Remote-Scope Promotion:

clarified  rectified  verified

Pittsburgh, 30 October 2015

OOPSLA

John Wickerson  Imperial  UK
Mark Batty  Kent  UK
Brad Beckmann  AMD  US
Ally Donaldson  Imperial  UK

Pittsburgh, 30 October 2015
In brief

- **Remote-scope promotion** is a GPU programming extension from **AMD** for efficient **work-stealing**
In brief

- **Remote-scope promotion** is a GPU programming extension from **AMD** for efficient **work-stealing**

- We **formalised** the design (at SW and HW level). This led to a **corrected** and **improved** implementation.

**Synchronization Using Remote-Scope Promotion**

Marc S. Orr\(^8\), Shuai Che\(^5\), Ayse Yilmazer\(^5\), Bradford M. Beckmann\(^5\), Mark D. Hill\(^6\), David A. Wood\(^7\)

\(^8\)University of Wisconsin–Madison
Computer Sciences

\(^6\)AMD Research


1. Introduction

As processors evolve to support more threads, synchronizing among those threads becomes increasingly expensive. This is particularly true for massively-threaded, through-socket architectures, such as graphics processing unit (GPU) systems.
In brief

- **Remote-scope promotion** is a GPU programming extension from **AMD** for efficient **work-stealing**

- We **formalised** the design (at SW and HW level). This led to a **corrected** and **improved** implementation.

- Formalise **early** in the design process!
1. Background: What is RSP?
2. Adding RSP to the OpenCL memory model
3. A formalised implementation of OpenCL+RSP
This talk

1. Background: What is RSP?
2. Adding RSP to the OpenCL memory model
3. A formalised implementation of OpenCL+RSP
C11: flat thread structure
OpenCL: thread groupings
GPUs: hierarchical memory
Memory scopes
Memory scopes
Memory scopes

store(x, 42)
Memory scopes

store(x, 42)

load(x)
Memory scopes

store\((x, 42, WG)\)  load\((x, WG)\)
Memory scopes

\text{store}(x,42,WG)

\text{load}(x,WG)
Memory scopes

\[ \text{store}(x,42,WG) \]

\[ \text{load}(x,WG) \]

faulty!
Memory scopes

store(x,42,DV)

load(x,DV)
Memory scopes

\[ \text{store}(x,42,\text{DV}) \]

\[ \text{load}(x,\text{DV}) \]

\( \text{workgroup} \)

\( \text{T0} \quad \text{T1} \)

\( \text{T2} \quad \text{T3} \)

\( \text{ok!} \)
Example: work-stealing
Example: work-stealing
Example: work-stealing

store(headA,_,WG) //pop
Example: work-stealing

store(headA,_,WG) //pop

store(headA,_,WG) //push

workgroup A

T0
T1

workgroup B

T2
T3

tailA  headA  tailB  headB
Example: work-stealing

store(headA, _, WG) //push

store(headA, _, WG) //pop

workgroup A

workgroup B

T0  T1

T2  T3

headA

tailA

tailB

headB
Example: work-stealing

store(headA,_,WG) //push
store(headA,_,WG) //pop
load(headA,_,???) //steal
Example: work-stealing

store(headA,_,WG) //push

store(headA,_,WG) //pop

load(headA,_,???) //steal

no way to plug this hole in OpenCL!
Remote-scope promotion

store(x, 42, WG)

load(x, DV)
Remote-scope promotion

\[ \text{store}(x, 42, \text{DV}) \]

\[ \text{load}(x, \text{DV}) \]
Remote-scope promotion

store(x,42,WG)

load(x,DV,remote)
Remote-scope promotion

store(x,42,DV,remote)

load(x,WG)
Work-stealing

store(headA,_,WG) //push

store(headA,_,WG) //pop

store(headA,_,DV,remote) //steal

workgroup A

workgroup B

T0  T1

T2  T3

tailA  headA  headB  tailB
This talk

1. Background: What is RSP?

2. Adding RSP to the OpenCL memory model

3. A formalised implementation of OpenCL+RSP
Scope inclusion in OpenCL
Scope inclusion in OpenCL

• Operations A and B only synchronise if they have inclusive scopes.
Scope inclusion in OpenCL

- Operations A and B only synchronise if they have **inclusive scopes**.

