THE SEMANTICS OF TRANSACTIONS AND WEAK MEMORY IN X86, POWER, ARM, AND C++

Nathan Chong
Arm Ltd.

Tyler Sorensen
Imperial

John Wickerson
Imperial

UCL PPLV Seminar, Thursday 10 May 2018
OUTLINE
OUTLINE

- Weak memory
OUTLINE

• Weak memory

• Transactions
• Weak memory

• Transactions

• Weak memory and transactions
OUTLINE

• Weak memory
• Transactions
• Weak memory and transactions
• Validating our models
OUTLINE

• Weak memory
• Transactions
• Weak memory and transactions
• Validating our models
• The problem with lock elision
• Weak memory

• Transactions

• Weak memory and transactions

• Validating our models

• The problem with lock elision

• Related and future work
WEAK MEMORY
WEAK MEMORY

\[
\begin{array}{c|c}
\text{MOV} [x] 1 & \text{MOV} [y] 1 \\
\text{MOV} r0 [y] & \text{MOV} r1 [x]
\end{array}
\]
WEAK MEMORY

\[
\begin{align*}
\text{MOV } [x] & \ 1 \quad | \quad \text{MOV } [y] & \ 1 \\
\text{MOV } r0 & \ [y] \quad | \quad \text{MOV } r1 & \ [x]
\end{align*}
\]

\[
\begin{align*}
r0=1 \\
r1=1
\end{align*}
\]
WEAK MEMORY

\[
\begin{array}{c|c}
\text{MOV} & [x] & 1 & \quad \text{MOV} & [y] & 1 \\
\text{MOV} & r0 & [y] & \quad \text{MOV} & r1 & [x] \\
\hline
r0=1 & r0=0 \\
r1=1 & r1=1
\end{array}
\]
WEAK MEMORY

\[
\begin{align*}
\text{MOV } & [x] \ 1 & \quad \text{MOV } & [y] \ 1 \\
\text{MOV } & r0 \ [y] & \quad \text{MOV } & r1 \ [x] \\
r0=1 & \quad r0=0 & \quad r0=1 \\
r1=1 & \quad r1=1 & \quad r1=0
\end{align*}
\]
WEAK MEMORY

\[ \begin{align*}
\text{MOV} [x] & \ 1 & & \text{MOV} [y] & \ 1 \\
\text{MOV} \ r0 & \ [y] & & \text{MOV} \ r1 & \ [x]
\end{align*} \]

\[
\begin{align*}
\text{r0}=1 & & \text{r0}=0 & & \text{r0}=1 \\
\text{rl}=1 & & \text{rl}=1 & & \text{rl}=0
\end{align*}
\]

SC
WEAK MEMORY

\[
\begin{align*}
\text{MOV} & \ [y] 1 \quad \text{MOV} & \ [x] 1 \\
\text{MOV} & \ r0 \ [y] \quad \text{MOV} & \ r1 \ [x]
\end{align*}
\]

\[
\begin{align*}
r0=1 \quad r1=1 & \quad r0=1 \quad r1=1 \\
r0=0 \quad r1=1 & \quad r0=1 \quad r1=0 \\
r1=1 \quad r0=1 & \quad r1=0 \quad r1=0
\end{align*}
\]

SC
WEAK MEMORY

\[
\begin{align*}
\text{MOV} & \quad [x] \quad 1 \quad \text{MOV} & \quad [y] \quad 1 \\
\text{MOV} & \quad r0 \quad [y] \quad \text{MOV} & \quad r1 \quad [x]
\end{align*}
\]

r0=1    r0=0    r0=1    r0=0
r1=1    r1=1    r1=0    r1=0

SC

x86
WEAK MEMORY

\[
\begin{align*}
\text{MOV} & \ [x] 1 & \text{MOV} & \ [y] 1 \\
\text{MOV} & \ r0 \ [y] & \text{MOV} & \ r1 \ [x]
\end{align*}
\]

\[
\begin{array}{cccc}
r0=1 & r0=0 & r0=1 & r0=0 \\
rl=1 & rl=1 & rl=0 & rl=0 \\
\end{array}
\]

SC

x86
WEAK MEMORY

\[
\begin{align*}
\text{MOV } [x] &\ 1 & \text{MOV } [y] &\ 1 \\
\text{MOV } r0 [y] &\ & \text{MOV } r1 [x] &\
\end{align*}
\]

\[
\begin{align*}
r0=1 &\quad r0=0 &\quad r0=1 &\quad r0=0 \\
r1=1 &\quad r1=1 &\quad r1=0 &\quad r1=0 \\
\end{align*}
\]
WEAK MEMORY

ARM

| MOV [x] 1 | MOV [y] 1 |
| MOV r0 [y] | MOV r1 [x] |

r0=1  r0=0  r0=1  r0=0  r1=1  r1=1  r1=0  r1=0

IBM

x86

Intel
WEAK MEMORY

MOV [x] 1
MOV r0 [y]
MOV r1 [x]

MOV [y] 1
MOV r0 [y]
MOV r1 [x]

r0=1  r0=0  r0=1  r0=0
r1=1  r1=1  r1=0  r1=0

x86

ARM

NVIDIA

IBM
MOV [x] 1
MOV [y] 1
MOV r0 [y]  MOV r1 [x]

r0=1  r0=0  r0=1  r0=0
r1=1  r1=1  r1=0  r1=0
WEAK MEMORY

MOV [x] 1
MOV r0 [y] MOV

r0=1 r0=0
r1=1 r1=1

MOV
MOV

r0=1
r1=0

MOV
MOV

r0=0
r1=0

MOV
MOV

r1=0

ARM®

AMD

NVIDIA®

C++

THE PROGRAMMING LANGUAGE

IBM

OpenCL

intel®
Weak memory

- MOV [x] 1
- MOV [y] 1
- MOV r0 [y]
- MOV r1 [x]

- r0=0
- r1=1
- r0=1
- r1=0

- ARM
- AMD
- NVIDIA
- Java
- C++
- IBM
- OpenCL
- Intel
WEAK MEMORY IS HARD!

