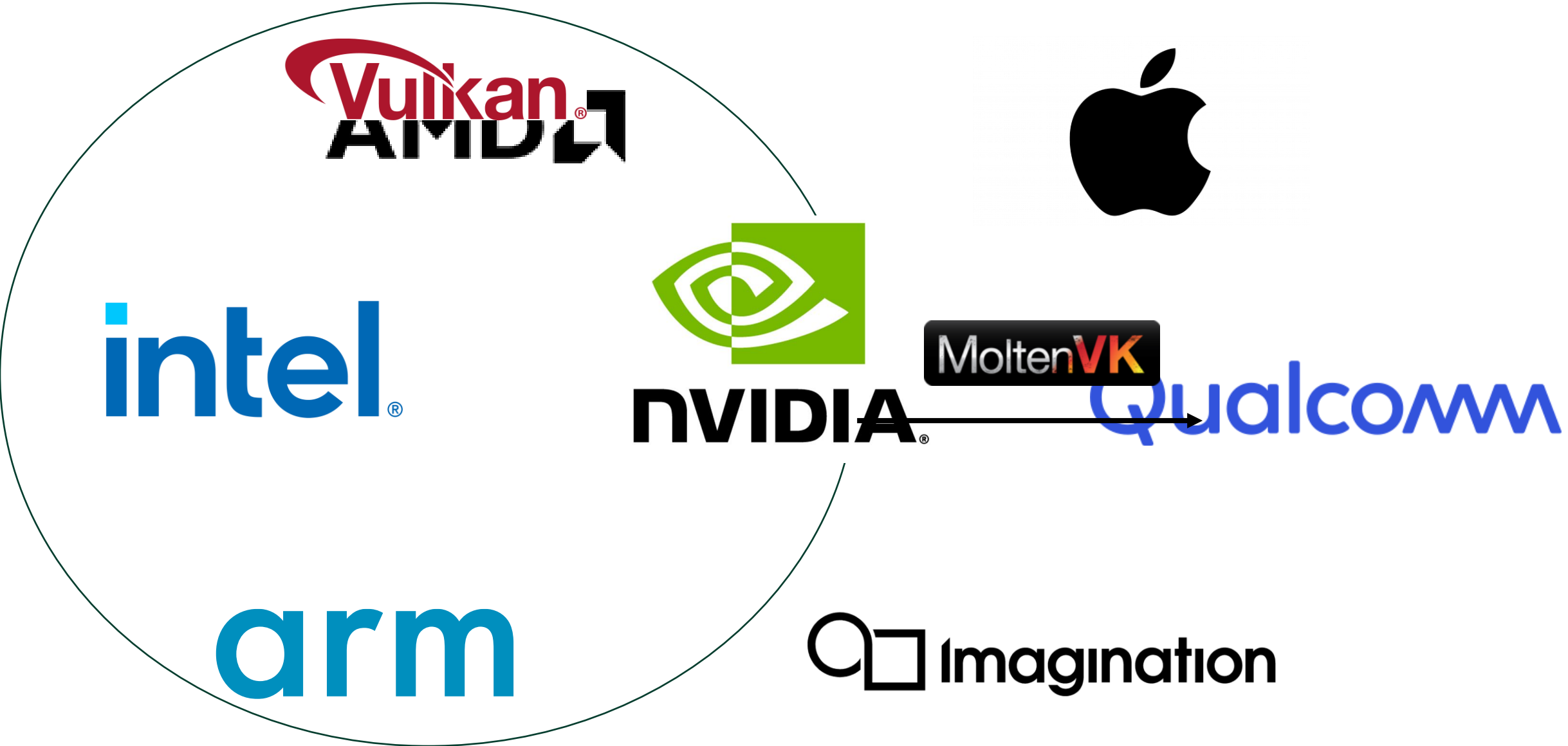
Vulkanised 2024
The 6th Vulkan Developer Conference
Sunnyvale, California | February 5-7, 2024

Testing the Vulkan Memory Model

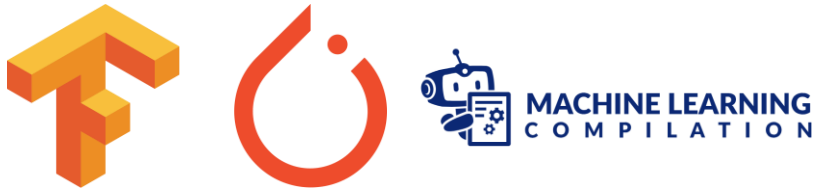
Reese Levine, PhD Candidate, UC Santa Cruz

