Murphi Final Report

timdunn@umich.edu

Approach

I started this assignment by reading the existing IV protocol in Murphi until I fully understood how it worked. Next, I began working from that code base and fully replaced the IV protocol with the `MSI` protocol from the textbook, which assumes network ordering [1].

My initial approach was to repeatedly iterate the following steps: compile, run Murphi, detect unsatisfied invariants, and tweak the existing protocol to avoid the new issue introduced by random-order packet delivery. This process was time-consuming and difficult. After many hours of work and little tangible progress, I eventually gave up on this approach and decided to redesign the whole protocol at once from a higher level.

While spending hours fixing individual coherency protocol flaws, I realized that all my issues were the result of state inconsistencies between the `Dir`ectory and `Proc`essor, stemming from the fact that the `Dir`ectory assumed the `Proc`essor had received packets which it hadn't yet, and that the `Proc`essor had left a transient state when it in fact hadn't. To redesign the protocol, I wrote out the simplified protocol from the textbook, and then slowly worked my way through each possible state, adding `Dir`ectory transient states and acknowledgements from the `Proc`essor to `Dir`ectory wherever necessary. After reworking my high-level design a few times and debugging, the `MSI` protocol worked! From there, extending unordered `MSI` to unordered `MESI` was fairly straightforward.

Protocol

Messages for my `MSI` protocol implementation were sent over four virtual channels. The table below shows the organization of message types into channels such that for each possible sequence of messages and responses, a later response will not be passed along the same channel as an earlier message, which would introduce the possibility of deadlock.

RequestChannel	ForwardChannel	ResponseChannel	OrderChannel
GetS	FwdGetS	Data	DataAck
GetM	FwdGetM, Inv	Data, InvAck	DataAck
PutS		PutAck	PutAckAck
PutM		PutAck	PutAckAck

As the table above shows, my protocol introduces `DataAck` and `PutAckAck` messages, which were not necessary for the protocol described in [1], which assumes an in-order network. Since for this assignment we could not assume that messages would be received in the order in which they were sent, the simplest approach to ensure coherence was to introduce several transient states for both the `Proc`essors and the `Dir`ectory, and use the new acknowledgement message types to regulate movement from one state to another.

Details are shown the the attached full state diagrams. My protocol may involve unnecessary stalling, but correctness (not efficiency) is the stated goal of this project. Note that although

a chain of four messages are sent when a Processor requests permission to read/write a new memory location (e.g. GetS→FwdGetS→Data→DataAck), it is still a 3-hop protocol because the Processor receives new data after the third hop, and the Data message is passed directly from the Owner to the Processor without first travelling through the Directory.

In addition to each Processor storing a value, Processors also needed to keep track of the number of Ack requests they were expecting and had received for certain states. This was done by introducing acks_needed and acks_received variables. Messages were extended beyond containing information regarding the mtype, dst, src, vc, and val, to also include a who and ack. These additions told processors where to send data for FwdGetS/FwdGetM requests and how many acknowledgements the Processor should wait for.

Optimization

The optimization I chose to implement for this project was adding an E state and implementing an MESI protocol. This required the introduction of PutE and EData messages (which were placed in the RequestChannel and ResponseChannel, respectively), as well a few new Processor (E) and Directory (E, E_P, E_A) states. I really learned a lot from this project, but felt that there would be no additional benefit in struggling to ensure the correctness of a more "difficult" optimization, since I had already learned a lot from redesigning protocols and investigating coherency protocol edge cases.

References

[1] Sorin, Daniel J., Mark D. Hill, and David A. Wood. "A primer on memory consistency and cache coherence." Synthesis lectures on computer architecture 6.3 (2011): 1-212.

Full State Machine Descriptions

Deviations from the in-order MSI protocol described in [1] are shown in red.

MSI: Processor

	load	store	evict	FwdGetS	FwdGetM	Inv	PutAck	Data from Dir, ack=0	Data from Dir ack>0	Data from Owner	InvAck, ack>0	InvAck, ack=0	DataAck
I	send GetS to Dir / IS_D	send GetM to Dir / IM_AD											
IS_D						stall		send DataAck to Dir / S		send DataAck to Dir + Owner / S			
IM_AD				stall	stall			send DataAck to Dir / M	- / IM_A	send DataAck to Dir + Owner / M	ack--		
IM_A				stall	stall						ack--	send DataAck to Dir / M	
S		send GetM to Dir / SM_AD	send PutS to Dir / SI_A			send InvAck to Req / I							
SM_AD				stall	stall	send InvAck to Req / IM_AD		send DataAck to Dir / M	- / SM_A	send DataAck to Dir + Owner / M	ack--		
SM_A				stall	stall						ack--	send DataAck to Dir / M	
M			send PutM+Data to Dir / MI_A	send Data to Req + Dir / MS_A	send Data to Req / MI_AA								
MI_A				send Data to Req + Dir / MI_SI_AA	send Data to Req / MI_II_AA		send PutAckAck to Dir / I						
SI_A						send InvAck to Req / II_A	send PutAckAck to Dir / I						
II_A							send PutAckAck to Dir / I						
MI_AA							stall						- / I
MI_II_AA							stall						- / II_A
MS_A						stall	stall						- / S
MI_SI_AA						stall	stall						- / SI_A

