Netconf 2023

Willem de Bruijn

Summary

- SO_DEVMEM: direct GPU data placement
  - memory allocation (page_pool)
  - queue allocation
- AF_XDP + BPF
  - RSS
  - busypoll kthread

Q: when to refactor complex code

- skb_segment
- ip(6)_append_data

maintainable code vs stable backport complexity

SO_DEVMEM

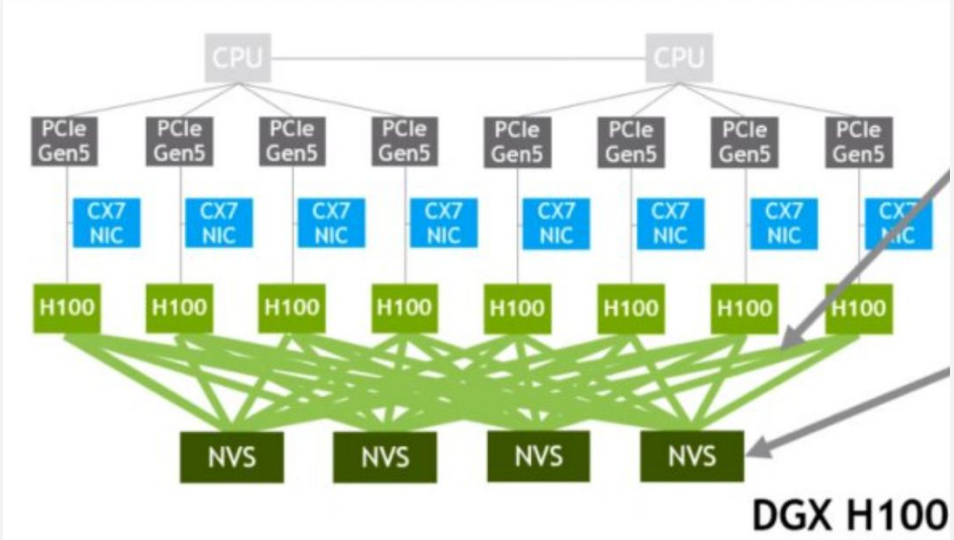
Background

Direct payload to/from GPU

4+ accelerators per CPU: exceed RAM BW

Protocol processing by kernel stack

Header-split



Memory Allocation

from page_pool to mem_pool providers:

net: huge page backed page_pool

- huge pages
- device mem
- io_uring
- af_xdp?

static_branch to avoid hot path regressions

Memory Allocation: challenges

- replace posted buffers
  - without link down+up of entire device (ndo_close)

- clean deallocation
  - (unclean) process exit
  - while buffers (1) posted, (2) in kernel stack

Memory Allocation: no more struct page

no more special struct page variants (ZONE_DEVICE)

LSB pointer trick, or union skb_frag_t?

```
typedef struct bio_vec skb_frag_t;
```

```
struct bio_vec {
        struct page     *bv_page;
        unsigned int    bv_len;
        unsigned int    bv_offset;
};
```

Queue allocation: today's poor man's hack

```
# limit RSS set

ethtool -X $DEV equal N-1

ethtool -N $DEV flow-type tcp4 src-port 8000 action N-1
```

- guess and pre-allocate worst case queue count
- strand rx posted memory in idle queues

Queue allocation

- Per queue alloc/free
  - rather than ethtool -L and ndo_close+ndo_open

- API 1: netlink admin commands
  - containerization: privileged admin process
  - create_queue, create_mempool
  - flow-steering: ntuple (or tc flower?)
- API 2: flow steering inferred from sk 3 or 5 tuple
  - setsockopt(SOL_SOCKET, SO_BINDTOQUEUE, ..)
- API 3: see also
  - verbs API
  - devlink subfunctions

Queue allocation

- rx queue allocation and fill: link down + up

Currently we do:

user -> thin ethtool API ----------> driver
netdev core <---'

By "straighten" I mean more of a:

user -> thin ethtool API ---> netdev core ---> driver

flow. This means core maintains the full expected configuration, queue count and their parameters and driver creates those queues as instructed.

I'd imagine we'd need 4 basic ops:
- queue_mem_alloc(dev, cfg) -> queue_mem
- queue_mem_free(dev, cfg, queue_mem)
- queue_start(dev, queue info, cfg, queue_mem) -> errno
- queue_stop(dev, queue info, cfg)

AF_XDP

RSS

Relax condition

```
if (xs->dev != xdp->rxq->dev || xs->queue_id != xdp->rxq->queue_index)
        return -EINVAL;
```

if XDP_COPY || XDP_SHARED_UMEM

Busypoll kthread

process on CPU A, kernel on CPU B translating phys <-> af_xdp desc

attempt: immediately reschedule (threaded) napi handler

```
echo 1 > /sys/class/net/eth0/threaded

echo 1000000 > /sys/class/net/eth0/napi_defer_hard_irqs

echo 10 > /sys/class/net/eth0/gro_flush_timeout
```

deserves a real API?

see als netdevconf 2.1: "busy polling: past, present, future"