Journey to the Centre of the OpenCL Memory Model

John Wickerson
Imperial College London
COLLABORATORS

Mark Batty
George Constantinides
Nadesh Ramanathan
Alastair Donaldson
Tyler Sorensen
Brad Beckmann
THIS TALK
• Weak memory
THIS TALK

- Weak memory
- Formalising the OpenCL memory model
THIS TALK

• Weak memory

• Formalising the OpenCL memory model

• Implementing the OpenCL memory model on GPUs
THIS TALK

• Weak memory

• Formalising the OpenCL memory model

• Implementing the OpenCL memory model on GPUs

• Implementing the OpenCL memory model on FPGAs
WEAK MEMORY
WEAK MEMORY

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

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

r0=1
r1=1
WEAK MEMORY

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

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

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

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

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

\[
\begin{array}{cccc}
r0=1 & r0=0 & r0=1 & r0=0 \\
r1=1 & r1=1 & r1=0 & r1=0
\end{array}
\]
WEAK MEMORY

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

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

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

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

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

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

MOV [y] 1
MOV r1 [x]
r0=0
r1=0

x86

IBM

intel
WEAK MEMORY

MOV [x] 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
WEAK MEMORY

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

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

x86

ARM
AMD
NVIDIA
IBM
WEAK MEMORY

ARM®

AMD

NVIDIA®

MOV [x] 1
MOV r0 [y]
MOV

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

MOV

x86

C++

IBM

THE PROGRAMMING LANGUAGE

int

intel®
WEAK MEMORY

ARM

AMD

NVIDIA

Java

C++

IBM

intel

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

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

x86
WEAK MEMORY

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

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

- Weak memory

- Formalising the OpenCL memory model
- Implementing the OpenCL memory model on GPUs
- Implementing the OpenCL memory model on FPGAs
OpenCL

CPU

GPU
For an atomic operation $B$ that reads the value of an atomic object $M$, if there is a `memory_order_seq_cst` fence $X$ sequenced-before $B$, then $B$ observes either the last `memory_order_seq_cst` modification of $M$ preceding $X$ in the total order $S$ or a later modification of $M$ in its modification order. [OpenCL 2.0 standard, 2015]
For an atomic operation $B$ that reads the value of an atomic object $M$, if there is a `memory_order_seq_cst` fence $X$ sequenced-before $B$, then $B$ observes either the last `memory_order_seq_cst` modification of $M$ preceding $X$ in the total order $S$ or a later modification of $M$ in its modification order. [OpenCL 2.0 standard, 2015]
For an atomic operation $B$ that reads the value of an atomic object $M$, if there is a memory_order_seq_cst fence $X$ sequenced-before $B$, then $B$ observes either the last memory_order_seq_cst modification of $M$ preceding $X$ in the total order $S$ or a later modification of $M$ in its modification order.

[OpenCL 2.0 standard, 2015]
For an atomic operation \( B \) that reads the value of an atomic object \( M \), if there is a memory_order_seq_cst fence \( X \) sequenced-before \( B \), then \( B \) observes either the last memory_order_seq_cst modification of \( M \) preceding \( X \) in the total order \( S \) or a later modification of \( M \) in its modification order.

[OpenCL 2.0 standard, 2015]
For an atomic operation $B$ that reads the value of an atomic object $M$, if there is a `memory_order_seq_cst` fence $X$ sequenced-before $B$, then $B$ observes either the last `memory_order_seq_cst` modification of $M$ preceding $X$ in the total order $S$ or a later modification of $M$ in its modification order. [OpenCL 2.0 standard, 2015]
THIS TALK

- Weak memory

- Formalising the OpenCL memory model

- Implementing the OpenCL memory model on GPUs

- Implementing the OpenCL memory model on FPGAs
OpenCL
OpenCL to AMD GPU
Model of GPU hardware

The state of work-group

Notation

methods, and exposes a queue

tain which addresses have been flushed. We assume that the

own device/work-group/thread identifier, threads can ascer-

be flushed to the lower levels of the cache; by inserting flush

ogy [10]. It contains a queue of addresses that may need to

nent introduced as part of AMD's QuickRelease technol-

or

hygiene bit (\(C\)). The synchronisation fifo is a hardware compo-

Figure 4.

Figure 3.

