Understanding POWER multiprocessors

Susmit Sarkar[1] Peter Sewell[1]
Jade Alglave[2,3] Luc Maranget[3]
Derek Williams[4]

[1]University of Cambridge [2]Oxford University [3]INRIA [4]IBM

June 2011

Programming shared-memory multiprocessors

No Sequential Consistency (SC) *and not since 1972*

But what *do* we get?

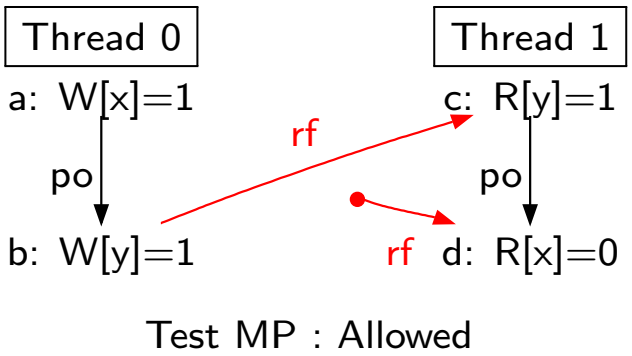
"Relaxed Memory", differing on different architectures:

- x86, SPARC — Relatively strong, better understood;
- POWER/ARM — Weaker, widely used, not widely understood;
- High-level languages — Different again

  Models informed by POWER/ARM features

Relaxed memory behaviour: Message Passing

Thread 0	Thread 1
`x = 1`	`while (y == 0)`
`y = 1`	`  {};`
	`r = x` (read 0?)



Test MP : Allowed

- Forbidden on SC, or x86-TSO
- Allowed on POWER ($\sim$ 1e6 in 2e9 on a POWER7)

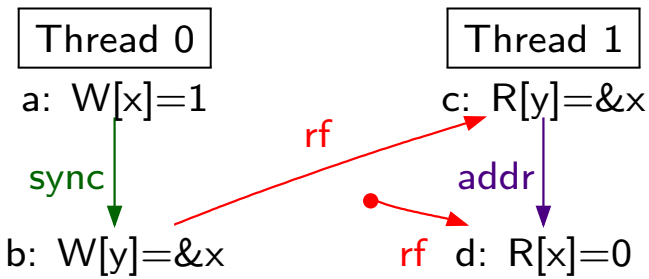
What is going on?

Visible Microarchitectural Effects:

- Out-of-order, and Speculative Execution
- Buffering of Stores and Loads
- Topology of Interconnection

Enforcing order where needed

Thread 0	Thread 1
`x = 1`	`while (y == 0)`
`sync()`	`  {};`
`y = &x`	`r = *y` (read 0?)



Test MP+sync+addr : Forbidden

- sync: writes in order
  - On the same thread; and
  - When propagating to other threads
- Dependency: reads in order
  - Later read not issued until resolved

POWER model in general: . . . How do we find out?

Architecture Manuals:

- Ambiguous prose

> *"all that horrible horribly incomprehensible and confusing [...] text that no-one can parse or reason with — not even the people who wrote it"*
>
> — Anonymous Processor Architect, 2011[1]

Concrete Implementation:

- Proprietary
- Extremely complex, and too low-level
- Changes across generations

[1] Not Derek Williams!

Our work

Rigorous Architecture

Do lots of tests (borrow, handwrite, autogenerate) on Power G5, 6, and 7

Discuss with designers/architects

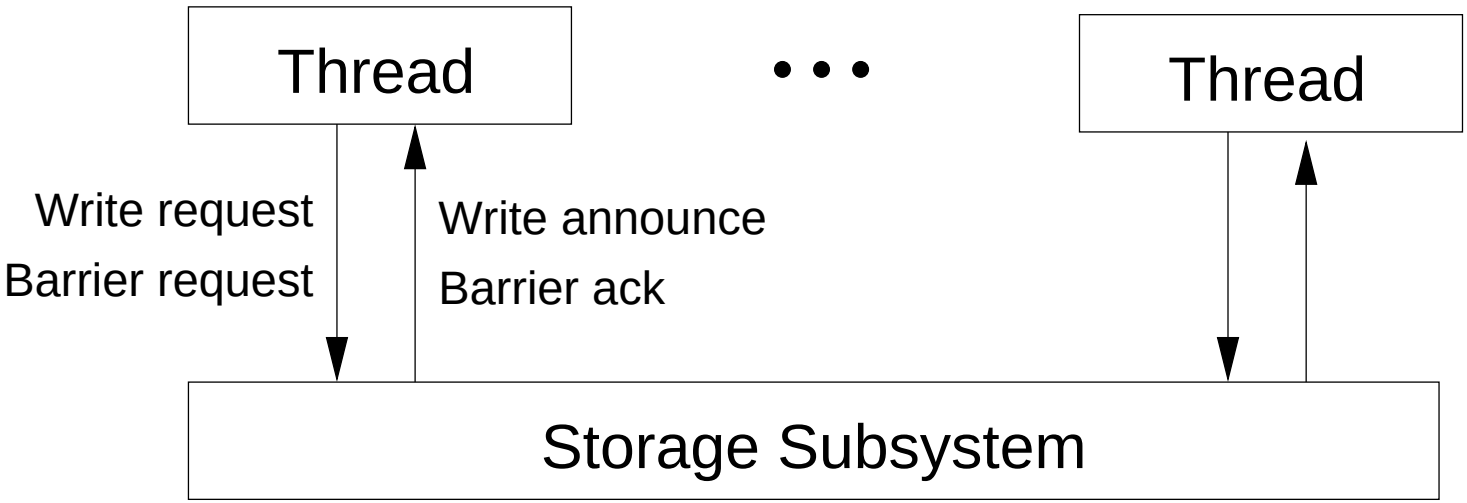
Develop an abstract operational model

- Matches observed behaviour (intentionally looser in some aspects)
- Simple enough to understand

