



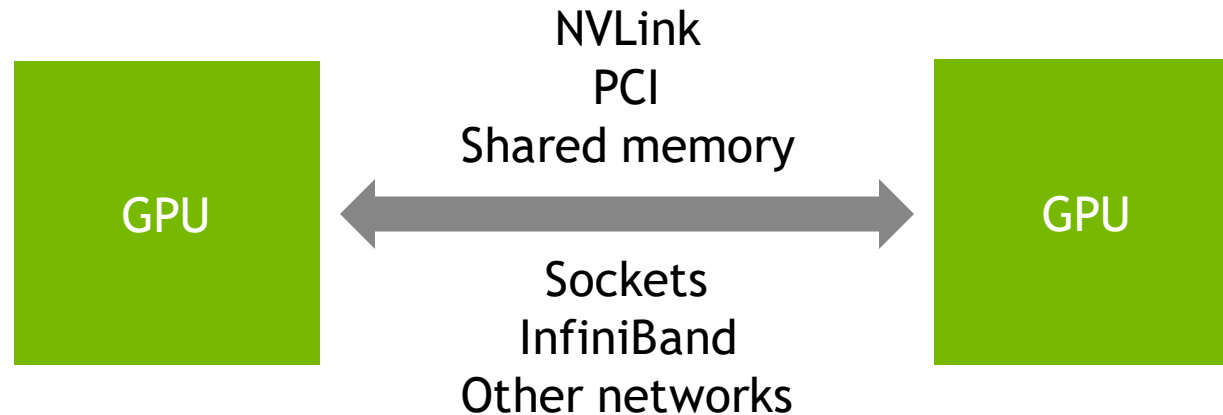
S31880 - NCCL: HIGH-SPEED INTER-GPU COMMUNICATION FOR LARGE-SCALE TRAINING

Sylvain Jeaugey



OPTIMIZED INTER-GPU COMMUNICATION

NCCL : **N**VIDIA **C**ollective **C**ommunication **L**ibrary
Communication library running on GPUs, for GPU buffers.



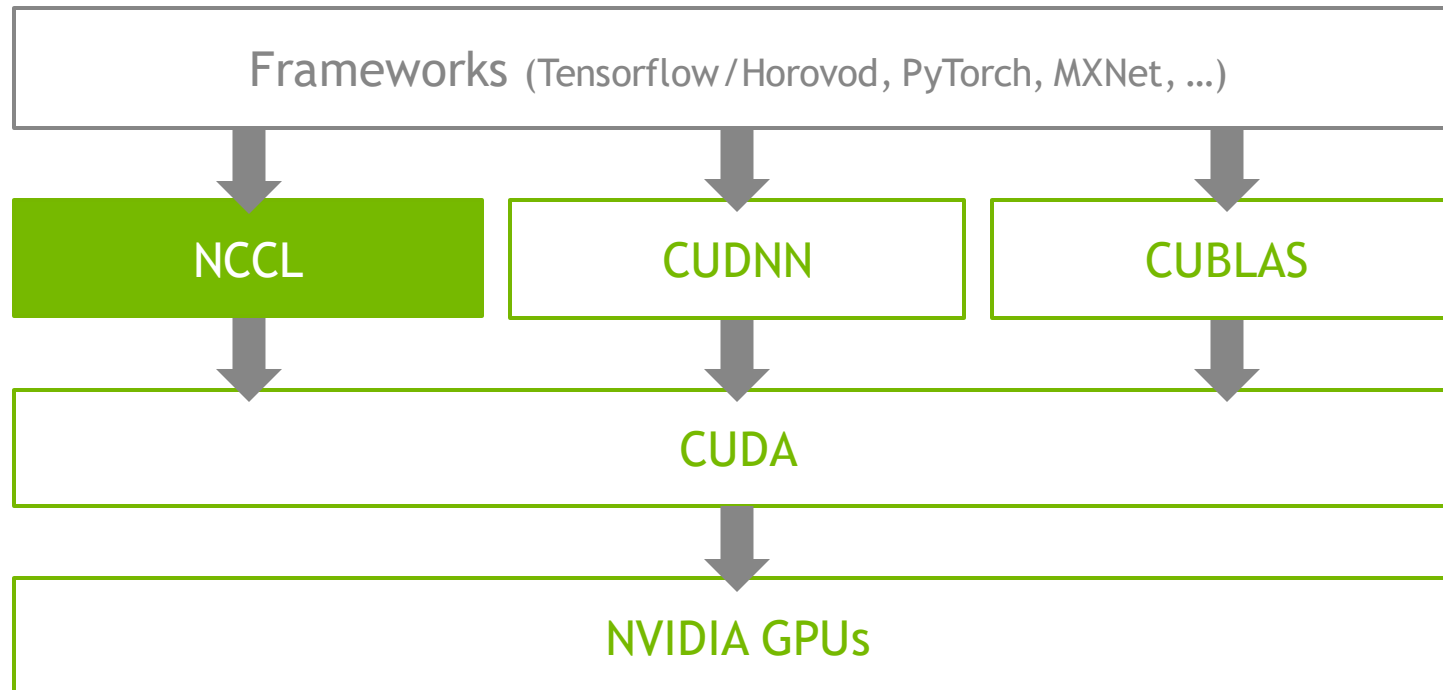
Binaries : <https://developer.nvidia.com/nccl> and in NGC containers

