Kevin Lee 997237805
Calvin Fernandes 997588873

Lab Report: Cache Coherence

1 PRE-LAB

1. Transient states are required because state transitions and their associated actions cannot be done instantaneously or atomically in a real system. Transient states are also needed to prevent the system from violating the single write/multiple readers invariant. Ex. Have to issue a read request and wait for a data reply only then a transition can be made. The sequence previously described is not atomic.

2. Coherence protocols use stalls because actions must wait if a cache block is in a transient state, and actions can take several cycles. In these transient states we are waiting for data or acks from either the directory or owner depending on the situation. Stalls are also done to try and enforce some sort of atomicity between actions that are conceptually atomic but are not realistically.

3. Deadlock is a condition where no forward progress can be made due to a cyclical dependence. For example, if processor A sends a request to processor B, but then stalls. Instructions queue up for processor A and stall, followed by B sending the proceed signal. However, this signal will get stuck in the queue leading to a deadlock. Deadlocks can be avoided by using separate queues depending on the type of signals sent.

   Example:
   Block X is in M state in P2 cache. P1 issue load for Block X and misses in its local cache. P1 asks directory (read request) to get shared permissions. Directory forwards request to P2. Then P1 issues store request on block X but this block is in a transient state causing a stall on P1's end. P2 sees the fwd request from P1 and invalidates itself along with sending data to P1. This data is behind the store request to the same block. Cyclic dependence occurred. Data reply cannot be processed till it hits the head of the queue. However, the store request is sitting there.

4. When a cache entry needs to move from the modified to invalid state (such as in an eviction), a PutM signal must be sent to the directory. In this case, the directory must transition that block to the invalid state. The processor sending the PutM must wait for this transition, before transitioning to the invalid state, to maintain consistency with the directory. Put-Acks are used as a response to PutM messages.

5. The sender of data is determined by the current state of the cache block in the directory. If the block is listed as invalid or shared in the directory, the directory will send the data. If the block is listed as modified in the directory, then the data must come from the processor that has the block in the modified state.

6. A put-ack is sent by the directory to the requesting processor, telling it that it is okay to transition from the modified/shared state to invalid state. An inv-ack is sent by one or several processor to another processor telling it that they have invalidated the data (for example when you have a block shared by multiple processors and another processor wants M permissions). Once a processor receives inv-acks from all processors that had the block shared, it can transition to the modified state.

2 METHODOLOGY

1. Let P1 (a processor) have the need to acquire write permissions to block A (it is in invalid state to begin with) and send a GETM request to the directory. Block A happens to be shared and therefore the case will occur when the directory will send the data along with an ack>0 back to the requestor (P1). The FSM will decrement the ack counter every time an ack is sent back. Therefore the FSM can just count the number of acks in order to find out when the last one will occur. This will happen in the IM_A/SM_A states – these are the states that have TBE entries and they will receive the Inv-Ack replies.

2. A PUTM from a non-owner core can occur in this scenario:
   P1 has block X in M state. P2 does a store to block X and it misses. P2 sends a GetM to the directory. Right at that point P1 decides to evict block X and sends a PutM request to the directory. It so happens that P2's (GetM) request gets to the directory first. The directory switches the block's permissions to the new owner (P2) and the directory issues the Fwd-GetM to P1. Now the directory sees PutM from P1 and it is a non-owner.

3. If the directory has a block in the shared state and the block needs to go to the invalid state and there are multiple sharers. Then we need to know when there are no more sharers remaining. If there are sharers remaining we stay in the S state waiting for more PutS events. If there are no more sharers remaining then the block must transition to the invalid state. We have two different sets of actions based on the number of sharers. This is why we need to differentiate.

4. Consider the case where there are 2 sharers that share block A. Lets call these sharers P1 and P2. Now P2 wants to acquire write permissions to the block. It sends a GetM to directory. However, at the same time P1 send PutS requests to the directory as it wants to evict the block from its cache. It so happens that the requests are serialized and the GetM is seen by the directory first followed by the PutS from P1. The directory will change ownership of the block to P2 and send an invalidation request to P1. Then the block is in the M state and it will receive a PutS_NotLast that was pending in the queue from the previous sharer, P1 who is now pending invalidation. This is the case where it occurs.

