A Cache Coherency Scheme for an Asynchronous Packet-Switched Shared Memory Multiprocessor

Sheran Ales and Syed Mahmud
Dept. of Electrical & Computer Engineering
Wayne State University
Detroit, MI 48202.

Abstract

This paper analyses the problems encountered in designing a bus-based cache coherence protocol for an asynchronous packet switched multiprocessor system having private caches for each processor and describes such an implementation, showing the algorithm used in maintaining cache coherency. Multiple copies of the data are allowed to exist. Since there is no directory that keeps track of all the processors caching data, multiple messages need to be broadcast to all caches whenever coherency needs to be maintained. On the other hand, the Scaleable Coherent Interface (SCI) protocol [1,2] maintains a doubly linked-list of all caches sharing each data with the head pointer maintained at a memory controller. This paper will compare the above two schemes and discuss their corresponding performance and design issues for both small and large scaleable multiprocessors.

1 Introduction

Cache memory is used to bridge the gap between the faster processors and the slower memory. In shared memory multiprocessor systems, a cache is associated with each processor so as to reduce the memory access latency and memory-bus traffic. However, the use of these local caches introduces the data inconsistency problem. This problem can be solved using a suitable cache coherency protocol. Snoopy cache protocols are popular for bus-based systems because it takes advantage of the broadcast capabilities of the bus making it easy to implement such a coherency protocol. This is true for synchronous circuit-switched systems but becomes progressively more difficult for asynchronous packet-switched systems. However, due to communication latency problems, snoopy cache protocols do not scale well. There have been a number of cache coherence protocols discussed in the literature [2,4,6,7], but only a very few handle the packet-switched network [2]. With the present technology, the speed of the bus has increased immensely with the use of fiber-optics, making the packet-switched network more promising than ever. Broadcasting coherence messages is economical for small bus-based systems, which could be scaled up to a few buses. For scaleable high-performance multiprocessors, the directory schemes [2] are more attractive. In such schemes, a directory is maintained which stores information on which processor has cached a given memory block. This helps in reducing the bus traffic on a read or write, as now there is no necessity for broadcasting coherence messages but instead the directory can send messages to the specific processor-caches sharing the block. As the number of processors are increased, the memory needed to hold this information increases proportionately. But many limited directory schemes have been proposed in [6] that performs quite well compared to the full-directory scheme.

The DASH multiprocessor [5] uses a combination of the snoopy scheme and the directory scheme to implement the cache-coherence protocol. The DASH architecture consists of several processing clusters interconnected by a mesh network. Each processing node consists of several processors with private caches, a portion of the global memory, and the corresponding directory memory and controller. Caches within each cluster is kept consistent using a bus-based snoopy scheme, while the inter-cluster consistency is maintained with a directory-based scheme.

The Generalized bus (GB) multiprocessor is a bus-based system which is a variation of the multiple bus structure. The throughput of the asynchronous packet-switched GB [3] has been shown to be comparable but much more economical (number of connections per bus) when compared to the other existing bus architectures. Figure 1, shows the GB structure with N processors, M memory modules, B buses, with G_p processor groups and G_m memory groups. Each processor group of the GB is a cluster which consists of processors having private caches, and having a suitable bus-based cache-coherence scheme for maintaining the intra-cluster consistency. A suitable directory scheme could be used to maintain consistency between each processor group or cluster.

Therefore, for scaleable multiprocessors the combination of the bus-based protocols and the directory-based protocols is the most economical solution in maintaining cache coherency. It is evident that a suitable bus-based cache coherence scheme for a packet-switched single bus needs to be designed, so that it could be conveniently adapted for multiple bus structures.

2 Packet-Switched Cache Coherency Protocol

The cache coherence protocol developed uses a write-invalidate ownership scheme similar to the Illinois write-invalidate protocol [4], but with some additional transient states. These transients states were needed to remember the outstanding requests, thereby allowing the deadlocks from multiple requests and data integrity to be successfully resolved. Figure 2, shows a line in the cache directory controller, where N is the number of processors in the system. In this protocol the owner is either the last writer or the cache having an exclusive copy is the owner. The "Owner ID" specifies the owner of the corresponding data block, which could be either a cache or memory module specified by "Cache/Mem". The "Copy Count" tells the owner the number of caches sharing the block, which it needs for successful invalidation.

