Transactional Execution: Wait-Free HW Memory Ordering

Babak Falsafi

Team Members: Chi Chen, Shelley Chen, Mike Ferdman, Nikos Hardavellas, Jangwoo Kim, Stephen Somogyi, Tom Wenisch, Se-Hyun Yang
Alumni: Cem Fide, Chris Gniady, An-Chow Lai

Impetus Group
Computer Architecture Lab (CALCM)
Carnegie Mellon University
http://www.ece.cmu.edu/~impetus

The Big Debate: HW Memory Order

When shall hardware enforce order?

Always order:

- Always
  - » Software wants a “multiprogrammed CPU” behavior
- Easy to program & understand
- Assumed to exhibit inferior performance

Relax order:

- Order enforced through software annotation
- Let the hardware overlap write latency

The Big Misconception: A Large Performance Gap

But, strong ordering thought to hurt performance

Not true!

Memory only has to appear to be ordered

- Hardware can relax order while checkpointing state
- Roll back if relaxed order observed by others
- Result → SC + Speculation ≥ RC!

This is the Bart Simpson's approach to relaxing order:

"I didn't do it. Noone saw me doing it!"

The Case for Transactional Execution

Give me your favorite hardware model

- I will give you a wait-free implementation
- E.g., never wait for store acks in SC, or fences in RC

Even better:

- Expose the transactional model to software
- Let software trigger checkpoint, rollback
- Remove locks all together
- Allow for aggressive compiler optimizations
- And a number of other applications

Outline

- Overview
- Speculative Memory Ordering
- Other Applications
- Related Work
- Conclusions

Strong Memory Ordering

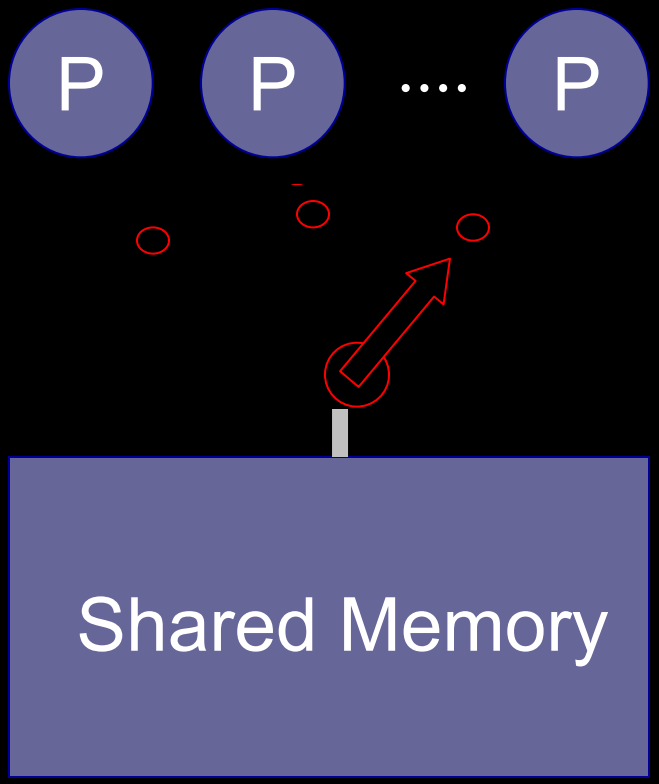
Sequential Consistency (SC) [LAMPORT]

Memory should appear

- in program order & atomic
- e.g., critical section
  - ① lock
  - ② modify data
  - ③ unlock

\+ intuitive programming

– slow implementations!



