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?

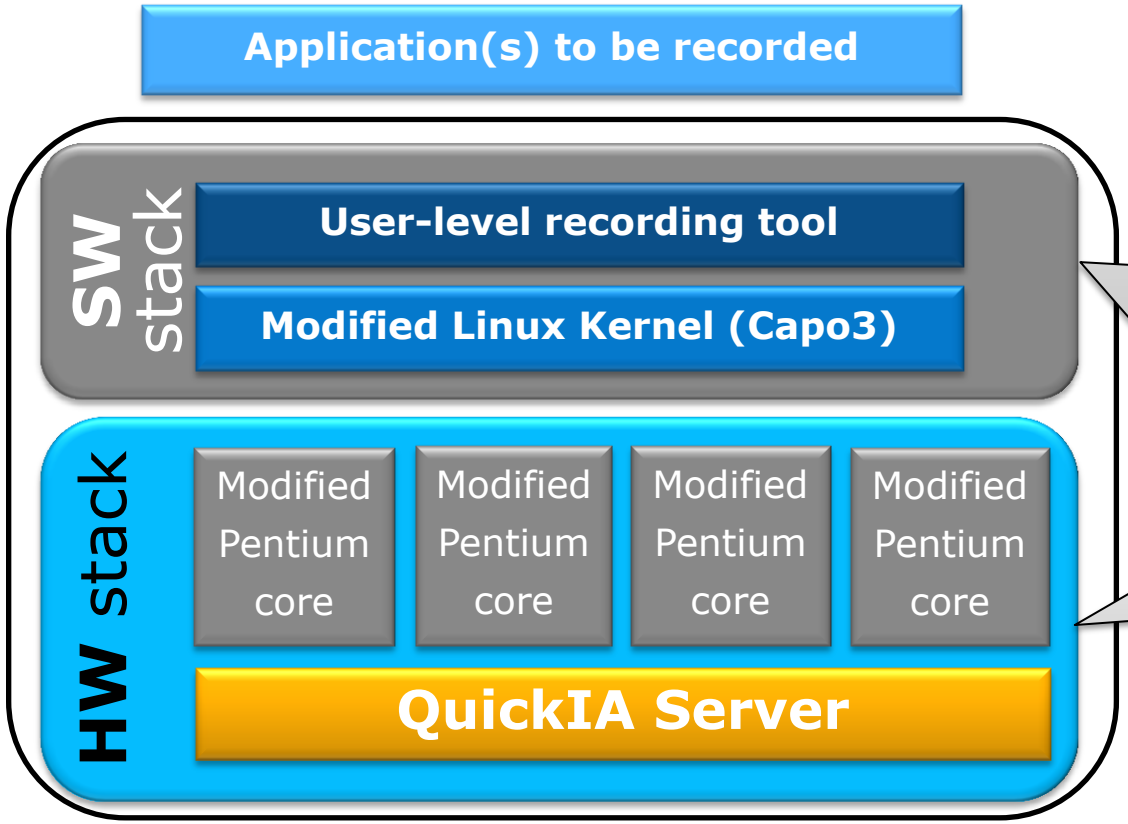
Contributions

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

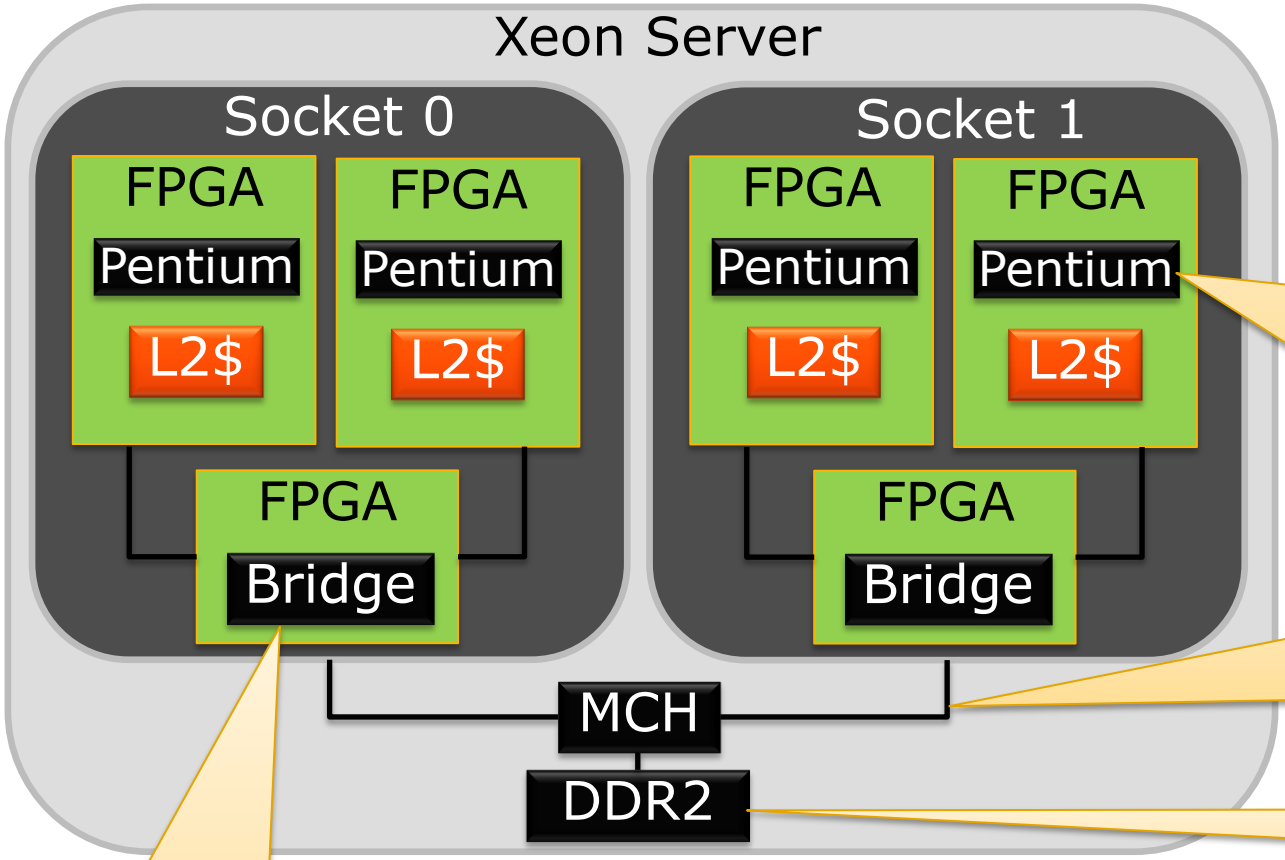


- Configures and manages hardware
- Records inputs to the program needed to replay
- Virtualizes recording hardware

