**Effective and Inexpensive (Memory) Race Recording**

Min Xu  
Thesis Defense  
05/04/2006

Electrical and Computer Engineering Department, UW-Madison  
Advisors: Mark Hill, Rashidov Bozik  
Committee: Remzi Arpaci-Dusseau, Miklos Lipasti, Barton Miller, David Wood

---

### Outline

1. Overview  
   - Increasingly useful to **replay multithreaded** code  
   - Race recording: key to dealing with nondeterminism

2. Thesis Contributions
   - Small Log Size  
   - Effective  
   - SC & TSO Applicability  
   - Low Runtime Overhead  
   - Inexpensive  
   - Order-Value Hybrid  
   - Set/LRU Approximation  
   - Low Cost Hardware

3. Motivation & Problem

4. Multithreaded Debugging

---

**A Case Study**

- Log recording: 1 byte/kilo-instr  
- Always-on recording: less than 2% overhead  
- Low cost: 24 KB RAM/core  
- Support both SC & TSO (x86-like)
Race Recording

Thread 1

Thread 2

Recording

X = 6

Replay

X = 6

A Good Race Recorder

Low runtime overhead

Applicability

Low cost

Long recording: small log

Desired & Existing Race Recorders

<table>
<thead>
<tr>
<th>Desired Recorder</th>
<th>Recording Length</th>
<th>Applicability</th>
<th>Overhead</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTR</td>
<td>Small Log Size</td>
<td>MP</td>
<td>Racy Code</td>
<td>SC</td>
</tr>
<tr>
<td>Log</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Little Hardware</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Our Recorder</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Problem Formulation

Reproduce exact same conflicts: no more, no less
Log All Conflicts

Dependence Log

Log I: 2→3
3→5
4→6

Log I: 2→3

Log Size: 5*16=80 bytes
(10 integers)

But too many conflicts

Netzer’s Transitive Reduction

TR Reduced Log

Log I: 2→3
3→5
4→6

Log I: 2→3

Log Size: 64 bytes
(8 integers)

The Intuition of the New RTR Algorithm

Regulate Replay (RTR)

Stricter Dependences to Aid Vectorization

New Reduced Log

Log I: 2→3

Log I: 2→3

Log Size: 48 bytes
(6 integers)

Compress Vectorized Dependencies

Vectorized Log

Log J: x=3, y=1
Log I: x=3, y=1

Log Size: 40 bytes
(5 integers)

Reduce log size to KB/core/second
Detect Conflicts

A.readers.add(I, 1)
B.writer = (I, 2)
if (C.writer != I)
    log(WAW)
foreach C.readers
    if (reader != I)
        log(WAR)
C.readers.clear()
C.writer = (I, 3)
B.writer = (I, 2)
C.writer = (J, 2)
if (B.writer != J)
    log(RAW)
B.readers.add(J, 3)
...

Expensive in software

Detect conflict in hardware with little runtime cost

Cache Evictions and Writebacks

OK with nonsilent eviction & directory eviction

Implement TR and RTR in Hardware

Ideal TR requires vector timestamps
  - Too expensive
  - New idea: Pairwise-TR (use scalar timestamp)
  - Enable pairwise transitive reduction

Optimal RTR algorithm is likely expensive
  - Implement a greedy RTR algorithm
  - One-pass, online algorithm
  - Keep a sliding window of vectorizable dependencies

Hardware Implementation
Correct, but more evictions \( \rightarrow \) more logged conflicts

Set/LRU better preserve reducibility
Small $\rightarrow$ more misses $\rightarrow$ but still small log

Decoupling $\Rightarrow$ Small timestamp memory (Set/LRU)
- e.g., 32-set, 64-way $\Rightarrow$ 99% transitive reduction
- Timestamps Memory $\Rightarrow$ 24 KB

From 192 KB to 24 KB: 8x reduction
Recording with Total Store Order (TSO)

Majority of existing MP are non-SC

TSO is well defined, x86-like

Order-Value-Hybrid Recording

Hybrid Recording with TR and RTR

Hybrid recording

- All loads get correct values
- Hardware similar to OoO SC [Gharachorloo et al. ’91]

