Designing Networking Stacks for ML Frameworks

Raghu Raja
raghu@enfabrica.net
Agenda

- Understanding Neural Networks
- Taxonomy of ML techniques
- New world, old problems - Design Challenges to consider
- Case study with NCCL
- Call to action
Neural Networks

16x16 Grid

Input Layer
(256 neurons – 1 neuron per image pixel)

Hidden Layers

Output Layer

Image

7

16x16 Grid
Growth of Model Sizes

Number of Parameters of Select Large Language and Multimodal Models, 2019–22

Source: Epoch, 2022 | Chart: 2023 AI Index Report

- GPT-3 175B (daVinci)
- Wu Dao 2.0
- Megatron-Turing NLG 530B
- Minerva (640B)
- Gopher
- PalM (540B)
- HyperClov
- PanGu-It
- Jurassic-1-Jumbo
- OPT-175B
- BLOOM
- GLM-130B
- Chinchilla
- GLM-130B
- GPT-NeoX-20B
- Jurassic-X
- DALL-E
- Codex
- ERNIE 3.0
- DALL-E 2
- CogView
- Wu Dao-Wen Yuan
- Stable Diffusion (LDM-KL-8-G)
- GPT-Neo
- GPT-J-6B
- ERNIE-GEN (large)
- ErnVe
- ErnVe
- T5-3B
- Meena
- Megatron-LM (Original, 8.3B)
- T5-11B
- Turing NLG
- Grover-Mega
- ERNIE-GEN (large)
Taxonomy of Parallelization Techniques

- Data Parallelism
- Model Parallelism
- Fully Sharded Data Parallelism
- Tensor Parallelism
- Pipeline Parallelism
- Mixture-of-Experts (MoE)
- Framework-specific techniques
Data Parallelism

- Data Parallelism
  - Same model across GPUs/compute elements
  - Train each with different samples
  - Stochastic Gradient Descent (SGD)
    - Asynchronous
      - Non-blocking across GPUs
    - Parameter Server model
  - Synchronous
    - AllReduce (Sum) of gradients from all GPUs before updating model
    - SGD with large mini-batch is the most common training model
- ~1.2B parameters
Model Parallelism

- Model Parallelism
  - Distribute model across compute elements
  - Given batch -> sequential progress across layers. Same input batch fed to all GPUs.
  - MPI-style tightly-couple communication requirements
- Scaling limitations, 8-16 GPUs. ~20B parameters
- Hybrid MP+DP
- Pipeline parallelism. ~100B parameters
- Framework-specific techniques - ~1T+ parameters
Mixture of Experts

- Limited to encoder-decoder / sequence-to-sequence tasks
- Reduced compute requirements
- Increased memory requirements
- Reduced parameter efficiency
- Google Gshard, Switch Transformer, DeepSpeed-MoE (MoS)
- Sub-linear scaling

Inference

- “Deploy” a model with learned weights and biases
- Strict SLAs: latency $O(\text{ms})$ and accuracy
- Deployment scale grows with user counts. High memory requirements.
- LLM / DLRM: Parameters, embeddings, user contexts.
Role of Collective Communication

- Initial parameter distribution and batch setup: Broadcast
- Gradient descent: AllReduce w/average in the backward pass
- Synchronization: Barrier
- Fully Sharded Data Parallel: ReduceScatter + AllGather
- Mixture-of-Experts: 2 x AlltoAll forward pass, 2 x AlltoAll backward pass
The Proliferation of *CCLs

- MPI: The OG CCL
- NCCL: NVIDIA GPUs
- RCCL: For AMD GPUs, via ROCm Hip instead of CUDA
- oneCCL: For Intel Xe GPUs, via oneAPI (libfabric and friends)
- UCC: Collectives over UCX, draws from HCOLL, SHArP, IBM PAMI, etc
- MSCCL = TACCL + SCCL: Topology aware, GC3 DSL
- Alibaba ACCL
- MCR-DL: Mixing network backends
- Open XLA / HLO collectives
- Baidu’s allreduce
- Huawei UCG w/OpenMPI
- Xilinx ACCL: For Alveo FPGAs
Networking for ML: Design Challenges

