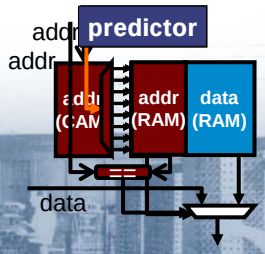
NoSQ:
Store-Load Communication without a Store Queue

Tingting Sha, Milo M.K. Martin, Amir Roth
University of Pennsylvania

{shatingt, milom, amir}@cis.upenn.edu
http://www.cis.upenn.edu/acg



MICRO-06 :: Dec 12, 2006

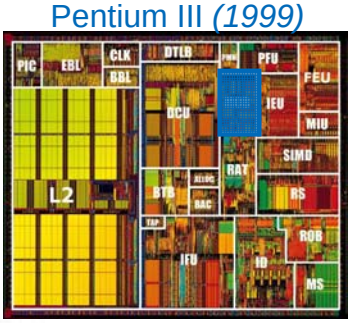
ACG University of Pennsylvania Architecture + Compilers Group

Store-Load Forwarding: Conventional

Associatively-searched store queue (SQ)

- Complex
- On load execution critical path
- Doesn't scale well
- Exists for benefit of 10% of loads

Pentium III *(1999)*



Store-Load Forwarding: Proposals

Reduce range and frequency of search
- Bloom-filtered SQ [Sethumadhavan+'03]
- Pipelined/chained SQ [Park+'03]
- Hierarchical/filtered SQ [Srinivasan+'04, Ghandi+'05, Torres+'05]
- Decomposed SQ [Roth'05; Baugh+'04]

Replace full associativity with set associativity
- Address-indexed forwarding cache [Stone+'05]

Replace search with speculative indexed access
- Speculative indexed SQ [Sha+'05], Fire-and-Forget [Subramaniam+'06]

Some forwarding structure still there
in the middle of the datapath

Store-Load Forwarding: Hmmm…

Observe I: can predict the exact forwarding store accurately
- Memory dependence prediction [Moshovos+'97, Chrysos+'98, …]
- No need to search SQ to find it

Observe II: store data already exists in register file
- No need to copy store input register → SQ → load output register

Should be able to eliminate SQ if …
- Can connect store's input data register to load's consumers
- Can verify in some way that doesn't require SQ

…we have the technology
- Speculative memory bypassing (SMB) [Moshovos+'97]
- Filtered in-order load re-execution [Cain+'04, Roth'05]

Store-Load "Forwarding": NoSQ

Predictor distinguishes bypassing/non-bypassing loads
- Non-bypassing loads just access the data cache
- Bypassing loads skip out-of-order execution (SMB)

Store vulnerability window (SVW) verifies and trains

Stores skip out-of-order execution too
- Don't participate in forwarding anymore

Commit pipeline extended to "execute" stores

No Store Queue (or any forwarding structure)!
No Load Queue!

Road Map

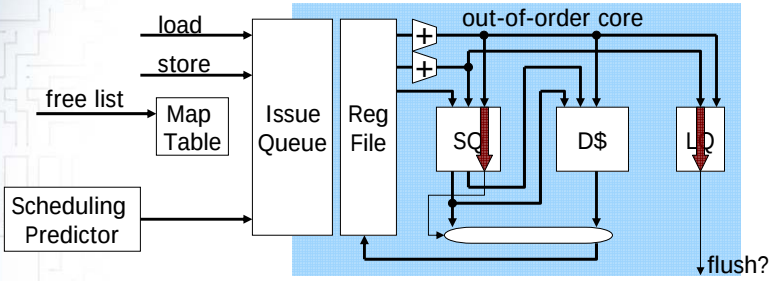
Overview

The road to NoSQ (prior work)
- Conventional store-load forwarding
- Load verification with filtered load re-execution
- Speculative memory bypassing

NoSQ
- Eliminating out-of-order stores and the store queue
- Eliminating the load queue
- Store-load bypassing predictor

