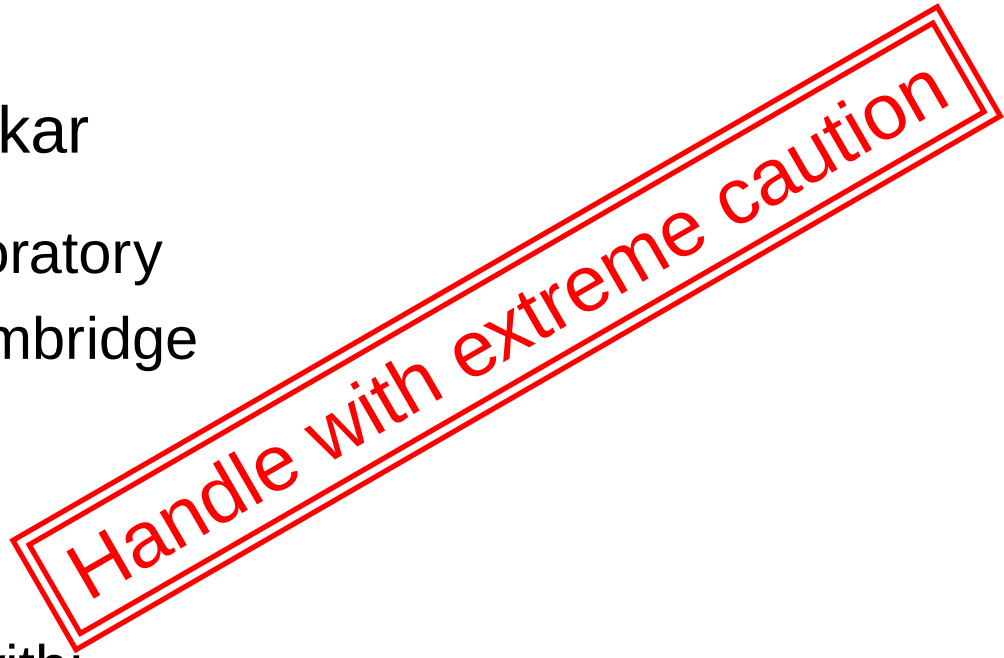
The Semantics of x86-CC Multiprocessor Machine Code

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

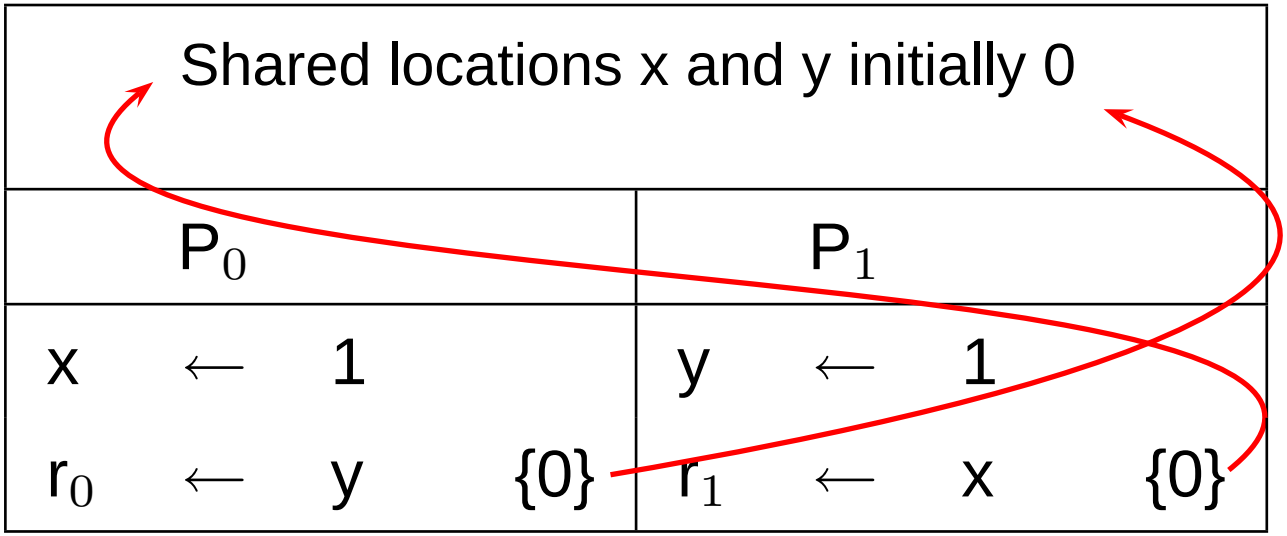
Consider (adapted from Intel manuals):

Shared locations x and y initially 0	
P_0	P_1
$x \leftarrow 1$	$y \leftarrow 1$
$r_0 \leftarrow y$ {0}	$r_1 \leftarrow x$ {0}

- 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):



- 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, $e.g.$

> *"Intel 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):

Shared locations x and y initially 0		
P_0	P_1	P_2
x $\leftarrow$ 1	$r_0 \leftarrow$ x {1} y $\leftarrow$ 1	$r_1 \leftarrow$ y {1} $r_2 \leftarrow$ x {0×}

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:

Shared locations x and y initially 0			
P_0	P_1	P_2	P_3
$x \leftarrow 1$	$y \leftarrow 1$	$r_0 \leftarrow y$ {1} $r_1 \leftarrow x$ {0}	$r_2 \leftarrow x$ {1} $r_3 \leftarrow y$ {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

Nov 2006	Intel manuals, rev 22	Extremely vague
Aug 2007	Intel white paper v1.0	Moderately clear except for “causality” (interpreted in x86-CC)
Sep 2007	AMD manual, rev 3.14	Unsound w.r.t. current hardware
Feb 2008	Intel manuals, rev 26	Arguably too weak for programmers
Nov 2008	Intel manuals, rev 29	Somewhat less clear (no clear interpretation of causality)
		Sound (as far as we know) w.r.t current hardware
		Surprisingly weak

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]

Shared locations x and y initially 0	
P_0	P_1
x ← 1	x ← 2
r_0 ← x {1}	y ← 2
r_1 ← y {0}	
Finally, x = 1	

- Disallowed on any reasonable interpretation of IWP model
- Disallowed on x86-CC
- Observed (~ 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*

Shared location x initially 0	
P_0	P_1
$x \leftarrow 1$	$x \leftarrow 2$
$r_0 \leftarrow x$ {2}	$r_1 \leftarrow x$ {1}

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