- x86 proved tricky to formalise correctly [Sarkar et al., POPL'09; Owens et al., TPHOLs'09]

- Bug found in deployed "Power 5" processors [Alglave et al., CAV'10]

- C++ specification did not guarantee its own key property [Batty et al., POPL'11]

- Routine compiler optimisations are invalid under Java and C++ memory models [Sevcik, PLDI'11; Vafeiadis et al. POPL'15]

- Behaviour of NVIDIA graphics processors contradicted NVIDIA's programming guide [Alglave et al., ASPLOS'15]
MODELLING WEAK MEMORY

\begin{align*}
\text{MOV} & \ [x] \ 1 \\
\text{MOV} & \ r0 \ [y] \\
\text{MOV} & \ [y] \ 1 \\
\text{MOV} & \ r1 \ [x]
\end{align*}
MODELLING WEAK MEMORY

\[
\begin{align*}
\text{MOV} & \ [x] \ 1 \quad || \quad \text{MOV} & \ [y] \ 1 \\
\text{MOV} & \ r0 \ [y] \quad || \quad \text{MOV} & \ r1 \ [x] \\
\end{align*}
\]

W 1  \quad W 1  \\
\text{R } 1 \quad \text{R } 1

r0=1  \quad r1=1
MODELLING WEAK MEMORY

MOV [x] 1  MOV [y] 1
MOV r0 [y]  MOV r1 [x]

W x 1  W y 1  W x 1  W y 1
r0=1  r1=1  r0=0  r1=1

R y 1  R x 1  R y 0  R x 1
po  po  po  po
rf  rf  fr  rf
MODELLING WEAK MEMORY

\[
\begin{align*}
\text{MOV} & \ [x] \ 1 \\
\text{MOV} & \ r0 \ [y] \\
\text{MOV} & \ r1 \ [x]
\end{align*}
\]
MODELLING WEAK MEMORY

\[
\begin{align*}
\text{MOV} \ [x] 1 & \quad \text{MOV} \ [y] 1 \\
\text{MOV} \ r0 \ [y] & \quad \text{MOV} \ r1 \ [x]
\end{align*}
\]
## MODELLING WEAK MEMORY

### MOVING WEAK MEMORY

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV [x] 1</td>
<td></td>
</tr>
<tr>
<td>MOV r0 [y]</td>
<td></td>
</tr>
<tr>
<td>MOV r1 [x]</td>
<td></td>
</tr>
<tr>
<td>MOV [y] 1</td>
<td></td>
</tr>
</tbody>
</table>

### Experiments

- **Case 1**: `r0=1, r1=1`  
  - `W x 1`  
  - `R y 1`  
  - `R x 1`  

- **Case 2**: `r0=0, r1=1`  
  - `W x 1`  
  - `R y 0`  
  - `R x 1`  

- **Case 3**: `r0=1, r1=0`  
  - `W x 1`  
  - `R y 1`  
  - `R x 0`  

- **Case 4**: `r0=0, r1=0`  
  - `W x 1`  
  - `R y 0`  
  - `R x 0`  

**SC:** ✅
MODELLING WEAK MEMORY

\[
\begin{array}{c}
\text{MOV} [x] 1 \\
\text{MOV} \ r0 \ [y] \\
\text{MOV} \ r1 \ [x]
\end{array}
\]

SC: ✓  
SC: ✓
MODELLING WEAK MEMORY

**MOV** [x] 1
**MOV** r0 [y]
**MOV** r1 [x]

**MOV** [y] 1
**MOV** r0 [y]
**MOV** r1 [x]
MODELLING WEAK MEMORY

\[
\begin{align*}
\text{MOV} & \ [x] \ 1 \\
\text{MOV} & \ r0 \ [y] \\
\text{MOV} & \ r1 \ [x] \\
\end{align*}
\]

- \( r0 = 1 \) \( r1 = 1 \)
- \( r0 = 0 \) \( r1 = 1 \)
- \( r0 = 1 \) \( r1 = 0 \)
- \( r0 = 0 \) \( r1 = 0 \)

SC: ✔️

SC: ✔️

SC: ✔️

SC: ❌
MODELLING WEAK MEMORY

\[
\begin{align*}
\text{MOV } [x] & \quad 1 \\
\text{MOV } r0 & \quad [y] \\
\text{MOV } r1 & \quad [x]
\end{align*}
\]

\[
\begin{align*}
\text{MOV } [y] & \quad 1 \\
\text{MOV } r0 & \quad [y] \\
\text{MOV } r1 & \quad [x]
\end{align*}
\]
MODELLING WEAK MEMORY

\[
\begin{align*}
\text{MOV} & [x] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [y] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [x] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [y] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [x] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [y] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [x] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [y] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [x] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [y] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [x] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [y] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [x] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [y] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [x] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [y] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [x] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]

\[
\begin{align*}
\text{MOV} & [y] 1 \\
\text{MOV} & r0 [y] \\
\text{MOV} & r1 [x] \\
\end{align*}
\]
MODELLING WEAK MEMORY

\[\begin{align*}
\text{MOV } [x] & \ 1 \\
\text{MOV } r0 & \ [y] \\
\text{MOV } r1 & \ [x] \\
\end{align*}\]
MODELLING WEAK MEMORY

\[
\begin{align*}
\text{MOV} & \ [x] 1 \\
\text{MOV} & \ r0 \ [y] \\
\text{MOV} & \ r1 \ [x] \\
\text{MOV} & \ [y] 1
\end{align*}
\]

- \( W \times 1 \)
- \( R \times 0 \)
- \( R \times 1 \)
- \( W \times 1 \)

\[ r0=1 \quad r1=1 \]
\[ r0=0 \quad r1=1 \]
\[ r0=1 \quad r1=0 \]
\[ r0=0 \quad r1=0 \]

SC: ✓

x86: ✓
• Weak memory

• Transactions

• Weak memory and transactions

• Validating our models

• The problem with lock elision

• Related and future work
Transactional Memory:
Architectural Support for Lock-Free Data Structures

Maurice Herlihy
Digital Equipment Corporation
Cambridge Research Laboratory
Cambridge MA 02139
herlihy@crl.dec.com

Abstract

The basic idea of a transactional memory system is to let the programmer specify, if its operations do not 
require global mutual exclusion, that they can run concurrently. The system then automatically 
ensures no race-conditions occur due to multiple concurrent transactions.

