TOWARDS UNIFIED ANALYSIS OF GPU CONSISTENCY

Haining Tong[1], *Natalia Gavrilenko*[2], *Hernán Ponce de León*[2],*and* ***Keijo Heljanko***[1, 3]

[1]University of Helsinki [2]Huawei Dresden Research Center

[3]Helsinki Institute for Information Technology

CPU MEMORY CONSISTENCY MODELS

- Memory consistency models document the consistency guarantees to the programmer
- The traditional memory model is **Sequential Consistency**: The result of any execution is the same as though the memory accesses executed by each processor were kept in order and the accesses among different processors were arbitrarily interleaved.
- Sequential consistency allows for simple reasoning for concurrent programming
- For performance reasons: **No commerical microprocessor or widely used programming language implements sequential consistency!**

EXAMPLE CONCURRENT X86 ASSEMBLY

Consider the following x86 assembly program with initial shared memory values x=0 and y=0:

Thread 0	Thread 1
MOV [x] <- 1 MOV EAX <- [y]	MOV [y] <- 1 MOV EBX <- [x]

What are the possible final values for registers EAX and EBX?

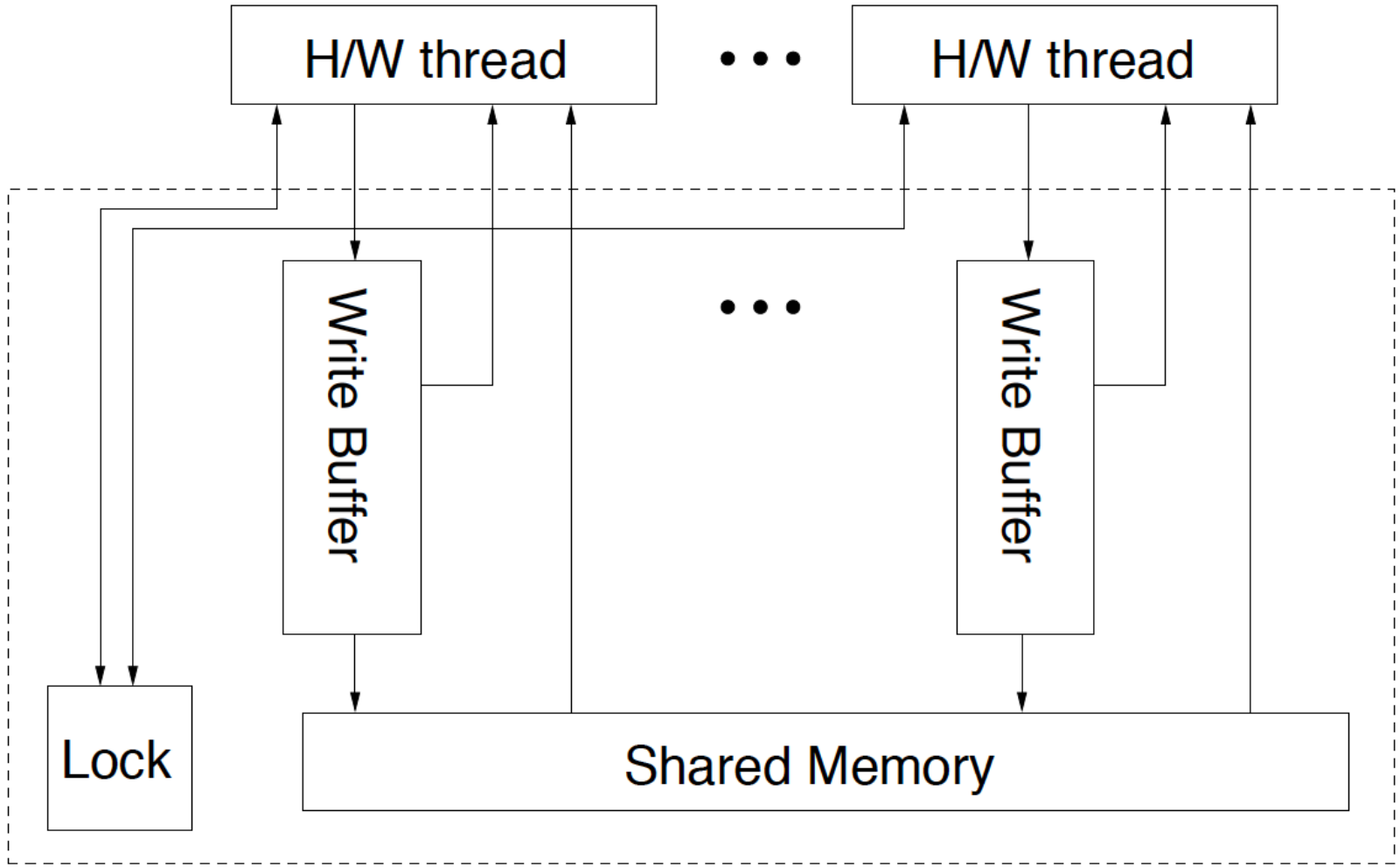
EXAMPLE CONCURRENT X86 ASSEMBLY

What are the possible final values of EAX **and** EBX**?**

- EAX = 1, EBX = 1
- EAX = 0, EBX = 1
- EAX = 1, EBX = 0
- When we run the code we also observe EAX = 0, EBX = 0!
- This is not something that is possible in any sequentially consistent execution!
- The x86 memory model is not sequentially consisten but a weaker memory model
- For example Dekker's Mutual Exclusion algorithm does not work on x86!!!
- A formalization of the x86 memory model is called **x86-TSO**

X86-TSO ABSTRACT MACHINE MODEL

x86-TSO memory subsystem model:



Write Buffers are FIFOs, writes commit in order (TSO = total store order)

X86-TSO MEMORY MODEL

References:

MPRI course “Weak memory concurrency”:

https://fzn.fr/teaching/mpri/2018/index.html

Peter Sewell, Susmit Sarkar, Scott Owens, Francesco Zappa Nardelli, Magnus O. Myreen: **x86-TSO: A rigorous and usable programmer’s model for x86 multiprocessors**. Commun. ACM 53(7): 89-97 (2010)

X86-TSO MEMORY MODEL IN CAT

```
1   "X86 TSO"
2
3   include "cos.cat"
4   include "x86fences.cat"
5   include "filters.cat"
6
7   (* All communication relations *)
8   let com = (co | fr | rf)
9
10  (* Uniproc *)
11  acyclic (po-loc | com) as uniproc
12
13  (* Atomic *)
14  empty rmw & (fre;coe)
15  let implied = po & W*R & ((M * A) | (A * M))
16
17  (* Communication relations for TSO *)
18  let com-tso = (co | fr | rfe)
19
20  (* Program order for TSO *)
21  let po-tso = ((po & ((R*M) | (W*W))) | mfence)
22
23  (* TSO global-happens-before *)
24  let ghb-tso = po-tso | com-tso | implied
25
26  acyclic ghb-tso as tso
```

- We have created tool Dartagnan to run programs under different memory models specified in CAT

