Qualcomm

drm/msm VM_BIND

Rob Clark

Principal Engineer, Qualcomm Technologies, Inc.

@qualcomm

VM - A process's virtual memory space

- One per drm device file `open()` – typically one per process
  - Plus some internal ones (dpu, gmu, etc)

- Kernel managed – kernel assigns iova for GEM obj mapped in VM
  - Legacy userspace
  - Also kernel internal VMs

- User managed – opt-in for userspace to manage the VM via VM_BIND
  - Must opt in before first GEM obj is mapped

- Data structure: `msm_gem_vm` extends `drm_gpuvm`

VMA (aka VA) – Virtual Memory Area

- Represents pages mapped in a VM
  - “a contiguous range of virtual addresses that have the same permission flags, and are backed by the same object”
- Can be backed by ranges of a GEM obj (BO)
  - A GEM obj can back multiple VMAs in a VM
- Or NULL mappings backed by PRR page
  - Partially Resident Region
  - Reads – return zero
  - Writes – dropped
- Data structure: `msm_gem_vma` extends `drm_gpuva`

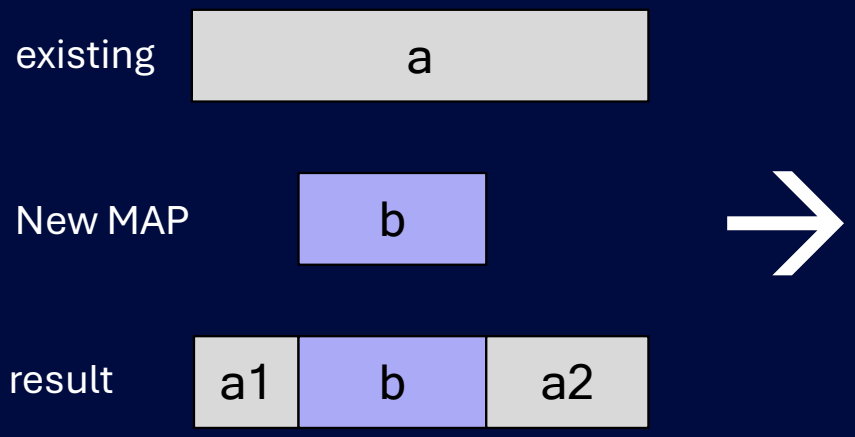
VM_BIND submitqueue

- Normal submitqueue is for SUBMIT ioctls (cmd execution)
- VM_BIND submitqueue is new for VM_BIND ioctls
- Userspace can create as many as it wants
  - Typically one for vk sparse queue
  - And one for internal use
- Synchronization
  - Ops enqueued on a submitqueue happen in FIFO order
  - Can use fence fd’s and/or syncobjs for cross-queue synchronization

VM_BIND ops

- VM_BIND ioctl contains 1 or more ops:
  - UNMAP / MAP / MAP_NULL
- MAP op takes a GEM handle + offset into that GEM obj
- All take an iova + range
- drm_gpuvm translates ops into discrete steps for creating/destroying VMAs
  - And corresponding pgtable updates
  - If an op overlaps an existing (or earlier) VMA, it can split (REMAP) or tear down (UNMAP) the existing VMA

VM_BIND ops → steps



1. REMAP step
   - Split ‘a’ into prev ‘a1’ and next ‘a2’ keeping the existing pgtable entries
   - Unmap a hole for ‘b’
2. MAP step
   - Insert new pgtable entries for ‘b’

drm_gpuvm conversion

- Replaces previous internal tracking of VMs and VMAs
  - Old scheme could only handle full GEM obj maps
  - Only one VMA per GEM obj per VM

- Hard vs Soft reference on GEM obj held by VMA
  - Previously the VMA implicitly was torn down when GEM obj freed
    - Avoids map/unmap on ever pageflip
  - But with drm_gpuvm, the VMA (vm_bo) holds a hard ref to GEM obj

  - Solution: deferred fb unpin!
    - As long as userspace holds some sort of handle to a GEM obj, defer unpin from the kms VM
    - `fb→prepare()` and `fb→cleanup()` take/drop `obj→vma_ref`
    - GEM handles and `dma_buf` fd's hold an extra ref to `obj→vma_ref`
    - Unmap from `kms→vm` deferred until `obj→vma_ref` drops to zero

drm_gpuvm conversion – locking order

- GEM obj lock vs VM lock (`vm->r_obj`)
  - To iterate VMs an obj is bound in, need to acquire obj lock first
  - To iterate/modify a VM, need to acquire VM lock first

- Solution: `drm_exec`!
  - Basically, a convenient wrapper for `ww_acquire_ctx`
  - Handles the deadlock/backoff/retry loop in a straightforward way
  - `msm_gem_lock_vm_and_obj()` for simple cases

All about preallocation!

- Shrinker vs. fence signaling path
  - Memory allocations can recurse into shrinker
  - Shrinker may need to wait on fences in order to reclaim memory
  - Ergo, allocations in fence signaling path can deadlock!
- Solution pt1: Bookkeeping done synchronously
  - Only the actual pgtable updates are done asynchronously
  - Allocating VMAs and updating VM is synchronous
- Solution pt2: Preallocate pgtable pages
  - We don’t know the state of the pgtables when the update is applied
  - We need to preallocate for worst case:
    - 4 lvl pgtable, PAGE_SIZE mapping -> need to preallocate 3 pages!!
  - Userspace can help by sorting ops by iova

In-place remap

- To know which VMAs to capture in devcoredump we rely on a VMA flag
  - `MSM_VM_BIND_OP_DUMP`
- Userspace sometimes needs to set this flag after after the VMA is created
  - VK_EXT_device_generated_commands
  - VK_EXT_descriptor_buffer

```
/*
 * Detect in-place remap.  Turnip does this to change the vma flags,
 * in particular MSM_VMA_DUMP.  In this case we want to avoid actually
 * touching the page tables, as that would require synchronization
 * against SUBMIT jobs running on the GPU.
 */
if (op->unmap.keep &&
    (arg->op->op == MSM_VM_BIND_OP_MAP) &&
    (vma->gem.obj == arg->op->obj) &&
    (vma->gem.offset == arg->op->obj_offset) &&
    (vma->va.addr == arg->op->iova) &&
    (vma->va.range == arg->op->range)) {
        /* We are only expecting a single in-place unmap+map cb pair: */
        WARN_ON(arg->kept);

        /* Leave the existing VMA in place, but signal that to the map cb: */
        arg->kept = true;

        /* Only flags are changing, so update that in-place: */
        unsigned orig_flags = vma->flags & (DRM_GPUVA_USERBITS - 1);
        vma->flags = orig_flags | arg->flags;

        return 0;
}
```

Debugging: vm_log

- Improper sync between VM_BIND and exec queues can be hard to debug!

- Modparam: `msm.vm_log_shift=N` (max 8)
  - Driver tracks ringbuffer of last 2^N updates per VM
  - Captured in devcoredump

Thank you

Follow us on:

For more information, visit us at qualcomm.com & qualcomm.com/blog