DMA-BUF Developments

John Stultz <john.stultz@linaro.org>
Sumit Semwal <sumit.semwal@linaro.org>
ION Destaging via DMA-BUF Heaps
ION History

- Introduced in Android ICS (4.0) 2011!
  - Pushed to staging ~2013
- Generic buffer sharing for multiple device pipelines
  - Trying to replace per SoC/gpu custom buffer sharing implementations
- Motivated the development of DMA-BUFs upstream & became an early user
- Pseudo-constraint solving interface
  - Bitfield of heap types you’d accept, and ION will allocate from one of those
- All ION heaps tied to common ION dma-buf exporter logic
  - Resulted in each vendor hacking ION common code for their custom heaps
- Also gave vendors a little too much flexibility w/ vague interface
- Fair amount of DMA-API abuse (would work on ARM, but no promises elsewhere)
- Tried to solve a lot of hard problems in one common subsystem
  - Made it extra difficult to upstream
DMA-BUF Heaps

● Part of work of Destaging ION
  ○ Not trying to solve all of what ION set out to do
  ○ Focusing on a userland interface for allocating specific types of dma-bufs

● Each heap is its own dmabuf exporter
  ○ Just provides a shared userland allocation interface
  ○ Each heap driver gets its own chardev, no multiplexing heaps through one interface
    ■ /dev/dma-heaps/system
    ■ /dev/dma-heaps/<your heap name here>
  ○ Helper functions to avoid duplication where possible
DMA-BUF Heaps TODOs

● Secure heaps (can we handle all types? - Do we need to?)
  ○ Is “Secure” really just a generic flag we need on existing heaps?
  ○ Should “Secure” be more than a signal bit flag, but a domain-id?
    ■ But how does userland figure out the domain-ids?
  ○ Always free to create your own dmabuf exporter driver for a strange secure heap

● Vendors: Please move your ION code over to DMA-BUF Heaps once its upstream!
DMA-BUF Heaps Feedback

- Not much feedback on the interface itself
- Most of the complicated feedback on the buffer ownership rules and cache handling in the System and CMA heap helper functions
DMA-BUF Cache Handling
DMA-BUF Usage
DMA-BUF Usage

DMA-BUF buffer

Cache

Cache

Cache

Cache
DMA-BUF Cache Handling
DMA-BUF Cache Handling

NOTE: Device cache handling left to driver!
DMA-BUF Cache Handling

DMA-BUF buffer

Flush

Cache

Read
DMA-BUF Cache Handling

DMA-BUF buffer

Invalidate

Write
DMA-BUF Cache Handling

DMA-BUF buffer → Invalidate

DMA-BUF buffer → Write
DMA-BUF Cache Handling

DMA-BUF buffer
DMA-BUF Device-Usage (implicit)

DMA-BUF buffer

dma_buf_attach()
dma_buf_map_attachment()
dma_buf_unmap_attachment()
dma_buf_detach()
DMA-BUF CPU-Usage (implicit)

- `mmap/vmap/kmap()`
- `dma_buf_begin_cpu_access()`
- `dma_buf_end_cpu_access()`
- `munmap/vunmap/kunmap()`

DMA-BUF buffer
DMA-BUF Implicit Ownership/Signaling

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
munmap/vunmap/kunmap()

Cache

DMA-BUF buffer

dma_buf_attach()
dma_buf_map_attachment()
dma_buf_unmap_attachment()
dma_buf_detach()
DMA-BUF Implicit Ownership/Signaling

Note: Sort of odd irregular symmetry!

mmap/vmap/kmap() would seem logically closer to dma_buf_map_attachment()!
DMA-BUF Implicit Ownership/Signaling

```
DMA-BUF buffer

mmap/vmap/kmap()

dma_buf_begin_cpu_access()

dma_buf_end_cpu_access()

munmap/vunmap/kunmap()


dma_buf_attach()

dma_buf_map_attachment()


dma_buf_unmap_attachment()


dma_buf_detach()
```
DMA-BUF Implicit Ownership/Signaling

 DMA-BUF buffer

- `dma_buf_begin_cpu_access()`
- `dma_buf_end_cpu_access()`
- `munmap/vunmap/kunmap()`

- `mmap/vmap/kmap()`
- `dma_buf_unmap_attachment()`
- `dma_buf_detach()`

- `dma_buf_attach()`
- `dma_buf_map_attachment()`
DMA-BUF Implicit Ownership/Signaling