J. Eliot B. Moss
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
moss@cs.umass.edu

structures avoid common problems of conventional locking techniques in high-

Priority inversion occurs when a thread is preempted while holding a lock.
TRANSACTIONAL MEMORY

- **X86:**
  
  ```
  XBEGIN
  MOV [x] 42
  MOV [y] 36
  XEND
  ```

- **Power:**
  
  ```
  tbegin
  stw x #42
  stw y #36
  tend
  ```

- **C++:**
  
  ```
  atomic {
    *x = 42;
    *y = 36;
  }
  ```
OUTLINE

• Weak memory
• Transactions
  • Weak memory and transactions
  • Validating our models
  • The problem with lock elision
  • Related and future work
WEAK MEMORY + TM = ?

\[
\begin{align*}
\text{MOV} & \ [x] 1 \\
\text{MOV} & \ r0 [y] \\
\text{MOV} & \ [y] 1 \\
\text{MOV} & \ r1 [x]
\end{align*}
\]

\[
\begin{align*}
\text{SC} & \\
\text{x86}
\end{align*}
\]
WEAK MEMORY + TM = ?

\[
\begin{array}{|c|}
\hline
\text{XBEGIN} & \text{XBEGIN} \\
\text{MOV} [x] \ 1 & \text{MOV} [y] \ 1 \\
\text{MOV} \ r0 \ [y] & \text{MOV} \ r1 \ [x] \\
\text{XEND} & \text{XEND} \\
\hline
\end{array}
\]

\[
\begin{align*}
\text{r0} &= 1 & \text{r0} &= 0 & \text{r0} &= 1 & \text{r0} &= 0 \\
\text{r1} &= 1 & \text{r1} &= 1 & \text{r1} &= 0 & \text{r1} &= 0 \\
\end{align*}
\]

SC

\[\text{x86}\]
WEAK MEMORY + TM = ?

XBEGIN
MOV [x] 1
MOV r0 [y]
XEND

XBEGIN
MOV [y] 1
MOV r1 [x]
XEND

r0=1   r0=0   r0=1   r0=0
r1=1   r1=1   r1=0   r1=0
WEAK MEMORY + TM = ?

Transactionally Safe UC

\[
\begin{align*}
\text{XBEGIN} & \quad \text{XBEGIN} \\
\text{MOV} [x] 1 & \quad \text{MOV} [y] 1 \\
\text{MOV} r0 [y] & \quad \text{MOV} r1 [x] \\
\text{XEND} & \quad \text{XEND}
\end{align*}
\]

\[
\begin{align*}
r0 = 1 & \quad r0 = 0 & \quad r0 = 1 & \quad r0 = 0 \\
r1 = 1 & \quad r1 = 1 & \quad r1 = 0 & \quad r1 = 0
\end{align*}
\]

transactional SC

SC

x86
WEAK MEMORY + TM = ?

sequential consistency
WEAK MEMORY + TM = ?

sequential consistency

weak consistency
WEAK MEMORY + TM = ?

weak consistency

sequential consistency

transactional consistent
WEAK MEMORY + TM = ?

transactional sequential consistency

sequential consistency

weak consistency

transactional weak consistency
WEAK MEMORY + TM = ?

- Transactional consistency
- Sequential consistency
- Weak consistency

- Transactional weak consistency:
  - Sequential consistency
READ-MODIFY-WRITES
READ-MODIFY-WRITES

```
ldxr r1, [x]
add r1, r1, #2
stxr r2, r1, [x]
str #1, [x]
```
READ-MODIFY-WRITES

\[
\begin{align*}
\text{ldxr} & \quad r1, [x] \\
\text{add} & \quad r1, r1, \#2 \\
\text{stxr} & \quad r2, r1, [x] \\
\text{str} & \quad \#1, [x]
\end{align*}
\]

\[
\begin{align*}
R^* x 0 \\
\text{rmwr} \quad \downarrow \text{po} \\
W^* x 2 \\
\text{fr} \quad \downarrow \text{co} \quad W x 1
\end{align*}
\]

\[
r1=2 \quad r2=0 \quad x=1
\]
**READ-MODIFY-WRITES**

\[
\begin{align*}
\text{ldxr } & r1, [x] \\
\text{add } & r1, r1, #2 \\
\text{stxr } & r2, r1, [x]
\end{align*}
\]
READ-MODIFY-WRITES

\[ \text{ldxr } r1, [x] \]
\[ \text{add } r1, r1, #2 \]
\[ \text{stxr } r2, r1, [x] \]
\[ \text{str } \#1, [x] \]

R* x 0

rmw po

W* x 2

fr

W x 1

co

r1=2  r2=0  x=1

R* x 1

rmw po

W x 1

rf

W* x 3

co

r1=3  r2=0  x=3
**READ-MODIFY-WRITES**

```
ldxr r1, [x]
add r1, r1, #2
stxr r2, r1, [x]
```

```
str #1, [x]
```

```
R* x 0
```

```
rmw po
```

```
W x 1
```

```
W* x 2
```

```
r1=2 r2=0 x=1
```

```
R* x 1
```

```
rmw po
```

```
W x 1
```

```
W* x 3
```

```
r1=3 r2=0 x=3
```
READ-MODIFY-WRITES

\[
\begin{align*}
\text{ldxr } & r1, [x] & \quad \text{str } & \#1, [x] \\
\text{add } & r1, r1, \#2 & \quad & \\
\text{stxr } & r2, r1, [x] \\
\end{align*}
\]

