Lessons Learned During the Development
of the CapoOne Deterministic
Multiprocessor Replay System

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

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

- Allows us to visit and recreate past states and events in computer
Motivation: Time Travel

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

- Wide range of uses:
Motivation: Time Travel

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

- Wide range of uses:
  - Debugging
Motivation: Time Travel

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

- Wide range of uses:
  - Debugging
  - Security
Motivation: Time Travel

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

- Wide range of uses:
  - Debugging
  - Security
  - High-availability
Motivation: Time Travel

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

- Wide range of uses:
  - Debugging
  - Security
  - High-availability

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

- Phase I: Initial Execution (a.k.a Recording)
How Deterministic Replay Works

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

- **Phase II: Replay**
How Deterministic Replay Works

- **Phase I: Initial Execution (a.k.a Recording)**
  - Execute and record certain non-deterministic events into log

- **Phase II: Replay**
How Deterministic Replay Works

- **Phase I: Initial Execution** (a.k.a. Recording)
  - Execute and record certain non-deterministic events into log
  - Sources of non-determinism: interrupts, memory access interleaving ...

- **Phase II: Replay**
How Deterministic Replay Works

- **Phase I: Initial Execution (a.k.a Recording)**
  - Execute and record certain non-deterministic events into log
  - Sources of non-determinism: interrupts, memory access interleaving ...

- **Phase II: Replay**
  - Restore to a previous checkpoint
How Deterministic Replay Works

- **Phase I: Initial Execution (a.k.a Recording)**
  - Execute and record certain non-deterministic events into log
  - Sources of non-determinism: interrupts, memory access interleaving ...

- **Phase II: Replay**
  - Restore to a previous checkpoint
  - Re-execute and use log to force software down the same execution path
SW-Based vs HW-Based Deterministic Replay
SW-Based vs HW-Based Deterministic Replay

SW Based Schemes
SW-Based vs HW-Based Deterministic Replay

**SW Based Schemes**

Flexible, integrate well with OS, apps
SW-Based vs HW-Based Deterministic Replay

**SW Based Schemes**

- Flexible, integrate well with OS, apps
- Very slow on multiprocessors
SW-Based vs HW-Based Deterministic Replay

**SW Based Schemes**
- Flexible, integrate well with OS, apps
- Very slow on multiprocessors

**HW Based Schemes**
SW-Based vs HW-Based Deterministic Replay

SW Based Schemes

- Flexible, integrate well with OS, apps
- Very slow on multiprocessors

VS

HW Based Schemes

- Fast multiprocessor execution
SW-Based vs HW-Based Deterministic Replay

**SW Based Schemes**
- Flexible, integrate well with OS, apps
- Very slow on multiprocessors

**HW Based Schemes**
- Poor integration with SW
- Fast multiprocessor execution
HW-Assisted Deterministic Replay

**HW-Assisted Schemes**

- Flexible, integrate well with OS, apps
- Fast multiprocessor execution
HW-Assisted Deterministic Replay

CapoOne

+ Flexible, integrate well with OS, apps
+ Fast multiprocessor execution
Today’s Agenda
Today’s Agenda

- Overview: Capo and CapoOne
Today’s Agenda

- Overview: Capo and CapoOne
- From full-system replay to sphere-based replay
Today’s Agenda

- Overview: Capo and CapoOne
- From full-system replay to sphere-based replay
-Exiting the replay sphere
Today’s Agenda

- Overview: Capo and CapoOne
- From full-system replay to sphere-based replay
- Exiting the replay sphere
- System Issues
Capo

(ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
Capo

(ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications
Capo (ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications
  - Narrow: compatible with any HW-Based replay system
Capo

(ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications
  - Narrow: compatible with any HW-Based replay system

- **Replay Sphere:** new abstraction
Capo

(ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications
  - Narrow: compatible with any HW-Based replay system

- **Replay Sphere**: new abstraction
  - Isolates SW that is being recorded (replayed) from the rest
Capo (ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications
  - Narrow: compatible 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
Capo (ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications
  - Narrow: compatible 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**: first implementation of Capo
Replay Sphere
Replay Sphere
Replay Sphere

FFT Thread 103
FFT Thread 128
OS
Replay HW
CPU 1
CPU 2
CPU 3
CPU 4
Replay Sphere

FFT Thread 103
FFT Thread 128
Vi Thread 39

OS
Replay HW
CPU 1
CPU 2
CPU 3
CPU 4
Replay Sphere

FFT
Thread 103
FFT
Thread 128
Vi
Thread 39
GCC
Thread 26

OS

Replay HW

CPU 1
CPU 2
CPU 3
CPU 4
Set of threads recorded and replayed as a unit and their address space
Replay Sphere

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

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

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

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

- **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
CapoOne: First Capo Implementation
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors

![Diagram showing CapoOne实施的例子](image)
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors

- Ubuntu Linux with modified kernel:

![Diagram showing threads and processes running on Ubuntu Linux]
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors

- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic

Ubuntu Linux
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic
- Split Replay Sphere Manager
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic
- Split Replay Sphere Manager
- Simulated HW-Based replay system:
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic
- Split Replay Sphere Manager
- Simulated HW-Based replay system:
  - DeLorean [Montesinos et al. ISCA’08]
DeLorean: Chunk-Based Record/Replay
DeLorean: Chunk-Based Record/Replay

\[ \begin{align*}
\text{P0} & : \text{st } X \\
& : \ldots \\
& : \text{st } Y \\
\text{P1} & : \text{ld } T \\
& : \ldots \\
& : \text{st } W
\end{align*} \]
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically
DeLorean: Chunk-Based Record/Replay

Add chunks atomically

HW groups consecutive dynamic instructions

Execute chunks atomically
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation
**DeLorean: Chunk-Based Record/Replay**

- HW groups consecutive dynamic instructions
- Execute chunks atomically
- Execute chunks in isolation
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

Pablo Montesinos

Lessons Learned during the CapoOne Development
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

Pablo Montesinos
Lessons Learned during the CapoOne Development
DeLorean: Chunk-Based Record/Replay

- HW groups consecutive dynamic instructions
- Execute chunks atomically
- Execute chunks in isolation

Commit on X
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

P0

st X
... st Y

chunk

P1

ld T
... st W

chunk

P0

st X
... st Y

commit

P1

ld T
... st W

commit

P0

st X
... st Y

P1

ld X
... ld X

collision on X
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

*chunk*
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

Record chunk commit order

Pablo Montesinos

Lessons Learned during the CapoOne Development
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

