Bare Metal Library

Abstractions for modern hardware

Cyprien Noel

Plan

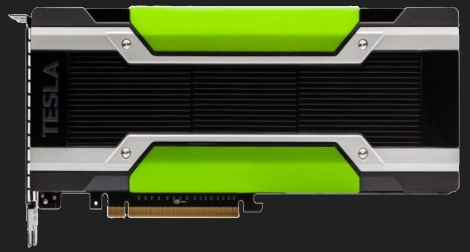
1. Devices, lots of them
2. It’s a problem
3. But solutions
    - Device-centric abstractions
    - Device-to-device flows

Myself

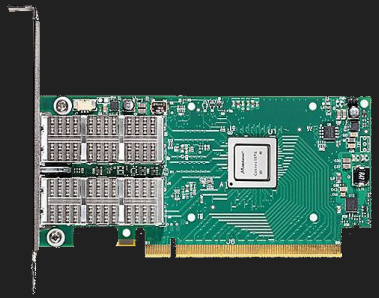
- High performance trading systems
  - Lock-free algos, distributed systems
- H2O
  - Distributed CPU machine learning, async SGD
- Flickr
  - Parallel deep learning - Multi-GPU Caffe
  - Distributed deep learning - CaffeOnSpark, RDMA, multicast, Hogwild
- UC Berkeley
  - NCCL Caffe, GPU cluster tooling
  - Bare Metal

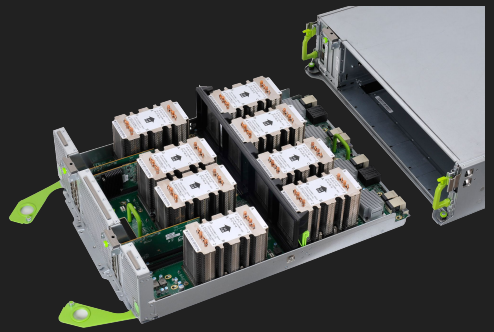
Trend


intel
XEON
inside™


TESLA

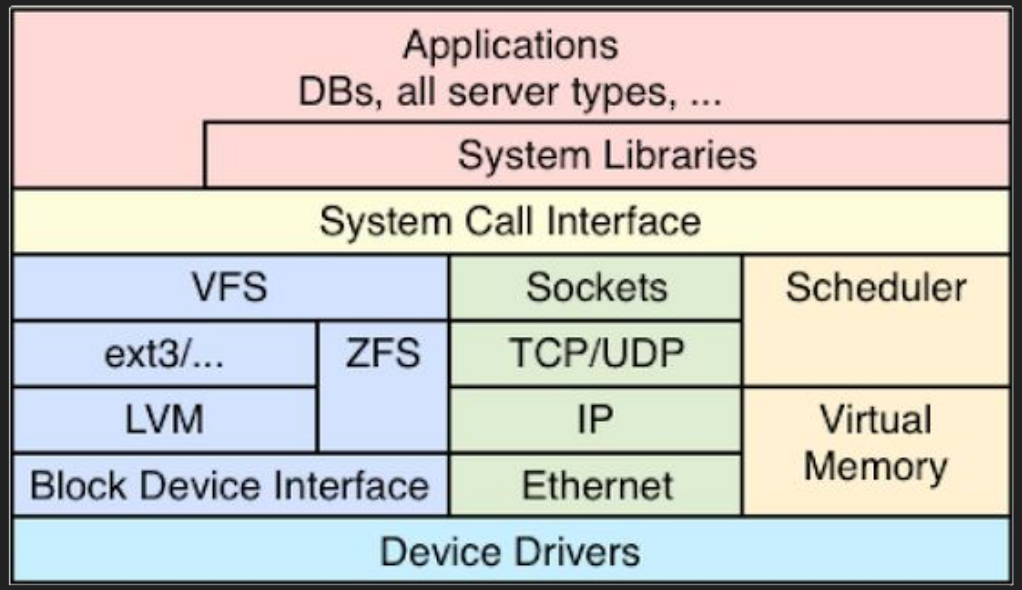

XILINX
VIRTEX
FPGA






NVMe SSD
960 EVO
M.2
250GB
SAMSUNG

ms software ➔ µs hardware



Number crunching ➔ GPU

FS, block io ➔ Pmem

Network stack ➔ RDMA

RAID, replication ➔ Erasure codes

Device mem ➔ Coherent fabrics

And more:
Video, crypto etc.