- Natalia Gavrilenko, Hernán Ponce de León, Florian Furbach, Keijo Heljanko, Roland Meyer: BMC for Weak Memory Models: Relation Analysis for Compact SMT Encodings. CAV (1) 2019: 355-365

GRAMMAR OF CAT (SUBSET)

$$
\begin{aligned}
\langle MCM \rangle &::= \langle assert \rangle \mid \langle rel \rangle \mid \langle MCM \rangle \wedge \langle MCM \rangle \\
\langle assert \rangle &::= acyclic(\langle r \rangle) \mid irreflexive(\langle r \rangle) \mid empty(\langle r \rangle) \\
\langle r \rangle &::= \langle b \rangle \mid \langle r \rangle \cup \langle r \rangle \mid \langle r \rangle \cap \langle r \rangle \mid \langle r \rangle \setminus \langle r \rangle \\
&\quad \mid \langle r \rangle^{-1} \mid \langle r \rangle^{+} \mid \langle r \rangle^{*} \mid \langle r \rangle ; \langle r \rangle \\
\langle b \rangle &::= po \mid rf \mid co \mid ad \mid dd \mid cd \mid sthd \mid sloc \\
&\quad \mid mfence \mid sync \mid lwsync \mid isync \mid isb \mid ish \\
&\quad \mid id(\langle set \rangle) \mid \langle set \rangle \times \langle set \rangle \mid \langle name \rangle \\
\langle set \rangle &::= \mathbb{E} \mid \mathbb{W} \mid \mathbb{R} \\
\langle rel \rangle &::= \langle name \rangle := \langle r \rangle
\end{aligned}
$$

ENCODING BUG FINDING INTO SMT

We create the SAT modulo theories formula:

$$\varphi_{acyc(prog)} \wedge \varphi_{model} \wedge \varphi_{assert}$$

The formula is satisfied if:

- There is an execution of acyclic unwinding of the progam
- Satisfying the memory model constraints
- That violates the assertion

MEMORY MODELS FOR GPUS

Inambiguous description of weak concurrency guarantees

Describe CPU hardware, programming languages, and since recently also GPU APIs

Formal GPU memory models:

- PTXv6.0 [1] for CUDA
- PTXv7.5 [2] for CUDA
- Vulkan [3]
- OpenCL [4]
- …

Tools:
Formal models can be used by analysis tools: Herdtools7, GPUVerify, Alloy-based Tools...

Problem:
Existing verification tools for formal models are impractical for real GPU code

Our work:
A practical and scalable verification tool for weak concurrency in GPUs

[1] Daniel Lustig, Sameer Sahasrabuddhe, and Olivier Giroux. A formal analysis of the NVIDIA PTX memory consistency model. ASPLOS 2019
[2] Daniel Lustig, Simon Cooksey, and Olivier Giroux. Mixed-proxy extensions for the NVIDIA PTX memory consistency model: Industrial product. ISCA '22
[3] https://github.com/KhronosGroup/Vulkan-MemoryModel
[4] Batty, Mark and Donaldson, Alastair F. and Wickerson, John. Overhauling SC atomics in C11 and OpenCL. POPL'16

PTX 7.5 MEMORY MODEL IN CAT

```
PTX
(* NVIDIA. Parallel Thread Execution ISA Version 7.5 *)

// Base relations:
// vloc: virtual address mapping to the same generic address
// sr: same-scope
// scta: same-cta
// syncbar: synchronize by barriers with same logical barrier resource (barID)
// sync_fence: synchronize by morally strong fence.sc

// Scope Tags:
// CTA: Cooperative thread array, is an array of threads that execute a kernel concurrently or in parallel.
//      The CTA scope is the set of all threads executing in the same CTA as the current thread.
// GPU: Graphics processing unit. The GPU scope is the set of all threads executing in the same cluster as the current thread.
// SYS: System. The SYS scope is the set of all threads in the current program.

// Proxy Tags:
// GEN: Generic memory space, this is the memory space that proxy-less PTX memory model(ptx-v6.0) applies to.
// SUR: Surface memory, used by graphics workloads
// TEX: Texture memory, used by graphics workloads
// CON: Constant memory, is a small, distinct address space reserved to hold fixed values.

// Comparing to the ptx-v7.5 model written in Alloy (https://github.com/NVlabs/mixedproxy)
// The ptx-v7.5 alloy model uses this Causality-axiom "irreflexive ((rf | co | fr)?; cause)".
// The ptx-v6.0 alloy model uses "irreflexive ((rf | fr)?; cause)".
// The ASPLOS'19 paper and the documentation refer to "rf | fr" (non-optional).
// We follow the later.

(********************)
(*   Auxiliaries    *)
(********************)

let sync_barrier = syncbar & scta

// Explicitly add transitivity for coherence since the co in PTX is not total
// and recompute fr based on the new co
let co = co+
let fr = rf^-1; co

(* Events *)
let strong-write = W & (RLX | REL)
let strong-read = R & (RLX | ACQ)
let strong-m = strong-write | strong-read
let strong-operation = strong-m | F

(* Proxy *)
let same-proxy = GEN * GEN | SUR * SUR | TEX * TEX | CON * CON
let po-vloc = po & vloc

(* Relations *)
// The operations are related in program order, or each operation is strong and
// specifies a scope that includes the thread executing the other operation.
let ms1 = (po | po^-1) | ([strong-operation]; sr; [strong-operation])
// Both operations are performed via the same proxy.
let ms2 = same-proxy
// If both are memory operations, then they overlap completely
let ms3 = ((M * M) & vloc) | ((_ * _) \ (M * M))
let morally-strong = (ms1 & ms2 & ms3) \ id

// This is probably equivalent to
// let rec observation = (morally-strong & rf) | (observation; rmw; observation)
// We opt to avoid the recursion
let observation = (morally-strong & rf) | rmw

let release-pattern = ([W & REL]; po-vloc?; [strong-write]) | ([F & ACQ_REL]; po; [strong-write])

let acquire-pattern = ([strong-read]; po-vloc?; [R & ACQ]) | ([strong-read]; po; [F & ACQ_REL])

let sync = morally-strong & (release-pattern; observation; acquire-pattern)
let cause-base = (po?; ((sync | sync_fence | sync_barrier); po?)+) | po

(********************)

(* Proxy-aware causality ordering *)
let proxy-fence-ops = [F]; (same-proxy & scta); [M]
let proxy-preserved-cause-base
                = ([GEN]; (vloc & cause-base); [GEN])
                | ([M]; (same-proxy & scta & vloc & cause-base); [M])
                | vloc & (cause-base & (proxy-fence-ops^-1); cause-base; [GEN])
                | vloc & ([GEN]; cause-base; cause-base & proxy-fence-ops)
                | vloc & (cause-base & (proxy-fence-ops^-1); cause-base; cause-base & proxy-fence-ops)
                | loc & ([GEN]; cause-base; [F & ALIAS]; cause-base; [GEN])
                | loc & (cause-base & (proxy-fence-ops^-1); cause-base; [F & ALIAS]; cause-base; [GEN])
                | loc & ([GEN]; cause-base; [F & ALIAS]; cause-base; cause-base & proxy-fence-ops)
                | loc & (cause-base & (proxy-fence-ops^-1); cause-base; [F & ALIAS]; cause-base; cause-base & proxy-fence-ops)
let cause = observation?; proxy-preserved-cause-base

(*******************************)
(*  PTX Memory Model Axioms    *)
(*******************************)

(* Axiom Coherence *)
empty (([W]; cause; [W]) & loc) \ co) as axiom-Coherence
// Make sure that all morally strong same-loc writes are related by co in either direction
empty (([W]; morally-strong; [W]) & loc) \ (co | co^-1) as axiom-Coherence2

(* Axiom FenceSC *)
// This is equivalent to: irreflexive (sync_fence ; cause) as axiom-FenceSC
empty (([F & SC]; cause; [F & SC]) \ sync_fence) as axiom-FenceSC

(* Axiom Atomicity *)
empty (((morally-strong & fr); (morally-strong & co)) & rmw) as axiom-Atomicity

(* Axiom No-Thin-Air *)
let dep = addr | data | ctrl
acyclic (rf | dep) as axiom-NoThinAir

(* Axiom Causality *)
irreflexive ((rf | fr); cause) as axiom-Causality
```

