# Viser: Providing Serializability in Hardware with Simplified Cache Coherence

Swarnendu Biswas
Ohio State University (USA)
biswass@cse.ohio-state.edu

## Abstract

While existing architectures like x86 and SPARC provide strong hardware memory consistency models, such as TSO, programming language memory models are more relaxed. This divide nullifies the usefulness of providing strong hardware memory models, since languages and compilers provide a weaker guarantee. Moreover, current shared memory systems implement complex cache coherence protocols which add to the complexity.

This work proposes a microarchitecture, called *Viser*, that ensures strong semantics—serializability of synchronization-free regions (SFRs)—in the absence of region conflicts *even* for racy program executions. Given an execution, Viser either reports a serializability violation or guarantees SFR-serializability, in effect providing the same guarantees provided by languages such as C++ and Java for data-race-free programs only. Viser's design also allows for greatly simplifying existing cache coherence protocols, without requiring any assumptions about language-level properties such as data-race-freedom.

***Categories and Subject Descriptors*** C.1.0 [*Computer Systems Organization*]: Processor Architectures—general; D.3.4 [*Programming Languages*]: Processors—compilers, run-time environments

***Keywords*** Memory models; cache coherence; region serializability; conflict exceptions; region conflicts; data races

## 1. Problem and Motivation

Existing architectures such as x86 and SPARC provide strong memory consistency models such as *total store order*. However, programming languages such as C++ and Java provide more relaxed memory models that provide strong semantics *only* for data-race-free programs. These languages provide no useful guarantees about the semantics of data races. This divide between the hardware and language memory models nullifies the usefulness of providing strong hardware memory models [1]. Other memory model alternatives such as *sequential consistency* also have drawbacks and are not particularly strong [1]. In addition, the performance and scalability of current shared-memory multiprocessor systems are limited by complex cache coherence protocols [3].

*SPLASH Companion'15*, October 25–30, 2015, Pittsburgh, PA, USA
ACM. 978-1-4503-3722-9/15/10
http://dx.doi.org/10.1145/2814189.2815375

## 2. Background and Related Work

There are two possible directions to address the suboptimal co-design of existing hardware and programming languages. The first option is to strengthen language memory models, but *data races* present a significant impediment [1]. Programming languages such as Java and C++ provide strong semantics—serializability of SFRs—but only in the absence of data races. Unfortunately, they provide few or no guarantees for racy executions [1]. Much prior work delves into the problem of detecting data races [4], but detecting all races in an execution incurs high overhead which prohibit their use as a basis for language semantics. Prior work avoids the expense of detecting all happens-before races by instead detecting conflicts between regions (e.g., SFRs and RFRs) [2, 5]. For *conflict-free* executions, regions execute serially, otherwise a serializability violation is reported. Existing region conflict detectors are however impractical: software-based detectors incur high overhead [2], whereas hardware approaches introduce significant complexity and are unscalable [5].

An alternate option is to improve performance and scalability of current shared-memory multiprocessor systems, by simplifying existing complex cache coherence protocols. A simple coherence protocol facilitates verifying correctness, improves scalability and potentially performance, and reduces energy requirements. However, applicability of prior work is severely limited by assumption about data-race-freedom at the language-level [3].

## 3. Viser: Serializability in Hardware

This work proposes *Viser*, which is a microarchitecture redesign to solve two important problems: 1) provide strong semantics *even* for racy executions that programming languages can rely on, and 2) leverage the opportunity to simplify the already-complex cache coherence protocols. Viser soundly and precisely checks for region (e.g., SFRs, RFRs) conflicts, and raises an exception whenever serializability is violated (implies a *true* data race). An exception-free execution in Viser implies region serializability. Our work also shows that such "region conflict exceptions" semantics can be used to simplify complicated coherence protocols.

Motivated by prior work [2], Viser uses a mix of eager and lazy techniques to detect region conflicts. Accesses to data that reside in private non-evicted cache lines use lazy versioning and lazy conflict detection, like in software transactional memory systems, because it is expensive to detect all write–write and write–read conflicts eagerly. For accesses that "hit" in a private cache, checking for conflicts is *delayed* till the cache line is evicted to a shared LLC or the core executes a synchronization operation. To check for conflicts for accesses to a private line, the private cache controller sends the access information to the LLC. Evicted cache lines (i.e., LLC lines) use eager versioning and eager conflict detection. To eagerly detect conflicts, the LLC stores precise access metadata for each evicted line for each core that has evicted the line. Future accesses to the LLC line from other cores can detect a conflict eagerly by comparing with the LLC line metadata. To account for eviction from the LLC, Viser writes back the line *along* with the metadata to memory—in effect, the memory creates an illusion of an unbounded LLC.

