Formal Verification of the MESI Cache Coherency Protocol using SPIN

Luke Marzen

ljmarzen@iastate.edu

December 13, 2023

1 Introduction

Cache coherency is critical for maintaining a up-to-date view of memory between multiple processors. To achieve coherency, a protocol is employed. MESI is a widely-used cache coherency protocol. The aim of this project is to verify the MESI cache coherency protocol by employing *Model Checking*. To this end, a parameterizable cache model is constructed in Promela to be verified in SPIN.

1.1 Overview

Background information is reviewed in Section 2. Section 3 begins with an overview of the construction of the model, then discusses the validation efforts in Subsection 3.1. Verification of the model is discussed in Subsection 3.2. Results are presented in Section 4 and conclusions are presented in Section 5.

2 Background

Processors have several levels of memory storage which are organized according to their proximity to the CPU and access-time. This concept is referred

to as the *Memory Hierarchy*. Memory hierarchies are motivated by practical constraints that prevent us from having indefinitely large memory with immediate access.

2.1 Cache Coherency

Modern multi-processor systems consist of multiple CPUs, each with their own registers and cache(s). Figure 1 illustrates this relationship. To maintain correct program execution for programs utilizing shared memory it is crucial that a *coherent*, up-to-date, view of memory is maintained between all CPUs. A *cache coherency protocol* describes the permitted transactions and how those changes are propagated to other caches in order to ensure the desired behavior is preserved. In section 3.2 we will define this more formally.

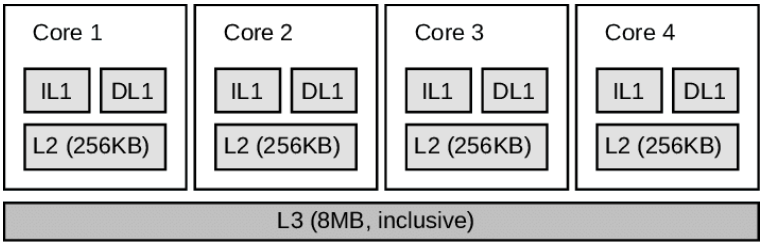

Figure 1: Cache architecture of the Intel Core i7 4790 processor. (Nakamoto 2018)

2.2 MESI

MESI is a popular cache coherency protocol originally proposed by (Papamarcos and Patel 1984). The name itself is an acronym for the four states that a cache block can be assigned: *Modified*, *Exclusive*, *Shared*, and *Invalid*. The MESI protocol can be defined by a finite-automata shown in Figure 2. Each private cache is connected to a *caching-agent* (i.e. a processor) as well as a shared bus which is used to communicate state transitions.

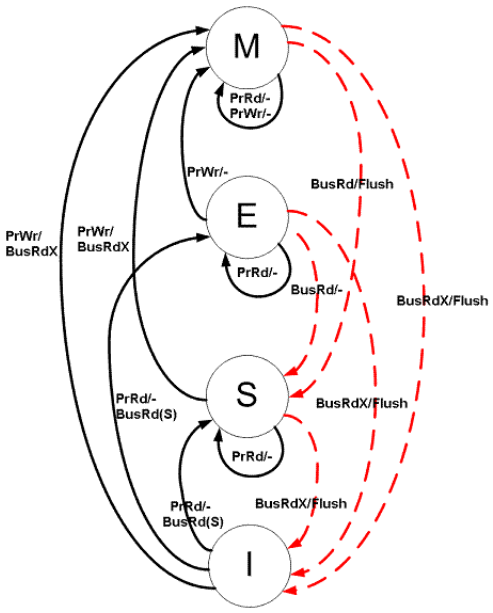

Figure 2: State transition diagram for Illinois MESI protocol. Transitions: Processor initiated transactions (black), Bus initiated transaction (red). (Culler, Singh, and Gupta 1998)

2.3 Snooping

Snooping is the first of two competing techniques used by cache coherency protocols to achieve communication of cache transactions and propagate state changes. This solution employs a shared interconnect or bus which cache transactions are broadcasted on to all other caches. Each caches is constantly monitoring or *snooping* this shared bus and updated the state of its cache accordingly. Snooping protocols are typically only reserved for processors with small core counts since the shared bus quickly becomes a bottleneck with the increased traffic generated by additional processors. MESI is a *snoopy* coherence protocol as it is implemented using a shared bus.

