Multiprocessor Memory Model Verification

Paul Loewenstein, Shailender Chaudhry,
Robert Cypher, Chaiyasit Manovit

21 August 2006

Summary

- Introduction.
- Related Work.
- Sparc TSO Memory Model.
- Strand Model.
- Golden Memory Model.
- Conclusion.
- Further work.

Introduction

- No mechanized formal methods in this work.
- Former formal methods work provided invaluable education.
- Architectural properties required to implement TSO understood.
- Simulation-based verification of RTL against those properties, which are strictly stronger than TSO.

Related Work

[1] A. E. Condon, M. D. Hill, M. Plakal, and D. J. Sorin. Using Lamport clocks to reason about relaxed memory models. In *Proceedings of Fifth International Symposium on High-Performance Computer Architecture*, Orlando, Florida, Jan. 1999.

[2] D. J. Sorin, M. Plakal, M. D. Hill, and A. E. Condon. Lamport clocks: Reasoning about shared memory correctness. Technical Report CS-TR-1367, University of Wisconsin-Madison, Mar. 1998.

Related Work

[3] SPARC International. *The SPARC Architecture Manual: Version 8*. Prentice-Hall, 1992.

[4] SPARC International: David L. Weaver & Tom Germond, Editors. *The SPARC Architecture Manual: Version 9*. Prentice-Hall, 1994.

Related Work

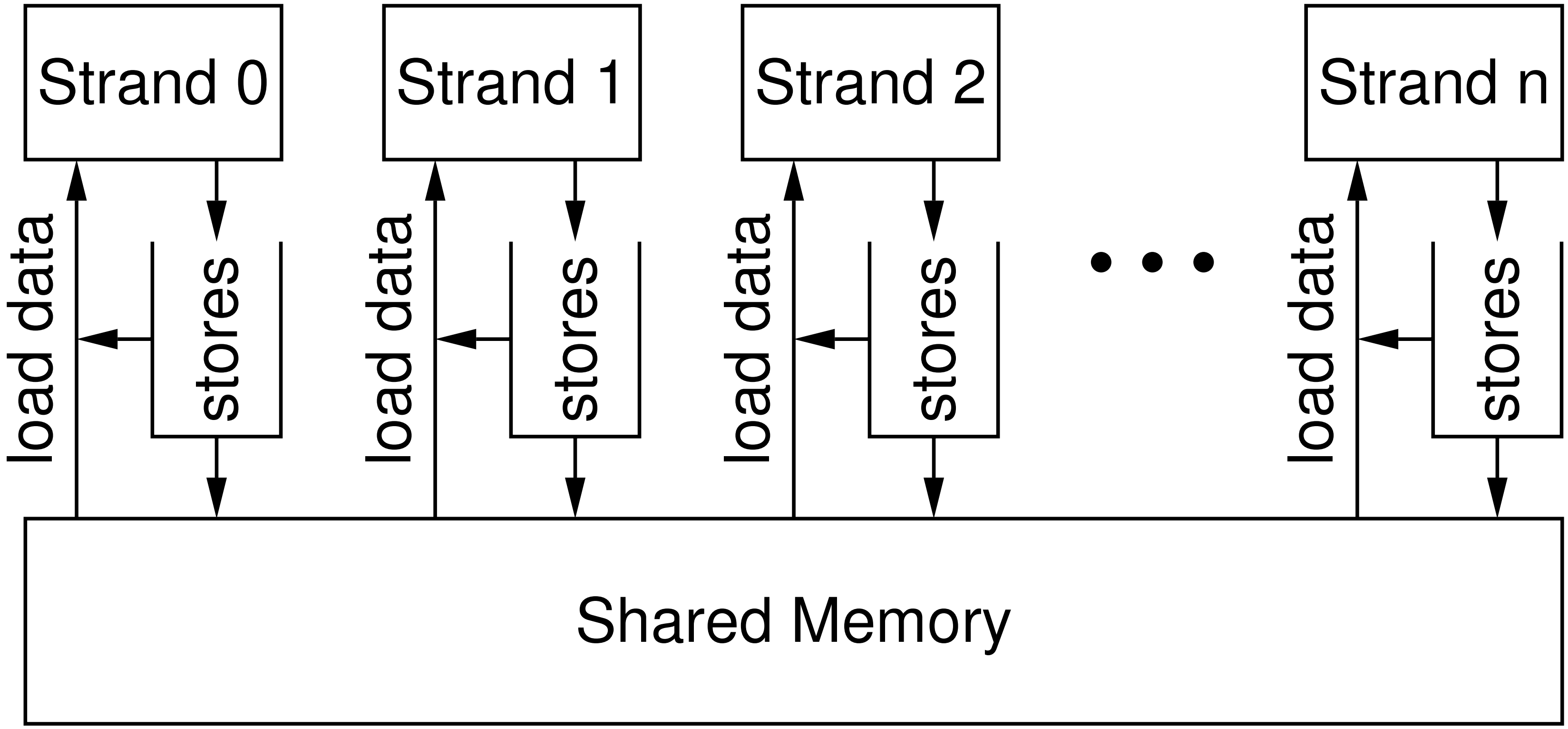
[5] C. Manovit and Sudheendra. Efficient algorithms for verifying memory consistency. In *SPAA'05: Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 245–252, New York, NY, 2005. ACM Press.

