Problem Statement

- Hardware places “constraints” on memory allocations because memory access logic contains limitations
- The limitations may depend on how the hardware is used
- Historically, these are accounted for by the hardware’s driver
- Problems arise when sharing memory between two devices or engines
  - No single driver aware of all limitations anymore
- How does one allocate memory compatible with a given set of devices?
  - iGPU/dGPU “PRIME” sharing, video encode/decode/capture + GPU processing, etc.
Definitions: Constraint

- An atom defining a single limitation of the memory access capabilities exposed by a device or engine
- Examples: address or pitch alignment, maximum buffer size or pitch, locality, contiguity, bank placement or interleaving
- Atomic: each constraint expresses a non-decomposable fully defined limitation
- Can be mergeable
  - (min pitch alignment 4) + (min pitch alignment 6) = (min pitch alignment 12)
Definitions: Usage

- Description of data stored in and operations to be performed on memory
- Examples: Pixel/Texel format, texturing, rendering, video encode/decode/capture, display at particular resolution and rotation
- May vary in detail depending on API or driver
Definitions: Constraint Set

- A list of individual constraints imposed on memory by a given usage
- Can be merged
- Merging multiple constraint sets results in a constraint set expressing the constraints imposed by the superset of the usage associated with each input constraint set
- Merging two constraint sets results in a set containing at most the sum of the number of elements in each input set
  - i.e., merging must not in itself generate new constraints

- (surf align >= 64)
- (pitch align >= 64, surf align >= 128) → (surf align >= 128, pitch align >= 64)
Design Goals

- Integrate with existing ecosystem
  - OpenGL, Vulkan, DRM-KMS, GBM, V4L, etc.
- Extensible
- Focus on constraint resolution alone
  - Not trying to solve allocator choice, IOMMU discovery/programming, etc.
High-Level Design Proposal

- Leverage existing API mechanisms to describe usage
  - Vulkan image properties, OpenGL formats/texture parameters, DRM-KMS planes, etc.
- Query constraints from existing APIs for a given usage in that API
  - Very similar to existing format modifier reporting and query APIs
- Constraints are reported separately for each modifier supported by a usage
  - E.g., set of constraints generated for each format+modifier[+external_only] tuple in OpenGL
- Provide logic to merge constraint sets
- Pass resulting constraint set to allocation API of choice at allocation time
Example Usage: Rendering and Displaying

KMS → Merge

Display constraints

OpenGL → Merge

Render constraints

GBM

Display + render constraints
Example Usage: Rendering and Displaying

KMS → Merge → GBM

OpenGL → Merge → GBM

(local video memory, 1k address alignment)

(2k address alignment)

(local video memory, 2k address alignment)
Case Study 1: Pitch Alignment Requirements

- Min pitch alignment: NV = 32 or 256, AMD* = 256, Intel* = 1, Bpp, or 64
- Want a surface that can be displayed, textured, and rendered to on all GPUs
- Query list of modifiers for each device, specifying this usage
  - Pointless here, given the layout is going to be linear
- Intersect modifiers
- Query and merge constraints for each usable modifier, specifying usage again
  - Merge operation for this type of constraint: Find least common multiple of all values
Case Study 1: Lessons

- Applicable to all minimum alignment: offset, pitch, width, height, etc.
- Generalizes to maximums and minimums by using other merge operators
- Relatively simple to code up and visualize
Case Study 2: Local (Video) Memory

- Discrete NVIDIA display engines can only scan out from video memory
- Naive solution is simple: a value-less constraint of type “local”
- Problem 1: Local to what? Could be many devices with local memory
- Problem 2: How local? Some devices have multiple local heaps of memory
- Proposed Solution: Constraint contains a list of heap supported IDs
- Each device can expose multiple heap IDs
- Heap IDs could be namespaced by subsystem (dma-buf heaps, DRM, etc.)
- Merge two locality constraints by intersecting heap list
Case Study 2: Lessons

● Constraint merging can fail: The intersected list may be empty!

● Likely need a “regular memory” heap or heaps that are universally recognized
  ○ Not necessarily universally supported
  ○ Should lack of a heap constraint imply compatibility only with the “regular memory” heap?

● The development of a good Heap ID mechanism is a large task by itself
Case Study 3: Unknown Constraints

- Design disperses constraint knowledge throughout system components
- The system components cannot realistically be version locked
  - E.g., old kernel, new userspace, or vice-versa
- At some point, a component will encounter a constraint it is not aware of
- How is this handled?
- Ignore it: Resulting allocation may not be compatible with usage
- Fail the merge or allocation: Fails when “implicit” constraints become explicit
  - E.g., all allocations go in video memory in existing code, new code relies on “local” constraint
Case Study 3: Unknown Constraints (Continued)

- Implicit constraint example: local/video memory

- Prior to constraints, GPU driver assumes allocations go in video memory
  - The local memory constraint is hence made implicit by this allocator

- After introduction of local memory constraint, allocator drops assumption
  - Only allocates memory in video memory when it sees the constraint

- Old driver component unaware of constraint allocates, expecting video memory
  - Bug: No constraint, so it gets non-video memory. Lack of forward/backward compatibility
Case Study 3: Unknown Constraints (Continued)

- Proposal: Version all components of the system
- Each constraint has an associated version it was introduced at
- Version of each constraint set included as a special-case, required constraint
  - Merge op is min()
- Allocators merge their version into the constraint set at allocation time
- Allocators preserve prior implicit constraints if constraint set version < explicit constraint version
- Ignore constraints > (resulting constraint set version)
Case Study 3: Lessons

