ECE552 Lab5 report

Zizhen Ji 1006678196, Jason Wei 1007064588

Pre-lab questions:

1. The transient states are necessary because the transitions from one stable state to another is not atomic. The delay during the states, without handling with transient states, will violate the one writer, multiple sharer invariance. For example, if one processor reads a block, while it's waiting for data, if the other processor write to it while it's waiting for the data, the reading processor will read the data when it should be invalidated.

2. The stalls are used in waiting for all actions to be done in a transient state, or the action is dependent on previous actions. This way, the coherence of the cache can be maintained.

3. The dead lock happens when previous and following action depends on each other. It can be handled by having separated network for each type of messages.

4. Put_Ack is the acknowledgement from directory to the cache who issued PUTS or PUTM when cache evicts a block.

5. Directory determines the sender of the data depends on the state of the memory block. It can have following options: a. Sender is memory: when block is invalid in the directory b. Sender is owner cache: when the block is in modified state, meaning one cache has the block c. Sender is a sharer: when block is in shared state, and other cache issued a getS

6. Put_Ack is response from the directory when cache issued putS or putM to evict one block of data. Inv_Ack is response from cache when it receives INV message from the directory or other cache due to a getM.

Methodology Questions:

1. The FSM knows how many Ack it needs to receive when the directory sends the number of ack to the cache controller. The cache controller then update the ack filed in TBE every time it receives an ack. Once the ack is decremented to zero, the cache controller knows it received the last ack.

2. The different virtual channels used by a PUTM request and a Fwd_GetM leads to possible out of order arrival of the two request. If one core P1 send GetM request to the directory, then the owner of the block, P2, sends a PUTM request to the directory. As the GetM arrives at the directory, the directory will set the owner to P1. However, if the PUTM of P2, which was the previous-owner of the block arrives later, the directory will see a PUTM for non-owner block.

3. PUS-Last is the request sent when the core evicting the block is the last sharer of the block. The directory will then have to put the block into invalid state. However, a normal putS will just remove the requestor from the sharer instead of changing the state of the block.

4. In the case of two cores P1 and P2 both in S state, if P1 sends a GetM request while P2 sends a PutS states, although P1 will forward an invalidation request to P2, the P2 can still send a PutS during the transition as the two messages use different channels. So, while the directory changed the block's state to M responding to P1's GetM, the PutS_Last request will eventually arrive while the block is in M state.

5. Because the invalidation requests are only sent to sharers when there is one cache trying to upgrade to M state. If the block is already in M state, considering one-writter-multiple-sharer invariance, there will be no sharer, thus, no invalidation request will be sent.

6. The SI_A state means the block was previously in S state, where the directory owns the data. A Fwd_GetS are only sent to a cache if it's the owner, meaning it has to be in M state. A previously S state cache won't be receiving Fwd_GetS request from other cores.

7. The verification was ran using all testers provided in the lab-handout, with different core configuration and with different instruction count up-to 1000000.

Protocol Modification:

All transitions at the cache side are implemented the same as what is provided in table 8.1 except adding Ifetch event to all the required transitions. The majority of the modifications are to table 8.2, where transient states were added to handle the latency at the directory side during the transitions.

Modified MSI protocol transition table for directory:

State	GetS	GetM	PutS-NotLast	PutS-Last	PutM+data from Owner	PutM+data from NonOwner	Data	Memory_Data	Memory Ack
I	send data to Req, add Req to Sharers/S	send data to Req, set Owner to Req/M	send Put-Ack to Req	send Put-Ack to Req		send Put-Ack to Req			
S	send data to Req, add Req to Sharers	send data to Req, send Inv to Sharers, clear Sharers, set Owner to Req/M	remove Req from Sharers, send Put-Ack to Req	remove Req from Sharers, send Put-Ack to Req/I		remove Req from Sharers, send Put-Ack to Req			
IS_D	stall	stall	Send put_Ack to req	Send put_Ack to req				Add req to sharer, send data/S	
M	Send Fwd-GetS to Owner, add Req and Owner to Sharers, clear Owner/S^D	Send Fwd-GetM to Owner, set Owner to Req	send Put-Ack to Req	send Put-Ack to Req	copy data to memory, clear Owner, send Put-Ack to Req/I	send Put-Ack to Req			
IM_D	stall	stall	Send Put_Ack			Sned Put_Ack		Set req to owner, send data/M	
MI_A	stall	stall	stall	stall		stall	stall		Send writeba

									ck ack, clear owner/I
MS_D	stall	stall	Remove req from sharer, send put Ack	stall	Remove req from sharer, send put Ack	Remove req from sharer, send put Ack			
MS_A	stall	stall	Remove req from sharer, send put Ack	stall	Remove req from sharer, send put Ack	Remove req from sharer, send put Ack			Clear owner/S

Division of work:

Zizhen Ji wrote MSI_cache.sm and report up until the transition table 8.2. Jason Wei wrote MSI_dir.sm and the transition table 8.2 in the report.