ECE 4100

Soham Gadgil

Project 3 Analysis

PART 1

Experiment 1:

Protocol	Runtime	Cache Misses	Cache Accesses	Silent Upgrades	$ - $ Transfers
MESI	317	7	12	0	4
MOSI	217	7	12	0	5
MOESIF	217	7	12	0	5

For this experiment, MOSI protocol will work best. The number of instructions in the trace is small so the runtime was small. The addition of the Owned state allows for dirty sharing of data so the dirty cache block can be moved around without updating memory which reduces run time. The MOESIF protocol had the same runtime which reflects that the addition of the Forwarder state did not have much impact on run time. However, addition of the Forwarder state introduced more transition states, which is why MOSI protocol was picked.

Experiment 2:

Protocol	Runtime	Cache Misses	Cache Accesses	Silent Upgrades	$ - $ Transfers
MESI	2267	30	104	1	8
MOSI	1167	30	104	0	19
MOESIF	683	34	104	1	28

For this experiment, MOESIF protocol will work best since the run time for MOESIF is the smallest and the cache to cache transfers are maximum. This can be explained because of the presence of the Owned and Forwarder states. Both these states are cache to cache transfer agents, so if a block in the F or the O state snoops a GET request, it can send data to the bus without having to get it from memory, which reduces runtime. As a result, there are more cache to cache transfers in MOESIF as compared to MESI and MOSI. Looking at the traces, we see that there are a lot of reads happening with a few writes in between and this would benefit from the Owned state since it allows one cache to have dirty data and the other blocks can continue to be in the Shared state.

Experiment 3:

Protocol	Runtime	Cache Misses	Cache Accesses	Silent Upgrades	$ - $ Transfers
MESI	2607	48	200	8	23
MOSI	3723	56	200	0	20
MOESIF	1425	48	200	8	35

For this experiment, MOESIF protocol will work best since the run time is the smallest, the number of cache misses are smallest, and the number of cache to cache transfers are maximum. Looking at the trace, it is observed that the addresses are varied. Each processor writes and reads different addresses. It can be seen from the table that MOSI does not do well in this scenario which can be because the Owned state Is not very useful. The only way to get to the owned state is through the modified state and because of different addresses, not a lot of GETS messages will be snooped by modified blocks. The exclusive state works well since there are different addresses, a block can remain in the exclusive state for a long time and this reduces the number of invalidation messages on the bus. However, MOESIF works the best since it's statistics are the best.

Experiment 4:

Protocol	Runtime	Cache Misses	Cache Accesses	Silent Upgrades	$ - $ Transfers
MESI	1447	19	60	3	5
MOSI	1869	29	60	0	11
MOESIF	551	19	60	3	14

For this experiment, MOESIF protocol will work best since the run time is the smallest, the number of cache misses are smallest, and the number of cache to cache transfers are maximum. For these traces, it can be observed that one processor (p0) does a lot of writes and the other processors do all reads. This seems to be a producer consumer scenario, where p0 is the producer and other processors are consumers. Thus, going into the Modified state and then into the Owned state works best since it allows dirty sharing of data. Also going into the Forwarded state from the Exclusive state helps in increasing cache to cache transfers. Thus, the MOESIF protocol works best.

Experiment 5:

Protocol	Runtime	Cache Misses	Cache Accesses	Silent Upgrades	$ - $ Transfers
MESI	1561	21	37	0	6
MOSI	1261	21	37	0	9
MOESIF	461	21	37	0	17

For this experiment, MOESIF protocol will work best since the run time is the smallest, the number of cache misses are smallest, and the number of cache to cache transfers are maximum. In this trace, there are very few instructions for each of the 8 processors. Most of the operations are reads with writes in between. Having the Owned state is useful here since once a block gets into the Owned state, it can respond to GETS messages while the other blocks remain in the Shared state. Thus, the MOESIF protocol works best.

Experiment 6:

Protocol	Runtime	Cache Misses	Cache Accesses	Silent Upgrades	$ - $ Transfers
MESI	4925	62	747	25	15
MOSI	6975	87	747	0	20
MOESIF	3125	62	747	25	33

For this experiment, MOESIF protocol will work best since the run time is the smallest, the number of cache misses are smallest, and the number of cache to cache transfers are maximum. In this trace, it can be observed that there are many reads with very few writes in between. AS a result, the Owned state is helpful since the Owner block can do dirty sharing while the other blocks remain in the Shared state. There are also a lot of reads happening of the same address which is helped by the Exclusive and Forwarder states since it reduces memory accesses. This can be seen from the table that MESI has a lower runtime than MOSI. However, MOESIF has the lowest runtime so it works the best.

Experiment 7:

Protocol	Runtime	Cache Misses	Cache Accesses	Silent Upgrades	$ - $ Transfers
MESI	3993	55	952	24	17
MOSI	5359	79	952	0	28
MOESIF	2909	55	952	24	28

For this experiment, MOESIF protocol will work best since the run time is the smallest, the number of cache misses are smallest, and the number of cache to cache transfers are maximum. In this trace, it can be observed that there are a lot of processors and the stream is either a burst of reads or alternating reads and writes to the same address. This will be helped by the Owned state since a Modified block can go to the Owned state when it snoops a GETS. Thus, MOESIF has the lowest run time and works the best.