- A and B have **inclusive scopes** iff A reaches B and B reaches A.
Scope inclusion in OpenCL

- Operations A and B only synchronise if they have **inclusive scopes**.

- A and B have **inclusive scopes** iff
  A reaches B and B reaches A.

- A reaches B iff
  A has **workgroup** scope and B is in the same workgroup, or
  A has **device** scope and B is in the same device, or
  A has **all-devices** scope.
Scope inclusion in OpenCL+RSP

• Operations A and B only synchronise if they have **inclusive scopes**.

• A and B have **inclusive scopes** iff
  A reaches B and B reaches A, or
  A reaches B and B is remote, or
  A is remote and B reaches A.

• A reaches B iff
  A has **workgroup** scope and B is in the same workgroup, or
  A has **device** scope and B is in the same device, or
  A has **all-devices** scope.
(* Scope annotations *)

let s_wi = memory_scope_work_item
let s_wg = memory_scope_work_group
let s_dev = memory_scope_device
let s_all = memory_scope_all_svm_devices

(* Inclusive scopes *)

let incl1 = ([s_wi] ; wi)
  | ([s_wg] ; wg)
  | ([s_dev] ; dev)
  | ([s_all] ; unv)

let incl = (incl1 & (incl1^-1))
  | ([remote] ; incl1)
  | ((incl1^-1) ; [remote])

(******************************)
(* Synchronisation *)
(******************************)

let acq = (mo_acq | mo_sc | mo_acq_rel) & (R | F | rmw)
let rel = (mo_rel | mo_sc | mo_acq_rel) & (W | F | rmw)

(* Release sequence *)
Testing OpenCL+RSP programs
Testing OpenCL+RSP programs

• We simulated the 12 litmus tests designed by the original developers to define their expectations of RSP.
Babillion:herd jpw48$
Babillion:herd jpw48$ less testsuite/RSPTests/RSP_Test1.litmus
OpenCL RSP_Test1

{
    [x]=0;
    [y]=0;
}

P0 (global atomic_int* x, global atomic_int* y) {
    atomic_store_explicit
        (x, 1, memory_order_release, memory_scope_work_group);
}

P1 (global atomic_int* x, global atomic_int* y) {
    atomic_store_explicit
        (y, 1, memory_order_release, memory_scope_work_group);
}

P2 (global atomic_int* x, global atomic_int* y) {
    int r0 = atomic_load_explicit
        (x, memory_order_acquire, memory_scope_work_group);
    int r1 = atomic_load_explicit
        (y, memory_order_acquire, memory_scope_work_group);
}

P3 (global atomic_int* x, global atomic_int* y) {
    int r2 = atomic_load_explicit_remote
        (y, memory_order_acquire, memory_scope_device);
    int r3 = atomic_load_explicit_remote
        (x, memory_order_acquire, memory_scope_device);
}

scopeTree (device (work_group P0 P1 P2) (work_group P3))
exists (2:r0=1 \ 2:r1=0 \ 3:r2=1 \ 3:r3=0)
testsuite/RSPTests/RSP_Test1.litmus (END)
Babillion:herd jpw48$ less testsuite/RSPTests/RSP_Test1.litmus
Babillion:herd jpw48$ ./herd -initwrites true -model opencl_rem.cat testsuite/RSPTests/RSP_Test1.litmus
Babillion: herd jpw48$ less testsuite/RSPTests/RSP_Test1.litmus
Babillion: herd jpw48$ ./herd -initwrites true -model opencl_rem.cat testsuite/RSPTests/RSP_Test1.litmus
test RSP_Test1 Allowed
States 16
2:r0=0; 2:r1=0; 3:r2=0; 3:r3=0;
2:r0=0; 2:r1=0; 3:r2=0; 3:r3=1;
2:r0=0; 2:r1=0; 3:r2=1; 3:r3=0;
2:r0=0; 2:r1=0; 3:r2=1; 3:r3=1;
2:r0=0; 2:r1=1; 3:r2=0; 3:r3=0;
2:r0=0; 2:r1=1; 3:r2=0; 3:r3=1;
2:r0=0; 2:r1=1; 3:r2=1; 3:r3=0;
2:r0=0; 2:r1=1; 3:r2=1; 3:r3=1;
2:r0=1; 2:r1=0; 3:r2=0; 3:r3=0;
2:r0=1; 2:r1=0; 3:r2=0; 3:r3=1;
2:r0=1; 2:r1=0; 3:r2=1; 3:r3=0;
2:r0=1; 2:r1=0; 3:r2=1; 3:r3=1;
2:r0=1; 2:r1=1; 3:r2=0; 3:r3=0;
2:r0=1; 2:r1=1; 3:r2=0; 3:r3=1;
2:r0=1; 2:r1=1; 3:r2=1; 3:r3=0;
Ok
Witnesses
Positive: 1 Negative: 15
Bad executions (0 in total):
Condition exists (2:r0=1 \ 2:r1=0 \ 3:r2=1 \ 3:r3=0)
Observation RSP_Test1 Sometimes 15
Hash=305e6af6b482d95960e572605703996c
Babillion: herd jpw48$
Testing OpenCL+RSP programs

