8 years of open drivers

Faith Ekstrand, Collabora
Iago Toral, Igalia
About Iago

- Graphics engineer at Igalia
- Contributor to Mesa since 2014
- Initial contributions in Mesa to Intel drivers for OpenGL and Vulkan
- Currently leading the Raspberry Pi 3D driver team at Igalia, mostly focused on the Vulkan stack
About Faith

- Active member of the Vulkan working group since before 1.0
- Led Linux Vulkan driver development at Intel for 6 years
- Now at Collabora working on Linux graphics over-all
- Mesa Maintainer, focused on:
  - NIR optimizing shader compiler
  - SPIR-V front-end
  - Common Vulkan runtime
  - NVK, the new open-source driver for NVIDIA hardware
History of Vulkan in Mesa

● Mesa Vulkan work started May 8, 2015 (that’s before the 1.0 release 😄)
● Initial Intel driver was released on February 16, 2016
  ○ Released in tandem with the Vulkan 1.0 spec
  ○ One of the first 4 conformant implementations
  ○ Merged into mesa/main about 2 months later
● The RADV driver for AMD hardware landed October 7, 2016
  ○ Initially not conformant, but some apps ran
● Since then, we’ve gained drivers for nearly every major HW vendor:
  ○ Qualcomm (Turnip), Arm (Panvk), Broadcom (V3DV), Imagination (PVR), and NVIDIA (NVK)
● Lavapipe, utilizing the LLVMpipe CPU rasterization core
History of Vulkan in Mesa

Mesa also has a common runtime, shared across HW drivers

- Some of this was built as part of the initial Intel efforts:
  - NIR optimizing shader compiler core
  - SPIR-V front-end for NIR
  - Window-system code for X11 and Wayland

- Other pieces have been added later:
  - Dispatch and vkGet*ProcAddr() handling
  - vkQueueSubmit() threading and timeline semaphores
  - Render passes in terms of VK_KHR_dynamic rendering
  - Pipelines in terms of VK_EXT_shader_object (still WIP)
A timeline of Vulkan in Mesa

As of February 1, 2024

Intel
AMD
Qualcomm
Arm
Lavapipe
Raspberry PI
Imagination
Nvidia

Pre-Merged
Merged
1.0
1.1
1.2
1.3
# The Mesa Matrix (https://mesamatrix.net)

## Vulkan 1.3

<table>
<thead>
<tr>
<th>Extension</th>
<th>mesa</th>
<th>AMD</th>
<th>Arm</th>
<th>Broadcom</th>
<th>Intel</th>
<th>Nvidia</th>
<th>PowerVR</th>
<th>Qualcomm</th>
<th>Software</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_KHR_copy_commands2</td>
<td>100.0%</td>
<td>100.0%</td>
<td>8.7%</td>
<td>78.3%</td>
<td>100.0%</td>
<td>91.3%</td>
<td>95.7%</td>
<td>21.7%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>VK_KHR_dynamic_rendering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_KHR_format_feature_flags2</td>
<td>2023-07-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-09-05</td>
<td>2024-01-14</td>
<td>2023-11-17</td>
<td>2023-11-17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_KHR_shader_integer_dot_product</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_KHR_shader_non_semantic_info</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_KHR_shader_terminate_invocation</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_KHR_synchronization2</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_KHR_zero_initialize_workgroup_memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_EXT_4444_formats</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_EXT_extended_dynamic_state</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_EXT_extended_dynamic_state2</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_EXT_inline_uniform_block</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_EXT_pipeline_creation_cache_control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_EXT_pipeline_creation_feedback</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-07-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td>2023-08-04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_EXT_private_data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_EXT_image_robustness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_EXT_shader_demote_to_helper_invocation</td>
<td>2024-01-09</td>
<td>2024-01-09</td>
<td>2024-01-09</td>
<td>2024-01-09</td>
<td>2024-08-04</td>
<td>2024-08-04</td>
<td>2024-08-04</td>
<td>2024-08-04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_EXT_subgroup_size_control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VK_EXT_texture_compression_astc_hdr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Image of the Mesa Matrix]
Layered drivers

- **Venus:**
  - VirtIO-GPU Vulkan driver.
  - Allows guest OS to use the host's native Vulkan implementation.
  - Merged upstream in April 2021.