LIMITATIONS OF THE EXISTING TOOLS

- **Herdtools7 [1]**
  - GPU support existed for OpenCL and an old PTX models, but has been deprecated
- **GPUverify [2]**
  - Doesn’t support weak memory models
- **Alloy-based tools (PTX [3] and Vulkan [4])**
  - Support fine-grained weak concurrency, but does not scale beyond basic litmus tests

[1] https://github.com/herd/herdtools7

[2] https://github.com/mc-imperial/gpuverify

[3] https://github.com/NVlabs/mixedproxy

[4] https://github.com/KhronosGroup/Vulkan-MemoryModel

PRACTICAL EXAMPLE: XF-BARRIER

```
__kernel void xf_barrier(global atomic_uint* flag, global uint* in, global uint* out) { \\ [1]

        in[get_global_id(0)] = 1;

        if (get_group_id(0) == 0) {
            if (get_local_id(0) < get_num_groups(0)) {
                while (atomic_load_explicit(&flag[get_local_id(0)], memory_order_acquire) == 0);
            }
            barrier(CLK_GLOBAL_MEM_FENCE);
            if (get_local_id(0) < get_num_groups(0)) {
                atomic_store_explicit(&flag[get_local_id(0)], 0, memory_order_release);
            }
        } else {
            barrier(CLK_GLOBAL_MEM_FENCE);
            if (get_local_id(0) == 0) {
                atomic_store_explicit(&flag[get_group_id(0) - 1], 1, memory_order_release);
                while (atomic_load_explicit(&flag[get_group_id(0) - 1], memory_order_acquire) == 1);
            }
            barrier(CLK_GLOBAL_MEM_FENCE);
        }

        for (unsigned int i = 0; i < get_global_size(0); i++) {
            out[get_global_id(0)] += in[i];
        }
}
```

[1] Shucai Xiao and Wu-chun Feng. Inter-block GPU communication via fast barrier synchronization. IPDPS 2010

PRACTICAL EXAMPLE: XF-BARRIER

```
__kernel void xf_barrier(global atomic_uint* flag, global uint* in, global uint* out) { \\ [1]

        in[get_global_id(0)] = 1;

        if (get_group_id(0) == 0) {
            if (get_local_id(0) < get_num_groups(0)) {
                while (atomic_load_explicit(&flag[get_local_id(0)], memory_order_acquire) == 0);
            }
            barrier(CLK_GLOBAL_MEM_FENCE);
            if (get_local_id(0) < get_num_groups(0)) {
                atomic_store_explicit(&flag[get_local_id(0)], 0, memory_order_release);
            }
        } else {
            barrier(CLK_GLOBAL_MEM_FENCE);
            if (get_local_id(0) == 0) {
                atomic_store_explicit(&flag[get_group_id(0) - 1], 1, memory_order_release);
                while (atomic_load_explicit(&flag[get_group_id(0) - 1], memory_order_acquire) == 1);
            }
            barrier(CLK_GLOBAL_MEM_FENCE);
        }

        for (unsigned int i = 0; i < get_global_size(0); i++) {
            out[get_global_id(0)] += in[i];
        }
}
```

Leader workgroup (one)
Responsible for the synchronization of the other workgroups

Follower workgroups (multiple)

[1] Shucai Xiao and Wu-chun Feng. Inter-block GPU communication via fast barrier synchronization. IPDPS 2010

ALLOY-BASED TOOLS

- **Only support a custom pseudo-assembly as input**
  - Manually rewrite source code to match the expected format

```
__kernel void xf_barrier(global atomic_uint *flag, global uint* in, global uint* out) {

        in[get_global_id(0)] = 1;

        if (get_group_id(0) == 0) {
            if (get_local_id(0) < get_num_groups(0)) {
                while (atomic_load_explicit(&flag[get_local_id(0)], memory_order_acquire) == 0);
            }
            barrier(CLK_GLOBAL_MEM_FENCE);
            if (get_local_id(0) < get_num_groups(0)) {
                atomic_store_explicit(&flag[get_local_id(0)], 0, memory_order_release);
            }
        } else {
            barrier(CLK_GLOBAL_MEM_FENCE);
            if (get_local_id(0) == 0) {
                atomic_store_explicit(&flag[get_group_id(0) - 1], 1, memory_order_release);
                while (atomic_load_explicit(&flag[get_group_id(0) - 1], memory_order_acquire) == 1);
            }
            barrier(CLK_GLOBAL_MEM_FENCE);
        }

        for (unsigned int i = 0; i < get_global_size(0); i++) {
            out[get_global_id(0)] += in[i];
        }
}
```

