Bridging the gap in the RISC-V memory models

Stefanos Kaxiras$^{1,2}$
Alberto Ros$^{1,3}$

$^1$Eta Scale AB, Sweden
$^2$Uppsala University
$^3$University of Murcia
RISC-V Memory Model

The memory consistency model:
• Specifies the values returned by every load
• Contract between the programmer and the machine

Main memory model in RISC-V:
• RVWMO: RISC-V Weak Memory Order

Two memory model extensions:
– Zam: for misaligned atomics
– Ztso: for RISC-V Total Store Order (RVTSO)
RISC-V Weak Memory Model

- Defined by 13 ordering rules
- Complex
- “bugs” already found w.r.t C++11 memory model and non-Multi-Copy-Atomic architectures [C. Trippel, Micro Top Picks]

[Dan Lustig, RISC-V Memory Consistency Model Tutorial]
RISC-V Total Store Order

Defined as an extension of RVWMO

- All loads acquire semantics
- All stores release semantics
- All AMOs acquire & release semantics
- BUT: \( \text{RC}_{PC} \) not \( \text{RC}_{SC} \):
  \( \text{rl} \to \text{aq} \) does not apply: relaxes store\( \to \)load

[Dan Lustig, RISC-V Memory Consistency Model Tutorial]
How to compare?

• Advantage RVTSO → simpler semantics

• But what is its cost/performance?

• Perception:
  – **WMO is Faster**: enforces less ordering
    • TSO enforces more
  – **WMO is Cheaper**: fences enforce order
    • TSO must enforce all order in HW
Fallacies

TSO Enforces more ordering
• Not true
  – [Ros, Kaxiras, Micro’16, Micro Top Picks Honorable Mention, Ros & Kaxiras “Racer”]
• Hardware can **speculatively reorder** memory operations
• Reacts only on **conflicts** and enforces order
• *In contrast WMO: Statically fencing code for every possible conflict even if the conflict does not appear at runtime!*

TSO More costly to implement: requires speculation
• Not true
• Non-speculative load-load reordering
  – [Ros, Kaxiras, ISCA’17, Micro Top Picks]
• Non-speculative store-store reordering
  – [Ros, Kaxiras, ISCA’18]
Ordering in Consistency Models

<table>
<thead>
<tr>
<th>SC</th>
<th>TSO</th>
<th>WMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld → ld</td>
<td>ld → ld</td>
<td>ld → ld</td>
</tr>
<tr>
<td>ld → st</td>
<td>ld → st</td>
<td>ld → st</td>
</tr>
<tr>
<td>st → st</td>
<td>st → st</td>
<td>st → st</td>
</tr>
<tr>
<td>st → ld</td>
<td>st → ld</td>
<td>st → ld</td>
</tr>
</tbody>
</table>

Speculative reordering: Enforce ordering on dynamic events (misspeculation)

Enforce ordering via statically-placed fences:
- fence_SS
- fence_SL
## Ordering in Consistency Models

<table>
<thead>
<tr>
<th></th>
<th>SC</th>
<th>TSO</th>
<th>WMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld → ld</td>
<td>ld → ld</td>
<td>ld → ld</td>
<td>ld → ld</td>
</tr>
<tr>
<td>ld → st</td>
<td>ld → st</td>
<td></td>
<td>ld → st</td>
</tr>
<tr>
<td>st → st</td>
<td>st → st</td>
<td></td>
<td>st → st</td>
</tr>
<tr>
<td>st → ld</td>
<td>st → ld</td>
<td></td>
<td>st → ld</td>
</tr>
</tbody>
</table>

### NON-SPECULATIVELY:
- Allow load-load reordering (hit-under-miss, MLP) but guarantee `ld → ld` when it can be observed [ISCA’17]
Ordering in Consistency Models

<table>
<thead>
<tr>
<th>SC</th>
<th>TSO</th>
<th>WMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld (\rightarrow) ld</td>
<td>ld (\rightarrow) ld</td>
<td>ld (\rightarrow) ld</td>
</tr>
<tr>
<td>ld (\rightarrow) st</td>
<td>ld (\rightarrow) st</td>
<td>ld (\rightarrow) st</td>
</tr>
<tr>
<td>st (\rightarrow) st</td>
<td>st (\rightarrow) st</td>
<td>st (\rightarrow) st</td>
</tr>
<tr>
<td>st (\rightarrow) ld</td>
<td>st (\rightarrow) ld</td>
<td>st (\rightarrow) ld</td>
</tr>
</tbody>
</table>

