# io_uring for DRM

Liviu Dudau

Arm Ltd

## Who am I

- Kernel contributor for about 10 years
- Maintainer for Arm Ltd display drivers for the last 8 years
- Now involved with Panthor kernel driver
- Maintainer for the Mali CSF firmware in linux-firmware
- Interfacing with community for Arm' GPU software team
- Attending 2nd XDC in Montreal (wave)

## What problem are you trying to solve?

- Modern GPUs want to move job submission outside the kernel
- Mesa's Gallium strongly assumes that the kernel is in charge of submitting jobs
- Vulkan implementations would really like to do the submission from user space
- Performance can be gained from allowing user space to submit jobs directly to the HW or firmware
- Synchronization between all three sides is harder than just doing everything in the kernel

## What if we can have the cake and eat it?

- There is a mechanism in the kernel for doing asynchronous submission of jobs
- Allows for a mixed world where user space is in charge of creating the jobs
- Leaves the kernel to do the low level work of talking to the hardware and fetching the result of the command execution
- Should offer some of the benefits of doing most of the work in user space and all the drawbacks of accepting unchecked user commands
- io_uring to rule them all!

# Why io_uring?

- Allows us to decouple the job submission in user space from actual submission to hardware
- We avoid paying the context switch cost for small submissions where we don't care that much about the job state
- We can sanity check the job before actual submission (TBD)
- We can mix direct kernel and user space submission in the same app, based on performance needs rather than startup flags
- crazy idea: JIT switching between sending jobs one by one to the kernel vs batching them in user space
- Job submission to hardware stays the same
- Fence signalling stays in the kernel
- Implementing io_uring at the drm_ level means all upstream drivers gain support for user space submissions

# Quick recap on io_uring

- User space has access to a submission queue and a completion queue
- Jobs added to the submission queue can be executed immediately by the kernel or a kthread can be started after a number of submissions
- User space reads from the completion queue to find out the result of the submission
- Kernel thread runs submission jobs in order, but they can complete in any order
- Not very different from what AMD has proposed for their user space submission

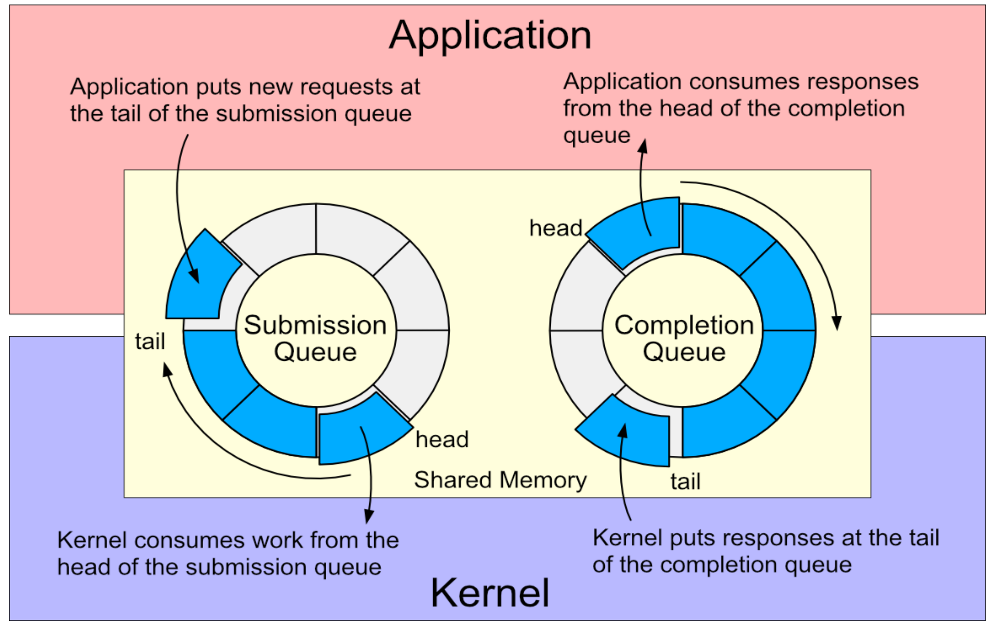
Application
Application puts new requests at the tail of the submission queue
Application consumes responses from the head of the completion queue
head
tail
Submission Queue
Completion Queue
head
Shared Memory
tail
Kernel consumes work from the head of the submission queue
Kernel puts responses at the tail of the completion queue
Kernel

Image by Donald Hunter

# What am I proposing?

- Add support for io_uring in the DRM subsystem
- Work gets handed over to the kernel drivers via a submission queue that wraps an actual GPU job submission
- The submission queue item only contains pointers to buffers and fences, so it should eliminate kernel copies of the submission
- Rest of the DRM submission path stays the same as now
- When job finishes we create a completion item that gets added to the io_uring's completion queue

## io_uring for DRM (cont)

- The io_uring stays at the DRM framework level, so it should be usable by all GPU kernel drivers
- We might need to understand specific details about each GPU job to be able to check the submission, or we can call the driver to do the check (like drm_atomic_check() for display drivers)
- We can filter out requests that don't make sense in the DRM context

## What's the catch?

- io_uring can potentially allow for some "interesting" features that are hard to control (pass a framebuffer that is actually a network socket or a file)
- App needs more permissions/capabilities to be able to use the io_uring buffers
- Feature is completely disabled in ChromeOS and disabled for apps in Android since 2023
- Not sure about the mapping with drm_scheduler (1:1, 1:N)

# Request for feedback

- Is this a good idea?
- Does anyone want it?
- Should we have only one kthread per FD or one per drm_scheduler?
- If io_uring is not acceptable, can we have a restricted version of it? AMD's proposed user mode submission has a pretty similar structure
- Should we restrict the IOCTLs to just DRM ones, or allow for "creative" freedom?
- Rust bindings?

Thanks!