Record chunk commit order

Pablo Montesinos

Lessons Learned during the CapoOne Development
DeLorean: Chunk-Based Record/Replay

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

Record chunk commit order

Pablo Montesinos
Lessons Learned during the CapoOne Development
DeLorean: Chunk-Based Record/Replay

HW records chunk commit order in Processor Interleaving Log

HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

Record chunk commit order
DeLorean: Chunk-Based Record/Replay

- HW groups consecutive dynamic instructions
- Execute chunks atomically
- Execute chunks in isolation
- Record chunk commit order

- HW records chunk commit order in Processor Interleaving Log
- Also records size of some irregular chunks in Chunk Size Log
DeLorean: Chunk-Based Record/Replay

- HW records chunk commit order in Processor Interleaving Log
- Also records size of some irregular chunks in Chunk Size Log
- During replay: generate same chunks and commit them in same order
From Full-System Replay to Sphere-Based Replay
From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere
From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere
- Log entries record R-thread IDs, not processor IDs
From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere
  - Log entries record R-thread IDs, not processor IDs
  - Information about other non-deterministic events is discarded
From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere
  - Log entries record R-thread IDs, not processor IDs
  - Information about other non-deterministic events is discarded

- Memory Interleaving Log must only include interleaving of instructions from R-threads within same sphere
From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere
  - Log entries record R-thread IDs, not processor IDs
  - Information about other non-deterministic events is discarded

- Memory Interleaving Log must only include interleaving of instructions from R-threads within same sphere
  - Changed DeLorean’s chunk truncation rules to enforce isolation
From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere
  - Log entries record R-thread IDs, not processor IDs
  - Information about other non-deterministic events is discarded

- Memory Interleaving Log must only include interleaving of instructions from R-threads within same sphere
  - Changed DeLorean’s chunk truncation rules to enforce isolation
  - Example: System calls cause chunk truncation in CapoOne, not in DeLorean
New Chunking Rules Provide R-Thread Isolation
New Chunking Rules Provide R-Thread Isolation

DeLorean

Processor P1

Processor Interleaving
Log
New Chunking Rules Provide R-Thread Isolation

DeLorean

Processor P1

\[ \ldots \]

500
501

\textit{inst m syscall}

Processor Interleaving Log
New Chunking Rules Provide R-Thread Isolation

DeLorean

Processor P1

500
inst m
501
syscall

Processor Interleaving Log
New Chunking Rules Provide R-Thread Isolation

DeLorean

Processor P1

500
501

... inst m
syscall

OS
Syscall
Handler

Processor Interleaving
Log
New Chunking Rules Provide R-Thread Isolation

DeLorean

Processor P1

Processor Interleaving Log

OS Syscall Handler

inst m syscall

500 501

999
New Chunking Rules Provide R-Thread Isolation

DeLorean

Processor P1

\[ \ldots \text{inst m syscall} \]

LOG

Syscall Handler

Processor Interleaving
Log

P1
New Chunking Rules Provide R-Thread Isolation

DeLorean
Processor P1

... inst m syscall
Log
Syscall Handler

CapoOne
R-Thread R1

Processor Interleaving Log
P1

R-thread Interleaving Log
New Chunking Rules Provide R-Thread Isolation

DeLorean

Processor P1

... inst m syscall

Syscall Handler

Log

Processor Interleaving Log

P1

CapoOne

R-Thread R1

... inst m syscall

R-thread Interleaving Log

500 501

999
New Chunking Rules Provide R-Thread Isolation

DeLorean
Processor P1

inst syscall

Syscall Handler

500
501
999

Processor Interleaving Log
P1

CapoOne
R-Thread R1

inst syscall

500
501

R-thread Interleaving Log
New Chunking Rules Provide R-Thread Isolation

DeLorean
Processor P1

CapoOne
R-Thread R1

Processor Interleaving Log
P1

R-thread Interleaving Log
R1

inst m syscall
Syscall Handler

inst m syscall
New Chunking Rules Provide R-Thread Isolation

DeLorean

Processor P1

500
501
999

... inst m

syscall

Log

Syscall Handler

Processor Interleaving

Log

P1

CapoOne

R-Thread R1

500
501

... inst m

syscall

Log

OS

Syscall Handler

R-thread Interleaving

Log

R1
New Chunking Rules Provide R-Thread Isolation

- Inform HW to stop logging interleaving of instructions from current proc
- Always do it when execution leaves replay sphere
- Otherwise, OS instructions might become part of interleaving log
**New Chunking Rules Provide R-Thread Isolation**

- Inform HW to stop logging interleaving of instructions from current proc
- Always do it when execution leaves replay sphere
- Otherwise, OS instructions might become part of interleaving log
New Chunking Rules Provide R-Thread Isolation

RSM and HW must work together to avoid any OS instruction to pollute the Memory Interleaving Log

- Inform HW to stop logging interleaving of instructions from current proc
- Always do it when execution leaves replay sphere
- Otherwise, OS instructions might become part of interleaving log
Handling Interrupts
Handling Interrupts

- Ensure interrupt handler code is not part of Memory Interleaving Log
Handling Interrupts

- Ensure interrupt handler code is not part of Memory Interleaving Log
- Balance conflicting demands because of chunk-based execution:
Handling Interrupts

- Ensure interrupt handler code is not part of Memory Interleaving Log
- Balance conflicting demands because of chunk-based execution:

Size of Memory Interleaving Log
Handling Interrupts

- Ensure interrupt handler code is not part of Memory Interleaving Log
- Balance conflicting demands because of chunk-based execution:

```
Size of Memory Interleaving Log
```

```
Interrupt Latency
```

Pablo Montesinos

Lessons Learned during the CapoOne Development
Handling Interrupts

- Ensure interrupt handler code is not part of Memory Interleaving Log
- Balance conflicting demands because of chunk-based execution:

![Diagram](Size of Memory Interleaving Log, Interrupt Latency, Wasted Work)
Handling Interrupts: Three Approaches
Handling Interrupts: Three Approaches

Finish First
### Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
</table>

**Finish First**
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish First</td>
<td></td>
<td></td>
<td></td>
<td>R-thread R1</td>
<td></td>
</tr>
</tbody>
</table>