- 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



In-order superscalar Pentium processor (P54c) running @ 60 Mhz

Cache coherent, full access to memory and I/O

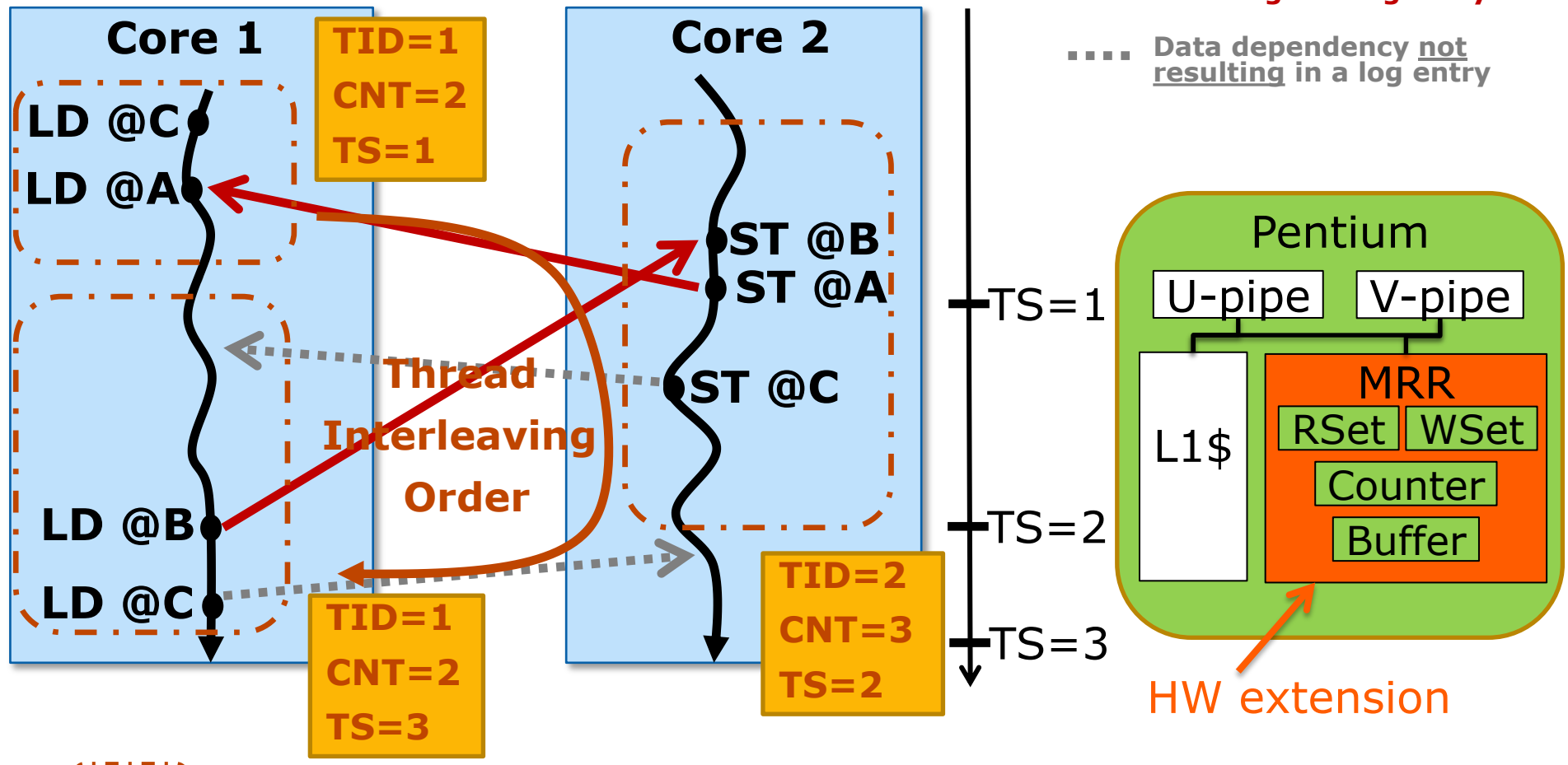
8 GB of DRAM @ 24MB/s

Connects Pentium FPGAs to Intel Front Side Bus

MCH-Memory Controller Hub

Memory Race Recording (MRR)

SW threads running on 2 cores



