EECS 570
Lecture 17
Memory Consistency Verification
Winter 2022
Prof. Yatin Manerkar
http://www.eecs.umich.edu/courses/eecs570/

Acknowledgements: Daniel Lustig, Caroline Trippel, Michael Pellauer, Margaret Martonosi
Outline

- Out-of-thin-air values
- Dependencies
- Cumulativity
- MCM Verification – why formal?
- PipeCheck
  - CCICheck and COATCheck
- TriCheck
  - RISC-V case study
Out of thin air (OOTA) values

• This section draws heavily on [Boehm and Demsky MSPC 2014]

• Consider the following program:
  □ Should r1 = 42, r2 = 42 be allowed?

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) r1 = x;</td>
<td>(i2) r2 = y;</td>
</tr>
<tr>
<td>(i2) y = r1;</td>
<td>(i3) x = 42;</td>
</tr>
</tbody>
</table>

Outcome: r1 = 42, r2 = 42

• How about the following program?

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) r1 = x;</td>
<td>(i2) r2 = y;</td>
</tr>
<tr>
<td>(i2) y = r1;</td>
<td>(i3) x = r2;</td>
</tr>
</tbody>
</table>

Outcome: r1 = 42, r2 = 42

□ If r1 = r2 = 42, the value 42 has appeared “out of thin air”!
□ Typical cases involve C11 relaxed atomics and Ld-St dependencies
OOTA in Hardware and Software

• Today, virtually all commercial ISAs enforce dependencies
  - Result: no dependency cycles, and no OOTA
• However, the software situation is different
• Consider:

```c
if (x) {
    a[i++] = 1;
} else {
    a[i++] = 2;
}
```

• A compiler may want to hoist the increment of i out of the conditional
  - But this would break the Ld-St dependency!
OOTA in Hardware and Software

• What about long, obscure dependence chains?

```c
y.store(f(x.load(memory_order_relaxed)),
memory_order_relaxed);
```

- If we want to preserve dependencies, f would also need to maintain dependencies here (even if it had no atomic accesses and was compiled separately)

• **Key takeaway:** developing a programming language specification that forbids out-of-thin-air values but allows reasonable compiler optimizations has proven to be extremely difficult
Why does OOTA matter?

• If OOTA is permitted, how do we prove that e.g., malicious code cannot generate secret passwords out of thin air?

• Certain kinds of OOTA computation can break the SC-for-DRF guarantee
Dependencies

• Can be used to enforce local ordering in some MCM scenarios
  □ e.g., can use dependency to order mp’s loads

• Address dependency:
  □ Result of a load is address for subsequent load or store
    lwz r1,0(r2);
    xor r3,r1,r1;
    lwzx r4,r3,r5;

• Data dependency:
  □ Result of a load is data for a subsequent store
    lwz r1,0(r2);
    xor r3,r1,r1;
    addi r3,r3,1;
    stw r3,0(r4);
Dependencies

- Control dependency:
  - Result of load controls branch condition, and load or store is after branch in program order [Sarkar et al. PLDI 2011]
    
    ```
    lwz r1,0(r2)
    cmpw r1,r1
    beq LC00
    LC00:
    lwz r3,0(r4)
    ```
  
  - ARM, Power, RISC-V enforce load-store control dependencies but not load-load control dependencies
    - Reasoning: want to be able to predict a branch and execute loads after the branch
    - Speculative stores that can be seen by other cores before the speculation commits are a really bad idea
    - Can put isync (Power) or isb (ARM) after the branch to enforce load-load control dependency
Cumulativity

• In an MCA or rMCA system, fences need only enforce ordering locally (among operations on the same core)
  □ Below, LOCFENCE denotes a fence that only orders instructions on the same core

Luke to Read Causality (WRC) litmus test

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
<th>Core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) x = 1;</td>
<td>(i2) r1 = x; (i3) LOCFENCE; (i4) y = 1;</td>
<td>(i5) r2 = y; (i6) LOCFENCE; (i7) r3 = x;</td>
</tr>
</tbody>
</table>

□ But what about an nMCA system?
  ○ Local ordering is not enough! (dependencies don’t work either!)
  ○ Fences must be cumulative!
  ○ ISAs with nMCA like Power have cumulative fences
Cumulativity

- Lightweight cumulative fence (lwsync on Power)
  - Set A: any writes **observed** by the fencing core before the fence (even those from other cores)
  - Set B: writes after the fence (set B)
  - lwsync orders writes in set A before those in set B