![Figure 1. The Generalized-Bus architecture](image1)

![Figure 2. Cache Line](image2)
States:
- INValid
- MODified EXClusive (Owner)
- UNMODified EXClusive (Owner)
- UNMODified SourCe (Owner)
- UNMODified SHareD
- Wait 1 (Wait for the Dec-Ack signal after block replacement due to a read-miss)
- Wait 2 (Wait for the Dec-Ack signal after block replacement due to a write-miss)
- Wait 3 (Wait for the block after a read-miss)
- Wait 4 (Wait for the ownership after a write-hit)
- Wait 5 (Wait for the ownership after a write-miss)
- Wait 6 (Wait for Inv.-Ack. signals, Owner)
- Wait 7 (Wait for self-read-miss)
- Wait 8 (Wait for self-ownership-request)

2.1 Algorithm

INVALID or BLOCK NOT-PRESENT
- Processor Activities
  Read:
  Assume that memory Block-i is referenced.
  If the cache is full, Then (The cache is full)
  Begin
  A block (say memory Block-j) will be selected for replacement. If the cache is the owner of Block-j, Then
  Begin
    If Block-j is modified then it will be written into memory.
    The count value of Block-j will be decremented by 1.
    All other directory bits of Block-i are initialized to 0.
  End
  End
  Else
  Begin
  A decrement-count signal for Block-j will be sent to the owner. The state of Block-j is initialized to WAIT-1, where it will be waiting for a Dec-Ack signal from the owner. All other directory bits of Block-i are initialized to 0.
  End
  Write:
  Assume that memory Block-i is referenced.
  If the cache is full, Then
  Begin
  (The cache is full)
  A block (say memory Block-j) will be selected for replacement. If the cache is the owner of the block-j, Then
  Begin
    If Block-j is modified, then it will be written into memory. The count value of Block-j will be decremented by 1. All other directory bits of Block-i are initialized to 0.
  End
  Else
  End

DIRTY (The cache is the owner of the block)
- Processor Activities
  Read:
  The operation can be performed without any delay.
  Write:
  The operation can be performed without any delay.

- Bus Activities
  Read-Add:
  Block-count is incremented by 1. The block will also be written into memory.
  State is changed to UNMOD_SRC.

  Ownership-Request:
  The ownership is relinquished. The block is supplied with the count value. The state of the block is changed to INVALID. Broadcast a change-in-ownership signal.

UNMODIFIED SOURCE (The cache is the owner of the block)
- Processor Activities
  Read:
  Read operation can be performed without any delay.
  Write:
  An invalidation signal is sent to the bus. State is changed to WAIT-6, where it is waiting for receiving Invalid-Ack signals.

  Bus Activities
  Read-Add:
  Block-count is incremented by 1. The block will also be written into memory.

  Ownership-Request:
  If the ownership-request is due to a write hit on a shared-clean block, Then
  Begin
  Block-count is decremented by 1. The ownership is relinquished.
  The block is supplied with the count value. The state of the block is changed to INVALID. Broadcast a change-in-ownership signal.
  End

  Decrement-Count:
  Decrement the count by 1 and send a Dec-Ack signal.

UNMODIFIED SHARED
- Processor Activities
  Read:
  Read operation can be performed without any delay.
  Write:
  An ownership request is sent to the current owner of the block. State is changed to WAIT-4.
WAIT 1 (Wait for Dec-Ack signal after block replacement due to a read-miss)

- Processor Activities
  No processor activity is possible if every processor is allowed to have only one process.

- Bus Activities
  Change-in-Ownership:
  Another decrement-count signal for Block-i is sent to the new owner.

Dec-Ack:
Broadcast a read-miss signal for Block-i. The state of Block-i is initialized to WAIT-3, where it will be waiting to receive data for Block-i from the owner.

Invalidation Signal for Block-i:
Considered as a Dec-Ack signal and do the same operations which are done for Dec-Ack signal.

WAIT 2 (Wait for Dec-Ack signal after block replacement due to a write-miss)

- Processor Activities
  No processor activity is possible if every processor is allowed to have only one process.

- Bus Activities
  Change-in-Ownership:
  Another decrement-count signal for Block-i is sent to the new owner.

Dec-Ack:
Broadcast an ownership-request signal for Block-i. The state of Block-i is initialized to WAIT-3, where it will be waiting to receive the content of Block-i from the owner along with the ownership.

Invalidation Signal for Block-i:
Simalar to a Dec-Ack signal.

WAIT 3 (Wait for the self-read-miss signal)

- Processor Activities
  No processor activity is possible if every processor is allowed to have only one process.

- Bus Activities
  Self-Read-Miss:
  Change the state of Block-i to WAIT-7.

WAIT 4 (Wait for the block after a read-miss)

- Processor Activities
  No processor activity is possible if every processor is allowed to have only one process.

- Bus Activities
  Change-in-Ownership:
  Another read-miss signal is sent to the new owner. Change the state of Block-i to WAIT-3

Block Arrived:
The data and block count is loaded.
If block-count = 1
Then load the block in Exclusive-Clean State.
Else load the block in Shared-Clean State.

