Towards a Formal Model of Shared Memory Consistency for Intel Itanium™

Prosenjit Chatterjee and Ganesh Gopalakrishnan *  
School of Computing, University of Utah  
Technical Report UUCS-01-003  
http://www.cs.utah.edu/formal_verification/

Abstract

We provide a simple formal model for Itanium™ shared memory consistency [1, 2] covering a core set of instructions. Existing descriptions of Itanium shared memory consistency are based on an informal collection of ordering rules as well as several examples. Our operational model employs widely understood data structures such as buffers and memories, and expresses ordering constraints precisely using a collection of non-deterministic rules. This can enable the construction of reliable prototype implementations, formal verification against implementations, formal verification against other formal models, as well as verification of synchronization routines. Our model covers all published ordering constraints, and also sheds light on tricky concepts such as causality.

1 Introduction

The Itanium™ shared memory consistency model [1, 2] is described in terms of a collection of ordering rules, constraints stated in English, and examples of legal and illegal executions. While good for initial understanding, such descriptions often leave many details unanswered. This can make it difficult for programmers to write reliable MP libraries. As far as we know, a formal specification (operational or otherwise) has not yet been published for Itanium.

In this paper, we provide a simple execution oriented (operational) model for the Itanium shared memory consistency reverse-engineered from [1, 2]. We believe that it accurately (and more completely) describes alternative descriptions publicly available. The availability of an operational model can help designers build executable prototypes to gain deeper understanding. In addition, they can use model-checkers to gain a deeper understanding with respect to synchronization routines as well as specific ordering issues [3, 4]. Like any formal specification, an operational model runs the risk of being over- or under-specified. In this paper we point out, as space permits1 how we have strived to avoid these risks.

Our model deals with cacheable memory instructions consisting of acquire loads (written ld.acq), ordinary loads (ld), release stores (st.rel), and ordinary stores (st), as well as memory fences. It does not currently handle atomic read-modify-writes, non-cacheable memory, or special rules pertaining to data dependencies involving registers [1, Section 13.2]. Despite its simplicity, our model captures all published ordering properties of the instructions we consider, and also sheds more light on corner cases pertaining to causality. While operational models have been proposed for commercial shared memory systems (notably for Sparc V9 [5]), a notable feature of our operational model is its use of a few explicit devices such as vector timestamps [6] to clearly describe the tricky notion of causality.

2 Overview of the Itanium Memory Model

The Itanium memory model can be understood in terms of program and global visibility (“visibility”) orders. For memory operations of type ‘store’, visibility refers to when the effects of the store become apparent to all processors. For memory operations of type ‘load,’ visibility refers to when the execution of load appears to have been carried out for the processor carrying out the load. (All other processors do not directly observe the load happening.) As in [1], for two different memory operations X and Y, X,Y specifies that X is before Y in program order, X → Y indicates that Y must be visible only after X is visible. Further, if

---

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

1Details appear in our webpage.