Experiment 8:

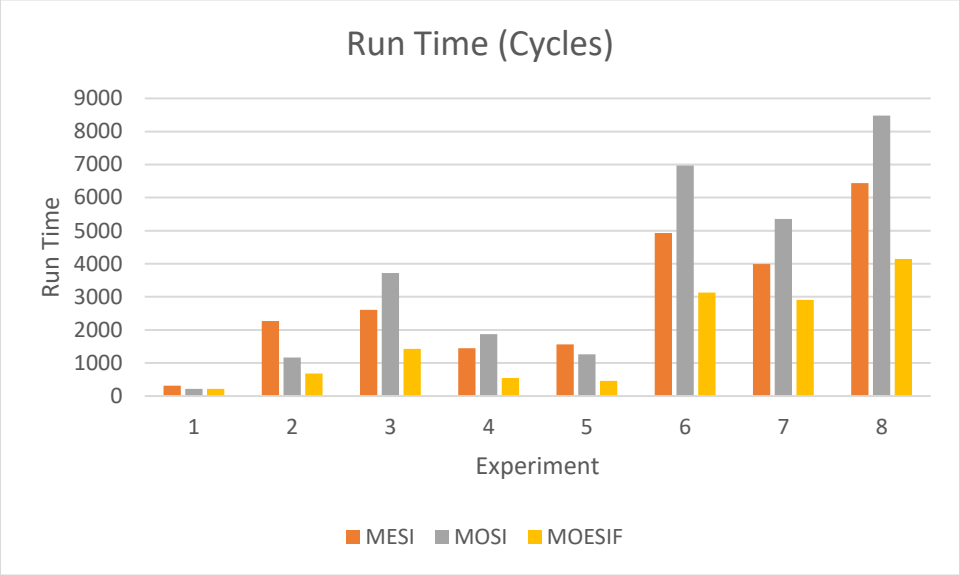
Protocol	Runtime	Cache Misses	Cache Accesses	Silent Upgrades	$ - $ Transfers
MESI	6441	92	800	19	30
MOSI	8477	110	800	0	28
MOESIF	4141	92	800	19	53

For this experiment, MOESIF protocol will work best since the run time is the smallest, the number of cache misses are smallest, and the number of cache to cache transfers are maximum. In this trace, it can be observed that there are many processors but there are a lot of reads very few writes in between. The Owned state is not very helpful in this case since there would not be a lot of opportunities to go to the Modified state and you can only get to the Owned state through Modified. The Exclusive state is helpful since it reduces invalidation requests and can silently upgrade. This can be seen from the table in that the runtime of MESI is less than that of MOSI. However, MOESIF has the lowest run time and works the best.

PART 2

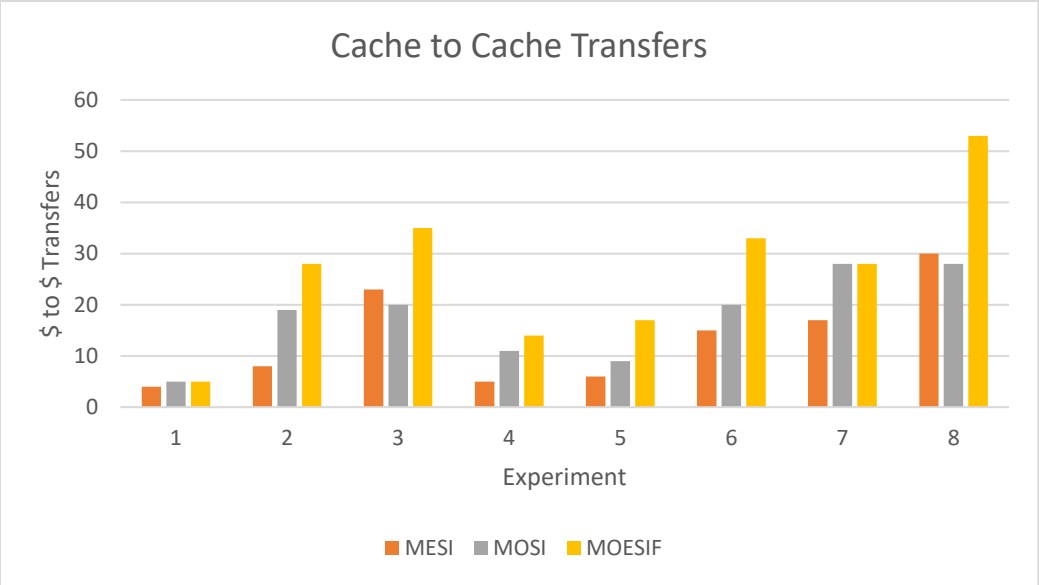
I would use the MOESIF protocol for the system.