Diversity of GPUs



GPGPU Applications

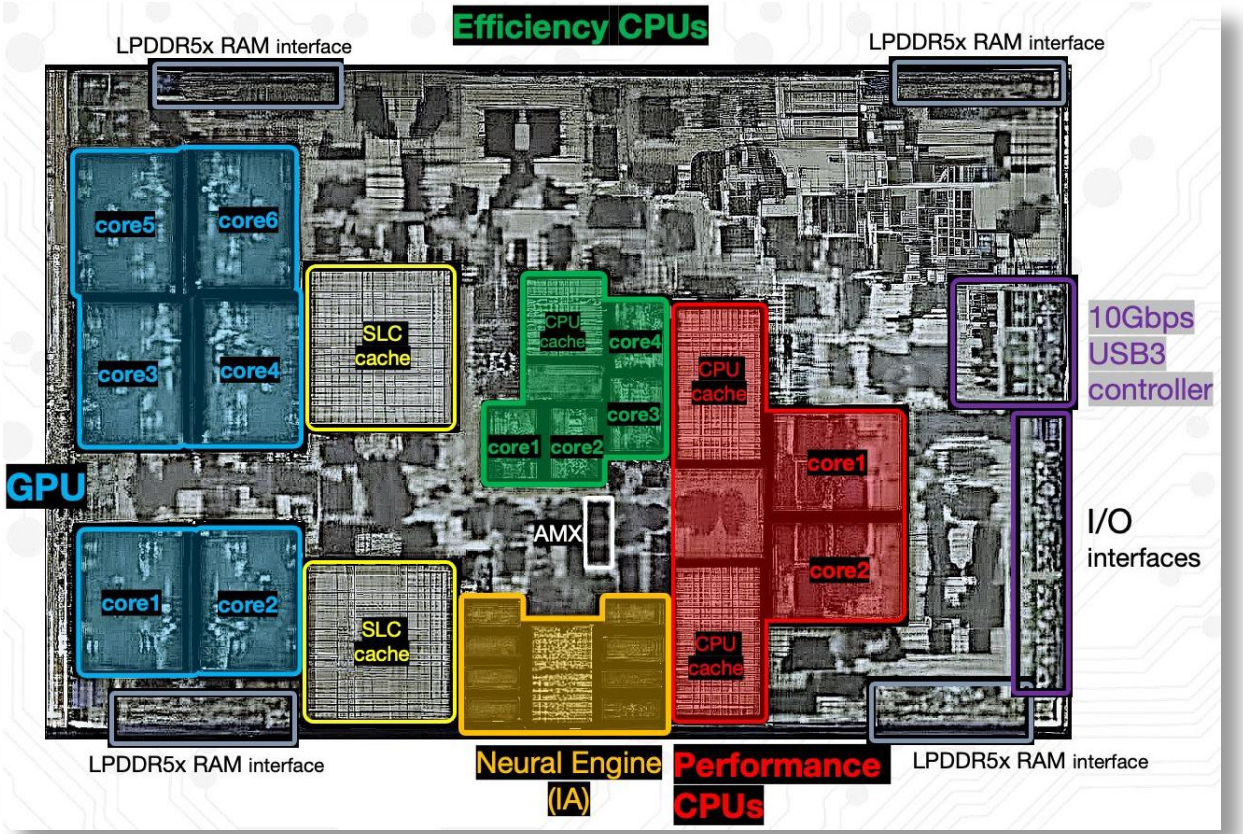
Machine Learning



High Performance Computing

NVIDIA

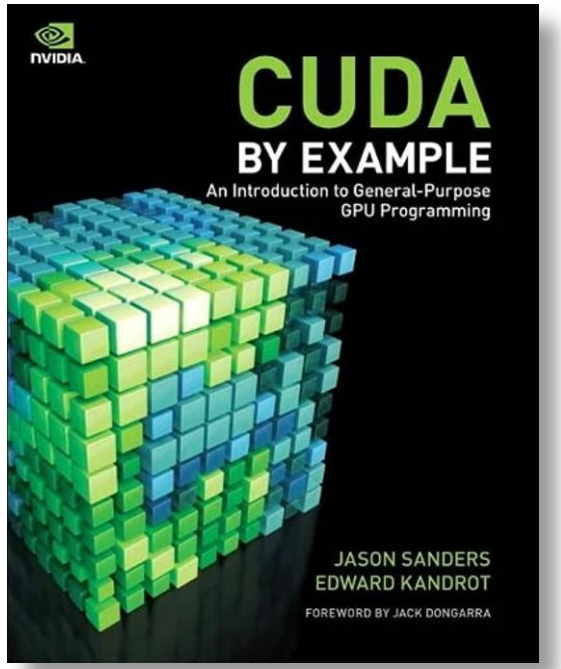
Edge Device Acceleration



A17 Pro Die, @Frederic_Orange

Fine-grained synchronization

- Mutexes
- Prefix Scans
- Concurrent Queues
- Collective Communication



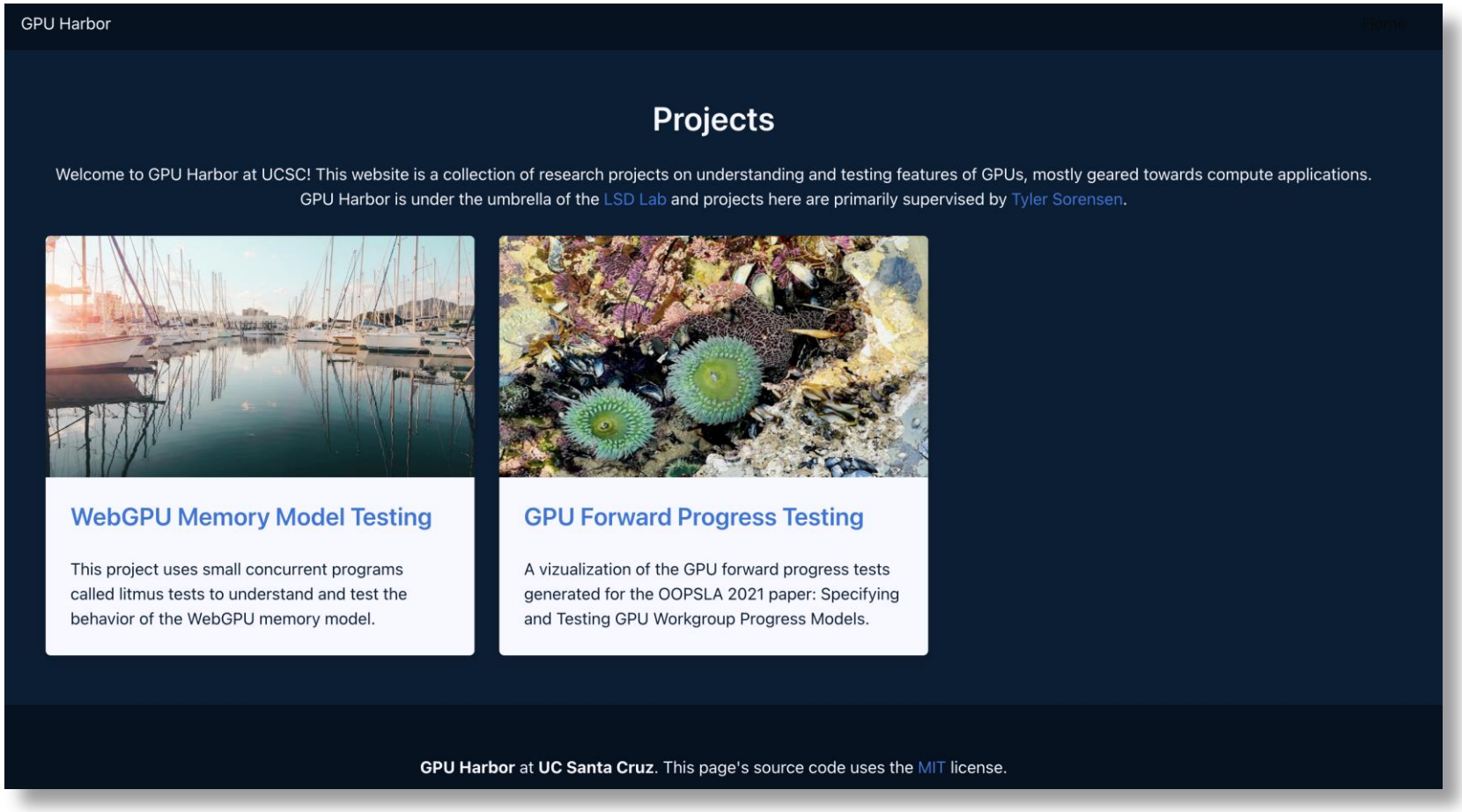
UNREAL ENGINE

NVIDIA NVLINK

CXL Compute Express Link

Testing the foundations of fine-grained synchronization

https://gpuharbor.ucsc.edu



* Chrome, laptop/Android

Memory Consistency

- **Memory Consistency Specifications (MCSs):** how we reason about programs that synchronize/access common memory locations

