Asynchronous Systems Cache Coherence - Project Design

Karthik Reddy - 109721778

Project Description:

Comprehensive comparison of cache coherence protocols in DistAlgo[1].
Measure the performance metrics of various cache coherence protocols on benchmarks and compare with other protocols.

Project Design:

Consider a distributed shared memory system. Each process(think Processor) has its own local cache(think L1,L2 cache) of data, cached from a large memory database(think main memory). We demonstrate and compare the performance of various cache coherence protocols, which are used to make these local caches of memory, coherent.

There are 3 main classes. Processor, MESI_cache_controller and MESI_directory_controller.

Processor: Each Processor runs as a separate process, each having its own trace file. Each Processor process communicates with its own Cache Controller. Cache Controller runs as a separate process. Processor and Cache controller communicate through messages. Processor reads the trace file and issues load/store instructions to the Cache Controller process. Cache Controller once it executes the instruction acknowledges the Processor which sends the next instruction for the cache to execute.

MESI_cache_controller: This is the Cache Controller process associated with each Processor object. Cache controller is internally represented as a state machine. There are two types of transitions possible: local and remote. Remote transitions are in response to the other caches or the directory controller. Each cache controller has a reference to the Directory Controller. The cache controller communicates(unicast message) with the directory controller as per the MESI protocol. The cache controller also receives messages from the directory controller.

MESI_directory_controller: This is the Directory Controller object. Receives messages from cache controllers. Queues up these requests and according to MESI protocol, performs actions. Internally represented as a state machine. Stores “state”, “owner” and a list of “sharers” for every cache line. Uses this information to track the status of the cache and the data they cache.

My MESI implementation in DistAlgo:

Please refer to “mesi_protocol.txt“ for information on the implementation. It is an extracted Pydoc file which explains briefly the implementation, various API within the code and instructions to execute the driver program.

Currently, the driver method(used to run and measure the performance metrics), Processor, Cache Controller, Directory Controller have been implemented, albeit partially. Their basic setup and communication between the various components has been done. MESI protocol has been understood including various race conditions which might occur and how to break them. Most of this logic has been obtained from [2]. A large number of API have been declared but not all of them have been defined yet.

Brief description of MESI protocol:

Source code at [3].

Every cache line is marked with one of the four following states (coded in two additional bits):

Modified
: The cache line is present only in the current cache, and is dirty; it has been modified from the value in main memory. The cache is required to write the data back to main memory at some time in the future, before permitting any other read of the (no longer valid) main memory state. The write-back changes the line to the Exclusive state.

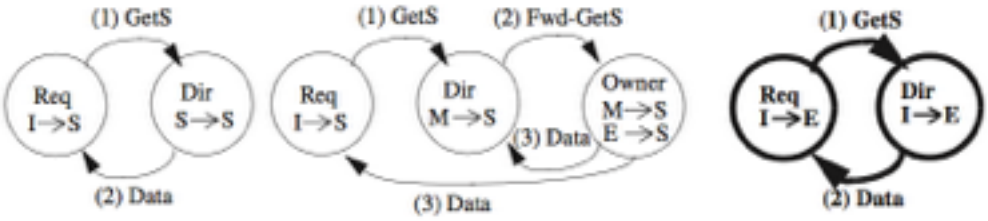
Exclusive
: The cache line is present only in the current cache, but is clean; it matches main memory. It may be changed to the Shared state at any time, in response to a read request. Alternatively, it may be changed to the Modified state when writing to it.

Shared
: Indicates that this cache line may be stored in other caches of the machine and is clean; it matches the main memory. The line may be discarded (changed to the Invalid state) at any time.

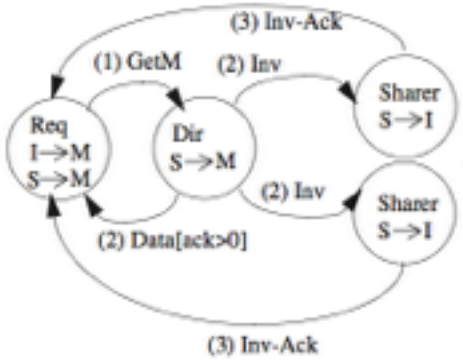
Invalid
: Indicates that this cache line is invalid (unused).

More information on the MESI is provided by [2].

Following are the state diagrams and state machine(cache and directory controller) transitions of the MESI protocol.

