Device Memory TCP

Transferring data from/to device memory efficiently

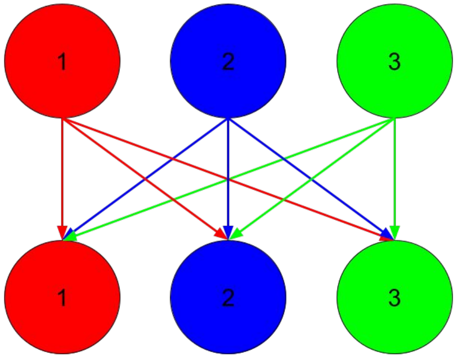
Netdev 0x17, 2023

Mina Almasry, on behalf of Willem de Bruijn, Eric Dumazet, & Kaiyuan Zhang
almasrymina@google.com

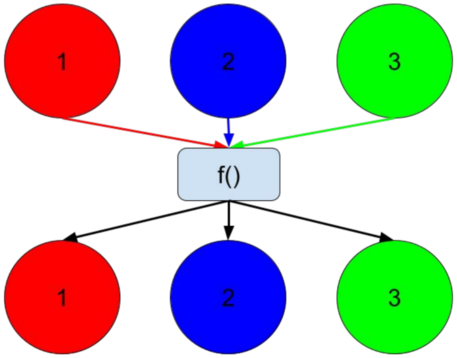
Problem space: Large ML jobs

- Machine learning jobs that span many nodes.
- Data held on each nodes in **GPU** memory.
- Jobs requires efficient data transfer between GPUs on different nodes.

Allgather

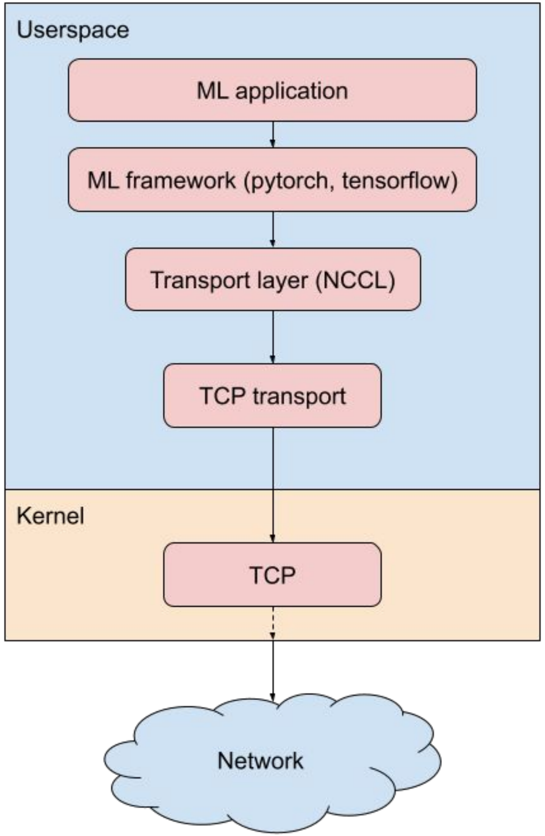


Allreduce



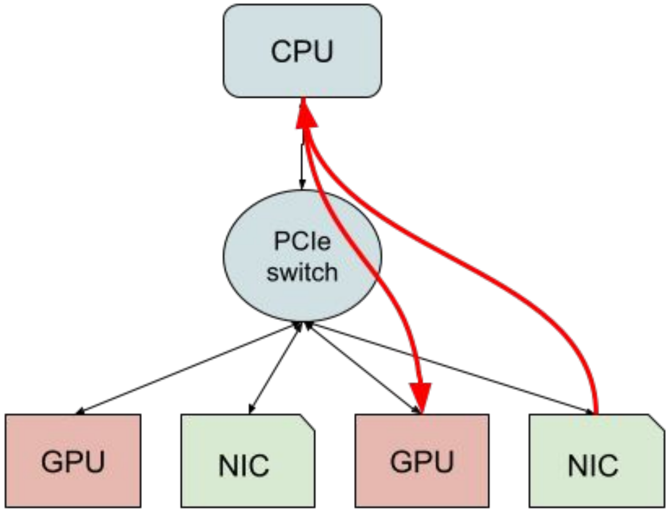
Problem space: Node zoom-in

- TCP requires a host memory bounce buffer.
    - Consumes memory bandwidth.
    - Consumes PCIe bandwidth.



