Sync Points in the Intel Gfx Driver

Jesse Barnes
Intel Open Source Technology Center

Agenda

- History and other implementations
  - Other I/O layers - block device ordering
  - NV_fence, ARB_sync
  - EGL_native_fence_sync, Android Sync Framework
  - DMA fence
- Current i915 state of affairs
- Motivation and requirements
- Explicit sync in i915

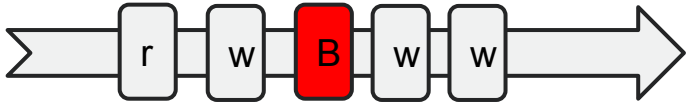
Questions to keep in mind

- What if…
  - you don’t have buffer handles or explicit buffer allocation?
  - you just pass the driver a pointer to a command stream with no additional info?
  - you’re using direct command submission from GL, CL, or media without kernel driver involvement?
  - you want to allow some user space scheduling in your display server (e.g. Wayland, SurfaceFlinger)?
- How do I…
  - debug performance problems or lockups?
  - synchronize execution between different hardware blocks?

Block devices

- I/O barriers on storage used for things like journaling filesystems
  - Write metadata, barrier metadata, write data, or similar
  - Tough to implement on some storage systems due to lack of physical medium flush
  - Not exported as a separate object for IPC or inter-driver sync
  - Exists only in I/O stream for targeted block device

Block devices (cont)