\[
\begin{align*}
R^* x 0 & \quad \text{fr} & \quad W x 1 \\
\text{rmw po} & \quad \text{co} & \quad & \\
W^* x 2 & \quad & & \\
r1=2 & \quad r2=0 & \quad x=1 \\
\end{align*}
\]

\[
\begin{align*}
R x 0 & \quad \text{fr} & \quad W x 1 \\
r1=2 & \quad r2=1 & \quad x=1 \\
\end{align*}
\]

\[
\begin{align*}
R^* x 1 & \quad \text{rf} & \quad W x 1 \\
\text{rmw po} & \quad \text{co} & \quad & \\
W^* x 3 & \quad & & \\
r1=3 & \quad r2=0 & \quad x=3 \\
\end{align*}
\]
READ-MODIFY-WRITES

```assembly
ldxr r1, [x]
add r1, r1, #2
stxr r2, r1, [x]
str #1, [x]
```

![Diagram](attachment:image.png)

- **R** x 0
  - rmw
  - po
  - W x 1
  - co
- **W** x 2
- **R** x 1
  - rmw
  - po
  - W x 1
  - rf
- **W** x 3

- r1 = 2, r2 = 0, x = 1
- r1 = 2, r2 = 1, x = 1
- r1 = 3, r2 = 0, x = 3
READ-MODIFY-WRITES

\[
\begin{align*}
\text{ldxr } r1, [x] & \quad \text{str } #1, [x] \\
\text{add } r1, r1, \#2 & \\
\text{stxr } r2, r1, [x] & \\
\end{align*}
\]

\[
\begin{align*}
R^* x 0 & \quad \text{fr} \\
\text{rmw} & \quad \text{po} \\
W^* x 2 & \quad \text{co} \\
\end{align*}
\]

\[
\begin{align*}
\text{fr} & \\
W x 1 & \\
\end{align*}
\]

\[
\begin{align*}
r1=2 & \\
r2=0 & \\
x=1 & \\
\end{align*}
\]

\[
\begin{align*}
R x 0 & \quad \text{fr} \\
W x 1 & \\
\end{align*}
\]

\[
\begin{align*}
r1=2 & \\
r2=1 & \\
x=1 & \\
\end{align*}
\]

\[
\begin{align*}
R^* x 1 & \quad \text{rf} \\
\text{rmw} & \quad \text{po} \\
W^* x 3 & \quad \text{co} \\
\end{align*}
\]

\[
\begin{align*}
\text{fr} & \\
W x 1 & \\
\end{align*}
\]

\[
\begin{align*}
r1=3 & \\
r2=0 & \\
x=3 & \\
\end{align*}
\]

\[
\begin{align*}
R x 1 & \quad \text{rf} \\
W x 1 & \\
\end{align*}
\]

\[
\begin{align*}
r1=3 & \\
r2=1 & \\
x=1 & \\
\end{align*}
\]

13
READ-MODIFY-WRITES

\[ \text{ldxr } r1, [x] \quad \text{str } \#1, [x] \]
\[ \text{add } r1, r1, \#2 \]
\[ \text{stxr } r2, r1, [x] \]

**Example Diagrams:**

- **R* x 0**
  - rmw \( \rightarrow \) po
  - W x 1
  - co
  
  \( r1=2 \quad r2=0 \quad x=1 \)

- **R x 0**
  - fr
  - W x 1
  
  \( r1=2 \quad r2=1 \quad x=1 \)

- **R* x 1**
  - rmw \( \rightarrow \) po
  - W x 1
  - co
  
  \( r1=3 \quad r2=0 \quad x=3 \)

- **R x 1**
  - rf
  - W x 1
  
  \( r1=3 \quad r2=1 \quad x=1 \)
READ-MODIFY-Writes

\[
\begin{align*}
\text{ldxr} & \ r1, [x] \\
\text{add} & \ r1, r1, #2 \\
\text{stxr} & \ r2, r1, [x] \\
\text{str} & \ #1, [x]
\end{align*}
\]
READ-MODIFY-WRITES

\[
\begin{align*}
\text{ldxr} & \quad r1, [x] \\
\text{add} & \quad r1, r1, #2 \\
\text{stxr} & \quad r2, r1, [x] \\
\text{str} & \quad #1, [x]
\end{align*}
\]

**Case 1:**
- Initial values: $r1=2$, $r2=0$, $x=1$
- Execution sequence:
  - $\text{ldxr} r1, [x]$
  - $\text{add} r1, r1, #2$
  - $\text{stxr} r2, r1, [x]$
- Final states: $r1=2$, $r2=0$, $x=1$

**Case 2:**
- Initial values: $r1=2$, $r2=1$, $x=1$
- Execution sequence:
  - $\text{ldxr} r1, [x]$
  - $\text{add} r1, r1, #2$
  - $\text{stxr} r2, r1, [x]$
- Final states: $r1=2$, $r2=1$, $x=1$

**Case 3:**
- Initial values: $r1=3$, $r2=0$, $x=3$
- Execution sequence:
  - $\text{ldxr} r1, [x]$
  - $\text{add} r1, r1, #2$
  - $\text{stxr} r2, r1, [x]$
- Final states: $r1=3$, $r2=0$, $x=3$

**Case 4:**
- Initial values: $r1=3$, $r2=0$, $x=1$
- Execution sequence:
  - $\text{ldxr} r1, [x]$
  - $\text{add} r1, r1, #2$
  - $\text{stxr} r2, r1, [x]$
- Final states: $r1=3$, $r2=0$, $x=1$

