Summary

In summary, the key to our work was (1) we took a more programmer-centric view of the problem compared to the more prevalent hardware-centric view at that time, and (2) our persistence in seeking the minimal possible constraints for the hardware interface. This resulted in our redefining the problem in programmer-centric terms, enabling a better understanding of some of the fundamental issues. It is perhaps worth noting that when we began this work, the problem seemed deceptively simple, and a highly respected senior colleague actually warned us that we were getting into what appeared to be a closed area!

Acknowledgments

We thank those acknowledged in the original paper and Jim Goodman who we regrettably omitted. We thank the sponsors of this research for the confidence they showed in a new professor: Michael Foster and Zeke Zalcstein of the National Science Foundation, Rae McLellan of Bell Labs, Chuck Thacker then of DEC, Doug Johnson of TI, Chris Hsiung then of Cray, and the University of Wisconsin Graduate School. We want to especially note the acceleration brought to our careers by the NSF's Presidential Young Investigator program.

Biographies

Sarita V. Adve continued work on memory consistency models for her Ph.D. thesis, under the supervision of Mark Hill and supported by an IBM graduate fellowship. She joined Rice University as an assistant professor in 1993, where she has worked on techniques to improve and evaluate the performance of shared-memory systems. She received an NSF CAREER award in 1995, an IBM Partnership award in 1997, and an Alfred P. Sloan research fellowship in 1998.

Mark D. Hill continued research on memory consistency models, large caches, translation lookaside buffers, and page tables. With Professors James R. Larus and David A. Wood, he co-founded the Wisconsin Wind Tunnel project that has developed new methods and new designs for parallel computer systems. After earning tenure, Hill went on sabbatical to Sun Microsystems where he worked on high-end servers. Hill is now Professor and Romnes Fellow at the University of Wisconsin-Madison, Information Director of ACM SIGARCH, and a Senior Member of the IEEE.

References

[1] S. V. Adve, V. S. Pai, and P. Ranganathan. Recent Advances in Memory Consistency Models for Hardware Shared-Memory Systems. *To appear in the Proceedings of the IEEE, special issue on distributed shared-memory systems*, 1999.

[2] Sarita V. Adve and Kourosh Gharachorloo. Shared Memory Consistency Models: A Tutorial. *IEEE Computer, special issue on shared-memory multiprocessing*, pages 66–76, December 1996.

[3] Sarita V. Adve and Mark D. Hill. Weak Ordering - A New Definition. In *Proc. 17th Ann. Intl. Symp. on Computer Architecture*, pages 2–14, May 1990.

[4] William W. Collier. *Reasoning about Parallel Architectures*. Prentice-Hall, Englewood Cliffs, New Jersey, 1992. Parts of this work originally appeared as IBM technical reports in 1984 and 1985.

[5] Michel Dubois, Christoph Scheurich, and Faye A. Briggs. Memory Access Buffering in Multiprocessors. In *Proc. 13th Ann. Intl. Symp. on Computer Architecture*, pages 434–442, June 1986.

[6] Kourosh Gharachorloo, Anoop Gupta, and John Hennessy. Two Techniques to Enhance the Performance of Memory Consistency Models. In *Proc. Intl. Conf. on Parallel Processing*, pages I355–I364, 1991.

[7] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *Proc. 17th Ann. Intl. Symp. on Computer Architecture*, pages 15–26, May 1990.

[8] James R. Goodman. Cache Consistency and Sequential Consistency. Technical Report #61, SCI Committee, March 1989. Also available as Computer Sciences Technical Report #1006, University of Wisconsin, Madison, February 1991.

[9] Allan Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Lawrence Rudolph, and Marc Snir. The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer. *IEEE Trans. on Computers*, pages 175–189, February 1983.

[10] Mark D. Hill. Multiprocessors Should Support Simple Memory Consistency Models. *IEEE Computer*, to appear in 1998.

[11] Pete Keleher, Alan L. Cox, and Willy Zwaenepoel. Lazy Release Consistency for Software Distributed Shared Memory. In *Proc. 19th Ann. Intl. Symp. on Computer Architecture*, pages 13–21, 1992.