Initialize: `*x = 0 && *y = 0`

thread 0	thread 1
a: $W_{\{rlx\}}$ `*x = 1`	c: $R_{\{rlx\}}$ `r0 = *y`
b: $W_{\{rlx\}}$ `*y = 1`	d: $R_{\{rlx\}}$ `r1 = *x`

- ***Message Passing*** litmus test: same pattern found in mutexes (locks)

Memory Consistency

Initialize: `*x = 0 && *y = 0`

thread 0	thread 1
a: $W_{\{rlx\}}$ `*x = 1`	c: $R_{\{rlx\}}$ `r0 = *y`
b: $W_{\{rlx\}}$ `*y = 1`	d: $R_{\{rlx\}}$ `r1 = *x`

Specification: *Sequential Consistency*—events have a total order, respecting per-thread program order

Schedule A

a

b

c

d

r0 == 1 && r1 == 1

Schedule B

c

d

a

b

r0 == 0 && r1 == 0

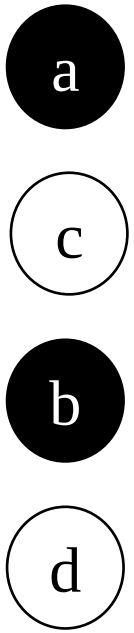
Memory Consistency

Initialize: `*x = 0 && *y = 0`

thread 0	thread 1
a: $W_{\{rlx\}}$ `*x = 1`	c: $R_{\{rlx\}}$ `r0 = *y`
b: $W_{\{rlx\}}$ `*y = 1`	d: $R_{\{rlx\}}$ `r1 = *x`

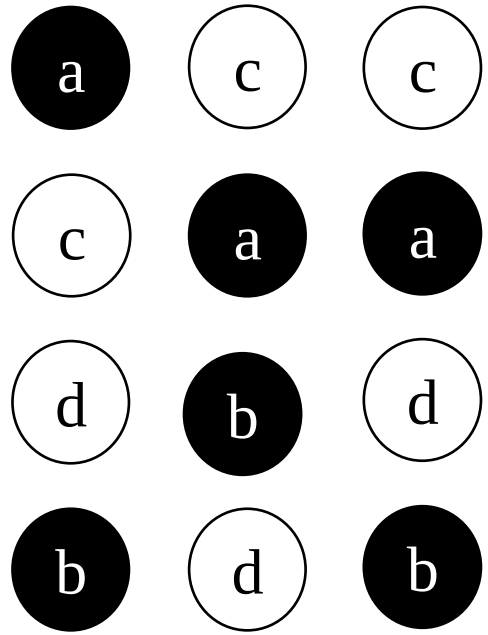
Specification: *Sequential Consistency*—events have a total order, respecting per-thread program order

Schedule C



r0 == 0 && r1 == 1

Schedule D-F



r0 == 0 && r1 == 1

Memory Consistency

Let’s try it out:

https://gpuharbor.ucsc.edu

GPU Harbor

gpuharbor.ucsc.edu

GPU Harbor

Home

Projects

Welcome to GPU Harbor at UCSC! This website is a collection of research projects on understanding and testing features of GPUs, mostly geared towards compute applications. GPU Harbor is under the umbrella of the LSD Lab and projects here are primarily supervised by Tyler Sorensen.

WebGPU Memory Model Testing

This project uses small concurrent programs called litmus tests to understand and test the behavior of the WebGPU memory model.

GPU Forward Progress Testing

A vizualization of the GPU forward progress tests generated for the OOPSLA 2021 paper: Specifying and Testing GPU Workgroup Progress Models.

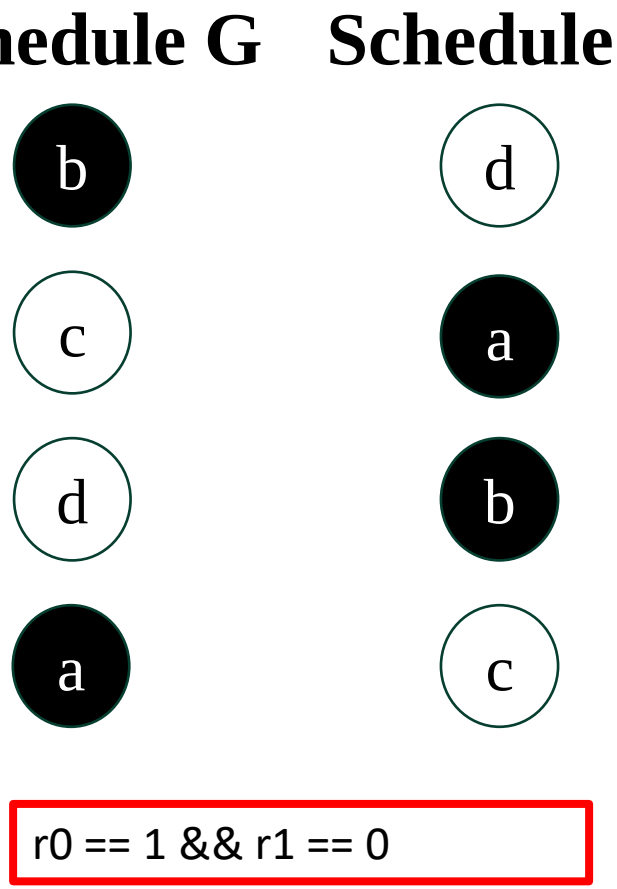
Memory Consistency

Initialize: `*x = 0 && *y = 0`

thread 0	thread 1
a: $W_{\{rlx\}}$ `*x = 1`	c: $R_{\{rlx\}}$ `r0 = *y`
b: $W_{\{rlx\}}$ `*y = 1`	d: $R_{\{rlx\}}$ `r1 = *x`

