Testing the Vulkan Memory Model

Reese Levine, PhD Candidate, UC Santa Cruz
Diversity of GPUs

- Vulkan
- AMD
- Intel
- NVIDIA
- MoltenVK
- Qualcomm
- arm
- Imagination
GPGPU Applications

Machine Learning

High Performance Computing

Edge Device Acceleration
Fine-grained synchronization

• Mutexes

• Prefix Scans

• Concurrent Queues

• Collective Communication
Testing the foundations of fine-grained synchronization

https://gpuharbor.ucsc.edu
Memory Consistency

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

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

thread 0
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: \( *x = 0 \) && \( *y = 0 \)

\[
\begin{array}{c|c}
\text{thread 0} & \text{thread 1} \\
\hline
a: W_{rlx} \quad *x = 1 & c: R_{rlx} \quad r0 = *y \\
b: W_{rlx} \quad *y = 1 & d: R_{rlx} \quad r1 = *x \\
\end{array}
\]

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

Schedule A

\[
\begin{array}{c}
a \\
b \\
c \\
d \\
\end{array}
\]

\[ r0 == 1 \land r1 == 1 \]

Schedule B

\[
\begin{array}{c}
c \\
d \\
a \\
b \\
\end{array}
\]

\[ r0 == 0 \land r1 == 0 \]
Memory Consistency

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

<table>
<thead>
<tr>
<th>thread 0</th>
<th>thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a: W_{rlx} *x = 1</td>
<td>c: R_{rlx} r0 = *y</td>
</tr>
<tr>
<td>b: W_{rlx} *y = 1</td>
<td>d: R_{rlx} r1 = *x</td>
</tr>
</tbody>
</table>

Schedule C

Schedule D-F

r0 == 0 && r1 == 1

r0 == 0 && r1 == 1

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

Let’s try it out:

https://gpuharbor.ucsc.edu
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 visualization of the GPU forward progress tests generated for the OOPSLA 2021 paper: Specifying and Testing GPU Workgroup Progress Models.
Memory Consistency

Initialize: \( *x = 0 \land *y = 0 \)

<table>
<thead>
<tr>
<th>thread 0</th>
<th>thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a: ( W_{rlx} ) ( *x = 1 )</td>
<td>c: ( R_{rlx} ) r0 = *y</td>
</tr>
<tr>
<td>b: ( W_{rlx} ) ( *y = 1 )</td>
<td>d: ( R_{rlx} ) r1 = *x</td>
</tr>
</tbody>
</table>

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

Correspond to compiler/hardware optimizations

\[ r0 == 1 \land r1 == 0 \]
Apple M2 Results

Without Stress

With Stress

Histogram of Observed Behaviors
Log Scale

Sequential  Sequential Interleaving  Weak Behavior

Done

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?

<table>
<thead>
<tr>
<th>Initialize: ( *x = 0 \land *y = 0 )</th>
</tr>
</thead>
</table>
| \begin{align*}
\text{thread } 0 \\
\text{a: } & W_{rlx} *x = 1 \\
\text{b: } & F_{rel} \\
\text{c: } & W_{rlx} *y = 1
\end{align*} | \begin{align*}
\text{thread } 1 \\
\text{d: } & R_{rlx} r0 = *y \\
\text{e: } & F_{acq} \\
\text{f: } & R_{rlx} r1 = *x
\end{align*} |
| Condition: \( r0 == 1 \land r1 == 0 \) |
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 Cao
New York University
USA
Minggan Cho
UC Davis
USA
Andrew Quinn
UC Santa Cruz
USA
Tyler Sorensen
UC Santa Cruz
USA
Raph Levien
Google
Canada
David Neto
Google
Canada

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 it is exemplified by numerous bugs in such 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 presents a mutation testing framework for evaluating MCS testing environments. MC Mutants. This approach mutates MCS tests such that the mutant simulates bugs that might occur. A testing environment can then be evaluated using a mutation score. We utilize MC Mutants in two novel contributions: (1) parallel testing environments, and (2) an MCS testing confidence strategy that is provably more effective than existing MCS frameworks.