**Case 5:**
- Initial values: $r1=3$, $r2=1$, $x=1$
- Execution sequence:
  - $\text{ldxr} r1, [x]$
  - $\text{add} r1, r1, #2$
  - $\text{stxr} r2, r1, [x]$
- Final states: $r1=3$, $r2=1$, $x=1$
READ-MODIFY-WRITES

\[
\begin{align*}
\text{ldxr} & \quad r1, [x] \\
\text{add} & \quad r1, r1, #2 \\
\text{stx}r & \quad r2, r1, [x] \\
\text{str} & \quad #1, [x]
\end{align*}
\]
READ-MODIFY-WRITES

\[\begin{align*}
\text{ldxr} & \quad r1, [x] \\
\text{add} & \quad r1, r1, \#2 \\
\text{stxr} & \quad r2, r1, [x] \\
\text{str} & \quad \#1, [x]
\end{align*}\]

- **R* x 0**
  - rmw po
  - \(W \times 2\)
  - r1=2, r2=0, x=2
  - **Wrong**

- **R* x 0**
  - rmw po
  - \(W \times 2\)
  - r1=2, r2=0, x=1
  - **Correct**

- **R x 0**
  - fr
  - \(W \times 1\)
  - r1=2, r2=1, x=1
  - **Correct**

- **R* x 1**
  - rmw po
  - \(W \times 3\)
  - r1=3, r2=0, x=3
  - **Correct**

- **R* x 1**
  - rmw po
  - \(W \times 3\)
  - r1=3, r2=0, x=1
  - **Wrong**

- **R x 1**
  - rf
  - \(W \times 1\)
  - r1=3, r2=1, x=1
  - **Wrong**

- **R x 1**
  - rf
  - \(W \times 1\)
  - r1=3, r2=1, x=1
  - **Correct**
READ-MODIFY-WRITES

R* x 0
rmw → po
W* x 2
fr → W x 1
co

r1=2  r2=0  x=2
AXIOMS FOR TRANSACTIONS
AXIOMS FOR TRANSACTIONS
AXIOMS FOR TRANSACTIONS

R* x 0
rmw po
W* x 2

W x 1

fr

co

X

R x 0
po
W x 2

fr

co

X

R x 0
po
W x 1

fr

W x 1

X

R x 1

rf

X

W x 1

X
AXIOMS FOR TRANSACTIONS
acyclic($p_{loc} \cup com$)  
(Coherence)
empty($rmw \cap (fr_e ; co_e)$)  
(RMWIsol)
acyclic($hb$)  
(Order)

where $ppo = ((W \times W) \cup (R \times W) \cup (R \times R)) \cap po$

$tfence = po \cap (((\neg stxn ; stxn) \cup (stxn ; \neg stxn))$

$L = \text{domain}(rmw) \cup \text{range}(rmw)$

$implied = [L] ; po \cup po ; [L] \cup tfence$

$hb = mfence \cup ppo \cup implied \cup rf_e \cup fr \cup co$

acyclic($\text{stronglift}(com, stxn)$)  
(STRONGIsol)
acyclic($\text{stronglift}(hb, stxn)$)  
(TxnOrder)
ARM TRANSACTIONS

\[
\text{acyclic}(p_{\text{loc}} \cup \text{com}) \quad \text{(Coherence)}
\]
\[
\text{acyclic}(\text{ob}) \quad \text{(Order)}
\]

where \(dob = \text{(order imposed by dependencies, elided)}\)

\[
aob = \text{(order imposed by atomic RMWs, elided)}
\]

\[
bob = \text{(order imposed by barriers, elided)}
\]

\[
tfence = po \cap ((\neg stxn \cup stxn) \cup (stxn \cap \neg stxn))
\]

\[
ob = com_e \cup dob \cup aob \cup bob \cup tfence
\]

\[
\text{empty}(rmw \cap (fr_e \cup co_e)) \quad \text{(RMWISOL)}
\]

\[
\text{acyclic}(\text{stronglift}(\text{com}, stxn)) \quad \text{(STRONGISOL)}
\]

\[
\text{acyclic}(\text{stronglift}(ob, stxn)) \quad \text{(TxnOrder)}
\]

\[
\text{empty}(rmw \cap tfence^*) \quad \text{(TxnCancelsRMW)}
\]
POWER TRANSACTIONS

\begin{align*}
\text{acyclic}(po_{loc} \cup com) & \quad \text{(Coherence)}
\text{empty}(rmw \cap (fr_e; co_e)) & \quad \text{(RMWISOL)}
\text{acyclic}(hb) & \quad \text{(Order)}
\text{where } ppo = (preserved\ program\ order,\ elided) \\
\text{tfence} = po \cap ((\neg stxn; stxn) \cup (stxn; \neg stxn)) \\
\text{fence} = \text{sync} \cup \text{tfence} \cup (lwsync \setminus (W \times R)) \\
\text{ihb} = ppo \cup \text{fence} \\
\text{thb} = (fr_e \cup ((fr_e \cup co_e)^*; ihb))^*; (fr_e \cup co_e)^*; rf_e^? \\
\text{hb} = (rf_e^?; ihb; rf_e^?) \cup \text{weaklift}(thb, stxn)
\end{align*}
\begin{align*}
\text{acyclic}(co \cup prop) & \quad \text{(PROPAGATION)}
\text{where } efence = rf_e^?; \text{fence}; rf_e^?
\text{prop}_1 = [W]; efence; hb^*; [W] \\
\text{prop}_2 = \text{com}_e^*; efence^*; hb^*; (\text{sync} \cup \text{tfence}) ; hb^* \\
\text{tprop}_1 = rf_e; stxn; [W] \\
\text{tprop}_2 = stxn; rf_e \\
\text{prop} = \text{prop}_1 \cup \text{prop}_2 \cup \text{tprop}_1 \cup \text{tprop}_2
\end{align*}
\begin{align*}
\text{irreflexive}(fr_e; prop; hb^*) & \quad \text{(Observation)}
\text{acyclic}(\text{stronglift}(com, stxn)) & \quad \text{(StrongISOL)}
\text{acyclic}(\text{stronglift}(hb, stxn)) & \quad \text{(TXNORDER)}
\text{empty}(rmw \cap \text{tfence}^*) & \quad \text{(Txn Cancels RMW)}
\end{align*}
POWER TRANSACTIONS
POWER TRANSACTIONS
POWER TRANSACTIONS
POWER TRANSACTIONS
POWER TRANSACTIONS
POWERS TRANSACTIONS
POWER TRANSACTIONS
POWER TRANSACTIONS
POWER TRANSACTIONS
POWER TRANSACTIONS
acyclic($po_{loc} \cup com$)  
(\textbf{Coherence})