Evaluation

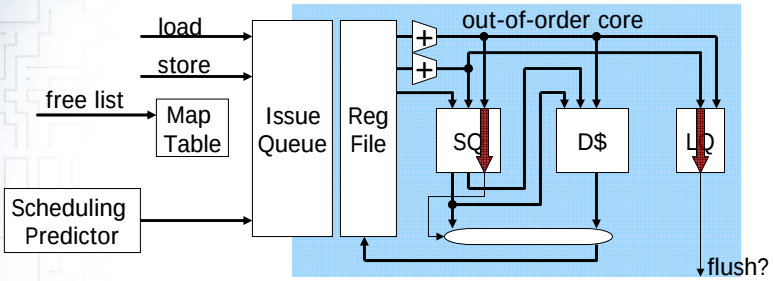
Conventional Design



Loads

- Execute: search SQ, write address into LQ

Conventional Design



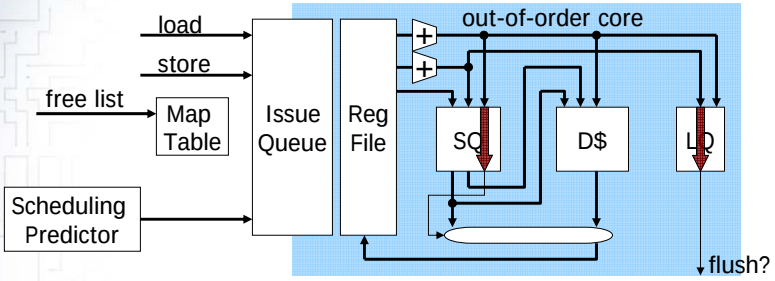
Loads

- Execute: search SQ, write address into LQ

Stores

- Execute: write address/data to SQ, search LQ for early loads

Conventional Design



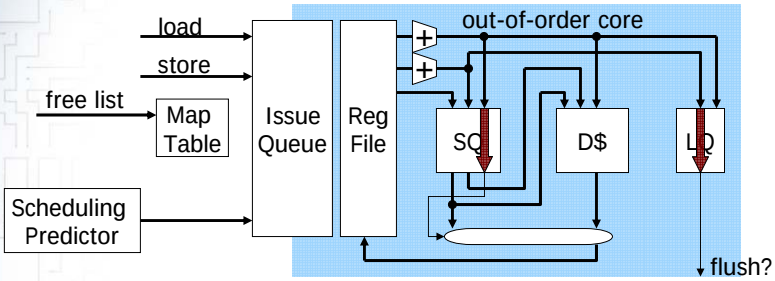
Loads

- Execute: search SQ, write address into LQ

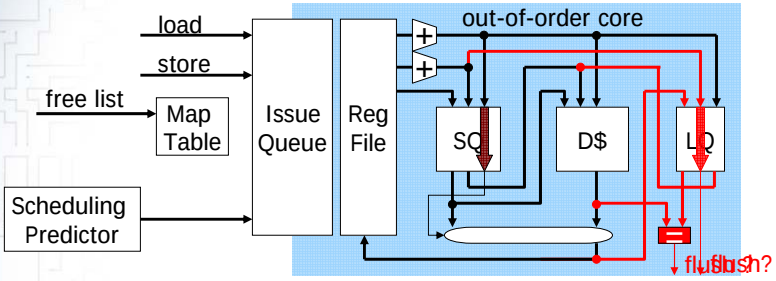
Stores

- Execute: write address/data to SQ, search LQ for early loads
- Commit: use data/address from SQ to write D$

+ In-Order Load Re-Execution [Cain+'04]



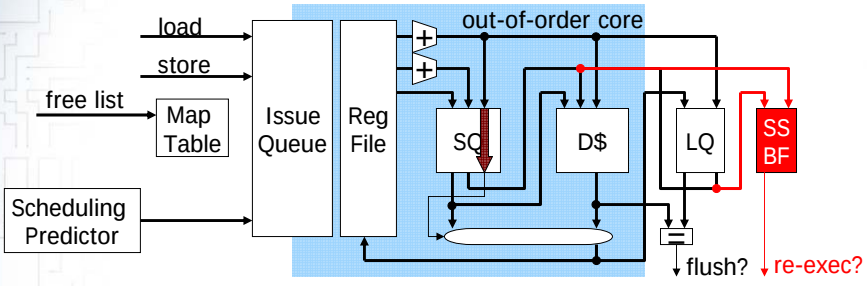
+ In-Order Load Re-Execution [Cain+'04]



Replace LQ search with load re-execution prior to commit

- Squash if in-order value != out-of-order value
- \+ Moves load queue out of core datapath
- \+ Can verify any form of load speculation
- – Consumes a lot of cache bandwidth if all loads are speculative

+ Store Vulnerability Window (SVW) [Roth'05]



Don't re-execute if no store to load's address in long time

- Store Sequence Numbers (SSNs): formalize time
- SSN Bloom Filter (SSBF): SSN of youngest store to address
- Store commit: update SSBF
- Load commit: read SSBF, skip re-execution if entry is older than…
  - Forwarding? …forwarding store
  - Non-forwarding? …youngest committed store at time of execute