**NON-SPECULATIVELY:**
- Allow load-load reordering (hit-under-miss, MLP) but guarantee \(ld \rightarrow ld\) *when it can be observed* [ISCA’17]
- Allow store-store reordering (store coalescing) but guarantee \(st \rightarrow st\) *when it can be observed* [ISCA’18]
- \(ld \rightarrow st\) : inconsequential
- Store atomicity
Speculative load Reordering in TSO

- Core
- Store Buffer
- Cache
- Shared Cache/Dir/Memory

- ld → ld
- ld → st
- st → st
- st → ld
Squash & Re-execute

ld → ld
ld → st
st → st
st → ld

Core
Store Buffer
Cache

Shared Cache/Dir/Memory
A New *Non-Speculative* Solution

- Core
  - \( \text{ld} \rightarrow \text{ld} \)
  - \( \text{ld} \rightarrow \text{st} \)
  - \( \text{st} \rightarrow \text{st} \)
  - \( \text{st} \rightarrow \text{ld} \)

- Store Buffer
  - \( \text{st} \rightarrow \text{ld} \)

- Cache

- Shared Cache/Dir/Memory

Stefanos Kaxiras
A New Non-Speculative Solution

ld → ld
ld → st
st → st
st → ld

NACK: Don’t come out now or you might see me

Shared Cache/Dir/Memory
A New *Non-Speculative* Solution

**Core**
- **ld → ld**
- **ld → st**
- **st → st**
- **st → ld**

**Store Buffer**
- **st x**

**Cache**
- Don’t come out now or you might see me

**Shared Cache/Dir/Memory**

**ACK:** You can come out now, I’m decent

**NACK:** Don’t come out now or you might see me
A New Non-Speculative Solution

Key Insight: In a tight spot, use uncachable transactions ("Tear-off" data) → no Deadlock, no Livelock
A New *Non-Speculative* Solution

**Core**

- \( \text{ld} \rightarrow \text{ld} \)
- \( \text{ld} \rightarrow \text{st} \)
- \( \text{st} \rightarrow \text{st} \)
- \( \text{st} \rightarrow \text{ld} \)

**Store Buffer**

**Cache**

**ACK:** You can come out now, I'm decent

**NACK:** Don't come out now or you might see me

**Same performance & traffic as speculative solution;**

**BUT: No SPECULATION COST**

**Opens new possibilities:**
- In-order cores, OoO Commit, No Load Queue!
Memory Consistency Models

- **TSO**
  - `ld → ld`
  - `ld → st`
  - `st → st`
  - `st → ld`

- **RC**
  - Enforce ordering via statically-placed fences

### Non-Spec.
- Enforce `ld → ld`
- Ordering on conflicts

• Can we do the same for `st → st`?
• Why? Coalescing in the SB!
  - Violates `st → st` when coalescing non-consecutive stores
  - Solution: Atomic group writes

- `fence_LL`
- `fence_LS`
- `fence_SS`
- `fence_SL`
Non-Speculative Store Coalescing

Atomic group writes in L1 → deadlock

Known solutions:
1. Mutual exclusion (centralized resource) [TCC ISCA’04, BulkSC ISCA’07, Racer Micro’06]
2. Transactional (spec.) [Oklahoma PDTSA’03, Store-wait-free ISCA’07]
Non-Speculative Store Coalescing

Simple Solution: Order in \{\text{atomic group}\} does not matter \(\rightarrow\) use a non-deadlocking order

Lexicographical order: physical address

Core

Store Buffer

Cache

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Simple Solution: Order in \{\text{atomic group}\} does not matter \rightarrow use a non-deadlocking order

Lexicographical order: physical address

Core

Store Buffer

Cache

Core

Store Buffer

Cache

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Simple Solution: Order in {atomic group} does not matter → use a non-deadlocking order

Lexicographical order: physical address

wait
Non-Speculative Store Coalescing

Simple Solution: Order in \{atomic group\} does not matter \rightarrow use a non-deadlocking order

Lexicographical order: physical address

wait
Non-Speculative Store Coalescing

Simple Solution: Order in {atomic group} does not matter \(\rightarrow\) use a non-deadlocking order