MSI: Directory

	GetS	GetM	PutS not last	PutS last	PutM+Data from Owner	PutM+Data from non-Owner	Data	DataAck	PutAckAck
I	send Data to Req, add Req to Sharers / XS_A	send Data to Req, set Owner to Req / XM_A	send PutAck to Req / I_P	send PutAck to Req / I_P		send PutAck to Req, I_P			
S	send Data to Req, add Req to Sharers / XS_A	send Data to Req, send Inv to Sharers, clear Sharers, set Acks, set Owner to Req / XM_A	remove Req from Sharers, send PutAck to Req / S_P	remove Req from Sharers, send PutAck to Req / I_P		remove Req from Sharers, send PutAck to Req / S_P			
M	send FwdGetS to Owner, add Req+Owner to Sharers, clear Owners / MS_DA	send FwdGetM to Owner, set Owner to Req / XM_A	send PutAck to Req / M_P	send PutAck to Req / M_P	store Data, clear Owner, send PutAck to Req / I_P	send PutAck to Req / M_P			
MS_D	stall	stall	remove Req from Sharers, send PutAck to Req / MS_DP	remove Req from Sharers, send PutAck to Req / MS_DP		remove Req from Sharers, send PutAck to Req / MS_DP	store Data, if no sharers / I else / S		
XM_A	stall	stall	stall	stall	stall	stall	stall	- / M	
MS_DP	stall	stall	stall	stall	stall	stall	stall		- / MS_D
XS_A	stall	stall	stall	stall	stall	stall		- / S	
I_P	stall	stall	stall	stall	stall	stall			- / I
S_P	stall	stall	stall	stall	stall	stall			if no sharers / I else / S
M_P	stall	stall	stall	stall	stall	stall			- / M
MS_DA	stall	stall	stall	stall	stall	stall	stall	- / MS_D	

MESI: Processor

	load	store	evict	FwdGetS	FwdGetM	Inv	PutAck	Exclusive Data from Dir	Data from Dir, ack=0	Data from Dir ack>0	Data from Owner	InvAck, ack>0	InvAck, ack=0	DataAck
I	send GetS to Dir / IS_D	send GetM to Dir / IM_AD												
IS_D						stall		send DataAck to Dir / E	send DataAck to Dir / S		send DataAck to Dir + Owner / S			
IM_AD				stall	stall				send DataAck to Dir / M	- / IM_A	send DataAck to Dir + Owner / M	ack--		
IM_A				stall	stall							ack--	send DataAck to Dir / M	
S		send GetM to Dir / SM_AD	send PutS to Dir / SI_A			send InvAck to Req / I								
SM_AD				stall	stall	send InvAck to Req / IM_AD			send DataAck to Dir / M	- / SM_A	send DataAck to Dir + Owner / M	ack--		
SM_A				stall	stall							ack--	send DataAck to Dir / M	
M			send PutM+Data to Dir / MI_A	send Data to Req + Dir / MS_A	send Data to Req / MI_AA									
E		- / M	send PutE to Dir / MI_A	send Data to Req + Dir / MS_A	send Data to Req / MI_AA									
MI_A				send Data to Req + Dir / MI_SI_AA	send Data to Req / MI_II_AA		send PutAckAck to Dir / I							
SI_A						send InvAck to Req / II_A	send PutAckAck to Dir / I							
II_A							send PutAckAck to Dir / I							
MI_AA							stall							- / I
MI_II_AA							stall							- / II_A
MS_A						stall	stall							- / S
MI_SI_AA						stall	stall							- / SI_A

MESI: Directory

	GetS	GetM	PutS not last	PutS last	PutE from Owner	PutE from non-Owner	PutM+Data from Owner	PutM+Data from non-Owner	Data	DataAck	PutAckAck
I	send exclusive Data to Req, set Owner to Req / E_A	send Data to Req, set Owner to Req / XM_A	send PutAck to Req / I_P	send PutAck to Req / I_P		send PutAck to Req, I_P		send PutAck to Req, I_P			
S	send Data to Req, add Req to Sharers / XS_A	send Data to Req, send Inv to Sharers, clear Sharers, set Acks, set Owner to Req / XM_A	remove Req from Sharers, send PutAck to Req / S_P	remove Req from Sharers, send PutAck to Req / I_P		remove Req from Sharers, send PutAck to Req / S_P		remove Req from Sharers, send PutAck to Req / S_P			
M	send FwdGetS to Owner, add Req+Owner to Sharers, clear Owners / MS_DA	send FwdGetM to Owner, set Owner to Req / XM_A	send PutAck to Req / M_P	send PutAck to Req / M_P		send PutAck to Req / M_P	store Data, clear Owner, send PutAck to Req / I_P	send PutAck to Req / M_P			
E	send FwdGetS to Owner, add Req+Owner to Sharers, clear Owners / MS_DA	send FwdGetM to Owner, set Owner to Req / XM_A	send PutAck to Req / E_P	send PutAck to Req / E_P	clear Owner, send PutAck to Req / I_P	send PutAck to Req / E_P	store Data, clear Owner, send PutAck to Req / I_P	send PutAck to Req / E_P			
MS_D	stall	stall	remove Req from Sharers, send PutAck to Req / MS_DP	remove Req from Sharers, send PutAck to Req / MS_DP		remove Req from Sharers, send PutAck to Req / MS_DP		remove Req from Sharers, send PutAck to Req / MS_DP	store Data, if no sharers / I else / S		
XM_A	stall	stall	stall	stall	stall	stall	stall	stall	stall	- / M	
E_A	stall	stall	stall	stall	stall	stall	stall	stall	stall	- / E	
MS_DP	stall	stall	stall	stall	stall	stall	stall	stall	stall		- / MS_D
XS_A	stall	stall	stall	stall	stall	stall	stall	stall		- / S	
I_P	stall	stall	stall	stall	stall	stall	stall	stall			- / I
S_P	stall	stall	stall	stall	stall	stall	stall	stall			if no sharers / I else / S
M_P	stall	stall	stall	stall	stall	stall	stall	stall			- / M
E_P	stall	stall	stall	stall	stall	stall	stall	stall			- / E
MS_DA	stall	stall	stall	stall	stall	stall	stall	stall	stall	- / MS_D	