empty($rmw \cap (fr_e ; co_e)$)  
(RMWIsol)

acyclic($hb$)  
(Order)

where $ppo =$ (preserved program order, elided)

\[ tfence = po \cap ((\neg stxn ; stxn) \cup (stxn ; \neg stxn)) \]
\[ fence = sync \cup tfence \cup (lwsync \setminus (W \times R)) \]
\[ ihb = ppo \cup fence \]
\[ thb = (rf_e \cup ((fr_e \cup co_e)^* ; ihb))^* ; (fr_e \cup co_e)^* ; rf_e^? \]
\[ hb = (rf_e^? ; ihb ; rf_e^?) \cup \text{weaklift}(thb, stxn) \]

acyclic($co \cup prop$)  
(\textbf{Propagation})

where $efence = rf_e^? ; fence ; rf_e^?$

\[ prop_1 = [W] ; efence ; hb^* ; [W] \]
\[ prop_2 = com_e^* ; efence^* ; hb^* ; (sync \cup tfence) ; hb^* \]
\[ tprop_1 = rf_e ; stxn ; [W] \]
\[ tprop_2 = stxn ; rf_e \]

\[ prop = prop_1 \cup prop_2 \cup tprop_1 \cup tprop_2 \]

irreflexive($fr_e ; prop ; hb^*$)  
(Observation)

acyclic($\text{stronglift}(com, stxn)$)  
(\textbf{StrongIsol})

acyclic($\text{stronglift}(hb, stxn)$)  
(TxnOrder)

empty($rmw \cap tfence^*$)  
(TxnCancelsRMW)
C++ TRANSACTIONS

irreflexive(hb; com*)  \hspace{1cm} (HbCom)
where sw = (synchronises-with, elided)
\hspace{1cm} ecom = com \cup (co; rf)
\hspace{1cm} tsw = weaklift(ecom, stxn)
\hspace{1cm} hb = (sw \cup tsw \cup po)^+

empty(rmw \cap (fr_e; co_e)) \hspace{1cm} (RMWIsol)
acyclic(po \cup rf) \hspace{1cm} (NoThinAir)
acyclic(psc) \hspace{1cm} (SeqCst)
where psc = (constraints on SC events, elided)

empty(cnf \setminus Ato^2 \setminus (hb \cup hb^{-1})) \hspace{1cm} (NoRace)
where cnf = ((W \times W) \cup (R \times W) \cup (W \times R)) \cap sloc \setminus id
• Weak memory

• Transactions

• Weak memory and transactions

• Validating our models

• The problem with lock elision

• Related and future work
# Model Validation

| Arch. | \( |E| \) | Synthesis time (s) | Forbid | Allow |
|-------|---------|------------------|--------|-------|
|       |         |                  | T      | S     | \( \neg S \) | T | S | \( \neg S \) |
| x86   | 2       | 4                | 0      | 0     | 0           | 2 | 2 | 0   |
|       | 3       | 22               | 4      | 0     | 4           | 24| 23| 1   |
|       | 4       | 87               | 22     | 0     | 22          | 99| 99| 0   |
|       | 5       | 260              | 42     | 0     | 42          | 249| 244| 5   |
|       | 6       | 4402             | 133    | 0     | 133         | 895| 832| 63  |
|       | 7       | \( >7200 \)      | 307    | 0     | 307         | 2457| 1901| 556 |
| **Total (x86):** | | 508 | 0 | 508 | 3726 | 3101 | 625 |
| Power | 2       | 13               | 2      | 0     | 2           | 7 | 7 | 0   |
|       | 3       | 58               | 9      | 0     | 9           | 44| 44| 0   |
|       | 4       | 318              | 60     | 0     | 60          | 184| 175| 9   |
|       | 5       | 9552             | 353    | 0     | 353         | 1517| 1330| 187 |
|       | 6       | \( >7200 \)      | 922    | 0     | 922         | 5043| 4407| 636 |
| **Total (Power):** | | 1346 | 0 | 1346 | 6795 | 5963 | 832 |
MODEL VALIDATION

- Adding/coalescing/extend transactions should not introduce new behaviours.
Adding/coalescing/extending transactions should not introduce new behaviours.

Counterexample:
• Adding/coalescing/extending transactions should not introduce new behaviours.

• Counterexample:

\[
\begin{array}{c}
R^* \times 0 \\
\text{rmw} \Downarrow \text{po} \\
W^* \times 1
\end{array}
\]
Adding/coalescing/extending transactions should not introduce new behaviours.

Counterexample:
Adding/coalescing/extending transactions should not introduce new behaviours.

Counterexample:
Adding/coalescing/extending transactions should not introduce new behaviours.

Counterexample:

C++ transactions compile soundly to x86/Power/Arm transactions via the usual mapping
OUTLINE

• Weak memory

• Transactions

• Weak memory and transactions

• Validating our models

• The problem with lock elision

• Related and future work
LOCK ELISION

\[
\{\begin{align*}
\text{lock}() \\
\text{ldr} & \quad \text{W5},[X0] \\
\text{add} & \quad \text{W5,W5,#2} \\
\text{str} & \quad \text{W5,[X0]} \\
\text{unlock}() \end{align*}\}
\]

\[
\{\begin{align*}
\text{lock}() \\
\text{mov} & \quad \text{W7,#1} \\
\text{str} & \quad \text{W7,[X0]} \\
\text{unlock}() \end{align*}\}
\]

\[
x := x + 2
\]

\[
x := 1
\]
LOCK ELISION

Loop:

\[
\begin{align*}
\text{ldaxr} & \quad W2, [X1] \\
\text{cbnz} & \quad W2, \text{Loop} \\
\text{mov} & \quad W3, \#1 \\
\text{stxr} & \quad W4, W3, [X1] \\
\text{cbnz} & \quad W4, \text{Loop} \\
\text{ldr} & \quad W5, [X0] \\
\text{add} & \quad W5, W5, \#2 \\
\text{str} & \quad W5, [X0] \\
\text{stlr} & \quad WZR, [X1]
\end{align*}
\]

\[
\begin{align*}
\text{txbegin} \\
\text{txabort} \\
\text{L1:} \\
\text{txend}
\end{align*}
\]

\[
\begin{align*}
\{ & \quad x := x + 2 \\
\} & \quad x := 1
\end{align*}
\]
LOCK ELISION

