Cache Coherence

This lesson discusses the problems and solutions for coherence. Different coherence protocols are discussed, including: MSI, MOSI, MOESI, and Directory. Each has advantages and disadvantages depending upon the program being executed and the number of cores in the system.

Cache Coherence Problem

The programmer expects to see shared memory.
Since each core has its own cache, cache coherence can become a problem because each cache can have its own copy of the same memory location.

Incoherent - each cache copy behaves as an individual copy, instead of as the same memory location.

Coherence Definition

There are 3 requirements for coherence

1. If a core reads a memory location, the data is receives was written by the last valid write.

2. If a core writes to a memory location, when another core reads that same memory location, it should see the same value. Any core should be able to read the last valid write to a memory location.

3. All cores should agree on the order of the writes to a memory location.

How to Get Coherence

- Don't do caches. The main memory will be coherent : This leads to poor performance
-All cores share the same L1 cache. This leads to poor performance
-Use private write through caches. This is not coherent.

To maintain coherence property 2:

-Broadcast all writes so other cores can update their caches. This is write-update coherence.
-A write will make any other copies of the data invalid, so other cores will not be able to use old data. This is write-invalidate coherence.

To maintain coherence property 3:

-Snooping: all writes are put on a shared bus and the cores snoop the bus to get the updated information for their caches.
-Directory based: each block state is maintained by a directory. When a write occurs the directory reflects the state change.

Write-Update Snooping Coherence
Cores snoop to check for writes by other cores. When a write is detected, any other copies of the block are updated by the new data.

Update Vs. Invalidate Coherence
If an application has a burst of write to one address -- invalidate is the better method.
If an application writes to different words in the same block -- invalidate is the better method.
If one core writes and another core reads often -- update is the better method

All modern processors use the invalidate process, because it is better when a thread moves to another core.

Write Update Optimization
Avoiding memory writes:

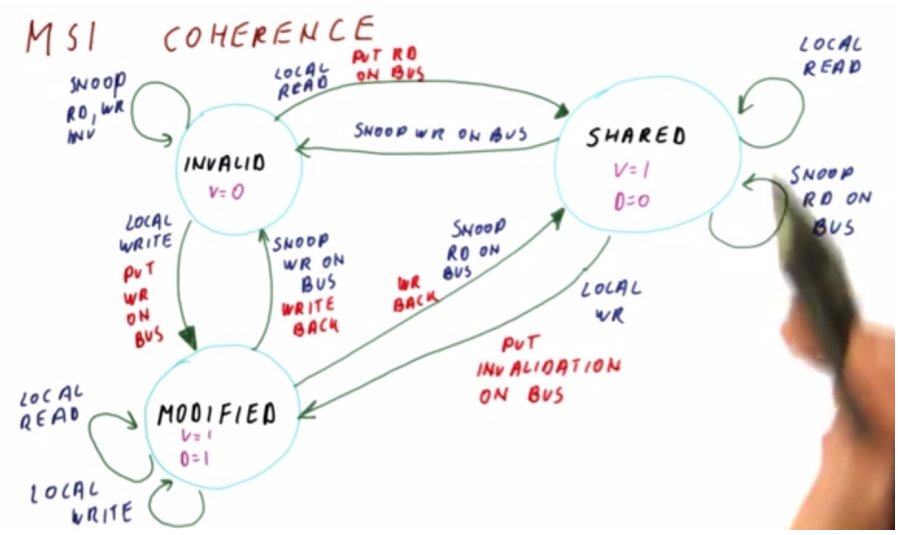
Writes need to be broadcast on the bus, so memory throughput becomes a bottleneck. To improve the bottleneck, the writes should be delayed to memory. To maintain coherence a dirty bit is added to the cache for each block. When the data is written by a core, all other caches are updated. Dirty data is updated in memory when the block is replaced in the cache.

Dirty Bit Benefits

-Writes to memory are greatly reduced.
-Reads from memory are also greatly reduced.

Write Invalidate Snooping Coherence
There is a shared bit in write-invalidate snooping.
A write causes other copies to be invalidated, this will cause a miss if a core wants to access the data. The newly written cache is the only valid copy and it will respond to any requests for the data. If a read is requested, the shared bit will be set to 1, showing there is more than copy of this data.
Disadvantage: there is a miss on all the readers when a core writes
Advantage: if a core needs to update the same block two or more times, the reads and writes can be done locally after the first write.

MSI Coherence
This is an invalidation based protocol.



Cache to Cache Transfers

Cache to cache transfers occur when a cache (C1) owns the data (the block is in the 'M' state) and a read request for the data (C2) is detected on the bus. C1 must supply the data because it has the only valid copy of the data.

Possible methods to do this:

1. Abort and retry
   Downside of this approach - there needs to be two memory latencies to get the data to the requester.

2. Intervention
   The core that owns the data intervenes and tells the memory it will respond. An intervention signal must be added to the bus.
   Disadvantage of this method: hardware is more complex

Modern processors use the Intervention method.

Avoiding Memory Writes on Cache to Cache Transfers

When using the Intervention method, the memory needs to be written when there is a read of modified data.

It would be better if the memory was only written when the block is kicked out the cache -- the Owner would be a new block state that is responsible for responding to read requests and updating memory.

MOSI Coherence

The 'O' state is like the 'S' state except:

1. when a read is detected - the owner responds
2. write-back to memory when the block is replaced

M = a core has modified the data and has the only valid copy of the data
S = at least one core has the block in its cache and it is clean
O = a core has modified the data, and has shared the modified data with at least one other core

M(O)SI Inefficiency
There is still inefficiencies in MOSI. When going from a Shared state to a Modified state, the block must pass through the invalid state. To eliminate this step a new state is introduced, the Exclusive state.

The E State
The exclusive state is used when a core is the only core that has a clean copy of the data. When a block is in the 'E' state it can move to the 'M' state directly because no other core has a copy of the data.

Directory Based Coherence
Snooping downside: every request must be broadcast, which means there must be one bus. This leads to a bottleneck and snooping can be scalded to more than 16 processors.

To eliminate the need to broadcast, while still observing the coherence requirements, a directory can be used.

Directory
A directory is:
    Distributed across all cores, each core has its own 'slice'

Each slice serves a set of blocks.
The directory keeps track of which caches have the block, for valid states only.

The Directory Entry
The directory entry has:
    1 Dirty bit
    1 bit/Cache Present/Not Present
        0 = block is not present in a valid state in the cache
        For example:
        for an 8 core system there are 8 bits for signifying present/not present
The directory communication requires an acknowledgement from the cores after a request.

Cache Misses with Coherence
The three 'C's' are now four:
    Compulsory, Conflict, Capacity
    Coherence Miss - a miss caused by coherence.

For example: A core (C1) reads a memory location, then another core (C2) writes to that location. When C1 attempts to read the memory location again, the data is invalid. This is a miss due to coherence.

Two types of coherence misses:

True Sharing- different cores access the same data

False Sharing- different cores are accessing different memory locations, but these memory locations are in the same block. From the standpoint of coherence, data in the same block are the same data.