Speculative Memory Bypassing [Moshovos+'97]

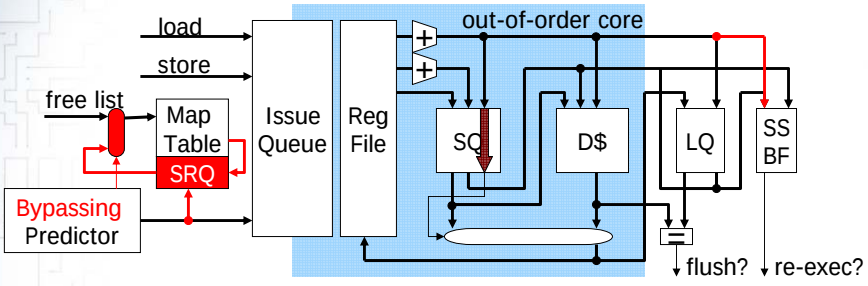
Raw insns:	*Renamed insns:*	*SMB renamed insns:*
add R1, 4 → R2	add P1, 4 → **P2**	add P1, 4 → **P2**
store R2 → A	store **P2** → **A**	store **P2** → A
load A → R3	load **A** → **P3**	load A → **P2**
sub R3, 4 → R4	sub **P3**, 4 → P4	sub **P2**, 4 → P4

Convert DEF-store-load-USE to DEF-USE

- Extend register renaming
  - Map-table[store.input] := Map-table[DEF.output]
  - Predict store-load dependence
  - Map-table[load.output] := Map-table[store.input]

+ Store-load removed from dataflow graph

+ Speculative Memory Bypassing



SRQ (store register queue): maps store to input data register

Originally: verify bypassing loads by executing out-of-order

Modification: verify using SVW-filtered re-execution [Petric+'05]

+ Bypassing loads skip out-of-order execution

+ Bypassing loads (~10%) never access data cache

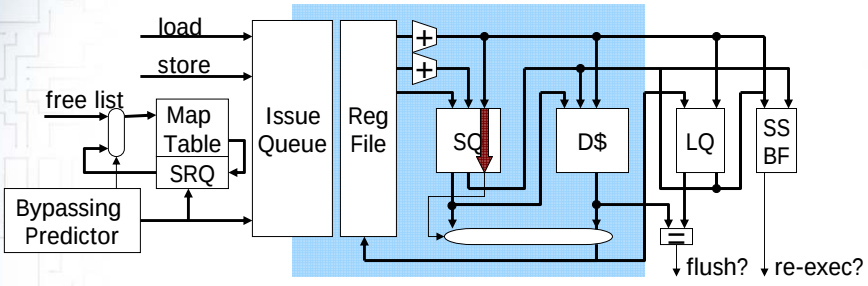
NoSQ: A New Use of SMB

Load re-execution / SVW filtering: target design simplification
- Eliminate associative load queue search

Traditional SMB: targets performance improvement
- SMB as opportunistic complement to store queue
  - Only 10% of loads forward → only 4% gains
- "Not worth the effort" [Loh+'02]

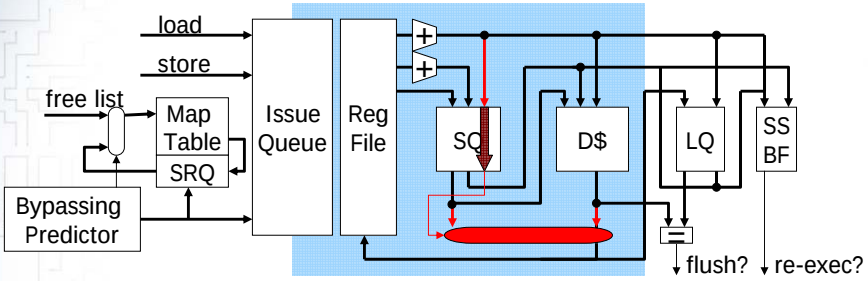
NoSQ SMB: targets design simplification
- SMB as exclusive replacement for store queue

NoSQ 1: Remove Store Queue from Load Path



Loads don't need to access store queue during execution
- Bypassing loads: skip out-of-order execution (SMB)
- Non-bypassing loads: get values from the data cache

