QuickRec: Prototyping an Intel Architecture Extension for Record and Replay of Multithreaded Programs

**Intel:** Gilles Pokam, Klaus Danne, Cristiano Pereira, Rolf Kassa, Tim Kranich, Shiliang Hu, Justin Gottschlich

**UIUC:** Nima Honarmand, Nathan Dautenhahn, Samuel T. King, Josep Torrellas
Record and Replay (RnR) Systems

• ‘Fear of debugging’ is one major block for adoption of shared memory parallel programming
  • Hard to diagnose and reproduce, source of much frustration

• Goal: Reproduce (replay) an execution so that each instruction inputs same values as the recorded one
  • Non-determinism: input data and thread interleaving

• Lots of research in the past 10 years but no complete hardware and software prototypes
  • What’s overhead and complexity? How much software enabling? Can it be always-on?
Contributions

1. HW/SW implementation of a user-level record and replay on an *Intel architecture* multi-core CPU
   a) Pentium cores augmented with a memory race recorder
   b) Full software stack: **recorder** and **replayer**

2. Recorder and replayer implementation description in the face of hardware intricacies
   a) Intel Relaxed memory model
   b) Intel non-atomic CISC instructions

3. Full characterization of the system
QuickRec Recording System

SW stack
- Application(s) to be recorded
- Modified Linux Kernel (Capo3)

User-level recording tool

HW stack
- Modified Pentium core
- Modified Pentium core
- Modified Pentium core
- Modified Pentium core

QuickIA Server

- Configures and manages hardware
- Records inputs to the program needed to replay
- Virtualizes recording hardware
- Records shared-memory interleaving with hardware extensions

- QuickIA is a dual socket Xeon platform where the CPUs are replaced by FPGAs
- Platform is capable of booting unmodified Linux OS
QuickIA Hardware Overview

Xeon Server

Socket 0
- FPGA
- Pentium L2$
- FPGA Bridge

Socket 1
- FPGA
- Pentium L2$
- FPGA Bridge

MCH
- DDR2

Connects Pentium FPGAs to Intel Front Side Bus

In-order superscalar Pentium processor (P54c) running @ 60 Mhz
Cache coherent, full access to memory and I/O
8 GB of DRAM @ 24MB/s

MCH-Memory Controller Hub
Memory Race Recording (MRR)

SW threads running on 2 cores

- MRR chunk denotes a timestamped group of instructions

Thread Interleaving Order

Data dependency resulting in a log entry

Data dependency not resulting in a log entry

Thread Interleaving

MICRO’09 – Architecting a Memory Race Recording in a Modern CPU
Cristiano Pereira – ISCA 2013, Jun 23 --27, Tel-Aviv, ISRAEL
Memory Order Recording Challenges

• Total Store Order Memory Model
  • Problem: Retired instruction count is insufficient to order instructions during replay
  • Solution: Track # of pending stores in store buffer (MICRO’11, CoreRacer)

• Instruction Atomicity Violation (IAV)
  • x86 macro memory operations do not execute atomically
    • Some operations require more than one load/store
  • IAV: intervening data dependency with another processor before completion of all memory operations
Instruction Atomicity Violation (IAV)

\[
\text{inc (@A)} = \begin{cases} 
\mu op_1: \text{tmp} = (@A) \\
\mu op_2: \text{tmp} = \text{tmp} + 1 \\
\mu op_3: (@A) = \text{tmp} 
\end{cases}
\]

**Solution:**

1. Track retirement of memory micro ops with an IAV counter
2. Reset when macro op retires
3. Log counter value if non-zero

See paper for details and replay algo.

Only first memory op. belongs to the chunk
Recording Software Stack (Capo3)

Inputs: syscalls, signals, etc.
Execution of syscalls, signals, etc.
Chunk data from processor
Serialized chunk log
Serialized Input log

Chips are isolated with few callbacks throughout the kernel

See paper for details
Platform Characterization

- Set of SPLASH benchmarks executed to completion
- Experimented with 1, 2, 4 and 8 threads
- Capo3 implemented in Linux 3.0.8
- Goals were to understand:
  - Performance overhead sources
  - Bandwidth requirements
  - Scalability
  - Chunk behavior

See paper for complete characterization
Recording overhead is entirely due to the SW stack (mostly: input logs)
Chunk Length (4p run)

CDF of chunk length

Bandwidth requirements

- Measured: 80KB/s on average
- Estimated in a modern Xeon server: 17.9 MB/s (1TB in 3 days w/ compression)
- Small fraction of available bandwidth

Average chunk length for 4p is 39K instructions. Small bandwidth requirements.
Percentage of IAV Chunks

Chunks with IAV are not just a corner case

Tight loop with several multi-memory operations
Lessons Learned

• RnR hardware can be built with **few touch points**
  – No changes in cache structures or coherence protocols
  – Easy to implement and validate

• Biggest challenge is to deal with micro-architecture idiosyncrasies that affect replay
  – TSO and IAV

• Software stack causes most recording overhead
  – We will speed up software stack → always-on usage model
Thank you!
Back up
Memory Ordering Idiosyncrasies

- IA memory model allows load instructions to retire before prior stores to different addresses

Retirement order:
1st: Store @C
2nd: Load @A

Stores drain from store buffer post-retirement

Assume ‘chunk creation’ happens before ‘Store @C’ is drained from buffer

Chunk order is potentially incorrect because retirement order does not match memory order

Solution Insight: add # of pending stores to a chunk