- Both backwards and forwards compatibility are important
- Ignoring unknown constraints should produce behavior no worse than using an older version of the component that expressed the constraint
Low-Level Design Proposal

- Constraint sets are variable-length lists of `uint64_t` items
- First list item is a header with a “type” field and size
  - Explicit length necessary for parsing/skipping unknown constraint types
- Most constraints will be vendor-agnostic
- Implement merging logic and helper functions/macros in “shared library”
  - Small, but needs a home. Current proposal: header-only library in kernel uapi headers.
- Recommend guess-and-check logic for choosing an allocator
  - DRM devices and userspace APIs that use them will likely be good guesses
Low-Level Design Proposal: EGL/OpenGL

```c
int eglQueryDmaBufConstraintsEXT(EGLDisplay dpy,
                                  EGLint format,
                                  EGLuint64KHR modifier,
                                  EGLBoolean external_only,
                                  EGLint max_set_size,
                                  void *constraints,
                                  EGLint *set_size);
```
struct VkImageDrmConstraintSetFormatProperties { // Extends VkImageFormatProperties2
    VkStructureType sType;
    void* pNext;
    uint32_t drmConstraintSetMaxSize;
    uint32_t drmConstraintSetSize;
    void* pDrmConstraintSet;
};

struct VkMemoryDrmConstraintSetAllocateInfo { // Extends VkMemoryAllocateInfo
    VkStructureType sType;
    const void* pNext;
    uint32_t drmConstraintSetSize;
    void* pDrmConstraintSet;
};


struct VkImageDrmConstraintSetCreateInfo {  // Extends VkImageCreateInfo

    VkStructureType sType;
    const void* pNext;
    uint32_t drmConstraintSetSize;
    const void* pDrmConstraintSet;

};

Low-Level Design Proposal: Vulkan (continued)
Low-Level Design Proposal: DRM-KMS

- Usage described by test-only atomic commits
  - e.g. “rotation” may change constraints
- Inputs: framebuffer width, height, format, modifier
- Output: set of constraints

- drmModeAddFB2WithModifiers with zero bo_handles gives a test-only FB
- New OUT_FB_CONSTRAINTS_PTR blob property used to collect constraints

- Maybe need to pass a list of modifiers as input?
struct gbm_bo *

gbm_bo_create_with_constraints(struct gbm_device *gbm,
    uint32_t width, uint32_t height,
    uint32_t format,
    const uint64_t *modifiers,
    unsigned int modifier_count,
    const void **constraint_sets,
    const unsigned int *set_sizes);
Low-Level Design Proposal: Constraint Merge Library

```c
static __u64
drm_constraint_merge(void *set_out, __u64 out_size,  
                      const void *set_a, __u64 a_size,  
                      const void *set_b, __u64 b_size);
```

WIP code here: https://gitlab.freedesktop.org/emersion/drm-constraints
Workshop

Tomorrow, 15:35 UTC

https://xdc2020.x.org/event/9/contributions/634/

Questions?
Bibliography

- **Unix Device Memory**, James Jones, XDC 2016: [https://www.x.org/wiki/Events/XDC2016/Program/Unix_Device_Memory_Allocation.pdf](https://www.x.org/wiki/Events/XDC2016/Program/Unix_Device_Memory_Allocation.pdf)
- **gralloc.h**, The Android Open Source Project: [https://android.googlesource.com/platform/hardware/libhardware/+/froyo/include/hardware/gralloc.h](https://android.googlesource.com/platform/hardware/libhardware/+/froyo/include/hardware/gralloc.h)
- **Destaging ION**, Marta Rybczyńska (2019): [https://lwn.net/Articles/792733/](https://lwn.net/Articles/792733/)
- **gbm: add gbm_[bo,surface]_create_with_modifiers2**, mesa MR: [https://gitlab.freedesktop.org/mesa/mesa/-/merge_requests/3197](https://gitlab.freedesktop.org/mesa/mesa/-/merge_requests/3197)
- **BoF: Negotiating DMA-BUF Heaps**, Ezequiel Garcia, Linux Plumbers Conference 2020: [https://linuxplumbersconf.org/event/7/contributions/818/](https://linuxplumbersconf.org/event/7/contributions/818/)
  - James’ summary: [https://paste.sr.ht/blob/1513715dbec4662ac5bcd63d754ce7fd44673a4e](https://paste.sr.ht/blob/1513715dbec4662ac5bcd63d754ce7fd44673a4e)
- Email archive for this talk: [https://lists.sr.ht/~emersion/drm-constraints/%3CiT7khYWdVd_qhl345mZnJ-dOu9yH5Oojrrjd5rbZLUz6B4O_0qEDuTd51f-n0CjKEWtbnP4cNjR0wENyYtJ8u6d_yaPIZeFNId4KMFNbnno%3D%40emersion.fr%3E](https://lists.sr.ht/~emersion/drm-constraints/%3CiT7khYWdVd_qhl345mZnJ-dOu9yH5Oojrrjd5rbZLUz6B4O_0qEDuTd51f-n0CjKEWtbnP4cNjR0wENyYtJ8u6d_yaPIZeFNId4KMFNbnno%3D%40emersion.fr%3E)
- Notes for this talk: [https://paste.sr.ht/~emersion/e10214caff180f1d8f85f560a3c2beac6a96bc0c](https://paste.sr.ht/~emersion/e10214caff180f1d8f85f560a3c2beac6a96bc0c)