ECE 4100
Project 3 Report (original E implementation)
By Sho Ko

Analysis

After I match the validation logs, I run each of 8 experiments with 4 protocols (MSI, MESI, MOSI, MOESIF). There are total 32 runs. The parameter that defines a better protocol is the run time cycles, because what people only care about run time. However, run time is associated with cache misses, silent upgrades, and $ to $ transfers. The less the cache misses, the smaller the run time. The more the silent upgrades, the smaller the run time. The more the $ to $ transfers, the smaller the run time. I put run time, cache misses, silent upgrades, and $ to $ transfers of using different protocols for each experiment in the form of tables.

We have 3 basic states MSI. In addition, we experiment with state E, O, and F.

State E means the block is the only clean copy across all PEs, so when it sees a GETM, it can be silently upgraded to state M without being broadcasted on the bus. It reduces bus traffic.

State O means the block is the only dirty copy across all PEs, with other possible blocks in S state. Therefore, when it sees a GETS, it can directly forward the data and avoid going back to memory. It functions as $ to $ transfer agent and reduces miss penalty.

State F means the block is a clean copy who is responsible for forwarding the data. There may be other clean copies across all PEs. Therefore, when it sees a GETS, it can directly forward the data and avoid going back to memory. It functions as $ to $ transfer agent and reduces miss penalty.

However, having all addition states like E, O, and F will not necessarily enhance the performance of the chip multiprocessor. If we don't need an additional state, we better get rid of it because introducing an additional state will increase the overheads to enforce the coherence protocol.

Experiment1

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	317	7	0	4
MOSI	217	7	0	5
MESI	317	7	0	4
MOESIF	217	7	0	5

For experiment1, because MOSI and MOESIF have 1 more $ to $ transfers than MSI and MESI, O is useful for this experiment. Because MOSI and MOESIF have the same $ to $ transfers, F is not useful for this experiment. In addition, because there are no silent upgrades in any of the 4 protocols, E is not useful for this experiment. Therefore, we choose MOSI, which achieves the best run time by including O and excluding unnecessary E and F to avoid overheads.

Experiment2

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	2367	30	0	7
MOSI	1167	30	0	19
MESI	2267	30	1	8
MOESIF	683	34	1	28

For experiment2, because MOSI has significantly more $ to $ transfers than MSI and MESI, O is useful for this experiment. Because MOESIF has significantly more $ to $ transfers than MOSI, F is also useful for this experiment. Therefore, both $ to $ transfer agents are useful for this experiment. In addition, MESI and MOESIF have 1 more silent upgrades than MSI and MOSI, so E is can be helpful but doesn't make too much of an impact. As a result, we choose MOESIF, which achieves the best run time by including O and F.

Experiment3

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	3723	56	0	20
MOSI	3723	56	0	20
MESI	2607	48	8	23
MOESIF	1425	48	8	35

For experiment3, because MOSI has similar $ to $ transfers as MSI and MESI, O is not useful for this experiment. Because MOESIF has significantly more $ to $ transfers than MSI, MOSI, and MESI, F is useful for this experiment. In addition, because both MSI and MOSI have 0 silent upgrades while both MESI and MOESIF have 8 silent upgrades, E is useful for this experiment. Therefore, we choose MOESIF, which achieves the smallest run time by including E and F.

Experiment4

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	2265	27	0	5
MOSI	1869	29	0	11
MESI	1447	19	3	5
MOESIF	551	19	3	14

For experiment4, because MOSI has more $ to $ transfers than MSI and MESI, O is useful for this experiment. Because MOESIF has more $ to $ transfers than MOSI, F is also useful for this experiment. Therefore, both $ to $ transfer agents are useful for this experiment. In addition, because both MSI and MOSI have 0 silent upgrades while both MESI and MOESIF have 3 silent upgrades, E is also useful for this experiment. Therefore, we choose MOESIF, which achieves the smallest run time by including E, O, and F.