***Microarchitecture design and coherence protocol.*** The design of Viser assumes a three-level cache hierarchy, with writeback and inclusive private L1 and L2 caches, and a shared non-inclusive LLC. Viser uses a directory-based coherence protocol, with the directory embedded in the LLC. The coherence protocol in Viser requires only two states: *invalid* and *valid*. A *valid* line means that the data is consistent in the system. Unlike a directory-based protocol, the LLC does *not* maintain *read sharers*, thereby reducing space overhead and avoiding a significant source of scalability bottleneck. Each cache line in Viser adds one read bit and one write bit *per* byte to maintain precise access information. To eagerly detect conflicts, the LLC needs to maintain precise access information for each evicted line from each core. Naïvely maintaining the metadata in the LLC is impractical, therefore the design uses a sparse representation indexed with line addresses that sits alongside the LLC and stores the access metadata.

***Ensuring serializability.*** In the absence of conflicts, serializability of regions is guaranteed if the following conditions are satisfied: a) a region's writes appear to be atomic, and (b) the values read in a region are consistent. To ensure serializability, each core writes back its writes and validates its reads in private cache lines (i.e., not yet evicted) *at* region boundaries. Accesses that are part of evicted lines are not processed since any conflict on them will be detected eagerly by the LLC. In Viser, a core repeatedly executes the following steps at a region boundary *until* either a conflict is detected or a consistent snapshot is read:

1. **Pre-commit:** Write back the write access information and the updated values for dirty lines to the LLC.
2. **Read validation:** Validate reads using value validation.
3. **Post-commit:** Clear the access information pertaining to the current core from the LLC, and invalidate all lines from its private caches.

A successful validation of a region's accesses ensures that the writes appear to happen atomically, and all the reads observe a consistent snapshot of values, which implies serializability.

## 4. Results and Contributions

***Results.*** We have implemented Viser with a Pin-based simulator. Our Pintool instruments programs and generates a trace of events which is consumed by a Java backend that models a Viser core. To evaluate Viser's performance, we have also implemented a directory-based MESI coherence protocol. Our initial experiments show that Viser improves performance by 8% on the average over a traditional MESI system for the PARSEC 3.0 benchmarks with the *simmedium* input size. The Viser protocol increases on-chip network traffic (in flits) by 60%, whereas it decreases traffic between the LLC and memory by 65%. Viser's simpler cache coherence protocol translates to better scalability and reduced energy requirements due to fewer coherence messages being issued, and has the added benefit of being easy to validate [3].

***Contributions.*** This work proposes Viser, a microarchitecture design that associates strong semantics even to racy program executions. For an arbitrary program execution, Viser guarantees to either report a serializability violation or regions execute *as if* serially. Viser introduces holistic but simple changes to existing processor architecture and cache coherence protocols to efficiently detect SFR conflicts. Viser's simplified directory-based cache coherence protocol is expected to scale well and has reduced energy requirements. We expect Viser will provide a significant advancement in enabling stronger semantics for all program executions.

## References

[1] S. V. Adve and H.-J. Boehm. Memory Models: A Case for Rethinking Parallel Languages and Hardware. *CACM*, 53:90–101, 2010.

[2] S. Biswas, M. Zhang, M. D. Bond, and B. Lucia. Valor: Efficient, Software-Only Region Conflict Exceptions. In *OOPSLA*, 2015.

[3] B. Choi, R. Komuravelli, H. Sung, R. Smolinski, N. Honarmand, S. V. Adve, V. S. Adve, N. P. Carter, and C.-T. Chou. DeNovo: Rethinking the Memory Hierarchy for Disciplined Parallelism. In *PACT*, pages 155–166, Washington, DC, USA, 2011.

[4] C. Flanagan and S. N. Freund. FastTrack: Efficient and Precise Dynamic Race Detection. In *PLDI*, pages 121–133, 2009.

[5] B. Lucia, L. Ceze, K. Strauss, S. Qadeer, and H.-J. Boehm. Conflict Exceptions: Simplifying Concurrent Language Semantics with Precise Hardware Exceptions for Data-Races. In *ISCA*, pages 210–221, 2010.