Shawn Zhai 1006979389
Victor Wu 1007069039

Pre-lab Questions

1. Why are transient states necessary?

Transient states are necessary to handle the time period during which a block is in the process of transitioning between two stable states. In real hardware, state transitions are not atomic, as messages take time to traverse the interconnect network, reach the destination, and get processed. During this time, transient states help ensure that coherence invariants are not violated, such as when the Single-Writer, Multiple-Reader invariant needs to be upheld while waiting for an acknowledgment or data reply.

2. Why does a coherence protocol use stalls?

A coherence protocol uses stalls to prevent any conflicting memory operations from happening when a block is in a transient state. If a block is in a transient state, it implies that a coherence request is pending, and it is waiting for an external event (e.g., a data reply or an acknowledgment). To avoid coherence violations and maintain consistency, any local load or store request for the block is stalled until the coherence transition completes, allowing the system to safely maintain the correct memory state.

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

Deadlock in coherence protocols occurs when there is a cyclic dependency between resources or events, and no forward progress is possible. For instance, if Core 1 is waiting for data from Core 2 while Core 2 is also waiting for a resource held by Core 1, a deadlock occurs. To avoid deadlock, virtual networks are used to logically separate messages of different types, such as cache-initiated requests, directory-initiated requests, and cache responses. By having separate queues and avoiding cyclic resource dependencies, deadlock is prevented.

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 or WB-Ack messages are used to acknowledge the receipt of a write-back message. They indicate that the directory has received the write-back from a cache and has updated its entry accordingly. Put-Ack messages are used in response to PUTS (Put Shared) and PUTM (Put Modified) requests, where a cache writes back the data block to the directory, either because it is being evicted or because another core has requested exclusive ownership.

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 to the requesting core depends on the current state of the block in the system. The possible options are:

- **Directory**: If the block is in the shared state and the directory has an up-to-date copy of the data, the directory can directly send the data reply.

- **Previous Owner (Cache)**: If the block is in the modified state in another cache, the previous owner must send the data reply to the requesting core, as it has the latest modified version of the block.

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

A Put-Ack message is sent by the directory in response to a cache's PUTS or PUTM message, acknowledging the receipt of a write-back of a shared or modified block. An Inv-Ack message, on the other hand, is sent by a cache in response to an invalidation request from the directory, indicating that the cache has successfully invalidated the specified block.

Methodology Questions

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

The FSM knows it has received the last Inv Ack for a TBE entry by maintaining a pending acknowledgment counter in the TBE. When an invalidate request is sent to multiple caches holding a shared copy, the counter is incremented to reflect the number of expected acknowledgments. Each time an Inv Ack is received, the FSM decrements this counter. When the pending acknowledgment counter reaches zero, the FSM knows that all caches have acknowledged the invalidation, indicating that the block can be safely transitioned to the invalid state or granted exclusive ownership to the requesting cache.

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.

The PUTM from a non-owner core can occur due to out-of-order arrival of coherence messages sent through different virtual channels. Consider a scenario where Core C1 sends a GetM request to the directory to obtain exclusive ownership, while Core C2 is the current owner and sends a PUTM to write back the modified data. The directory first receives C1's GetM, forwards it to C2, and updates the ownership to C1. However, if the PUTM from C2 arrives after this ownership change, it results in the directory receiving a PUTM from a core that is no longer the owner. This scenario is possible due to the two different signals (PUTM and Fwd-GetM) being transmitted through separate virtual channels, leading to potential out-of-order delivery.

3. Why do we need to differentiate between a PUTS and a PUTS-Last request?

PUTS is used when a cache evicts a Shared block but other sharers still exist, prompting the directory to update its sharer list without changing the block state to Invalid. PUTS-Last, on the other hand, indicates that the evicting cache was the last sharer, allowing the directory to mark the block as Invalid and stop tracking sharers.

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.

In the scenario where C1 and C2 both hold the block in the shared state, C2 sends a GetM request to the directory to obtain exclusive access. The directory forwards an invalidation request to C1 and prepares to transition the block to modified

state. During this time, C1 independently decides to evict its Shared copy and sends a PUTS to the directory. Due to the use of different virtual channels, C1's PUTS can arrive after the invalidation request is forwarded, leading to the directory receiving a PUTS-Last request while it has already transitioned the block to modified state, hence resulting in a PUTS-Last for a block initially in the modified state.

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

It is not possible to receive an invalidation request for a cache block in the modified state because the modified state guarantees exclusive ownership of the most recent data, meaning no other cache holds a copy of that block. Invalidation requests are used to invalidate shared copies across multiple caches, which is unnecessary in the modified state since no other cache has the block. Instead, if another core requests the block, the directory directly interacts with the owner to either transfer the block or downgrade its state instead of sending it 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 Fwd-GetS request is only sent when a block is in the modified state and needs to transition to shared state to accommodate multiple readers. Once a block is in the shared state, it is managed by the directory, which provides data directly to any subsequent GetS requests without involving any specific cache. Thus, when a block is in the SI_A state, it cannot receive a Fwd-GetS request because the directory can directly serve the data to requesting cores without needing to forward the request to a cache that is in the process of invalidating its copy.

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)?

The verification was conducted using up to 16 cores and 1 million load counts to maximize concurrency and coverage of different states. To ensure comprehensive testing, we used random combinations of cores and load counts to evaluate the robustness of the random tester. Additionally, to verify that all possible finite state machine (FSM) transitions were exercised, we included print statements for detailed analysis and debugging.

Protocol Modifications

Table	Modification	Description
8.1	Combined IFetch event with Load event	This is because the L1 cache is for both data and instruction
8.2	Added the IS_D transient state	It is to handle the transition when a GETS request arrives at the directory waiting for the data from memory

8.2	Added the IM_D transient state	It is to handle the transition when a GETM request arrives at the directory waiting for the data from memory
8.2	Added the MI_A transient state	It is to handle the transition when a PUTM request is received from the owner and the directory is waiting for the writeback acknowledgment from memory before transitioning to the invalid state.
8.2	Added the MS_AD transient state	It is to handle the transition when a GETS request arrives for a block in the modified state and the directory needs to forward the request to the owner, collect the modified data, and transition to the shared state. This state waits for data from the owner and processes writeback requests before proceeding to the next appropriate state.
8.2	Added the MS_A transient state	It is used to handle the transition when a block in the modified state is being moved to the shared state, and the directory is waiting for a writeback acknowledgment from memory.

Work Distribution

Shawn: implemented and debugged the cache protocol, wrote the report
Victor: implemented and debugged the directory protocol, wrote the report