Only considering application and common OS code, with no unaligned/mixed-size accesses (no self-modifying code, device memory, or page table changes)

The model structure

Overall structure:



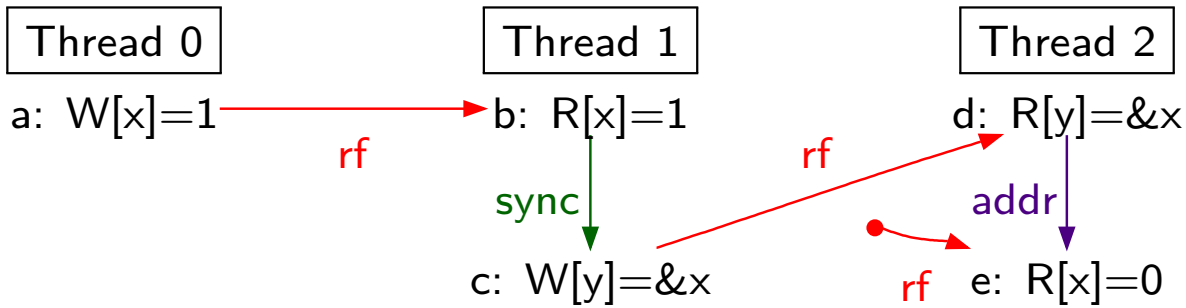
- Some aspects are thread-only, some storage-only, some both
- Threads and Storage Subsystem: Abstract state machines

  Speculative execution in Threads;
  Topology-independent Storage Subsystem
- Formally: transitions, guarded by preconditions, change state, and synchronize with each other

Cumulativity: Programming on many threads

Thread 0	Thread 1	Thread 2

`x = 1`	`while (x == 0)`
	`  {};`
	`sync()`
	`y = &x`
		`while (y == 0)`
		`  {};`
		`r = *y` (read 0?)



Test WRC+sync+addr : Forbidden

The sync is *cumulative*: it keeps (a) and (c) in order for all threads
Flipping the dependency and barrier does *not* recover SC

Model Excerpt

Propagate write to another thread

The storage subsystem can propagate a write w (by thread tid) that it has seen to another thread tid', if:

- the write has not yet been propagated to tid';
- w is coherence-after any write to the same address that has already been propagated to tid'; and
- all barriers that were propagated to tid before w (in $s.events\_propagated\_to\ (tid)$) have already been propagated to tid'.

Action: append w to $s.events\_propagated\_to\ (tid')$.

Explanation: This rule advances the thread tid' view of the coherence order to w, which is needed before tid' can read from w, and is also needed before any barrier that has w in its "Group A" can be propagated to tid'.

Overall Model Size

Explanation in $\sim$3 pages of prose

- Microarchitectural intuitions
- No extraneous concrete details

$\sim$2500 lines of machine-processed math

- In LEM [ITP'11], a simple new semantic metalanguage
- Can extract executable code, and theorem-prover code

Validating the model

- Extract executable code from definition, exhaustively enumerate possible behaviours of tests
- Run many iterations of tests on real hardware (Power G5, 6, 7)

Excerpt of results:

Test	Model	POWER 6	POWER 7
WRC+sync+addr	Forbid	ok 0 / 16G	ok 0 / 110G
WRC+data+sync	Allow	ok 150k / 12G	ok 56k / 94G
PPOCA	Allow	unseen 0 / 39G	ok 62k / 141G
PPOAA	Forbid	ok 0 / 39G	ok 0 / 157G
LB	Allow	unseen 0 / 31G	unseen 0 / 176G

- Agreed with key IBM Power designers/architects

Validating the model

- Extract executable code from definition, exhaustively enumerate possible behaviours of tests
- Run many iterations of tests on real hardware (Power G5, 6, 7)

Excerpt of results:

Test	Model	POWER 6	POWER 7
WRC+sync+addr	Forbid	ok 0 / 16G	ok 0 / 110G
WRC+data+sync	Allow	ok 150k / 12G	ok 56k / 94G
PPOCA	Allow	unseen 0 / 39G	ok 62k / 141G
PPOAA	Forbid	ok 0 / 39G	ok 0 / 157G
LB	Allow	unseen 0 / 31G	unseen 0 / 176G

- Agreed with key IBM Power designers/architects

Validating the model

- Extract executable code from definition, exhaustively enumerate possible behaviours of tests
- Run many iterations of tests on real hardware (Power G5, 6, 7)

Excerpt of results:

Test	Model	POWER 6	POWER 7
WRC+sync+addr	Forbid	ok 0 / 16G	ok 0 / 110G
WRC+data+sync	Allow	ok 150k / 12G	ok 56k / 94G
PPOCA	Allow	unseen 0 / 39G	ok 62k / 141G
PPOAA	Forbid	ok 0 / 39G	ok 0 / 157G
LB	Allow	unseen 0 / 31G	unseen 0 / 176G

- Agreed with key IBM Power designers/architects

Summing up

- A mathematically precise, empirically validated, operational model of POWER
- Microarchitectural intuitions, but abstract: no implementation details

Rigorous Architecture

- Can reason about low-level code above it (static analysis tools)
- Can build on for software verification (e.g. compiler verification)
- Can use as specification to test implementations
- . . . Lots to be done!