NoSQ 1: Remove Store Queue from Load Path

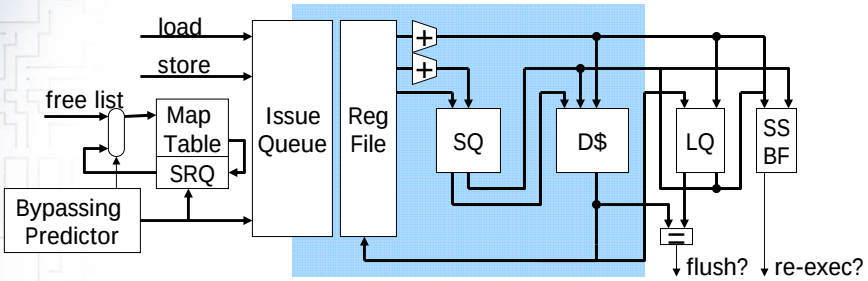


Loads don't need to access store queue during execution

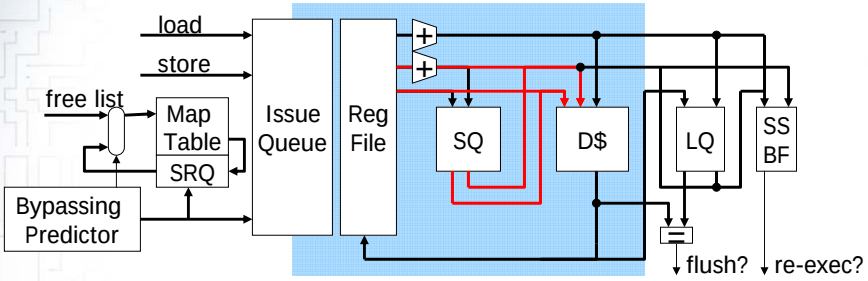
- Bypassing loads: skip out-of-order execution (SMB)
- Non-bypassing loads: get values from the data cache

Remove store queue from the the load path

NoSQ 2: Remove Store Queue



NoSQ 2: Remove Store Queue



Move store execution from out-of-order to commit

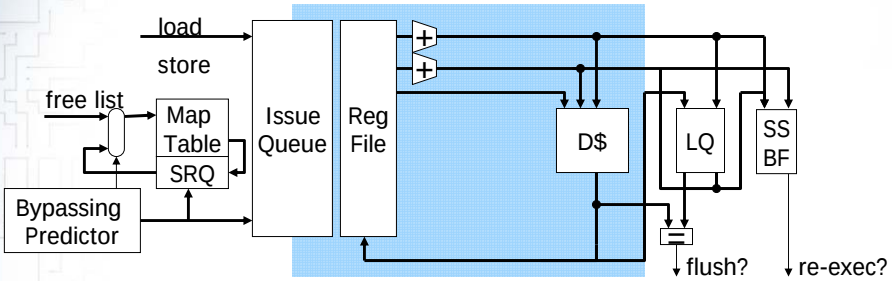
- Extend ROB to remember store registers, offsets and data sizes
- Elongate commit pipeline to read register file, calculate address

+ No additional regfile ports, adders: out-of-order ports → in-order ports

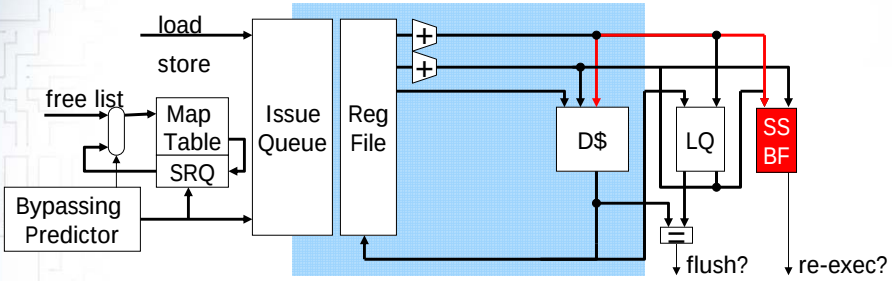
Don't dispatch stores to out-of-order core

Eliminate store queue

NoSQ 3: Remove Load Queue



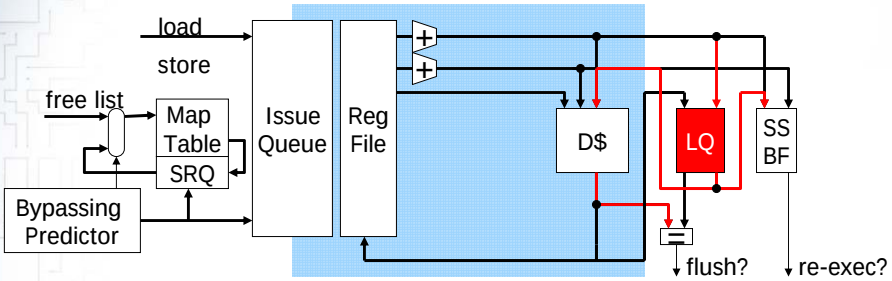
NoSQ 3: Remove Load Queue



Generate addresses for bypassed loads at commit (to verify)