Operation of the Memory Controller

Bus Activities

Read-Addr:
If OW=1 Then
Begin
Block-count is incremented by 1. The block is sent with the Block-count. If Block-count = 1, then OW bit is set and a change-in-ownership signal is broadcast.
End

Ownership-Request:
If OW=1 Then
Begin
If the ownership-request is due to a write miss. Then Block-count is incremented by 1. The block is sent with the Block-count. OW bit is set and a change-in-ownership signal is broadcast.
End

Change-in-Ownership:
If the ownership is transferred to the memory then OW bit is set.
Decrement-Count:
If OW=1, then Block-count is decremented by 1 and a Dec-Ack signal is sent.

Block Arrived from Source:
The block is written into memory.
2.2 Different Types of Bus Activities

<table>
<thead>
<tr>
<th>Activity</th>
<th>Type of Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-Miss</td>
<td>first: broadcast, successive: one-to-one</td>
</tr>
<tr>
<td>Ownership-Request</td>
<td>(write-hit: one-to-one; write-miss: first: broadcast, successive: one-to-one)</td>
</tr>
<tr>
<td>Change-In-Ownership</td>
<td>(broadcast)</td>
</tr>
<tr>
<td>Block-Arrived</td>
<td>(one-to-one)</td>
</tr>
<tr>
<td>Decrement-Count</td>
<td>(one-to-one)</td>
</tr>
<tr>
<td>Decrement-Acknowledge</td>
<td>(one-to-one)</td>
</tr>
<tr>
<td>Invalid-Acknowledge</td>
<td>(one-to-one)</td>
</tr>
<tr>
<td>Invalidation</td>
<td>(broadcast)</td>
</tr>
</tbody>
</table>

3 Comparison with the SCI Protocol

- Since the head of the linked-list in the SCI protocol is located in the memory controller, the memory controller will be heavily utilized which becomes a major bottleneck in the system. In "our" protocol the messages are more uniformly distributed as there is no centralized control, also making it more fault-tolerant. However, the SCI could have a distributed directory for increased fault-tolerance.

- In "our" protocol there is some broadcast messages. The packets arrive into each cache controller queue. If the controller is slow it only slows down its particular transaction but does not affect the other transactions. In bus-snooping mechanisms, the cache controllers should be fast, otherwise the speed will be limited by the slowest cache. Also by having a SOURCE and COUNT the number of messages are effectively lessened. SCI has to sequentially traverse the linked-list to know the next shared location.

- In the case of adding shared data block, the memory controller in the SCI scheme always adds the new entry to the list and head-ownership is passed sequentially through the list. The addition of new entries in the sharing-list is performed in FIFO defined by the arrival of requests at the memory controller. In "our" protocol, at the time the block is supplied by the source, its COUNT will be incremented indicating the addition of a new shared block. The COUNT value in the source only need be updated.

- When the cache block has to be replaced, The SCI scheme needs to delete the corresponding entry from its list, which involves the update of the back_id in the next entry and the forward_id in the previous entry (closer to the memory controller). In the case of the tail entry, only the previous entry pointer is updated. During these operations, the entry is converted to a locked state. In "our" protocol, a decrement message is sent to the owner, while it remains locked (wait state) until an acknowledgment is received.

- In the case of a write operation, an exclusive copy must be obtained prior to the write. In the SCI scheme, only the head has the authority to purge other entries from the list. The purging is done similar to the deletion process. Again the block waits until the whole process is complete. In "our" protocol, the ownership is first obtained, thereafter an invalidation is broadcast and the block waits until all acknowledge messages are received.

- Likewise, many read misses in the SCI require more inter-node communication, since only the head can supply the block. Also if a miss is replacing a valid block, the replaced block must be detached from its sharing list. When these additions or deletions need to be done on sharing lists, the SCI protocol becomes locked until the transaction is complete. However, "our" protocol maintains different wait states to distinguish various events allowing the use of context switching of programs during these waits.

- The SCI scales well as it is a directory-based scheme that could be distributed, there is two pointers per cached copy plus one pointer per cache line in memory. "Our" protocol is beneficial for packet broadcasting in small clusters that used bus-based protocols. It is much faster in cases where there are few processors per cluster but has the overhead of any broadcast protocol.

Conclusions

With the advent of better technology in buses, cluster based scaleable multiprocessors and even small bus-based systems will still be conceived. A corresponding packet-switched cache coherence protocol has been developed for bus-based systems. Eight wait states were incorporated to maintain data integrity during sharing, which is inevitable in an asynchronous scheme. Simultaneous invalidations too should not generate deadlocks or starvation. These wait states could be well utilized through context switching. The SCI protocol [2] is a point-to-point scheme which is good in different network architectures including bus-based schemes. "Our" protocol was developed as an extension to the existing broadcast protocols that exist for bus-based systems. It can be easily incorporated into the cluster based systems discussed previously. It shows the complexity involved in maintaining cache coherence in an asynchronous packet switched bus, which had previously not been given much attention.

References