xf-barrier.litmus

```
NEWWG
NEWSG
NEWTHREAD
st.av.scopedev.sc0 x0 = 1
// LC00:
ld.atom.acq.scopedev.sc0.semsc0 f1 = 0
// LC01:
cbar.acq.rel.scopewg.semsc0 00
st.atom.rel.scopedev.sc0.semsc0 f1 = 0
ld.vis.scopedev.sc0 x0 = 0
ld.vis.scopedev.sc0 x1 = 0
ld.vis.scopedev.sc0 x2 = 0
ld.vis.scopedev.sc0 x3 = 0
ld.vis.scopedev.sc0 x4 = 0
ld.vis.scopedev.sc0 x5 = 0
NEWTHREAD
st.av.scopedev.sc0 x1 = 1
// LC10:
ld.atom.acq.scopedev.sc0.semsc0 f2 = 0
// LC11:
cbar.acq.rel.scopewg.semsc0 00
st.atom.rel.scopedev.sc0.semsc0 f2 = 0
ld.vis.scopedev.sc0 x0 = 0
ld.vis.scopedev.sc0 x1 = 0
ld.vis.scopedev.sc0 x2 = 0
ld.vis.scopedev.sc0 x3 = 0
ld.vis.scopedev.sc0 x4 = 0
ld.vis.scopedev.sc0 x5 = 0
NEWWG
NEWSG
NEWTHREAD
st.av.scopedev.sc0 x2 = 1
cbar.acq.rel.scopewg.semsc0 10
st.atom.rel.scopedev.sc0.semsc0 f1 = 1
// LC20:
ld.atom.acq.scopedev.sc0.semsc0 f1 = 1
// LC21:
cbar.acq.rel.scopewg.semsc0 11
ld.vis.scopedev.sc0 x0 = 0
ld.vis.scopedev.sc0 x1 = 0
ld.vis.scopedev.sc0 x2 = 0
ld.vis.scopedev.sc0 x3 = 0
ld.vis.scopedev.sc0 x4 = 0
ld.vis.scopedev.sc0 x5 = 0
NEWTHREAD
st.av.scopedev.sc0 x3 = 1
cbar.acq.rel.scopewg.semsc0 10
cbar.acq.rel.scopewg.semsc0 11
ld.vis.scopedev.sc0 x0 = 0
ld.vis.scopedev.sc0 x1 = 0
ld.vis.scopedev.sc0 x2 = 0
ld.vis.scopedev.sc0 x3 = 0
ld.vis.scopedev.sc0 x4 = 0
ld.vis.scopedev.sc0 x5 = 0
NEWWG
NEWSG
NEWTHREAD
st.av.scopedev.sc0 x2 = 1
cbar.acq.rel.scopewg.semsc0 20
st.atom.rel.scopedev.sc0.semsc0 f2 = 1
// LC20:
ld.atom.acq.scopedev.sc0.semsc0 f2 = 1
// LC21:
cbar.acq.rel.scopewg.semsc0 22
ld.vis.scopedev.sc0 x0 = 0
ld.vis.scopedev.sc0 x1 = 0
ld.vis.scopedev.sc0 x2 = 0
ld.vis.scopedev.sc0 x3 = 0
ld.vis.scopedev.sc0 x4 = 0
ld.vis.scopedev.sc0 x5 = 0
NEWTHREAD
st.av.scopedev.sc0 x3 = 1
cbar.acq.rel.scopewg.semsc0 20
cbar.acq.rel.scopewg.semsc0 21
ld.vis.scopedev.sc0 x0 = 0
ld.vis.scopedev.sc0 x1 = 0
ld.vis.scopedev.sc0 x2 = 0
ld.vis.scopedev.sc0 x3 = 0
ld.vis.scopedev.sc0 x4 = 0
ld.vis.scopedev.sc0 x5 = 0
SATISFIABLE consistent[X] && #dr=0
```

ALLOY-BASED TOOLS

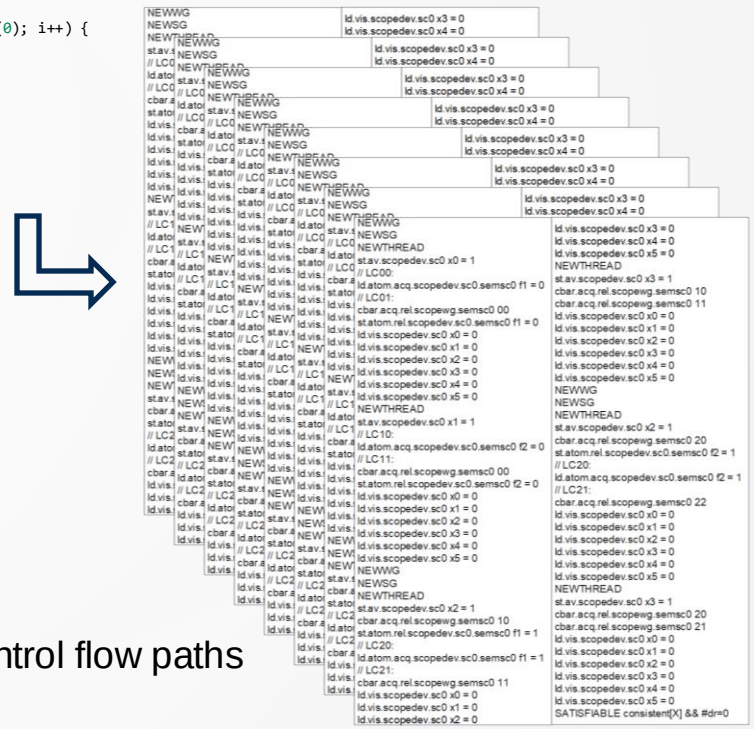
```
__kernel void xf_barrier(global atomic_uint *flag, global uint* in, global uint* out) {

        in[get_global_id(0)] = 1;

        if (get_group_id(0) == 0) {
            if (get_local_id(0) < get_num_groups(0)) {
                while (atomic_load_explicit(&flag[get_local_id(0)], memory_order_acquire) == 0);
            }
            barrier(CLK_GLOBAL_MEM_FENCE);
            if (get_local_id(0) < get_num_groups(0)) {
                atomic_store_explicit(&flag[get_local_id(0)], 0, memory_order_release);
            }
        } else {
            barrier(CLK_GLOBAL_MEM_FENCE);
            if (get_local_id(0) == 0) {
                atomic_store_explicit(&flag[get_group_id(0) - 1], 1, memory_order_release);
                while (atomic_load_explicit(&flag[get_group_id(0) - 1], memory_order_acquire) == 1);
            }
            barrier(CLK_GLOBAL_MEM_FENCE);
        }

        for (unsigned int i = 0; i < get_global_size(0); i++) {
            out[get_global_id(0)] += in[i];
        }
}
```

- **Only support a custom pseudo-assembly as input**
  - Manually rewrite source code to match the expected format
- **Don't support control flow**
  - Manually write all control flow paths



ALLOY-BASED TOOLS

- **Only support a custom pseudo-assembly as input**
  - Manually rewrite source code to match the expected format
- **Don't support control flow**
  - Manually write all control flow paths
- **Scalability is limited**
  - Extract minimal patterns from the code and verify them one-by-one

Alloy-Vulkan

Out-of-memory at about **20** instructions

(e.g., SB, MP, or LB test with 8 threads)

Alloy-PTX

Out-of-memory at about **60** instructions

(e.g., SB, MP, or LB test with 28 threads)