2.4 Directory-Based

Directory-based implementations manage coherence through a centralized directory that stores the state and location of each cache block. The directory

coordinates state transitions when a processor performs a memory operation. Directory-based implementations avoid the bottleneck of the shared bus approach, making it more scalable and ideal for processors with larger core counts.

3 Model

For the purposes of verification, a model for the MESI cache coherency protocol was constructed in SPIN/Promela version 6.5.2 (Holzmann 1997). SPIN was selected since it utilizes Promela, which is a particularly strong language when it comes to expressing current algorithms. The model has been parameterized in terms of number of processors, cache size, and memory size. The caches are implemented to be symmetric. This is assumption was made two primary reasons. Firstly, it is common in real world systems for caches across on the same level to be symmetric. And second, this assumption greatly simplifies the model creation in Promela as well as the expression of validation and verification LTL statements as we will see in sections 3.1 and 3.2. The caches are implemented as direct-mapped caches with only a single level of caches. The depth of the memory hierarchy does not matter for the sake of verifying the MESI protocol since there are no differences in the algorithm when operating between an L1 and L2 cache or between an L2 and L3 or main memory. The block size is 1bit since the data itself is irrelevant to the verification of MESI and increasing the block size will result in statespace explosion.

3.1 Validation

Before we can assert that our verification is correct it is important to take steps to validate that the model accurately represents what we are attempting to verify. To this end, LTL statements have been written which aim to prove that there exists a trace in which various memory system operations and bus operations can occur. The statements are all written as negations, so that a violation of the LTL provides a counter trace that proves the property that we are aiming to show.

The first property to validate is that memory can eventually be written to. It is difficult to express this statement in Promela for all memory addresses, so we leverage the fact that the model is symmetric and thus it suffices to show that this property holds for only a single memory address. We negate the statement so that SPIN will return a counter example, which will prove memory can eventually be written to.

$$\neg \begin{bmatrix} \Diamond(\text{MAIN\_MEMORY}[0] = 0) \\ \wedge \Diamond(\text{MAIN\_MEMORY}[0] = 1) \end{bmatrix} \tag{1}$$

The same proof structure as before is used, this time to show that there exists a trace in which a cache line can eventually reach each of the MESI states. Since the caches are symmetrical, it suffices to show this property for any cache block in any cache.

$$\neg \begin{bmatrix} \Diamond(\text{CACHES}[0].\text{lines}[0].\text{state} = \texttt{Invalid}) \\ \wedge \Diamond(\text{CACHES}[0].\text{lines}[0].\text{state} = \texttt{Shared}) \\ \wedge \Diamond(\text{CACHES}[0].\text{lines}[0].\text{state} = \texttt{Exclusive}) \\ \wedge \Diamond(\text{CACHES}[0].\text{lines}[0].\text{state} = \texttt{Modified}) \end{bmatrix} \tag{2}$$

The counter trace to this LTL formula proves that there exists a trace in which a cache block will eventually be written to.

$$\neg \begin{bmatrix} \Diamond(\text{CACHES}[0].\text{lines}[0].\text{data} = 0) \\ \wedge \Diamond(\text{CACHES}[0].\text{lines}[0].\text{data} = 1) \end{bmatrix} \tag{3}$$

A counterexample to this LTL formula proves that that there exists a trace in which a cache line can hold data from multiple different memory addresses. This property is only true of configurations with `MEMORY_SIZE` > `CACHE_SIZE` since this model uses a direct-mapped cache.

$$\neg \begin{bmatrix} \Diamond(\text{CACHES}[0].\text{lines}[0].\text{tag} = 0) \\ \wedge \Diamond(\text{CACHES}[0].\text{lines}[0].\text{tag} \neq 0) \end{bmatrix} \tag{4}$$

The objective of the below LTL is to show that there exists a trace in which all the bus transactions eventually occur. This LTL statement is particularly useful when determining if the cache configuration is sufficiently

complex to invoke all the bus side transitions.

$$\neg \begin{bmatrix} \Diamond(\text{BUS.op} = \texttt{None}) \\ \wedge\ \Diamond(\text{BUS.op} = \texttt{BusRd}) \\ \wedge\ \Diamond(\text{BUS.op} = \texttt{BusRdX}) \\ \wedge\ \Diamond(\text{BUS.op} = \texttt{BusUpgr}) \\ \wedge\ \Diamond(\text{BUS.op} = \texttt{Flush}) \end{bmatrix} \tag{5}$$

The above statements were used to validate each configuration described in section 4. Additionally, multiple assert statements were added to the model to further improve confidence in the validity of the model. First, there is an assert statement that detects if a processor has modified and flushed data that another processor has a valid copy of. Second, there is an assert statement that asserts that when data is read, the value in cache matches the value present in main memory.

