Formally specifying memory consistency models and automatically generating executable specifications

Prosenjit Chatterjee and Ganesh Gopalakrishnan*
School of Computing, University of Utah
{prosen | ganesh}@cs.utah.edu
Technical Report UUCS-01-012

Abstract

Memory ordering properties of shared memory multiprocessors are more subtle and less well understood than cache coherence. These properties tend to be processor or platform specific and are not always formally specified. It is difficult to compare even those platforms whose memory ordering properties have been clearly specified as each such platform is usually specified in its own definitional framework.We present a generic and formal specification scheme to specify any realistic memory consistency model that gives an intuitive undertstanding to architects, and implementors of platforms whose memory model is being defined and also a common definitional framework to compare memory models. Another contribution of the paper is to generate an executable specification automatically, given the specification of any memory consistency model expressed in our newly defined framework. This alternative specification can be used to generate all possible outcomes of small assembly-language multiprocessor programs in a given memory model, which is very helpful for understanding the subtleties of the model. The executable specification can also check the correctness of assembly language programs including synchronization routines.

1 Introduction

Shared-memory multiprocessors are increasingly employed both as servers (for computations, databases, files, and the web) and as clients. To improve performance, multiprocessor system designers use a variety of complex and interacting optimizations . These optimizations include cache coherence via snooping or directory protocols, out-of-order processors, store buffers. These optimizations add considerable complexity at the architectural level and even more complexity at the implementation level. Directory protocols, for example, require the system to transition from many shared copies of a block to one exclusive one. Unfortunately, these transitions must be implemented with many non-atomic lower-level transitions that expose additional race conditions, buffering requirements, and forward-progress concerns. Due to this complexity, industrial product groups spend more time verifying their system than actually designing and optimizing it.

To verify a system, engineers should unambiguously define what "correct" means. For a shared-memory system, "correct" is defined by a memory consistency model. A memory consistency model defines for programmers the allowable behavior of hardware.

Sequential Consistency is the most intuitive model for writing shared memory programs[**?**] mainly because all the memory operations in an *execution* that obeys SC can be viewed as a total order 'as if' the program has been run in a uniprocessor. However, many commercial processors implement more relaxed memory consistency models in an effort to improve performance. An example is the insertion of FIFO or coalescing store buffers, non-blocking loads and so on. SPARC Total Store Order (TSO)[**?**], relax the SC requirement where in the total ordering of memory operations a store(st) can appear after a load(ld) that follows it in program order. More relaxed models, such as Compaq(DEC) Alpha[**?**], allow re-ordering between any two instructions . Cray3TD[**?**], PowerPc[**?**] implements

*This work was supported by National Science Foundation Grants CCR-9987516 and CCR-0081406