COATCheck, Future Possibilities, and Wrap-up

COATCheck: Verifying Memory Ordering at the Hardware-OS Interface

Daniel Lustig, Geet Sethi[+], Michael Pellauer*,
Margaret Martonosi, Abhishek Bhattacharjee [+]

Princeton University [+]Rutgers University *NVIDIA

ASPLOS 2016

http://check.cs.princeton.edu/

Simple Motivating Example

Initially: [x]=0, [y]=0	
Thread 0	**Thread 1**
St [x] ← 1 Ld [y] → r1	St [y] ← 2 Ld [x] → r2
Proposed outcome: r1=2, r2=1	

Permitted if x and y are different addresses

Initially: [x]=0, [y]=0	
Thread 0	**Thread 1**
St PA1←1 Ld PA2→r1	 St PA2←2 Ld PA1→r2
Outcome r1=2, r2=1 permitted	

Forbidden if x and y are synonyms

Initially: [x]=0, [y]=0	
Thread 0	**Thread 1**
St PA1←1 Ld PA1→r1	 St PA1←2 Ld PA1→r2
Outcome r1=2,r2=1 forbidden	

“Transistency Model”

- Memory ordering verification is fundamentally incomplete unless it explicitly accounts for address translation
- Superset of consistency which captures all address translation-aware sets of ordering rules
- Most prior techniques ignore the implications of virtual-to-physical address translation on memory ordering
  - E.g., synonyms, and page permission updates
- Microarchitectural events and OS behavior can affect memory ordering in ways for which standard memory model analysis can be fundamentally insufficient

Ongoing Work

- We've seen how memory model bugs can result in in correct program outcomes that are intermittent/unpredictable
- Currently, we are applying our techniques of exhaustive enumeration and checking of event orderings to other domains
  - Security – does an underlying implementation provide certain security guarantees?
  - IoT – how do we reason about many concurrently acting IoT devices?

Takeaways

- Memory consistency modes matter
  - Reliability, correctness, and portability
  - Performance
  - Security
- Intuitive “checking” through automated verification
- Move memory model verification earlier in the design processes
- Evaluate across interfaces and design boundaries
  - If interfaces are often source of bugs
- Speed of approach enables new oppertunities
  - Comprehenseive and fast verification for iterative design

http://check.cs.princeton.edu/