Loop:
\text{ldaxr} \ W2, [X1]
\text{cbnz} \ W2, \text{Loop}
\text{mov} \ W3, \#1
\text{stxr} \ W4, W3, [X1]
\text{cbnz} \ W4, \text{Loop}
\text{ldr} \ W5, [X0]
\text{add} \ W5, W5, \#2
\text{str} \ W5, [X0]
\text{stlr} \ WZR, [X1]

\text{txbegin}
\text{txabort}

\text{txabort}
\text{txend}
LOCK ELISION

Loop:
✓ `ldaxr W2,[X1]
cbnz W2,Loop
mov W3,#1
stxr W4,W3,[X1]
cbnz W4,Loop
ldr W5,[X0]
add W5,W5,#2
str W5,[X0]
stlr WZR,[X1]

txbegin
ldr W6,[X1]
cbz W6,L1
txabort
L1:
mov W7,#1
str W7,[X0]
txend
LOCK ELISION

Loop:
✓ ldaxr  W2,[X1]
✓ cbnz  W2,Loop
  mov  W3,#1
  stxr  W4,W3,[X1]
  cbnz  W4,Loop
  ldr  W5,[X0]
  add  W5,W5,#2
  str  W5,[X0]
  stlr  WZR,[X1]

  txbegin
  ldr  W6,[X1]
  cbz  W6,L1
  txabort

  L1:
  mov  W7,#1
  str  W7,[X0]
  txend
LOCK ELISION

Loop:
\[
\begin{align*}
&\text{✓ ldaxr } W2, [X1] \\
&\text{✓ cbnz } W2, \text{Loop} \\
&\text{mov } W3, \#1 \\
&\text{stxr } W4, W3, [X1] \\
&\text{cbnz } W4, \text{Loop} \\
&\text{✓ ldr } W5, [X0] \\
&\text{add } W5, W5, \#2 \\
&\text{str } W5, [X0] \\
&\text{stlr } WZR, [X1]
\end{align*}
\]

\[
\begin{align*}
&\text{txbegin} \\
&\text{ldr } W6, [X1] \\
&\text{cbz } W6, \text{L1} \\
&\text{txabort} \\
&\text{L1:} \\
&\text{mov } W7, \#1 \\
&\text{str } W7, [X0] \\
&\text{txend}
\end{align*}
\]
LOCK ELISION

Loop:
✓ ldaxr  W2,[X1]
✓ cbnz  W2,Loop
  mov  W3,#1
  stxr  W4,W3,[X1]
  cbnz  W4,Loop
✓ ldr  W5,[X0]
  add  W5,W5,#2
  str  W5,[X0]
  stlr  WZR,[X1]
✓ txbegin
  ldr  W6,[X1]
  cbz  W6,L1
  txabort
L1:
  mov  W7,#1
  str  W7,[X0]
  txend
LOCK ELISION

Loop:
✓ ldaxr  W2,[X1]
✓ cbnz   W2,Loop
  mov   W3,#1
  stxr  W4,W3,[X1]
  cbnz  W4,Loop
✓ ldr   W5,[X0]
  add   W5,W5,#2
  str   W5,[X0]
  stlr  WZR,[X1]
  ✓ txbegin
✓ ldr   W6,[X1]
  cbz   W6,L1
  txabort
  L1:
✓ mov   W7,#1
✓ str   W7,[X0]
✓ txend
LOCK ELISION

Loop:
✓ load W2,[X1]
✓ compare W2,Loop
mov W3,#1
stx W4,W3,[X1]
✓ compare W4,Loop
✓ load W5,[X0]
add W5,W5,#2
str W5,[X0]
stlr WZR,[X1]
✓ tx begin
✓ load W6,[X1]
✓ compare W6,L1
tx abort
L1:
mov W7,#1
str W7,[X0]
tx end
LOCK ELISION

Loop:

- `ldaxr W2,[X1]`
- `cbnz W2,Loop`
- `mov W3,#1`
- `stxr W4,W3,[X1]`
- `cbnz W4,Loop`
- `ldr W5,[X0]`
- `add W5,W5,#2`
- `str W5,[X0]`
- `stlr WZR,[X1]`

- `txbegin`
- `ldr W6,[X1]`
- `cbz W6,L1`
- `txabort`

L1:

- `mov W7,#1`
- `str W7,[X0]`
- `txend`
LOCK ELISION

Loop:
✓ ldaxr W2,[X1] ✓ txbegin
✓ cbnz W2,Loop ✓ ldr W6,[X1]
mov W3,#1 ✓ cbz W6,L1
stxr W4,W3,[X1] ✓ txabort
cbnz W4,Loop ✓ L1:
✓ ldr W5,[X0] mov W7,#1
add W5,W5,#2 str W7,[X0]
str W5,[X0] txend
stlr WZR,[X1]
LOCK ELISION

Loop:
✓ ldaxr W2,[X1]
✓ cbnz W2,Loop
✓ mov W3,#1
✓ stxr W4,W3,[X1]
✓ cbnz W4,Loop
✓ ldr W5,[X0]
✓ add W5,W5,#2
✓ str W5,[X0]
✓ stlr WZR,[X1]
✓ txbegin
✓ ldr W6,[X1]
✓ cbz W6,L1
✓ txabort
✓ L1:
✓ mov W7,#1
✓ str W7,[X0]
✓ txend
LOCK ELISION