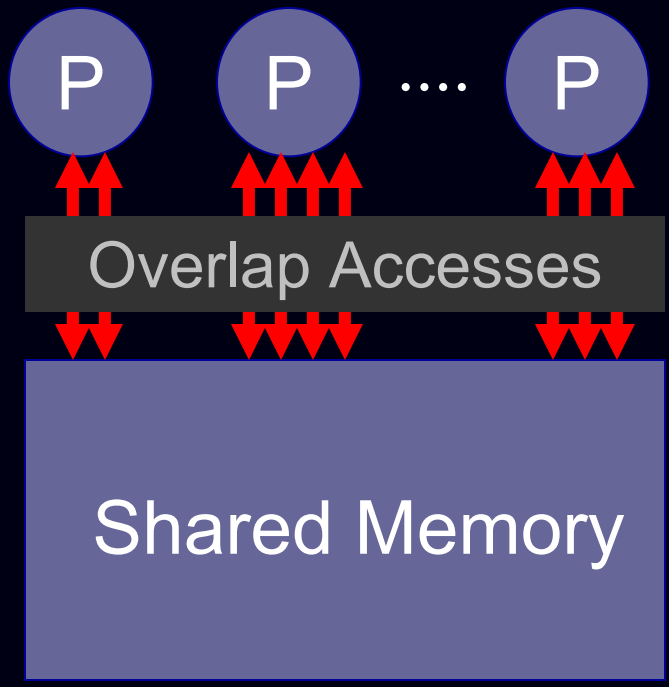
Relaxed Memory Ordering

Overlap memory accesses

- software enforces order (e.g., first lock, then data)
- special “ordering” instructions

E.g., Release Consistency (RC)

- [Gharachorloo, et al.]
- allows any (re-)ordering
- \+ faster implementations
- – careful SW annotation
- – always enforces order at fence



Can We Build a Wait-Free SC?

Observation [Gniady et al., ISCA'99]:

- SC must only appear in program order
- need order only when others race to access

SC hardware can emulate RC iff

- overlap accesses speculatively
- checkpoint state in program order (transaction)
- roll back in case of a race

+ no help from software → SC programming
+ infrequent rollback → better than RC performance

