RDMA WITH GPU MEMORY VIA DMA-BUF

Jianxin Xiong

Intel Corporation
RDMA OVERVIEW
RDMA WITH SYSTEM MEMORY

▪ RDMA is “DMA + network”

<table>
<thead>
<tr>
<th>RDMA op</th>
<th>Initiator</th>
<th>Direction</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>DMA read</td>
<td>→</td>
<td>DMA write</td>
</tr>
<tr>
<td>Read</td>
<td>DMA write</td>
<td>←</td>
<td>DMA read</td>
</tr>
</tbody>
</table>

▪ DMA requires proper setup of the memory
  • Memory pages are “pinned”
  • Bus addresses are used
  • Usually done at the time of “memory registration”
  • For user space buffer in system memory
    • `get_user_pages()`
    • `sg_alloc_table()` / `sg_set_page()` / `sg_next()` / …
    • `dma_map_sg()`

OpenFabrics Alliance Workshop 2020
**RDMA WITH GPU MEMORY**

- **GPU memory is local**
  - The NIC driver can’t pin the memory directly
  - The NIC driver doesn’t know the DMA address

- **Cooperation between the NIC driver and the GPU driver is needed**

- **Peer-Direct from Mellanox**
  - Plug-in interface for kernel RDMA core
  - Each GPU driver provides a plug-in module
  - Plug-ins are queried one-by-one when memory is registered, until the ownership is claimed
  - Only available in MOFED

- **Can we have a non-proprietary upstream solution?**
  - Our proposal is to use dma-buf
DMA-BUF OVERVIEW
Dma-buf is a standard mechanism in Linux kernel for sharing buffers between different device drivers.
DMA-BUF API (EXPORTER)

- Create a new dma-buf object
  
  ```c
  struct dma_buf *dma_buf_export(const struct dma_buf_export_info *exp_info);
  ```

  ```c
  struct dma_buf_export_info {
    const char *exp_name;
    struct module *owner;
    const struct dma_buf_ops *ops;
    size_t size;
    int flags;
    struct dma_resv *resv;
    void *priv;
  };
  ```

- Associate with a file descriptor
  
  ```c
  int dma_buf_fd(struct dma_buf *dmabuf, int flags);
  ```

```c
struct dma_buf_ops {
  bool cache_sgt_mapping; /* bold means mandatory */
  bool dynamic_mapping;
  int (*attach)(struct dma_buf *, struct dma_buf_attachment *);
  void (*detach)(struct dma_buf *, struct dma_buf_attachment *);
  struct sg_table *(*map_dma_buf)(struct dma_buf_attachment *, enum dma_data_direction);
  void (*unmap_dma_buf)(struct dma_buf_attachment *, struct sg_table *, enum dma_data_direction);
  void (*release)(struct dma_buf *);
  int (*begin_cpu_access)(struct dma_buf *, enum dma_data_direction);
  int (*end_cpu_access)(struct dma_buf *, enum dma_data_direction);
  int (*mmap)(struct dma_buf *, struct vm_area_struct *vma);
  void *(*map)(struct dma_buf *, unsigned long);
  void (*unmap)(struct dma_buf *, unsigned long, void *);
  void *(*vmap)(struct dma_buf *);
  void (*vunmap)(struct dma_buf *, void *vaddr);
};
```
DMA-BUF API (IMPORTER)

- Retrieve dma-buf object
  ```c
  struct dma_buf *dma_buf_get(fd);
  void dma_buf_put(dma_buf);
  ```

- Attach device to dma-buf
  ```c
  struct dma_buf_attachment *dma_buf_attach(dma_buf, dev);
  struct dma_buf_attachment *dma_buf_dynamicAttach(dma_buf, dev, flag);
  void dma_buf_detach(dmabuf, attach);
  ```
  *The exporter could check if the backing storage is accessible to dev*

- Map to DMA address
  ```c
  struct sg_table *dma_buf_map_attachment(attach, direction);
  void dma_buf_unmap_attachment(attach, sg_table, direction);
  ```
  *This is when the exporter need to determine the backing storage location and pin the pages*

---

CPU access functions:
```
int dma_buf_begin_cpu_access();
int dma_buf_end_cpu_access();
void *dma_buf_kmap();
void dma_buf_kunmap();
int dma_buf_mmap();
void *dma_buf_vmap();
void dma_buf_vunmap();
```
USE DMA-BUF FOR GPU MEMORY RDMA
GPU SOFTWARE CHANGES

- **Dma-buf is supported by many existing GPU drivers**
  - As part of DRM / GEM / PRIME
  - Accessed by ioctl() over /dev/dri/card<n>, for example:

<table>
<thead>
<tr>
<th>command</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRM_IOCTL_MODE_CREATE_DUMB</td>
<td>Allocate a “dumb” buffer</td>
</tr>
<tr>
<td>DRM_IOCTL_I915_GEM_CREATE</td>
<td>Allocate a “GEM” buffer</td>
</tr>
<tr>
<td>DRM_IOCTL_PRIME_HANDLE_TO_FD</td>
<td>Get the dma-buf file descriptor</td>
</tr>
</tbody>
</table>

- Current GPU driver implementations may not be optimized for P2P access
  - On-going improvements. e.g. [https://www.spinics.net/lists/amd-gfx/msg32469.html](https://www.spinics.net/lists/amd-gfx/msg32469.html)

- **User space library needs to provide an interface to retrieve the dma-buf fd**
  - As a property of allocated memory object (e.g. as the IPC handle)
  - Applications don’t want to call ioctl directly
RDMA DRIVER CHANGES

- Core: support importing dma-buf as user memory via specialized `ib_umem_get()`

```c
struct ib_umem *
ib_umem_get(
    struct ib_udata *udata,
    unsigned long addr,
    size_t size, int access);
```

- Uverbs: define two new uverbs commands for memory registration
  - `IB_USER_VERBS_CMD_REG_MR_FD`
  - `IB_USER_VERBS_CMD_REREG_MR_FD`
  - These two commands require two extra parameters when compared with the non-FD version:
    - `fd_type`: type of the file descriptor, allow future extension
    - `fd`: the file descriptor

```c
struct ib_umem *
ib_umem_dmabuf_get(
    struct ib_udata *udata,
    unsigned long addr,
    size_t size, int dmabuf_fd,
    int access);
```
Add two functions to the `ib_device` structure for interfacing with the vendor drivers

```c
struct ib_device {
    ...
    struct ib_mr *(*reg_user_mr_fd)( ......., int fd_type, int fd, int acc, .... );
    int (*rereg_user_mr_fd)( ......., int fd_type, int fd, int acc, .... );
};
```

Vendor RDMA drivers: implement the two functions

- Implementation is optional
  - Only needed if the vendor driver want to support dma-buf
  - Can choose to only support reg, but not rereg
- Set `ib_dev->dev.uverbs_cmd_mask accordingly`
- Implementation is straightforward
  - Take the non-fd version, and replace `ib_umem_get()` with `ib_umem_dmabuf_get()`
RDMA LIBRARY CHANGES

- Add two new functions to the Verbs API

```c
struct ibv_mr *ibv_reg_mr_fd (  
    struct ibv_pd *pd,  
    void *addr,  
    size_t length,  
    enum ibv_mr_fd_type fd_type,  
    int fd,  
    int access );
```

```c
int ibv_rereg_mr_fd (  
    struct ibv_mr *mr,  
    int flags,  
    struct ibv_pd *pd,  
    void *addr,  
    size_t length,  
    enum ibv_mr_fd_type fd_type,  
    int fd,  
    int access);  
```

- Again, these functions have two extra parameters compared with the non-fd version
RDMA LIBRARY CHANGES (CONT)

- Add two uverbs command functions to interface with the kernel driver

  ```c
  int ibv_cmd_reg_mr_fd( ......, int fd_type, int fd, int access, ......);
  int ibv_cmd_rereg_mr_fd( ......, int fd_type, int fd, int access, ......);
  ```

- Add two functions to the `verbs_context_ops` structure for interfacing with vendor libraries

  ```c
  struct verbs_context_ops {
      ......
      struct ibv_mr *(*reg_mr_fd)( ......, enum ibv_mr_fd_type fd_type, int fd, int access );
      int (*rereg_mr_fd)( ......, enum ibv_mr_fd_type fd_type, int fd, int access );
  };
  ```

- Implement these two functions in the vendor specific RDMA library (provider)
  - Simply call the "ibv_cmd_" versions of these functions
OFI CHANGES

- New fields in the `fi_mr_attr` structure allow `fd` being passed for memory registration

```c
struct fi_mr_attr {
    ......
    enum fi_hmem_iface iface;  /* The API used for memory allocation */
    union {
        uint64_t reserved;
        ......
        int fd;
    } device;
};
```

- Must use `fi_mr_regattr`

- **Providers need to recognize these fields and handle the registration properly**
  - Support is indicated by the FI_HMEM capability bit
STATUS AND FUTURE WORK

▪ A software prototype has been implemented
  • Based on upstream Linux kernel 5.6 and most recent user space rdma-core libraries
  • GPU: Intel GPUs that use the i915 driver
  • RDMA NIC: Mellanox ConnectX-4 EDR, upstream driver

▪ Next steps
  • Getting the RDMA driver changes into upstream Linux kernel
    • First RFC patch set was sent to the linux-rdma list and reviewed
    • Revised RFC patch set is being worked on
    • Depend on GPU drivers being able to pin device memory via dma-buf interface, which is not there yet at upstream
  • Getting the RDMA library changes into upstream rdma-core
  • Upstream the OFI changes