Hybrid + TR & RTR

- TR will not use the omitted WAR in reduction
- RTR vectorize dependencies more conservatively

Evaluation Method & Results
Simulation Method

Commercial server hardware
- GEMS: http://www.cs.wisc.edu/gems
- Full-system (OS + application) executions
- 4-core CMP (Sequential Consistent)
  - 1-way in-order issue, 2 GHz,
  - 64KB I/D L1, 4MB L2, 64byte lines, MOSI directory

Commercial server software
- Apache - static web serving
- SpecJBB - middleware
- OLTP - TPC-C like
- Zeus - static web serving

Log Size: 1 byte/kilo-instr

Well within the capability of current machines
- Long recording (days - months) need improvement

Runtime Overhead

Execution Time

Interconnection Msg. B/W

Benefits of RTR and Set/LRU (Log Size)

Why RTR and Set/LRU Work Well?

RTR
- Processors execute instructions at similar speed
- Therefore, we can find “vectorizable” dependencies

Set/LRU
- Temporal locality makes the LRU timestamps old
- We only need to know if a timestamp is "old-enough"

Sensitivity and Scalability

A design space of the timestamp memory (TSM)
- Size: smaller TSM -> larger log
- Read/write timestamp: should be used when TSM is large
- Partial timestamp: 24-bit enough
- Associativity: higher better for RTR

Scalability of the recorder
- Studied with modest processors (2 to 16p)
- Commercial workloads, not scientific workloads
- Log size increase slowly with number of cores
Conclusion & My Other Research

Race Recording

- Race recording → Key to combat nondeterminism
- My thesis → An effective & inexpensive Recorder
  - RTR algorithm → small log size
  - Coherence piggyback → Negligible slowdown
  - Timestamp approximation → Low hardware cost
  - Order-value hybrid → support SC & TSO

Future work
- Improve race recording algorithm
- Improve race recorder implementation
- Study race replay

Serializability Violation Detector [PLDI’05]

- Like a race detector
- No a priori annotation requirement
- Intend to detect bugs “actually” happen
  - Check for a 2-Phase-Locking condition

Publications

- FDR (ISCA’03)
  - Adopted by UCSD BugNet (ISCA’05)

- SVD (PLDI’05)
  - Cited by Vaziri et al. (POPL’06)
  - Influenced new data race definition

- RTR, Set/LRU & Hybrid
  - Submitted for publication

Acknowledgements

- Joint work with my advisors
  - Mark Hill, Ras Badik

- Ph.D. Committee
  - David Wood, Mikko Lipasti, Remzi Arpaci-Dusseau, Barton Miller

- Multifacet Group
  - Milo Martin, Dan Sorin, Carl Mauer, Brad Beckmann, Kevin Moore, Alaa Alameldeen, Mike Marty, Luke Yen

- Affiliates & Companies
  - Joe Emery, CJ Newburn, Peter Hsu, Bob Sok, Eric Bach, Gang Luo, Alex Chow, IBM, Intel, Microsoft, Sun
Deterministic Replay is Useful

Deterministic Replay is logically recreating a program execution

Present applications
- Cyclic Debugging ([Pancake & Netzer '93])
- Fault Tolerance (ExtraVirt [Lucchetti et al. '05])
- Intrusion Analysis (ReVirt [Dunlap et al. '02])

Future applications
- Data Recovery
- Replay-based Synchronization

Multicore and Multithreading

Multicore is common
- AMD X2
- IBM Power 5/6, Cell
- Intel Pentium D, Core Duo
- Sun SPARC T1

Multithreading is common
- Server: high throughput
- Scientific: high performance
- Desktop/embedded: low response time

Race Recording: Key to Determinism

Races: general race & data race [Netzer & Miller]
- Both cause nondeterminism
- Race recording can help, but

Existing race recorders are inadequate
- Some generate large logs
- Some have high runtime overhead
- Some have high hardware cost (space overhead)
- Support only sequential consistency

Need a better race recorder

Deterministic Replay & Fault Tolerance

Fault Recovery
- Replay after a failure

Fault Detection
- Replay then compare

Recording/Replay & Debugging

Online Recorder

Deterministic Replayer

Future: Record/Replay & Undo/Redo

VM as a software platform
- Ease software development
- Fine granularity in Undo and Redo
**Future: Replay-based Synchronization**

- **Three steps**
  - Coarse-grain sync. → fine-grain sync. → hardware sync.