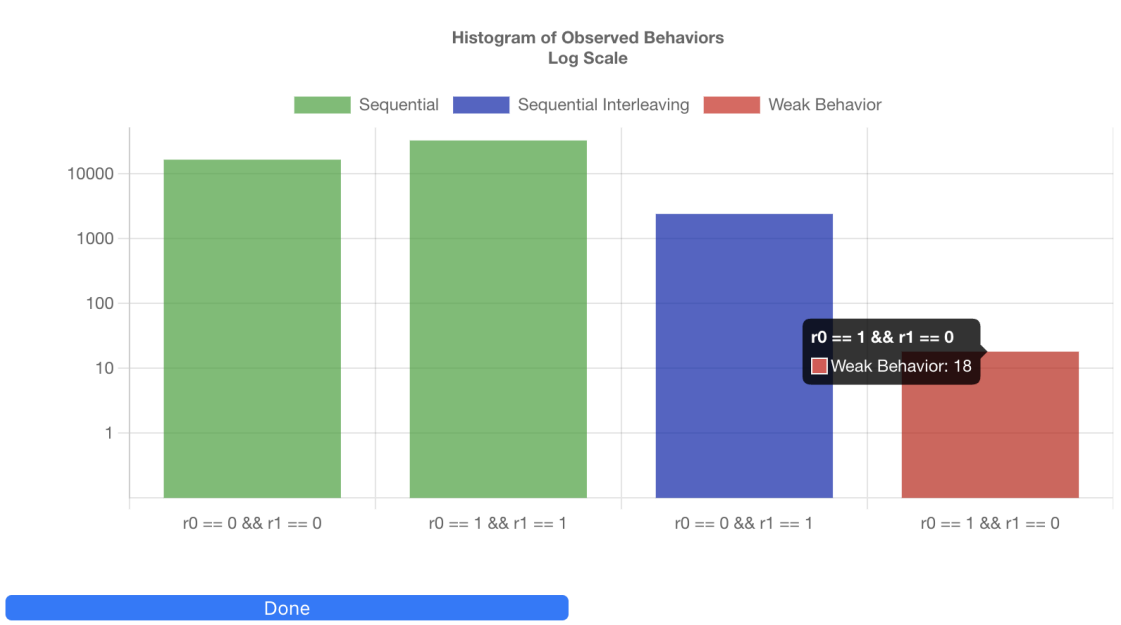
Relaxed MCSs allow ***weak*** behaviors beyond sequential consistency

Correspond to compiler/hardware optimizations



Apple M2 Results

Without Stress



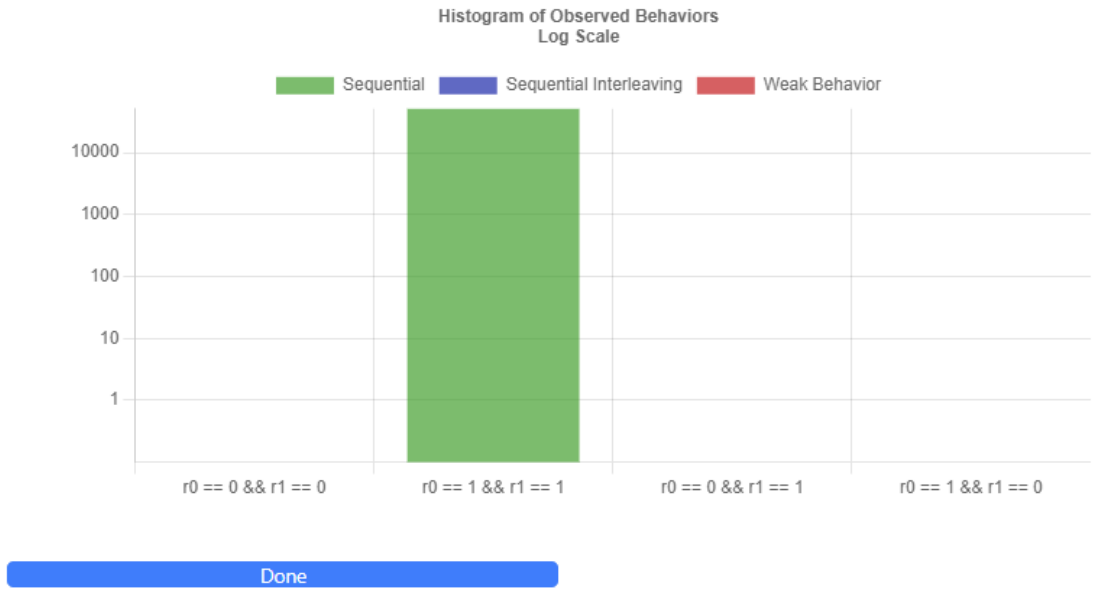
Done

With Stress

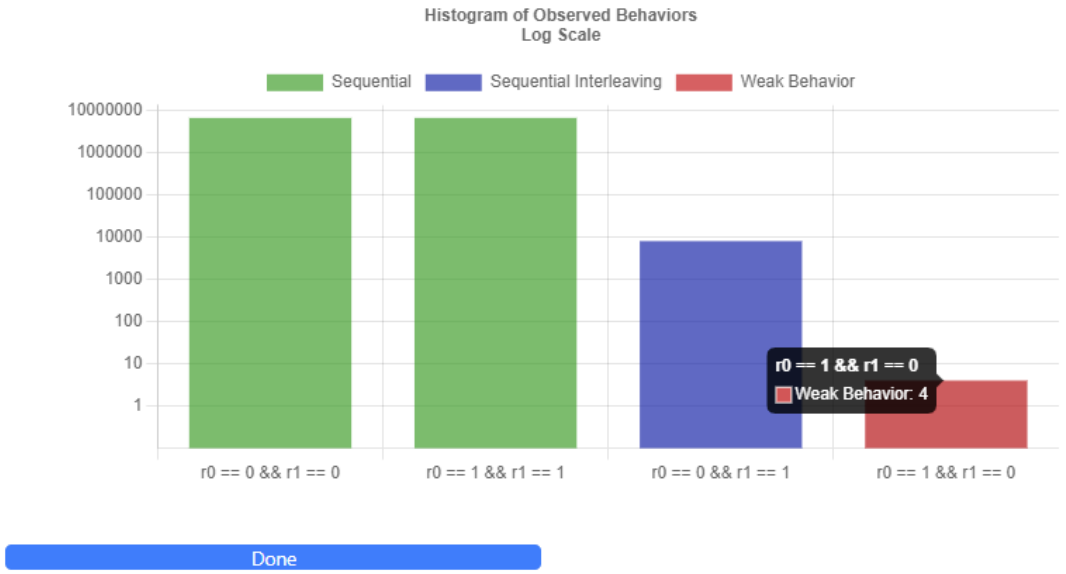
Done

NVIDIA RTX 3050 Ti Results

Without Stress



With Stress



Disallowing Weak Behaviors

Adding synchronization *disallows* weak behaviors (in an MCS that supports release/acquire fences)

Given behavior diversity, how do we

1.) know if tests are effective at uncovering bugs?

2.) know if the applications we write are portable?