Lexicographical order: physical address

\{a, b\} \rightarrow \{b, a\}

wait

Stefanos Kaxiras
Non-Speculative Store Coalescing

Simple Solution: Order in \{atomic group\} does not matter \(\rightarrow\) use a *non-deadlocking* order

Lexicographical order: physical address

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Simple Solution: Order in {atomic group} does not matter \(\rightarrow\) use a non-deadlocking order

Lexicographical order: physical address

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Simple Solution: Order in \{atomic group\} does not matter \(\rightarrow\) use a non-deadlocking order

Lexicographical order: physical address

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Simple Solution: Order in \{atomic group\} does not matter $\rightarrow$ use a non-deadlocking order

Lexicographical order : physical address

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Simple Solution: Order in \{\text{atomic group}\} does not matter $\rightarrow$ use a non-deadlocking order

Lexicographical order: physical address

Core

Store Buffer

Cache

Core

Store Buffer

Cache

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Simple Solution: Order in \{atomic group\} does not matter $\rightarrow$ use a non-deadlocking order

Lexicographical order: physical address
Non-Speculative Store Coalescing

Resource Deadlocks!

An **atomic group** does not fit in one of the set-associative structures (L1, L2, Dir, LLC, …) needed for a write

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Resource Deadlocks!

Key insight: manage the formation of an atomic group in the SB:

- **always fits in any set-associative structure** in the system
- **OR: has to wait** for others occupying shared resources
Non-Speculative Store Coalescing

Resource Deadlocks!

Key insight: *manage the formation of an atomic group in the SB:*

- **always fits in any set-associative structure** in the system
- OR: **has to wait** for others occupying shared resources

There is a global **sub-address (part of the address) lex order** that guarantees deadlock-freedom in all set-associative structures of the hierarchy!
Non-Speculative Store Coalescing

Resource Deadlocks!

Key insight: manage the formation of an atomic group in the SB:

- **always fits in any set-associative structure** in the system
- OR: **has to wait** for others occupying shared resources

There is a global sub-address (part of the address) lex order that guarantees deadlock-freedom in all set-associative structures of the hierarchy!
Non-Speculative Store Coalescing

Resource Deadlocks!

Key insight: manage the formation of an atomic group in the SB

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Resource Deadlocks!

Key insight: *manage the formation of an atomic group in the SB*

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Resource Deadlocks!

Key insight: manage the formation of an atomic group in the SB

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Resource Deadlocks!

Key insight: manage the formation of an atomic group in the SB

Core

Store Buffer

Cache

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Core

Store Buffer

{ a }

{ b }

{ c }

Cache

b c

Key insight: manage the formation of an atomic group in the SB

Resource Deadlocks!

Shared Cache/Dir/Memory
Non-Speculative Store Coalescing

Resource Deadlocks!

Key insight: manage the formation of an atomic group in the SB

This is TSO.

Results: SAME coalescing potential as Release Consistency (RC) [ISCA’18]
Store Atomicity

• Store atomicity has a huge impact on memory model implementations
  – Multi-Copy-Atomic (MCA): all cores see same store at same time
  – read-own-write-early MCA (rMCA): core can see own stores (in SB) early (e.g., x86)
  – non-MCA: everything goes

• In TSO, a store atomicity “violation” (rMCA, nMCA) may appear as a violation of load→load (e.g., source of complexity in the x86-LSB)

• Observation: it’s not a crime, if you don’t get caught
  – A store atomicity violation only matters if it causes load→load to be violated
  – Find the conditions that allow this to happen and prevent it!
Store Atomicity

• Same as every other consistency rule: react dynamically to conflicting accesses and appear to be store atomic
• Clean Stote-Atomic-TSO: like SC without the store→load
• Speculation & rollback
  – Negligible HW overhead
  – Speculative Store-Atomic-TSO: 3% overhead over rMCA TSO
  – MCA TSO: > 40% overhead over rMCA TSO
  (paper under submission)
Conclusions

• Enforcing order **dynamically** in TSO potentially better than static fencing in WMO …

• In the past reordering in TSO meant speculation overhead

• We have shown (in TSO):
  – *non-speculative* ld $\rightarrow$ ld
    • No LQ needed!
  – *non-speculative* st $\rightarrow$ st (coalescing)
    • Coalescing on par with RC
  – speculative-MCA TSO in rMCA architectures

• Future:
  – Speculative and non-speculative store atomicity, in nMCA architectures
Thank you!

Questions ?