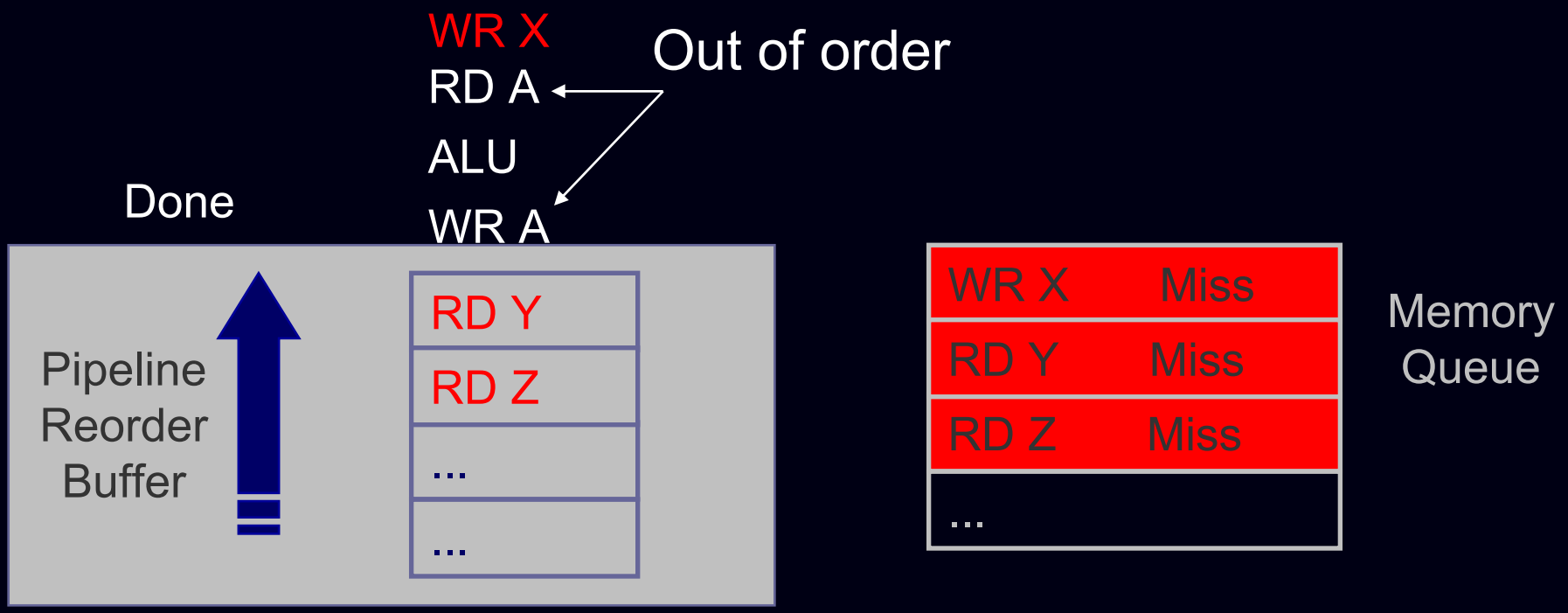
Execution in SC Memory System

- WR X, RD Y, RD Z access remote memory
- Either all addresses unrelated → need not be ordered
- Or common case → no contention on related addresses

① WR X blocks pipeline

② Can not overlap RD Y & RD Z with WR X

Execution in RC Memory System



- ❑ Software guarantees X, Y, Z, A are unrelated
- \+ Accesses to A complete while WR X is pending
- \+ Overlaps remote accesses to X, Y, Z

Execution in RC with Fence

- If addresses related, the fence operation blocks the pipeline
- Must wait for prior accesses to complete
- Long memory latency is exposed

SC++: Transactional Execution of SC

Transaction semantics:

- Start with a pending store
- Commit with a store acknowledgement
- Roll back when access by another CPU to a speculatively-accessed out-of-program-order block