5. It is not possible to get an Invalidation request for a cache block in the modified state because there is only 1 owner. Invalidate requests are usually done when there are sharer(s) that need to be invalidated when 1 core wants write permission for the block. Let us consider a scenario when there are 2 sharers (A and B). Both sharers happen to want

M permissions for the same block and therefore both send a GetM to the directory. Let's say that the requests got serialized in the queue and sharer A happened to get there first. The directory will send an Invalidate to sharer B and make sharer A the owner. At this point the directory is updated to the modified state therefore when B's GetM makes it there it will not send an invalidate request because it sees that the block is now in the M state and instead will send a Fwd-GetM. The same case would apply when both processors (A and B) want M permissions of the same block but don't have in their cache (invalid state).

6. In the SI_A transient state we are going from state S to state I and we are waiting for an invalidate event to occur. In order to get into the SI_A transient state, a replacement has to take place while the block is in the S state. When a replacement event occurs, a PutS is sent to the directory and the directory would remove this sharer from the list of sharers. A forward GetS is only used when there is an existing owner of the the block with M permissions and someone else wants read permissions thereby demoting the owner to a sharer. In this case, this is a sharer to begin with and not an owner so therefore a fwd-gets event will never occur. Also consider another scenario when MI_A transient state is occurring so that means the block is moving from M to I and is waiting for an ack, in this scenario, if you get a fwd-getS you would transition over to the SI_A state. Therefore you have already received a fwd-getS that has caused the transition to the SI_A state and you would not be able to get this again (same as the first case).

7. The verification testing was exhaustive in order to determine if the cache coherence protocol was working correctly. In order to ensure that the random tester has exercised all possible scenario. All transition states were coded. Then each transition state was commented out one at a time. If we would see an error saying that an event occurs and we don't have a case to handle it then it means that the transition state that was commented out was the one that was exercising that particular scenario. Debug print messages along with grep could also be used to check if that transition was executing.

3 MODIFICATIONS

Cache Controller modifications (changes that had to be added to table 8.1)
Coupled Ifetch events with the load events
Had to add a transition to IM_AD and SM_AD when event Data_from_Dir_Ack_Cnt_Last occurs

Directory Controller modifications (changes that had to be added to table 8.2)
Add transient states IM_D, IS_D and MI_A in order to access memory
Had to break up transition from I to S and make it I to IS_D and then from IS_D to S and had to implement stall states for IS_D when events other than a Memory_Data are encountered. Memory_acks had to be acknowledged and popped
Had to break up transition from I to M and make it I to IM_D and then from IM_D to M and had to implement stall states for IS_D when events other than a

Memory_Data are encountered. Memory_acks had to be acknowledged and popped
Had to break up transition from M to I and make it M to MI_A and MI_A to I when a Memory ack occurs. All other events for MI_A had to stall
Had to add a transition for when S_D experiences a Memory_Ack and the memory queue had to be popped

4 WORK BREAKDOWN

Coding work was done by Calvin and iteratively verified and reasoned with by Kevin. Debug work was done by both partners.

5 FEEDBACK

This lab was quite difficult to do due to the lack of debugging capabilities that were previously offered by sim-safe. A new language had to be learned which also added complexity and added difficulty to the lab. Also, the time given to do this lab is barely enough. Many courses have their work due at around this time and are worth a lot more which actually encourages students to push this lab to the very end which could cause them to not complete the lab. It is understood that the purpose of this lab is to show how hard it is to implement a cache coherence protocol in real life with the introduction of transient states, TBEs etc. I would highly recommend implementing an easier protocol such as a VI protocol which would simplify the debugging process. Also, there is far too much information to read. Please cut down on the information overload. I don't believe it is very feasible to squeeze around 25 pages of very hard reading during the end of the semester when a lot of other projects are due. The amount of functions provided at the end also can be very confusing. Try to limit the choice so that students have an easier time identifying what is relevant. I feel like this lab would be better if it was scheduled in place of the prefetcher lab and the prefetcher lab was due in the last week.