Write-to-Read Causality (WRC) litmus test

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
<th>Core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) ( x = 1 );</td>
<td>(i2) ( r_1 = x );</td>
<td>(i5) ( r_2 = y );</td>
</tr>
<tr>
<td>(i3) lwsync;</td>
<td>(i6) lwsync;</td>
<td>(i7) ( r_3 = x );</td>
</tr>
<tr>
<td>(i4) ( y = 1 );</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Power forbids:** \( r_1 = 1, r_2 = 1, r_3 = 0 \)

- lwsyncs do not order writes with respect to subsequent reads
  - Do not enforce a total order!
  - Similar to release-acquire semantics
- Aside: the lwsync on core 2 could be replaced with a dependency
Cumulativity

- lwsyncs cannot forbid IRIW!

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1;</td>
<td>y = 1;</td>
<td>r1 = x;</td>
<td>r3 = y;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lwsync;</td>
<td>lwsync;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r2 = y;</td>
<td>r4 = x;</td>
</tr>
</tbody>
</table>

Power allows: r1 = 1, r2 = 0, r3 = 1, r4 = 0

- Must use heavyweight cumulative fence (sync on Power)
  - All writes observed by the fencing core before the fence must be made visible to all other cores, and all reads before the fence must be performed before any memory operations after the fence are performed

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1;</td>
<td>y = 1;</td>
<td>r1 = x;</td>
<td>r3 = y;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sync;</td>
<td>sync;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r2 = y;</td>
<td>r4 = x;</td>
</tr>
</tbody>
</table>

Power forbids: r1 = 1, r2 = 0, r3 = 1, r4 = 0
MCM Verification
The Need for MCM Verification

- MCMs specified at interfaces between layers of hardware/software stack, including:
  - Instruction Set (ISA)
  - High-level languages (e.g. C/C++, Java)

- Similar ordering verification necessary in other domains:
  - Distributed systems
  - Security
MCM Verification - Why Formal?

- Multiprocessors are nondeterministic
- Some program runs could exhibit bugs, others could not
  - How can we be sure?

---

**Test Configuration**

<table>
<thead>
<tr>
<th>Test Order</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>5010437</td>
<td>3</td>
</tr>
<tr>
<td>4989500</td>
<td>3</td>
</tr>
<tr>
<td>63</td>
<td>0</td>
</tr>
</tbody>
</table>

**Witnesses**

Positive: 63, Negative: 9999937
MCM Verification - Why Formal?

• Dynamic verification
  - TSOTool [Hangal et al. ISCA 2004]
  - DVMC [Meixner and Sorin TDSC 2009]
  - litmus [Alglave et al. TACAS 2011]

• None of these approaches can **ensure** correctness - even for a given test program!

• **Formal** verification: techniques based on rigorous mathematical analysis
  - Excel at checking all possible cases for a given situation
  - Example: PA2! (and the rest of this lecture)
Instruction-Level Analysis

- What if we use one node per instruction?
  - Works ok for mp...

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) x = 1;</td>
<td>(i3) r1 = y;</td>
</tr>
<tr>
<td>(i2) y = 1;</td>
<td>(i4) r2 = x;</td>
</tr>
<tr>
<td><strong>SC Forbids:</strong></td>
<td><strong>r1 = 1, r2 = 0</strong></td>
</tr>
</tbody>
</table>

![Diagram showing inter-core communication](Diagram.png)

EECS 570
Instruction-Level Analysis

...but what about other litmus tests?

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) x = 1;</td>
<td>(i4) y = 1;</td>
</tr>
<tr>
<td>(i2) r1 = x;</td>
<td>(i5) r3 = y;</td>
</tr>
<tr>
<td>(i3) r2 = y;</td>
<td>(i6) r4 = x;</td>
</tr>
</tbody>
</table>

TSO Allows: r1 = 1, r2 = 0, r3 = 1, r4 = 0

Can’t just say cyclic = forbidden and acyclic = allowed!

- Have to treat different edges differently

What if we want to do microarchitectural verification?

- Would like a model that reflects the inner workings of the microarchitecture
Does hardware correctly implement ISA MCM?

Microarchitecture

Coherence Protocol (SWMR, DVI, etc.)

