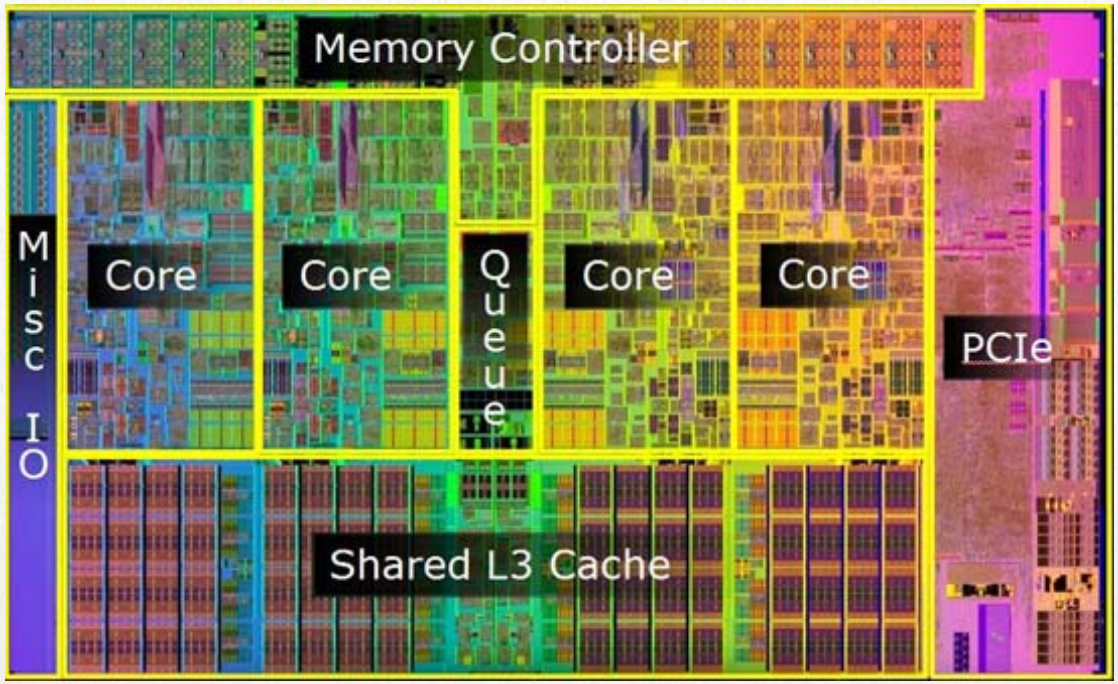
ArMOR: Defending Against Memory Consistency Model Mismatches in Heterogeneous Architectures