Required HW for SC++

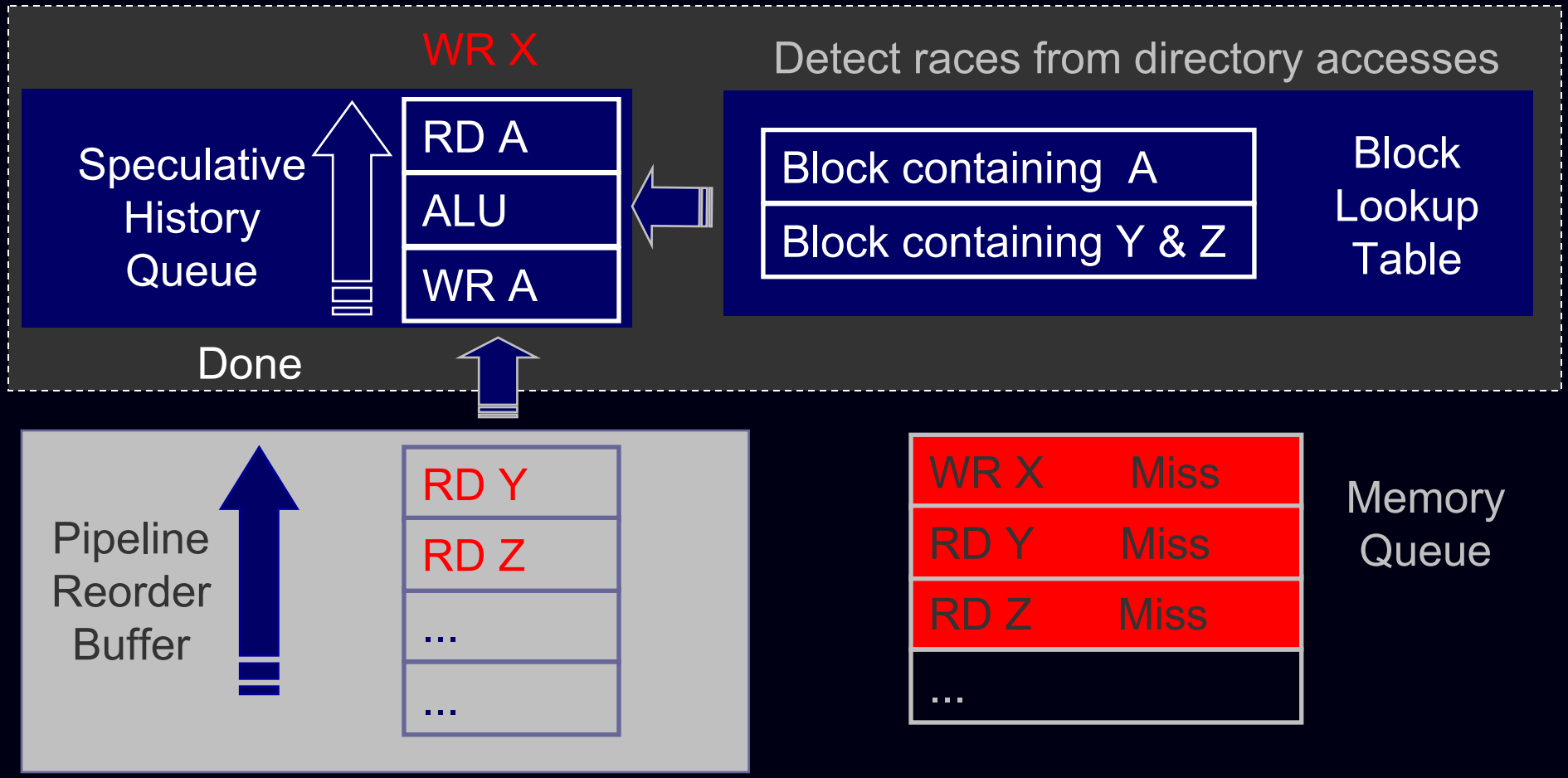
① H/W support for relaxing all order

② Checkpointing storage to tolerate memory latency
- Old CPU state
- Old memory state

③ Fast lookup to detect possible order violation
- upon cache invalidations and replacements

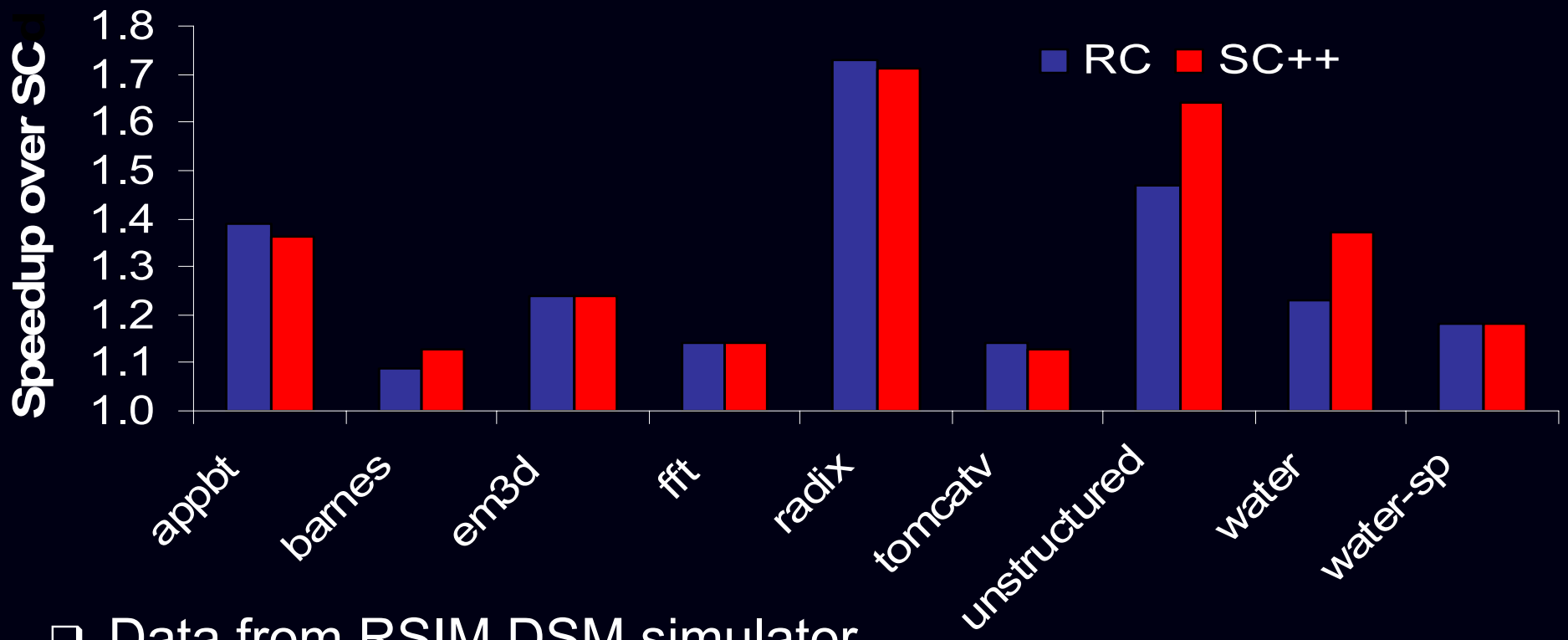
④ Infrequent rollbacks
- Typical of well-behaved parallel applications
- Rollbacks are due to false sharing or data races