18
# Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>R-thread R1</td>
<td></td>
</tr>
</tbody>
</table>

**Finish First**
# Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>R-thread R1</td>
<td></td>
</tr>
</tbody>
</table>

**Finish First**

[Diagram of R-thread R1 and inst m]
# Handling Interrupts: Three Approaches

## Finish First

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>R-thread R1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>inst m</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>999</td>
<td>. . .</td>
<td></td>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>

---

Pablo Montesinos

Lessons Learned during the CapoOne Development
### Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
</table>

#### Finish First

- **R-thread R1**
  - Initial state: 0
  - Final state: 999
  - Instruction: `inst m`
## Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
</table>

### Finish First

![Diagram showing handling interrupts]

- **R-thread R1**
- **OS Fault Handler**
- **inst m**
- **999**
- **0**
# Handling Interrupts: Three Approaches

## Finish First

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
</table>

- **R-thread R1**
  - 0
  - `inst m`
  - 999

- **OS Fault Handler**
  - `
    ...
    ...
  `
# Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish First</td>
<td>✔️</td>
<td></td>
<td>R-thread R1</td>
<td></td>
</tr>
</tbody>
</table>
Handling Interrupts: Three Approaches

Finish First

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
</table>

Original Execution

R-thread R1

0

... inst m ...

999

OS Fault Handler

...
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish First</td>
<td>✔️</td>
<td>✔️</td>
<td>🟥</td>
</tr>
</tbody>
</table>

Original Execution

- R-thread R1
  - inst m
  - OS Fault Handler
  - . . .
  - 0
  - 999

Chunk Size Log
## Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✔️</td>
<td>✔️</td>
<td>-</td>
</tr>
</tbody>
</table>

### Commit Now
### Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✔️ ✔️</td>
<td>✔️</td>
<td></td>
<td>R-thread R1</td>
<td></td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish First</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Commit Now</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Original Execution
R-thread R1

Chunk Size Log

inst m
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Original Execution</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R-thread R1</td>
<td>0</td>
<td>500</td>
<td></td>
</tr>
</tbody>
</table>

**Chunk Size Log**
# Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish First</td>
<td>![Check Mark]</td>
<td>![Check Mark]</td>
<td>![Red Bar]</td>
<td>R-thread R1</td>
<td>7</td>
</tr>
</tbody>
</table>

## Commit Now

```
0
500
```

... inst m
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
</tbody>
</table>

**Commit Now**

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td>7</td>
</tr>
<tr>
<td>ChunkID</td>
<td>500</td>
</tr>
</tbody>
</table>

```
0
inst m
500
```
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Finish First</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Size</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Wasted Work</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt Latency</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Commit Now

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td>7</td>
</tr>
</tbody>
</table>

Chunk Size

0

500

inst m
## Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish First</td>
<td>![Checkmark] ![Checkmark]</td>
<td>![Red Bar]</td>
</tr>
</tbody>
</table>

### Commit Now

- Original Execution
- Chunk Size Log
  - R-thread R1
  - 7 | 500

![Diagram](image-url)
## Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Original Execution**

<table>
<thead>
<tr>
<th>R-thread R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\cdot\cdot\cdot]</td>
</tr>
<tr>
<td>Inst m</td>
</tr>
<tr>
<td>OS Int Handler</td>
</tr>
</tbody>
</table>

**Chunk Size Log**

| 7 | 500 |
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish First</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Commit Now

Original Execution

R-thread R1

Chunk Size Log

| 7 | 500 |
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish First</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Commit Now

---

Original Execution

R-thread R1

<table>
<thead>
<tr>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
</tbody>
</table>
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Original Execution**

- R-thread R1

<table>
<thead>
<tr>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

Diagram:
- **Finish First**:
  - Log Size: ✓
  - Wasted Work: ✓
  - Interrupt Latency: -

- **Commit Now**:
  - Log Size: -
  - Wasted Work: -
  - Interrupt Latency: -

Diagram labels:
- **inst m**: 500
- **OS Int Handler**: 999
## Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td></td>
<td>✔️</td>
<td></td>
</tr>
</tbody>
</table>

### Original Execution

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>500</td>
</tr>
</tbody>
</table>

![Diagram showing the execution process with Log and Chunk Size Log details.]
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Finish First</th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Commit Now</th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Original Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>500</td>
</tr>
<tr>
<td>inst m</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>OS Int Handler</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>999</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>500</td>
</tr>
</tbody>
</table>
## Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Squash Now</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish First</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>R-thread R1</td>
<td></td>
</tr>
<tr>
<td>Commit Now</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Squash Now</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th>Method</th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish First</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td>Commit Now</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td>Squash Now</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram](image-url)

- **Original Execution**
- **Chunk Size Log**
- **R-thread R1**

---

**Pablo Montesinos**
Lessons Learned during the CapoOne Development
## Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finish First</td>
<td>✔️</td>
<td>✔️</td>
<td>-</td>
</tr>
<tr>
<td>Commit Now</td>
<td>-</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Squash Now</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Diagram:

- **Original Execution**
  - Chunk Size Log
  - R-thread R1
  - Inst m

- **Chunk Size Log**
  - 0
  - 500
  - Inst m

---

Pablo Montesinos

Lessons Learned during the CapoOne Development
# Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>![check]</td>
<td>![check]</td>
<td>![x]</td>
<td>![lightning bolt]</td>
<td>![lightning bolt]</td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td>![x]</td>
<td>![check]</td>
<td>![check]</td>
<td>![lightning bolt]</td>
<td>![lightning bolt]</td>
</tr>
<tr>
<td><strong>Squash Now</strong></td>
<td>![x]</td>
<td>![check]</td>
<td>![check]</td>
<td>![lightning bolt]</td>
<td>![lightning bolt]</td>
</tr>
</tbody>
</table>
# Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>R-thread R1</td>
<td></td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Squash Now</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Handling Interrupts: Three Approaches

- **Finish First**
  - Log Size: √
  - Wasted Work: √
  - Interrupt Latency: -

- **Commit Now**
  - Log Size: -
  - Wasted Work: √
  - Interrupt Latency: √

- **Squash Now**

### Original Execution

- R-thread R1
- OS Int Handler

### Chunk Size Log

- inst m
- 0
- 999
## Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Squash Now</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td></td>
</tr>
<tr>
<td>OS Int Handler</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>inst m</td>
</tr>
<tr>
<td></td>
<td>999</td>
</tr>
</tbody>
</table>
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Squash Now</strong></td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Original Execution