A machine state

Cache

Addr

Val

\(\begin{array}{c}
\text{SyState} \\
\text{DvState} \\
\text{WgState} \\
\text{ThState} \\
\text{Reg} \\
\text{Val} \\
\text{rmw: Lock} \\
\text{Addr} \\
\text{Val} \\
\text{Hygiene} \\
\text{Freshness} \\
\text{Addr} \\
\text{Val} \\
\text{Hygiene} \\
\text{Freshness} \\
\text{Addr} \\
\text{Val} \\
\text{Fifo} \\
\text{Addr} \\
\text{Lock} \\
\end{array}\)

\(\text{L1: Cache} \)

\(\text{L2: Cache} \)

\(\text{Global} \)

\(\text{Addr} \)

\(\text{Val} \)
OpenCL ➔ AMD GPU
<table>
<thead>
<tr>
<th></th>
<th>na Or WG</th>
<th>DV (not remote)</th>
<th>DV (remote)</th>
</tr>
</thead>
<tbody>
<tr>
<td>r=load(x)</td>
<td>LD r x</td>
<td>INV\textsubscript{L1} WG</td>
<td>FLU\textsubscript{L1} DV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LD r x</td>
<td>INV\textsubscript{L1} WG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LD r x</td>
</tr>
<tr>
<td>store(x,r)</td>
<td>ST r x</td>
<td>FLU\textsubscript{L1} WG</td>
<td>FLU\textsubscript{L1} WG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ST r x</td>
<td>ST r x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>INV\textsubscript{L1} DV</td>
</tr>
<tr>
<td>r=fetch\textsubscript{inc}(x)</td>
<td>INC\textsubscript{L1} r x</td>
<td>FLU\textsubscript{L1} WG</td>
<td>FLU\textsubscript{L1} DV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INC\textsubscript{L1} WG</td>
<td>INV\textsubscript{L1} WG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INC\textsubscript{L2} r x</td>
<td>INV\textsubscript{L1} DV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>na or WG</td>
<td>DV (not remote)</td>
<td>DV (remote)</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>r=load(x)</code></td>
<td>LD <code>r x</code></td>
<td>INV\textsubscript{L1} WG LD <code>r x</code></td>
<td>FLU\textsubscript{L1} DV INV\textsubscript{L1} WG LD <code>r x</code> {LK x}</td>
</tr>
<tr>
<td><code>store(x,r)</code></td>
<td>ST <code>r x</code></td>
<td>FLU\textsubscript{L1} WG ST <code>r x</code></td>
<td>FLU\textsubscript{L1} WG ST <code>r x</code> {LK x}</td>
</tr>
<tr>
<td><code>r=fetch_inc(x)</code></td>
<td>INC\textsubscript{L1} <code>r x</code></td>
<td>FLU\textsubscript{L1} WG INC\textsubscript{L1} WG INC\textsubscript{L2} <code>r x</code></td>
<td>FLU\textsubscript{L1} DV INC\textsubscript{L1} WG INC\textsubscript{L2} <code>r x</code> {LK x} {LK\textsubscript{rmw}}</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>r = load(x)</th>
<th>na or WG</th>
<th>DV (not remote)</th>
<th>DV (remote)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD r x</td>
<td>INV$_{L1}$ WG LD r x</td>
<td>FLU$<em>{L1}$ DV INV$</em>{L1}$ WG LD r x</td>
<td>LK x</td>
</tr>
</tbody>
</table>

message passing error

<table>
<thead>
<tr>
<th>store(x, r)</th>
<th>ST r x</th>
<th>FLU$_{L1}$ WG ST r x</th>
<th>FLU$_{L1}$ WG ST r x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>INV$_{L1}$ DV</td>
<td>LK x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r = fetch_inc(x)</th>
<th>INC$_{L1}$ r x</th>
<th>FLU$<em>{L1}$ WG INC$</em>{L1}$ r x</th>
<th>FLU$<em>{L1}$ DV INC$</em>{L1}$ r x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>INV$<em>{L1}$ WG INC$</em>{L2}$ r x</td>
<td>LK x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INV$_{L1}$ DV</td>
<td>LK$_{rmw}$ x</td>
</tr>
<tr>
<td></td>
<td>na or WG</td>
<td>DV (not remote)</td>
<td>DV (remote)</td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>r=load(x)</code></td>
<td>LD <code>r x</code></td>
<td>INV\textsubscript{L1} WG LD <code>r x</code></td>
<td>FLU\textsubscript{L1} DV { LK <code>x</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>INV\textsubscript{L1} WG LD <code>r x</code></td>
<td>INV\textsubscript{L1} DV } LK <code>x</code></td>
</tr>
</tbody>
</table>

| `store(x,r)`   | ST `r x` | FLU\textsubscript{L1} WG ST `r x` | FLU\textsubscript{L1} WG \} LK `x` |
|                |          | INV\textsubscript{L1} DV \} LK `x` | RMW atomicity error |