Zeyu Bu, Yao Jiang

11/3/2015

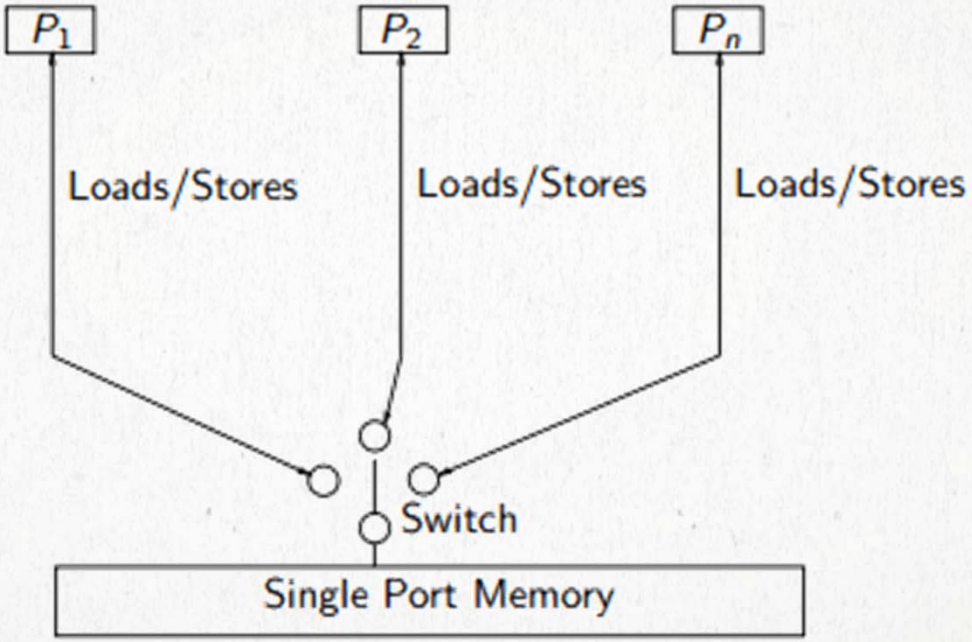
Review – Memory Consistency Model

- A ***Consistency Model*** is a contract between the software and the memory
  - It states that the memory will work correctly but only if the software obeys certain rules
  - Sequential Consistency (SC), Total Store Order (TSO) and Relaxed Memory Order (RMO).



Review - SC

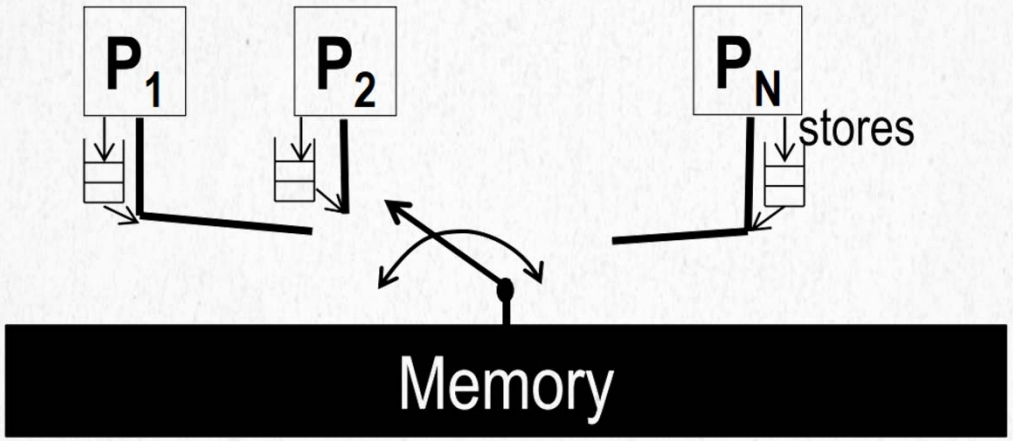
- Sequential Consistency
  - The result of any execution is the same as if the operations of all the processors were executed in some sequential order



Sequential consistency is inefficient: we want to weaken the model further

Review - TSO

- Total Store Order
  - Loads can be performed before previous stores from the same thread



Review - RMO

- Relaxed Memory Order
  - Relaxes all order! (except for accesses to the same address)
  - Provides explicit fence (barrier) instructions
  - The mask bits determine what order to enforce