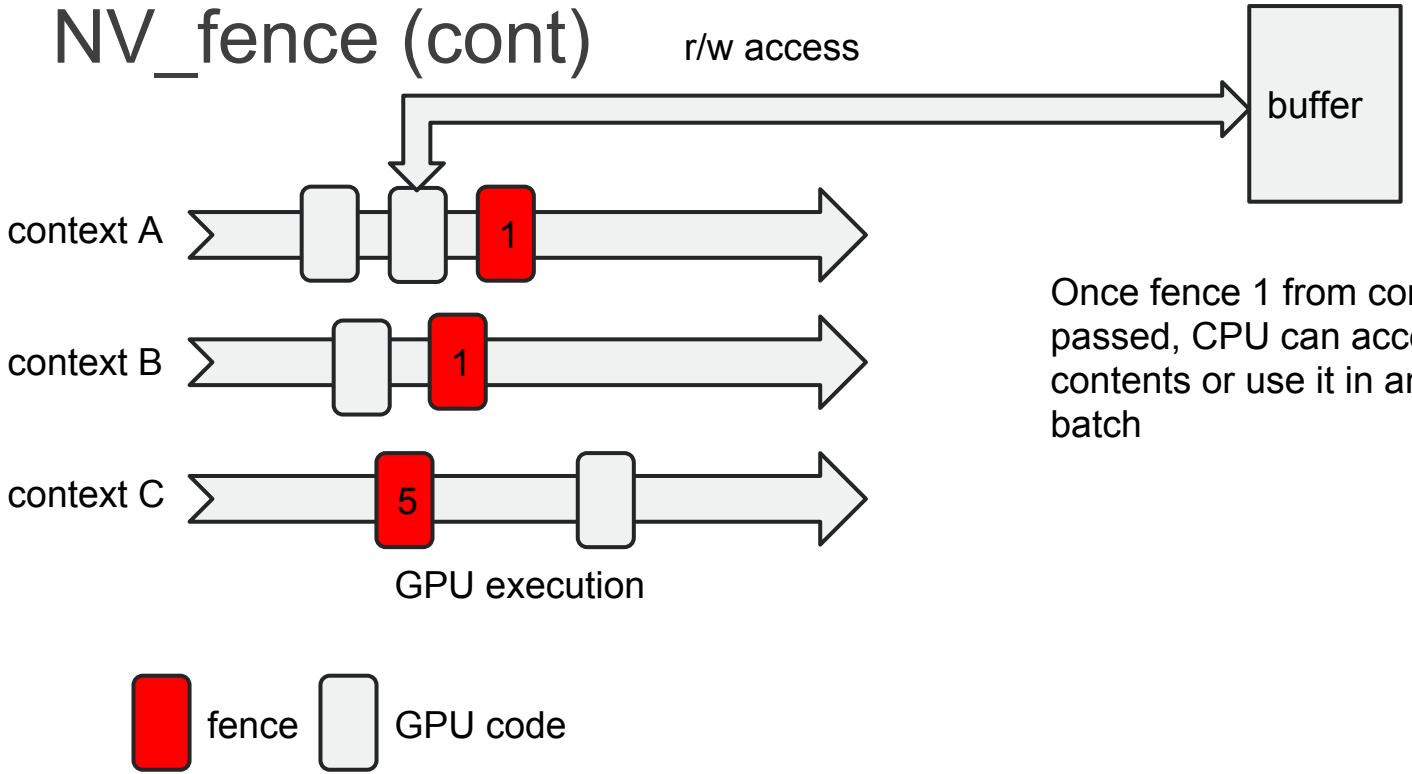


r/w stream from app. Previous read and write must complete prior to later writes due to barrier.

NV_fence

- Ancient history - added to nVidia's GL 1.2.1 circa 2000
- Extended GL with a "partial finish" mechanism
- Useful for coordinating access to buffers shared between CPU and GPU without doing a glFinish() on a whole bunch of commands

NV_fence (cont)