- MRR chunk denotes a timestamped group of instructions

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)

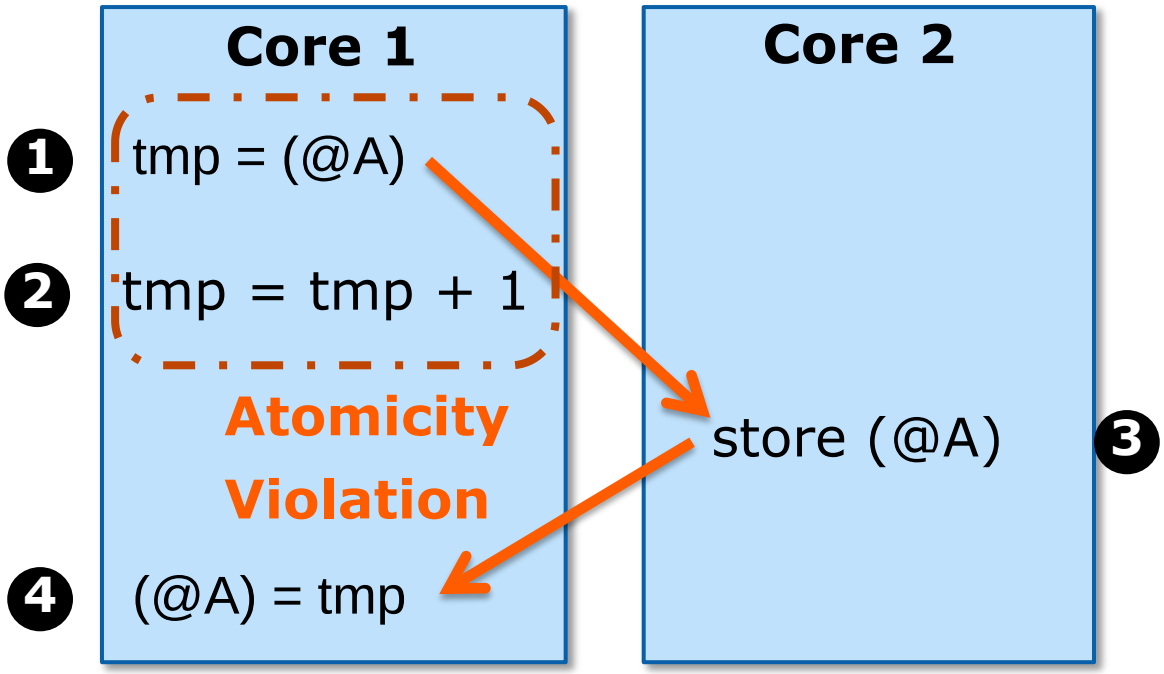
- Instruction Atomicity Violation (IAV)
  - x86 macro memory operations do not execute atomically
    - Some operations require more than one load/store
  - IAV: intervening data dependency with another processor before completion of all memory operations

