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 ⇐ Efficient Recording

Recording System State & I/O

Evaluation ⇐ With full-system commercial workloads

Conclusions

Why Deterministic Replay?

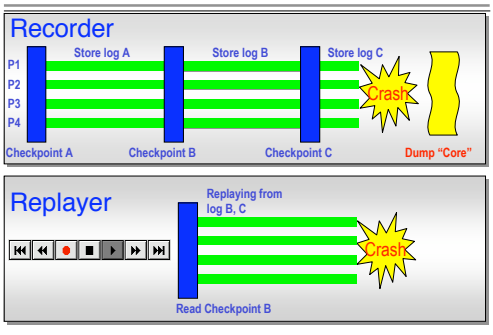
Software Bugs Happens In the Field
- Differences between development & deployment
- Data races (Web server, Database)
- I/O interactions (OS, Device Driver)

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



The Solution

Online Recorder ⇐ Focus of this work
- 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 ⇐ Not emphasized in this work
- Post-mortem replay of pre-crash execution
- Possibly on a different machine off-site
- Based on existing technology
  - i.e. Simics full-system simulator

Outline

Overview

Recording Multithreading ⇐ Efficient Recording
- 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

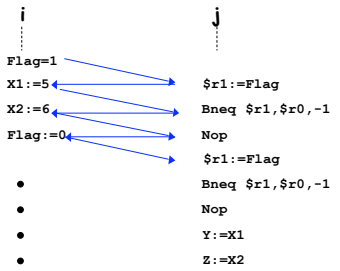
RecPlay '00
- Record only synchronizations
- Not deterministic if have data races

Our Approach

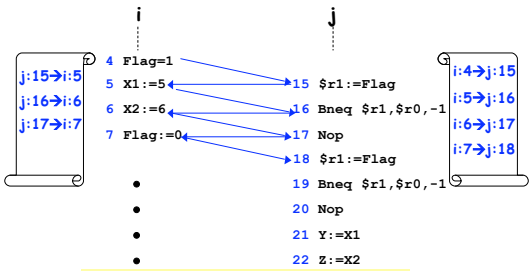
Uses existing cache coherence hardware
- Low overhead, not affect program behavior
- Works for program with races
- Adapts Netzer's algorithm in hardware
- only record sync. if data race free

An Example
- Progressively refine the recording algorithm

Example: Record **SC** Order

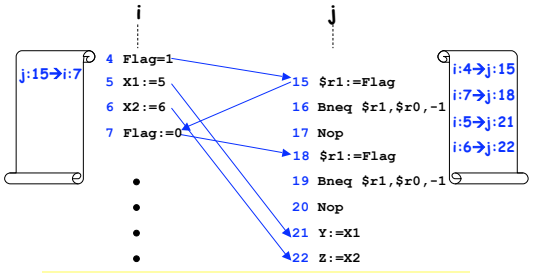


Example: Record **SC** Order



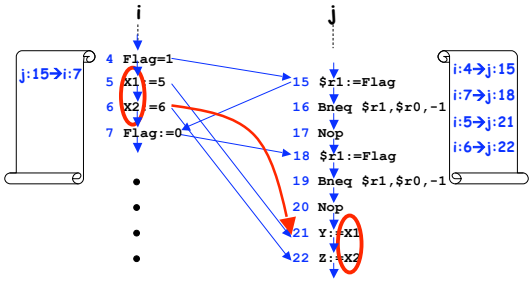
Need to add processor instruction count (IC)
The very same interleaving is recorded, but

Example: Record **Word** Conflict Order

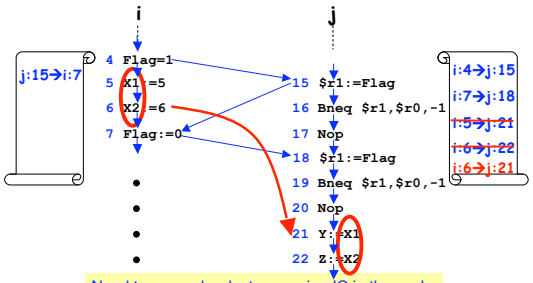


Recording just word conflict can enable deterministic replay
Hard to remember word accesses and too many arcs …

Example: Record **Block** Conflict Order

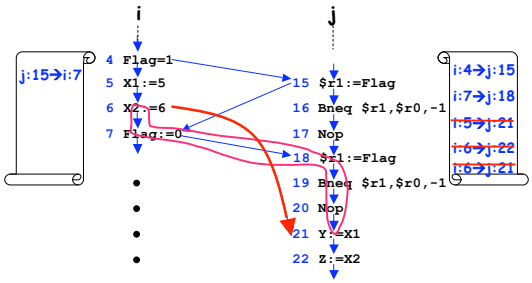


Example: Record **Block** Conflict Order

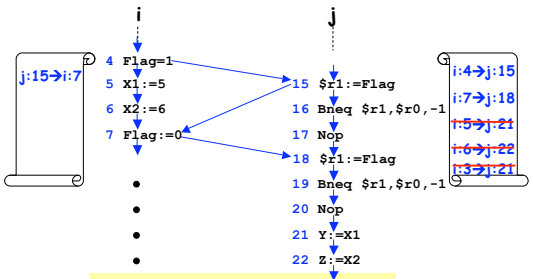


Need to remember last accessing IC in the cache
But, can we do better?

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_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 ⇐ With full-system commercial workloads

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

Conclusions

Target System

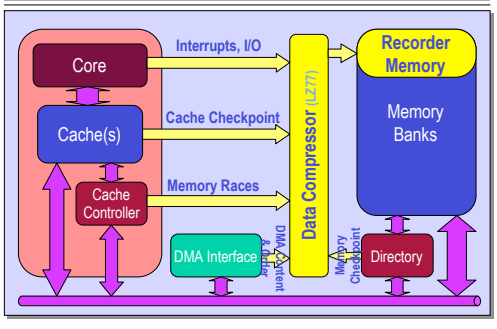
Commercial Server H/W

- 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

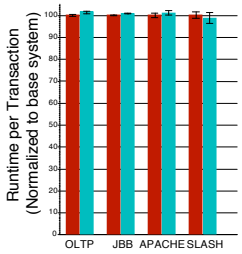
Commercial Server S/W

- Unmodified commercial server benchmarks
  - Apache, Slash, SPEC JBB, OLTP

An Example System



Runtime Overhead



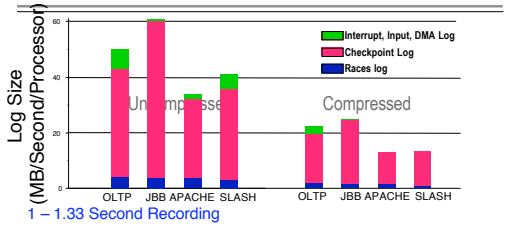
Slowdown

- 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)