EXPLICIT SYNCHRONIZATION

Lauri Peltonen

XDC, 8 October, 2014
WHAT IS EXPLICIT SYNCHRONIZATION?

- Fence is an abstract primitive that marks completion of an operation
- Implicit synchronization
  - Fences are attached to buffers
  - Kernel manages fences automatically based on buffer read/write access
  - Currently used by DRM (dma-buf fences)
- Explicit synchronization
  - Fences are passed around independently
  - Kernel takes and emits fences to/from user space when submitting work
  - Currently used on Android (sync fence fd’s)
ADVANTAGES

- Improved performance of bindless graphics APIs
- Better alignment with user space graphics APIs
- Allow parallel processing of user space suballocations
- Fits in nicely with explicit buffer handoffs
Bindless graphics and Compute APIs allow building very large working sets that any given command buffer can reference.

- References can be by runtime-generated virtual address rather than slots or enums.

These working sets can be shared across multiple contexts or command queues.

- Implicit sync may force serialization in these cases.

Locking and updating fences for every active buffer is costly.

- Working set sizes can be thousands of buffers.
Developers are demanding explicit control of the driver behavior and hardware whenever possible.

Current Generation OpenGL is defined in terms of explicit synchronization:
- EGLSync
- GLSync

“Hidden” ordering dependencies and stalls because of implicit sync are at odds with these design philosophies.
User space drivers and applications use suballocation for performance reasons

- By definition, kernel has no visibility into this process

Operations on separate portions of a buffer should be allowed to proceed in parallel

- Even if they reside in one kernel-visible buffer
Modern processors have many specialized engines
- Video processing
- 3D/2D graphics
- CPU cores

Each of these may have its own caches, memory compression engines, or other specialized memory access quirks

When buffers are shared between them, engine-specific state transitions may be needed
- May be costly operations. May be difficult to perform just-in-time.
- Simplest solution is for user space to request them explicitly
- Might as well do explicit synchronization in the same code path
IMPLICIT SYNC EXAMPLE
IMPLICIT SYNC EXAMPLE

nouveau_pushbuf_kick(push1, chan1);

for (each buffer in working set)
    acquire ww mutex
for (each buffer in working set)
    program wait fence cmd
submit work
for (each buffer in working set) {
    store fence
    release ww mutex
}
IMPLICIT SYNC EXAMPLE

nouveau_pushbuf_kick(push1, chan1);

// push2 has no dependencies, but kernel enforces a wait
nouveau_pushbuf_kick(push2, chan2);

waiting

1

2
IMPLICIT SYNC EXAMPLE

nouveau_pushbuf_kick(push1, chan1);

// push2 has no dependencies, but kernel enforces a wait
nouveau_pushbuf_kick(push2, chan2);

// push2 depends on push1 only, but user space cannot
// express that to kernel
nouveau_pushbuf_kick(push3, chan3);

nouveau_pushbuf_kick(
    struct nouveau_pushbuf *push,
    struct nouveau_object *chan)
EXPLICIT SYNC EXAMPLE

Channel 1  Channel 2  Channel 3
EXPLICIT SYNC EXAMPLE

int fence1 = -1;
nouveau_pushbuf_kick_fence(push1, chan1, -1, &fence1);
// now fence1 == 1

nouveau_pushbuf_kick_fence(
    struct nouveau_pushbuf *push,
    struct nouveau_object *chan,
    int waitFenceFd,
    int *emitFenceFd)
EXPLICIT SYNC EXAMPLE

```
int fence1 = -1;
nouveau_pushbuf_kick_fence(push1, chan1, -1, &fence1);
// now fence1 == 1

int fence2 = -1;
nouveau_pushbuf_kick_fence(push2, chan2, -1, &fence2);
// now fence2 == 2
```

nouveau_pushbuf_kick_fence(
    struct nouveau_pushbuf *push,
    struct nouveau_object *chan,
    int waitFenceFd,
    int *emitFenceFd)
EXPLICIT SYNC EXAMPLE

```
int fence1 = -1;
nouveau_pushbuf_kick_fence(push1, chan1, -1, &fence1);
// now fence1 == 1

int fence2 = -1;
nouveau_pushbuf_kick_fence(push2, chan2, -1, &fence2);
// now fence2 == 2

// the last operation depends on only
nouveau_pushbuf_kick_fence(push3, chan3, fence1, NULL);
```

```
struct nouveau_pushbuf_kick_fence(
    struct nouveau_pushbuf *push,
    struct nouveau_object *chan,
    int waitFenceFd,
    int *emitFenceFd)
```
EXPLICIT SYNC EXAMPLE

int fence1 = -1;
nouveau_pushbuf_kick_fence(push1, chan1, -1, &fence1);
// now fence1 == 1

int fence2 = -1;
nouveau_pushbuf_kick_fence(push2, chan2, -1, &fence2);
// now fence2 == 2

// the last operation depends on 1 and 2
int merged = sync_merge(fence1, fence2);
nouveau_pushbuf_kick_fence(push3, chan3, merged, NULL);

nouveau_pushbuf_kick_fence(
    struct nouveau_pushbuf *push,
    struct nouveau_object *chan,
    int waitFenceFd,
    int *emitFenceFd)
RESIDENCY AND PINNING

- When we need to swap out or unmap a buffer, we need to wait until it is no longer accessed by hw

- This is not the perf-critical case, so we can be conservative in order to optimize the critical path. For example, on Nouveau:
  - Store one fence to channel vm at each submit
  - Use that fence when evicting or unmapping buffers
  - No need to lock / update fences to every buffer individually at submit?

- All this is driver specific logic, not common DRM
PATH FROM IMPLICIT SYNC -> EXPLICIT SYNC

- No need to disrupt existing model
  - If a particular device is happy with implicit sync, it can keep using it
- Allow kernel and user space drivers that prefer explicit to opt-in:
  - Allow user space to handle intra-driver synchronization explicitly
  - Allow user space to associate synchronization primitives with buffers for backwards compatibility with current APIs and drivers
  - Move towards tracking working sets rather than individual buffers for object lifetime/work completion/paging purposes
THANKS!

- drivers/staging/android-sync.c

- [RFC] Explicit synchronization for Nouveau (+ RFC patches)
  - dri-devel@lists.freedesktop.org, nouveau@lists.freedesktop.org

- Let’s discuss more over lunch/dinner!
BACKUP
Circular dependencies can be avoided, if fences are only generated in kernel when work is submitted

- This guarantees that user space cannot ask kernel to wait for a fence whose work will be submitted later

Deadlocks can be avoided, if additionally all submitted work completes in finite time

- This assumption might fail for implicit fences also
- Timeout mechanisms
EXPLICIT SYNC VS. ANDROID SYNC FD’S

- Could also be a process local handle?
  - But should support conversion to and from Android sync fd’s