Bulk Moving Mechanism on LRU for DRM/TTM

Huang Rui (Ray Huang)
<ray.huang@amd.com>
AGENDA

Background

LINUX® GPU Kernel

GPUVM - GART memory setup (1-level)

GPUVM - 4-level paging

Per-VM Buffer Object

Eviction (Buffer Migration)

VM Key List Definitions

New List Operation for Bulk Moving

LRU Policy for Buffer Migration in Bulk Moving

Bulk Moving Approach in TTM

Bulk Moving use case in AMDGPU

Performance Improvement Data

Contribution

Q & A
Who am I?
- My name is Huang Rui (Ray), I am from AMD Linux® graphic driver team and focus on kernel driver and libdrm for new ASIC bring-up and new feature development several years.
- Patch work profile:
  - https://git.kernel.org/pub/scm/linux/kernel/git/rui/linux.git

Why we proposed the solution of bulk moving?
- Investigating performance with the F1 2017 game benchmark showed that the application caused a large number of buffers being created.
- The validation and LRU management of these buffers in the TTM and driver infrastructure was found to be non-optimal for this scenario.
- This led to a redesign of the buffer migration process in the code.
- This talk demonstrates the practical techniques to efficiently profile and analyze the scenario and identify the design changes needed to address it.
LINUX® GPU Kernel

- Linux® GPU Kernel for AMD
  - Device Init
  - Interrupt Handle
  - GMC (GFX memory controller)
  - PSP (Platform Security Processor)
  - Display
  - GFX
  - SDMA
  - MM block (UVD/VCE/VCN)
GART memory setup (1-level)

- **GART memory** is GPU visible system memory
  - Allocate GART table BO in the video memory for mapping to system memory.
  - GPU will read the data from the page table entries in the GART table BO to convert to the physical address (DMA bus address).
GPUVM - 4-level paging

- **VMID 0**
  - System context domain that only used by kernel mode.
  - GART table is created by 1-level paging (flat page table)

- **VMID 1 ~ 15**
  - Other context domain that used by user mode.
  - The page table is setup when the thread is created and is 4-level.
Per-VM Buffer Object

- **History:**
  - Each BO (too many) in the BO list that needs be validated during CPU bound games
  - Solution: decrease the work of BO list parser relevant.

- **New mechanism (Per-VM) that is to ensure the BO always valid for command submission.**
  - Add flag for UMD (Vulkan)
  - Share reservation object with VM root BO
  - Allow eviction and swap out when sharing same reservation
  - Ensure the Per-VM BO always valid
Eviction (Buffer Migration)

- Buffer migration approach:
/* BOs who needs a validation */
struct list_head evicted;

/* PT BOs which relocated and their parent need an update */
struct list_head relocated;

/* per VM BOs moved, but not yet updated in the PT */
struct list_head moved;

/* All BOs of this VM not currently in the state machine */
struct list_head idle;

/* regular invalidated BOs, but not yet updated in the PT */
struct list_head invalidated;

spinlock_t invalidated_lock;

/* BO mappings freed, but not yet updated in the PT */
struct list_head freed;
New List Operation For Bulk Moving

```c
/**
 * list_bulk_move_tail - move a subsection of a list to its tail
 * @head: the head that will follow our entry
 * @first: first entry to move
 * @last: last entry to move, can be the same as first
 *
 * Move all entries between @first and including @last before @head.
 * All three entries must belong to the same linked list.
 */

static inline void list_bulk_move_tail(struct list_head *head,
                                        struct list_head *first,
                                        struct list_head *last)
{
    first->prev->next = last->next;
    last->next->prev = first->prev;

    head->prev->next = first;
    first->prev = head->prev;

    last->next = head;
    head->prev = last;
}
```
Least Recently Used (LRU) algorithm is used for TTM on eviction (buffer migration)
Bulk Moving Approach in TTM

```c
void ttm_bo_bulk_move_lru_tail(struct ttm_lru_bulk_move *bulk)
{
    unsigned i;

    for (i = 0; i < TTM_MAX_BO_PRIORITY; ++i) {
        struct ttm_lru_bulk_move_pos *pos = &bulk->tt[i];
        struct ttm_mem_type_manager *mman;
        if (!pos->first)
            continue;
        dma_resv_assert_held(pos->first->base.resv);
        dma_resv_assert_held(pos->last->base.resv);
        mman = &pos->first->dev->mman[TTM_PL_TT];
        list_bulk_move_tail(&mman->lru[i], &pos->first->lru,
                             &pos->last->lru);
    }

    for (i = 0; i < TTM_MAX_BO_PRIORITY; ++i) {
        struct ttm_lru_bulk_move_pos *pos = &bulk->vram[i];
        struct ttm_mem_type_manager *mman;
        if (!pos->first)
            continue;
        dma_resv_assert_held(pos->first->base.resv);
        dma_resv_assert_held(pos->last->base.resv);
        mman = &pos->first->dev->mman[TTM_PL_VRAM];
        list_bulk_move_tail(&mman->lru[i], &pos->first->lru,
                             &pos->last->lru);
    }

    for (i = 0; i < TTM_MAX_BO_PRIORITY; ++i) {
        struct ttm_lru_bulk_move_pos *pos = &bulk->swap[i];
        struct list_head *lru;
        if (!pos->first)
            continue;
        dma_resv_assert_held(pos->first->base.resv);
        dma_resv_assert_held(pos->last->base.resv);
        lru = &pos->first->dev->glob->swap_lru[i];
        list_bulk_move_tail(lru, &pos->first->swap, &pos->last->swap);
    }
}
```

EXPORT_SYMBOL(ttm_bo_bulk_move_lru_tail);
Bulk Moving Use Case in AMDGPU

- **Legacy approach**
  - AMDGPU driver will move all PD/PT and Per-VM BOs into idle list. Then move all of them on the end of LRU list one by one. The result of this is that many BOs are moved to the end of the LRU again and again, which has a serious impact on performance.

- **Bulk Moving**
  - Collect all PD/PT and Per-VM BOs and bulk move them to the end of LRU list one time instead of one by one. This will reduce cost during the buffer moving.
### Performance Improvement Data

**GPU: Radeon™ RX Vega**
**Video Memory: 8G**
**System Memory: 16G**
**OS: Ubuntu 18.04 LTS**

<table>
<thead>
<tr>
<th></th>
<th>The Talos Principle (Vulkan)</th>
<th>Clpeak (OpenCL™)</th>
<th>BusSpeedReadback (OpenCL™)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>147.7 FPS</td>
<td>76.86 us</td>
<td>0.319 ms(1K) 0.314 ms(2K)</td>
</tr>
<tr>
<td>Original + WA (don’t move PT BOs on LRU)</td>
<td>162.1 FPS</td>
<td>42.15 us</td>
<td>0.254 ms(1K) 0.241 ms(2K)</td>
</tr>
<tr>
<td>Bulk Move</td>
<td>163.1 FPS</td>
<td>40.52 us</td>
<td>0.244 ms(1K) 0.252 ms(2K)</td>
</tr>
</tbody>
</table>

Bulk move will get the highest FPS and lowest latency.
Contribution

- Christian König <Christian.Koenig@amd.com>
  - He raised the original idea of bulk moving for the optimization of buffer migration.
  - I worked with him to deliver the completed solution in the kernel driver.

- Alex Deucher <alexander.deucher@amd.com>
  - Maintain and lead AMDGPU kernel driver to support the solution upstream.
  - Actively review and refine the quality of Linux® AMDGPU driver stack.
Thank You and Q&A
DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD’s Standard Terms and Conditions of Sale. GD-18