Initialize: `*x = 0 && *y = 0`

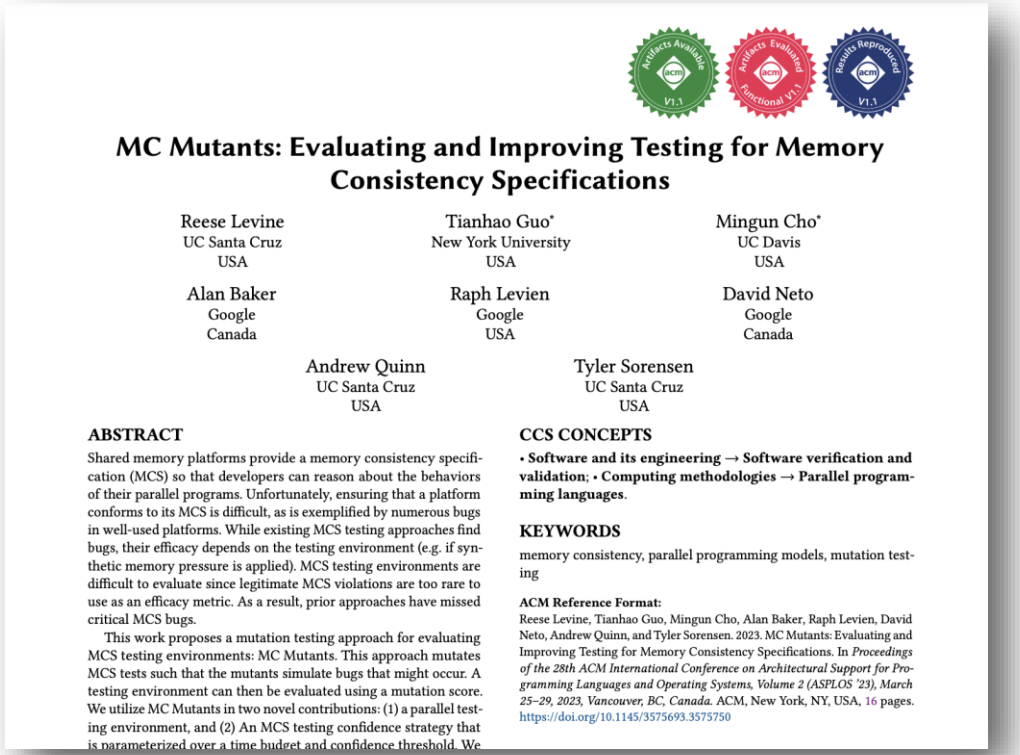
thread 0	thread 1
a: $W_{\{rlx\}}$ `*x = 1`	d: $R_{\{rlx\}}$ `r0 = *y`
b: $F_{\{rel\}}$	e: $F_{\{acq\}}$
c: $W_{\{rlx\}}$ `*y = 1`	f: $R_{\{rlx\}}$ `r1 = *x`

Condition: `r0 == 1 && r1 == 0`

Talk Outline

- Portable Testing: MC Mutants + GPUHarbor
- Memory models are not all we need
- Case Study: Prefix Scan

MC Mutants + GPUHarbor



MC Mutants: Evaluating and Improving Testing for Memory Consistency Specifications

Reese Levine
UC Santa Cruz
USA

Tianhao Guo*
New York University
USA

Mingun Cho*
UC Davis
USA

Alan Baker
Google
Canada

Raph Levien
Google
USA

David Neto
Google
Canada

Andrew Quinn
UC Santa Cruz
USA

Tyler Sorensen
UC Santa Cruz
USA

ABSTRACT

Shared memory platforms provide a memory consistency specification (MCS) so that developers can reason about the behaviors of their parallel programs. Unfortunately, ensuring that a platform conforms to its MCS is difficult, as is exemplified by numerous bugs in well-used platforms. While existing MCS testing approaches find bugs, their efficacy depends on the testing environment (e.g. if synthetic memory pressure is applied). MCS testing environments are difficult to evaluate since legitimate MCS violations are too rare to use as an efficacy metric. As a result, prior approaches have missed critical MCS bugs.

This work proposes a mutation testing approach for evaluating MCS testing environments: MC Mutants. This approach mutates MCS tests such that the mutants simulate bugs that might occur. A testing environment can then be evaluated using a mutation score. We utilize MC Mutants in two novel contributions: (1) a parallel testing environment, and (2) An MCS testing confidence strategy that is parameterized over a time budget and confidence threshold. We

CCS CONCEPTS

• **Software and its engineering → Software verification and validation**; • **Computing methodologies → Parallel programming languages**.

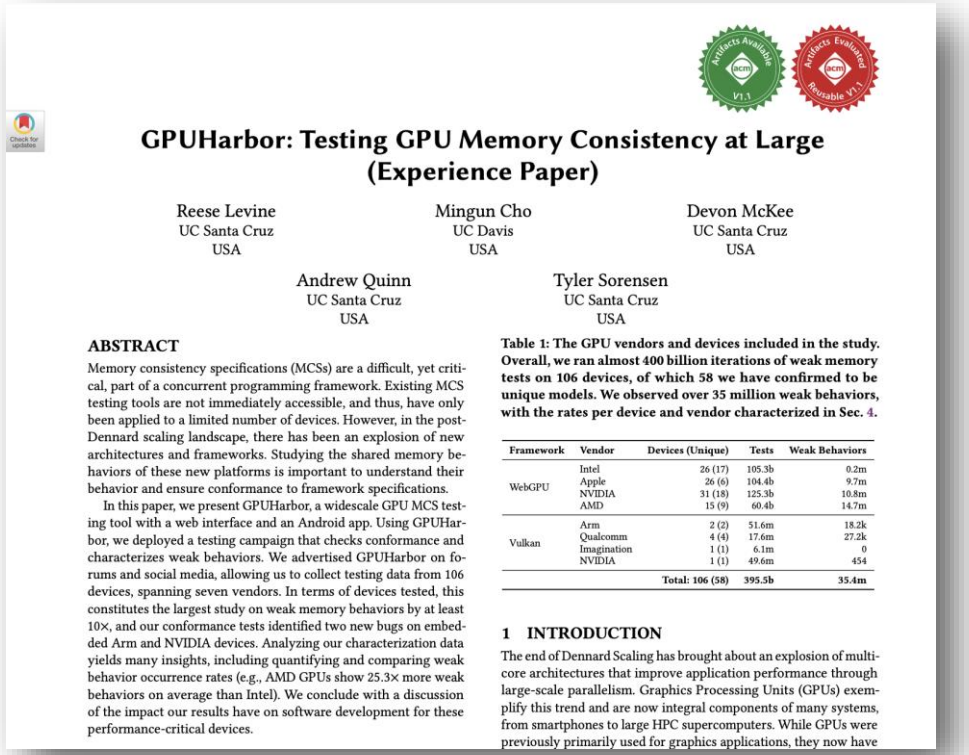
KEYWORDS

memory consistency, parallel programming models, mutation testing

ACM Reference Format:

Reese Levine, Tianhao Guo, Mingun Cho, Alan Baker, Raph Levien, David Neto, Andrew Quinn, and Tyler Sorensen. 2023. MC Mutants: Evaluating and Improving Testing for Memory Consistency Specifications. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3575693.3575750

ASPLOS 2023
Distinguished Paper, Distinguished Artifact



GPUHarbor: Testing GPU Memory Consistency at Large (Experience Paper)

Reese Levine
UC Santa Cruz
USA

Mingun Cho
UC Davis
USA

Devon McKee
UC Santa Cruz
USA

Andrew Quinn
UC Santa Cruz
USA

Tyler Sorensen
UC Santa Cruz
USA

ABSTRACT

Memory consistency specifications (MCSs) are a difficult, yet critical, part of a concurrent programming framework. Existing MCS testing tools are not immediately accessible, and thus, have only been applied to a limited number of devices. However, in the post-Dennard scaling landscape, there has been an explosion of new architectures and frameworks. Studying the shared memory behaviors of these new platforms is important to understand their behavior and ensure conformance to framework specifications.

