PipeCheck: Verifying Consistency Model Implementations

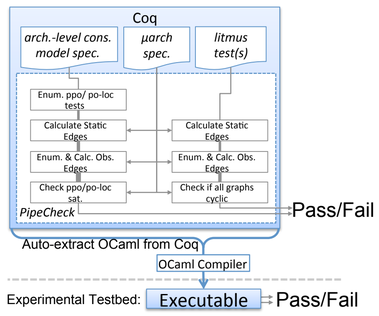
Daniel Lustig, Caroline Trippel, Michael Pellauer (Intel), and Margaret Martonosi, Princeton University

Motivation

- CPUs are prone to memory **reordering bugs** which impact both correctness and performance, e.g.:
  - AMD Phenom TLB bug
  - Intel Haswell/Broadwell TSX bug
- Support ISTC-CC Goals:
  - **Specialization:** need to verify increasingly diverse architectural designs
  - **Automation:** PipeCheck verification is entirely automated in software
  - **Big Data/to the Edge:** code and/or data migration assumes memory consistency correctness to guarantee data arrival and readiness conditions

PipeCheck Overview

- Methodology and **automated tool** for verifying that a microarchitecture correctly implements the memory ordering rules of the specification
- Treat preserved program order (PPO) as a **proposition to be verified**, rather than simply as an assumption



- Compare **micro-architectural "happens-before" graphs** against architectural expectations

Background

Rules in a memory consistency model:

1. **Preserved Program Order (PPO)**: the set of orderings enforced by default
2. Fences: used to enforce orderings not enforced by PPO
3. Dependencies
4. Reading writes early, e.g., from a processor's own store buffer

First Op. \ Second Op.	Load	Store
Load	Y	Y
Store	N (mfence)	Y

Ex: PPO for Total Store Order (TSO)

Formalization of memory consistency models is an active research area even today

Litmus Tests

- Most common existing approach for verification
- Example for Ld→Ld and St→St reordering:

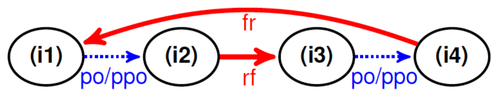
Core 0	Core 1
(i1) *x = 1	(i1) r1 = *y
(i2) *y = 1	(i1) r2 = *x
Proposed outcome: r1 = 1; r2 = 0	

- Proposed outcome may be **forbidden** or **permitted**, depending on the model:

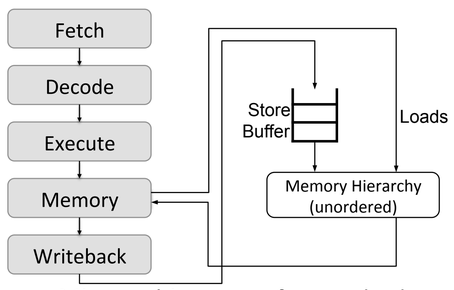
Memory Model	Proposed Result in above litmus test is:
SC, TSO:	Forbidden
ARM, Power:	Permitted

- Unfortunately, testing cannot guarantee 100% coverage

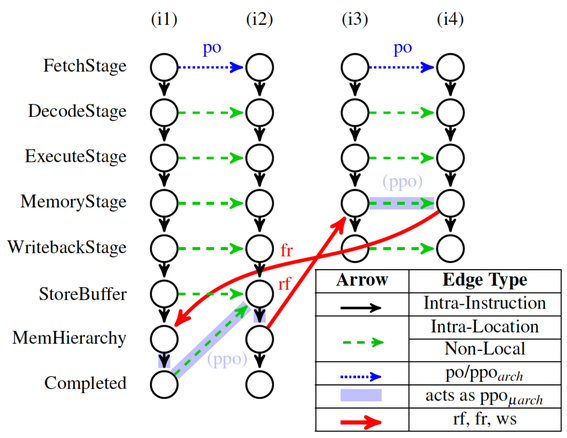
PipeCheck μ-Happens-Before Graphs



Architecture-level analysis of above litmus test under x86-TSO [Alglave, FMSD '09]