If mmask = b0010, only load after store is allowed

Load after Load	Store after Load
Load after Store	Store after Store

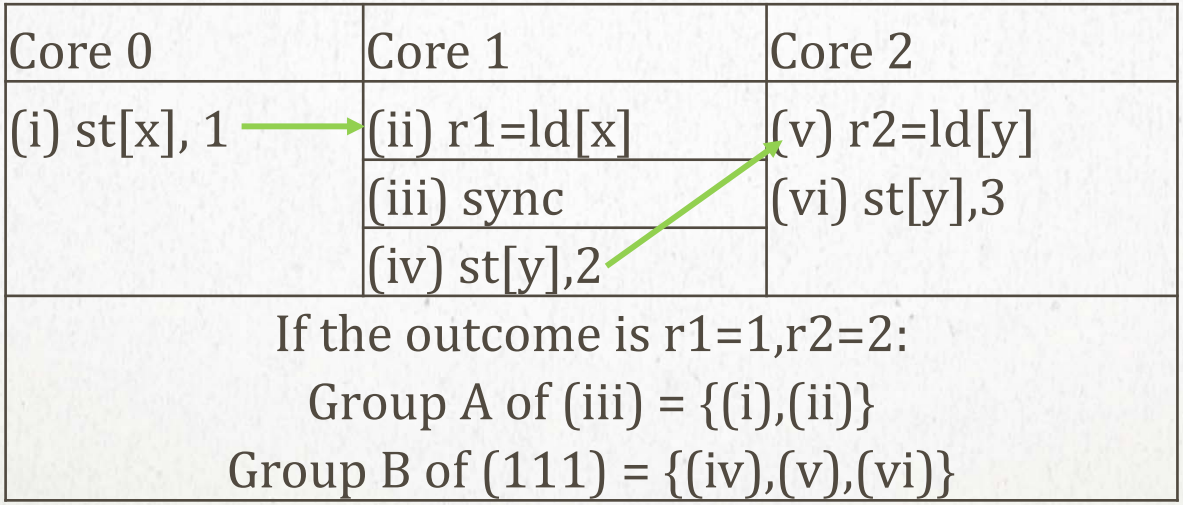
	Load	Store
Load	✓	✓
Store	—	✓

Review – Atomicity and Cumulativity

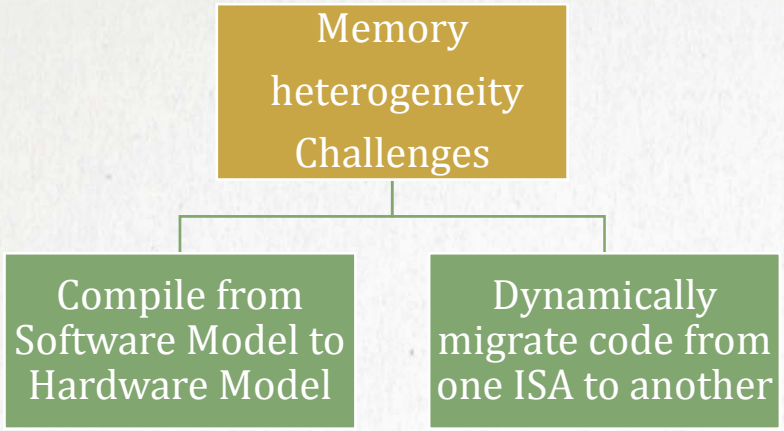
- Store Atomicity
  - **Single-Copy Store Atomicity:** Stores must become visible to all cores in the system at a single time
  - **Multi-Copy Store Atomicity:** A thread can read its own write early
- Fence Cumulativity:
  - A property where a fence may order operations from other threads.
  - Operations placed into two classes, A and B, relative to fence.
  - A-cumulativity and B-cumulativity rules to determine which class memory operations belong to.

Review - Cumulativity

- A-cumulativity (AC) requires that instructions (from any thread) that have performed prior to an access in group A are also members of group A.
- B-cumulativity (BC) requires that instructions (from any thread) that perform after a load that returns the value of a store in group B are also themselves in group B.



Contribution



ArMOR

Shim

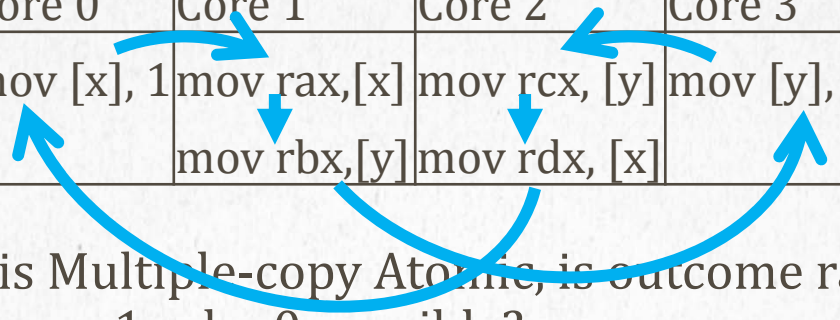
- Translate code compiled for one memory model to execute on hardware implementing a different model

Memory Ordering Specification Tables (MOST)

- Algorithmically compare two consistency models

Memory model heterogeneity

Core 0	Core 1	Core 2	Core 3
mov [x], 1	mov rax,[x] mov rbx,[y]	mov rcx, [y] mov rdx, [x]	mov [y], 1



- If this is Multiple-copy Atomic, is outcome rax=1, rbx=0, rcx=1, rdx=0 possible?

NO

Core 0	Core 1	Core 2	Core 3

stw [x], 1	lwz r1,[x]
	lwsync
	lwz r2,[y]
		lwz r3, [y]
		lwsync
		lwz r4, [x]
			stw [y], 1

- What if it is A and B Acumulative? Is outcome r1=1, r2=0, r3=1, r4=0 possible?