CSC 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


GPUHarbor: Testing GPU Memory Consistency at Scale

Reese Levine
UC Santa Cruz
USA
Minggan Cho
UC Davis
USA
Andrew Quinn
UC Santa Cruz
USA
Tyler Sorensen
UC Santa Cruz
USA
Devon McKee
UC Santa Cruz
USA

ABSTRACT

Memory consistency specifications (MCSs) are difficult to verify and validate, part of a concurrent programming framework. Existing MCS testing tools are not immediately usable, and thus, have neither been applied to a limited number of devices. In the post-Demand 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, an end-to-end GPU MCS testing tool with a web interface and an Android app. Using GPUHarbor, we deployed a testing campaign that showed conformance and characterization weak behaviors. We validated GPUHarbor on 2308 devices and 18 benchmarks, allowing us to collect testing data from 163 devices, spanning seven vendors. In terms of device tested, this constitutes the largest study of weak memory behaviors by at least 3×, and our conformance tests identified new bugs on benchmark and NVIDIA devices. Analyzing our characterization results yields many insights, including quantifying and comparing weak behaviors across vendors (e.g., A10G GPUs show 13% more weak behaviors on average than their). 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 80 million iterations of weak memory tests on 163 devices, of which 30 we have confirmed to be unique models. We observed over 35 million weak behaviors, with the rate per device and vendor characterized in Sec. 4.

<table>
<thead>
<tr>
<th>Device</th>
<th>Vendor</th>
<th>Bugs</th>
<th>Rate</th>
<th>Weak Behaviors</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100</td>
<td>NVIDIA</td>
<td>350</td>
<td>81,761</td>
<td>35,582</td>
</tr>
<tr>
<td>T4</td>
<td>NVIDIA</td>
<td>57</td>
<td>58,999</td>
<td>33,029</td>
</tr>
<tr>
<td>8080</td>
<td>NVIDIA</td>
<td>119</td>
<td>54,955</td>
<td>31,944</td>
</tr>
<tr>
<td>8800</td>
<td>NVIDIA</td>
<td>200</td>
<td>52,919</td>
<td>28,091</td>
</tr>
<tr>
<td>7000</td>
<td>NVIDIA</td>
<td>98</td>
<td>58,999</td>
<td>26,529</td>
</tr>
<tr>
<td>6800</td>
<td>NVIDIA</td>
<td>50</td>
<td>54,955</td>
<td>25,145</td>
</tr>
<tr>
<td>5000</td>
<td>NVIDIA</td>
<td>15</td>
<td>52,919</td>
<td>20,246</td>
</tr>
<tr>
<td>A10G</td>
<td>NVIDIA</td>
<td>98</td>
<td>36,897</td>
<td>25,924</td>
</tr>
<tr>
<td>A1600</td>
<td>NVIDIA</td>
<td>119</td>
<td>29,396</td>
<td>18,944</td>
</tr>
<tr>
<td>6000</td>
<td>NVIDIA</td>
<td>10</td>
<td>54,955</td>
<td>18,091</td>
</tr>
<tr>
<td>5000</td>
<td>NVIDIA</td>
<td>11</td>
<td>52,919</td>
<td>15,799</td>
</tr>
<tr>
<td>2080</td>
<td>NVIDIA</td>
<td>98</td>
<td>36,088</td>
<td>15,813</td>
</tr>
</tbody>
</table>

Total: 5,200 30,981 35,716

1 INTRODUCTION

The end of Demand Scaling has brought about a revolution of multi-core architectures that improve application performance through large-scale parallelization. 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

ASPLOS 2023
Distinguished Paper, Distinguished Artifact

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

<table>
<thead>
<tr>
<th>Framework</th>
<th>Vendor</th>
<th>Devices (Unique)</th>
<th>Tests</th>
<th>Weak Behaviors</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebGPU</td>
<td>Intel</td>
<td>26 (17)</td>
<td>105.3b</td>
<td>0.2m</td>
</tr>
<tr>
<td></td>
<td>Apple</td>
<td>26 (6)</td>
<td>104.4b</td>
<td>9.7m</td>
</tr>
<tr>
<td></td>
<td>NVIDIA</td>
<td>31 (18)</td>
<td>125.3b</td>
<td>10.8m</td>
</tr>
<tr>
<td></td>
<td>AMD</td>
<td>15 (9)</td>
<td>60.4b</td>
<td>14.7m</td>
</tr>
<tr>
<td>Vulkan</td>
<td>Arm</td>
<td>2 (2)</td>
<td>51.6m</td>
<td>18.2k</td>
</tr>
<tr>
<td></td>
<td>Qualcomm</td>
<td>4 (4)</td>
<td>17.6m</td>
<td>27.2k</td>
</tr>
<tr>
<td></td>
<td>PowerVR</td>
<td>1 (1)</td>
<td>6.1m</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>NVIDIA</td>
<td>1 (1)</td>
<td>49.6m</td>
<td>454</td>
</tr>
</tbody>
</table>

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

- **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

• 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);
```
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

<table>
<thead>
<tr>
<th>Intel UHD Graphics 770 Integrated GPU 1 GB data</th>
<th>Rate (GBPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcpy</td>
<td>33.06</td>
</tr>
<tr>
<td>prefix sum</td>
<td>24.56</td>
</tr>
</tbody>
</table>
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?

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