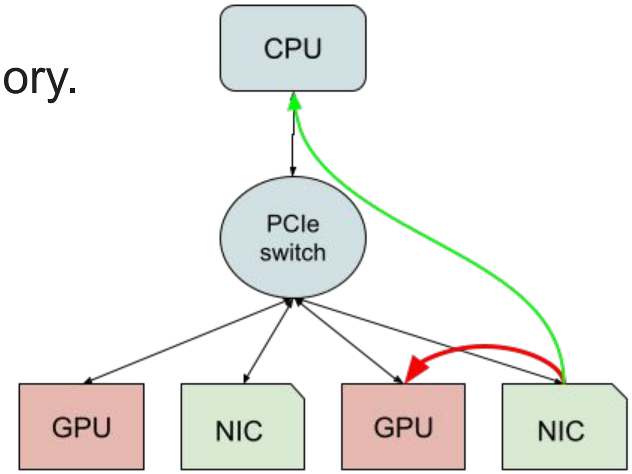
Problem space: PCIe bandwidth utilization

- Data goes from NIC -> root complex (host bounce buffers) -> GPU and vice versa.
- Stresses shared PCIe links.
- Google Cloud A3 VMs (8 H100 GPUs)
- Nvidia DGX H100 Systems (8 H100 GPUs)



Proposed solution: Device memory TCP

- Eliminate host memory bounce buffer.
- Transfer data directly to/from device memory.
- Packet payload lands directly into device memory.
- Packet header lands into host memory.
- RFC on the mailing list.



Journey of an RX packet: Device memory setup.

- Device memory abstraction of choice: dma-buf.
  - Standard in-kernel abstraction for device memory.
  - Device memory owner is an ‘exporter’.
  - Device memory user is an ‘importer’.
  - dma-buf APIs handle mapping/unmapping.
  - Not struct paged…
- User allocates device memory, obtains a dma-buf handle to the device memory.

Journey of an RX packet: NIC setup

- User ‘binds’ RX-queue to dma-buf.
  - Netdev netlink APIs
  - `page_pool` handles memory allocation from the dma-buf.
- Configures RSS to steer all other traffic to other queues
  - `ethtool -X <if> equal 15`
- Configures flow steering to steer their traffic to that queue:
  - `ethtool -N <if> src-ip <ip> dst-ip <ip>... queue 15`

Journey of an RX packet: page_pool

- Idea based on Jakub's memory-provider RFC.

```
+struct pp_memory_provider_ops {
+        int (*init)(struct page_pool *pool);
+        void (*destroy)(struct page_pool *pool);
+        struct page *(*alloc_pages)(struct page_pool *pool, gfp_t gfp);
+        bool (*release_page)(struct page_pool *pool, struct page *page);
+};
+
```

- Enables plugging in 'memory-providers' to the `page_pool`, supporting different memory types.

Journey of an RX packet: page_pool

- Dma-buf memory provider takes care of allocating `PAGE_SIZE` slices from the dma-buf and feeding them to the `page_pool`.
- But, dma-buf has no pages… `page_pool_iovs`!

```c
struct page_pool_iov {
        struct dmabuf_genpool_chunk_owner *owner;

        refcount_t refcount;

        struct page_pool *pp;
};

dma_addr_t
page_pool_iov_dma_addr(const struct page_pool_iov *ppiov);

unsigned long
page_pool_iov_virtual_addr(const struct page_pool_iov *ppiov);

int page_pool_iov_refcount(const struct page_pool_iov *ppiov);

void page_pool_iov_get_many(struct page_pool_iov *ppiov,
                            unsigned int count);

void page_pool_iov_put_many(struct page_pool_iov *ppiov,
                            unsigned int count);
```

Journey of an RX packet: nonpaged memory support

- `page_pool`, drivers, and `skb_frag_t` all use `page*` today.
- So much code churn… LSB pointer trick to reduce code churn.
- The LSB on `page_pool_iov*` is set and it's cast to `page*`.

```
#define PP_DEVMEM 0x01UL

static struct page *mp_dmabuf_devmem_alloc_pages(struct page_pool *pool,
                                                 gfp_t gfp)
{
        ...
        ppiov = netdev_alloc_devmem(binding);
        ...
        ppiov = (struct page_pool_iov *)((unsigned long)ppiov | PP_DEVMEM);
        return (struct page *)ppiov;
}

static inline bool page_is_page_pool_iov(const struct page *page)
{
        return (unsigned long)page & PP_DEVMEM;
}

static inline struct page_pool_iov *page_to_page_pool_iov(struct page *page)
{
        if (page_is_page_pool_iov(page))
                return (struct page_pool_iov *)((unsigned long)page &
                                                ~PP_DEVMEM);

        DEBUG_NET_WARN_ON_ONCE(true);
        return NULL;
}
```