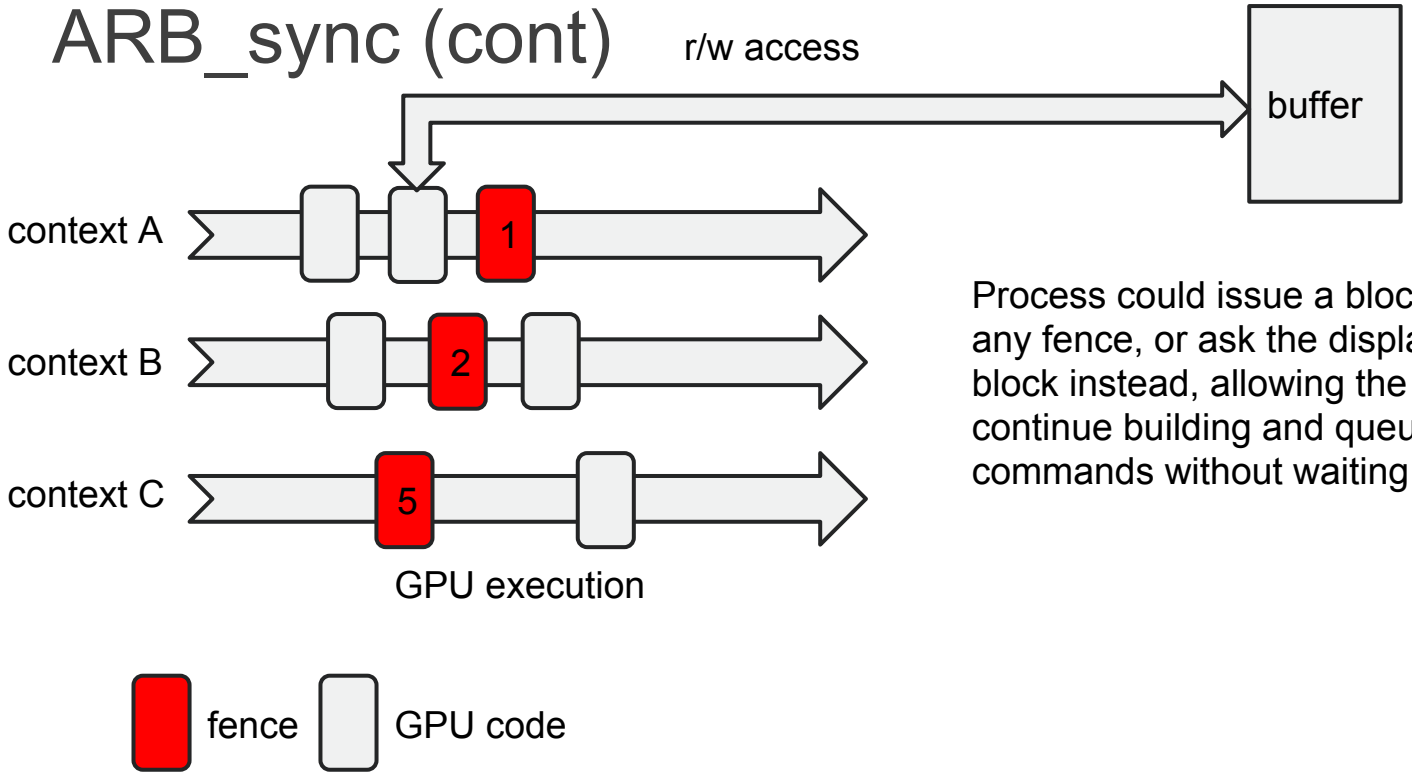


Once fence 1 from context A has passed, CPU can access buffer contents or use it in another batch

ARB_sync

- Slightly less ancient - added to GL 3.2 circa 2009
- Similar to NV_fence with some changes
- Adds client/server distinction, allowing client to continue running while server blocks for completion
- Namespace shared across contexts
- Again, useful for CPU/GPU memory sharing situations

ARB_sync (cont)



Process could issue a blocking wait on any fence, or ask the display server to block instead, allowing the process to continue building and queuing commands without waiting.

