Store Vulnerability Window (SVW): Re-Execution Filtering for Enhanced Load Speculation

Amir Roth
University of Pennsylvania
Load Speculation

• Load/store unit: performance critical, but complex
  • Ambiguous dependences → associative search → scalability issues

• Overcome with **load speculation (wrt older stores)**
  • “Load speculation”: no-dependence on unresolved stores
  • Speculative forwarding: identity of forwarding store
  • Redundant load elimination: no conflicting store since first load
  • Not value prediction

• Common question: **how to verify speculation?**
  • “Load speculation”: associative load queue search
  • Speculative forwarding: same
  • Redundant load elimination: ??
Verification: Load Re-Execution

```
orig-value != re-executed-value ? flush
```

- **Load re-execution**: general-purpose load verifier
  - Re-execute loads in-order prior to commit
  - Flush/restart if re-executed value != original value

- **Performance**
  - No associative load queue search
  - No false squashes (value based)
    - $D\$ bandwidth contention
    - Reduced effective capacity of LSQ, regfile
    - Multi-cycle $D\$ → critical loop with store retirement
  - **Obvious key**: re-execute as few loads as possible
SVW: Store Vulnerability Window

- **SVW**: general-purpose load re-execution filter
  - Loads access new table to see if they can safely skip re-execution

- **Performance**
  - Additional pipeline stage? Additional table access?
  - SSBF is much smaller than D$ (e.g., 1KB vs. 64KB)
    - High bandwidth
    - Single-cycle access → no critical loop
  
  + Reduces re-executions by 5-20X
Talk Outline

• SVW framework
• SVW applied to...
  • Load speculation
  • Speculative forwarding
  • Redundant load elimination
• Something new ... and pretty cool
• Summary
SVW Concept

- Observe: load mis-speculation is a function of stores
  - No store to load’s address in a long time $\rightarrow$ no need to re-execute

- SVW formalizes this notion
  - Each load vulnerable to some window of stores older than itself...
  - ...and no stores older than this window
  - Load’s **Store Vulnerability Window (SVW)**
  - Re-execute load only if it collides with store in its SVW

- Size of SVW is important: too small (wrong), too big (slow)
SVW Implementation

- At RENAME: a way of physically specifying store windows...
  - Monotonically increasing store sequence numbers (SSNs)
  - store.SSN = SSN\textsubscript{RENANE}++
  - \textit{load.SVW} = \textit{SSN-of-youngest-”don’t-care”-store}
    - Speculation specific
- At SVW: ...the address collision test
  - \textit{SSBF} (Store Sequence Bloom Filter): address-indexed table of SSNs
  - \textit{SSBF}[store.addr] = store.SSN
  - \textit{load.must-re-execute} = (\textit{SSBF}[load.addr] > \textit{Id.SVW})
    - Same for all forms of speculation
SVW for “Load Speculation”

- Load speculation
  - (Intelligently) issue loads before older unresolved stores
  - Widely-used: e.g., store sets [Moshovos+’97, Chrysos+Emer’98, Yoaz+’99]

- Load speculation+REX
  - Replace load queue search with load re-execution [Cain+Lipasti’04]

- SVW for load speculation+REX
  - \texttt{load.SVW} = SSN-of-youngest-”don’t care”-store
  - “Don’t care” stores? Those that committed before load dispatched
    - Speculation restricted to instruction window
  - Implementation: \texttt{load.SVW} = SSN_{COMMIT}
Load Speculation+SVW: Example

- Three events (Rename, Exec, SVW/Re-exec) in the life of a load
- Addresses: A, B, C, D, SSNs/SVWs: 0, 1, …64, 65, 66…
- LSQ: load store

\[t=X\]
Rename load, establish SVW
\[\text{load.SVW} = \text{SSN}_{\text{COMMIT}} (64)\]

\[t=X+5\]
Execute load (ambiguous, but correct)
Also commit store 65

\[t=X+10\]
SVW load, check address collision
\[\text{SSBF[load.addr]} (0) > \text{load.SVW} (64)?\]
No, no need to re-execute
Load Speculation+SVW: Other Example

- Same setup

$t=X$
 Rename load, establish SVW
 load.SVW = SSN_{COMMIT} (64)

$t=X+5$
 Execute load (ambiguous, and wrong)
 Also commit store 65

$t=X+10$
 SVW load, check address collision
 SSBF[load.addr] (66) > load.SVW (64)?
 Yes, must re-execute
(How Well) Does This Work?