Litmus Test

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td>(i3) Ld r1 ← [y]</td>
</tr>
<tr>
<td>(i2) St [y] ← 1</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Under TSO: Forbid r1=1, r2=0

Instruction level analysis of litmus test

<table>
<thead>
<tr>
<th></th>
<th>Observable</th>
<th>Unobservable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permitted</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>Forbidden</td>
<td>BUG</td>
<td>OK</td>
</tr>
</tbody>
</table>

SC/TSO/RISC-V MCM?
(for the litmus test)
PipeCheck: Executions as μhb Graphs

Core 0


Core 1


Litmus Test mp

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St [x] ← 1</td>
<td>(i3) Ld r1 ← [y]</td>
</tr>
<tr>
<td>(i2) St [y] ← 1</td>
<td>(i4) Ld r2 ← [x]</td>
</tr>
</tbody>
</table>

Under TSO: Forbid r1=1, r2=0

[Lustig et al. MICRO 2014]

F D X M W SB

po fr rf
PipeCheck: Verifying Correctness

- Exhaustive enumeration of executions using $\mu$hb graphs
- Cyclic graph $\rightarrow$ forbidden by $\mu$arch
- Acyclic graph $\rightarrow$ allowed by $\mu$arch

Litmus Test mp

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) St $x$ $\leftarrow$ 1</td>
<td>(i3) Ld $r1$ $\leftarrow$ $y$</td>
</tr>
<tr>
<td>(i2) St $y$ $\leftarrow$ 1</td>
<td>(i4) Ld $r2$ $\leftarrow$ $x$</td>
</tr>
</tbody>
</table>

Under TSO: Forbid $r1=1$, $r2=0$

<table>
<thead>
<tr>
<th></th>
<th>Observable (≥ 1 Graph Acyclic)</th>
<th>Not Observable (All Graphs Cyclic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allowed</td>
<td>OK</td>
<td>OK (stricter than necessary)</td>
</tr>
<tr>
<td>Forbidden</td>
<td>Consistency violation!</td>
<td>OK</td>
</tr>
</tbody>
</table>
CCICheck: Add ViCLs

- Use pipeline model from PipeCheck, but add ViCL nodes and edges

<table>
<thead>
<tr>
<th>Litmus Test co-mp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 0</td>
</tr>
<tr>
<td>(i1) St [x] ← 1</td>
</tr>
<tr>
<td>(i2) St [x] ← 2</td>
</tr>
</tbody>
</table>

In TSO: r1=2, r2=2 Allowed
COATCheck: Adding Virtual Memory

- [Lustig et al. ASPLOS 2016]
- Virtual memory can affect consistency too!
- Consider:

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i1) x = 1;</td>
<td>(i3) y = 2;</td>
</tr>
<tr>
<td>(i2) r1 = y;</td>
<td>(i4) r2 = x;</td>
</tr>
<tr>
<td><strong>Outcome:</strong> r1 = 2, r2 = 1</td>
<td></td>
</tr>
</tbody>
</table>

- r1=2, r2=1 is fine if x and y are different addresses
- But what if they are synonyms?
  - Outcome should then be forbidden!
- Also, what about hardware like page table walkers?
  - Accesses memory, but is not mentioned in the user program!
- Transistency models: consistency + address translation
  - Approx. transistency model for x86 [Hossain*, Trippel*, Martonosi ISCA 2020]
Specifying Microarchitectures [Lustig et al. ASPLOS 2016]

Microarchitecture specification in µSpec DSL

Axiom "PO_Fetch":
for all microop "a", "b",
SameCore a b /
\ ProgramOrder a b =>
AddEdge ((a, Fetch), (b, Fetch)).

Axiom "Execute_stage_is_in_order":
for all microop "i1",
...

- Microarchitecture described in µSpec DSL [Lustig et al. ASPLOS 2016]
  - AND (\), OR (\), NOT (~), implication (=>), for all, exists
- Built-in predicates can reference instructions, nodes, and edges
  - e.g. ProgramOrder i j
  - e.g. AddEdge ((i1, Fetch), (i2, Fetch))
TriCheck: Model both Hardware and Software

• When defining a new ISA’s MCM, we want it to:
  □ be relaxed enough to allow the hardware optimizations we want
  □ be strong enough to be able to implement HLL MCMs like C11

• How can we ensure this?
  □ TriCheck [Trippel et al. ASPLOS 2017] allows one to do this for litmus test suites
  □ Discovered numerous issues with the draft specification of the RISC-V MCM
  □ Issues subsequently fixed by a memory model working group; new MCM ratified in 2018
TriCheck: Hardware-Software Interoperability

High-Level Languages (HLL)

Compiler

Instruction Set (ISA)

Microarchitecture

Processor RTL (Verilog)

