Chapter 8
RUNTIME VERIFICATION IN MULTI-CORES

Abstract. In this chapter we shift the focus of our discussion to multi-core processors and issues specific to their runtime verification. In Chapter 4 we overviewed features of modern multi-core designs and described the growing challenge of their verification. As we pointed out, this arduous task is exacerbated by the increasing complexity of the shared memory communication subsystem and the need to verify two of its major system-wide properties: cache coherence and memory consistency. In this chapter we present two runtime solutions designed specifically for this purpose. The first technique, called Dynamic Verification of Memory Consistency (DVMC), designed by Meixner, et al. is a checker-based solution, which employs multiple distributed monitors to validate different aspects of communication at runtime. The second solution, Caspar, was developed by us as a patching approach that uses on-die matchers, programmed with patterns describing known bugs, to identify errors. Moreover, to be effective at runtime, both solutions include not only a detection, but also a recovery mechanism, so bugs can be sidestepped and forward progress can be maintained. Thus, as a part of our discussion, we also overview the recovery techniques used in both DVMC and Caspar.

8.1 Dynamic Verification of Memory Consistency

As discussed in Chapter 4, multi-core architectures have become a staple of modern, high-performance processor design, due to their beneficial power/performance characteristics. Since the introduction of the first mainstream dual-core processors, the number of computational elements in these devices have been increasing steadily from generation to generation. While the architecture of individual cores remains relatively simple, compared to the high-end superscalar out-of-order pipelines of the previous decade, on-die interconnect and communication subsystems in these designs are substantially more complex. Furthermore, as the number of cores grows, the interconnect medium starts to move away from relatively unscalable buses to complex interconnect fabric, such as point-to-point or mesh networks. This, in turn,
leads to non-predictable communication latencies that depend on the network topology and the application’s data transfer patterns. As a consequence, the verification of the communication subsystem is becoming more and more important and resource demanding, compared to the validation of individual cores. In particular, this entails verifying such properties as cache coherence and memory consistency, described earlier in Section 4.2. Finally, as design and production timelines keep shrinking, processor manufacturers are forced to release their products well before they are fully verified, exposing end-users to potential escaped errors. To combat the issue, researchers have proposed runtime (dynamic) verification solutions, which augment a baseline design with specialized hardware components, monitoring the state of the system and initiating global recovery if an erroneous operation is detected.

One of the first solutions to address the correctness of operation in multi-core shared memory systems at runtime was developed at Duke University [MS06]. This work, called Dynamic Verification of Memory Consistency (DVMC) is a checker-based solution, which augments individual cores with a set of distributed checkers, dedicated hardware blocks that ascertain that the runtime behavior of the system matches the consistency model specified at design time. If an error is detected, the checkers initiate system-wide recovery, using a checkpointing scheme proposed by Sorin, et al. in [SMHW02], called SafetyNet, that we briefly overview later in this chapter. Conceptually, DVMC attempts to verify the three invariants of multi-core operation: uniprocessor ordering, allowable reordering, and cache coherence. The authors prove that satisfying these three properties is a sufficient condition to guarantee memory consistency. Below we discuss each invariant and then mechanisms deployed to check them in detail.

**Uniprocessor ordering.** This invariant states that if a shared memory location was accessed by a single core only, load operations issued to this address must return the value of the last store issued by the core to the same address. Thus, private data in any core’s cache must remain unchanged, unless modified by the core itself. Runtime verification of this invariant can be enforced by a DIVA-like replay of all memory operations before they commit, to check that values received during replay match those obtained during execution (see Section 6.3 for details of the DIVA technique). To implement this invariant, DVMC requires an extra stage in the processor pipeline, called the verification stage, which re-issues all memory operations just before they retire. Note, however, that in this stage memory operations are issued in commit order, thus correctness of out-of-order execution is checked against the underlying architectural model of memory accesses. The replay of store operations, however, must still be speculative, to prevent these accesses from affecting the architectural state. Thus, the authors augment the system with a verification cache (Figure 8.1), which captures write accesses and holds them until they retire.

**Allowable reordering** is another global property of memory consistency models, which identifies how the order of memory operations can be changed. Consistency models weaker than sequential consistency allow some accesses to bypass preceding operations to improve overall system performance in presence of high-latency