YES

Motivation

TRADITIONAL WAY TO DESCRIBE MEMORY CONSISTENCY MODEL

	Load	Store
Load	✓	✓
Store	—	✓

Multiple-copy Atomic

	Load	Store
Load	✓	✓
Store	—	✓

A and B Cumulative

Could you really tell the difference by such description?

MOST - Objective

- MOST is used to define not just preserved program order, but also fences or any other type of ordering enforcement mechanism.

Abb.	Description
$\checkmark_S$	Single-copy atomic
$\checkmark_M$	Multiple-copy atomic
$\checkmark_N$	Non-atomic
—	Unordered

(a) Store→store

Abb.	Description
✓	Ordered
$\checkmark_L$	Locally ordered
—	Unordered

(b) Store→load

Abb.	Descrip.
✓	Ordered
—	Unordered

(c) Other

MOST – Multiple-Copy Atomic Store

- How to describe Multiple-Copy Atomic Store using MOST?

	Load	Store
Load	✓	✓
Store	—	✓

	Load	Store
Load	✓	✓
Store	—	$\checkmark_M$

- Stores are Multiple-Copy Atomic

MOST – Per-Address Orderings

Core 0	Core 1	Core 2

st [x], 1
ld r1, [x]
ld r2, [y]
	st [y], 1	ld r3, [y]
		ld r4, [x]

TSO PPO

- Is outcome: r1=r3=1, r2=r4=0 allowed?

- According to the coherence, accesses from the **same thread** to the **same address** generally must maintain the ordering specified by program order.
- However, the same **store → load** ordering may not need to be enforced from the point of view of any **remote observers.**

MOST – Per-Address Orderings

	Ld	St
Ld	✓	✓
St	—	$\checkmark_M$

(a) TSO (partial)

- How to describe TSO PPO using MOST if the Per-Address Ordering is specified?

	Load to Diff. Address	Load to Same Address	Store
Load	✓	✓	✓
Store	—	$\checkmark_L$	$\checkmark_M$

MOST – Fence Cumulativity

Core 0	Core 1	Core 2
(i) st [x], 1	(ii) r1 = ld [x] (iii) sync (iv) st [y], 2	(v) r2 = ld [y] (vi) st [y], 3

	PO+ SA Ld	PO+ DA Ld	BC Ld	PO St	BC St
PO Ld	✓	✓	✓	✓	✓
AC Ld	✓	✓	✓	✓	✓
PO St	$✓_L$	—	—	$✓_N$	$✓_N$
AC St	—	—	—	$✓_N$	$✓_N$

Power lwsync

	PO Ld	BC Ld	PO St	BC St
PO Ld	✓	✓	✓	✓
AC Ld	✓	✓	✓	✓
PO St	✓	✓	$✓_S$	$✓_S$
AC St	✓	✓	$✓_S$	$✓_S$

Power sync

- How to describe Power lwsync and Power snc using MOST if the A-cumulativity and B-cumulativity are specified?

Comparing and Manipulating MOSTs

	PO+ SA Ld	PO+ DA Ld	PO St
PO Ld	✓	✓	✓
PO St	$✓_L$	—	$✓_M$

Refine: Step 1

	PO+ SA Ld	PO+ DA Ld	BC Ld	PO St	BC St
PO Ld	✓	✓	?	✓	?
AC Ld	?	?	?	?	?
PO St	$✓_L$	—	?	$✓_M$	?
AC St	?	?	?	?	?

- Step 1:

The first is to find the set of categories that should be used as the row and/or the column headings for the

Refine: Step 2

- Step 2:

The second step is to fill in the cells of the newly-refined MOST.

	PO+ SA Ld	PO+ DA Ld	BC Ld	PO St	BC St
PO Ld	✓	✓	✓	✓	✓
AC Ld	✓	✓	✓	✓	✓
PO St	$✓_L$	—	✓	$✓_M$	✓
AC St	✓	✓	✓	$✓_M$	$✓_M$