- **Dozen:**
  - Vulkan over Direct3D 12
  - Vulkan support on Microsoft platforms where native Vulkan is not available.
  - Merged upstream in March 2022.

- **Zink:**
  - OpenGL over Vulkan.
  - OpenGL support on platforms where native OpenGL is not available.
  - Merged upstream in October 2019.
Useful environment variables

- **MESA_VK_TRACE=rmv,rgp,rra**
  - Comma-separated list of trace types to dump for offline analysis.
  - All currently supported trace types are for AMD Radeon tools:
    - Radeon Memory Visualizer: `rmv` (RADV, ANV)
    - Radeon GPU profiler: `rgp` (RADV)
    - Radeon Raytracing Analyzer: `rra` (RADV)

- **MESA_VK_TRACE_FRAME**
  - Specifies a frame index to capture.

- **MESA_VK_TRACE_TRIGGER**
  - Triggers frame capture when specified trigger file is created.
Useful environment variables

- **MESA_VK_WSI_PRESENT_MODE=fifo|relaxed|mailbox|immediate**
  - Overrides present mode in VkSwapchainCreateInfoKHR::presentMode.

- **MESA_VK_HEADLESS_SWAPCHAIN**
  - Forces all swapchains to be headless, as if VK_EXT_headless_surface was used.

- **MESA_VK_ABORT_ON_DEVICE_LOSS (*)**
  - Aborts execution upon seeing VK_ERROR_DEVICE_LOST.

- **MESA_VK_ENABLE_SUBMIT_THREAD**
  - Forces use of a submit thread on all queue submissions.
Useful environment variables

● Per-driver variables
  ○ These expose many driver-specific options.
  ○ Some accept ‘help’ as value to dump the list of accepted options.
    ■ INTEL_DEBUG (Anvil)
    ■ RADV_DEBUG, ACO_DEBUG (Radv)
    ■ TU_DEBUG (Turnip)
    ■ V3D_DEBUG (V3DV)
    ■ NVK_DEBUG (NVK)
    ■ PANVK_DEBUG (Panvk)
Useful Vulkan layers

- **VK_LAYER_MESA_device_select**
  - Forces device selection on platforms with multiple Vulkan implementations available.
  - Build with `-Dvulkan-layers=device-select`

```bash
$ MESA VK_DEVICE_SELECT=list vkcube
selectable devices:
  GPU 0: 8086:46a6 "Intel(R) Graphics (ADL GT2)" integrated GPU
  GPU 1: 14e4:46a6 "V3D 4.2.14" integrated GPU

$ MESA VK DEVICE SELECT=8086:46a6 vkcube
Selected GPU 0: Intel(R) Graphics (ADL GT2), type: IntegratedGpu
```
Useful Vulkan layers

- **VK_LAYER_MESA_overlay**
  - Renders an overlay with selected statistics.
  - Build with -Dvulkan-layers=overlay.
  - `VK_LAYER_MESA_OVERLAY_CONFIG=help` for available stats.
Benefits of open drivers

● **Linux OS integration**
  ○ You already have a Mesa driver 😊
  ○ Packaged by all the Linux and BSD distros
  ○ No binary downloads from independent 3rd parties.
Benefits of open drivers

- Linux OS integration
- Application debugging
  - We aren’t hiding anything! (You can look at the code)
  - All the Driver developer tools are available to you
    - Environment variables
    - Command buffer streams
    - Shader compiler dumps
  - Most drivers support VK_pipeline_executable_properties integration for RenderDoc.
Benefits of open drivers
Benefits of open drivers

