A “Flight Data Recorder” for Enabling Full-system Multiprocessor Deterministic Replay

Min Xu, Rastislav Bodik, Mark D. Hill
http://www.cs.wisc.edu/multifacet
June 9th, 2003

- Software bugs cost **time & money**
- Hardware is getting **cheaper**
- Use hardware to aid software debugging?
Brief Overview

Approach: Full-system Record-Replay
- Add H/W “Flight Data Recorder”
- Target cache-coherence multiprocessor server
- Enables S/W deterministic replay

Full-system Evaluation: Low Overhead
- Piggyback on coherence protocol: little extra H/W
- Non-trivial recording interval: 1 second
- Negligible runtime overhead: less than 2%
- Can be “Always On”
Outline

Overview
- Why Deterministic Replay?
- The Debugging Scenario
- The Solution

Recording Multithreading

Recording System State & I/O

Evaluation

Conclusions

Efficient Recording

With full-system commercial workloads
Why Deterministic Replay?

Software Bugs Happens In the Field
- Differences between development & deployment
- Data races (Web server, Database)
- I/O interactions (OS, Device Driver)

Debugging Usually happens In the Lab
- Need to replay the buggy execution

Use Core Dump?
- Captures the final application state
- Not enough for “race” bugs

Need Better “Core Dump”
- Enable faithfully replaying prior to the failure
The Debugging Scenario

Recorder

P1  Store log A  P2
P3  Store log B  P4
P4  Store log C

Checkpoint A  Checkpoint B  Checkpoint C

Crash

Dump “Core”

Replayer

Read Checkpoint B

Replaying from log B, C

Crash
The Solution

**Online Recorder**
- Like airplane flight data recorder
- "Always on" even on deployed system
- H/W based (no change to S/W)
  - Transparent to S/W
  - Minimal performance impact

**Offline Replayer**
- Post-mortem replay of pre-crash execution
- Possibly on a different machine off-site
- Based on existing technology
  - i.e. Simics full-system simulator

Focus of this work
Not emphasized in this work
Outline

Overview

Recording Multithreading
  - What to record?
  - An example
    - Practical recorder hardware

Recording System State & I/O

Evaluation

Conclusions
What to Record?

Multithreading Problem
- Record order of instruction interleaving

Assume Sequential Consistency (SC)
- Accesses (appear to have) total order
Previous Record-Replay Approaches

**InstantReplay ’87**
- Record order or memory accesses
- overhead may affect program behavior

**Netzer ’93**
- Record optimal trace
- too expensive to keep track of all memory locations

**Bacon & Goldstein ’91**
- Record memory bus transactions with hardware
- high logging bandwidth

**RecPlay ’00**
- Record only synchronizations
- Not deterministic if have data races
Our Approach

Uses existing cache coherence hardware
- Low overhead, not affect program behavior
- Works for program with races
- Adapts Netzer’s algorithm in hardware
- only record sync. if data race free

An Example
- Progressively refine the recording algorithm
Example: Record SC Order

\begin{align*}
\text{Flag} &= 1 \\
X_1 &= 5 \\
X_2 &= 6 \\
\text{Flag} &= 0
\end{align*}

\begin{align*}
\text{Flag} &= \text{Flag} \\
\text{Bneq} \ r1, r0, -1 \\
\text{Nop} \\
\text{Bneq} \ r1, r0, -1 \\
\text{Nop} \\
Y &= X_1 \\
Z &= X_2
\end{align*}
Example: Record SC Order

Need to add processor instruction count (IC)
The very same interleaving is recorded, but
Example: Record Word Conflict Order

Recording just word conflict can enable deterministic replay

Hard to remember word accesses and too many arcs ...
Example: Record Block Conflict Order

4 Flag=1
5 X1 := 5
6 X2 := 6
7 Flag := 0

i:4 → j:15

i:7 → j:18

i:5 → j:21

i:6 → j:22

15 $r1 := Flag$
16 Bneq $r1,$r0,-1
17 Nop
18 $r1 := Flag$
19 Bneq $r1,$r0,-1
20 Nop
21 Y := X1
22 Z := X2
Example: Record Block Conflict Order