Chunk Size Log

- R-thread R1
  - OS Int Handler
  - 0
  - inst m
  - 999

---

Pablo Montesinos
Lessons Learned during the CapoOne Development
## Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Squash Now</strong></td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Original Execution

- R-thread R1
- OS Int Handler

### Chunk Size Log

- 0
- 999

---

Pablo Montesinos

Lessons Learned during the CapoOne Development
Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Squash Now</strong></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

**Original Execution**
- R-thread R1
- OS Int Handler
- 0
- 999

**Chunk Size Log**
- `inst m`
## Handling Interrupts: Three Approaches

<table>
<thead>
<tr>
<th></th>
<th>Log Size</th>
<th>Wasted Work</th>
<th>Interrupt Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finish First</strong></td>
<td>✔️</td>
<td>✔️</td>
<td>▢</td>
</tr>
<tr>
<td><strong>Commit Now</strong></td>
<td></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Squash Now</strong></td>
<td>✔️</td>
<td></td>
<td>✔️</td>
</tr>
</tbody>
</table>

### Original Execution

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>OS Int Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>500</td>
<td></td>
</tr>
<tr>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>

### Chunk Size Log

<table>
<thead>
<tr>
<th>inst m</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>999</td>
</tr>
</tbody>
</table>
Handling Interrupts: Three Approaches

Highly non-deterministic events, such as interrupts, can be treated as deterministic events

CapoOne uses SquashNow: easy to implement and little overhead
Handling Faults
Handling Faults

- Commit all instructions up to faulting one and log chunk size
## Handling Faults

Commit all instructions up to faulting one and log chunk size

<table>
<thead>
<tr>
<th></th>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td>7</td>
<td>500</td>
</tr>
</tbody>
</table>
Handling Faults

- Commit all instructions up to faulting one and log chunk size
Handling Faults

- Commit all instructions up to faulting one and log chunk size
Handling Faults

Commit all instructions up to faulting one and log chunk size

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td>7</td>
<td>500</td>
</tr>
<tr>
<td>500</td>
<td>501</td>
<td>inst m</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inst n</td>
</tr>
</tbody>
</table>
Handling Faults

- Commit all instructions up to faulting one and log chunk size

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td>7</td>
</tr>
</tbody>
</table>

OS Fault Handler

<table>
<thead>
<tr>
<th>inst m</th>
<th>inst n</th>
<th>500</th>
<th>501</th>
</tr>
</thead>
</table>
Handling Faults

- Commit all instructions up to faulting one and log chunk size

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td>7</td>
</tr>
<tr>
<td>500 501</td>
<td>500</td>
</tr>
<tr>
<td>inst m inst n</td>
<td></td>
</tr>
<tr>
<td>OS Fault Handler</td>
<td>0</td>
</tr>
<tr>
<td>inst n</td>
<td></td>
</tr>
<tr>
<td>inst z</td>
<td>999</td>
</tr>
</tbody>
</table>

Pablo Montesinos

Lessons Learned during the CapoOne Development
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

![Diagram showing chunk size and log entries for different cases of replay.](image)
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

Original Execution

Chunk Size Log

Replay Case I

Replay Case II

Replay Case III

R-thread R1

500

... inst m

501

inst n

500

OS Fault Handler

0

... inst n

999

... inst z

... inst m
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

![Diagram showing original execution and replay cases](image)
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

Original Execution

<table>
<thead>
<tr>
<th>R-thread R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
</tr>
<tr>
<td>inst n</td>
</tr>
<tr>
<td>501</td>
</tr>
<tr>
<td>inst m</td>
</tr>
<tr>
<td>OS Fault Handler</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>inst n</td>
</tr>
<tr>
<td>999</td>
</tr>
<tr>
<td>inst z</td>
</tr>
</tbody>
</table>

Chunk Size Log

| 7 | 500 |

Replay Case I

<table>
<thead>
<tr>
<th>R-thread R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
</tr>
<tr>
<td>inst m</td>
</tr>
<tr>
<td>501</td>
</tr>
<tr>
<td>inst n</td>
</tr>
</tbody>
</table>

Replay Case II

<table>
<thead>
<tr>
<th>R-thread R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
</tr>
<tr>
<td>inst m</td>
</tr>
<tr>
<td>501</td>
</tr>
<tr>
<td>inst n</td>
</tr>
</tbody>
</table>

Replay Case III

<table>
<thead>
<tr>
<th>R-thread R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
</tr>
<tr>
<td>inst m</td>
</tr>
<tr>
<td>501</td>
</tr>
<tr>
<td>inst n</td>
</tr>
</tbody>
</table>
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>Chunk Size Log</th>
<th>Replay Case I</th>
<th>Replay Case II</th>
<th>Replay Case III</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(7)</td>
<td>500</td>
<td>(500)</td>
<td>(500)</td>
</tr>
<tr>
<td>inst m</td>
<td>inst n</td>
<td>OS</td>
<td>Fault Handler</td>
<td>inst n</td>
</tr>
<tr>
<td>500, 501</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inst n</td>
<td>inst z</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inst z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>999</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

Original Execution

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>500</th>
<th>501</th>
<th>0</th>
<th>999</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Fault Handler</td>
<td>inst n</td>
<td>inst n</td>
<td>inst z</td>
<td></td>
</tr>
</tbody>
</table>

Chunk Size Log

| | 7 | 500 |

Replay Case I

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>500</th>
<th>0</th>
<th>999</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Fault Handler</td>
<td>inst m</td>
<td>inst n</td>
<td>inst z</td>
</tr>
</tbody>
</table>

Replay Case II

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>500</th>
<th>0</th>
<th>999</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Fault Handler</td>
<td>inst m</td>
<td>inst n</td>
<td>inst z</td>
</tr>
</tbody>
</table>

Replay Case III

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>500</th>
<th>0</th>
<th>999</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Fault Handler</td>
<td>inst m</td>
<td>inst n</td>
<td>inst z</td>
</tr>
</tbody>
</table>
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

```
Original Execution
R-thread R1

500
501

OS Fault Handler

inst n

500
501

Chunk Size
Log
7
500

Replay
Case I
R-thread R1

500
501

OS Fault Handler

inst n

500
501

Replay
Case II
R-thread R1

500
501

OS Fault Handler

inst n

500
501

Replay
Case III
R-thread R1

500
501

OS Fault Handler

inst n

500
501
```