- Linux OS integration
- Application debugging
- Source code access
  - You can look under the hood
  - See how software drives the actual hardware
  - Build in debug mode and set breakpoints inside the driver
  - Maybe you can even try to fix your issue yourself!
Benefits of open drivers

- Linux OS integration
- Application debugging
- Source code access
- Better access to driver developers
  - We respond directly to issues
    - https://gitlab.freedesktop.org/mesa/mesa/-/issues/
  - You can join our IRC channels and ask questions
Benefits of open drivers

Julien Barnoin
@julienbarnoin@mastodon.gamedev.place

Sometimes open source really delivers 😍

Working on my Vulkan-based engine I've been hitting this obscure shader bug for a few months, and on sunday I finally created and shared a repro program and submitted a bug report to Mesa. @pixelcluster immediately investigated and delivered a fix, it was merged into main the next day, and then released on wednesday in Mesa 23.3.4.

A 3 day turnaround time to release is pretty incredible. Thanks to all who contributed!

gitlab.freedesktop.org/mesa/me...
Benefits of open drivers

Relaxed atomic loads in while loops being optimized away

Describe the issue

It appears that while loops that include a relaxed atomic memory load are being optimized away. Here are a couple amber tests that show the problem: https://gist.github.com/reeselevine/935febcc7a8c4c192c234c54522f0cb0

The infinite loop test should hang, since the flag is never updated, and on an Nvidia Quadro RTX 4000 it does, as expected. However, on the Intel GPU, the test passes immediately.

To confirm this is not due to an optimization that is simply removing an infinite loop, I wrote a second example that mimics message passing, where one thread updates data and sets a flag, while the other spins on the flag and then loads the data. This test should always pass, however on the Intel GPU the amber test fails.

Interestingly, if the commented out fence in the message-passing amber test (line 98) is uncommented, the test passes. It also happens if the load on line 96 is strengthened to an acquire. Therefore, this issue seems only to affect relaxed memory accesses.
Hey [Name], [Name] and I talked more about this today, and came to a better understanding of what's going on. The issue is that while it's true we can't rely on relaxed atomics for synchronization, the relaxed spin loop along with the following `OpMemoryBarrier` are enough to induce the ordering mentioned above, so we should expect the test to pass. So from the perspective of the memory model, it's not true that the loop has no side effects.

I went ahead and modified the amber test I have above here: [https://gist.github.com/reeselevine/935febcc7a8c4192c234c54522f0cb0#file-spooky-message-passing-amber](https://gist.github.com/reeselevine/935febcc7a8c4192c234c54522f0cb0#file-spooky-message-passing-amber)

As written, the test fails, but if store on line 104 is uncommented, the test passes. It does appear to be a little spooky, in that observing the result of the loop completely changes the semantics of the program.

I agree with the argument that practically speaking, would be nice if the loop did not get removed, so I'll try to have an MR to handle that and we can test it. On top of that, I think other atomic operations (e.g. max) would've prevented the loop to be removed, so making the atomic load behave the same is consistent.

However, I'm still a bit puzzled with the expectations about relaxed memory ordering. Still thinking about the loop example: Is there a guarantee that the relaxed atomic store will be visible by a relaxed atomic load in another thread? Or an infinite loop could be a valid result?
Benefits of open drivers

- Linux OS integration
- Application debugging
- Source code access
- Better access to driver developers
- Better cross-company collaboration
  - All the Mesa developers talk to each other
  - We all work on the common runtime and shader compiler
  - Most of us are also Khronos members
    - Pre-release extension work happens in Khronos gitlab
Want to help out?

Mesa lives at [https://gitlab.freedesktop.org/mesa/mesa/](https://gitlab.freedesktop.org/mesa/mesa/)
Want to help out?

Mesa lives at https://gitlab.freedesktop.org/mesa/mesa/

And also https://mesa3d.org
Want to help out?

Mesa lives at https://gitlab.freedesktop.org/mesa/mesa/

And also https://mesa3d.org

We’re also in the #dri-devel channel on the OFTC IRC server
Any Questions?
Thanks!