• We simulated the 12 litmus tests designed by the original developers to define their expectations of RSP.

• We found 8 were good, but:
  2 had unintentional races,
  1 enforced broken behaviour, and
  1 forbade reasonable behaviour.
Testing OpenCL+RSP programs

• We simulated the 12 litmus tests designed by the original developers to define their expectations of RSP.

• We found 8 were good, but:
  2 had unintentional races,
  1 enforced broken behaviour, and
  1 forbade reasonable behaviour.

• We also found (and fixed) bugs in their work-stealing queue implementation.
This talk

1. Background: What is RSP?
2. Adding RSP to the OpenCL memory model
3. A formalised implementation of OpenCL+RSP
Implementing RSP
Implementing RSP

- Model of GPU hardware
Implementing RSP

- Model of GPU hardware
- Assembly-like language
Implementing RSP

- Model of GPU hardware
- Assembly-like language
- Compiler mapping from OpenCL+RSP operations to sequences of assembly instructions
Implementing RSP

- Model of GPU hardware
- Assembly-like language
- Compiler mapping from OpenCL+RSP operations to sequences of assembly instructions
- Can then prove that all behaviours of the compiled program are allowed by the OpenCL+RSP memory model.
Model of GPU hardware

\[
\begin{align*}
x & \in \text{Addr} \\
r & \in \text{Reg} \\
v & \in \text{Val} \quad \text{def} \quad \mathbb{Z} \\
\text{FifoEl} & \quad \text{def} \quad \text{Addr} \cup \{\text{FLUSH}_{d,w,t} \mid d, w, t \in \mathbb{N}\} \\
\text{Fifo} & \quad \text{def} \quad \text{FifoEl queue} \\
\text{Hygiene} & \quad \text{def} \quad \{\text{CLEAN}, \text{DIRTY}\} \\
\text{Freshness} & \quad \text{def} \quad \{\text{VALID}, \text{INV’D}\} \\
\text{CacheEntry} & \quad \text{def} \quad \text{Val} \times (\text{hy: Hygiene}) \times (\text{fr: Freshness}) \\
C & \in \text{Cache} \quad \text{def} \quad (\text{Addr} \to \text{CacheEntry}) \times (\text{fifo: Fifo}) \\
\text{Lock} & \quad \text{def} \quad \{\text{🔒, ┇}\} \\
\text{ThState} & \quad \text{def} \quad \text{Reg} \to \text{Val} \\
\text{WgState} & \quad \text{def} \quad \text{ThState list} \times (\text{L1: Cache}) \times (\text{rmw: Lock}) \\
\text{DvState} & \quad \text{def} \quad \text{WgState list} \times (\text{L2: Cache}) \times \\
& \quad \text{lockfile: Addr} \to \text{Lock} \\
\text{Global} & \quad \text{def} \quad \text{Addr} \to \text{Val} \\
\Sigma & \in \text{SyState} \quad \text{def} \quad \text{DvState list} \times (\text{gl: Global})
\end{align*}
\]
Model of GPU hardware