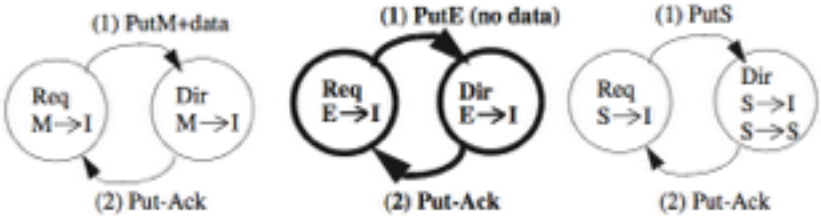


Transitions from I to S. **Transition from I to E.**



The only sharer might be the requestor, in which case no Invalidation messages are sent and the Data message from the Dir to Req has an AckCount of zero.

Transitions from I or S to M. Transition from E to M is silent.



Transition from M or E or S to I

TABLE 8.4: MESI Directory Protocol—Directory Controller

	GetS	GetM	PutS-NotLast	PutS-Last	PutM+data from Owner	PutM from Non-Owner	PutE (no data) from Owner	PutE from Non-Owner	Data
I	send Exclusive data to Req, set Owner to Req/E	send data to Req, set Owner to Req/M	send Put-Ack to Req	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		**remove Req from Sharers, send Put-Ack to Req**	
E	**forward GetS to Owner, make Owner sharer, add Req to Sharers, clear Owner/S^D**	**forward GetM to Owner, set Owner to Req/M**	**send Put-Ack to Req**	**send Put-Ack to Req**	**copy data to mem, send Put-Ack to Req, clear Owner/I**	**send Put-Ack to Req**	**send Put-Ack to Req, clear Owner/I**	**send Put-Ack to Req**	
M	forward GetS to Owner, make Owner sharer, add Req to Sharers, clear Owner/S^D	forward GetM to owner, set Owner to Req	send Put-Ack to Req	send Put-Ack to Req	copy data to mem, send Put-Ack to Req, clear Owner/I	send Put-Ack to Req		**send Put-Ack to Req**	
S^D	stall	stall	remove Req from Sharers, send Put-Ack to Req	remove Req from Sharers, send Put-Ack to Req		remove Req from Sharers, send Put-Ack to Req		**remove Req from Sharers, send Put-Ack to Req**	copy data to LLC/mem/S

TABLE 8.3: MESI Directory Protocol—Cache Controller

	load	store	replacement	Fwd-GetS	Fwd-GetM	Inv	Put-Ack	Exclusive data from Dir	Data from Dir (ack=0)	Data from Dir (ack>0)	Data from Owner	Inv-Ack	Last-Inv-Ack
I	send GetS to Dir/IS^D	send GetM to Dir/IM^{AD}											
IS^D	stall	stall	stall			stall		-/E	-/S		-/S		
IM^{AD}	stall	stall	stall	stall	stall				-/M	-/IM^A	-/M	ack--	
IM^A	stall	stall	stall	stall	stall							ack--	-/M
S	hit	send GetM to Dir/SMAD	send PutS to Dir/SI^A			send Inv-Ack to Req/I							
SM^{AD}	hit	stall	stall	stall	stall	send Inv-Ack to Req/IM^{AD}			-/M	-/SM^A	-/M	ack--	
SM^A	hit	stall	stall	stall	stall							ack--	-/M
M	hit	hit	send PutM+data to Dir/MI^A	send data to Req and Dir/S	send data to Req/I								
E	**hit**	**hit/M**	**send PutE (no data) to Dir/EI^A**	**send data to Req and Dir/S**	**send data to Req/I**								
MI^A	stall	stall	stall	send data to Req and Dir/SI^A	send data to Req/II^A		-/I						
EI^A	**stall**	**stall**	**stall**	**send data to Req and Dir/SI^A**	**send data to Req/II^A**		**-/I**						
SI^A	stall	stall	stall			send Inv-Ack to Req/II^A	-/I						
II^A	stall	stall	stall				-/I						

Team Member's role

Parag Gupta : MSI
Karthik Reddy : MESI
Paul Mathew : MI, feasibility Token Coherence
Amit Khandelwal : MOESI
Garima Gehlot : MOSI

References

[1] Yanhong A. Liu, Bo Lin, and Scott Stoller. "DistAlgo Language Description". https://github.com/DistAlgo/distalgo
[2] Martin, Milo MK, Mark D. Hill, and David Wood. "Token coherence: Decoupling performance and correctness." Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on. IEEE, 2003.
[3] karthik reddy. https://github.com/karthikbox/cache_coherence/tree/mesi_protocol