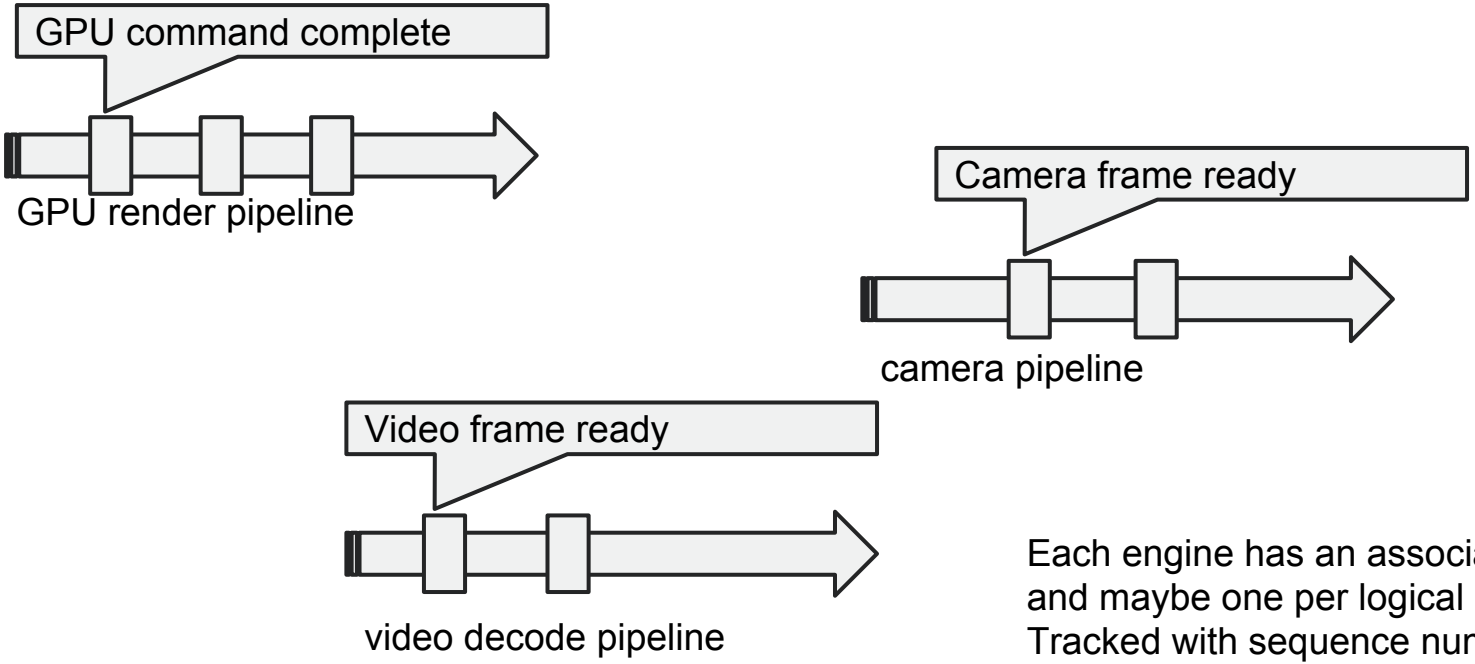
EGL_native_fence_sync

- Added by Android folks at Google circa 2012
- Designed to sit on top of underlying OS sync object support
- Extends EGL_fence_sync with underlying FDs
- Uses Android Sync Framework underneath on Android

Android Sync Framework

- Added to Android, currently in staging branch
- Designed to support multiple kernel drivers
- Allows inter-process and inter-device synchronization
- Exposes userland ABI for waiting on and merging sync fences, as well as debug
- Actual sync fences created and exported by individual drivers
- Internals use one “timeline” per command streamer or logical engine in each device/driver (e.g. render engine batches, display flips/vblanks, camera frames)

Android Sync Framework (cont)



Each engine has an associated timeline, and maybe one per logical context as well. Tracked with sequence numbers or some other hardware status indicator.

DMA fences

- Upstream solution (thanks Rob & Maarten!)
- Comparable to Android Sync Framework internals
- Simplified to a single fence struct with signaling and other callbacks
- Used in nouveau, radeon, and other drivers for internal command tracking
- Replaces a lot of similar code across drivers for seqno & batch tracking

Current i915 status

- Doesn’t use DMA fences
- All synchronization is implicit
  - Except in Android devices, which have sync framework support
- Sync is done using buffers
- Submission is also ordered, no scheduling (yet)
- Easy for userspace to use, but otoh easy to add bubbles to the pipeline
- Buffers can be used for explicit sync using buffer busy queries (see SNA) and buffer sharing
  - Downside is extra complexity for shared buffers, as you don’t want to fully synchronize on those

Explicit synchronization

- Buffer independent sync allows for the items above
- i915 plans (currently underway by Tvrtko)
  - add flag to execbuf to allow the return of a sync fence
  - sync fence will support Android Sync Framework ABI
  - internals will use DMA fence objects
  - other entry points (page flip, mode set) will optionally return sync fences as well
  - allows for asynchronous mode sets and flips with contingent completion
  - execbuf and other entry points will take sync fences to allow for internal sync and good pipeline utilization
  - GPU scheduler will be added as well, futher re-ordering requests relative to current behavior
- DRI/i965
  - ARB_sync could be implemented in terms of sync fences

Questions answered

- What if…
  - you don’t have buffer handles? Add a sync fence to your command stream.
  - you just pass the driver a pointer to a command stream with no additional info? Get a sync fence back from the command submission.
  - you’re using ring3 direct submission without kernel driver involvement? Request a sync fence from the kernel driver when needed.
  - you want to allow some user space scheduling in your display server (e.g. Wayland, SurfaceFlinger)? Send your sync fences to the display server, allowing it to intelligently pick buffers to use and schedule work.
- How do I…
  - debug performance problems or lockups? Track sync fences between processes and in the kernel.
  - synchronize execution between different hw blocks? Use sync fences in userspace and/or in the kernel.

Q & A