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
munmap/vunmap/kunmap()

dma_buf_attach()
dma_buf_map_attachment()
dma_buf_unmap_attachment()
dma_bufDetach()
DMA-BUF Implicit Ownership/Signaling

```
- mmap/vmap/kmap()
- dma_buf_begin_cpu_access()
- dma_buf_end_cpu_access()
- munmap/vunmap/kunmap()

- dma_buf_attach()
- dma_buf_map_attachment()
- dma_buf_unmap_attachment()
- dma_buf_detach()
```
DMA-BUF Implicit Ownership/Signaling

- mmap/vmap/kmap()
- dma_buf_begin_cpu_access()
- dma_buf_end_cpu_access()
- munmap/vunmap/kunmap()
- dma_buf_attach()
- dma_buf_map_attachment()
- dma_buf_unmap_attachment()
- dma_buf_detach()
DMA-BUF Implicit Ownership/Signaling

mmap/vmap/kmap()

dma_buf_begin_cpu_access()

dma_buf_end_cpu_access()

munmap/vunmap/kunmap()

Cache

DMA-BUF buffer

dma_buf_attach()

dma_buf_map_attachment()

dma_buf_unmap_attachment()

dma_buf_detach()}
DMA-BUF Implicit Ownership/Signaling

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
munmap/vunmap/kunmap()

DMA-BUF buffer

dma_buf_attach()
dma_buf_map_attachment()
dma_buf_unmap_attachment()
dma_buf_detach()
DMA-BUF Implicit Ownership/Signaling

- mmap/vmap/kmap()
- dma_buf_begin_cpu_access()
- dma_buf_end_cpu_access()
- munmap/vunmap/kunmap()

DMA-BUF buffer
- dma_buf_attach()
- dma_buf_map_attachment()
- dma_buf_unmap_attachment()
- dma_buf_detach()
DMA-BUF Implicit Ownership/Signaling

- `mmap/vmap/kmap()`
- `dma_buf_begin_cpu_access()`
- `dma_buf_end_cpu_access()`
- `munmap/vunmap/kunmap()`
- `dma_buf_attach()`
- `dma_buf_map_attachment()`
- `dma_buf_unmap_attachment()`
- `dma_buf_detach()`

DMA-BUF buffer
DMA-BUF Implicit Ownership/Signaling

DMA-BUF buffer

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
munmap/vunmap/kunmap()

dma_buf_attach()
dma_buf_map_attachment()
dma_buf_unmap_attachment()
dma_buf_detach()
Things to Note

• CPU Cache maintenance on dma_buf_map/unmap_attachment() really adds up!
  ○ For each frame * for each device * map/unmap
  ○ CPU may never touch the buffer!

• While implicit ownership diagrams might suggest exclusion, nothing enforces it!
  ○ dma_buf_begin_cpu_access() does not prevent dma_buf_map_attachment()
  ○ Or vice-versa!
DMA-BUF API Rules

Is there not exclusion between cpu mapping and device mappings?

I’m concerned we need some guardrails on this or folks are going to mess this up.

We should avoid giving people footguns.
DMA-BUF API Rules

Whooo! Look what I can do with my footguns! <pew!> <pew!> <pew!>
DMA-BUF Fences

● Similar to Android Sync framework prior to it
● Provides method for **explicit** signaling
● Allows additional parallelization:
  ○ One device can map a buffer (but not use it) while another is still writing to it
  ○ Fence signal passes ownership

● Where as DMA-BUF exporters expected to do cache management in their dma_buf_ops functions
  ○ No fence hooks in dma_buf_ops!
  ○ Fences signaling or waiting code doesn’t do any cache management itself
  ○ Has to be handled explicitly by the driver around the signal/wait calls!

● Key point: dma_buf_map/unmap_attachment() stops being an ownership transfer!
  ○ Question: What does that mean for cache handling?
DMA-BUF cpu-usage (explicit)

mmap/vmap/kmap()
dma_buf_begin_cpu_access()

dma_buf_end_cpu_access()
munmap/vunmap/kunmap()
DMA-BUF buffer

- dma_buf_attach()
- dma_buf_map_attachment()
- dma_fence_wait_()
- dma_fence_signal()
- dma_buf_unmap_attachment()
- dma_buf_detach()
DMA-BUF device-usage (explicit)

- dma_buf_attach()
- dma_buf_map_attachment()
- dma_fence_wait_*(*)
- dma_fence_signal()
- dma_buf_unmap_attachment()
- dma_buf_detach()

NOTE: dma_buf_attach/detach() and dma_buf_map/unmap_attachment() have dmabuf exporter hooks! Exporter can handle cache bits there.