3.2 Verification

(Harrison 2010) has defined coherency for the MESI protocol as follows,

$$\Box \begin{bmatrix} \forall i.\ \big(\text{Cache}(i) \in \{\texttt{Modified}, \texttt{Exclusive}\}\big) \\ \quad \rightarrow \forall j.\ \big(\neg(j = i) \rightarrow\ \text{Cache}(j) = \texttt{Invalid}\big) \end{bmatrix}. \tag{6}$$

If we enumerate the permitted states (shown in Table 1) for a cache line given any pair of caches we see that Harrison's definition does not sufficiently describe the relation of the shared state between caches. A stronger definition is as follows,

$$\Box \begin{bmatrix} \quad \forall i.\ \big(\text{Cache}(i) \in \{\texttt{Modified}, \texttt{Exclusive}\}\big) \\ \qquad \rightarrow \forall j.\ \big(\neg(j = i) \rightarrow\ \text{Cache}(j) = \texttt{Invalid}\big) \\ \wedge\ \forall i.\ \big(\text{Cache}(i) \in \{\texttt{Shared}\}\big) \\ \qquad \rightarrow \forall j.\ \big(\neg(j = i) \rightarrow\ \text{Cache}(j) = \texttt{Shared} \\ \qquad\qquad\qquad\qquad \vee\ \text{Cache}(j) = \texttt{Invalid}\big) \end{bmatrix}. \tag{7}$$

Dynamically expressing a statement over all i, j in SPIN is challenging so we take advantage of the fact that the caches are symmetrical in this

	M	E	S	I
M	✗	✗	✗	✓
E	✗	✗	✗	✓
S	✗	✗	✓	✓
I	✓	✓	✓	✓

Table 1: The permitted states that a cache line can have for any pair of caches.

model. Therefore if there is a violation of the permitted states between any two caches the same violation can exist between every other combination of caches. Hence it suffices to show that no violation exists between the first and second cache.

3.3 Selecting Sufficiently Large Structures

The model has been created parameterized cache and memory sizes. Note that as mentioned previously block size has been fixed to 1bit since the data stored in memory is irrelevant and if MESI works with 1bit block sizes it will work with any greater finite block size. Statespace explosion will be a limiting factor to verifying the MESI protocol so it is important to minimize the size of the caches and memory structures while maintaining enough complexity to achieve coverage of all MESI the transitions. Consider the simplest possible configuration of 2 processors with caches that have a capacity of 1 block and with an equally small main memory. It is clear that a conflict miss can never occur since the caches can effectively hold the entire contents of memory. This is not only not realistic but it also prevents coverage of the FLUSH bus operation. Equation 5 is useful for the purpose identifying whether the model is sufficiently complex so that all bus operations can eventually be issued. Additionally, SPIN reports unreached states which in this case can be used as an indicator of whether structures are sufficiently sized to reach all the decisions in the model. Through preliminary experiments and reasoning as explained above the minimum configuration required to achieve coverage of all MESI transitions is a cache size of 1 block and a memory is of 2 blocks.

3.4 Model Simplification

The statespace explosion will be the biggest limiting factor to verification. This was confirmed with some preliminary experimentation where the first version of the model required over 110GB of memory to verify a 3 processor configuration. Verifying a 4 processor configuration was not initially achievable using SPIN since the statespace depth was beyond 1410065407, which appears to be a hard limit that cannot be overcome with the `-mN` flag. Hence it is desirable to simplify the model as much as possible while not compromising the behavior of interest. This sections presents model simplifications that where utilized. After implementing each of these simplifications/optimizations the model was re-validated.

3.4.1 Optimizing Non-Deterministic Selection

There are multiple ways to implement non-deterministic selection in Promela. The most straightforward of which is to use the `select` keyword which Promela provides for this purpose. The `select` keyword can be problematic since SPIN appears to convert this into an equivalent `do`-loop which introduces iteration. This iteration can dramatically increase the search depth required for verification, increases statespace, and decreases verification speed compared to an equivalent `if` statement. To this end, if the bounds of the selection are known at compile-time the statement should be expanded to the corresponding `if` construct.