Comparing and Manipulating MOSTs

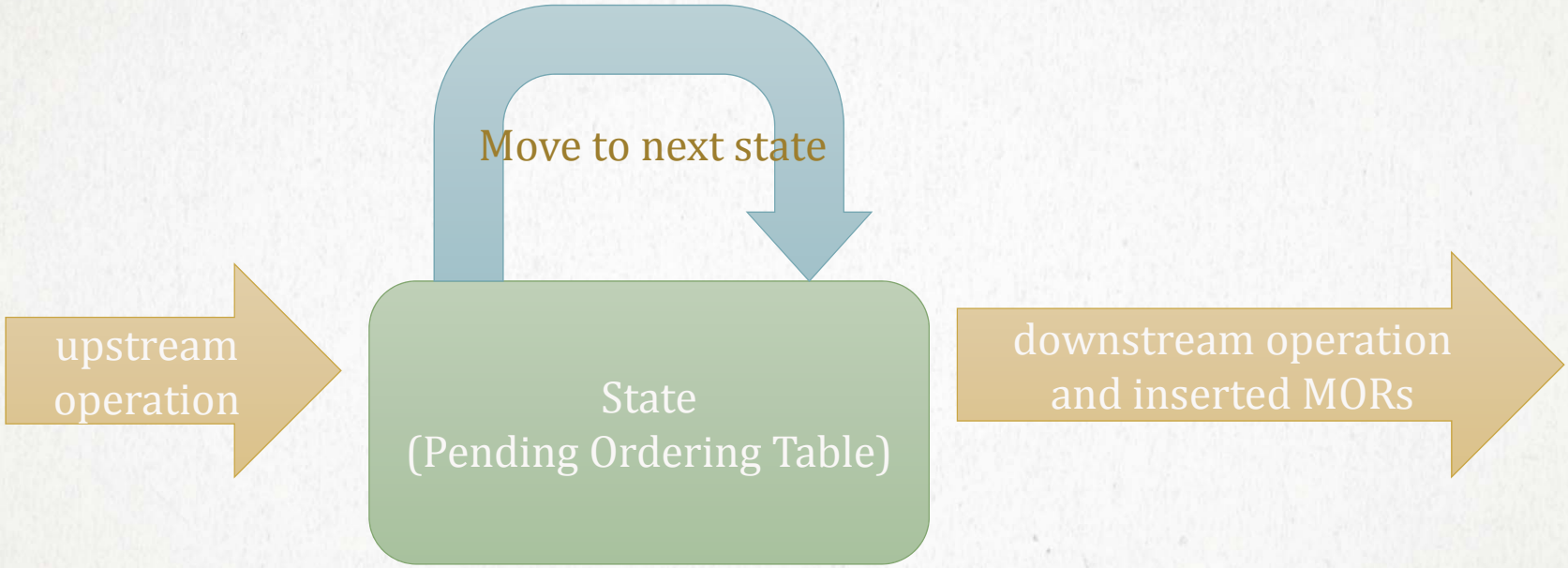
	PO+ SA Ld	PO+ DA Ld	BC Ld	PO St	BC St
PO Ld	—	—	—	—	—
AC Ld	—	—	—	—	—
PO St	—	—	✓	$\checkmark_{M-N}$	$\checkmark_{M-N}$
AC St	✓	✓	✓	$\checkmark_{M-N}$	$\checkmark_{M-N}$

Subtracting Power lwsync from (a properly refined) TSO PPO.