Exporter has no hooks for fence_wait/signal()!
DMA-BUF Explicit Ownership/Signaling

Cache

DMA-BUF buffer

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
munmap/vunmap/kunmap()

dma_buf_attach()
dma_buf_map_attachment()
dma_fence_wait_*()
dma_fence_signal()
dma_buf_unmap_attachment()
dma_buf_detach()
DMA-BUF Explicit Ownership/Signaling

This provides more logical symmetry!

mmap/vmap/kmap() mirror
dma_buf_map_attachment()
begin_cpu_access() mirrors
dma_fence_wait()
DMA-BUF Explicit Ownership/Signaling

mmap/vmap/kmap()

dma_buf_begin_cpu_access()

dma_buf_end_cpu_access()

munmap/vunmap/kunmap()

dma_fence_signal()

dma_buf_unmap_attachment()

dma_buf_detach()

dma_buf_map_attachment()

dma_fence_wait_*()
DMA-BUF Explicit Ownership/Signaling

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
munmap/vunmap/kunmap()

dma_fence_signal()
dma_buf_unmap_attachment()
dma_buf_detach()
dma_buf_map_attachment()

dma_fence_wait_*(

dma_buf_attach()
DMA-BUF Explicit Ownership/Signaling

DMA-BUF buffer

- mmap/vmap/kmap()
- dma_buf_begin_cpu_access()
- dma_buf_end_cpu_access()
- munmap/vunmap/kunmap()
- dma_buf_attach()
- dma_buf_map_attachment()
- dma_fence_wait_*( )
- dma_fence_signal()
- dma_buf_unmap_attachment()
- dma_buf_detach()
DMA-BUF Explicit Ownership/Signaling

```
DMA-BUF buffer

- dma_buf_attach()
- dma_buf_map_attachment()
- dma_fence_wait_*()
- dma_fence_signal()
- dma_buf_unmap_attachment()
- dma_buf_detach()

mmap/vmap/kmap()
- dma_buf_begin_cpu_access()
- dma_buf_end_cpu_access()
- munmap/vunmap/kunmap()

Cache
```
DMA-BUF Explicit Ownership/Signaling

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
munmap/vunmap/kunmap()

dma_buf_attach()
dma_buf_map_attachment()
dma_fence_wait_*()
dma_fence_signal()
dma_buf_unmap_attachment()
dma_buf_detach()
DMA-BUF Explicit Ownership/Signaling

DMA-BUF buffer
- `dma_buf_attach()`
- `dma_buf_map_attachment()`
- `dma_fence_wait_*()`
- `dma_fence_signal()`
- `dma_buf_unmap_attachment()`
- `dma_buf_detach()`

Cache

mmap/vmap/kmap()
- `dma Buf begin cpu access()`
- `dma Buf end cpu access()`
- munmap/vmunmap/kunmap()
DMA-BUF Explicit Ownership/Signaling

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
munmap/vunmap/kunmap()

dma_fence_wait_*(

dma_fence_signal()
dma_buf_unmap_attachment()
dma_buf_detach()

dma_buf_attach()
dma_buf_map_attachment()
DMA-BUF Explicit Ownership/Signaling

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
munmap/vunmap/kunmap()

 DMA-BUF buffer

dma_buf_attach()
dma_buf_map_attachment()
dma_fence_wait_*( )
dma_fence_signal()
dma_buf_unmap_attachment()
dma_bufDetach()
DMA-BUF Explicit Ownership/Signaling

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
munmap/vunmap/kunmap()
DMA-BUF Explicit Ownership/Signaling

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
munmap/vunmap/kunmap()

DMA-BUF buffer

dma_buf_attach()
dma_buf_map_attachment()  
dma_fence_wait_*()  
dma_fence_signal()  
dma_buf_unmap_attachment()  
dma_buf_detach()  

Cache
DMA-BUF Explicit Ownership/Signaling

DMA-BUF buffer

mmap/vmap/kmap()
dma_buf_begin_cpu_access()
dma_buf_end_cpu_access()
ummap/vunmap/kunmap()
dma_buf_attach()
dma_buf_map_attachment()
dma_fence_wait_*()
dma_fence_signal()
dma_buf_unmap_attachment()
dma_buf_detach()
DMA-BUF Explicit Ownership/Signaling

mmap/vmap/kmap()

dma_buf_begin_cpu_access()

dma_buf_end_cpu_access()

munmap/vunmap/kunmap()

dma_buf_attach()

dma_buf_map_attachment()

 dma_fence_wait_*()

 dma_fence_signal()

 dma_buf_unmap_attachment()

 dma_buf_detach()
DMA-BUF Explicit Ownership/Signaling

- mmap/vmap/kmap()
- dma_buf_begin_cpu_access()
- dma_buf_end_cpu_access()
- munmap/vunmap/kunmap()

DMA-BUF buffer

- dma_buf_attach()
- dma_buf_map_attachment()
- dma_fence_wait_*(x)
- dma_fence_signal()
- dma_buf_unmap_attachment()
- dma_buf_detach()
Now there’s multiple ways to handle things?