- Plenty of FLOPS, oversubscribed and underutilized
- 800G ethernet is here. Networking bandwidth has caught up with memory bandwidth
- Link utilization and load balancing: Collectives → Incast issues → Congestion control
- But we also want faster collectives!
- Mosaic of L3/L4 transport design choices: message semantics, ordering constraints, etc
- Model sizes outgrowing accelerator HBM capacities
- Desire to build vendor-agnostic software stacks
- Managing complex system topologies across networking, compute, and storage.
System Architectures

Scale accelerators
Under-utilized cores
Expensive

Scale Hosts / CXL.mem
Oversubscribed PCIe BW
Performance Hit
Enfabrica - ACF-Switch

ECMP / Port Group

Switch Network Interfaces

Copy Engines

Control Plane
Buffer
Memory Controller

Switch Compute Interfaces

Multi-destination

Pages

Packets

Memory Controller

Copy Engines

Switch Compute Interfaces

Multi-destination

Pages

Packets

ECMP / Port Group

Switch Network Interfaces

Copy Engines

Control Plane
Buffer
Memory Controller

Switch Compute Interfaces

Multi-destination

Pages

Packets

ECMP / Port Group

Switch Network Interfaces

Copy Engines

Control Plane
Buffer
Memory Controller

Switch Compute Interfaces

Multi-destination

Pages

Packets
An Incarnation of ACF-S

Multi-Terabit Network Fabric
N x 800 / 400 / 200 / 100 Gigabit Ethernet

Multi-Terabit Compute Fabric
High Fanout PCIe Gen5 / CXL 2.0+ / UCIe
blvd: NCCL Plugin and more

torch.nn.parallel.DistributedDataParallel

torch.distributed

Gloo  MPI  NCCL

blvd

IB Verbs

ncclNet_t -> irecv/isend
ncclCollNet_t

(See Shrijeet Mukherjee’s talk from earlier today for details: “RDMA and Linux TCP”)
blvd: Design Highlights

- Zero-copy from the app → NCCL → blvd → TCP → H/W
- Works with upstream kernel without needing custom hooks for peer-to-peer communication
- librdmacm API for connection management. Standard libibverbs APIs.
- Supports all NCCL PTR types: HOST, CUDA, DMABUF
- Message framing extensions
- Test vehicle to prove out ACF architectural concepts and primitives
- Existing verbs plugin is not general enough. Non-standard assumptions and custom hooks for P2P communication.
FPGA-based Emulation Platform

- AMD Ryzen motherboard
- Virtex UltraScale+ XCVU37P
- Xilinx PCIe Gen3 x16 lane (126Gbits/s) interface to the CPU
- 2 x PCIe Gen3 x16 lanes to the GPUs, limited to ~40Gbps
- 4 x 100G Ethernet CMACs with routing lookup BCAM
- Running unmodified 5.13.x kernel
- Purely an experimental platform
Evaluation Results

NCCL Operation Time

Time (ms)

Message size

Baseline
TCP
blvd

1M 4M 16M 64M 256M
Evaluation Results

NCCL Bus Bandwidth

Baseline TCP
blvd

Message size

Bandwidth (Gbps)

https://github.com/NVIDIA/nccl-tests/blob/master/doc/PERFORMANCE.md
Next Steps

Scales in Horizontal slices for User Data Store

O(n) Cost Scaling w/ GPU, CPU

Hub-and-Spoke, Dynamic Dispatch of User Data
Enables horizontal slicing and much more …

O(log(n)) Dynamic Cost Scaling w/ GPU, CPU
Tiered, Equidistant Memory Latency
Call to Action

● For the *CCL builders and maintainers
  ◦ Better plugin interfaces that have an impedance match with hardware vendor efforts
  ◦ Allow more control over the topology management
  ◦ Better documentation of design choices in addition to user/admin guides

● For the OFA community
  ◦ Co-design ML stacks, beyond writing plugins.
  ◦ More active involvement in ML framework development communities

● For the HPC community: Come up with better names for designs! /s
Thank You.

Questions, comments, or collaboration:

raghu@enfabrica.net  +  hello@enfabrica.net