-processor do fine-grained data interaction frequently, so a lot of requests, data response and confirm information will be generated, which greatly increases the traffic in interconnection structure and brings heavy burden of power consumption.

There are some mechanisms that designed to lower the power consumption for Cache coherence.

5.2. Optimization strategies for reduce the power consumption

With the development of processors, hardware issues, like power consumption, become more and more important. Multi-processor do fine-grained data interaction frequently, so a lot of requests, data response and confirm information will be generated, which greatly increases the traffic in interconnection structure and brings heavy burden of power consumption. [9]

There are some mechanisms that designed to lower the power consumption for Cache coherence.

5.2.1. Multicast mechanism

Large amount of messages broadcast can significantly increasing the traffic for on-chip interconnection structure, which means large dynamic power consumption. And that is an important reason for broadcast based protocols have not been widely adopted in large-scale system. In cache coherence protocol, messages often just need to send to a part of storages, but not all the storages. Therefore, if we can apply

multicast mechanism to let the message only be sent to the necessary storages, the power consumption of message broadcast will goes down. And hence, seeking an appropriate multicast mechanism for particular interconnection structure is a good way to reduce the power consumption of message broadcast. [10]

5.2.2. Prediction mechanism

Least Frequently Used (LFU) cache replacement strategy is based on the fact that if a cache line used frequently now, it is likely to be used frequently in future nearby. The design of prediction mechanism for cache coherence protocol is just like this idea. If once the data copy in processor A become invalid, and processor B provide the copy of data, then it is likely that still processor B provide the data copy when it become invalid in processor A again. That shows the high locality of data invalidation in multi-processor. Based on this locality, we can use a predictor to predict which cache can provide this copy of data. And then send the request message only to this cache. Prediction mechanism can, to some extent, avoid the power consumption of message broadcast.

5.2.3. Reduce the remote data transmission

In snooping protocol, each cache is snooping on the coherence message from bus. And therefore, the entire on-chip interconnect structure and all the communication interfaces are almost keeping active. According to the physical location of storages, if we let the nearest storage to provide the data, remote data transmission can be reduced. In this case, only the link between requester and data provider is active. And therefore, the power consumption of the coherence protocol can be reduced.

5.2.4. Trade-off between storage cost and power cost

In most cases, it's not possible to reduce both communication traffic and storage cost. Designers need to make a trade-off between them. For example, directory-based protocol can avoid the message broadcast which used in snooping bus protocol. But it requires some additional storage space to save the states of directory. So this protocol reduced its power cost by consuming more storage space.

5.3. Verification of cache coherence protocol

Cache coherence protocol is an important factor to ensure the correct behavior for shared memory multi-processor system. Designers often need to spend a lot of time verifying the protocol. The verification of cache coherence protocol includes data consistency, protocol integrity and the problem of deadlock. So it's necessary to do verification in the early stage of design, and that is very helpful for finding the defects of the coherence protocol. [11]

In multi-processor system, verification of cache coherence protocol becomes more difficult. Especially, in multiple chip multi-core system, the protocol should consider not only the coherence problem inside the chip, but also the coherence between chips. Interfaces between different layers of protocol are hard to implement, and may generate a lot of intermediate states. The scale of state space will expand rapidly, and make it difficult to use finite state machine to do the verification. How to give an appropriate method to verify the cache coherence protocol is an issue requires further research.

5.4. Fault-tolerant mechanism for Cache coherence protocol

Fault-tolerant mechanism and accuracy are two different concepts for Cache coherence protocol. The correctness of the Cache coherence protocol is responsible for ensuring that the processors can execute in correct order according to the request. And it allows multiple processors have a consistent view for shared storage space. Fault-tolerant mechanism is a tolerance and repair mechanisms that used for cope with abnormalities [12].

With the decreasing size of circuit and increasing frequency of processor, systems become more and more sensitive to errors. For example, abnormally loss of Invalid message in snooping bus protocol or confirmation message in directory-based protocol will lead to data coherence problem. And these problems are difficult to notice.

The fault tolerance of the system can be improved by applying some hardware or software methods. For example, using hardware method to provide high fault-tolerant link, or using software method to do error detection and help the system resuming from errors. Fault-tolerant mechanism is necessary for cache coherence protocols, especially for those applications which require high level of reliability.

6. Summary

This paper analyzes cache coherence problem in shared memory multi-processor, and introduced three typical coherence protocols. Snooping bus protocol fits for small or medium scale systems; directory-based protocol works better on large-scale systems; token coherence protocol is a good choice for the applications which have special requirements. Each protocol has advantages and limitations, designer need to

choose an appropriate protocol according to the feature of application.

The paper also analyzes the challenges for future design from four aspects: Collaborative design of cache coherence protocol and on-chip interconnection structure; optimization strategy for reduce the power consumption; verification of cache coherence protocol; and fault-tolerant mechanism for cache coherence protocol.

In sum, a design of an efficient cache coherence protocol is the key to improve the performance of shared memory multi-processor system.

References

[1] HUANG, An-wen, and Min-xuan ZHANG. "Key Techniques of Cache Coherence Protocol for Multi-Core Processor." *Computer Engineering & Science* (2009): S1.

[2] Metcalf, Christopher D. "Managing cache coherence." U.S. Patent 8,392,661, issued March 5, 2013.

[3] Al-Hothali, Samaher, Safeeullah Soomro, Khurram Tanvir, and Ruchi Tuli. "Snoopy and Directory Based Cache Coherence Protocols: A Critical Analysis."*Journal of Information & Communication Technology* 4, no. 1 (2010): 01-10.

[4] Sorin, Daniel J., Mark D. Hill, and David A. Wood. "A primer on memory consistency and cache coherence." *Synthesis Lectures on Computer Architecture* 6, no. 3 (2011): 1-212.

[5] Hum, Herbert HJ, and James R. Goodman. "Forward state for use in cache coherency in a multiprocessor system." U.S. Patent 6,922,756, issued July 26, 2005.

[6] Brown, Jeffery A., Rakesh Kumar, and Dean Tullsen. "Proximity-aware directory-based coherence for multi-core processor architectures." In*Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pp. 126-134. ACM, 2007.

[7] Martin, Milo MK, Mark D. Hill, and David A. Wood. "Token coherence: A new

framework for shared-memory multiprocessors." *Micro, IEEE* 23, no. 6 (2003): 108-116.

[8] Ainsworth, Thomas William, and Timothy Mark Pinkston. "Characterizing the Cell EIB on-chip network." *Micro, IEEE* 27, no. 5 (2007): 6-14.

[9] Patel, Avadh, and Kanad Ghose. "Energy-efficient mesi cache coherence with pro-active snoop filtering for multicore microprocessors." In *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on*, pp. 247-252. IEEE, 2008.

[10] Atoofian, Ehsan, and Amirali Baniasadi. "A power-aware prediction-based cache coherence protocol for chip multiprocessors." In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1-8. IEEE, 2007.

[11] Meixner, Albert, and Daniel J. Sorin. "Dynamic verification of memory consistency in cache-coherent multithreaded computer architectures."*Dependable and Secure Computing, IEEE Transactions on* 6, no. 1 (2009): 18-31.

[12] Fernández-Pascual, Ricardo, Jose M. Garcia, Manuel E. Acacio, and José Duato. "A low overhead fault tolerant coherence protocol for CMP architectures." In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pp. 157-168. IEEE, 2007.