The XF-Barrier example with 9 threads contains **457** instructions

OUR SOLUTION

- **Control flow support**
  - No need to write all control flow paths

```
{
  locID0 = 0; locID1 = 1; locID2 = 0; locID3 = 1;
  gloID0 = 0; gloID1 = 1; gloID2 = 2; gloID3 = 3;
  groupID0 = 0; groupID1 = 0;
  numGroups = 2;
  x0=0; x1=0; x2=0; x3=0;
  f1=0;
}
P0@sg 0, wg 0, qf 0               | P1@sg 0, wg 0, qf 0               | P2@sg 0, wg 1, qf 0               | P3@sg 0, wg 1, qf 0               ;
 st.av.dv.sc0 x0, 1               | st.av.dv.sc0 x1, 1                | st.av.dv.sc0 x2, 1                | st.av.dv.sc0 x3, 1                ;
 ld.vis.dv.sc0 r13, groupID0      | ld.vis.dv.sc0 r13, groupID0       | ld.vis.dv.sc0 r13, groupID1       | ld.vis.dv.sc0 r13, groupID1       ;
 bne r13, 0, LC00                 | bne r13, 0, LC10                  | bne r13, 0, LC20                  | bne r13, 0, LC30                  ;
 ld.vis.dv.sc0 r14, locID0        | ld.vis.dv.sc0 r14, locID1         | ld.vis.dv.sc0 r14, locID2         | ld.vis.dv.sc0 r14, locID3         ;
 ld.vis.dv.sc0 r15, numGroups     | ld.vis.dv.sc0 r15, numGroups      | ld.vis.dv.sc0 r15, numGroups      | ld.vis.dv.sc0 r15, numGroups      ;
 bgt r14, r15, LC01               | bgt r14, r15, LC11                | bgt r14, r15, LC21                | bgt r14, r15, LC31                ;
 LC02:                            | LC12:                             | LC22:                             | LC32:                             ;
 ld.atom.acq.dv.sc0.semsc0 r10, f1| ld.atom.acq.dv.sc0.semsc0 r10, f2 | ld.atom.acq.dv.sc0.semsc0 r10, f1 | ld.atom.acq.dv.sc0.semsc0 r10, f2 ;
 beq r10, 0, LC02                 | beq r10, 0, LC12                  | beq r10, 0, LC22                  | beq r10, 0, LC32                  ;
 LC01:                            | LC11:                             | LC21:                             | LC31:                             ;
 cbar.acq_rel.wg.semsc0 00        | cbar.acq_rel.wg.semsc0 10         | cbar.acq_rel.wg.semsc0 20         | cbar.acq_rel.wg.semsc0 30         ;
 bgt r14, r15, LC03               | bgt r14, r15, LC13                | bgt r14, r15, LC23                | bgt r14, r15, LC33                ;
 st.atom.rel.dv.sc0.semsc0 f1, 0  | st.atom.rel.dv.sc0.semsc0 f2, 0   | st.atom.rel.dv.sc0.semsc0 f1, 0   | st.atom.rel.dv.sc0.semsc0 f2, 0   ;
 LC00:                            | LC10:                             | LC20:                             | LC30:                             ;
 cbar.acq_rel.wg.semsc0 01        | cbar.acq_rel.wg.semsc0 11         | cbar.acq_rel.wg.semsc0 21         | cbar.acq_rel.wg.semsc0 31         ;
 bne r14, 0, LC04                 | bne r14, 0, LC14                  | bne r14, 0, LC24                  | bne r14, 0, LC34                  ;
 st.atom.rel.dv.sc0.semsc0 f1, 1  | st.atom.rel.dv.sc0.semsc0 f2, 1   | st.atom.rel.dv.sc0.semsc0 f1, 1   | st.atom.rel.dv.sc0.semsc0 f2, 1   ;
 LC05:                            | LC15:                             | LC25:                             | LC35:                             ;
 ld.atom.acq.dv.sc0.semsc0 r11, f1| ld.atom.acq.dv.sc0.semsc0 r11, f2 | ld.atom.acq.dv.sc0.semsc0 r11, f1 | ld.atom.acq.dv.sc0.semsc0 r11, f2 ;
 beq r11, 1, LC05                 | beq r11, 1, LC15                  | beq r11, 1, LC25                  | beq r11, 1, LC35                  ;
 LC04:                            | LC14:                             | LC24:                             | LC34:                             ;
 cbar.acq_rel.wg.semsc0 02        | cbar.acq_rel.wg.semsc0 12         | cbar.acq_rel.wg.semsc0 22         | cbar.acq_rel.wg.semsc0 32         ;
 LC03:                            | LC13:                             | LC23:                             | LC33:                             ;
 ld.vis.dv.sc0 r0, x0             | ld.vis.dv.sc0 r0, x0              | ld.vis.dv.sc0 r0, x0              | ld.vis.dv.sc0 r0, x0              ;
 ld.vis.dv.sc0 r1, x1             | ld.vis.dv.sc0 r1, x1              | ld.vis.dv.sc0 r1, x1              | ld.vis.dv.sc0 r1, x1              ;
 ld.vis.dv.sc0 r2, x2             | ld.vis.dv.sc0 r2, x2              | ld.vis.dv.sc0 r2, x2              | ld.vis.dv.sc0 r2, x2              ;
 ld.vis.dv.sc0 r3, x3             | ld.vis.dv.sc0 r3, x3              | ld.vis.dv.sc0 r3, x3              | ld.vis.dv.sc0 r3, x3              ;

forall(P0:r0 == 1 /\ P0:r1 == 1 /\ P0:r2 == 1 /\ P0:r3 == 1 /\ P1:r0 == 1 /\ P1:r1 == 1 /\ P1:r2 == 1 /\ P1:r3 == 1
  /\ P2:r0 == 1 /\ P2:r1 == 1 /\ P2:r2 == 1 /\ P2:r3 == 1 /\ P3:r0 == 1 /\ P3:r1 == 1 /\ P3:r2 == 1 /\ P3:r3 == 1)
```

```
Total verification time: 0.821 secs
```

