PipeProof: Automated Memory Consistency Proofs for Microarchitectural Specifications
Yatin A. Manerkar, Daniel Lustig (NVIDIA), Margaret Martonosi, Aarti Gupta
Nominated for Best Paper

The Need for All-Program MCM Verification:

- Does a microarchitecture obey its MCM for all programs?
- Prior work is either incomplete or manual verification.
- Can we automatically prove correctness for all programs?

ISA-Level MCMs and Microarchitectural Ordering Specifications

- ISA-Level MCMs defined in terms of acyclicity, indistinguishability, etc. of relational patterns [Agarwala et al., TOPLAS 2014]
- Microarchitectural ordering specifications defined as set of tupic axioms [Lustig et al., ASPLOS 2016]

The Transitive Chain (TC) Abstraction

- Abstractions enable a finite representation of an infinite set of executions
- PipeProof’s novel TC Abstraction represents sequence (chain) of ISA-level edges as a single edge (transitive connection) from start to end of chain
- Intermediate instructions in chain are not explicitly modelled
- Verification of infinite number of ISA-level cycles is verified across a finite set of transitive connections
- Microarchitectural support of abstraction automatically proven as a supporting proof

Microarchitectural Correctness

- Chain Invariants
  - Abstractly represent repeated ISA-level edge patterns
  - Sometimes needed for refinement loop to terminate
  - Inductively proven by PipeProof before their use elsewhere

Memorization Optimization

- Base PipeProof algorithm examines some ISA-level cycles multiple times
- Memorization eliminates redundant checks of cycles that have already been verified

PipeProof Block Diagram

- Run PipeProof on two microarchitectures
  - simpleSC (SC) and simpleTSO (TSO)
- 3-stage in-order pipelines
- simpleTSO relays Write-Read ordering

Covering Sets Optimization

- PipeProof must conduct verification across all possible transitive connections
- Each decomposition creates a new set of transitive connections
- This can quickly lead to a case explosion
- The Covering Sets Optimization eliminates redundant transitive connections
- Graph A has a transitive connection from x to z
- Graph B has a transitive connection from y to z, but also has an edge from x to y through transitivity
- Correctness of A \( \Rightarrow \) Correctness of B, because B contains all of A’s edges \+ an edge from y to z
- B does not need to be explicitly checked, and is eliminated by the Covering Sets Optimization

TC Abstraction Support Proof

- Ensure that ISA-level pattern and parent support TC Abstraction
- Proof is inductive
  - **Base Case**: No initial ISA-level edges guarantee a transitive connection
  - **Inductive Case**: Does extending the transitive chain extend the transitive connection?

Results

- Ran PipeProof on two microarchitectures
  - simpleSC (SC) and simpleTSO (TSO)
- 3-stage in-order pipelines
- simpleTSO relays Write-Read ordering

PipeProof: Automated All-Program Microarchitectural Memory Consistency Verification
- Users need only provide ISA-level and microarch model, mappings, and chain invariants.
- Designers no longer need to choose between completeness and automation
- Transitive Chain Abstraction allows inductive modelling and verification of the infinite set of all possible executions
- Abstraction is automatically refined as necessary to prove correctness
- Verified simple microarchitectures implementing SC and TSO in < 1 hour!
- Covering Sets Optimization and Memorization greatly reduce runtime

Conclusions

Code available at: https://github.com/ymanerka/pipeproof

[Image of diagram and text]