Capo:
A Software-Hardware Interface for Practical Deterministic Multiprocessor Replay

Pablo Montesinos, Matthew Hicks, Samuel T. King and Josep Torrellas

Department of Computer Science
University of Illinois at Urbana-Champaign
Motivation: Time Travel

- Allows us to visit and recreate past states and events in computer

- Wide range of uses:
  - Debugging
  - Security

- Enabled by using Deterministic Replay of Execution
How Deterministic Replay Works

- Phase I: **Initial Execution** (a.k.a. **Recording**)

- Phase II: **Replay**
SW-Based Deterministic Replay

- Flexible, integrate well with rest of SW stack
- Very slow or non-applicable to multiprocessor execution:
  - Software is slow at capturing memory access interleaving
HW-Based Deterministic Replay of Multiprocessors

HW can record interleaving of shared-memory accesses effectively:

- Small Memory Access Interleaving Log
- Little overhead

Limitation: integration with SW stack is poor
Limitations of HW-Based Replay of Multiprocessors

- Past proposals focused only on HW primitive for recording and replaying
  - How does it integrate with the SW stack?

- Cannot separate SW being recorded/replayed from the rest
- Paradox: where does the SW that manages the logs go?
- Require complex VMM or simulator to replay execution
- Can't mix recording, replay and normal execution simultaneously in the machine

We must adapt HW-based replay systems and carefully integrate them with SW in order to make HW-based replay practical.
Capo Contributions

- **SW-HW interface** for practical HW-assisted deterministic replay
  - Works with any HW-based replay system

- **Replay Sphere**: new abstraction
  - Isolates SW that is being recorded (replayed) from the rest
  - Separates the responsibilities of the HW and the SW components

- **CapoOne**: Linux-based prototype
Replay Sphere: Isolating Processes

- **Replay Sphere**: Set of threads recorded and replayed as a unit and their address space
- Only user-mode threads run inside spheres
- Threads inside a sphere: R-threads

Replay spheres and processes:
- R-threads that share memory must run within same sphere
- Many processes can run within the same sphere
Replay Sphere: Separating Responsibilities

**HW:**
- Records memory access interleaving of R-threads running within same sphere
- Produces per-sphere **Memory Access Interleaving Log**
- Enforces same memory access interleaving during replay

**SW (Replay Sphere Manager):**
- Logs the other sources of non-determinism that affect the sphere
- Produces per-sphere **Input Log**
  - Includes system call return values, signals, data copied into the sphere...
- Injects data from log into sphere during replay
Other Replay Sphere Manager Responsibilities

- Assign the same virtual memory addresses during recording/replay
- Assign the same IDs to R-threads during recording/replay
- Manage Memory Access Interleaving Log and Input Log
- Manage replay HW resources
Capo’s HW Interface

- Works with any HW-based replay system

- **Per-processor** R-Thread Control Block:
  - Sphere ID register
  - R-Thread ID register

- **Per-sphere** Replay Sphere Control Block:
  - Mode register: specifies whether the sphere is recording or replaying
  - Log pointers: insert to / remove from Memory Access Interleaving Log
Virtualizing the Replay HW

- Replay sphere manager schedules spheres into hardware contexts
Three Key Challenges

1. Ensuring deterministic interleaving when OS copies data into a sphere
2. Using fewer processors during replay than were used during recording
3. Emulating vs. re-executing system calls
OS Copies Data into Spheres

- Problem: interleaving between OS copies and R-threads not recorded
- Solution: insert `copy_to_user` into sphere:
  - HW can log memory access interleaving
  - `copy_to_user` exits sphere once copy is over
Replaying with a Lower Processor Count

- Problem: R-thread that should replay next log entry not scheduled in CPU

- Solution 1: HW detects problem and raises interrupt
  - Efficient, but it requires additional HW and SW support

- Solution 2: SW inspects Interleaving Log and tries to prevent problem
  - Not trivial, requires changes to OS scheduler

- Solution 3: Do nothing, simply wait for OS to schedule R-thread
  - Simple, but can hurt performance
CapoOne: First Capo Implementation

- Simulated replay HW:
  - DeLorean HW system [Montesinos ISCA'08]
  - Augmented with Capo's HW interface

- Modified 2.6.24 Linux kernel
  - Supports replay spheres, R-threads
  - New, deterministic copy_to_user

- Split Replay Sphere Manager:
  - User-level component based on ptrace
  - Kernel-level component schedules spheres and R-threads
Also in the Paper

- CapoOne implementation details
- Lessons learned during CapoOne’s development
- Emulating vs. Re-Executing System calls
- Using Capo with different HW-Based replay systems
CapoOne Evaluation Setup

- Two HW configurations
  - Simulated DeLorean HW replay system (SIMICS): 4 x86 processors
  - Real hardware: 4-Core x86 Intel processor without DeLorean HW

- SW: Ubuntu 7.10 with Replay Sphere Manager
  - Modified 2.6.24 Kernel

- Benchmarks:
  - Scientific Benchmarks: SPLASH-2
  - System benchmarks: Apache, Compilation
Overall Log Size

- Memory Access Interleaving Log takes most of the space
- Small overall log: 3.17 bits/kilo-instruction
Recording Performance

- Moderate overhead: 21% for SPLASH2 and 41% average for system apps
- Minimal timing distortion for debugging concurrency defects
Replay Performance: SPLASH-2

- Emulating system calls reduces cycles during replay
- Replay takes only 80% more cycles
- R-Threads must wait for their turn to commit
Conclusions

- Capo enables practical replay of execution for systems with replay HW
  - The Replay Sphere is a powerful abstraction
  - Enable mixing recording, replay and standard execution

- CapoOne: first Capo prototype
  - Working system
  - Good performance (21-41% recording overhead, 80% replay overhead)
  - Good for debugging concurrency defects
Capo: A Software-Hardware Interface for Practical Deterministic Multiprocessor Replay

Pablo Montesinos, Matthew Hicks, Samuel T. King and Josep Torrellas

Department of Computer Science
University of Illinois at Urbana-Champaign
CapoOne: Basic HW Operation

- Two new per-processor registers: RSID, R-threadID
- Arbiter now supports concurrent spheres
- Manages an Interleaving Log for each of them
Today’s Agenda: Towards the Perfect Replay System

- **DeLorean**: New hardware replay engine
  - Very efficient multiprocessor support
  - Vastly improved log requirements

- **Capo**: New SW-HW interface for replay
  - Makes HW-based replay systems practical

- Evaluation

- Future work
Overall DeLorean System

- Interrupt, I/O and DMA logs are common to other HW-based schemes.
CapoOne: HW Implementation

- No need for DeLorean’s Interrupt Log, DMA Log nor I/O Log
  - PI Log becomes the per-sphere Interleaving Log
  - CS Log becomes a per-R-thread Log

- Chunks only have instructions from one application (or the kernel)
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs
- Some have to be re-executed
  - Thread management (clone)
  - Address space modification (mprotect)
Implicit Dependencies

- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space.
- RSM can express these dependencies to hardware so these interactions can be recorded.