Using Lamport Clocks to Reason About Relaxed Memory Models

Anne E. Condon, Mark D. Hill, Manoj Plakal, Daniel J. Sorin

Computer Sciences Department
University of Wisconsin-Madison
{condon,markhill,plakal,sorin}@cs.wisc.edu

*. This work is supported in part by Wright Laboratory Avionics Directorate, Air Force Material Command, USAF, under grant #F33615-94-1-1525 and ARPA order no. B550, National Science Foundation with grants MIP-9225097, MIPS-9625558, CCR 9257241, and CDA-9623632, a Wisconsin Romnes Fellowship, and donations from Sun Microsystems.
Motivation & Problem

- Shared-memory multiprocessors must be correct!

- Correct == Implementing a memory consistency model
  - Most memory consistency models define correctness through (off-line) existence of a total/partial order on memory references
  - E.g., Sequential Consistency (SC) and TSO require a total order. Alpha requires a partial order.

- Existence of required order is not evident

- Modern high-performance implementations:
  - Cache coherence (e.g., snooping, directories)
  - Very aggressive (e.g., out-of-order processors, hierarchies of non-blocking caches, interleaved memories)
  - Optimizations $\Rightarrow$ memory operations re-ordered/non-atomic
Our Proposal: Use Lamport Clocks

• Most consistency models require (off-line) existence of order

• Borrow from Lamport’s logical clocks:
  • Assign timestamps to memory references in any execution
  • Prove that every load returns value written by previous store to same address in timestamp ordering
  • No hardware added!

• Since proof works for orderings created by any execution:
  • Every execution satisfies consistency model
  • Therefore, system satisfies consistency model
Alternative Methods

- Informal Techniques:
  - Extensive simulation and stress testing
  - Thought experiments (a.k.a. hand-waving)

- Formal Techniques:
  - State-space search of finite-state coherence engines
  - Verification using theorem-provers

- Current promising research in formal systems:
  - Symbolic states [Pong & Dubois, SPAA’93]
  - Aggregation of transactions [Park & Dill, SPAA’96]
  - Term rewriting [Shen & Arvind]
Where Our Scheme Fits In

Formal Power Enough for:
- Formal methods
- Lamport Clocks
- Holy Grail

Ease of Use Enough For:
- A = Architects
- I = Implementors
- V = Verifiers

Formal methods

Lamport Clocks

Holy Grail

Informal methods

A = Architects
I = Implementors
V = Verifiers

A + I + V
A + I
A
V
I + V
A + I + V
Outline

• Motivation, Problem & Summary
• Background
• Memory Consistency Models
• Lamport Clocks
• Our Verification Technique
• Summary
Memory Consistency Models

- SC is like multiprogrammed uniprocessor

- For each execution
  - There exists a total order
  - That respects the program order of each processor
  - Loads return the value of the last store in that order
Memory Consistency Models (cont’d)

**SPARC TSO** - allows FIFO write buffers between CPU and cache

- For each execution
  - There exists a total order
  - That respects the program order of each processor
  - Loads return the value of the last store in that order *unless the store is in (an abstraction of) the write buffer*

**COMPAQ Alpha** - allow re-ordering between memory barriers

- For each execution
  - There exists a *partial* order
  - That respects *a subset of* program order of each processor
  - Loads return the value of the last store in that order
Directory Coherence Protocol Example

• Events at a processor:
  • Memory operations (LD, ST)
  • Coherence transactions (Get-Exclusive, Invalidate)
**Lamport Clocks (CACM, 1978)**

- Problem: notion of “happens before” in a distributed system
- Solution: Use logical clocks (counters) at each node
- Update clocks respecting causality and real-time local order

<table>
<thead>
<tr>
<th>Physical time</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Logical order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 A</td>
<td>2.1 B</td>
<td>1.2 C</td>
<td>1.3 E</td>
<td>1.1 A</td>
</tr>
<tr>
<td>2.1 B</td>
<td></td>
<td>2.1</td>
<td>2.3 F</td>
<td>1.2 C</td>
</tr>
<tr>
<td>1.2 C</td>
<td></td>
<td>3.2 D</td>
<td>3.3 G</td>
<td>1.3 E</td>
</tr>
<tr>
<td>3.2 D</td>
<td></td>
<td></td>
<td>4.3 H</td>
<td>2.1 B</td>
</tr>
<tr>
<td>3.3 G</td>
<td></td>
<td></td>
<td></td>
<td>2.3 F</td>
</tr>
<tr>
<td>4.3 H</td>
<td></td>
<td></td>
<td></td>
<td>3.2 D</td>
</tr>
<tr>
<td>5.3 I</td>
<td></td>
<td></td>
<td></td>
<td>3.3 G</td>
</tr>
<tr>
<td>5.3 I</td>
<td></td>
<td></td>
<td></td>
<td>4.3 H</td>
</tr>
<tr>
<td>6.1 J</td>
<td></td>
<td></td>
<td></td>
<td>5.3 I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.1 J</td>
</tr>
</tbody>
</table>
Outline

