The Semantics of x86-CC Multiprocessor Machine Code

Susmit Sarkar
Computer Laboratory
University of Cambridge

Joint work with:
Peter Sewell, Scott Owens, Tom Ridge, Magnus Myreen (U.Cambridge)
Francesco Zappa Nardelli, Jade Alglave, Thomas Braibant (INRIA)

POPL, January 2009
Relaxed memory in multiprocessors

• Traditional assumption (concurrent algorithms, semantics, verification): single shared memory state

• Falsified on actual multiprocessors:
  – local store buffers,
  – shadowing register files,
  – hierarchies of caches
  – …

• Only approximately consistent, or relaxed view of memory
Relaxed memory: makes reasoning hard

Consider (adapted from Intel manuals):

<table>
<thead>
<tr>
<th>Shared locations x and y initially 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_0 )</td>
</tr>
<tr>
<td>( x \leftarrow 1 )</td>
</tr>
<tr>
<td>( r_0 \leftarrow y {0} )</td>
</tr>
</tbody>
</table>

- Observed (630 / 100,000 in one test run): dual core Intel Core2
- Not a new problem (Dubois et al: early ’80s; IBM S/370)
- …but still not well understood (cf. Linux barrier semantics discussion)
- Needed: precise description of real architecture’s memory model
Relaxed memory: makes reasoning hard

Consider (adapted from Intel manuals):

<table>
<thead>
<tr>
<th></th>
<th>( P_0 )</th>
<th>( P_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \leftarrow 1 )</td>
<td>( y \leftarrow 1 )</td>
<td></td>
</tr>
<tr>
<td>( r_0 \leftarrow y {0} )</td>
<td>( r_1 \leftarrow x {0} )</td>
<td></td>
</tr>
</tbody>
</table>

- Observed (630 / 100,000 in one test run): dual core Intel Core2
- Not a new problem (Dubois et al: early ’80s; IBM S/370)
- …but still not well understood (cf. Linux barrier semantics discussion)
- Needed: precise description of real architecture’s memory model
Actual hardware, and “architecture”

Physical hardware:

• can run programs on them
• particular h/w designs, e.g. Intel Core 2 Duo E6300
• internally well-defined and well-understood (but secret)

Architecture:

• abstraction “sufficient” to program on:
  – not the whole truth, *and not wholly the truth*
• informal prose descriptions
• tension: reveal enough for programmers, but without constraining future design, or reveal implementation, or exposing to liability
Imprecise descriptions: causality

Architecture defined by processor manuals, which are informal prose, e.g.

“Inel 64 memory ordering ensures transitive visibility of stores — i.e. stores that are causally related appear to execute in an order consistent with the causal relation”

and explained by litmus tests, e.g. (from Intel manuals):

<table>
<thead>
<tr>
<th>P₀</th>
<th>P₁</th>
<th>P₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>x ← 1</td>
<td>r₀ ← x  {1}</td>
<td>r₁ ← y  {1}</td>
</tr>
<tr>
<td>y ← 1</td>
<td>r₂ ← x  {0×}</td>
<td></td>
</tr>
</tbody>
</table>
Capturing “causality”: happens-before

Our model: postulate a transitive relation “happens before”, which is a transitive closure of:

- “read-from” edges from stores to loads, where the load reads from that store;
- preserved parts of program order (e.g. stores are not reordered with stores);
- obvious intra-instruction dependencies (e.g. INC: a load followed by a store);
- edges due to coherence conditions (e.g. stores to the same location have a total order)

... and insist all threads observe memory accesses consistent with it
Different threads have different view orders

From AMD manual:

Independent reads of independent writes:

<table>
<thead>
<tr>
<th></th>
<th>P₀</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>← 1</td>
<td>y ← 1</td>
<td>r₀ ← y {1}</td>
<td>r₂ ← x {1}</td>
</tr>
<tr>
<td>r₁</td>
<td>← x {0}</td>
<td>r₃ ← y {0}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Shared locations x and y initially 0

Our formalization:

Each processor has a different view order of relevant events (its own events, all memory writes, but not others’ reads)
Our paper

• Formalization of memory model in HOL (following manuals of Jul ’08)
  – *Write back* memory (userspace and common kernel code)
  – No page tables and exceptions

• Connect memory semantics with instruction semantics

• Testing the semantics
  – Symbolic execution through model
  – Testing on real hardware

• Mechanized metatheory about the model
  – There is an equivalent, more operational, model
  – “Data-race free” programs have sequential behaviour
The manuals (and x86-CC) are not sound for hardware (pointed out by Paul Loewenstein)

**Model:** per-location coherence condition (*P6: Stores to the same location have a total order*)

**(Very) simplified implementation:**

Each processor has a local store buffer (FIFO) for its stores. The processor can read this, but other processors cannot, yet.

⚠️ **Observable consequences:** do buffers maintain the coherence property?

Manuals suggest they do (and so does our model)
Real hardware does not (observed rarely, but reproducibly)
## Intel/AMD memory model descriptions

<table>
<thead>
<tr>
<th>Date</th>
<th>Source</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov 2006</td>
<td>Intel manuals, rev 22</td>
<td>Extremely vague</td>
</tr>
<tr>
<td>Aug 2007</td>
<td>Intel white paper v1.0</td>
<td>Moderately clear except for “causality” (interpreted in x86-CC)</td>
</tr>
<tr>
<td></td>
<td>AMD manual, rev 3.14</td>
<td>Unsound w.r.t. current hardware</td>
</tr>
<tr>
<td></td>
<td>Intel manuals, rev 26</td>
<td>Arguably too weak for programmers</td>
</tr>
<tr>
<td>Sep 2007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feb 2008</td>
<td>Intel manuals, rev 26</td>
<td>Somewhat less clear (no clear interpretation of causality)</td>
</tr>
<tr>
<td>Nov 2008</td>
<td>Intel manuals, rev 29</td>
<td>Sound (as far as we know) w.r.t current hardware</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Surprisingly weak</td>
</tr>
</tbody>
</table>
Summary and moralizing

Formal models of weak memory needed for real architectures

Do not trust any model, *even one in official manuals*, unless (at least):

- Fully formal model (in a proof assistant, preferably)
- Checked to give expected behaviour on example programs
- Extensively tested w.r.t real hardware
- Proved to be useful by metatheoretic investigations
- Proved to be useful by writing programs in it
- Validated by discussions with experts

More work needed!
Thank you!
Rev 26 model is unsound

[From Paul Loewenstein]

<table>
<thead>
<tr>
<th>Shared locations x and y initially 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>$P_0$</td>
</tr>
<tr>
<td>$x \leftarrow 1$</td>
</tr>
<tr>
<td>$r_0 \leftarrow x {1}$</td>
</tr>
<tr>
<td>$r_1 \leftarrow y {0}$</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

- Disallowed on any reasonable interpretation of IWP model
- Disallowed on x86-CC
- Observed ($\sim 1$ in $10^6$ times) on a Core 2 Duo
Rev 29 model is too weak

Delete *P6*: *Stores to the same location have a total order*

Add *P6’*: *Any two stores are seen in a consistent order by processors other than those performing the stores*

<table>
<thead>
<tr>
<th>Shared location x initially 0</th>
</tr>
</thead>
</table>
| $\begin{array}{c|c|}
| P_0 & P_1 \\
| \hline
| x \leftarrow 1 & x \leftarrow 2 \\
| r_0 \leftarrow x \{2\} & r_1 \leftarrow x \{1\} \\
| \end{array}$ |

- Disturbing for programmers
- Not observed on real hardware (not expected to)