SyState

DvState

WgState

ThState

Reg

Val

rmw: Lock

L1: Cache

Addr

Val

Hygiene

Freshness

Fifo

L2: Cache

Addr

Val

Hygiene

Freshness

Fifo

Addr

Val

Lock

Global
Model of GPU hardware

![Diagram of GPU hardware model](image)
Model of GPU hardware

Notation methods, and exposes a queue that addresses have been flushed. We assume that the markers (be flushed to the lower levels of the cache; by inserting flush

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

e introduced as part of AMD's QuickRelease technol-
or hygiene bit (C

\[ C \]

CacheEntry

x

v

r

SyState

DvState

WgState

ThState

Reg

Val

rmw: Lock

L1: Cache

Addr

Val

Hygiene

Freshness

Fifo

L2: Cache

Addr

Val

Hygiene

Freshness

Fifo

Addr

Lock

Global

Adj

Val

SyState

DvState

WgState

ThState

Reg

Val

rmw: Lock

L1: Cache

Addr

Val

Hygiene

Freshness

Fifo

L2: Cache

Addr

Val

Hygiene

Freshness

Fifo

Addr

Lock

Global

Adj

Val
Model of GPU hardware

Figure 3. Diagram of the model of GPU hardware.
Model of GPU hardware

- **Thread State**
  - Register (Reg)
    - Value (Val)
  - rmw: Lock

- **Device State**
  - Address (Addr)
    - Value (Val)
    - Hygiene
    - Freshness

- **Cache**
  - Address (Addr)
    - Value (Val)
    - Hygiene
    - Freshness

- **Fifo**
  - Address (Addr)
    - Lock
Model of GPU hardware

![Diagram of GPU hardware](image)

**SyState**

**DvState**

**WgState**

**ThState**

**Reg**

**Val**

**rmw: Lock**

**L1: Cache**

<table>
<thead>
<tr>
<th>Addr</th>
<th>Val</th>
<th>Hygiene</th>
<th>Freshness</th>
</tr>
</thead>
</table>

**L2: Cache**

<table>
<thead>
<tr>
<th>Addr</th>
<th>Val</th>
<th>Hygiene</th>
<th>Freshness</th>
</tr>
</thead>
</table>

**Fifo**

<table>
<thead>
<tr>
<th>Addr</th>
<th>Lock</th>
</tr>
</thead>
</table>

**Global**

<table>
<thead>
<tr>
<th>Addr</th>
<th>Val</th>
</tr>
</thead>
</table>
Model of GPU hardware

SyState

DvState

WgState

ThState

Reg

Val

rmw: Lock

L1: Cache

Addr

Val

Hygiene

Freshness

Fifo

L2: Cache

Addr

Val

Hygiene

Freshness

Fifo

Addr

Lock

Global

Addr

Val
Model of GPU hardware

![Model of GPU hardware diagram]

In the context of modeling GPU hardware, the state of thread, the state of work-group, and the state of a device are crucial. The diagram illustrates the relationship between the cache (L1 and L2) and different states such as WgState, DvState, and SyState. Each state is further divided into components such as Addr, Val, Hygiene, and Freshness. The diagram also includes a FIFO queue (Fifo) and lock (Lock) mechanisms, which are essential for managing data consistency and concurrent access.

The diagram helps in understanding how different states interact and how they influence the overall performance and reliability of GPU operations. This is particularly useful in the design and optimization of algorithms that run on GPUs, ensuring that data is accessed and manipulated efficiently and correctly.
Model of GPU hardware

Diagram of GPU hardware showing state variables and cache levels.
Model of GPU hardware

The model of GPU hardware is illustrated in the diagram above. It includes various states and data structures, such as:

- **SyState**: System state
- **DvState**: Device state
- **WgState**: Work-group state
- **ThState**: Thread state
- **L1: Cache**: Level 1 cache
- **L2: Cache**: Level 2 cache
- **Global**: Global state

Each state contains data structures for addresses, values, hygiene, and freshness. The diagram also includes a FIFO (First-In, First-Out) buffer, which is highlighted in purple.
Model of GPU hardware

Figure 4.