Runtime:



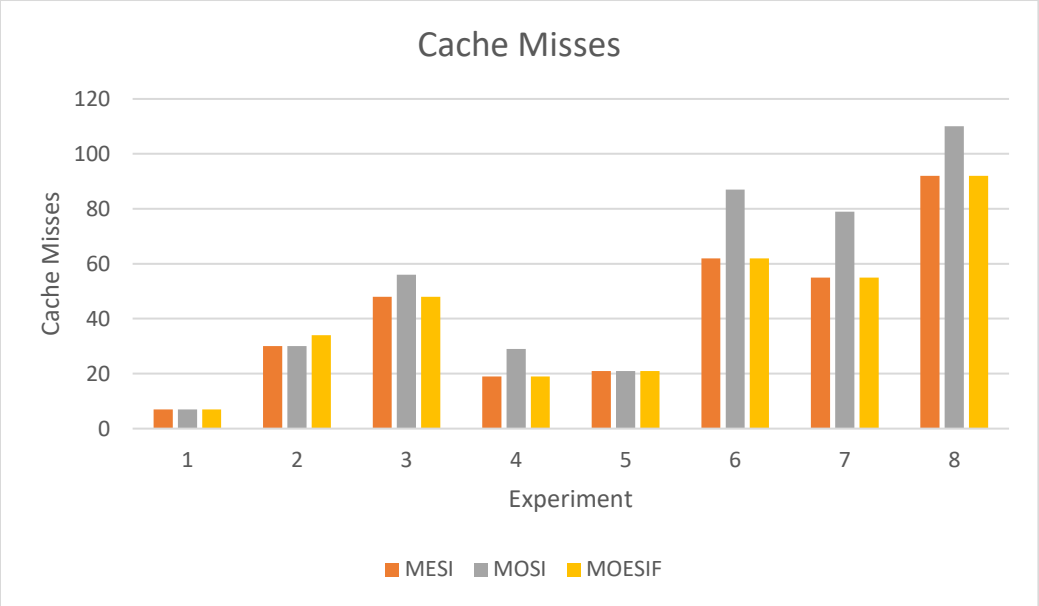
As it can be seen in the above graph, except for the first experiment, in all other cases, the MOESIF protocol has a lower runtime than MESI and MOSI which is mostly because of the presence of the additional Owned, Exclusive, and Forwarder states. Although adding more states increases complexity of implementation and increases the number of transient states, it reduces runtime by a lot which cannot be ignored. Since performance is the first thing architects look to improve, MOESIF seems to be the best option regardless of the number of processors. Also, we can see that in experiments with a smaller instruction count and less number of processors (1, 2, 5) MOSI has a smaller run time than MESI which can be because the owned state in MOSI really helps in these scenarios. For experiments with a larger instruction count and more processors (6, 7, 8), it is observed that MESI has a lower runtime than MOSI which can be because of the presence of Exclusive state which allows for silent upgrades and has a larger impact than the Owned state in MOSI.

Cache to Cache Transfers:



In most experiments, as it can be seen in the graph above, MOSI has more cache to cache transfers than MESI. This is because the Owned state in MOSI is a cache to cache transfer agent while MESI does not have such a state. However, MOESIF has the highest cache to cache transfers in all experiments, which means that the memory is not required to be accessed that much.

Cache Misses:



With regards to cache misses, MOSI has the highest number of cache misses which can be explained due to the absence of an Exclusive state for silent upgrades. MESI and MOESIF have the same number of cache misses for almost all experiments which can be because the additional O and F states in MOESIF do not affect the cache misses.

Conclusion:

After considering all the parameters mentioned above, I would use the MOESIF Protocol. It has lower runtime, less cache misses, and more cache to cache transfers in all experiments without depending a lot on the number of processors. The presence of the Owned state allows dirty sharing of data since a modified block can be moved to other caches without updating memory. The presence of the Exclusive state allows for silent upgrades to Modified state and reduces invalidations since only one cache has the copy. As a result, MOESIF gets the best of both worlds and I would use it to get the best performance even though it has a few drawbacks like more transient states. This would be regardless of the number of processors since the observations above suggest that MOESIF has lower runtime for all configurations.

PART 3

Limitations:

The main limitation of the simulator is the cache for each processor has infinite size with a single cycle look up time. This means that there would never be any evictions from the cache since there is always space for the new block. This is not realistic since in real systems, the cache has a limited size so when it is full, evictions must take place meaning the memory has to be accessed more often resulting in more number of cycles for execution. Thus, having a fixed size cache would be needed to make it more realistic.

The bus is modeled as an atomic bus which means that a query on the bus will prevent other requests till a DATA response. This makes the simulator simpler. However, in real systems, the buses are mostly non-atomic which makes the system more complex because more transient states are needed. Thus, the system should have a non-atomic bus to make it more realistic. The only requests supported by the bus are the ones in which someone replies with data like GETS and GETM. In real systems, there can be messages which do not have any replies, so the system should support such messages to make it more realistic.