Need to remember last accessing IC in the cache
But, can we do better?
Example: Apply Transitive Reduction
Example: Apply Transitive Reduction

Three arcs! No need to know syncs
Automatic sync only for race free program
Practical Recorder Hardware

Processor
- instruction count
  - 4 bytes per processor

Cache
- last access instruction count
  - 6.25% space overhead

Coherence Controller
- vector of instruction counters
  - $3 \times 4$ bytes per processor for 4-way multiprocessor

Finite Cache, Out-of-Order, Prefetch, etc.
- Recorder still applicable
- Details in the paper
Outline

Overview

Recording Multithreading

Recording System States & I/O
  – SafetyNet checkpoint hardware
  – Interrupts, I/O, DMA

Evaluation

Conclusions
SafetyNet Checkpoint Hardware

Problem

- To beginning of “replay” interval
- Logically take a snapshot of the system

Solution

- Adapt SafetyNet [Sorin et al. ISCA ‘02]
  - Processor Checkpointing
  - Memory Incremental logging
  - Slightly modified for longer interval
Recording I/O

Interrupts
- Not exceptions
- Record Interrupt type & IC

Instruction I/O
- Load: record values
- Store: ignored

DMA
- Record input values
- Record ordering: as pseudo thread
Outline

Overview
Recording Memory Races
Recording System State & I/O

Evaluation

– An example system
– Simulation methods
– Runtime, log size

Conclusions

With full-system commercial workloads
Target System

Commercial Server H/W

- Sequential Consistent CC-NUMA
- Full I/O: Interrupt, DMA, etc.
- Simulation system (Simics + Memory Simulator)
  - 4 way in-order issue, 1 GHz, 4 processors
  - 128KB I/D L1, 4MB L2, MOSI directory protocol

Commercial Server S/W

- Unmodified commercial server benchmarks
  - Apache, Slash, SPEC JBB, OLTP
An Example System

- Core
- Cache(s)
- Cache Controller
- DMA Interface
- Data Compressor (LZ77)
- Recorder Memory
- Memory Banks
- Directory

Flows:
- Interrupts, I/O
- Cache Checkpoint
- Memory Races
- DMA Content & Order
- Memory Checkpoint
- Cache Checkpoint
- Memory Races
Runtime Overhead

Slowdown

- Less than 2%
- statistically insignificant for 2 workloads
- No problem “always on”

Slowdown causes

- Extra traffic
- Stall by buffer overflow
- More blocking
- Extra coherence message on some get-shared’s
Log Size

1 – 1.33 Second Recording
- Buffer: 35 MB (7%); Bandwidth: 25 MB/Second/Processor

Efficient Race Log
- Longer recording is possible with better checkpoint scheme

Longer Recording
- Using disk can get longer replay: 320 GB disk = ~3 hours recording
Conclusion

Low Overhead Deterministic Replay
- Piggyback MP cache coherence hardware
- Modest extra hardware
- Modest overhead (less than 2% slowdown)
  - Minimal race recording with transitive reduction

Full-system Deterministic Replay
- Evaluated with commercial workloads
- Full-system recording (including OS, I/O)
Thank You

Questions?
## Flight Data Recorder vs. ReEnact

<table>
<thead>
<tr>
<th></th>
<th>Flight Data Recorder</th>
<th>ReEnact</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target System</strong></td>
<td>CC-NUMA</td>
<td>TLS</td>
</tr>
<tr>
<td><strong>Deterministic Replay?</strong></td>
<td>Yes</td>
<td>Yes*</td>
</tr>
<tr>
<td><strong>Race-detection?</strong></td>
<td>No**</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Effective Interval (instructions)</strong> Slowdown</td>
<td>&gt;100,000,000</td>
<td>&lt;100,000</td>
</tr>
<tr>
<td><strong>OS, I/O</strong></td>
<td>Yes</td>
<td>No (extendable?)</td>
</tr>
<tr>
<td><strong>Active during OS &amp; I/O?</strong></td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