Loop:

✓ ldaxr  W2,[X1]
✓ cbnz  W2,Loop
  mov  W3,#1
  stxr  W4,W3,[X1]
✓ cbnz  W4,Loop
✓ ldr  W5,[X0]
  add  W5,W5,#2
  str  W5,[X0]
✓ stlr  WZR,[X1]
✓ txbegin
✓ ldr  W6,[X1]
✓ cbz  W6,L1
✓ txabort
✓ L1:
  mov  W7,#1
  str  W7,[X0]
  txend
LOCK ELISION

Loop:

✓ ldaxr W2,[X1]  ✓ txbegin
✓ cbnz W2,Loop  ✓ ldr W6,[X1]
  mov W3,#1  ✓ cbz W6,L1
  stxr W4,W3,[X1]  ✓ txabort
  cbnz W4,Loop  ✓ L1:
✓ ldr W5,[X0]  ✓ mov W7,#1
  add W5,W5,#2  ✓ str W7,[X0]
  str W5,[X0]  ✓ txend
  stlr WZR,[X1]
LOCK ELISION

Loop:
✓ ldaxr W2,[X1] ✓ txbegin
✓ cbnz W2,Loop ✓ ldr W6,[X1]
✓ mov W3,#1 ✓ cbz W6,L1
   stxr W4,W3,[X1] ✓ txabort
   cbnz W4,Loop ✓ L1:
   ✓ ldr W5,[X0] ✓ mov W7,#1
   add W5,W5,#2 ✓ str W7,[X0]
   str W5,[X0] ✓ txend
   stlr WZR,[X1]
LOCK ELISION

Loop:
✓ ldaxr W2,[X1]
✓ cbnz W2,Loop
✓ mov W3,#1
✓ stxr W4,W3,[X1]
✓ cbnz W4,Loop
✓ ldr W5,[X0]
✓ add W5,W5,#2
✓ str W5,[X0]
✓ stlr WZR,[X1]
✓ txbegin
✓ ldr W6,[X1]
✓ cbz W6,L1
✓ txabort
✓ L1:
✓ mov W7,#1
✓ str W7,[X0]
✓ txend
LOCK ELISION

Loop:
✓ ldaxr W2,[X1] ✓ txbegin
✓ cbnz W2,Loop ✓ ldr W6,[X1]
✓ mov W3,#1 ✓ cbz W6,L1
✓ stxr W4,W3,[X1] ✓ txabort
✓ cbnz W4,Loop ✓ L1:
✓ ldr W5,[X0] ✓ mov W7,#1
add W5,W5,#2 ✓ str W7,[X0]
str W5,[X0] ✓ txend
stlr WZR,[X1]
LOCK ELISION

Loop:
✓ ldaxr W2, [X1]
✓ cbnz W2, Loop
✓ mov W3, #1
✓ stxr W4, W3, [X1]
✓ cbnz W4, Loop
✓ ldr W5, [X0]
✓ add W5, W5, #2
str W5, [X0]
stlr WZR, [X1]
✓ txbegin
✓ ldr W6, [X1]
✓ cbz W6, L1
✓ txabort
✓ L1:
✓ mov W7, #1
✓ str W7, [X0]
✓ txend
LOCK ELISION

Loop:
- **ldaxr** W2,[X1]
- **cbnz** W2,Loop
- **mov** W3,#1
- **stxr** W4,W3,[X1]
- **cbnz** W4,Loop
- **ldr** W5,[X0]
- **add** W5,W5,#2
- **str** W5,[X0]
- **stlr** WZR,[X1]

- **txbegin**
- **txabort**

L1:
- **cbz** W6,L1
- **txbegin**
- **ldr** W6,[X1]
- **mov** W7,#1
- **str** W7,[X0]
- **txend**
LOCK ELISION

Loop:
✓ ldaxr  W2,[X1]
✓ cbnz  W2,Loop
✓ mov  W3,#1
✓ stxr  W4,W3,[X1]
✓ cbnz  W4,Loop
✓ ldr  W5,[X0]
✓ add  W5,W5,#2
✓ str  W5,[X0]
✓ stlr  WZR,[X1]
✓ txbegin
✓ ldr  W6,[X1]
✓ cbz  W6,L1
✓ txabort
✓ L1:
✓ mov  W7,#1
✓ str  W7,[X0]
✓ txend
OUTLINE

• Weak memory
• Transactions
• Weak memory and transactions
• Validating our models
• The problem with lock elision
• Related and future work
RELATED WORK

- Dongol, Jagadeesan, and Riely (POPL '18):

  👎 atomicity only
  👎 not empirically validated
  👍 handle aborted transactions
  👎 establish metatheory
FUTURE DIRECTIONS
FUTURE DIRECTIONS

• Validate our model of C++ transactions
FUTURE DIRECTIONS

• Validate our model of C++ transactions

• Account for aborted/failed transactions
FUTURE DIRECTIONS

• Validate our model of C++ transactions

• Account for aborted/failed transactions

• Extend Power model to handle "rollback-only" transactions, and tsuspend and tresume instructions
FUTURE DIRECTIONS

• Validate our model of C++ transactions

• Account for aborted/failed transactions

• Extend Power model to handle "rollback-only" transactions, and `tsuspend` and `tresume` instructions

• Operational models
FUTURE DIRECTIONS

• Validate our model of C++ transactions

• Account for aborted/failed transactions

• Extend Power model to handle "rollback-only" transactions, and \texttt{tsuspend} and \texttt{tresume} instructions

• Operational models

• Verify code that implements or uses TM
THE SEMANTICS OF TRANSACTIONS AND WEAK MEMORY IN X86, POWER, ARM, AND C++

Nathan Chong
Arm Ltd.

Tyler Sorensen
Imperial

John Wickerson
Imperial

UCL PPLV Seminar, Thursday 10 May 2018