Thread 0

Thread 1

\[ x = 1; \]
\[ y = 1; \]

\[ r1 = y; \]
\[ r2 = x; \]

Outcome: \[ r1 = 1, r2 = 0 \]

Core 0

Core 1

\[ \text{str #1, [x]} \]
\[ \text{dmb} \]
\[ \text{str #1, [y]} \]

\[ \text{ldr r1, [y]} \]
\[ \text{dmb} \]
\[ \text{ldr r2, [x]} \]

Outcome: \[ r1 = 1, r2 = 0 \]

Axiom "PO_Fetch":

forall microop "i1", "i2",
SameCore i1 i2 \(\Rightarrow\) AddEdge ((i1, IF), (i2, IF)).

herd for HLL

[Batty et al. POPL 2016]

Allowed/Forbidden

Translate using compiler mappings

Forbidden HLL outcomes must be unobservable on hardware!

Microarch. MCM Verification
[Lustig et al. ASPLOS 2016]

Observable/Unobservable

[Batty et al. POPL 2016]

herd for HLL

[Batty et al. POPL 2016]
Case Study: Draft RISC-V MCM

- Draft MCM relaxed write atomicity...
- ...but did not strengthen fences to be cumulative!
- Result: impossible to ensure correct operation of all C11 programs!
- Solution: strengthen fences to be cumulative

### Write-to-Read Causality (WRC) litmus test

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x = 1;</code></td>
<td><code>a = x;</code></td>
<td><code>b = y;</code></td>
</tr>
<tr>
<td><code>y = 1;</code></td>
<td></td>
<td><code>c = x;</code></td>
</tr>
</tbody>
</table>

**Forbidden:** `a = 1, b = 1, c = 0`

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
<th>Core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(i1) x = 1;</code></td>
<td><code>(i2) r1 = x;</code></td>
<td><code>(i5) r2 = y;</code></td>
</tr>
<tr>
<td></td>
<td><code>(i3) FENCE;</code></td>
<td><code>(i6) FENCE;</code></td>
</tr>
<tr>
<td></td>
<td><code>(i4) y = 1;</code></td>
<td><code>(i7) r3 = x;</code></td>
</tr>
</tbody>
</table>

**Allowed:** `r1 = 1, r2 = 1, r3 = 0`
Same Addr Ld-Ld ordering

• Draft MCM did not enforce same-address load-load ordering
  ☐ Required by C11 atomics!

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1;</td>
<td>r1 = x;</td>
</tr>
<tr>
<td>x = 2;</td>
<td>r2 = x;</td>
</tr>
</tbody>
</table>

C11 Forbids: r1 = 2, r2 = 1

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1;</td>
<td>r1 = x;</td>
</tr>
<tr>
<td>x = 2;</td>
<td>r2 = x;</td>
</tr>
</tbody>
</table>

Draft RISC-V allows: r1 = 2, r2 = 1

• Note: blame can be debated
  ☐ Can fix the issue in one of two ways
    ☐ Enforce same-addr Ld-Ld ordering, OR
    ☐ Add fences after each atomic load
  ☐ However, sometimes one solution is better than others
Roach Motel Reorderings

- Draft MCM did not enforce write atomicity for atomics unless they were both acquire and release
- Result: roach motel movement forbidden for SC atomics!
- Solution: allow SC acquires and releases; don’t require acquire+release for SC order

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x.st(1, seq_cst); y.st(1, rlx);</td>
<td>r1 = y.ld(seq_cst); r2 = x.ld(seq_cst);</td>
</tr>
<tr>
<td><strong>C11 Allows:</strong> r1 = 1, r2 = 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core 0</th>
<th>Core 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>St.aq+rl x, 1; St x, 2;</td>
<td>Ld.aq+rl r1, y; Ld.aq+rl r2, x;</td>
</tr>
<tr>
<td><strong>Draft RISC-V forbids:</strong> r1 = 1, r2 = 0</td>
<td></td>
</tr>
</tbody>
</table>
More on the Check suite...

- Check out the ISCA 2019 tutorial on the tools!
- Things we didn’t discuss today:
  - RTLCheck [Manerkar et al. MICRO 2017]: Verifying memory consistency of RTL
  - PipeProof [Manerkar et al. MICRO 2018]: Verifying microarchitectures across all possible programs
  - C11 compilation bug for Power and ARMv7 [Manerkar et al. CoRR 2016]
  - CheckMate [Trippel et al. MICRO 2018]: hardware security verification