Experiment5

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	1661	21	0	5
MOSI	1261	21	0	9
MESI	1561	21	0	6
MOESIF	461	21	0	17

For experiment5, because MOSI has more $ to $ transfers than MSI and MESI, O is useful for this experiment. Because MOESIF has more $ to $ transfers than MOSI, F is also useful for this experiment. Therefore, both $ to $ transfer agents are useful for this experiment. In addition, because all 4 protocols have 0 silent upgrades, E is not useful for this experiment. Therefore, we choose MOESIF, which achieves the smallest run time by including O and F.

Experiment6

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	7775	87	0	12
MOSI	6975	87	0	20
MESI	4925	62	25	15
MOESIF	3125	62	25	33

For experiment6, because MOSI has significantly more $ to $ transfers than MSI and MESI, O is useful for this experiment. Because MOESIF has significantly more $ to $ transfers than MOSI, F is also useful for this experiment. Therefore, both $ to $ transfer agents are useful for this experiment. In addition, because both MSI and MOSI have 0 silent upgrades while both MESI and MOESIF have 25 silent upgrades, E is also useful for this experiment. Therefore, we choose MOESIF, which achieves the smallest run time by including E, O, and F.

Experiment7

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	6459	79	0	17
MOSI	5359	79	0	28
MESI	3993	55	24	17
MOESIF	2909	55	24	28

For experiment7, because MOSI and MOESIF has significantly more $ to $ transfers than MSI and MESI, O is useful for this experiment. Because MOESIF has the same $ to $ transfers as MOSI, F is not useful for this experiment. In addition, because both MSI and MOSI have 0 silent upgrades while both MESI and MOESIF have 24 silent upgrades, E is also useful for this experiment. Therefore, we choose MOESIF, which achieves the smallest run time by including E and O.

Experiment8

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	9477	110	0	18
MOSI	8477	110	0	28
MESI	6441	92	19	30
MOESIF	4141	92	19	53

For experiment8, because MOSI has similar $ to $ transfers as MESI, O is not useful for this experiment. Because MOESIF has significantly more $ to $ transfers than MSI, MOSI, and MESI, F is useful for this experiment. In addition, because both MSI and MOSI have 0 silent upgrades while both MESI and MOESIF have 19 silent upgrades, E is useful for this experiment. Therefore, we choose MOESIF, which achieves the smallest run time by including E and F.

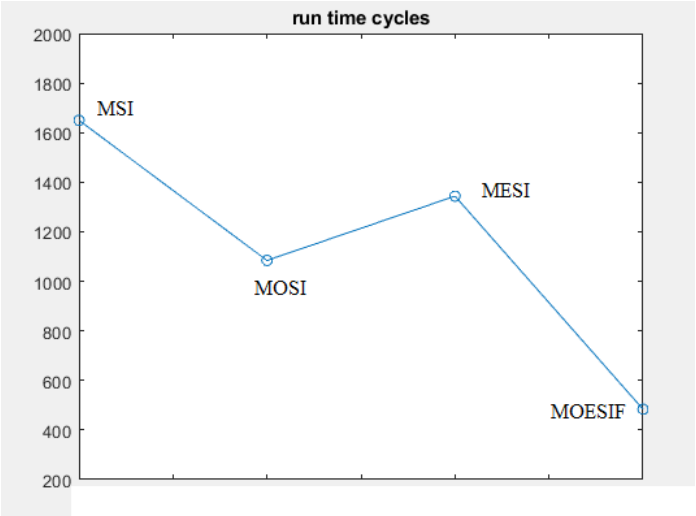
Then we consider the best protocol for 4-core, 8-core, and 16-core configurations. We also consider the best protocol regardless of how many cores. Suppose each experiment is equally important. I average the run time, cache misses, silent upgrades, and $ to $ transfers for each protocol in each scenario. Then I compare the speedup of each protocol over the most basic MI, which takes 1748.8 cycles in average.