OUR SOLUTION

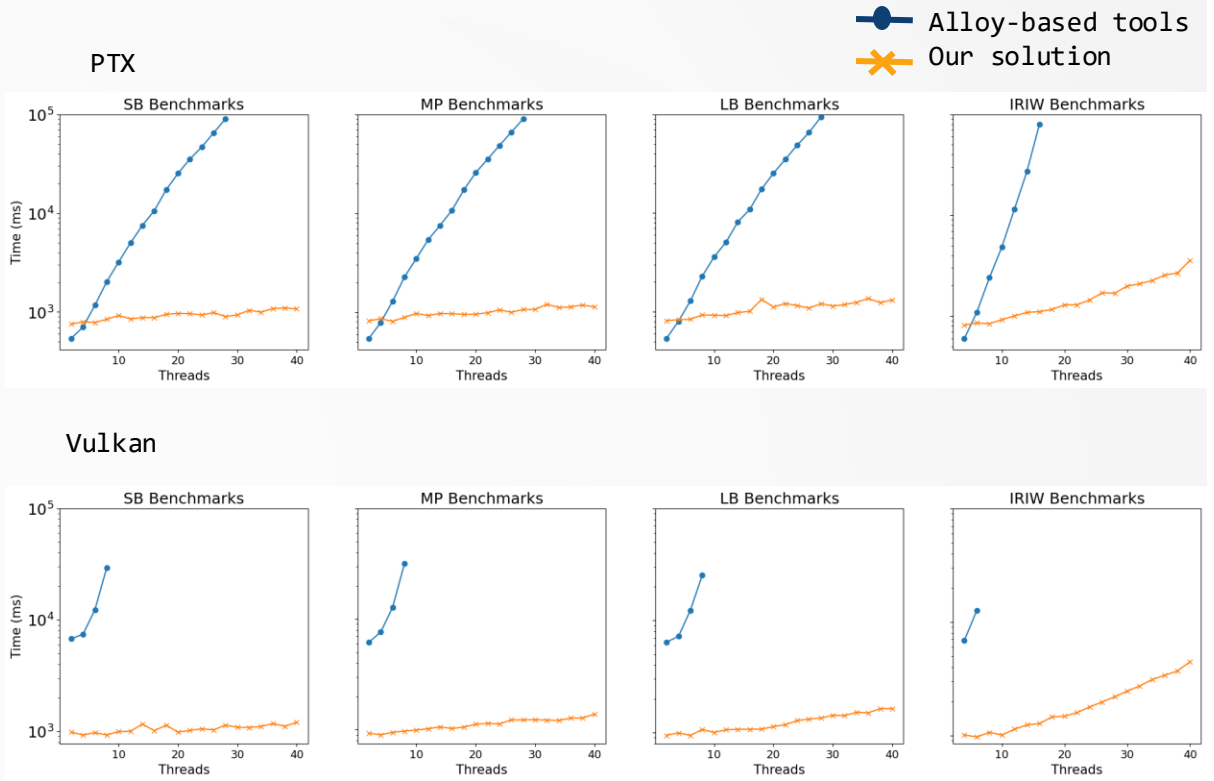
- **Control flow support**
  - No need to write all control flow paths
- **Real code (SPIR-V) support**
  - No need to re-write code in pseudo-assembly

```
> clspv kernel.cl --cl-std=CL2.0 --inline-entry-points \
  -spv-version=1.6 -o kernel.spv
> spirv-opt --upgrade-memory-model kernel.spv -o kernel.spv
> spirv-dis kernel.spv -o kernel.spv.dis
> cat annotation.txt kernel.spv.dis > out && mv out kernel.spv.dis
> java -jar dartagnan/target/dartagnan.jar cat/spirv.cat \
  --target=vulkan kernel.spv.dis
```

```
; @Input: %flag = {{0, 0, 0, 0, 0, 0, 0, 0}}
; @Input: %in = {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}
; @Input: %out = {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}
; @Output: forall (%out[0][0] == 4 and %out[0][1] == 4 and %out[0][2] == 4 and %out[0][3] == 4)
; @Config: 2, 1, 2
; SPIR-V
; Version: 1.6
; Generator: Google Clspv; 0
; Bound: 155
; Schema: 0
               OpCapability Shader
               OpCapability VulkanMemoryModel
               OpMemoryModel Logical Vulkan
       %133 = OpExtInstImport "NonSemantic.ClspvReflection.5"
               OpEntryPoint GLCompute %24 "xf_barrier" %gl_GlobalInvocationID %gl_LocalInvocationID
%gl_WorkGroupID %15 %19 %20 %21 %5
               OpSource OpenCL_C 200
               OpMemberDecorate %_struct_3 0 Offset 0
               OpMemberDecorate %_struct_3 1 Offset 16
```

OUR SOLUTION

- **Control flow support**
  - No need to write all control flow paths
- **Real code (SPIR-V) support**
  - No need to re-write code in pseudo-assembly
- **Scalability**
  - No need to extract patterns out of a kernel



Benchmark	Grid	|T|	|E|	Correct	Time (ms)
xf-barrier	3.3	9	457	✓	13598
xf-barrier-acq2rx-1	2.2	4	192	✗	2803
xf-barrier-acq2rx-2	2.2	4	192	✗	2792
xf-barrier-rel2rx-1	2.2	4	192	✗	2661
xf-barrier-rel2rx-2	2.2	4	192	✗	2777

OUR SOLUTION

Verification properties

- **Safety**
  - Check final state assertions
- **Data Races**
  - Detect model specified relations
- **Termination**
  - Detect potential deadlocks

OUR SOLUTION

Dartagnan
https://github.com/hernanponcedeleon/Dat3M

- **SMT-based**
  - Encode program semantics under a formal model as an SMT formula to detect property violations [1].
- **Static analysis**
  - Techniques [2,3] designed for CPU models are applied to GPUs to reduce SMT encoding size and improve performance.
- **More details in the paper**

[1] Hernán Ponce de León, Florian Furbach, Keijo Heljanko, Roland Meyer: Portability Analysis for Weak Memory Models. PORTHOS: One Tool for all Models. SAS 2017

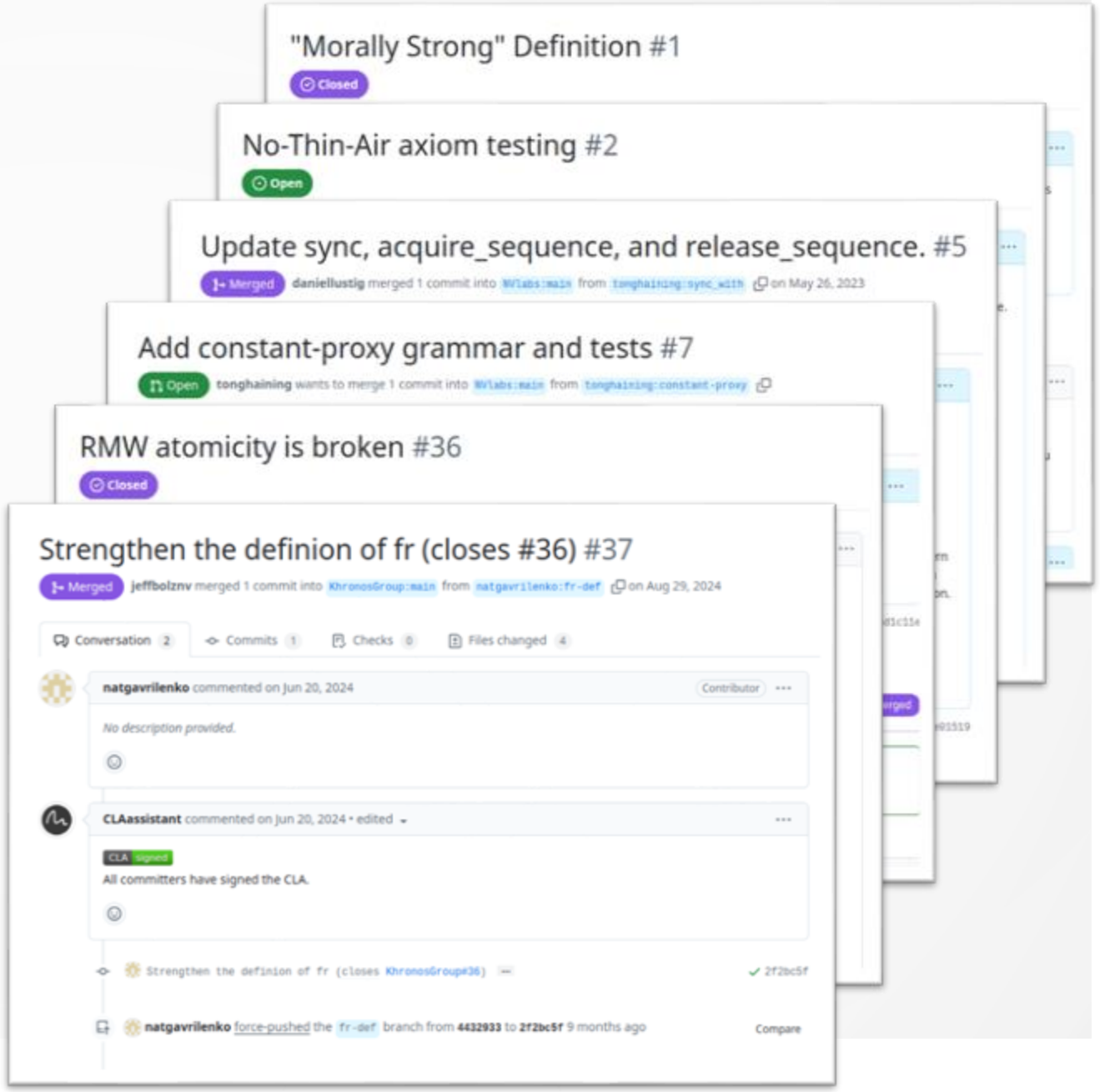
[2] Natalia Gavrilenko, Hernán Ponce de León, Florian Furbach, Keijo Heljanko, Roland Meyer: BMC for Weak Memory Models: Relation Analysis for Compact SMT Encodings. CAV 2019.