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

Def

De
Model of GPU hardware

SyState

DvState

WgState

ThState

Reg

Val

rmw: Lock

L1: Cache

Addr

Val

Hygiene

Freshness

Fifo

L2: Cache

Addr

Val

Hygiene

Freshness

Fifo

Global

Addr

Val

Addr

Lock
## Original scheme

<table>
<thead>
<tr>
<th></th>
<th>na or WG</th>
<th>DV (not remote)</th>
<th>DV (remote)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{r=load(x)}</td>
<td>LD \texttt{r x}</td>
<td>\texttt{INV}_{L1} \texttt{WG} \texttt{LD} \texttt{r x}</td>
<td>\texttt{FLU}<em>{L1} \texttt{DV} \texttt{INV}</em>{L1} \texttt{WG} \texttt{LD} \texttt{r x} { \texttt{LK x} }</td>
</tr>
<tr>
<td>\texttt{store(x,r)}</td>
<td>ST \texttt{r x}</td>
<td>\texttt{FLU}_{L1} \texttt{WG} \texttt{ST} \texttt{r x}</td>
<td>\texttt{FLU}<em>{L1} \texttt{WG} \texttt{ST} \texttt{r x} \texttt{INV}</em>{L1} \texttt{DV} { \texttt{LK x} }</td>
</tr>
<tr>
<td>\texttt{r=fetch_inc(x)}</td>
<td>\texttt{INC}_{L1} \texttt{r x}</td>
<td>\texttt{FLU}<em>{L1} \texttt{WG} \texttt{INC}</em>{L1} \texttt{WG} \texttt{INC}_{L2} \texttt{r x}</td>
<td>\texttt{FLU}<em>{L1} \texttt{DV} \texttt{INV}</em>{L1} \texttt{WG} \texttt{INC}<em>{L2} \texttt{r x} \texttt{INV}</em>{L1} \texttt{DV} { \texttt{LK x} \texttt{LK}_{rmw} }</td>
</tr>
</tbody>
</table>
## Original scheme

<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_L1 WG LD r x</td>
<td>FLU_L1 DV INV_L1 WG LD r x } LK x</td>
</tr>
<tr>
<td>store(x,r)</td>
<td>ST r x</td>
<td>FLU_L1 WG ST r x</td>
<td>FLU_L1 WG ST r x } LK x</td>
</tr>
<tr>
<td>r=fetch_inc(x)</td>
<td>INC_L1 r x</td>
<td>FLU_L1 WG INV_L1 WG INC_L2 r x</td>
<td>FLU_L1 DV INV_L1 WG INC_L1 DV } LK x LK_rmw</td>
</tr>
<tr>
<td>r=load(x)</td>
<td>LD r x</td>
<td>INV\textsubscript{L1} WG LD r x</td>
<td>FLU\textsubscript{L1} DV { LK r x }</td>
</tr>
<tr>
<td>store(x,r)</td>
<td>ST r x</td>
<td>FLU\textsubscript{L1} WG ST r x</td>
<td>FLU\textsubscript{L1} WG { LK r x }</td>
</tr>
<tr>
<td>r=fetch_inc(x)</td>
<td>INC\textsubscript{L1} r x</td>
<td>FLU\textsubscript{L1} WG INV\textsubscript{L1} WG INC\textsubscript{L2} r x</td>
<td>FLU\textsubscript{L1} DV { LK_{rmw} }</td>
</tr>
</tbody>
</table>

Original scheme

message-passing fails
## Original scheme

<table>
<thead>
<tr>
<th>Operation</th>
<th>DV (not remote)</th>
<th>DV (remote)</th>
</tr>
</thead>
</table>
| \( r = \text{load}(x) \) | \( \text{LD} \ r \ x \) | \( \text{INV}_{L1} \ \text{WG} \ \text{LD} \ r \ x \) | \( \text{FLU}_{L1} \ \text{DV} \)  
\( \text{INV}_{L1} \ \text{WG} \ \text{LD} \ r \ x \)  
\( \text{FLU} \)  
\( \text{L1} \)  
\( \text{LK} x \) |
| \( \text{store}(x,r) \) | \( \text{ST} \ r \ x \) | \( \text{FLU}_{L1} \ \text{WG} \ \text{ST} \ r \ x \) | \( \text{FLU}_{L1} \ \text{WG} \ \text{ST} \ r \ x \)  
\( \text{INV}_{L1} \ \text{DV} \)  
\( \text{LK} x \) |
| \( r = \text{fetch}\_\text{inc}(x) \) | \( \text{INC}_{L1} \ r \ x \) | \( \text{FLU}_{L1} \ \text{WG} \ \text{INC}_{L1} \ \text{WG} \ \text{INC}_{L2} \ r \ x \) | \( \text{FLU}_{L1} \ \text{DV} \)  
\( \text{INV}_{L1} \ \text{WG} \ \text{INC}_{L2} \ r \ x \)  
\( \text{INV}_{L1} \ \text{DV} \)  
\( \text{LK}_{\text{rmw}} x \) |