Pablo Montesinos

Lessons Learned during the CapoOne Development
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

![Diagram showing handling of faults during replay]

Original Execution

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>OS Fault Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst n</td>
<td>500</td>
</tr>
<tr>
<td>inst m</td>
<td>501</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chunk Size Log

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>500</td>
</tr>
</tbody>
</table>

Replay Case I

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>OS Fault Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst n</td>
<td>500</td>
</tr>
<tr>
<td>inst m</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Replay Case II

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>OS Fault Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst n</td>
<td>500</td>
</tr>
<tr>
<td>inst z</td>
<td>999</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Replay Case III

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>inst n</td>
<td>999</td>
</tr>
<tr>
<td>inst m</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>inst z</td>
<td>999</td>
</tr>
</tbody>
</table>

Chunk Size Log

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>inst m</td>
<td>500</td>
</tr>
<tr>
<td>inst n</td>
<td>999</td>
</tr>
</tbody>
</table>

Log Chunk Size

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>500</td>
</tr>
</tbody>
</table>

Replay Case I

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>OS Fault Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst n</td>
<td>500</td>
</tr>
<tr>
<td>inst m</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Replay Case II

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>OS Fault Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst n</td>
<td>500</td>
</tr>
<tr>
<td>inst z</td>
<td>999</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Replay Case III

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>inst n</td>
<td>999</td>
</tr>
<tr>
<td>inst m</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>inst z</td>
<td>999</td>
</tr>
</tbody>
</table>
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

Original Execution

<table>
<thead>
<tr>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
<tr>
<td>500</td>
</tr>
</tbody>
</table>

R-thread R1

<table>
<thead>
<tr>
<th>inst m</th>
<th>inst n</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>501</td>
</tr>
</tbody>
</table>

OS Fault Handler

<table>
<thead>
<tr>
<th>inst n</th>
<th>inst z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>999</td>
</tr>
</tbody>
</table>

Replay Case I

<table>
<thead>
<tr>
<th>R-thread R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst m</td>
</tr>
<tr>
<td>500</td>
</tr>
</tbody>
</table>

OS Fault Handler

<table>
<thead>
<tr>
<th>inst n</th>
<th>inst z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>999</td>
</tr>
</tbody>
</table>

Replay Case II

<table>
<thead>
<tr>
<th>R-thread R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst m</td>
</tr>
<tr>
<td>500</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>inst n</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>inst z</td>
</tr>
<tr>
<td>999</td>
</tr>
</tbody>
</table>

Replay Case III

<table>
<thead>
<tr>
<th>R-thread R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst m</td>
</tr>
<tr>
<td>500</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>inst n</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>inst z</td>
</tr>
<tr>
<td>999</td>
</tr>
</tbody>
</table>
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size Log</th>
<th>Replay Case I</th>
<th>Replay Case II</th>
<th>Replay Case III</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td>7 500</td>
<td>R-thread R1</td>
<td>R-thread R1</td>
<td>R-thread R1</td>
</tr>
<tr>
<td>OS Fault Handler</td>
<td></td>
<td>OS Fault</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inst n</td>
<td>inst n</td>
<td>inst n</td>
<td>inst n</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>501</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>500</td>
<td>501</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>inst z</td>
<td></td>
<td>inst z</td>
<td>inst z</td>
<td></td>
</tr>
<tr>
<td>999</td>
<td></td>
<td>999</td>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size Log</th>
<th>Replay Case I</th>
<th>Replay Case II</th>
<th>Replay Case III</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td>7[500]</td>
<td>R-thread R1</td>
<td>R-thread R1</td>
<td>R-thread R1</td>
</tr>
<tr>
<td>OS Fault Handler</td>
<td>inst n</td>
<td>OS Fault</td>
<td></td>
<td>inst z</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Handler</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>inst n</td>
<td></td>
<td>inst n</td>
<td>inst z</td>
</tr>
<tr>
<td></td>
<td>inst z</td>
<td></td>
<td>inst z</td>
<td></td>
</tr>
</tbody>
</table>

Pablo Montesinos
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

![Diagram showing handling of faults and replay cases]

Original Execution

<table>
<thead>
<tr>
<th>R-thread R1</th>
<th>Chunk Size Log</th>
<th>Replay Case I</th>
<th>Replay Case II</th>
<th>Replay Case III</th>
</tr>
</thead>
<tbody>
<tr>
<td>500 501</td>
<td></td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>999</td>
<td></td>
</tr>
<tr>
<td>999</td>
<td></td>
<td>999</td>
<td>inst z</td>
<td></td>
</tr>
<tr>
<td><strong>OS Fault Handler</strong></td>
<td></td>
<td>inst n</td>
<td>OS Fault Handler</td>
<td></td>
</tr>
<tr>
<td><strong>inst m</strong></td>
<td></td>
<td><strong>inst n</strong></td>
<td><strong>inst m</strong></td>
<td><strong>inst m</strong></td>
</tr>
<tr>
<td><strong>inst n</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>inst n</strong></td>
</tr>
<tr>
<td><strong>inst z</strong></td>
<td></td>
<td></td>
<td></td>
<td>inst q inst r</td>
</tr>
<tr>
<td><strong>inst r</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pablo Montesinos

Lessons Learned during the CapoOne Development
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

Original Execution

Replay Case I

Replay Case II

Replay Case III

Pablo Montesinos

Lessons Learned during the CapoOne Development
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size Log</th>
<th>Replay Case I</th>
<th>Replay Case II</th>
<th>Replay Case III</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td></td>
<td>R-thread R1</td>
<td>R-thread R1</td>
<td>R-thread R1</td>
</tr>
<tr>
<td></td>
<td>7 500</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS Fault Handler</td>
<td>inst n</td>
<td>inst m</td>
<td>inst m</td>
<td>inst m</td>
</tr>
<tr>
<td></td>
<td>inst z</td>
<td></td>
<td>inst z</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inst n</td>
<td></td>
<td>inst n</td>
<td>inst q</td>
</tr>
<tr>
<td></td>
<td>500 501</td>
<td></td>
<td>500 500</td>
<td>500 500</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
<td>799 800</td>
</tr>
<tr>
<td></td>
<td>999</td>
<td></td>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>