[12] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[13] Leslie Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Trans. on Computers*, C-28(9):690–691, September 1979.

[14] Robert H. B. Netzer and Barton P. Miller. Detecting Data Races in Parallel Program Executions. *Research Monographs in Parallel and Distributed Computing, MIT Press, 1991.*, August 1990.

[15] V. S. Pai, P. Ranganathan, S. V. Adve, and T. Harton. An Evaluation of Memory Consistency Models for Shared-Memory Systems with ILP Processors. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, pages 12–23, 1996.

[16] Dennis Shasha and Marc Snir. Efficient and Correct Execution of Parallel Programs that Share Memory. *ACM Trans. on Programming Languages and Systems*, 10(2):282–312, April 1988.

the notion of a data race. We realized the connection between their characterization and the application behavior that we were attempting to characterize for our weak models. It became clear (at least intuitively) that weak ordering and the weaker models we were trying to develop what appeared sequentially consistent to data-race-free programs.

We now had a formal understanding of what was needed from the application. We were still, however, grappling with hardware conditions for our new ideal memory model. Nevertheless, at this point, we thought we had a well-defined path that would lead us to the ideal model. All we needed was to determine the minimal set of hardware constraints that would provide sequentially consistent results for data-race-free programs, and call those constraints our new memory model (or so we thought).

Minimal conditions for the hardware and model

For almost three months, we frequently invented a new “model of the day.” We would formalize a set of conditions that appeared necessary and sufficient, but soon would discover another way to weaken those conditions. To prove or disprove the correctness of our conditions, we made use of the formal methods developed by Shasha and Snir [16] as well as ad hoc techniques. In late October 1989, we realized that not only were the absolutely minimal hardware constraints elusive, but also that a model defined in terms of the type of constraints we were proposing would be quite complicated.

At this point, we realized that we needed to move beyond viewing the model as purely a set of hardware constraints. The defining moment of this work came with the observation that weak models could be viewed simply as a contract between hardware and software. Given that we had already defined a set of conditions for software, the only necessary condition for hardware was to appear sequentially consistent for the proposed software. Further, we could develop different models by determining different software conditions; the hardware for those models would simply need to appear sequentially consistent to the specified software.

Subsequent Work

After the 1990 paper, most of our immediate work focused on formalizing the software conditions for which commonly used system optimizations would not violate sequential consistency, and on formulating further system relaxations that would not violate sequential consistency. Some of this work was joint with Kourosh Gharachorloo, Anoop Gupta, and John Hennessy of Stanford. A common theme throughout this work was that most problems at first appeared to have deceptively simple solutions; however, formally proving the correctness of the solutions proved to be quite difficult. Our eventual framework to do these proofs benefited immensely from previous formal work by Collier [4] and by Shasha and Snir [16].

The flexibility afforded by defining a memory model in the new programmer-centric way is arguably most evident in the software shared-memory work that followed later. Lazy Release Consistency [11], arguably the most widely cited algorithm for software shared-memory, is weaker than release consistency. However, both release consistency and lazy release consistency obey the data-race-free model since they both appear sequentially consistent for data-race-free programs. Thus, for programmers who write data-race-free programs, these systems are equivalent.

Over the last few years, a rich body of literature in the area of memory consistency models has developed. This includes new models for hardware and software shared-memory, performance evaluations, theoretical frameworks for formal specifications and proofs, and highly successful methods to reduce the hardware performance gap between consistency models. Most advances, however, have been in the domain of hardware and runtime systems. The performance impact of relaxed consistency models on compiler optimizations is still unclear. Programming languages and environments have also only recently begun to address the issue more explicitly, with many supporting relaxed models (e.g., Java, OpenMP, and POSIX). A tutorial on the subject and an overview of recent advances appear in [2,1].