Journey of an RX packet: nonpaged memory support

- `page_pool` does most of the heavy lifting.
- Handles any special casing for `page_pool_iov`
- Refcounting, dma_addr handling, pp info.
- Page-recycling and others work as-is with `page_pool_iov *`

```c
static inline dma_addr_t page_pool_get_dma_addr(struct page *page)
{
        dma_addr_t ret;

        if (page_is_page_pool_iov(page))
                return page_pool_iov_dma_addr(page_to_page_pool_iov(page));

        ret = page->dma_addr;

        ...

        return ret;
}
```

Journey of an RX packet: nonpaged memory support

- Drivers must use the `page *` they receive from `page_pool` as an opaque token.
- They (almost) already do this!
- `page_address()` is the main issue.

```
static inline dma_addr_t idpf_alloc_page(struct page_pool *pool,
                                         struct idpf_rx_buf *buf,
                                         unsigned int buf_size)
{
        if (buf_size == IDPF_RX_BUF_2048)
                buf->page = page_pool_dev_alloc_frag(pool, &buf->page_offset,
                                                     buf_size);
        else
                buf->page = page_pool_dev_alloc_pages(pool);

        ...

        return page_pool_get_dma_addr(buf->page) + buf->page_offset +
               pool->p.offset;
}
```

Journey of an RX packet: incoming packet

- NIC splits the packet into header + payload.
- Payload is DMA’d to a `page_pool_iov` in device memory.
  - Enables efficient data transfer.
- Header is DMA’d to a header buffer in host memory.
  - Enables the host kernel to parse the packet headers.
- NIC creates a ‘devmem’ skb and sends it up the stack.

Journey of an RX packet: devmem skb support

- Skbs are required to be either all devmem or host memory.
- Devmem skbs are marked with `skb->devmem` & `skb_frags_not_readable()`
- Results in some quirks:
  - Loopback.
  - Software checksum calculation.
  - TCP dump payload access.

Journey of an RX packet: recvmsg() uapi

- Device memory data can’t be copied to linear buffer or mapped to user’s address space.
- We provide ‘pointer’ to the memory in the dma-buf accessible from the userspace.

```
ssize_t ret = recvmsg(client_fd, &msg, MSG_SOCK_DEVMEM);
for (cm = CMSG_FIRSTHDR(&msg); cm; cm = CMSG_NXTHDR(&msg, cm)) {
        if (cm->cmsg_level != SOL_SOCKET ||
             cm->cmsg_type != SCM_DEVMEM_OFFSET){
                continue;
        }
        ...
        cmsg_devmem = (struct cmsg_devmem *)CMSG_DATA(cm);
        ...
        validate_buffer(buf_mem + cmsg_devmem->frag_offset,
                        cmsg_devmem->frag_size);
        ...
}
```

Journey of an RX packet: recvmsg() uapi

- Need the user to give us a signal ‘`setsockopt()`‘ that they’re done with the data.

```
cmsg_devmem = (struct cmsg_devmem *)CMSG_DATA(cm);

struct devmemtoken token = { cmsg_devmem->frag_token, 1 };

ret = setsockopt(client_fd, SOL_SOCKET,
                 SO_DEVMEM_DONTNEED, &token,
                 sizeof(token));
```

Device memory TX path

- Much more straightforward than RX path.
- Dma-buf to send can be passed to `sendmsg()` API.
- Largely follows the `MSG_ZEROCOPY` code path.
- Need to create `iov_iter` that is backed by `page_pool_iovs`.
- Re-uses the same devmem skb support as RX path.
- `skb_frag_dma_map` can grab the dma_addr from `page_pool_iovs`.

Initial results

- ~96% line rate at TCP level: 192 gbps bi-directional per NIC/GPU pair.
- Running in production.
- Transports exercised with production workloads: NCCL.
- Pytorch exercised with production workloads.
  - Tensorflow, JAX & others use the same transport primitives.
- ~3X better throughput than regular TCP NCCL transports*.
- Comparable network efficiency to RDMA-based NCCL transports for larger message sizes.

Possible follow up work

- io_uring support.
- Dynamic queue management.