Microarchitecture implementation details complicate replaying the memory ordering

Instruction Atomicity Violation (IAV)

inc (@A)

μop_1: tmp = (@A)
μop_2: tmp = tmp + 1
μop_3: (@A) = tmp



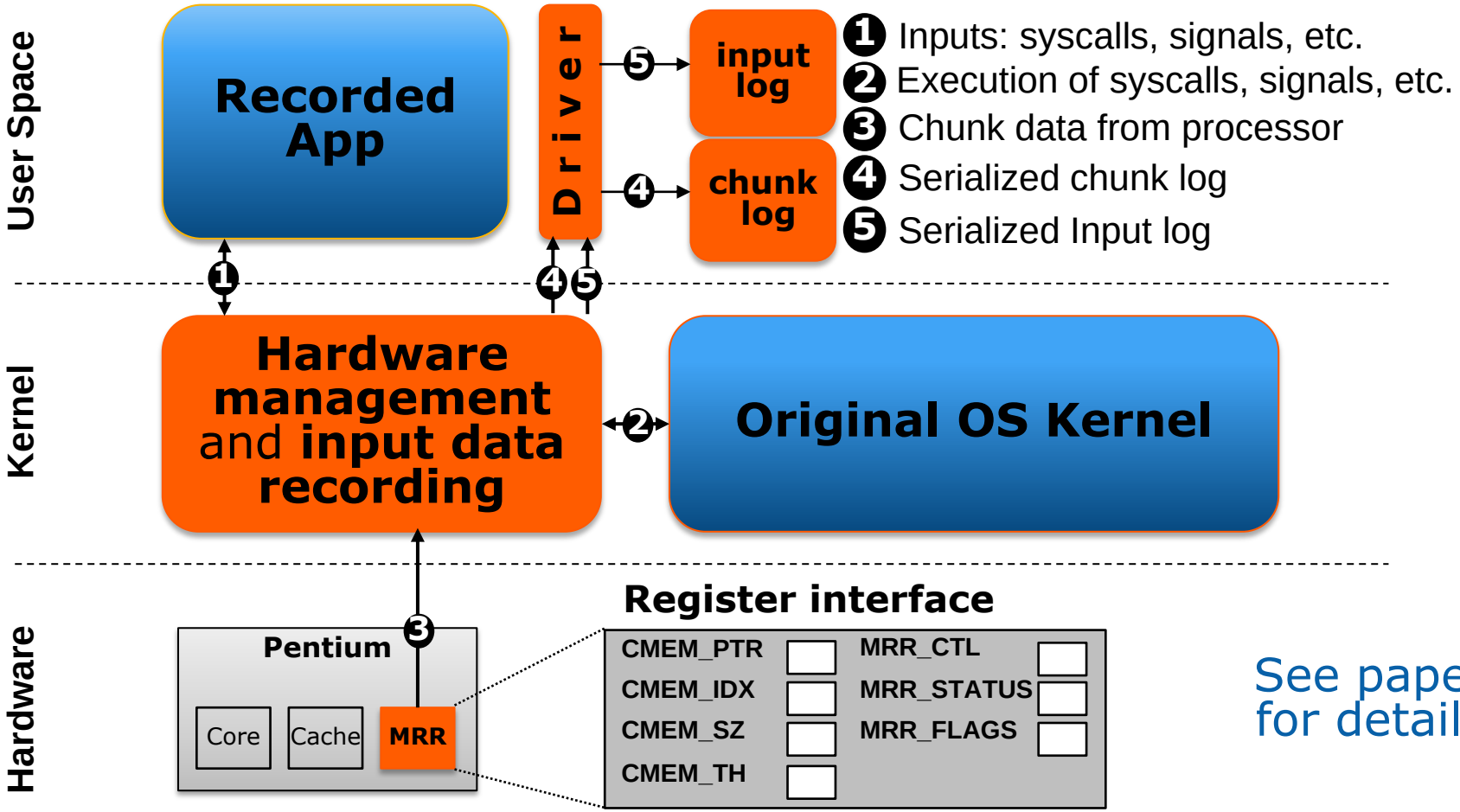
Only first memory op. belongs to the chunk

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.