[3] Thomas Haas, René Maseli, Roland Meyer, Hernán Ponce de León: Static Analysis of Memory Models for SMT Encodings. OOPSLA 2023.

IMPACT

Our benchmarking revealed several issues in existing tools and formal models:

- https://github.com/NVlabs/mixedproxy
- https://github.com/KhronosGroup/Vulkan-MemoryModel



VULKAN MEMORY MODEL BUG FOUND!

BROKEN ATOMICITY

```
 P0@sg 0, wg 0, qf 0           | P1@sg 1, wg 0, qf 0           ;
 st.av.dv.sc0 x, 1             |                               ;
 cbar.acq_rel.dv.semsc0 0      | cbar.acq_rel.dv.semsc0 0      ;
 rmw.atom.dv.sc0.add r0, x, 1 | rmw.atom.dv.sc0.add r0, x, 1 ;
exists
(P0:r0 == 1 /\ P1:r0 == 1)
```

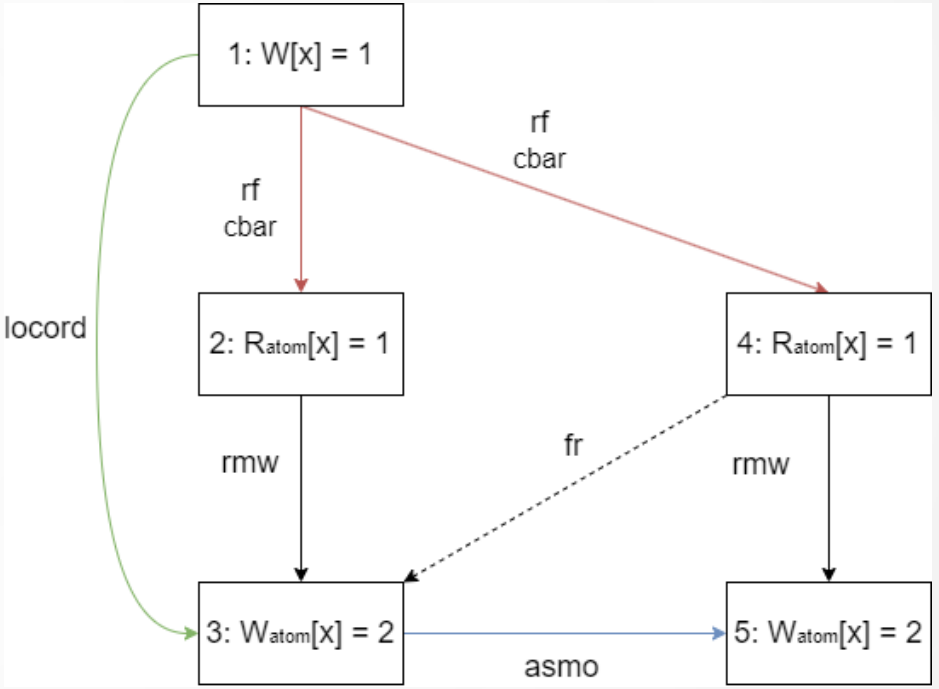
https://github.com/KhronosGroup/Vulkan-MemoryModel/issues/36

VULKAN MEMORY MODEL BUG FIXED!

ACCEPTED VULKAN STANDARD FIX

Constraint: empty rmw & (fre; (asmo & ext))

- Currently, the model defines fr as:
  let fr = (rf^-1; asmo) | (([IW]; rf)^-1; ((loc;[W]) \ id))
- Change fr definition to add locord:
  let fr = (rf^-1; ([W]; locord; [W]))
  | (rf^-1; asmo) | (([IW]; rf)^-1; ((loc;[W]) \ id))



https://github.com/KhronosGroup/Vulkan-MemoryModel/issues/36

TEMPTING OPTIMIZATION

Avoiding reported over-synchronization improves the performance of several GPU applications by up to 55%.

"Since all fences, including the narrowest scope, ensure ordering, a wider-scoped one is unnecessary."