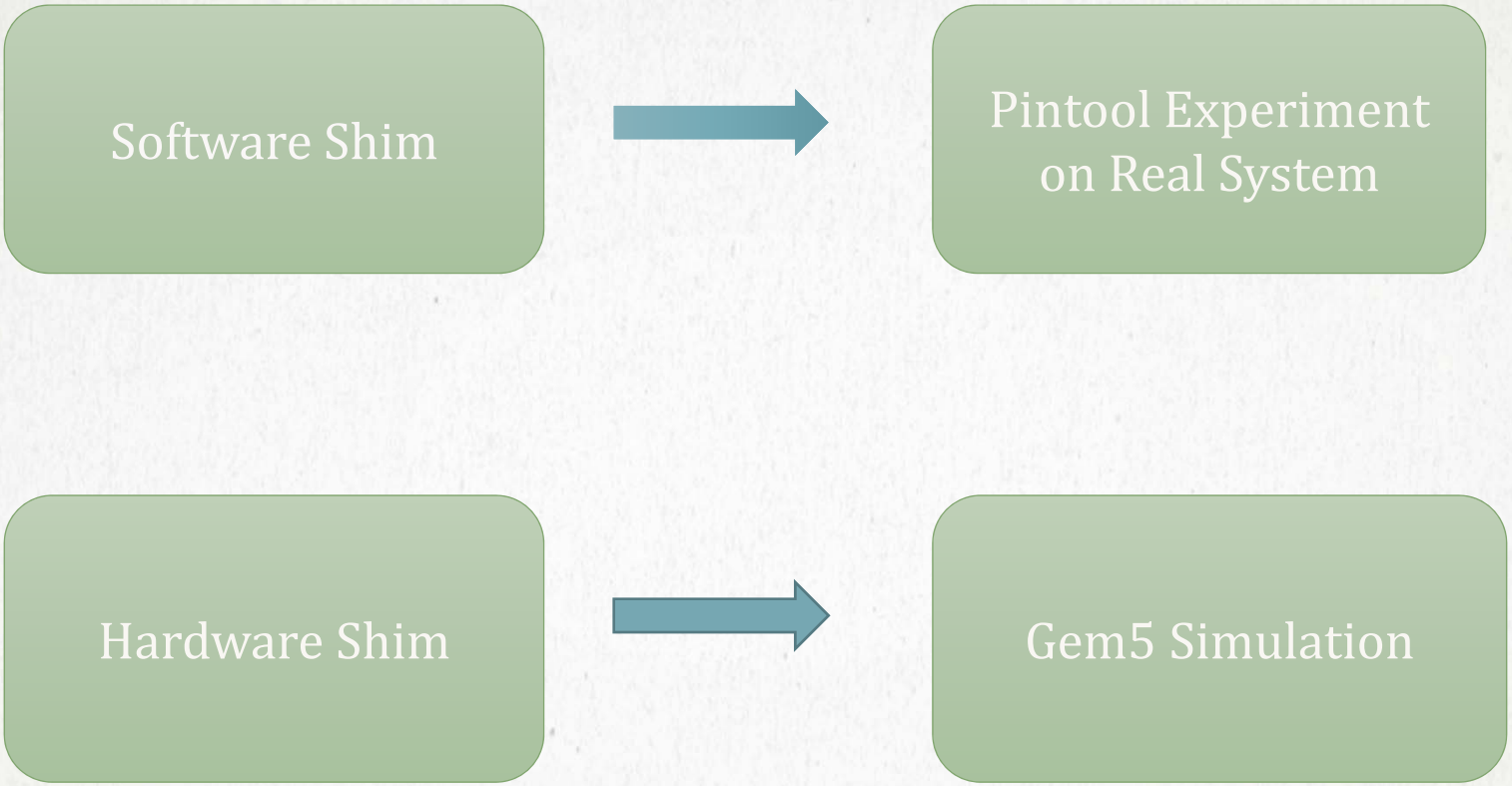
- Once two MOSTs have been refined (if necessary) into the same layout of rows and columns, then a comparison of the two can be defined by comparing each pair of corresponding cells.

Shim

- **Shim is an FSM that** translate code compiled for one memory model to execute on hardware implementing a different model