In this paper, we present GPUHarbor, a widescale GPU MCS testing tool with a web interface and an Android app. Using GPUHarbor, we deployed a testing campaign that checks conformance and characterizes weak behaviors. We advertised GPUHarbor on forums and social media, allowing us to collect testing data from 106 devices, spanning seven vendors. In terms of devices tested, this constitutes the largest study on weak memory behaviors by at least 10×, and our conformance tests identified two new bugs on embedded Arm and NVIDIA devices. Analyzing our characterization data yields many insights, including quantifying and comparing weak behavior occurrence rates (e.g., AMD GPUs show 25.3× more weak behaviors on average than Intel). We conclude with a discussion of the impact our results have on software development for these performance-critical devices.

Table 1: The GPU vendors and devices included in the study. Overall, we ran almost 400 billion iterations of weak memory tests on 106 devices, of which 58 we have confirmed to be unique models. We observed over 35 million weak behaviors, with the rates per device and vendor characterized in Sec. 4.

Framework	Vendor	Devices (Unique)	Tests	Weak Behaviors
WebGPU	Intel	26 (17)	105.3b	0.2m
	Apple	26 (6)	104.4b	9.7m
	NVIDIA	31 (18)	125.3b	10.8m
	AMD	15 (9)	60.4b	14.7m
Vulkan	Arm	2 (2)	51.6m	18.2k
	Qualcomm	4 (4)	17.6m	27.2k
	Imagination	1 (1)	6.1m	0
	NVIDIA	1 (1)	49.6m	454
		Total: 106 (58)	**395.5b**	**35.4m**

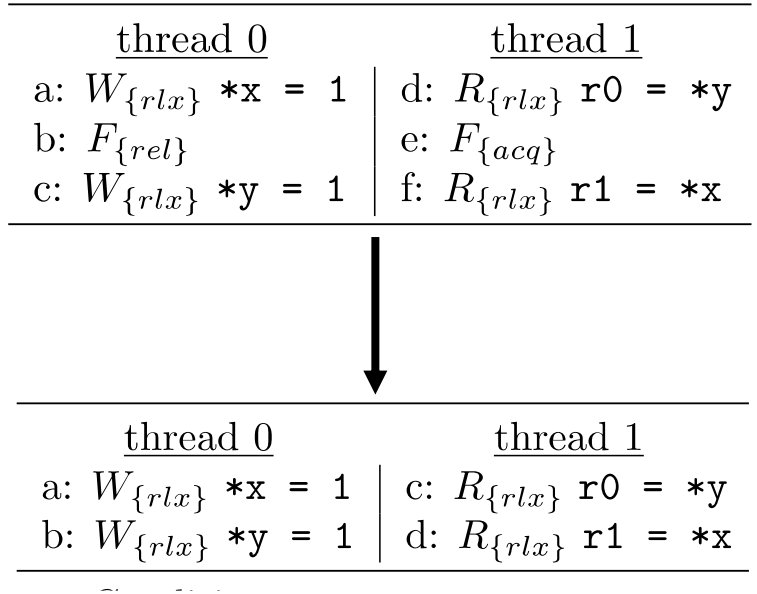
1 INTRODUCTION

The end of Dennard Scaling has brought about an explosion of multi-core architectures that improve application performance through large-scale parallelism. Graphics Processing Units (GPUs) exemplify this trend and are now integral components of many systems, from smartphones to large HPC supercomputers. While GPUs were previously primarily used for graphics applications, they now have

ISSTA 2023
Distinguished Artifact

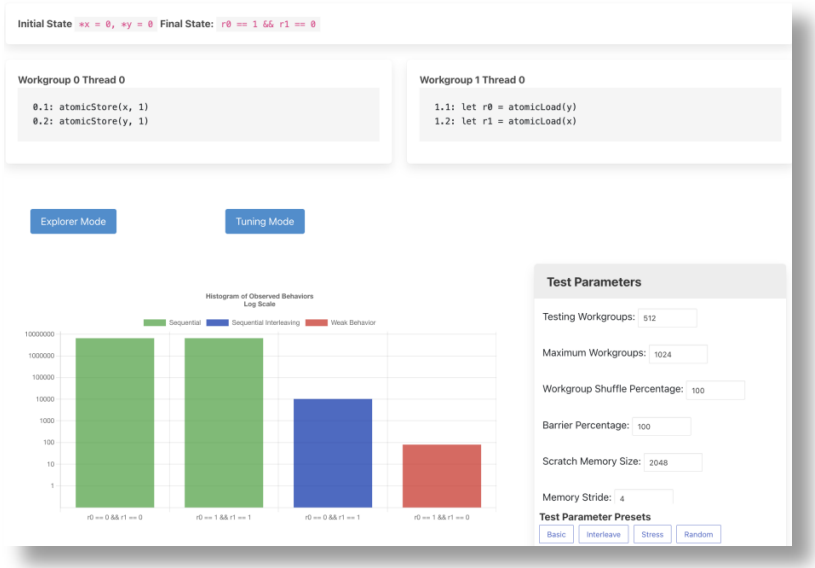
MC Mutants: Contributions

- **Developed** a way to evaluate the *effectiveness* of MCS testing, as well as new testing techniques
- **Evaluated** our methodology on four GPUs in **WebGPU**, GPU framework for browsers
- **Impact:** Discovered two bugs
  - Message Passing on AMD GPUs
  - Coherence on Intel GPU on a MacBook
- **Impact:** Integrated comprehensive MCS tests into the official WebGPU CTS