4-core

Experiment 1, 2, 4 use 4-core. The following is the average for each protocol.

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	1649.7	21.3	0	5.3
MOSI	1084.3	22	0	11.7
MESI	1343.7	18.7	1.3	5.7
MOESIF	483.7	20	1.3	15.7

For 4-core, MOESIF outperforms MSI, MOSI, and MESI in all parameters (run time cycles, cache misses, silent upgrades, and $ to $ transfers). The speedup of MOESIF over MSI is 3.41, over MOSI is 2.24, over MESI is 2.78.

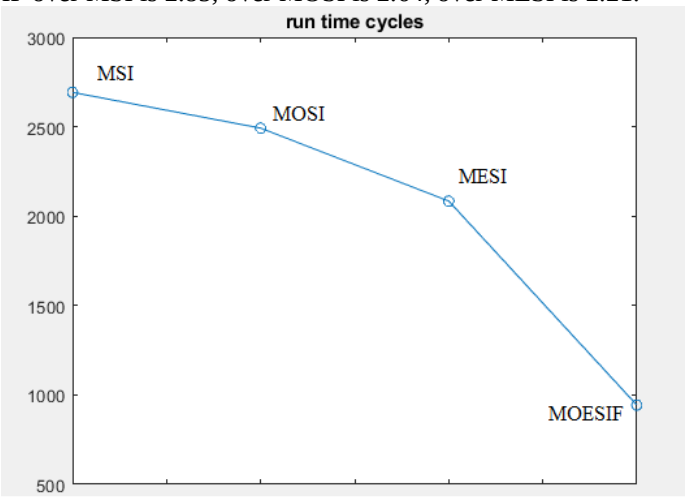


8-core

Experiment 3, 5 use 8-core. The following is the average for each protocol.

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	2692	38.5	0	12.5
MOSI	2492	38.5	0	14.5
MESI	2084	34.5	4	14.5
MOESIF	943	34.5	4	26

For 8-core, MOESIF outperforms MSI, MOSI, and MESI in all parameters (run time cycles, cache misses, silent upgrades, and $ to $ transfers). The speedup of MOESIF over MSI is 2.85, over MOSI is 2.64, over MESI is 2.21.

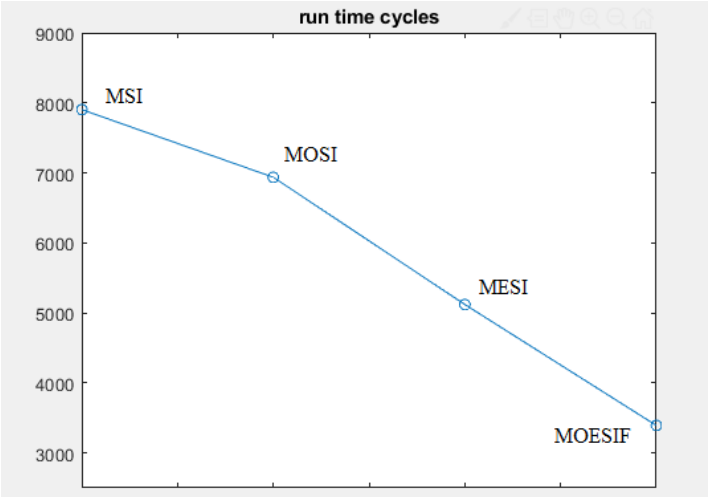


16-core

Experiment 6, 7, 8 use 16-core. The following is the average for each protocol.

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	7903.7	92	0	15.7
MOSI	6937	92	0	25.3
MESI	5119.7	69.7	22.7	20.7
MOESIF	3391.7	69.7	22.7	38

For 4-core, MOESIF outperforms MSI, MOSI, and MESI in all parameters (run time cycles, cache misses, silent upgrades, and $ to $ transfers). The speedup of MOESIF over MSI is 2.33, over MOSI is 2.05, over MESI is 1.51.