Pablo Montesinos | Lessons Learned during the CapoOne Development
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size Log</th>
<th>Replay Case I</th>
<th>Replay Case II</th>
<th>Replay Case III</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td>7 500</td>
<td>R-thread R1</td>
<td>R-thread R1</td>
<td>R-thread R1</td>
</tr>
<tr>
<td>. . . inst m</td>
<td>. . . inst m</td>
<td>. . . inst m</td>
<td>. . . inst m</td>
<td>. . . inst m</td>
</tr>
<tr>
<td>inst n</td>
<td>inst n</td>
<td>inst n</td>
<td>inst n</td>
<td>inst n</td>
</tr>
<tr>
<td>500 501</td>
<td>500 500</td>
<td>500 500</td>
<td>500 500</td>
<td>500 500</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OS Fault Handler</td>
<td>OS Fault Handler</td>
<td>OS Fault Handler</td>
<td>OS Fault Handler</td>
<td>OS Fault Handler</td>
</tr>
<tr>
<td>inst n</td>
<td>inst n</td>
<td>inst n</td>
<td>inst z</td>
<td>inst r</td>
</tr>
<tr>
<td>999</td>
<td>999</td>
<td>999</td>
<td>999</td>
<td>999</td>
</tr>
</tbody>
</table>
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

**Replay Case I**

- OS Fault Handler: 500
- Inst n: 0
- Inst m: 500

**Replay Case II**

- OS Fault Handler: 500
- Inst n: 0
- Inst m: 500
- Inst q: 799

**Replay Case III**

- OS Fault Handler: 500
- Inst n: 0
- Inst m: 500
- Inst r: 800
- Inst z: 199

---

Pablo Montesinos

Lessons Learned during the CapoOne Development

---
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

---

**Original Execution**

- R-thread R1
- Chunk Log: 7 500
- OS Fault Handler
- inst n
- inst n
- 0
- 500
- 501
- 0
- 500
- 0
- 0
- 0
- 0
- 0
- 0
- 0
- 999
- 999
- 999
- 999

---

**Replay Case I**

- R-thread R1
- OS Fault Handler
- inst n
- inst n
- 0
- 500
- 500
- 0
- 999
- 999
- 999
- 999

---

**Replay Case II**

- R-thread R1
- OS Fault Handler
- inst n
- inst z
- 0
- 500
- 500
- 999
- 999
- 999
- 999

---

**Replay Case III**

- R-thread R1
- OS Fault Handler
- inst n
- inst r
- inst q
- inst z
- 0
- 500
- 799
- 800
- 199
- 0
- 0
- 0
- 0

No other R-thread can commit
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size Log</th>
<th>Replay Case I</th>
<th>Replay Case II</th>
<th>Replay Case III</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td></td>
<td>R-thread R1</td>
<td>R-thread R1</td>
<td>R-thread R1</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS Fault Handler</td>
<td>inst m</td>
<td>inst m</td>
<td>inst m</td>
<td>inst m</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>501</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>999</td>
<td>999</td>
<td>199</td>
</tr>
</tbody>
</table>

No other R-thread can commit
Handling Faults

- Commit all instructions up to faulting one and log chunk size.
- Handling faults during replay can be tricky.

Faults are synchronous events, but they are not necessarily deterministic.

A fault can occur during replay but not during recording, or vice versa.
Handling Traps and Programmed Exceptions
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk

Traps and programmed exceptions are deterministic events: no need to record irregular-sized chunks
Injecting Data into Spheres

Replay Sphere 1 - Recording

Log 1

Replay Sphere Manager

OS
Injecting Data into Spheres

Replay Sphere 1 - Recording

Log 1

Replay Sphere Manager

OS
Injecting Data into Spheres

Replay Sphere 1 - Recording

R-thread R1

B
Y
E
\0
buf

R-thread R2

Log 1

Replay Sphere Manager

OS
Injecting Data into Spheres

Replay Sphere 1 - Recording

R-thread R1

read(&buf)

B
Y
E
\0
buf

R-thread R2

Log 1

Replay Sphere Manager

OS

Lessons Learned during the CapoOne Development
Injecting Data into Spheres

Replay Sphere 1 - Recording

Log 1

Replay Sphere Manager

OS

R-thread R2

buffer

BYE
"0
Injecting Data into Spheres

[Diagram showing Replay Sphere 1 - Recording with labels: Replay Sphere Manager, OS, Log 1, copy_to_user, R-thread R2, B, Y, E, \0, buf]
Injecting Data into Spheres

Replay Sphere 1 - Recording

OS

R-thread R2

Log 1

Replay Sphere Manager

read(&buf)

copy_to_user

buf
Injecting Data into Spheres

Replay Sphere 1 - Recording

Log 1

Replay Sphere Manager

OS

copy_to_user

R-thread R2

buf

\0

BYE

B

Replay Sphere 1 - Recording

Log 1
Injecting Data into Spheres

Replay Sphere 1 - Recording

Replay Sphere Manager

Log 1

copy_to_user

buf

R-thread R1

R-thread R2

OS
Injecting Data into Spheres

Replay Sphere 1 - Recording

Log 1

Replay Sphere Manager

copy_to_user

buf

R-thread R2

H
I
E
\0

OS
Injecting Data into Spheres

- Problem: interleaving between OS copies and R-threads not recorded
Injecting Data into Spheres

Problem: interleaving between OS copies and R-threads not recorded
Injecting Data into Spheres

Problem: interleaving between OS copies and R-threads not recorded
Injecting Data into Spheres

- Problem: interleaving between OS copies and R-threads not recorded
- Solution: insert `copy_to_user` into sphere:
Injecting 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

Replay Sphere 1 - Recording

- R-thread R2
  - X = buf[2]
  - buf[3] = Y

OS

Replay Sphere Manager

Log 1
Injecting 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
Injecting 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
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

<table>
<thead>
<tr>
<th>R-thread A</th>
<th>R-thread B</th>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_enter</td>
<td>read_exit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pablo Montesinos
Lessons Learned during the CapoOne Development
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

```
R-thread A | R-thread B
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>read( )</td>
<td></td>
</tr>
</tbody>
</table>
```

```
R-thread Interleaving Log | Input Log
|--------------------------|----------|
```
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

![Diagram of R-thread A and R-thread B with read( ) function and logs]

R-thread A

R-thread B

R-thread Interleaving Log

Input Log

A

read( )
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

<table>
<thead>
<tr>
<th>R-thread A</th>
<th>R-thread B</th>
</tr>
</thead>
</table>
| ```
read( )
```
| ```
read( )
```

