Lawrence Park (995936395) Sam Shao (1000084151)

Lab Assignment 5 Report: Coherence

1. Prelab Questions

1) Why are transient states necessary?

Shared memory accesses (i.e. block fetch, eviction, and etc.) takes different latencies to complete, and a valid coherence protocol must enforce atomicity of state transitions for correct operations. Transient states implement the atomicity of coherence FSMs by stalling cache accesses until a given memory access completes.

2) Why does a coherence protocol use stalls?

Stalls are necessary to guarantee atomicity of FSM transitions. The pending cache and coherence events cannot proceed until the FSM state resolves to a stable state.

3) What is deadlock and how can we avoid it?

Deadlock is a state at which no forward progress can be made in a system due to a cyclic decencies between two modules. Within the context of cache coherence protocol, a deadlock can occur between two processors waiting on each other to complete a transaction on a shared resource. These deadlocks can be avoided by allocating separate virtual network (i.e. queues, networks) for different message types – 1) cache-initiated, 2) directory-initiated, and 3) cache responses

4) What is the functionality of Put-Ack (i.e. WB-Ack) messages? They are used as a response to which message types?

Put-Ack messages are issued by the directory to the requestor processor that is trying to evict a cache block. To evict a cache block, the requestor sends out a PUTM or PUTS request to the directory, and awaits for Put-Ack to complete the eviction.

5) What determines who is the sender of a data reply to the requesting core? Which are the possible options?

The directory keeps tracks of the owners of cache blocks in the system. If a given block is shared, the data reply can be sent from the directory. If the block is in a modified state, the directory forwards the data request to the owner processor, and the owner will return the data back to the requestor.

6) What is the difference between a Put-Ack and an Inv-Ack message?

Put-Ack messages are issued by the directory to the requestor processor that is trying to evict a cache block. Inv-Ack messages are sent by processors that are currently sharing a given cache block upon receiving a Fwd-GetM message from the directory. The recipient of Inv-Ack messages is the processor that is trying to get a write access to the shared cache block; the Inv-Ack messages allow the upgrading core to determine that it has an exclusive access to its M block.

2. Methodology Questions

1) How does the FSM know it has received the last Inv Ack reply for a TBE entry?

The directory keeps a list of processors who are sharing a given block. When a processor sends out GETM request, the directory will send the block with an ack. counter greater than 0, if there are processors currently sharing the block. Once the directory response is received, the requestor can simply from the list to determine which Inv-Ack message constitute the last. This happens when the FSM of the requestor is in IM_A or SM_A states.

2) How is it possible to receive a PUTM request from a Non-Owner core? Please provide a set of events that will lead to this scenario.

Let's assume that P1 has a block A in M state. If another processor, P2, wishes to obtain M access to the block A, it will send a GETM request to the directory. If P1 evicts the block A, triggering a PUTM request, after the GETM request is received by the directory, the owner of the block A is changed to P2, and the PUTM request by P1 is from a non-owner core. P1, no-longer being the owner of the block A, will transition to II_A state after receiving Fwd_GETM request, triggered from P2.

3) Why do we need to differentiate between a PUTS and a PUTS-last request?

The directory needs to keep a list of processors sharing a block in S state, so that it can invalidate the block when there is no longer a processor holding the block. The PUTS-last event signifies that the evicting processor is the last sharer of the given block, and this event will trigger the directory to invalidate the block. On the other hand, the PUTS event does not trigger a FSM change since the block is still shared by the remaining processors.

4) How is it possible to receive a PUTS Last request for a block in modified state in the directory? Please provide a set of events that will make this possible.

Let's consider a case two processors - P1 and P2 - sharing block A. P2 sends a GETM request for write access; at the same time, P1 issues a PUTS to the directory due to cache eviction. If the P2's GETM request is received by the directory first, the directory sets the owner of the block to P2, and forwards an invalidation request to P2. The PUTS that was sent by P1 is now considered as a PUTS-Last for the block A which is marked as Modified.

5) Why is it not possible to get an Invalidation request for a cache block in a Modified state? Please explain.

The MSI protocol dictates that the modified cache block only have one processor accessing the block. Since there are no peer processors using the modified block, the directory will never forward an invalidation request.

6) Why is it not possible for the cache controller to get a Fwd-GetS request for a block in SI_A state? Please explain.

The SI_A state indicates the transient state where a shared block is waiting for an Invalidation event from the directory to complete its block eviction (i.e. S=>I). A forwarded GETS event is used to demote an access permission from M to S on the current owner to allow other processors to share the block in S state. The Fwd-GetS will not be issued to a processor, currently waiting to evict the shared block (i.e. in SI_A state), because it hasn't been the owner of the block in the first place.

7) Was your verification testing exhaustive? How can you ensure that the random tester has exercised all possible transitions?

To verify the MSI protocol implementation, it is important to check if all intended FSM transitions were hit during the random tests. We incrementally added the debug print statements to see if the coded transition were hit properly, and made sure that our changes did not cause any error.

3. Table Modifications

As per the lab instructions, we had to model transient states in the directory where data blocks in the directory are accessed from the main memory. This requires the introduction of the following transient states:

Transient States	**Description**
IM_D	I=>M intermediate state: GETM request is sent to the main memory; waiting for Memory Data to complete the transition.
IS_D	I=>S intermediate state: GETS request is sent to the main memory; waiting for Memory Data to complete the transition.
MI_A	M=>I intermediate state: send block data to the main memory for write-back after a PUTM_Owner event. Waiting for Memory Ack to complete the transition.

4. Division of Work

We completed this lab by pair-coding. We reasoned that we would make less mistakes by discussing each FSM transition together in the same room as we incrementally built the MSI protocol.