GPUHarbor: Scaling MC Mutants

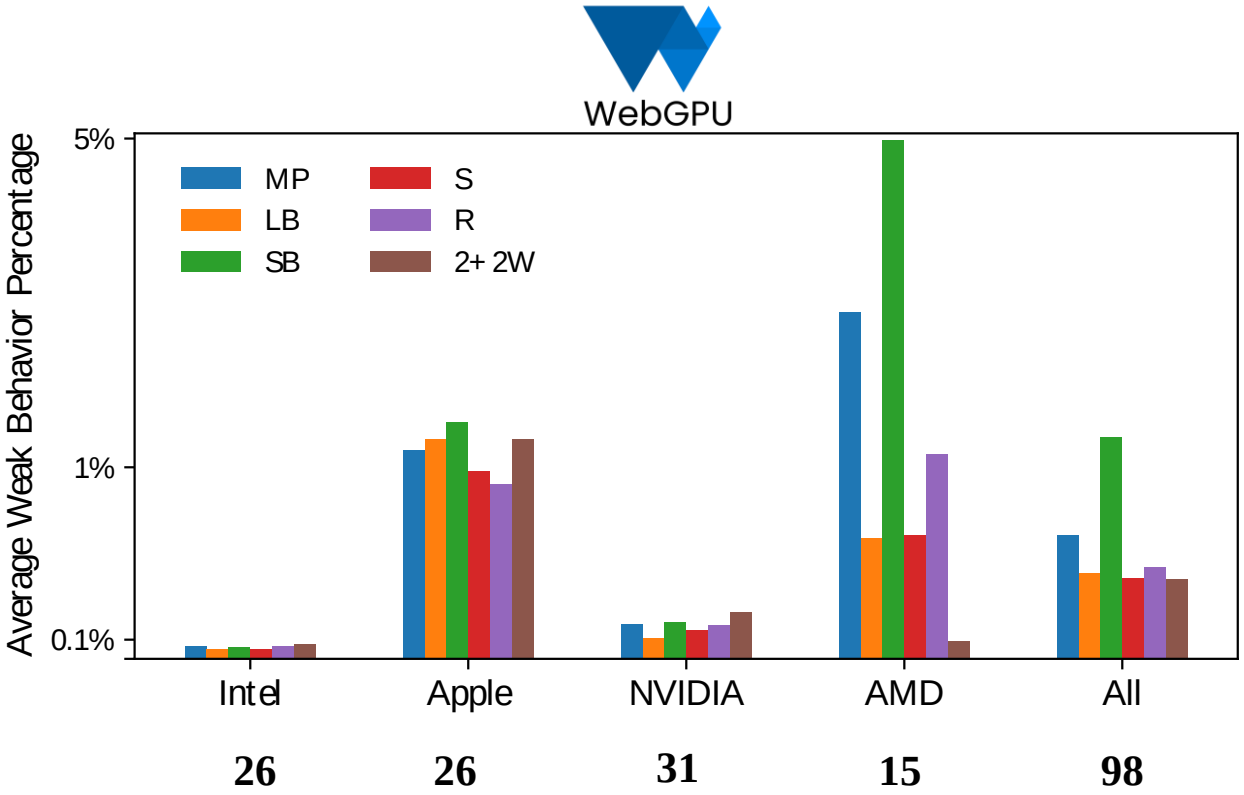
- MC Mutants only evaluated on four GPUs
- Website/Android app allow non-experts to run tests
- We utilize MC Mutants to test at scale
  - Total runtime: 31.1 hours, ~19 minutes per device
- **Similarity analysis** guides device choices when running CTSs

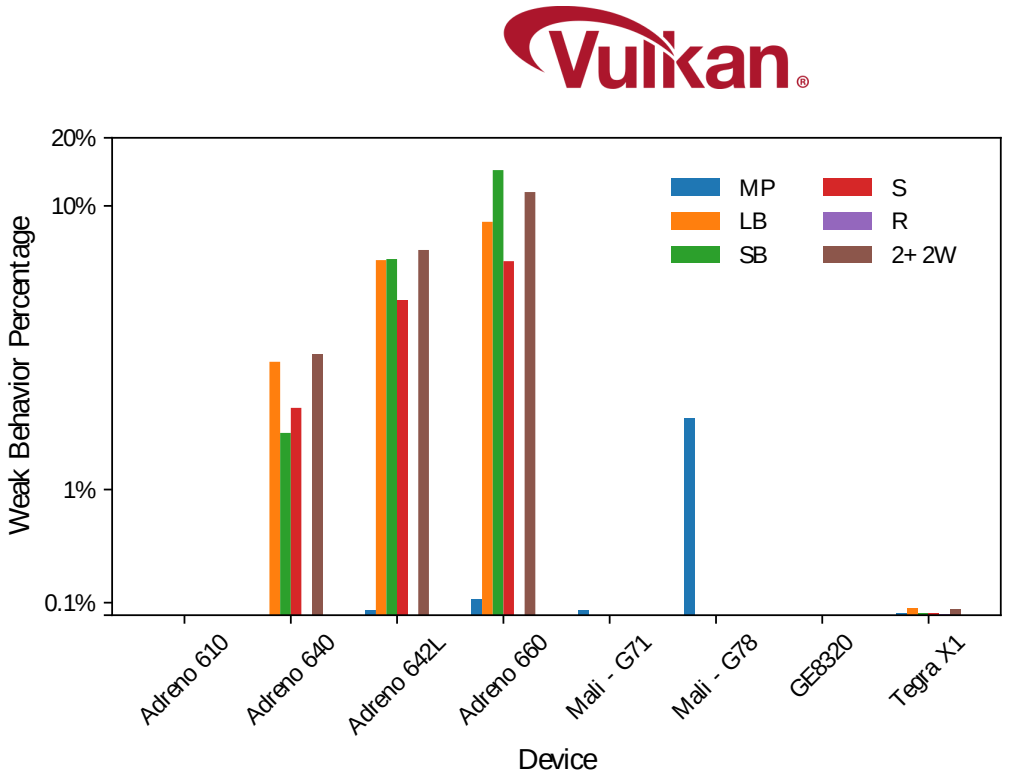


Framework	Vendor	Devices (Unique)	Tests	Weak Behaviors
WebGPU	Intel	26 (17)	105.3b	0.2m
	Apple	26 (6)	104.4b	9.7m
	NVIDIA	31 (18)	125.3b	10.8m
	AMD	15 (9)	60.4b	14.7m
Vulkan	Arm	2 (2)	51.6m	18.2k
	Qualcomm	4 (4)	17.6m	27.2k
	PowerVR	1 (1)	6.1m	0
	NVIDIA	1 (1)	49.6m	454
		Total: 106 (58)	**395.5b**	**35.4m**

Weak Behavior Characterization

- Analysis focuses on six weak memory tests: *Message Passing, Load Buffer, Store Buffer, Read, Store, 2+2 Write*





Bugs

- Bugs found in variations of a coherency litmus test

Initialize: `x = 0;`

thread 0	thread 1
a `S(x, 1);`	c `r0 = L(x);`
b `S(x, 2);`	d `r1 = L(x);`

Weak Behavior: `r0 > r1`

- **Arm**: observed on Mali-G71/G78 (Pixel) using Android/Vulkan, compiler bug

- **NVIDIA**: observed on Tegra X1, Quadro P620, Vulkan compiler bug affecting all pre-Volta GPUs

