Software and Hardware Support for Recording and Deterministically Replaying Concurrent Programs

Klaus Danne, Gilles Pokam, Cristiano Pereira
Intel Corporation

The paper gives a short introduction into the use cases of deterministic replay, the technology, proposed hardware support and recent achievements in memory race recording.

Use cases: Deterministically reconstructing a program execution has several use cases. First, it can be used to debug programs by being able to reconstruct a bug and observing the situation in enhanced debugging tools. Such tools can for example allow the illusion of stepping backwards trough the execution. This is especially useful for concurrency bugs in multi-threaded software that occur non-deterministically and are difficult to reproduce without replay. Second, replay can be used to create fault tolerant systems. A secondary machine is kept synchronous to a primary machine by replaying its execution right away in a virtual lock step mode. If the primary machine fails, the secondary takes over the execution. Third, replay can be used in the security domain. For example when an security hole in a system becomes known, the execution of the say past moth can be replayed and analyzed to identify whether the flaw was exploit and what actions the intruder performed. Those uses cases all employ deterministic replay, but have different requirements: For debugging, recording should have minimal impact on the execution itself. Otherwise it might happen that a bug only manifests when recording is turned off but is masked when recording is turned on. For fault-tolerance, record and replay speed must be close to native execution, since both limit the overall system performance. For the security use case, minimizing the amount of logging data is key, since one wants to keep logs of long time periods (weeks or months) for later analysis.

Technology: The common approach to implement deterministic replay is to record all non-deterministic events at runtime and to enforce these events during replay at the exact same position in the instruction stream. Non-deterministic events are program inputs, non-deterministic instructions such as reading the CPU ID or the timestamp register, DMA transfers, interrupts, OS signals, and – most challenging – the memory access interleaving of multiple threads. Except for the last one, all these events can be efficiently logged by extended system software, e.g. by an extended OS or an virtual machine monitor. This has been proven by academic as well as commercial systems [4]. However, recording the memory access interleaving, or more specifically recording the dependencies that result from different threads accessing the same memory locations, is likely to require hardware support in order to enable acceptable system performance.

Hardware Support: Most proposed hardware based approaches, also known as memory race recorders (MRR), piggyback on the cache coherence protocol to observe memory races. While early schemes where unrealistic costly in terms of the required hardware resources and the amount of logging data, two recent approaches show great improvements in both metrics. The common key method is to avoid recording of individual memory races, but to focus on recording the blocks of dynamically executed instructions that do not conflict with other threads. RERUN [1] for example, records such non-conflicting blocks, called episodes, by storing their length in terms of (memory-)instructions and a time stamp that orders the episodes of all threads. A deterministic replay with respect to memory races can be reconstructed by sequentially executing all episodes in order of increasing timestamps. I.e., a replayer will examine the log files to identify which thread should be dispatched next and for how many instructions it is allowed to execute until an episode of an different thread needs to be replayed. DeLorean [2] uses a similar concept of recording non-conflicting blocks, called chunks, but is based on a different multiprocessor execution environment. The hardware divides all execution into chunks that execute in isolation and invisible to other cores until they commit. When two concurrent chunks conflict, only one commits and the other is squashed and needs to execute again. To enable deterministic replay the order in which the chunks commit is logged. Since the hardware usually creates chunks us of a fix size, e.g. 1000 instructions, no additional information needs to be logged except for the rare cases where chunks need to end early due to events such as interrupts.

Both approaches need to be able to efficiently check whether two blocks of execution conflict or not, i.e. whether they have read or written to the same memory address. This can be done by employing signatures – hardware implementations of bloom-filters. They enable the approximation of large sets, such as the set of written memory addresses, in a small finite state. The approximation results in fault-conflicts which impacts performance but not correctness.

Chunk-based Memory Race Recorder for modern CMPs: While recording and deterministically replaying of software on uniprocessors has been proven feasible even by commercial products [4], the approaches for hardware support for MRR are still academic and may face additional challenges when being considered as features for today's or tomorrow's processors.

In the talk we discuss the mentioned use cases of deterministic replay, review the technology and discuss our recent achievements to make chunk-based MRR practical for modern CMPs [3]. In particular we show that MRR interactions with a cache hierarchy can degrade performance and presented a novel mechanism that mitigates this degradation. We introduce new mechanisms for snoop based caches that eliminate coherence traffic overhead. We finally show new techniques for improving replay speed and introduce a novel framework for evaluating the replay speed potential of MRR designs.

References

[1] D. Hower and M. Hill. Rerun: Exploiting episodes for lightweight memory race recording. In *Proceedings of the International Symposium on Computer Architecture*, 2008.

[2] P. Montesinos, L. Ceze, and J. Torrellas. Delorean: Recording and deterministically replaying shared-memory multiprocessor execution efficiently. In *Proceedings of the International Symposium on Computer Architecture*, 2008.

[3] Gilles Pokam, Cristiano Pereira, Klaus Danne, Rolf Kassa, and Ali-Reza Adl-Tabatabai. Architecting a chunk-based memory race recorder in modern cmps. In *Porceedings of the 42nd ACM/IEEE International Symposium on Microarchitecture, to appear*, 2009.

[4] Min Xu, Vyacheslav Malyugin, Jeffrey Sheldon, Ganesh Venkitachalam, and Boris Weissman. Retrace: Collecting execution trace with virtual machine deterministic replay. In *Proceedings of the 3rd Annual Workshop on Modeling, Benchmarking and Simulation, MoBS*, 2007.