DEEP LEARNING ON GPUS
Making DL training times shorter

Deeper neural networks, larger data sets ... training is a very, very long operation!
NCCL
A multi-GPU communication library

To other systems
Sockets (Ethernet)
Infiniband, with GPU Direct RDMA

Within a system
NVLink
PCIe
GPU Direct P2P
NCCL Architecture

Deep Learning Frameworks

Caffe | Caffe2 | Torch | TF | MXNET | CNTK

NCCL

CUDNN | CUBLAS

CUDA

NVIDIA GPUs
AGENDA

NCCL
  History
  Design

NCCL 2.0
  New features
  API Changes

Performance

Future
HISTORY

Q4 2015: NCCL 1.x

Open-source research project on github, helping Deep Learning frameworks compute on multiple GPUs with efficient collective operations.

Limited to intra-node.

Q2 2017: NCCL 2.x and beyond

NVIDIA Library, multi-node support and improved API.
DESIGN
What is NCCL?

Optimized collective communication library between CUDA devices.

Easy to integrate into any DL framework, as well as traditional HPC apps using MPI.

Runs on the GPU using asynchronous CUDA kernels, for faster access to GPU memory, parallel reductions, NVLink usage.

Operates on CUDA pointers. Operations are tied to a CUDA stream.

Uses as little threads as possible to permit other computation to progress simultaneously.
NCCL uses rings to move data across all GPUs and perform reductions.
NCCL uses **rings** to move data across all GPUs and perform reductions.

PCIe / QPI : 1 unidirectional ring
NCCL uses rings to move data across all GPUs and perform reductions.

PCle / QPI : 1 unidirectional ring

DGX-1 : 4 unidirectional rings
DESIGN
Kernels

Previous GPU in the ring

FIFO

Reduction

Next GPU in the ring

sendbuff
recvbuff
NCCL 2.0
NCCL 2.0
Inter-node communication

Inter-node communication using Sockets or Infiniband verbs, with multi-rail support, topology detection and automatic use of GPU Direct RDMA.

Optimal combination of NVLink, PCI and network interfaces to maximize bandwidth and create rings across nodes.
NCCL 2.0
Inter-node communication

Inter-node communication using Sockets or Infiniband verbs, with multi-rail support, topology detection and automatic use of GPU Direct RDMA.

Optimal combination of NVLink, PCI and network interfaces to maximize bandwidth and create rings across nodes.
NCCL 2.0
Processes, threads and GPUs

Supports a combination of processes (potentially across nodes), threads per process and GPUs per thread.
NCCL 2.0
Processes, threads and GPUs

Supports a combination of processes (potentially across nodes), threads per process and GPUs per thread.
NCCL 2.0
Processes, threads and GPUs

Supports a combination of processes (potentially across nodes), threads per process and GPUs per thread.
NCCL 2.0
Processes, threads and GPUs

Supports a combination of processes (potentially across nodes), threads per process and GPUs per thread.
NCCL 2.0 API

Group calls

NCCL 2.0 is introducing mandatory new verbs `ncclGroupStart/ncclGroupEnd` when managing multiple devices from a single thread

NCCL 1.x:
```c
for (int i=0; i<ngpus; i++) {
    cudaSetDevice(devices[i]);
    ncclAllReduce(..., comms[i], streams[i]);
}
```

NCCL 2.0:
```c
ncclGroupStart();
for (int i=0; i<ngpus; i++) {
    ncclAllReduce(..., comms[i], streams[i]);
}
ncclGroupEnd();
```
NCCL 2.0 API

Integration with parallel environments

Inter-node communicator creation still uses the NCCL 1.x verbs: ncclGetUniqueId/ncclCommInitRank

```c
if (rank == 0) ncclGetUniqueId(&id)
    My_Bcast(&id);
ncclCommInitRank(&comm, nranks, id, rank);
```

Multi-process + multi-GPU per process (from a single thread): combine ncclCommInitRank with ncclGroupStart/ncclGroupEnd

```c
if (rank == 0) ncclGetUniqueId(&id)
    My_Bcast(&id);
ncclGroupStart();
for (int i=0; i<ndev; i++) {
    cudaSetDevice(devices[i]);
    ncclCommInitRank(&comm, ndev*nranks, id, ndev*rank+i);
}
ncclGroupEnd();
```
NCCL 2.0 API

Others

Other small API adjustments over the NCCL 1.x API:

Counts are now of type `size_t` instead of `int`

`allGather` arguments order has been fixed to be similar to other operations

Additions/clarification on `datatypes`:
- integral: `int8 = char, uint8, int32 = int, uint32, int64, uint64`
- floating point: `float16 = half, float32 = float, float64 = double`

Clarifications and fixes for `allgather` and `reduce_scatter` send/receive `counts` and `in-place` operations
PERFORMANCE
PERFORMANCE

Intra-node performance

AllReduce bandwidth (OMB, size=128MB, in GB/s)
PERFORMANCE

Inter-node performance

AllReduce bandwidth (OMB, size=128MB, in GB/s)

- 2 nodes x 4 GPUs (IB EDR, PCI Switch)
- 4 nodes x 8 GPUs (DGX-1 : 4x IB EDR, 4x NVLink)

Legend:
- MPI
- Baidu Allreduce
- NCCL
PERFORMANCE
Deep Learning - CNTK

CNTK scaling
ResNet50, images/s
FUTURE
FUTURE
Top asked features

Additional communication primitives:
point-to-point communication
scatter (1 to N), gather (N to 1), alltoall (N to N)
neighbor collectives (send/receive in multiple dimensions)

User-defined reduction operations
also, trying to merge computation and communication better

Windows support

Please let us know your needs!

Connect with experts / NCCL session: Wed Apr 10, 4pm