Recording Software Stack (Capo3)



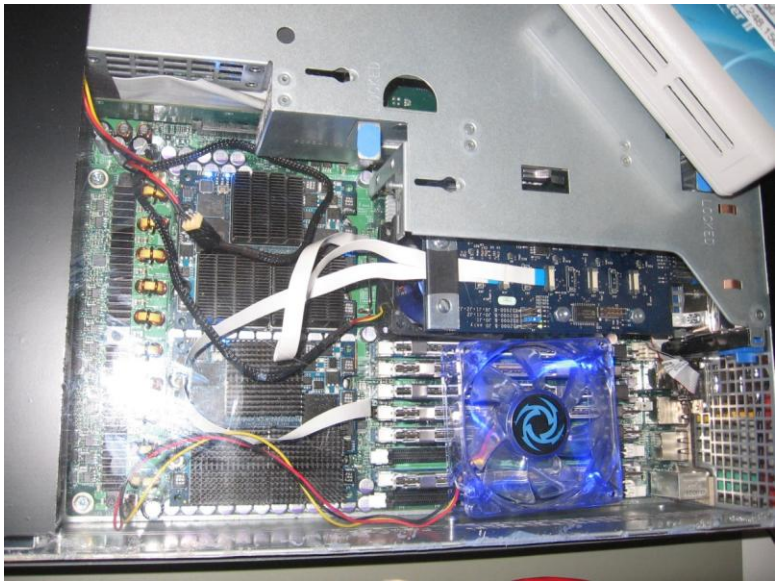
See paper for details

Changes are isolated with few callbacks throughout the kernel

Platform Characterization

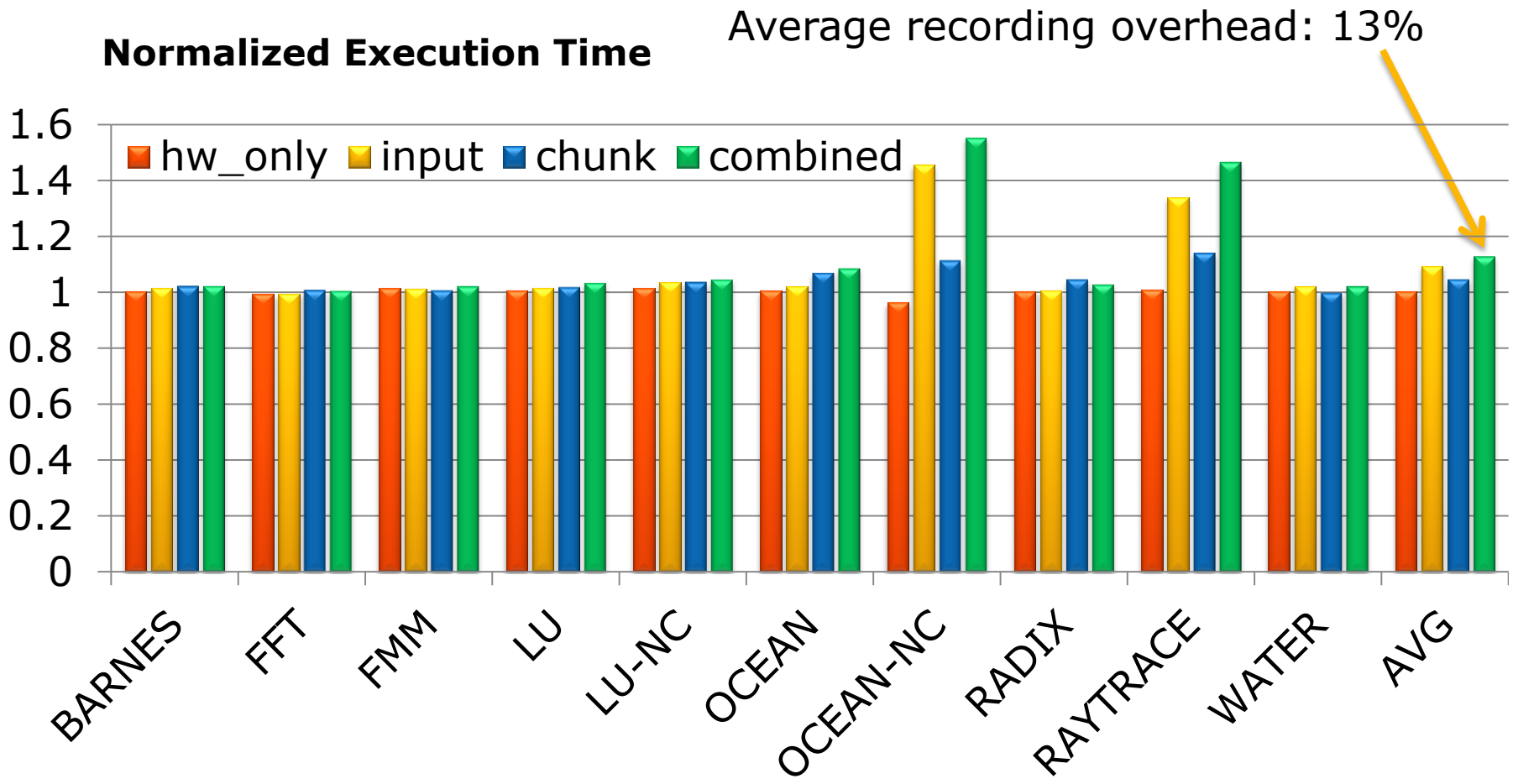
- Set of SPLASH benchmarks executed to completion
- Experimented with 1, 2, 4 and 8 threads
- Capo3 implemented in Linux 3.0.8
- Goals were to understand:
  - Performance overhead sources
  - Bandwidth requirements
  - Scalability
  - Chunk behavior