[6] A. Roy, S. Zeisset, C. J. Fleckenstein, and J. C. Huang. Fast and generalized polynomial time memory consistency verification. In *Proceedings Computer Aided Verification*, number 4144 in Lecture Notes in Computer Science, pages 503–516, Seattle, WA, Aug. 2006. Springer.

TSO Memory Model

- "Total Store Order"—deceptive name.
- Specification for the programmer.
- Defines a "memory order" for each multiprocessor execution.
- Rules ("axioms") that hold between the execution and the memory order.
- If a memory order exists that conforms to the rules, then the execution is a valid TSO execution.

TSO Memory Model



TSO Memory Model

- No concept of real-time.
- Defined in terms of:
  - Per-strand *program order* of executed instructions (including memory operations) and
  - System-wide *memory order* of memory operations.
- Memory order is constrained by program order according to TSO rules.
- Load value is defined in terms of memory order, program order and value of stores.

TSO Memory Model

- A load program-ordered before another load is also memory-ordered before that load:

$$\forall l_a l_b . l_a <_p l_b \Rightarrow l_a <_m l_b \tag{1}$$

- A store program-ordered before another store is also memory-ordered before that store:

$$\forall s_a s_b . s_a <_p s_b \Rightarrow s_a <_m s_b \tag{2}$$

- A load program-ordered before any store is also memory-ordered before that store:

$$\forall l s . l <_p s \Rightarrow l <_m s \tag{3}$$

- The value of a load to address a is the value of the latest store in memory order that is either program-ordered before the load or memory-ordered before the load:

$$\text{Value}(l_a) = \text{Value}(\underset{m}{\text{Max}}(\{s_a : s_a <_m l_a\} \cup \{s_a : s_a <_p l_a\}) \tag{4}$$

Strand Hardware Model

- For "strand" read "processor."
- Each strand, coupled with its program, can be considered as a finite state machine that inputs through memory loads and outputs through memory stores.
- The high-level (programmer's) view of a strand executes instructions from its program in order.
- The implementation of a strand executes instructions (calculates register updates etc.) out-of-order. This out-of-order execution is invisible to the programmer.
- In particular, memory accesses must appear to occur in TSO order, even when data is transferred out-of-order.
- Instructions retire in order (state change rendered irrevocable).

Strand Abstract Model

- In the verification (simulation) environment, the strand hardware model is run in parallel with an in-order model of Sparc processor behavior.
- In-order model is stepped when instructions retire (irrevocably update architectural state of strand).
- Golden memory model supplies data values for retired loads and accepts retired stores.