- **message-passing fails**: \( \text{INV}_{L1} \ \text{WG} \ \text{LD} \ r \ x \)  
\( \text{FLU} \)  
\( \text{L1} \)  
\( \text{LK} x \)
- **RMW atomicity fails**: \( \text{FLU}_{L1} \ \text{DV} \)  
\( \text{INV}_{L1} \ \text{WG} \ \text{INC}_{L2} \ r \ x \)  
\( \text{INV}_{L1} \ \text{DV} \)  
\( \text{LK}_{\text{rmw}} x \)
message-passing fails

RMW atomicity fails

unnecessary cacheline stalling

Original scheme

<table>
<thead>
<tr>
<th></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\text{L1} WG LD r x</td>
</tr>
<tr>
<td>store(x,r)</td>
<td>ST r x</td>
<td>FLU\text{L1} WG r x</td>
</tr>
<tr>
<td>r=fetch_inc(x)</td>
<td>INC\text{L1} r x</td>
<td>FLU\text{L1} WG INV\text{L1} WG INC\text{L2} r x</td>
</tr>
</tbody>
</table>

RMW atomicity fails

message-passing fails

unnecessary cacheline stalling
Revised scheme

<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>LD r x</td>
<td>LD r x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INV$_{L1}$ WG</td>
<td>FLU$_{L1}$ DV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>INV$_{L1}$ WG</td>
</tr>
<tr>
<td>store(x,r)</td>
<td>ST r x</td>
<td>FLU$_{L1}$ WG</td>
<td>FLU$_{L1}$ WG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ST r x</td>
<td>INV$_{L1}$ DV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>INV$_{L1}$ WG</td>
</tr>
<tr>
<td>r=fetch_inc(x)</td>
<td>INC$_{L1}$ r x</td>
<td>FLU$_{L1}$ WG</td>
<td>FLU$_{L1}$ WG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INC$_{L2}$ r x</td>
<td>INV$_{L1}$ DV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>INV$_{L1}$ WG</td>
</tr>
</tbody>
</table>
## Original scheme

<table>
<thead>
<tr>
<th></th>
<th>na or WG</th>
<th>DV (not remote)</th>
<th>DV (remote)</th>
</tr>
</thead>
</table>
| \( r = \text{load}(x) \) | \( \text{LD } r \ x \) | \( \text{INV}_{L1} \ W G \)
\( \text{LD } r \ x \) | \( \text{FLU}_{L1} \ D V \)
\( \text{INV}_{L1} \ W G \)
\( \text{LD } r \ x \) \} \( \text{LK} \ x \) |
| \( \text{store}(x, r) \) | \( \text{ST } r \ x \) | \( \text{FLU}_{L1} \ W G \)
\( \text{ST } r \ x \) | \( \text{FLU}_{L1} \ W G \)
\( \text{ST } r \ x \) \} \( \text{INV}_{L1} \ D V \) \} \( \text{LK} \ x \) |
| \( r = \text{fetch}\_\text{inc}(x) \) | \( \text{INC}_{L1} \ r \ x \) | \( \text{FLU}_{L1} \ W G \)
\( \text{INV}_{L1} \ W G \)
\( \text{INC}_{L2} \ r \ x \) | \( \text{FLU}_{L1} \ D V \)
\( \text{INV}_{L1} \ W G \)
\( \text{INC}_{L2} \ r \ x \) \} \( \text{LK} \ x \) \} \( \text{LK}_{rmw} \) \}
Proof of correctness

• Theorem stated in Isabelle, proved by hand.
Summary

• **Remote-scope promotion** is a GPU programming extension from **AMD** for efficient **work-stealing**

• We **formalised** the design (at SW and HW level). This led to a **corrected** and **improved** implementation.

• Formalise **early** in the design process!