Photograph of the QuickRec platform



See paper for complete characterization

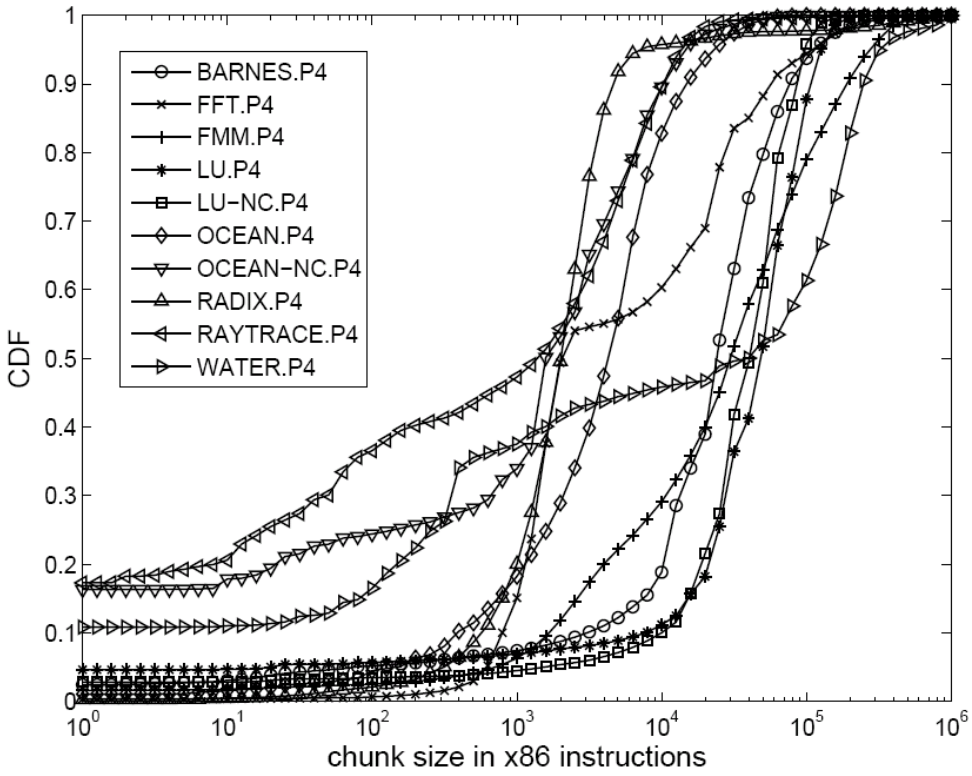
Performance Overhead (4p run)



Recording overhead is entirely due to the SW stack (mostly: input logs)

Chunk Length (4p run)