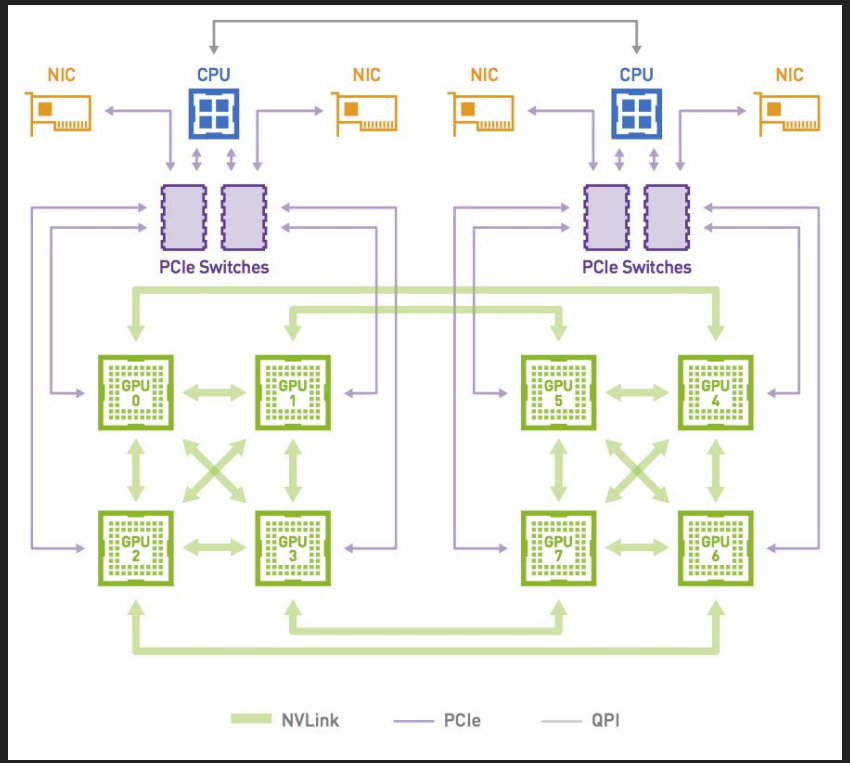
Good ol’ OS abstractions replaced by

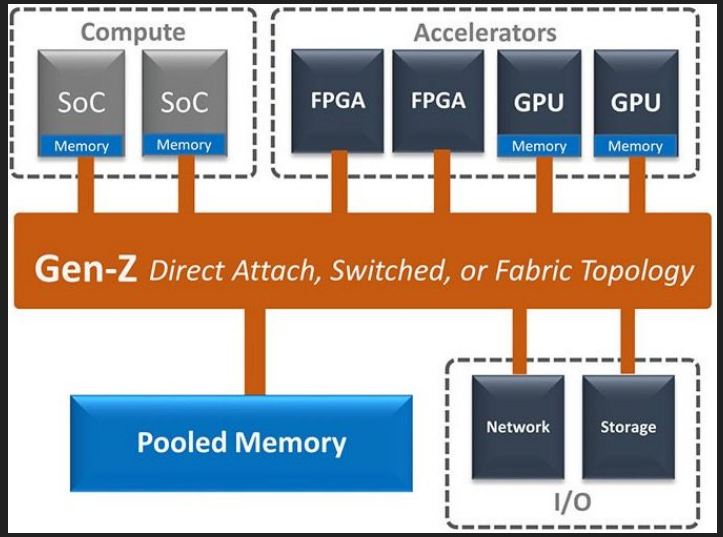
- CUDA
- OFED
- Libpmem
- DPDK
- SPDK
- Libfabric
- UCX
- VMA
- More every week...

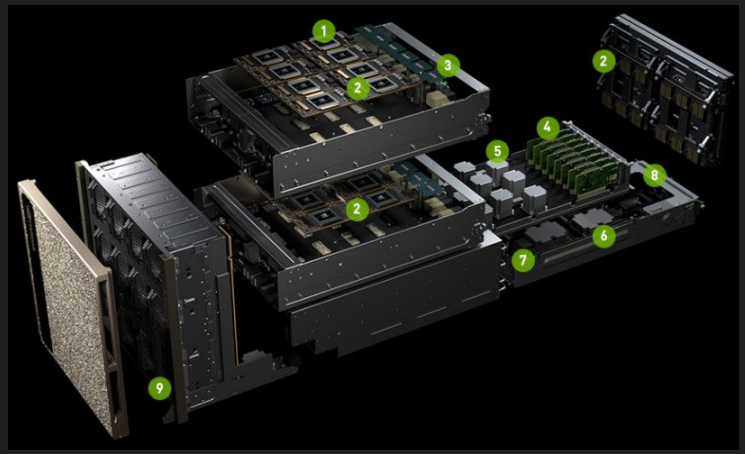
Faster, more powerful, but

- More complex
- Non-interoperable
  - Namespaces
  - Security models
  - Failure models

Device-to-device flows







Summary So Far - Big changes coming!