- ROB is already extended, pipeline already elongated

NoSQ 3: Remove Load Queue



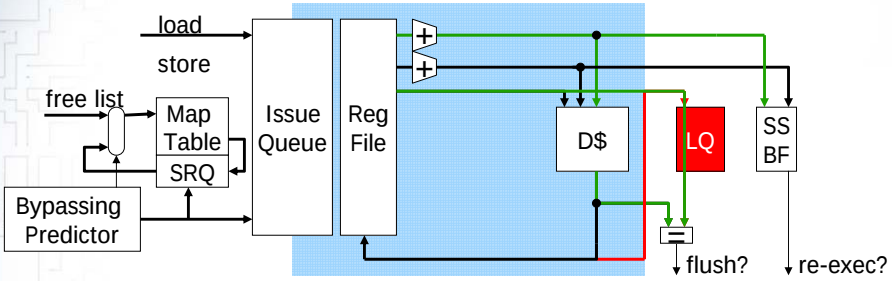
Generate addresses for bypassed loads at commit (to verify)

- ROB is already extended, pipeline already elongated

Re-generate addresses for non-bypassed loads at commit

- May need additional register read port

NoSQ 3: Remove Load Queue



Generate addresses for bypassed loads at commit (to verify)

- ROB is already extended, pipeline already elongated

Re-generate addresses for non-bypassed loads at commit

- May need additional register read port

Eliminate load queue

Road Map

Overview

From conventional to NoSQ

- Conventional store-load forwarding
- Load verification with filtered load re-execution
- Speculative memory bypassing

NoSQ

- Eliminating out-of-order stores and the store queue
- Eliminating the load queue
- NoSQ's store-load bypassing predictor

Evaluation

NoSQ's Store-Load Bypassing Prediction

Similar to previous store-load prediction, but more difficult

Load scheduling prediction: [Chrysos+'98]
- Can predict store conservatively, predicts only violating loads

Speculative forwarding prediction [Sha+'05]
- Must predict all loads, but benefits from store-load address check

Traditional bypassing prediction [Moshovos+'97]
- No address check, but can decline to predict difficult loads

NoSQ's bypassing prediction
- Must predict all loads precisely, no address check

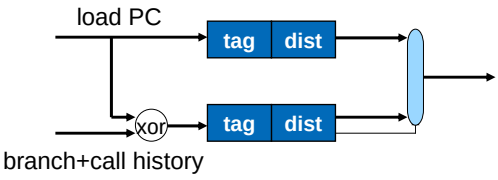
Distance-Based Dependence Prediction

Predictor interface: load PC → dynamic store

Load PC → store PC(s) → dynamic store
- E.g., Store Sets [Chrysos+'98], Speculative indexed SQ [Sha+'05]
  - – Store PC → dynamic store requires table
  - – Can only (easily) represent most recent instance of each store PC

Load PC → distance (in stores) to store → dynamic store
- E.g., [Lipasti+'97, Yoaz+'98]
  - + Can represent any store instance
  - + Dovetails with SVW: just compare/subtract distances/SSNs
    - Predict: load.SSN_{bypass} = SSN_{rename} – load.$distance_{bypass}$
    - Verify: load.SSN_{bypass} == SSBF[load.address]
    - Train: load.$distance_{bypass}$ = SSN_{commit} – SSBF[load.address]

Predictor Design



Each entry includes tag, store distance

Explicitly path-sensitive design

- Two tables: path-insensitive + path-sensitive
  - Both set-associative
- Predict: prefer path-sensitive prediction
- Train: update both tables on every load commit