CDF of chunk length

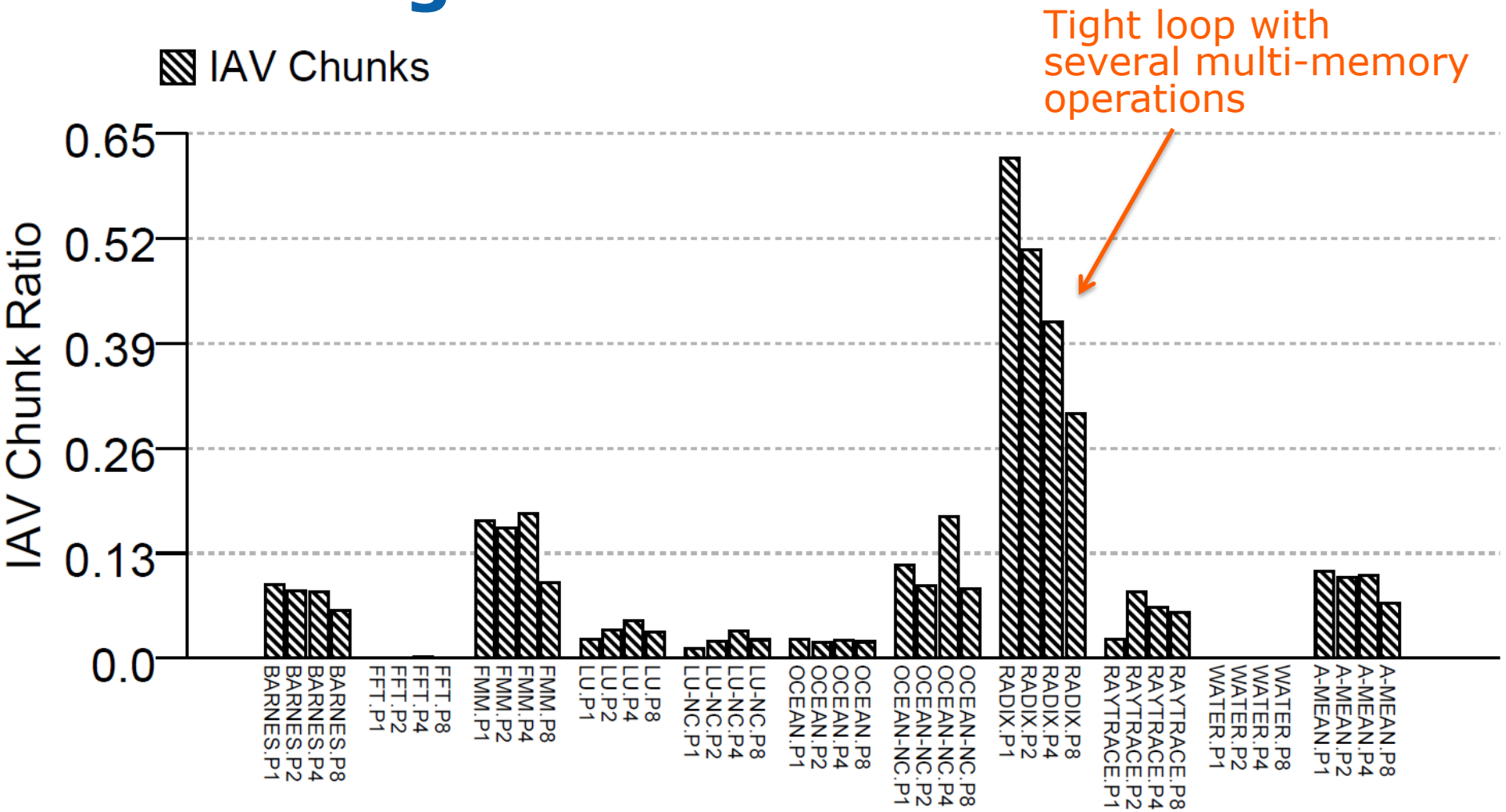


Bandwidth requirements

- 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

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

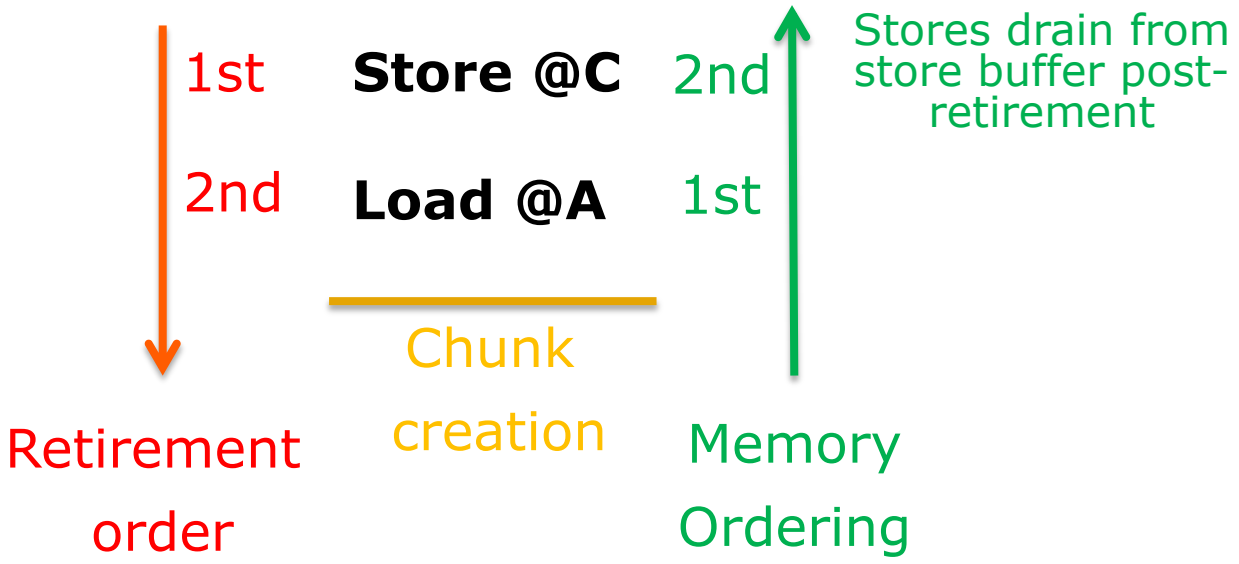
Q&A

Thank you!