<table>
<thead>
<tr>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A read_enter</td>
</tr>
</tbody>
</table>
Injecting Data into Spheres (II)

Look for interactions between R-thread Interleaving Log and Input Log

<table>
<thead>
<tr>
<th>R-thread A</th>
<th>R-thread B</th>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>read( )</td>
<td></td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>copy_to_user</td>
<td></td>
<td>read_enter</td>
<td></td>
</tr>
</tbody>
</table>
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

```
<table>
<thead>
<tr>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>
```

R-thread A
- `read()`
- `copy_to_user`

R-thread B
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

```
R-thread A           | R-thread B
---------------------|---------------------
read( ) RSM          |                       
copy_to _user        |                       

R-thread Interleaving Log

A
A
B

Input Log

A read_enter
```
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

```
<table>
<thead>
<tr>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>read_enter</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>
```

R-thread A

- `read( )`
- `RSM`
- `copy_to_user`

R-thread B
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

![Diagram showing interactions between R-thread A and R-thread B with RSM and copy to _user operations. The table shows the sequence of operations in both logs, highlighting the point where data is injected before returning.]
Injecting Data into Spheres (II)

Look for interactions between R-thread Interleaving Log and Input Log

- R-thread A
  - read()
  - copy_to_user
  - RSM

- R-thread B
  - RSM

<table>
<thead>
<tr>
<th>R-thread</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interleaving Log</td>
<td>Log</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

<table>
<thead>
<tr>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A read_enter</td>
</tr>
<tr>
<td>A</td>
<td>A read_exit inject before returning</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

R-thread A
- read()
- RSM
- copy_to_user
- RSM

R-thread B
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

```
R-thread A
read()
  
copy_to_user
  
RSM

RSM

R-thread B

<table>
<thead>
<tr>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A read.enter</td>
</tr>
<tr>
<td>A</td>
<td>A read.exit</td>
</tr>
<tr>
<td>B</td>
<td>inject before returning</td>
</tr>
</tbody>
</table>
```

R-thread A
R-thread B
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

<table>
<thead>
<tr>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A read_enter</td>
</tr>
<tr>
<td>A</td>
<td>A read_exit inject before returning</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

R-thread A
- read()
- copy_to_user
- RSM

R-thread B
- read()
Injecting Data into Spheres (II)

Look for interactions between R-thread Interleaving Log and Input Log
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

Diagram:
- R-thread A
  - read()
  - copy_to_user
  - RSM
- R-thread B
  - RSM
- R-thread A Interleaving Log
  - A
  - A
  - B
- Input Log
  - A read_enter
  - A read_exit
  - inject before returning
- R-thread B
  - read()
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

R-thread A
- read(
- copy_to_user
  - RSM

R-thread B
- read(
  - RSM

R-thread Interleaving Log
- A
- A
- B

Input Log
- A read_enter
- A read_exit
- inject before returning

R-thread A
- read(
  - RSM

R-thread B
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

```
R-thread A  R-thread B

R-thread Interleaving Log

A
A
B

Input Log

A read_enter
A read_exit  inject before returning

R-thread A  R-thread B

read( )
copy_to_user
RSM
```
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

```
R-thread A
  read()  
  RSM
  copy_to_user
  RSM

R-thread B
  RSM

R-thread Interleaving Log
  A
  A
  B

Input Log
  A  read_enter
  A  read_exit  inject before returning
  A
```

Pablo Montesinos

Lessons Learned during the CapoOne Development
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

```plaintext
R-thread A
  read()
  RSM
  copy_to_user

R-thread B

R-thread Interleaving Log
  A
  B

Input Log
  A  read_enter
  A  read_exit  inject before returning

R-thread A
  read()
  RSM
  copy_to_user

R-thread B
```

Pablo Montesinos
Lessons Learned during the CapoOne Development
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log
Injecting Data into Spheres (II)

Look for interactions between R-thread Interleaving Log and Input Log

- R-thread A
  - read( )
  - copy_to_user
  - RSM

- R-thread B
  - write( )
  - RSM

<table>
<thead>
<tr>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A read_enter</td>
</tr>
<tr>
<td>A</td>
<td>B write_enter</td>
</tr>
<tr>
<td>B</td>
<td>A read_exit</td>
</tr>
</tbody>
</table>

infect before returning
Injecting Data into Spheres (II)

Look for interactions between R-thread Interleaving Log and Input Log

```
R-thread A
  read()
  copy_to_user
  RSM

R-thread B
  write()
  RSM

RSM

Input Log

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>read_enter</td>
<td>write_enter</td>
</tr>
<tr>
<td>B</td>
<td>read_exit</td>
<td>inject before returning</td>
</tr>
</tbody>
</table>
```
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

<table>
<thead>
<tr>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A read_enter</td>
</tr>
<tr>
<td>A</td>
<td>B write_enter</td>
</tr>
<tr>
<td>B</td>
<td>A read_exit</td>
</tr>
</tbody>
</table>

```
R-thread A
- read()
- copy_to_user
- write()

R-thread B
- read_enter
- write_enter
- read_exit
```
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

<table>
<thead>
<tr>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A read_enter</td>
</tr>
<tr>
<td>A</td>
<td>B write_enter</td>
</tr>
<tr>
<td>B</td>
<td>A read_exit</td>
</tr>
</tbody>
</table>
Injecting Data into Spheres (II)

Look for interactions between R-thread Interleaving Log and Input Log

<table>
<thead>
<tr>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A read_enter</td>
</tr>
<tr>
<td>A</td>
<td>B write_enter</td>
</tr>
<tr>
<td>B</td>
<td>A read_exit</td>
</tr>
</tbody>
</table>
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log
Injecting Data into Spheres (II)

Look for interactions between R-thread Interleaving Log and Input Log

<table>
<thead>
<tr>
<th></th>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>read_enter</td>
<td>A read_exit</td>
</tr>
<tr>
<td>A</td>
<td>write_enter</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>read_exit</td>
<td></td>
</tr>
</tbody>
</table>

R-thread A

R-thread B

RSM

read( )
copy_to_user
write( )
RSM

R-thread A

read( )
RSM

R-thread B
Looking for interactions between R-thread Interleaving Log and Input Log

<table>
<thead>
<tr>
<th>R-thread Interleaving Log</th>
<th>Input Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A  read_enter</td>
</tr>
<tr>
<td>A</td>
<td>B  write_enter</td>
</tr>
<tr>
<td>B</td>
<td>A  read_exit  inject before returning</td>
</tr>
</tbody>
</table>