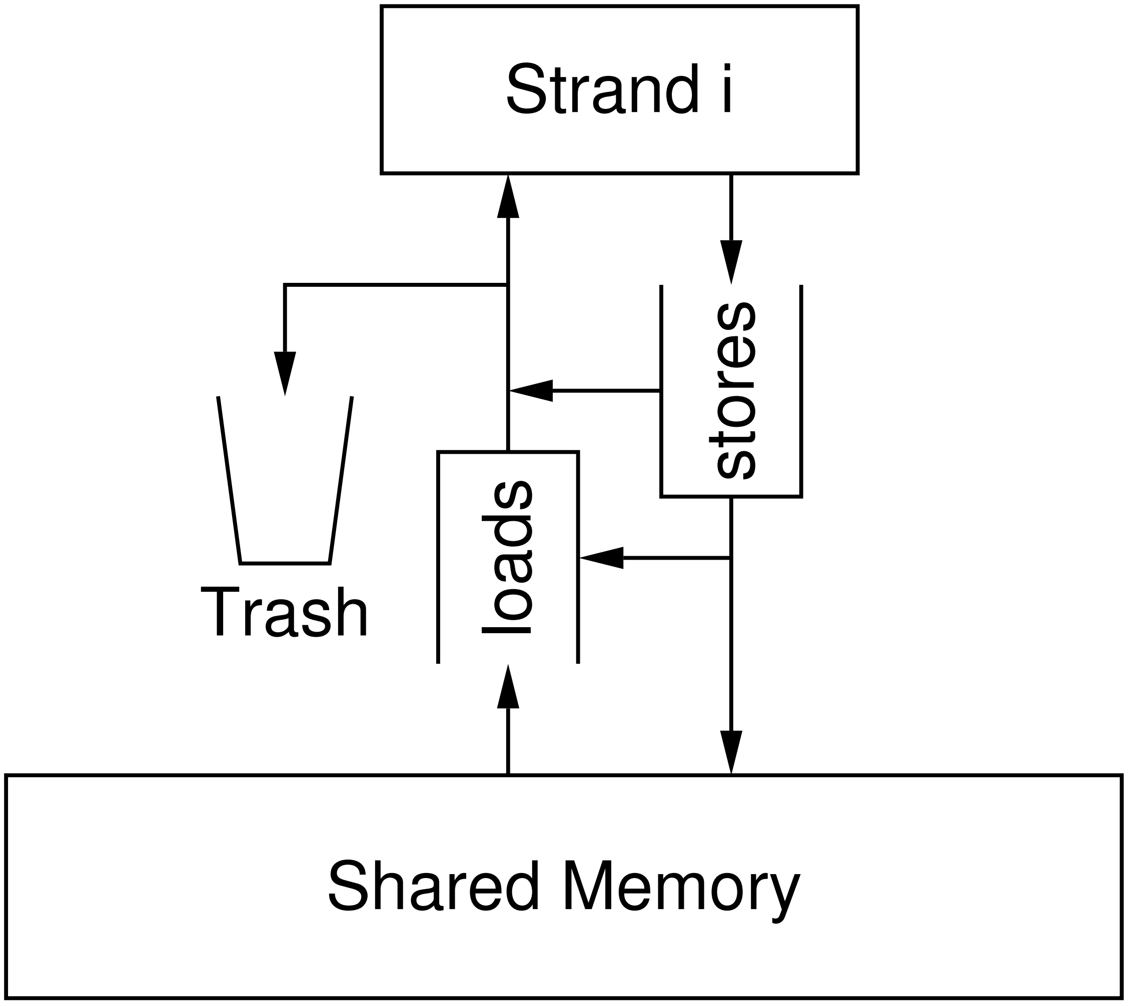
Loads and Stores

- Two events associated with each load or store:
  1. Retiring, when the architectural state of the strand is irrevocably updated.
  2. Committing, when a store can affect other strand’s loads, or when a load ceases to see other strand’s stores.
- Stores retire before committing.
- Loads commit before retiring.

Problem

- A hardware load reads memory and bypasses from older (including unretired) uncommitted stores before load retires.
- Until a load retires, it is speculative and may be discarded rather than retired.
- Strand abstract model doesn’t supply stores until retirement.
- Strand abstract model doesn’t accept data for loads until the loads retire.

Golden Memory Model



Golden Memory Model

- Model of architecture's implementation of TSO.
- Demonstrably implements TSO (by informal mathematical proof).
- No caches.
- Memory order compatible with real-time order across all strands (property of system architecture).
- Global Memory updated in real time.

Golden Memory Model

- Loads commit (read memory) before retirement.
- Stores commit (store to memory) after retirement.
- Golden memory cannot see older unretired stores when loads commit, because stores are not signaled to golden memory until retirement.
- Golden Memory doesn't need to finally determine load data value until load retires, when it can see the older stores that have not yet committed.
- This leaves the stores that committed between the committing of the load and the load's retirement, which are taken into account by bypassing commiting stores to the load snapshots.

Coupling Models Together

- Events supplied by hardware model:
  - Instruction retirement (anonymous—just step abstract strand model).
  - Committing of loads (taking snapshot). Can involve some adventurous probing of the hardware design.
  - Abandonment of speculative loads (discarding snapshot). Similarly adventurous probing.
  - Committing of stores.
  - Discarding of load snapshots.
- Events supplied by Strand abstract model:
  - Retirement of loads.
  - Retirement of stores.

Conclusion

- Much stronger checking of design, by verification against designer intent rather than TSO specification.
- Avoids complex analysis of simulation logfile to search for a valid memory order.
- Allows strand model to run in synchronism with hardware model running arbitrary multiprocessor programs.
- Close examination of implentation details to implement probing.
- No direct use of mechanized formal methods in this work.
- Formal methods education invaluable for generating insight into how hardware is intended to work.
- Depends on rigorous but informal reasoning about how the hardware model implements TSO.

Further Work

- If caches maintain coherence in non-real-time order, then golden model can get much more complex and may require formal verification against TSO.