Back up

Memory Ordering Idiosyncrasies

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

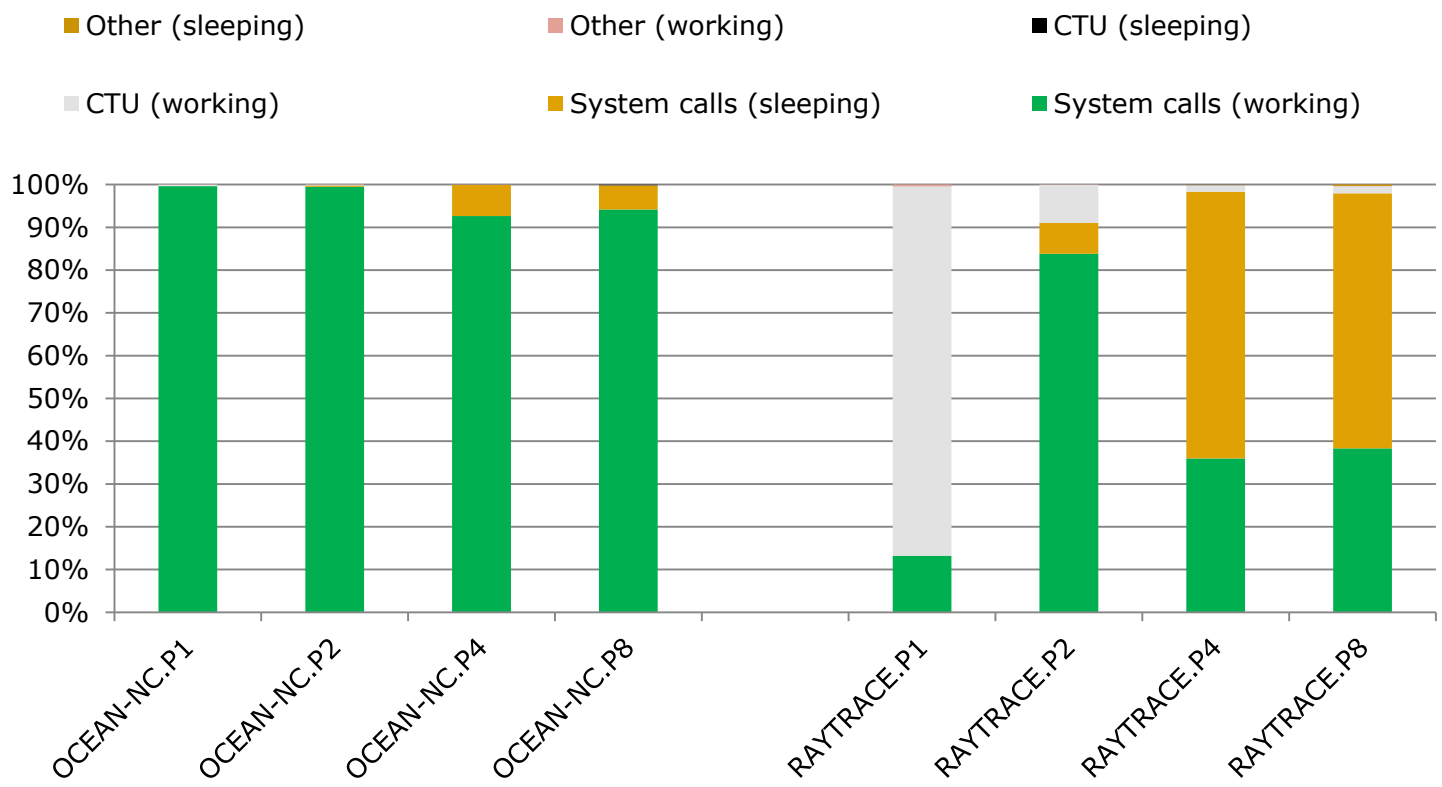


Assume 'chunk creation' happens before 'Store @C' is drained from buffer

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

Solution Insight: *add # of pending stores to a chunk*

Performance Overhead Breakdown




intel®
Leap ahead™