3.4.2 Reduce with Atomic

This trick abuses the atomic keyword and I came about it after reading (McKenney 2007). The idea is to create atomic regions to reduce state space without altering the behavior that we are interested in verifying. For example, consider a series of assignments and assertions that not visible to any other process. Wrapping these sequential operations in an atomic statement can greatly reduce statespace without compromising the integrity of the model.

3.4.3 Deterministic Acknowledgments

The previous simplifications are general and can be applied to other models, however this last optimization is specific to this application. This optimization involves selecting a deterministic acknowledgment sequence rather than a non-deterministically selected one. The order in which each processor acknowledges a bus message is not important to the MESI protocol itself and is left to the implementation. The sequence selected for this model is to acknowledge in order of ascending `_pid`.

4 Results

Configurations of 2, 3, and 4 caching agents have been successfully verified. Although one might initially assume that a protocol based on four states would have a small statespace this is not the reality. In fact, it is only possible to verify the MESI protocol for a small number of processors. The statespace grows astonishingly quickly. Each cache block has its own state information, each cache consists of multiple multiple cache blocks, each processor has its own cache and there is main memory above all the caches. Additionally, there are many intermediate states to handle communication between the processors and the web of transitions for each cache block. Statespace ex-

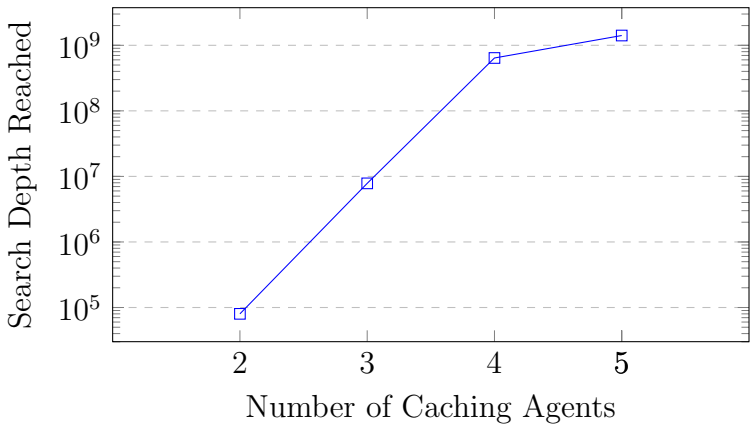

Figure 3: Search depth reached during verification.

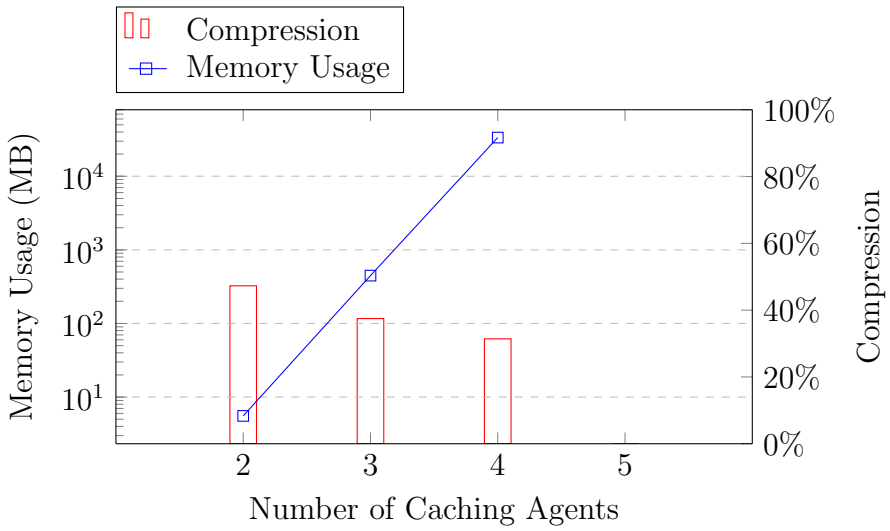

Figure 4: Actual memory usage for states and compression.

plosion is the greatest limiting factor for how large of a configuration can be verified. More specifically, verification is limited beyond 4 processors due to SPIN's max search depth of 1,410,065,407.