- CPU should orchestrate
  - Not in critical path
  - Device-to-device flows - data and control

What do we want?

- Stop over-specifying!
  - Location transparency: device-to-device, HA
  - Single namespace, security and failure models
- Simple model, slight extension of something familiar
- Forward and backward version compatible
- Thin efficient abstraction over hardware
- Stretch goals
  - Versioned git style
  - Capability system
  - Stateless apps, all state in versioned namespace

Proposal

- Single namespace
  - File system like, distributed
  - HA using hardware erasure codes
- Nodes are data, compute steps or devices
  - E.g. numpy, protobuf, CUDA kernel, compute graph
  - Node execution starts when inputs are ready
- Versioned
  - Branch abstraction
  - Atomic merge or abort

API: mmap

- Extension to classic mmap
  - Distributed
  - Typed - numpy, protobuf, other formats planned
- Python example

```
test = Test()
bm.mmap('/test', test)
i = test.field()
```

API: task

```
@bm.task
def compute(x, y):
    return x * y

# Runs locally
compute(1, 2)

# Might be rebalanced on cluster
data = bm.list()
bm.mmap("/data", data)
compute(data, 2)
```

API: branch

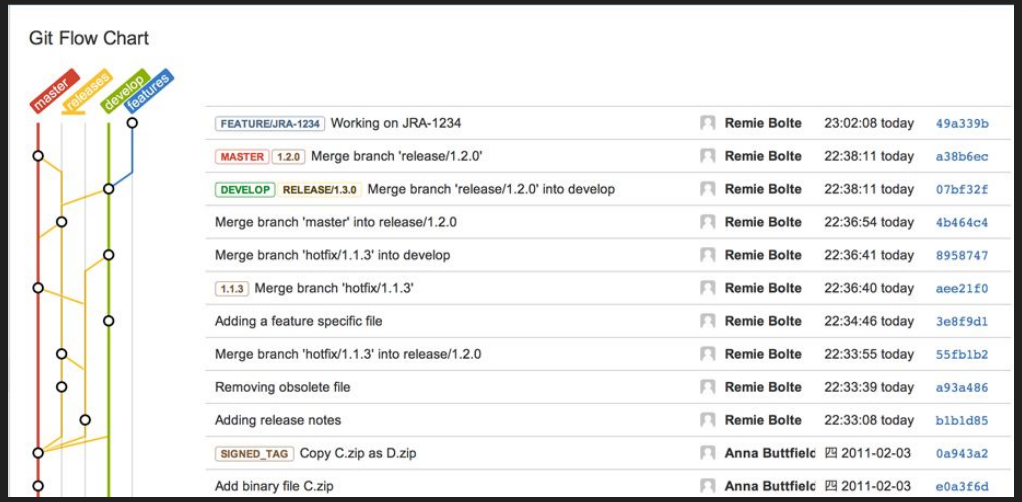
```
bm.branch = 'my_branch'
# ... modify dataset
bm.commit()


account1 = bm.mmap('/account1', my_model.account())
account2 = bm.mmap('/account2', my_model.account())
with bm.branch() as b:
   account1.set_balance(account1.balance() + 12)
   account2.set_balance(account2.balance() - 12)
```


Internals

- Hardware erasure codes for all state
- Device-to-device transfers - e.g. GPU Direct
- Device-to-device control - e.g. GPU Direct Async
- Work stealing - RDMA atomics
- Branches and lattice simplify a lot
  - No locks or coordination for most tasks
  - Atomicity - simplifies consistency
    - Replaces transactions, e.g. KV, queues, persistent memory
    - No file system fsync, msync (Very hard! Rajimwale et al. DSN ‘11)
  - Allows duplicate work merge
  - Generalized staging / production split

Data pull requests

- A data PR would contain
  - Data inputs
  - Code, compute graphs
  - Execution logs
- Single history tree
  - Code
  - Compilations
  - Executions
  - Data
- Persistent Jupyter Notebook?



Thank You!

Will be open sourced BSD

Contact me if interested - cyprien.noel@berkeley.edu

Thanks to our sponsor