Evaluation

Goal

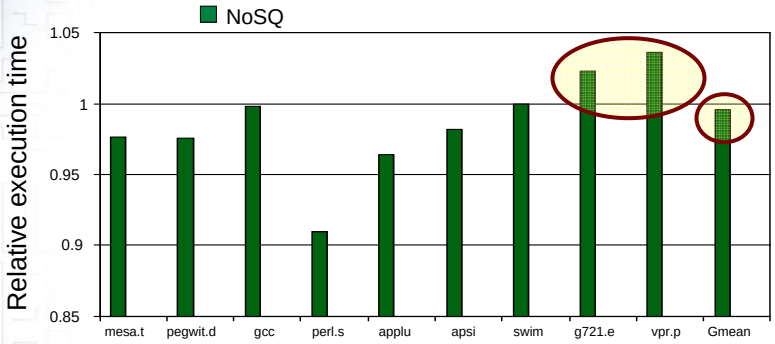
- No store queue or load queue
- Same (or better) IPC as conventional design
  - – Bypassing mis-predictions
  - – Deeper commit pipeline
  - + Latency benefit of SMB
  - + Reduced consumption of issue bandwidth and queue slots

Simulation environment

- SPECint2000, SPECfp2000, Mediabench (only show 9)

- Dynamically scheduled 4-way superscalar
- 128-ROB, 40-entry issue queue, 11-stage front-end/core
- Base: 24/48-entry SQ/LQ, 2K-entry Store Sets, 6 stage commit
- NoSQ: No SQ/LQ, 2K-entry predictor, 8 stage commit

NoSQ Performance



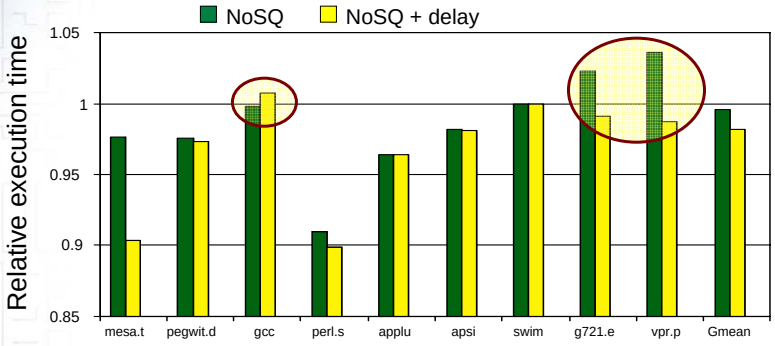
+ On average: slightly outperforms conventional design
  + Prediction accuracy is ~99.8% (15 mis-predictions per 10,000 loads)
– In a few cases: slowdowns
  - 10 of 47 benchmarks >1% slowdown (worst case 7%)

Avoid Bypass Mis-Predictions with Delay

Two kinds of bypassing mis-predictions
- Difficult to predict
  - Long signature, data dependent, predictor conflicts, etc.
- Simply cannot be bypassed
  - Narrow-store to wide-load
- NoSQ can do wide-store/narrow-load and narrow-store/narrow-load
  - See paper

Catch-all: convert bypassing to delayed non-bypassing load
- Inject it to the out-of-order core
- But delay it until the predicted bypassing store commits
- Load gets value from data cache
- Attach a confidence counter to each predictor entry

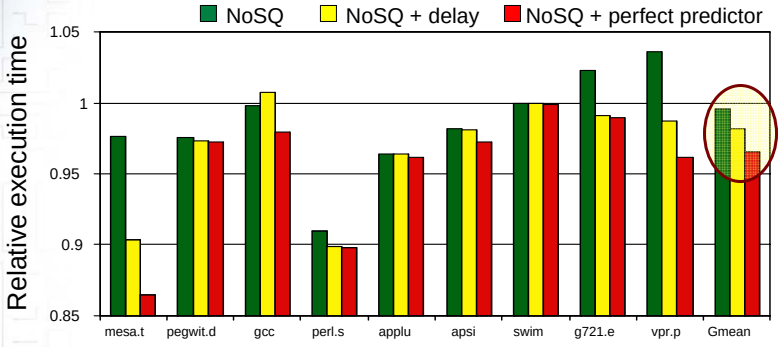
NoSQ with Delay



+ Robust performance: only 1 of 47 benchmarks > 1%
  – Overdelay hurts sometimes, but not too bad

+ As fast or faster than conventional in almost all cases

NoSQ with Perfect Predictor



+ Only 2% between realistic and perfect predictor

See paper for

- Predictor scalability, data cache accesses, etc.

Conclusions

Conventional store queue / load queue

– Complex, non scalable
– Exist for the benefit of 10% forwarding loads

NoSQ

- Exploits synergy of previously proposed mechanisms
  - Re-execution with SVW filter, speculative memory bypassing
- New highly accurate bypassing predictor

+ Simple, clean data path with no store queue/load queue
  - More scalable out-of-order core
  - Fits well with distributed, partitioned cores (e.g., SMT)

+ Outperforms conventional design