<p>| <code>r=fetch_inc(x)</code> | INC\textsubscript{L1} <code>r x</code> | FLU\textsubscript{L1} WG INC\textsubscript{L1} <code>r x</code> | FLU\textsubscript{L1} DV } LK <code>x</code> |
|                 |                      | INV\textsubscript{L1} WG INC\textsubscript{L2} <code>r x</code> | } LK_{rmw} |
|                 |                      | INV\textsubscript{L1} DV |</p>
<table>
<thead>
<tr>
<th></th>
<th>na Or WG</th>
<th>DV (not remote)</th>
<th>DV (remote)</th>
</tr>
</thead>
<tbody>
<tr>
<td>r=load(x)</td>
<td>LD r x</td>
<td>INV&lt;sub&gt;L1&lt;/sub&gt; WG LD r x</td>
<td>FLU&lt;sub&gt;L1&lt;/sub&gt; DV INV&lt;sub&gt;L1&lt;/sub&gt; WG LD r x } LK x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INV&lt;sub&gt;L1&lt;/sub&gt; WG LD r x</td>
<td></td>
</tr>
<tr>
<td>store(x,r)</td>
<td>ST r x</td>
<td>FLU&lt;sub&gt;L1&lt;/sub&gt; WG ST r x</td>
<td>FLU&lt;sub&gt;L1&lt;/sub&gt; WG ST r x } LK x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INV&lt;sub&gt;L1&lt;/sub&gt; DV</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r=fetch_inc(x)</td>
<td>INC&lt;sub&gt;L1&lt;/sub&gt; r x</td>
<td>FLU&lt;sub&gt;L1&lt;/sub&gt; WG INC&lt;sub&gt;L1&lt;/sub&gt; r x</td>
<td>FLU&lt;sub&gt;L1&lt;/sub&gt; DV INC&lt;sub&gt;L1&lt;/sub&gt; WG INC&lt;sub&gt;L2&lt;/sub&gt; r x } LK x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INV&lt;sub&gt;L1&lt;/sub&gt; WG INC&lt;sub&gt;L1&lt;/sub&gt; r x</td>
<td>INC&lt;sub&gt;L1&lt;/sub&gt; DV</td>
</tr>
</tbody>
</table>

- **message passing error**
- **RMW atomicity error**
- **unnecessary locking**
<table>
<thead>
<tr>
<th>na Or WG</th>
<th>DV (not remote)</th>
<th>DV (remote)</th>
</tr>
</thead>
</table>
| **r=load(x)** | LD r x | LD r x
|            | INV$_{L1}$ WG | FLU$_{L1}$ DV
|            |               | INV$_{L2}$ LG |
| **store(x,r)** | ST r x | FLU$_{L1}$ LG
|            | ST r x | ST r x |
|            |       | \{LK$_{rmw}$ |
| **r=fetch_inc(x)** | INC$_{L1}$ r x | FLU$_{L1}$ LG
|            | INC$_{L2}$ r x | FLU$_{L1}$ DV
|            | INV$_{L1}$ LG | INV$_{L1}$ LG |
|            |               | \{LK$_{rmw}$ |
"Only after the work by John and others in the programming languages community, have programmers gained confidence in using fine-grain inter-thread communication and synchronization on GPUs."

Dr Brad Beckmann, Principal Researcher, AMD
"Only after the work by John and others in the programming languages community, have programmers gained confidence in using fine-grain inter-thread communication and synchronization on GPUs."