Even with significant effort invested in reducing the statespace, search depth, memory usage, and elapsed time grow by 2 orders of magnitude with each additional processor. Figure 3 shows the maximum search depth reached versus number of caching agents. Figure 4 shows the memory used for storing states and additionally shows how compression improves as the number of caching agents increases. The time taken to verify each configuration is shown in Figure 5 along with the rate at which the state space was being searched in states per second. With each additional processor there is a notable reduction in the rate of exploration.

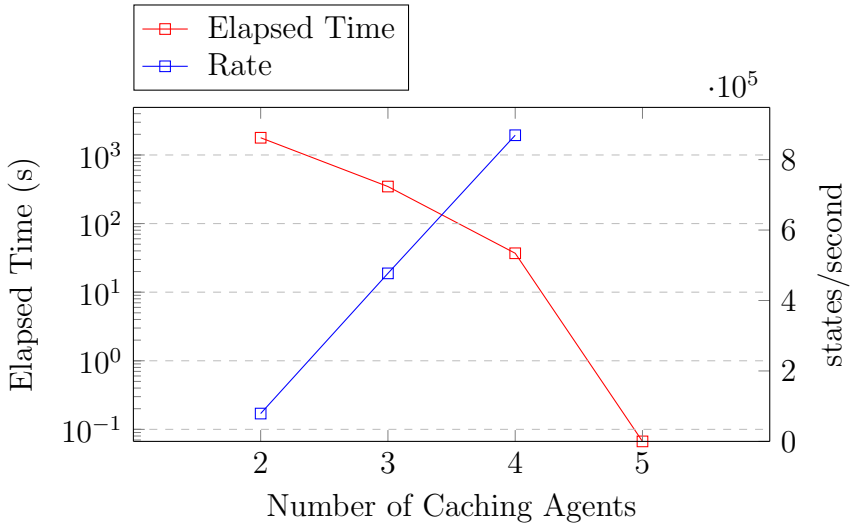

Figure 5: Verification time and rate.

5 Conclusion

The primary objective of this project is to verify the MESI Cache Coherency Protocol through *Model Checking*. To this end, SPIN will be used to explore the state space and validate that the coherency property is not violated. There are a few additional considerations when creating the model. MESI is modular in the sense that it can be extended beyond the state space of just two caching-agents, with the practical limits being communication overhead and physical footprint of the processor. To verify as many configurations as possible, the number of caching agents n as well as the cache size m will be incremented until the state space becomes too large to model in SPIN. While the size of the caches could vary across caching-agents, in practice it is more common for caches at the same level to be symmetrical. This simplifies creating the model in SPIN and limits the number of configurations. Metrics such as memory usage and execution time of the SPIN verification will recorded so observations can be made about the impacts cache size and number of caching agents. To test that the model correctly represents the protocol, the resulting state space generated by SPIN will be compared to the calculated expected state space for each cache configuration. Furthermore,

LTL specifications will be written to further test that each of the allowed transitions can occur, and that no disallowed transitions can occur.

All the code necessary to reproduce the results presented in this paper as well as the artifacts produced can be found on this project's github page: https://github.com/IowaStateAerospaceCourses-Rozier/applied-formal-methods-final-project-cache-me-outside.

References

Culler, David, Jaswinder Pal Singh, and Anoop Gupta. 1998. *Parallel Computer Architecture: A Hardware/Software Approach.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 9780080573076.

Harrison, John. 2010. "Formal Methods at Intel — An Overview." Intel Corporation. Presentation at the Second NASA Formal Methods Symposium, NASA HQ, Washington DC, 14th April 2010, 09:00–10:00. Accessed October 25, 2023. https://shemesh.larc.nasa.gov/NFM2010/talks/harrison.pdf.

Holzmann, G.J. 1997. "The model checker SPIN." *IEEE Transactions on Software Engineering* 23 (5): 279–295. https://doi.org/10.1109/32.588521.

McKenney, Paul. 2007. "Using Promela and Spin to verify parallel algorithms" (August). https://lwn.net/Articles/243851/.

Nakamoto, Aoi. 2018. "W-Shield: Protection against Cryptocurrency Wallet Credential Stealing." In *Workshop on Security and Privacy in E-Commerce 2018,* 71–107. Yokohama, Japan, June.

Papamarcos, Mark S., and Janak H. Patel. 1984. "A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories." In *Proceedings of the 11th Annual International Symposium on Computer Architecture,* 348–354. ISCA '84. New York, NY, USA: Association for Computing Machinery. ISBN: 0818605383. https://doi.org/10.1145/800015.808204. https://doi.org/10.1145/800015.808204.