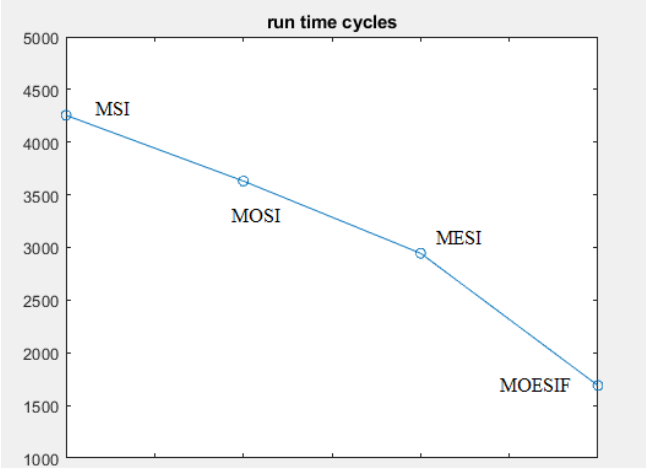


Best Protocol Overall

All 8 experiments are involved. The following is the average for each protocol.

	run time cycles	cache misses	silent upgrades	$ to $ transfers
MSI	4255.5	52.1	0	11
MOSI	3631	52.4	0	17.5
MESI	2944.8	41.8	10	13.5
MOESIF	1689	42.3	10	26.6

For 4-core, MOESIF outperforms MSI, MOSI, and MESI in all parameters (run time cycles, cache misses, silent upgrades, and $ to $ transfers). The speedup of MOESIF over MSI is 2.52, over MOSI is 2.15, over MESI is 1.74.



In conclusion, MOESIF outperforms other 3 protocols in 4-core, 8-core, 16-core, and overall, mainly because MOESIF includes E, O, and F states. E can be silently upgraded to M without being broadcasted on the bus. It can alleviate bus traffic. O and F are $ to $ transfer agents which perform transferring of data from one cache to the other and avoid the miss penalty of going back to the memory to get the data. MOESIF utilizes the 3 additional states to save many cycles.

Limitations of the Simulation

After finishing the project and running all the experiments, I realize that there are a few limitations in this project, and it can be improved in several ways.

1. The project assumes an atomic bus. Once a query is put on the bus, it will not respond to any other queries until it sees the data after 100 cycles. This model is a little too simple. In reality, however, the bus usually can process multiple queries concurrently, and the number of cycles for the bus to respond to a new query will depend on the current bus traffic. The more queries on the bus, the more cycles the bus will take to respond to a new query because of overheads. We can make the system more realistic by simulating a more complex bus with multiple queries on it concurrently. If the bus is in saturation and more queries come, we can put new queries on an arbitration queue until one query on the bus get its data.
2. The project assumes each core has one level of fully associative cache which has infinite size. It has a single lookup cycle, and we don't simulate cache block evictions and writebacks. In reality, however, the private cache of each core cannot have infinite size. Also, it doesn't have to be a fully associative cache. We can improve this project by integrating some parts of project1 into it. We can set the cache size C, block size B, and blocks per set S to be variables. We can also simulate that when a dirty block is evicted from cache, we increase the number of writebacks by 1. After we correctly implement this project, we can experiment with it to find the best cache configuration for each core (best is smallest run time cycles).
3. The project assumes if there is a miss, it takes 100 cycles to go to the memory and get the data. If there is a $ to $ transfer, it will see the data the next cycle. In reality, however, the number of cycles it takes for $ to $ transfer depends on the physical location of different cores. For example, Intel makes a mesh to allow data to be transferred from $ to $ in both directions. We can improve this project by setting the $ to $ transfer cycles to be a variable. One way to do this is that if two caches are x unit distance apart, the $ to $ transfer cycles is x. Therefore, it will take less cycles to transfer data between neighbor cores than to transfer data between cores far apart to each other.