* Need to disable TLS?
** Not in the recorder, but in the replayer
Scalability

More processors, more races log
- Not a quadratic increase
- e.g. 4p to 16p for 2x more log

Real systems have more I/O
- But, also more memory available for log
Protocol Changes

Get IC count from source processor
- W→R: Piggyback IC count to DataResponse msg
- W→W: Piggyback IC count to DataResponse msg
- R→W: Piggyback IC count to InvalidateAck msg

Cache block Writeback
- Snooping protocol
  - Eager IC update
  - Extra messages on interconnect
  - Not on critical path
- Directory based protocol
  - Lazy IC update
  - Extra latency for cache misses
Replayer (Full-system Simulator)

Input data to the replayer
- Checkpoint
- Execution log
- DMA log
- I/O log
- Exception log

Replay the execution
- Load system checkpoint: registers, TLB, etc
- Replay the MP execution order in partial order
- Replay the I/O and exceptions
- Proper device model needed to interrupt system output
- Memory inspection support
- Step forward/backward (enhanced debugger features)
Example: False Sharing

```plaintext
Example:

P1

31 Flag:=1
32 X1:=5 15 $r1:=Flag
33 X2:=6 16 Bneq $r1,$r0,-1 17 Nop
34 Flag:=0 18 $r1:=Flag
35 Private1:=3 19 Bneq $r1,$r0,-1
   
   20 Nop
   
   21 Y:=X1
   
   22 Z:=X2

P2

14 Private2:=2
15 $r1:=Flag
18(P1,34)
21(P1,32)
22(P1,33)
```
Example: False Sharing

```plaintext
19 Private2 := 2
20 Nop
21 Y := X1
22 Z := X2
```

### Notes
- `P1` and `P2` are two parallel processors.
- Lines 31-34 illustrate false sharing.
- `Flag := 1` in `P1` and `Flag := 0` in `P2` cause false sharing.
- `Private1 := 3` and `Private2 := 2` in `P1` and `P2` respectively.
- `Bneq $r1, $r0, -1` checks if `Flag` is 0.
- Nop instructions are used for synchronization.

---

**Example**: False Sharing

- `Flag := 1` in `P1` and `Flag := 0` in `P2` cause false sharing.
- `Private1 := 3` and `Private2 := 2` in `P1` and `P2` respectively.
- `Bneq $r1, $r0, -1` checks if `Flag` is 0.
- Nop instructions are used for synchronization.
**Example: False Sharing**

```assembly
P1

31 Flag:=1
32 X1:=5
33 X2:=6
34 Flag:=0
35 Private1:=3

P2

14 Private2:=2
15 $r1:=Flag$
16 Bneq $r1,$r0,-1
17 Nop
18 $r1:=Flag$
19 Bneq $r1,$r0,-1
20 Nop
21 Y:=X1
22 Z:=X2
```

Example: False Sharing
Example: False Sharing

P1

31 Flag=1
32 X1:=5
33 X2:=6
34 Flag:=0
35 Private1:=3

P2

14 Private2:=2
15 $r1:=Flag$
16 Bneq $r1,$r0,-1
17 Nop
18 $r1:=Flag$
19 Bneq $r1,$r0,-1
20 Nop
21 Y:=X1
22 Z:=X2

Example: False Sharing
Example: False Sharing

P1

31 Flag=1
32 X1:=5
33 X2:=6
34 Flag:=0
35 Private1:=3

P2

14 Private2:=2
15 $r1:=Flag$
16 Bneq $r1,$r0,-1
17 Nop
18 $r1:=Flag$
19 Bneq $r1,$r0,-1
20 Nop
21 Y:=X1
22 Z:=X2

Xu et al. ISCA'03: Flight Data Recorder 37
Example: False Sharing

P1

31 Flag:=1
32 X1:=5
33 X2:=6
34 Flag:=0
35 Private1:=3

P2

14 Private2:=2
15 $r1:=Flag$
16 Bneq $r1,$r0,-1
17 Nop
18 $r1:=Flag$
19 Bneq $r1,$r0,-1
20 Nop
21 Y:=X1
22 Z:=X2

Xu et al.  ISCA'03: Flight Data Recorder  38