- It's unclear that explicit and implicit ownership/signaling can coexist
  - Not advised to mix and match
  - But what is a generic DMA-BUF exporter to do?

- Exporters need to be written to support one or the other
  - Or we’re doing unnecessary cache flushing all over the place!

- Usage rules seem tied to DMA-BUF exporter’s implementation
  - Sort of fine if dma-buf from that exporter only used with a single driver
  - But sort of falls apart in the many-device pipeline

- Does this mean DMA-BUF isn’t really a neutral interchange format?

- What are the rules here?!
Now there’s multiple ways to handle things?

- Its unclear that explicit and implicit ownership/signaling can coexist
  - Not advised to mix and match
  - But what is a generic DMA-BUF exporter to do?

- Exporters need to be written to support one or the other
  - Or we’re doing unnecessary cache flushing all over the place!

- Usage rules seems tied to DMA-BUF exporter’s implementation
  - Sort of fine if dma-buf from that exporter only used with a single driver
  - But sort of falls apart in the many-device pipeline

- Does this mean DMA-BUF isn’t really a neutral interchange format?

- What are the rules here?! \[ THIS IS SPARTA! GRAPHICS! \]
DMA-BUF Ownership Statemachin (by Andrew Davis)
DMA-BUF Ownership Statemachine (by Andrew Davis)

Not fully complete, and doesn’t even consider the explicit signaling cases!

https://dreampuf.github.io/GraphvizOnline/
More problems

So far I’ve treated cpu mappings (mmap/kmap/vmap) as all the same

- But on some cases vmaps have to be flushed/invalidated separately!
- Can be handled in dma_buf_op for the implicit case, not clear it works in explicit
DMA-BUF Cache Management Optimizations

If we did have clearer rules, we could add some extra metadata to track “dirtied-by” state or “last-used” state.

Then only do lazy cache flushes/invalidations when the owner actually changes:
- `cpu_begin_access()`: If last user was cpu, don’t do anything
- `dmabuf_map_attachment()`: If last user was device, skip cache flushed

Trouble is: this doesn’t help with explicit signaling, as it breaks the rules.
What to do?

● Consolidate to always using explicit signaling?
  ○ Avoid mixing styles?
  ○ Rob Clark mentioned: “ioctl to retrieve an implicit fence”
  ○ Maybe provide fence hooks in dma_buf_ops to allow for dmabuf exporter?

● Consistent buffer “ownership” rules?
  ○ Maybe have begin/end_device_access() hooks?
  ○ Issue: This has trouble with device’s waiting on hardware fences, which may start DMA after fence fires, before control goes back to the driver.

● Provide some way to evaluate correctness?
  ○ If not through enforced exclusion, some other tool?

● Break the DMA-API assumption that cpu is the normal owner of DMA-BUF memory
  ○ dma_buf_begin/end_cpu_access() calls already hint at this.
  ○ Only do cpu cache handling on begin/end_cpu_access calls?
Linux Plumbers Conference Topics
Partial Cache Invalidations

- Desire to invalidate part of a buffer that has been modified, rather than the entire thing
- Example was when there is meta data in the buffer that needs to be flushed.
- Proposal: Range flushes
  - `dma_clean_range()`? `dma_flush_range()`?
  - Need clear articulation of the need to the community *and* an upstream user of the code.
- At LPC, Android team said they already have an api internally for range flushes
Kernel Graphics Buffers

- Built on-top of DMA-BUFs & similar in usage
- But contain extra meta-data related to graphics buffers
- Sounds a bit like GEM Buffers! (though via a fd rather than gem handle?)
- From LPC:
  - For most cases, using drm_fourcc for pixel format and drm format modifiers for layout specification seemed sufficient.
Imagine just wanting to invalidate a change to that top left corner.
Flush 2D Patterns

Imagine just wanting to invalidate a change to that top left corner (in blue)

Single Frame YUV420:

Position in byte stream:

Questions?

John Stultz <john.stultz@linaro.org>
Sumit Semwal <sumit.semwal@linaro.org>
Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org