Time travelling in multicore processors

Henry Liu and Ethan Zou
1. **Background on multicore/distributed systems**
2. **TARDIS Protocol**
3. **Optimizations and Evaluations**
   a. Delta Timestamps
   b. Various Lease Predictor Protocols
4. **Future Work and Acknowledgements**
- cores equivalent to processors
- faster performance → multiple cores
- data is shared by different cores, we need shared memory
Coherence

- If one processor modifies the data, how can other processors know the latest value?
- Having stale data and writing stale data results in error and **incoherence**
Outline

1. Background on multicore/distributed systems
2. TARDIS Protocol
3. Optimizations and Evaluations
   a. Delta Timestamps
   b. Various Lease Predictor Protocols
4. Future Work and Acknowledgements
Tardis

- Recently proposed protocol
- Very scalable and simple
- Uses timestamps to logically organize shared memory and ensure coherence
- Allows for “time traveling” of operations since they don’t have to be done in sequence of physical time
Library Example

Borrowing from 0 - 10

Borrowing from 0 - 20

Wants to edit, so jumps in time and edits at 21
Each cacheline in Tardis has a Read TimeStamp (RTS) and a Write TimeStamp (WTS)
- WTS - time of last store
- RTS - time of last read
- Private memory - data loaded at timestamp before rts
- Shared memory - rts is the longest private memory lease
- Cacheline Structure:

<table>
<thead>
<tr>
<th>WTS</th>
<th>RTS</th>
<th>Data</th>
</tr>
</thead>
</table>

TARDIS Example

Tasks:
Set A=2
Print B

Tasks:
Set B=3
Print A

Core 0
PTS = 0
Private Memory

Core 1
PTS = 0
Private Memory

WTS=0  RTS=0  A = 1
WTS=0  RTS=0  B = 0

Shared Memory
TARDIS Example

**Tasks:**
- Set A=2
- Print B

**Core 0**
- PTS = 0
- Private Memory

**Core 1**
- PTS = 0
- Private Memory

**Tasks:**
- Set B=3
- Print A

**Owner = Core 0**
- A
  - WTS=0
  - RTS=0
- B = 0

**Shared Memory**
TARDIS Example

Tasks:
- Set A=2
- Print B

Owner = Core 0

Core 0

WTS = 0
RTS = 0
PTS = 0
A = 2

Load A

Core 1

WTS = 0
RTS = 0
PTS = 0

Tasks:
- Set B=3
- Print A

Shared Memory

Owner = Core 0

A

WTS=0
RTS=0
B = 0
TARDIS Example

Tasks:
Set A=2
Print B

Tasks:
Set B=3
Print A
TARDIS Example

Tasks:
Set A=2
Print B

Tasks:
Set B=3
Print A

Owner = Core 0
A
WTS=0 RTS=11
B = 0

PTS = 1
Core 0
WTS1 RTS1 A = 2
WTS0 RTS11 B = 0
Private Memory

PTS = 0
Core 1
Private Memory
Tasks:
Set A=2
Print B

Tasks:
Set B=3
Print A
TARDIS Example

Tasks:
Set $A=2$
Print $B$

Core 0:
- $PTS = 1$
- Private Memory
- $A = 2$
- $WTS = 0$
- $RTS = 11$
- $B = 0$

Core 1:
- $PTS = 02$
- Private Memory
- $B = 3$
- $WTS = 12$
- $RTS = 12$
- $B = 0$

Tasks:
Set $B=3$
Print $A$

Owner = Core 0
A

Owner = Core 1
B

Shared Memory
TARDIS Example

Tasks:
Set A=2
Print B

Tasks:
Set B=3
Print A

Shared Memory

Core 0

WTS0 | RTS11 | B = 0

WTS1 | RTS1 | A = 2

PTS = 1

Private Memory

Core 1

WTS12 | RTS12 | B = 3

PTS = 12

Private Memory

Owner = Core 0

A

Owner = Core 1

B

Read Request

Share Request
TARDIS Example

Tasks:
- Set A=2
- Print B

Core 0
- PTS = 1
- WTS0 = 1, RTS11 = 2, B = 0
- Private Memory
- A = 2

Core 1
- PTS = 12
- WTS12 = 1, RTS12 = 2, B = 3
- Private Memory
- A = 2

Tasks:
- Set B=3
- Print A

Done
Outline

1. Background on multicore/distributed systems
2. TARDIS Protocol
3. Optimizations and Evaluations
   a. Delta Timestamps
   b. Various Lease Predictor Protocols
4. Future Work and Acknowledgements
Timestamp Compression

- Timestamp size should be small for space efficiency
- Data is 512 bits; timestamp originally 64 bits each (25% of data)
- Wts and rts are usually fairly close, so we use a base timestamp (bts) and a delta (difference) = rts-wts to represent rts and wts
- We then ran tests to determine the optimal bts
- Now 16 bits each (6.25% of data)
Timestamp Compression

![Timestamp Compression Graph](image.png)

- **Benchmarks**:
  - Radiosity
  - FMM
  - Barnes
  - Cholesky
  - Ray/Trace
  - Volfred
  - Ocean-C
  - Ocean-NC
  - FFT
  - Radix
  - LU-C
  - LU-NC
  - Water-NSQ
  - Water-SP
  - Average

- **Compression Settings**:
  - BTS
  - 14
  - 16
  - 64

- **Graph Details**:
  - Y-axis: Completion Time Difference
  - X-axis: Benchmark

---

20
Outline

1. Background on multicore/distributed systems
2. TARDIS Protocol
3. Optimizations and Evaluations
   a. Delta Timestamps
   b. Various Lease Predictor Protocols
4. Future Work and Acknowledgements
The Renewal Problem

- if we keep modifying data, timestamps will increase by the arbitrary value of 10
  - read-write intensive, want the lease to be something much less than 10
- read-only data, we keep renewing it, lease can be very large
- renew requests incur extra latency and network traffic
Minimizing Renewals

- an adaptively changing lease
  - lines that are written to frequently should have a small lease
  - lines written to less frequently/read-only should have longer lease

- two basic protocols
  - exponentially growing lease
  - linearly growing lease
Evaluations of Lease Protocols
Evaluations of Lease Protocols

Linear Lease Prediction

Normalized Throughput (Higher is better)

Benchmark
Outline

1. Background on multicore/distributed systems
2. TARDIS Protocol
3. Optimizations and Evaluations
   a. Delta Timestamps
   b. Various Lease Predictor Protocols
4. Future Work and Acknowledgements
Future Work

- better lease prediction algorithm
- Renew in batches
- Renew in the background
- Techniques to slow down timestamp increment
- Further timestamp compression
Acknowledgements

We would like to thank:

- our parents
- our mentor, Xiangyao Yu and Professor Srini Devadas
- the MIT PRIMES program