A Design for SC++



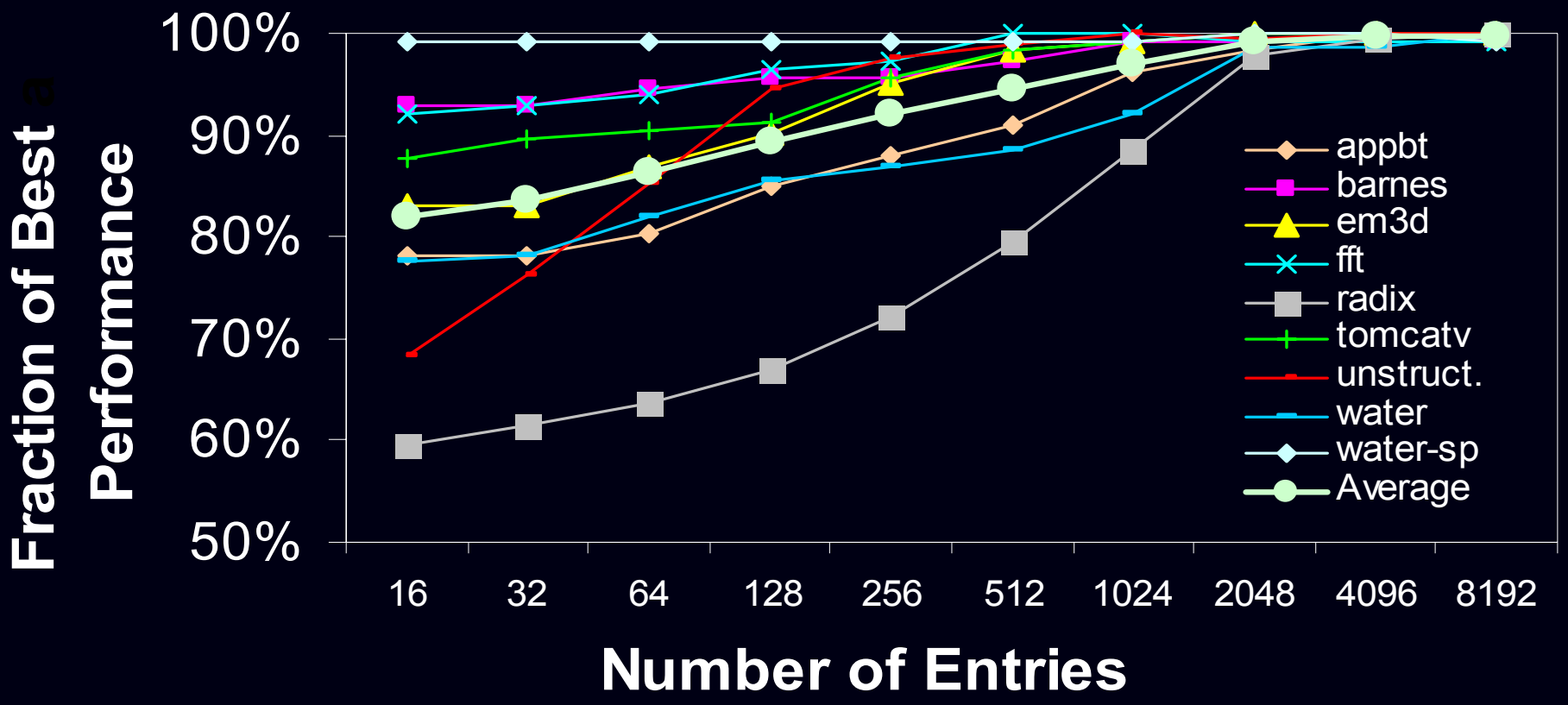
- Queue maintains computation history
- Table allows quickly detecting races