• What we care about
  • Reductions in re-execution rates
  • Speedups

• Experimental parameters
  • SPECint2000, Alpha ISA, -O4, no nops,
    • Train inputs, 2% sampling
  • 8-way out-of-order superscalar, 20 stage pipeline
    • 256-entry ROB, 48-entry SQ, 80 RS
    • 32KB, 2-way, 2-cycle I/D$, 512KB, 8-way, 15-cycle L2
  • 16-bit SSNs → wrap-around every 64K stores
    • On wrap-around: drain pipe (like store-barrier), clear SSBF
  • 512-entry SSBF → 1KB
Load Speculation: Results

- **Baseline:** load speculation with associative load queue (LQ)
- **Load speculation+REX**
  - Re-execution rate: ~9% Speedup: ~0.5%
  - Natural re-execution filter: only if issued with unknown stores...
Load Speculation: Results

- **Baseline**: load speculation with associative load queue (LQ)
- **Load speculation + REX**
  - Re-execution rate: \(~9\%\) Speedup: \(~0.5\%\)
  - Natural re-execution filter: only if issued with unknown stores...
- **Load speculation + REX + SVW**
  - Additional filter: ...and only if store in SVW wrote to load’s address
  - Re-execution rate: \(~1\%\) Speedup: \(~1.5\%\)
- **SVW** is an enhancer
SVW for Speculative Forwarding

• Attacks store queue (as opposed to load queue) scalability

• Observe: Conventional store queue performs two functions
  • Store retirement: non-associative, all stores
  • Store forwarding: associative, few (predictable) loads/stores

• Exploit: decompose store queue [Roth’04, Baugh+Zilles’04]
  • Retirement queue: large, non-associative, off critical path
  • Forwarding queue: small, associative, on critical path
  • Speculatively steer loads/stores to forwarding queue
  • Re-execute loads to check speculation/train steering predictor

• SVW for speculative forwarding: same
  • load.SVW = SSN_{COMMIT}
Speculative Forwarding: Results

- **Baseline**: 48-entry/2-port/4-cycle store queue
- **Speculative forwarding** (16/1/2)+REX
  - Re-execution rates: 100%, Speedups: –12%
  - No natural re-execution filter
Speculative Forwarding: Results

- **Baseline**: 48-entry/2-port/4-cycle store queue
- **Speculative forwarding (16/1/2)+REX**
  - Re-execution rates: 100%, Speedups: −12%
  - No natural re-execution filter
- **Speculative forwarding+REX+SVW**
  - Re-execution rate: 14%, speedups: 14%
- **SVW is an enabler**
SVW for Redundant Load Elimination

• Redundant load elimination
  • Identify redundant loads (e.g., by tracking register dependences)
  • Eliminate them (e.g., by pointing to output reg of original load)
    + Removes loads from execution core, reduces observed load latency
    - Must re-execute eliminated loads
      • Address-based invalidation is expensive and incomplete
      • Many instantiations [Sodani+’97, Onder+’99, Petric+’02,’05]

• SVW for redundant load elimination: a different definition
  • Eliminated load vulnerable to all stores ... starting at original load
    • Pass this SSN via reuse table
  • Notice: speculation operates outside instruction window
    • Load queue search is not an option
Redundant Load Elimination: Results

- Baseline: no load elimination
- **Redundant load Elimination+REX**
  - Re-execution rate: ~24%, Speedups: ~4%
  - Natural re-execution filter: only eliminated loads
Redundant Load Elimination: Results

- **Baseline**: no load elimination
- **Redundant load elimination + REX**
  - Re-execution rate: \(~24\%\), Speedups: \(~4\%\)
  - Natural re-execution filter: only eliminated loads
- **Redundant load elimination + REX + SVW**
  - Re-execution rate: \(~0.5\%\), Speedups: \(~8\%\)
  - SVW is an **enhancer** ... but a really good one
Something Cool: REX+SVW ... –REX!

- Originally: SVW as re-execution filter
  - orig-value != re-execute value? flush
  - SSBF hit? flush

- Alternatively: **SVW as re-execution substitute**
  - SSBF hit = re-execution failure, flush+restart
  - Thanks Vlad
REX+SVW–REX: Results

- **Load reuse**
  - Low filtered re-execution rates: < 1%
  - Almost as good as SVW+REX, often better than REX

- **Speculative forwarding (not shown)**
  - Higher filtered re-execution rates: 10+%
  - Noticeable performance degradation
  - Future: reduce re-executions even more
Summary

• **Load re-execution**
  + Supports simplifications in hard part of core (load/store)
  − Cost not exactly zero: D$ bandwidth, serialization, queue pressure

• **Store Vulnerability Window (SVW)**
  • Store tracking mechanism for reducing load re-executions
  + Enhances some optimizations (load speculation, load reuse)
  + Enables others (speculative forwarding)