- **Results**: higher performance

  - Works only if static control flow & fixed data addr
  - DSP kernels

**Race Recording Related Work**

<table>
<thead>
<tr>
<th>Total-order recorders</th>
<th>Partial order recorders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bacon '91 (Hardware)</td>
<td>Instant Replay '87</td>
</tr>
<tr>
<td>ReplPlay '00</td>
<td>Netzer '93</td>
</tr>
<tr>
<td>Zook '04</td>
<td></td>
</tr>
<tr>
<td>R &amp; R '90</td>
<td></td>
</tr>
<tr>
<td>Dalal '91 (Hardware)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bus transactions</th>
<th>Support Clocks</th>
<th>Scheduling</th>
<th>Variable version</th>
<th>Vector clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large lag</td>
<td>Small lag</td>
<td>Large lag</td>
<td>Low overhead</td>
<td>High overhead</td>
</tr>
<tr>
<td>Low overhead</td>
<td></td>
<td>Large lag</td>
<td>High overhead</td>
<td></td>
</tr>
<tr>
<td>(Sync only)</td>
<td></td>
<td>Small lag</td>
<td>Low overhead</td>
<td></td>
</tr>
<tr>
<td>(Non-MP)</td>
<td></td>
<td>Large lag</td>
<td>High overhead</td>
<td></td>
</tr>
</tbody>
</table>

**Correctness of Order-Value-Hybrid**

**Removing WAR dependencies**

- Say thread I read, thread J write
- Removing the WAR affects I’s read, not J’s write
- But, for every dependence removed, thread I reads correct value from the value log
- Therefore, all reads get the correct value

**TR and TSO**

**TR affects dependencies reduced by a WAR**

- The WAR itself may later be removed during replay
- Solution: Not use WAR in TR if the WAR can be removed
- Respond with a special flag when a loaded cache line is stolen

**RTR and TSO**

**The sliding window may expose the ordered loads**

- Shrink the sliding window to avoid it

**Deadlock Avoidance of RTR**

- Avoid deadlock by adhere to a SC total order

**Replay Cycle**

1. i:4  j:1  j:2  i:3  j:4
Recording Race-free Executions

No data races

Only need to record synchronization race

Deterministic replay up until the first data race

Replay Parallelism

Replay performance depends on

(1) Number of synchronizations
(2) Extra wait incurred by the synchronizations

Directory Protocols

Add sticky states in the directory
• Retain states after writebacks
• Need extra acknowledgements

Or, add extra timestamp memory in the directory
• Helps to avoid extra acknowledgements

A tradeoff
• Sticky states can be cheaper
• But extra timestamp memory can be faster

Snooping Protocols

Key problem is combined/implicit response
• Not a problem for AMD Hammer

Nonsilent Evictions

Out-of-Order & Hardware Prefetching

Speculative execution
• No IC assigned yet

Hardware prefetching
• No IC assigned

Key idea: receive observation
• Can associate a ld/st with current commit instruction

Directory eviction: more false conflict, like snooping
Unordered Messages in Interconnect

Message arrive out-of-order
Can affect reduction
But better add a sequence number
  • Reconstruct the message order
  • Enable IC compression by sending deltas

Integer Overflow

IC and timestamps may overflow
IC: make it 64bit, will not overflow for a long time
Timestamps: use approximation techniques
  • MSB of IC + LSB of Timestamps

Varying TSM Size

Varying Associativity

Varying Partial Timestamp Width

Log Size Scaling
In Retrospect ...

What are you most proud of?
- RTR improves TR after 13 years

What would you do differently if doing it again?
- "replaying me is deterministic" (just kidding)
- I wish I focused on race recording earlier

What the industry should do?
- Implement the recorder as a VMM extension