**Diagram:**
- **R-thread A**
  - `read()`
  - `copy_to_user`
- **R-thread B**
  - `write()`
  - `write()`
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log
Injecting Data into Spheres (II)

Look for interactions between R-thread Interleaving Log and Input Log

R-thread A

1. read()
2. copy_to_user
3. write()

R-thread B

1. read()
2. write()

RSM

R-thread Interleaving Log

- A
- A
- B

Input Log

- A read_enter
- B write_enter
- A read_exit

injected before returning
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

Circular dependences between Memory Interleaving Log and Input Log cause deadlocks
Cache Overflows

- CapoOne keeps speculative data in cache until commit
  - A chunk may access more lines mapping to a cache set than ways the set has
  - BulkSC-based system do not allow storing speculative state in memory

- When a cache would overflow, CapoOne must commit current chunk
  - Non-deterministic event: must log chunk size
Cache Overflows
## Cache Overflows

<table>
<thead>
<tr>
<th>Original Execution</th>
<th>Chunk Size Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-thread R1</td>
<td></td>
</tr>
</tbody>
</table>
Cache Overflows

Original Execution

Chunk Size Log

R-thread R1

...
Cache Overflows

ROB

<table>
<thead>
<tr>
<th>Head</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst m</td>
</tr>
<tr>
<td>inst n</td>
</tr>
<tr>
<td>inst t</td>
</tr>
</tbody>
</table>

Original Execution

R-thread R1

Chunk Size Log

...
Cache Overflows

ROB

Original Execution
R-thread R1

Chunk Size Log

Head
inst t
inst n
inst m
Cache Overflows

ROB

Original Execution
R-thread R1

Chunk Size Log

inst m

Head

inst n

inst m

inst m
Cache Overflows

ROB

Original Execution

Chunk Size Log

R-thread R1

inst t

inst m

inst n

Head
Cache Overflows

- The instruction causing the overflow might not be at the head of ROB
Cache Overflows

- The instruction causing the overflow might not be at the head of ROB.
Cache Overflows

The instruction causing the overflow might not be at the head of ROB
Cache Overflows

- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay
The instruction causing the overflow might not be at the head of ROB

Can happen during initial execution and/or during replay
Cache Overflows

- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay
The instruction causing the overflow might not be at the head of ROB

Can happen during initial execution and/or during replay
Cache Overflows

- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay
**Cache Overflows**

- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay
Cache Overflows

- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay
Cache Overflows

- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay
Cache Overflows

The instruction causing the cache overflow might not be at the head of ROB

HW must ensure that chunk boundaries are consistent

Can happen during initial execution and/or during replay
Lessons Learned During the Development of the CapoOne Deterministic Multiprocessor Replay System

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

Department of Computer Science
University of Illinois at Urbana-Champaign
Overall DeLorean System
Overall DeLorean System

Proc 0
- Chunk A
  - CS Log
  - Interrupt Log

Proc 1
- Chunk B
  - CS Log
  - Interrupt Log

Arbiter
- PI Log
Overall DeLorean System
Overall DeLorean System

Proc 0

- Chunk A
- CS Log
- Interrupt Log
- I/O Log

Network

Proc 1

- Chunk B
- CS Log
- Interrupt Log
- I/O Log

Arbiter

PI Log
Overall DeLorean System
Interrupt, I/O and DMA logs are common to other HW-based schemes
CapoOne: HW Implementation
CapoOne: HW Implementation

Proc 0
- Chunk A
  - CS Log
  - Interrupt Log
  - I/O Log

Network
- DMA Log
- PI Log

Proc 1
- Chunk B
  - CS Log
  - Interrupt Log
  - I/O Log
No need for DeLorean’s Interrupt Log, DMA Log nor I/O Log
CapoOne: HW Implementation

- No need for DeLorean’s Interrupt Log, DMA Log nor I/O Log
- PI Log becomes the per-sphere Interleaving Log
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
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
Emulating vs Re-Executing System Calls

During replay, the RSM emulates most system calls:
Emulating vs Re-Executing System Calls

During replay, the RSM emulates most system calls:
Emulating vs Re-Executing System Calls

During replay, the RSM emulates most system calls:

- R-thread 1
- R-thread 2
- read()
Emulating vs Re-Executing System Calls

During replay, the RSM emulates most system calls:
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
- RSM injects return values from Sphere Input Log, squashes outputs
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
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
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
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
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
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
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

![Diagram showing the interaction between RSM, Log, and OS with replaying threads.](image-url)
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)
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)
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
Implicit Dependencies

- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space
Implicit Dependencies

- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space
Implicit Dependencies

- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space

```c
while(1){
    *x = *x+1
}
```
Implicit Dependencies

- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space

```
while(1){
  *x = *x+1
}
```
Implicit Dependencies

- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space.
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
Replay Sphere
Replay Sphere

OS

Replay HW

CPU 1
CPU 2
CPU 3
CPU 4
Replay Sphere
Replay Sphere
Replay Sphere
Replay Sphere

- Set of threads recorded and replayed as a unit and their address space
Replay Sphere

- Set of threads recorded and replayed as a unit and their address space
- Only user-mode threads run inside spheres
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 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
- R-threads that share memory run within same sphere
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
- R-threads that share memory run within the same sphere
Self-Modifying Code

- Handling self-modifying code in chunk-based systems is laborious:
  - HW adds instruction fetches to read set
  - Instruction fetches are checked against local write set

- In CapoOne, when a processor detects it has modified code

- However,
CapoOne: First Capo Implementation
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps

- Simulated HW-Based replay system:
  - DeLorean [Montesinos et al. ISCA’08]
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps

- Simulated HW-Based replay system:
  - DeLorean [Montesinos et al. ISCA’08]
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps
- Simulated HW-Based replay system:
  - DeLorean [Montesinos et al. ISCA’08]
- Ubuntu Linux with modified kernel:
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps
- Simulated HW-Based replay system:
  - DeLorean [Montesinos et al. ISCA’08]
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps

- Simulated HW-Based replay system:
  - DeLorean [Montesinos et al. ISCA’08]

- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps
- Simulated HW-Based replay system:
  - DeLorean [Montesinos et al. ISCA’08]
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic
- Split Replay Sphere Manager