• Motivation, Problem & Summary

• Background

• Our Verification Technique
  • Our extensions to Lamport’s solution
  • Applying our technique
  • Case study of SC system
  • Extending technique to other memory models

• Summary
Lamport Clocks

• Why not reason about correctness by:
  (1) Think of every dynamic load/store getting a timestamp
  (2) Have memory hand out the timestamps
  (3) Show every load returns an appropriate value

• Can do (1) and (3), but real implementations can’t do (2)
  • Don’t have a single physical memory
  • Often have distributed coherent caches
## Comparing Our Solution to Lamport’s Solution

<table>
<thead>
<tr>
<th></th>
<th>Lamport’s Solution</th>
<th>Our Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>Local Events and Messages</td>
<td>Memory Operations and Coherence Transactions</td>
</tr>
<tr>
<td>Logical Clock</td>
<td>2-tuple</td>
<td>3-tuple</td>
</tr>
<tr>
<td>Increment</td>
<td>Local Events (in real-time order)</td>
<td>Loads/Stores (in program order)</td>
</tr>
<tr>
<td>Update</td>
<td>Message Receives</td>
<td>Coherence Message Receives</td>
</tr>
<tr>
<td>Break Ties</td>
<td>Node ID</td>
<td>Node ID</td>
</tr>
</tbody>
</table>
A (brief) Case Study: A Directory Protocol

- SC system using an SGI Origin 2000-like invalidation-based directory protocol:

- How we timestamp Loads/Stores:
  - Loads/Stores are *bound* to the coherence transactions that ensured the appropriate coherence state

  - Global time = \( \max( \text{global time of transaction to which load/store is bound,} \)
    \( \text{global time of previous load/store in program order} ) \)

  - Local time = 1 + local time of previous load/store in program order with same global time,
    1, if this is the first.
Case Study (contd)

- Example of timestamping at a single node:

<table>
<thead>
<tr>
<th>Physical time</th>
<th>Lamport time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get-Exclusive A</td>
<td>1.0.1 Get-Exclusive A</td>
</tr>
<tr>
<td>LD A</td>
<td>1.1.1 LD A</td>
</tr>
<tr>
<td>Get-Shared B</td>
<td>1.2.1 ST A</td>
</tr>
<tr>
<td>ST A</td>
<td>2.0.1 Get-Shared B</td>
</tr>
<tr>
<td>LD B</td>
<td>2.1.1 LD B</td>
</tr>
</tbody>
</table>
Example of timestamping across nodes:

<table>
<thead>
<tr>
<th>Physical Time</th>
<th>Lamport Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>N₁</strong></td>
<td><strong>N₂</strong></td>
</tr>
<tr>
<td>send GETX A</td>
<td>store B</td>
</tr>
<tr>
<td>bind load A</td>
<td>1.11.2</td>
</tr>
<tr>
<td>recv INV A</td>
<td>2.0.2</td>
</tr>
<tr>
<td>send ack</td>
<td></td>
</tr>
<tr>
<td>recv ACK</td>
<td>perform load A, invalidate A</td>
</tr>
<tr>
<td>store A</td>
<td>3.1.1</td>
</tr>
</tbody>
</table>
**TSO**

- For each execution
  - There exists a total order
  - That respects the program order of each processor
  - Loads return the value of the last store in that order *unless the store is in (an abstraction of) the write buffer*

- *Wisconsin TSO*
  - Break up store into Store-Private and Store-Public
    - Store-Private to write buffer
    - Store-Public to cache
  - Loads return the value of the last Store-Public or the value of the last Store-Private if the Store-Private is at that processor.
  - TSO proof done for SC protocol with added FIFO write buffer
Alpha

- For each execution
  - There exists a *partial* order
  - That respects *a subset of* the program order of each processor
  - Loads return the value of the last store in that order

- *Wisconsin Alpha*
  - Partial order only respects
    - order between references to the same address
    - memory barriers
  - Arbitrarily create a total order out of the partial order
  - Loads return the value of the last store in this total order
  - Alpha proof done for SC protocol with coalescing write buffer
Summary

• Memory consistency model requires (off-line) existence of order

• Construct order on-line by assigning timestamps

• Prove requirements of model in constructed order

• Since proof works for any execution:
  • Every execution is correct
  • Therefore, system is correct

• In our view
  • Ease of use: Formal Verification < Lamport < Informal
  • Formal Power: Informal < Lamport < Formal Verification
Future Work

- Other memory systems (clusters of SMPs)
- Systems with I/O
- Handling deadlock, starvation, & live-lock
- A good way of formally specifying protocols
- Automating the whole process (thereby losing our jobs)