(global) `sum = 0, flag = 0;`

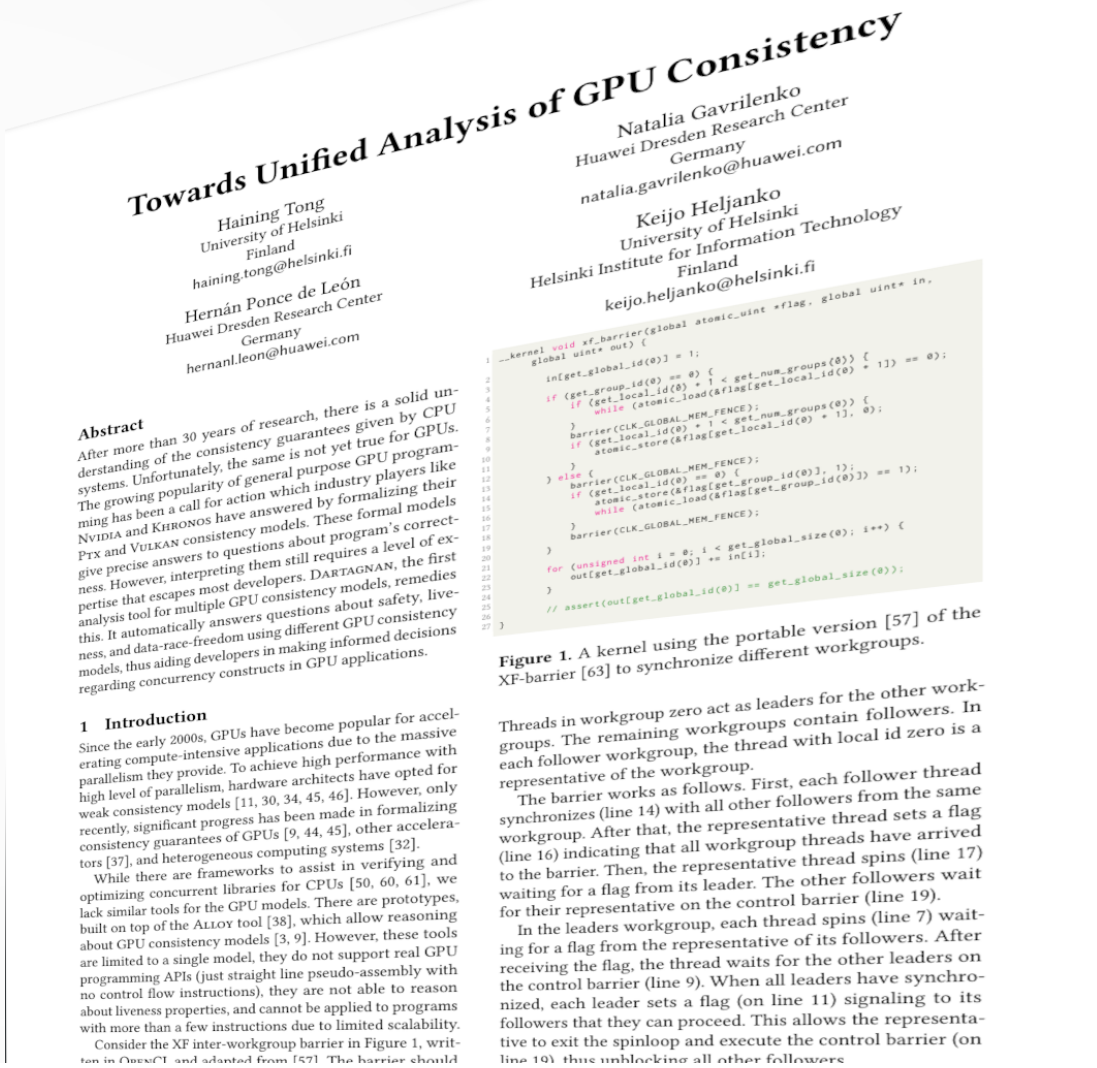
```
1   W1 st.volatile [sum], SUM;          // sum = SUM;
2   F1 membar.gl;                       // __threadfence();
3   W2 st.volatile [flag], 1;           // flag = 1;
4   ...
5   LB_1:
6   R1 ld.volatile %r1, [flag];         // while (flag == 0);
7      setp.eq %p1, %r1, 0;             //
8      @%p1 bra LB_1;                   //
9   F2 membar.cta;                      // __threadfence_block();
10  R2 ld.volatile %r2, [sum];          // aggregate += sum;
```

(a) PTX representation of program in Figure 1a.

https://www.csa.iisc.ac.in/~arkapravab/papers/MICRO24_ScopeAdvice.pdf

MORE DETAILS

- ASPLOS'24fall
- https://doi.org/10.1145/3622781.3674174



Towards Unified Analysis of GPU Consistency

Haining Tong
University of Helsinki
Finland
haining.tong@helsinki.fi

Hernán Ponce de León
Huawei Dresden Research Center
Germany
hernanl.leon@huawei.com

Natalia Gavrilenko
Huawei Dresden Research Center
Germany
natalia.gavrilenko@huawei.com

Keijo Heljanko
University of Helsinki
Helsinki Institute for Information Technology
Finland
keijo.heljanko@helsinki.fi

Abstract

After more than 30 years of research, there is a solid understanding of the consistency guarantees given by CPU systems. Unfortunately, the same is not yet true for GPUs. The growing popularity of general purpose GPU programming has been a call for action which industry players like Nvidia and Khronos have answered by formalizing their Ptx and Vulkan consistency models. These formal models give precise answers to questions about program's correctness. However, interpreting them still requires a level of expertise that escapes most developers. DartagnaN, the first analysis tool for multiple GPU consistency models, remedies this. It automatically answers questions about safety, liveness, and data-race-freedom using different GPU consistency models, thus aiding developers in making informed decisions regarding concurrency constructs in GPU applications.

1 Introduction

Since the early 2000s, GPUs have become popular for accelerating compute-intensive applications due to the massive parallelism they provide. To achieve high performance with high level of parallelism, hardware architects have opted for weak consistency models [11, 30, 34, 45, 46]. However, only recently, significant progress has been made in formalizing consistency guarantees of GPUs [9, 44, 45], other accelerators [37], and heterogeneous computing systems [32].

While there are frameworks to assist in verifying and optimizing concurrent libraries for CPUs [50, 60, 61], we lack similar tools for the GPU models. There are prototypes, built on top of the Alloy tool [38], which allow reasoning about GPU consistency models [3, 9]. However, these tools are limited to a single model, they do not support real GPU programming APIs (just straight line pseudo-assembly with no control flow instructions), they are not able to reason about liveness properties, and cannot be applied to programs with more than a few instructions due to limited scalability.

Consider the XF inter-workgroup barrier in Figure 1, written in OpenCL and adapted from [57]. The barrier should

```
1  __kernel void xf_barrier(global atomic_uint *flag, global uint* in,
2      global uint* out) {
3    in[get_global_id(0)] = 1;
4    if (get_group_id(0) == 0) {
5      if (get_local_id(0) < get_num_groups(0)) {
6        while (atomic_load(&flag[get_local_id(0) * 1]) == 0);
7      }
8      barrier(CLK_GLOBAL_MEM_FENCE);
9      if (get_local_id(0) < get_num_groups(0)) {
10       atomic_store(&flag[get_local_id(0) * 1], 0);
11     }
12   } else {
13     barrier(CLK_GLOBAL_MEM_FENCE);
14     if (get_local_id(0) == 0) {
15       atomic_store(&flag[get_group_id(0)], 1);
16       while (atomic_load(&flag[get_group_id(0)]) == 1);
17     }
18     barrier(CLK_GLOBAL_MEM_FENCE);
19   }
20   for (unsigned int i = 0; i < get_global_size(0); i++) {
21     out[get_global_id(0)] += in[i];
22   }
23   // assert(out[get_global_id(0)] == get_global_size(0));
24 }
```

Figure 1. A kernel using the portable version [57] of the XF-barrier [63] to synchronize different workgroups.

Threads in workgroup zero act as leaders for the other workgroups. The remaining workgroups contain followers. In each follower workgroup, the thread with local id zero is a representative of the workgroup.

The barrier works as follows. First, each follower thread synchronizes (line 14) with all other followers from the same workgroup. After that, the representative thread sets a flag (line 16) indicating that all workgroup threads have arrived to the barrier. Then, the representative thread spins (line 17) waiting for a flag from its leader. The other followers wait for their representative on the control barrier (line 19).

In the leaders workgroup, each thread spins (line 7) waiting for a flag from the representative of its followers. After receiving the flag, the thread waits for the other leaders on the control barrier (line 9). When all leaders have synchronized, each leader sets a flag (on line 11) signaling to its followers that they can proceed. This allows the representative to exit the spinloop and execute the control barrier (on line 19), thus unblocking all other followers