Performance Comparison [ISCA'99]



- Data from RSIM DSM simulator
- 16, 1 GHz MIPS R10000 processors
- Up to 70% gap between SC & RC

Sensitivity to Queue Size [PACT'02]



- Queue size varies across apps (& systems)
- History is highly bursty
- Can spill history to L2

The HW Memory Ordering Debate

Must MP HW support relaxed memory models for high performance?

Our answer:

No! Transactional execution can bridge the performance gap among HW memory models!

Can We Build a Wait-Free Relaxed System?

This helps get dusty-deck SW to run fast

- Solaris is TSO-compatible
- Can implement a wait-free TSO

Relaxed systems always enforce order at fence ops

Transaction semantics:

- Start with a pending fence op
- Commit with the acks to accesses prior to fence
- Roll back in case of race to accesses prior to fence

What Else Can We Do With Transactions?

1. Value prediction in hardware
   - Relaxes order inadvertently [Sorin et al.]
2. Lock/synchronization elision
   - Also proposed as “Transactional Memory”
   - Avoid the lock accesses altogether
3. Data race detection
4. Transient-error detection/recovery

There will soon be some form of checkpoint/recovery built in!

Transactional Execution: The Compiler Perspective

Aggressive compiler optimizations

- can lead to incorrect code

Expose the transactions to SW

- compiler triggers checkpoint/commit
- hardware does the rollback detection/recovery
- rollback allows for alternate code to execute

Can substantially reduce

- error detection overhead
- recovery overhead

Ongoing work with Markus Mock at UPitt

Related Work

Transactional execution to relax order:

- Speculatively-relaxed Loads
  - » [Gharachorlou et al.] → MIPS R10K
- Speculative Retirement [Adve et al.]
- Forecast on bridging the perf. Gap [Hill]

Other work on Transactional Execution:

- Transactional Memory [Moss & Herlihy]
- Speculative synchronization [Rajwar & Goodman, Torrellas et al.]
- Data race detection [Torrellas et al., Hill et al.]
- Fault-tolerant DSM [Sorin et al.]

Conclusions

Wait-free implementation of memory order

- Bridges the performance gap among HW models
- HW overhead depends on the model assumed

Transactional execution:

- Can provide wait-free implementations of systems
- Has other key applications
- Can be exposed to SW for aggressive compiler opts

For More Information

Please visit our web site

Impetus Group
Computer Architecture Lab (CALCM)
Carnegie Mellon University
http://www.ece.cmu.edu/~impetus