Microarchitecture of a standard five-stage pipeline



PipeCheck microarchitectural analysis of the litmus test on the given pipeline

- μhb = "microarchitecturally happens before"
- **Cycle** in μhb graph → the proposed execution is **forbidden**
- **No cycle** in μhb graph → the proposed execution is **allowed**
- Explains ***why*** a particular outcome is allowed or forbidden

Verification Approach

- Arch.-level spec determines **permitted vs. forbidden**
- With μarch-level spec., PipeCheck calculates **observable or not**

	Observable	Not Observable
Permitted	OK	OK (μarch stricter than necessary)
Forbidden	Pipeline bug!!	OK

Two test categories:

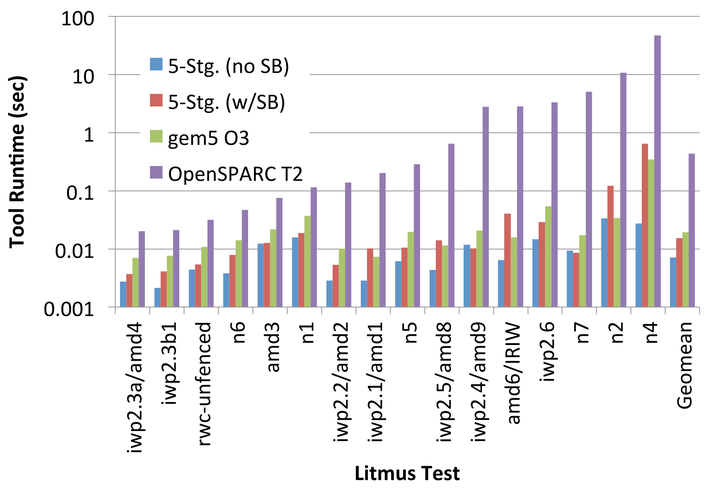
1. **PPO satisfaction tests**: check that all orderings required by PPO table are enforced by pipeline
2. **Litmus tests:** check that there is no execution in which a forbidden test outcome is observed

Verification Results

- Tested four real and simulated pipelines specified to be TSO

Pipeline	LoC in μarch model
5-Stage w/o St. Buf.	37
5-Stage w/ St. Buf.	67
gem5 O3	106
OpenSPARC T2	115

- PipeCheck verification is fast:



- Litmus tests highlight discrepancies between arch. and μarch. specs:

Litmus Test	Expected	5-Stage w/o St. Buf	5-Stage w/ St. Buf	gem5 O3	OpenSPARC T2
iwp2.1/amd1	Forbid	✔	✔	Observed[2]	✔
iwp2.2/amd2	Forbid	✔	✔	✔	✔
iwp2.3a/amd4	Permit	Not obs.[1]	✔	✔	✔
iwp2.3b	Permit	✔	✔	✔	✔
iwp2.4/amd9	Permit	Not obs. [1]	✔	✔	✔
iwp2.5/amd8	Forbid	✔	✔	Observed[2]	✔
iwp2.6	Forbid	✔	✔	✔	✔
amd3	Permit	Not obs. [1]	✔	✔	✔
amd6	Forbid	✔	✔	Observed[2]	✔
n1	Permit	Not obs. [1]	✔	✔	✔
n2	Forbid	✔	✔	Observed[2]	✔
n4	Forbid	✔	✔	✔	✔
n5	Forbid	✔	✔	✔	✔
n6	Permit	✔	✔	✔	✔
n7	Permit	Not obs. [1]	✔	✔	✔
rwc-unfenced	Permit	Not obs. [1]	✔	✔	✔

- [1]: 5-Stage pipeline w/o store buffer is stricter than necessary
- [2]: PipeCheck found a bug in the gem5 O3 pipeline (fixed in latest version)

Advanced Techniques

- PipeCheck supports:
  - inter-dependence of coherence and consistency
  - techniques such as speculative load reordering, even when they technically violate the architecture-level specification
- Full details described in paper

Conclusion

- We define **microarchitecture-level** happens-before graphs and uses them to verify pipeline implementations agains their architectural memory model specifications
- PipeCheck **tool** performs verification **automatically** given both specs

Intel Science & Technology Center for Cloud Computing
Carnegie Mellon University
Georgia Tech
intel
PRINCETON UNIVERSITY
UC Berkeley
UNIVERSITY of WASHINGTON