Dr Brad Beckmann, Principal Researcher, AMD

Key publication: ACM OOPSLA '15 (with Batty, Beckmann, & Donaldson)
OpenCL

GPU

AMD

nVIDIA

The C++ Programming Language
OpenCL

CUDA

The C++ Programming Language

AMD GPU

NVIDIA

IBM
OpenCL

C++ Programming Language

GPU
AMD
NVIDIA

CPU
IBM
Intel
OpenCL

The C++ Programming Language

GPU
AMD
NVIDIA

CPU
IBM
Intel
ARM
OpenCL

The C++ Programming Language

GPU
AMD
NVIDIA

IBM
intel
ARM
CPU
Model of the ARM architecture
Model of the 'transactional lock elision' process

Model of the ARM architecture
lock()  
LDR W5, [X0]  
ADD W5, W5, #2  
STR W5, [X0]  
unlock()  

lock()  
MOV W7, #1  
STR W7, [X0]  
unlock()  

Model of the ARM architecture  
Model of the 'transactional lock elision' process
Model of the ARM architecture

Model of the 'transactional lock elision' process

```
lock()
LDR W5,[X0]
ADD W5,W5,#2
STR W5,[X0]
unlock()
```

```
lock()
MOV W7,#1
STR W7,[X0]
unlock()
```

Key publication: ACM PLDI '18 (with Chong & Sorensen)
THIS TALK

- Weak memory
- Formalising the OpenCL memory model
- Implementing the OpenCL memory model on GPUs
- Implementing the OpenCL memory model on FPGAs
r = atomic_load(&y, memory_order_acquire);
r = atomic_load(&y, memory_order_acquire);

**not supported**
r = atomic_load(&y, memory_order_acquire);
lock();
r = y;
unlock();
## SCHEDULING ATOMICS

<table>
<thead>
<tr>
<th>Clock cycle:</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>r1 = x</td>
<td></td>
<td></td>
<td>load x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2 = atomic_load(&amp;y, acquire)</td>
<td></td>
<td></td>
<td></td>
<td>load y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r3 = z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>load z</td>
<td></td>
</tr>
</tbody>
</table>
# Scheduling Atomics

<table>
<thead>
<tr>
<th>Clock cycle:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1 = x</td>
<td></td>
<td><strong>load x</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2 = atomic_load(&amp;y, acquire)</td>
<td></td>
<td></td>
<td><strong>load y</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r3 = z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>load z</strong></td>
<td></td>
</tr>
</tbody>
</table>
## SCHEDULING ATOMICS

<table>
<thead>
<tr>
<th>Clock cycle:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1 = x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2 = atomic_load(&amp;y, acquire)</td>
<td></td>
<td>load y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r3 = z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>load z</td>
</tr>
</tbody>
</table>
SCHEDULING ATOMICS

<table>
<thead>
<tr>
<th>Clock cycle:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1 = x</td>
<td></td>
<td></td>
<td></td>
<td>load x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2 = atomic_load(&amp;y, acquire)</td>
<td>load y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r3 = z</td>
<td></td>
<td></td>
<td>load z</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"Too aggressive!"
## Scheduling Atomics

<table>
<thead>
<tr>
<th>Clock cycle:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1 = x</td>
<td></td>
<td></td>
<td>load x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2 = atomic_load(&amp;y, acquire)</td>
<td></td>
<td>load y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r3 = z</td>
<td></td>
<td></td>
<td></td>
<td>load z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### SCHEDULING ATOMICS

<table>
<thead>
<tr>
<th>Clock cycle:</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>r1 = x</td>
<td></td>
<td></td>
<td></td>
<td>load x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2 = atomic_load(&amp;y, acquire)</td>
<td></td>
<td>load y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r3 = z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>load z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"Just right!"
SCHEDULING ATOMICS

<table>
<thead>
<tr>
<th>Clock cycle:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1 = x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2 = atomic_load(&amp;y, acquire)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>load y</td>
<td></td>
</tr>
<tr>
<td>r3 = z</td>
<td></td>
<td></td>
<td></td>
<td>load z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"Too aggressive!"
### SCHEDULING ATOMICS

<table>
<thead>
<tr>
<th>Clock cycle:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1 = x</td>
<td></td>
<td>load x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2 = atomic_load(&amp;y, acquire)</td>
<td></td>
<td>load y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r3 = z</td>
<td></td>
<td>load z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"Too aggressive!" ... or is it?
SCHEDULING ATOMICS

a) Performance of global analysis compared to thread-local analysis, using weak atomics

b) Performance of weak atomics compared to SC atomics, using global analysis
SCHEDULING ATOMICS

a) Performance of global analysis compared to thread-local analysis, using weak atomics

b) Performance of weak atomics compared to SC atomics, using global analysis

Key publications: ACM FPGA '17, IEEE Trans. on Computers '17, IEEE FCCM '18 (with Ramanathan, Fleming & Constantinides)
THE FUTURE?
"Lack of C-to-RTL formal verification"

The biggest problem with HLS, according to a worldwide survey of 750 hardware engineers