Although memory consistency models are now well-understood, there is no consensus yet about the best consistency model. At the time of this writing, commercial multiprocessors supporting sequential consistency and relaxed consistency models are available. The Digital Alpha and IBM PowerPC processor architectures support relaxed models similar to DRF or release consistency, Intel IA-32 and current SPARC processors support derivatives of a relaxed model called processor consistency, while processors from HP and MIPS support sequential consistency. Recent hardware optimizations that reduce the hardware performance gap between various consistency models [6,15], the lack of quantitative data on the benefits of relaxed models for compiler optimizations, an absence of widely used programming standards for shared-memory, and the requirement on vendors to keep their systems backward compatible are some of the factors that have made a consensus difficult. One of us (Mark Hill) has recently used some of these factors to make an argument for returning the hardware/software interface to sequential consistency [10].

To appear in *Selected Papers from the First 25 International Symposia on Computer Architecture*,
Gurindar S. Sohi, editor, ACM Press, 1998.

A Retrospective on "Weak Ordering—A New Definition"

Sarita V. Adve
Dept. of Electrical & Computer Engineering
Rice University
Houston, TX 77005 USA
sarita@ece.rice.edu

Mark D. Hill
Computer Sciences Dept.
University of Wisconsin-Madison
Madison, WI 53705 USA
markhill@cs.wisc.edu

Introduction

We began work on "*Weak Ordering—A New Definition"* [3] in early 1989 while Sarita Adve was a first-year graduate student and Mark Hill a second-year assistant professor at Wisconsin. It now seems obvious that an interface for shared-memory must be defined. It also seems obvious that such an interface must consider interactions among reads and writes to all shared-memory locations, and must not refer to hardware structures such as caches and write buffers. In early 1989, however, most work related to shared-memory semantics was on cache coherence. Such work reasoned about interactions between reads and writes to a given cache line in isolation, focusing on hardware protocols to ensure that the effect of a newly written value eventually propagated to all processor caches. Only a few papers had been written about a more comprehensive model of memory [8, 9, and references in the main paper].

Our work was primarily motivated by the pioneering work on weak ordering by Dubois, Scheurich, and Briggs [5]. The motivation and intuition behind weak ordering were compelling. However, as originally defined, weak ordering had two problems: (1) the definition was hardware-centric and did not seem to be appropriate as a programming model, and (2) the definition appeared to unnecessarily constrain hardware. These observations steered us towards the following two questions:

- What are the *minimal* conditions that a shared-memory model must impose on hardware?
- How could the shared-memory model be best presented to programmers?

For a while, we viewed the above two questions somewhat independently. The defining moment of this work was when we realized the connection between the two questions and redefined the memory model to be a contract between hardware and software. Specifically, we saw a weakly ordered system as one that provided Lamport's sequential consistency [13] to data-race-free programs. Our model was subsequently dubbed *data-race-free (DRF)* or *data-race-free-0 (DRF0)*.

We next describe the process that led to the paper and briefly summarize later work in the area. Release consistency and the notion of properly labeled programs were developed concurrently with our work and are based on similar ideas [7].

The Process

Search for a weaker model for hardware

Our initial work was hardware and performance-centric, and focused on defining a set of conditions that were less constraining for hardware than Dubois et al.'s weak ordering. We would consider common application characteristics, and develop hardware constraints that would give "reasonable behavior" for those (informally characterized) applications. In this process, we defined multiple models that relaxed consistency requirements in different ways at different points in the program (e.g., at the acquire or release of a semaphore). These models, although less constrained than Dubois et al.'s weak ordering, were nevertheless similar in style to the definition of weak ordering, and suffered from the drawbacks we seeked to alleviate.

A characterization of software

Our first key departure from Dubois et al.'s work was to use partial orders instead of real time in our specifications. Using any notion of real time made the specifications harder to understand from the programmer's viewpoint and unnecessarily constrained hardware. The use of partial orders was motivated largely by the work of Lamport (e.g., [12]) and of Rob Netzer and Bart Miller [14], our colleagues at Wisconsin.

The second important step, in Summer 1989, came from making a deeper connection with the work by Netzer and Miller [14]. They were working on detecting data races in a program, and used a variant of Lamport's happened-before partial order relation for formalizing