- **Apple**: Observed on GPUs from NVIDIA, Intel, AMD on Macbooks using WebGPU

Talk Outline

- Portable Testing: MC Mutants + GPUHarbor 
- Memory models are not all we need
- Case Study: Prefix Scan

Fairness

- GPUs are not fair schedulers
  - Sorensen, Salvador, Raval et al., *Specifying and Testing GPU Workgroup Progress Models*, OOPSLA 2021

Thread 0	Thread 1
`while(Exch(m,1)==1);`	`while(Exch(m,1)==1);`
`Store(m,0);`	`Store(m,0);`

- C++ Standard
  - *The implementation should make atomic stores visible to atomic loads, and atomic loads should observe atomic stores, within a reasonable amount of time.*
- Vulkan: includes *availability/visibility* operations
- E.g. can *relaxed* atomic loads be hoisted out of loops?
  - Hopefully, answer should be no
  - https://gitlab.freedesktop.org/mesa/mesa/-/issues/4475

```
while(atomic_load(x, mem_order_relaxed) == 0) {
  …
}
fence(mem_order_acquire);
```

```
val t = atomic_load(x, mem_order_relaxed);
while(t == 0) {
  …
}
fence(mem_order_acquire);
```

Talk Outline

- Portable Testing: MC Mutants + GPUHarbor 
- Memory models are not all we need 
- Case Study: Prefix Scan

Prefix Scan

Goal

- Saturate memory bandwidth, match performance of memcpy kernel

Issues

- Subgroup APIs not stable
- Implementations may contain memory model bugs
- Forward progress not guaranteed by Vulkan

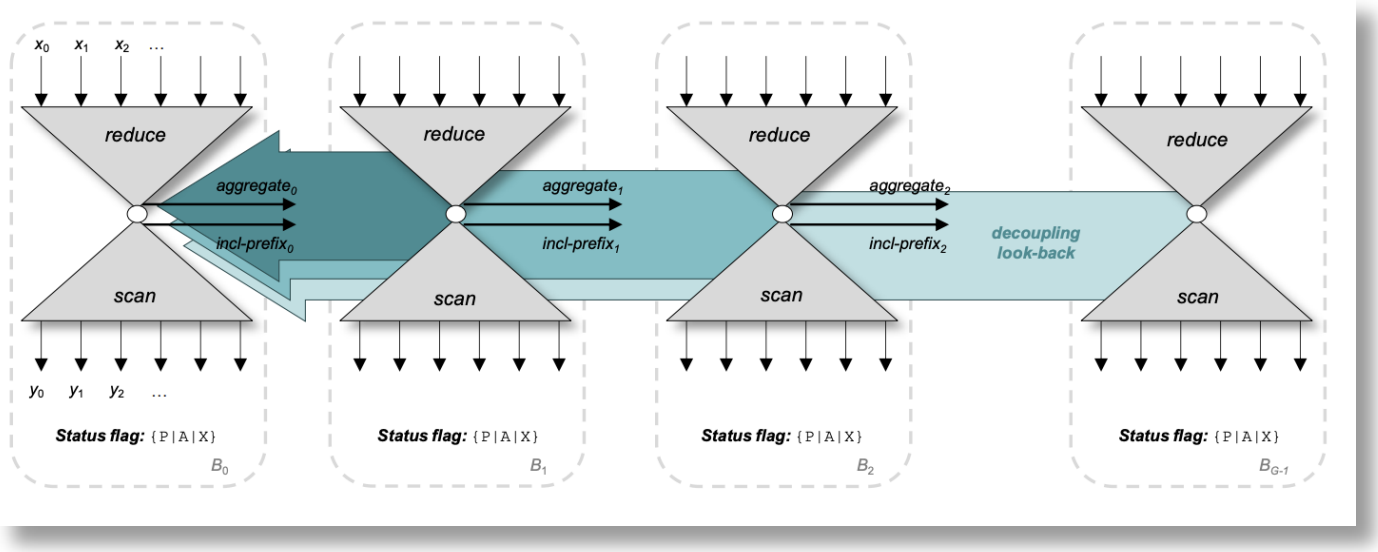


Figure 5, *Single-pass prefix scan*, Merrill et al., 2016

Intel UHD Graphics 770 Integrated GPU
1 GB data

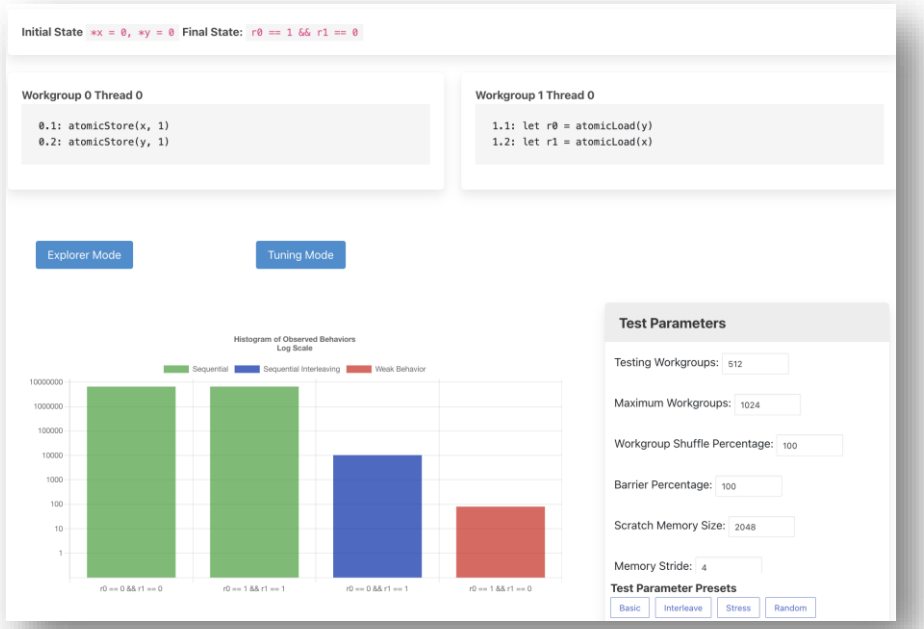
	Rate (GBPS)
memcpy	33.06
prefix sum	24.56

Takeaways + Discussion

- Stress testing techniques necessary for GPUs
- What are your use cases for synchronization, forward progress?
- Memory model, forward progress, other features?
- Do we really need release/acquire for real applications?

Testing the Vulkan Memory Model

- Portable testing necessary for finding bugs, providing confidence in Vulkan memory model
- Memory models need to be thought of in conjunction with other concurrency features
- Specifications ultimately motivated by use cases: prefix scan, yours?



https://gpuharbor.ucsc.edu

- Presented by **Reese Levine**, PhD candidate at UC Santa Cruz
  - https://reeselevine.github.io/

UC SANTA CRUZ
BaskinEngineering