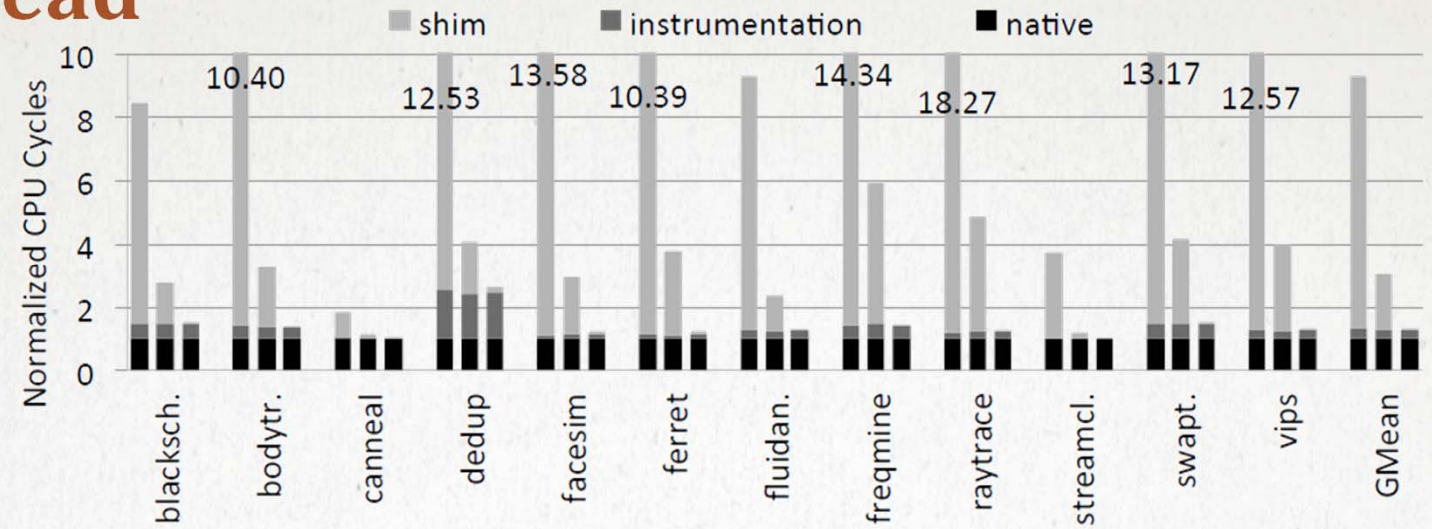


Experiment Setup



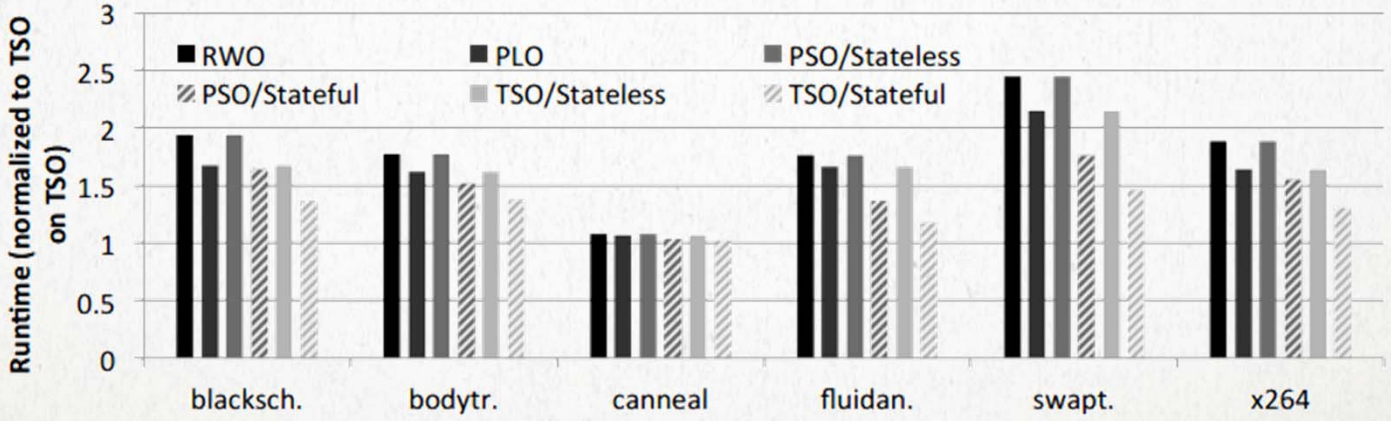
Overhead

Pintool Experiment



From left to right, the three bars represent the stateless, stateful, and ISA-assisted stateful cases, respectively.

Hardware Simulation



Simulated performance with x86-SC software and varying hardware models, normalized to x86-TSO software on x86-TSO hardware as described in the text.

Takeaway

- Architectures should provide a way to optionally make stores multiple-copy atomic.
- The more downstream MORs are available, the more intelligent the translation can be.
- ISAs and intermediate representations should maintain consistency metadata even if it is redundant with respect to preserved program order.
- Non-multiple-copy-atomic architectures cannot ignore cumulativity.

Questions?

DEBATES TOPIC

- Is it worthwhile to use ArMOR in Inter-MCM Translation?
- Other than Inter-MCM Translation, is ArMOR useful for other application area related to Memory Consistency.

Thank you for your listening !

Zeyu Bu, Yao Jiang

University of Michigan
Department of EECS
11/3/2015