ECE552 Lab 5 Report

Marko Ciric (1006723967), Raymond Huynh (1007058238)

Prelab Questions

1. Why are transient states necessary?

Transient states are necessary since they help facilitate smooth transitions between cache states by managing intermediate steps during operations like reads, writes, and invalidations. They prevent potentially inconsistent transitions by coordinating actions such as writing to a block, cache invalidations, and directory updates. For example, when a processor writes to a block that is shared, a transient state like "Pending" allows the system to ensure other caches are properly notified and invalidated before the block transitions to M.

2. Why does a coherence protocol use stalls?

Stalls in a coherence protocol are used to ensure data consistency and synchronization between caches. They occur when a processor must wait for coherence actions, like invalidations, updates, or directory responses, before proceeding with read or write operations. They help prevent conflicts and race conditions by ensuring that memory operations are sequential and that caches see consistent data. By temporarily delaying operations, all necessary coherence actions are completed before a processor can safely access or modify memory, preventing data loss or overwriting.

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

A deadlock is when a program cannot proceed with execution due to an interleaving of inter-dependent events, causing an infinite cycle. For example, action A may require the results of action B, but action B may require the results of action A, causing the execution to effectively halt. A solution to this is virtual networks, where each type of message request has their own effective queue, identified by tags, such that there are no conflicts from a response blocking another request.

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 is used to acknowledge that a writeback to memory has been completed. The cache will wait for a Put-Ack before switching the block to I or S, ensuring that the directory and/or cache is properly updated. Since it is used during cache eviction or when the memory updates, it is used as a response to the PutM and PutS message types.

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

The sender of a data reply is determined by who has the most recent copy of the cache block. Generally, this means that whichever block is in the M state, as located by the directory, will be

the sender. Otherwise, if there is no clear block in the M state, the sender may have other cache blocks in the S state, memory, or the directory.

6. What is the difference between a Put-Ack and an Inv-Ack message?
A Put-Ack is used when writeback to memory is completed and the block can transition to I or S, whereas an Inv-Ack is used to acknowledge that a block has successfully been invalidated (I state).

Section 5 Questions

1. How does the FSM know it has received the last Inv Ack reply for a TBE entry?
Since the directory has a list of all the sharers of a particular block, the directory is able to send the cache controller a count of how many Inv-Acks to expect. This count in the TBE entry goes down for every Inv-Ack reply; once 0 is reached, the cache controller will know that the last Inv-Ack reply has been sent.

2. How is it possible to receive a PUTM request from a NonOwner core? Please provide a set of events that will lead to this scenario.
This can be done by the following set of events:

1. Core 1 tries to obtain ownership over a block by making a GetM request to the directory.
2. Core 2, the current owner of the block, sends a PutM request to the directory
3. The GetM request is received by the directory, and transfers ownership from Core 2 to Core 1 for the block
4. The PutM request from Core 2 is received by the directory, but Core 2 has already lost ownership at this point

3. Why do we need to differentiate between a PUTS and a PUTS-Last request?
We need to differentiate between a PUTS and a PUTS-Last request because the directory must retain the block until it receives a PUTS-Last request, which indicates that there are no more sharers (no more copies of the block in the cache), before the directory can invalidate the block.

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.
This can be done by the following set of events:

1. Block is initially in S state across multiple caches
2. Core 1's cache sends a GetM request to the directory to acquire write access
3. Directory updates block state from S to M and sends Inv messages to all the block's sharers and tracks the AckCount to make sure all sharers are accounted for
4. Before receiving Inv, one of the sharers evicts the block and sends a PUTS-Last request, resulting in the block that updated to the M state in the directory receiving the request

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

Only blocks in the Shared state can receive an invalidation request. Since the cache block is in the Modified state, it means that it is the Owner and thus any GetM or GetS requests are forwarded directly to the block, rather than any Invalidation requests.

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.

Only the Owner core receives a Fwd-GetS request. Additionally, upon a Fwd-GetS request, the Owner will switch to the S state and provide its own data, rather than the data of other blocks in the Shared state. Therefore, since a block in SI_A state is not the Owner, it will never receive the Fwd-GetS request.

7. Was your verification testing exhaustive? How can you ensure that the random tester has exercised all possible transitions (i.e., all {State, Event} pairs)?

Our verification testing was fairly exhaustive. We were able to pass all core configurations up to 16, with loads up to 1M and cache sizes and associations varying from the default, to 1kB to 16kB, and full to 8-way associativity. But still it is difficult to verify that the random tester has entered all possible transitions outside the basic ones, making it complicated to cover the edge cases. In theory, by increasing the number of cores and instructions (loads) and decreasing the cache size it should improve the probability of encountering an edge case, but we did not take every precaution. We tried increasing the number of cores past 16 and it failed, but we did not have the time to debug it to see what edge case transitions were missing.

Protocol Modifications

Below are the tables that document all protocol modifications with respect to Tables 8.1 and 8.2

Table 8.1 Modified (New) Transitions

State, Event	**Modification**
I, Inv	Send Inv_Ack
I, WB_Ack	Pop request from queue
IM_AD, Data_from_Dir_Ack_Cnt_Last	Follows same actions as Data_from_Dir_No_Acks but also updates the AckCount
SM_AD, Data_from_Dir_Ack_Cnt_Last	Follows same actions as Data_from_Dir_No_Acks but also updates the AckCount

Table 8.2 Modified (New) Transitions

State, Event	**Modification**
I, GetS	Add Req to Sharers/S, request data from memory
I, GetM	Set Owner to Req/M, request data from memory
IS_D, Memory_Data	Send data to Req/S
IS_D, {GetS. GetM}	Stall
IM_D, {GetS, GetM, PutM+data from NonOwner}	Stall
IM_D, MemoryData	Send data to Req/S
MS_D, Data	Write data to directory, copy data to memory/S
MS_D, {GetS. GetM, PutS_Last}	Stall
MS_D, {PUTS_NotLast, PUTM_Owner, PUTM_NotOwner}	Remove Req from Sharers/S, send Put-Ack to Req
MS_A, Memory_Ack	-/S
MS_A, {GETS, GETM, PUTM_NotOwner, PUTM_Owner, PUTS_Last, PUTS_NotLast}	Stall
MI_A, Memory_Ack	Send Put-Ack to Req, clear Owner
MI_A, {GETS, GETM, PUTM_Owner, PUTM_NotOwner, PUTS_Last, PUTS_NotLast}	Stall
S_D, All events	Removed and broken up into constituent transient states (IS_D, MS_D, MS_A)

Statement of Work

Raymond oversaw the states and transitions in MSI-dir.sm and Marko oversaw the states and transitions in MSI-cache.sm. The report was written collaboratively.