Source code : <https://github.com/nvidia/nccl>

Perf tests : <https://github.com/nvidia/nccl-tests>

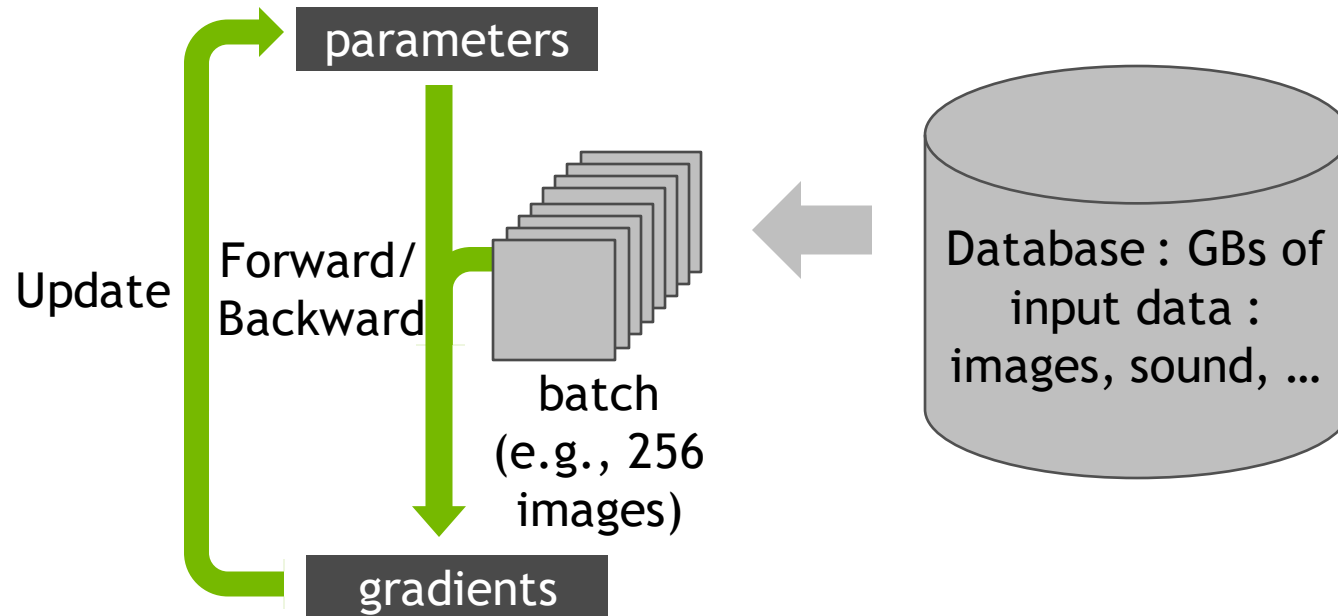
NCCL

DL stack



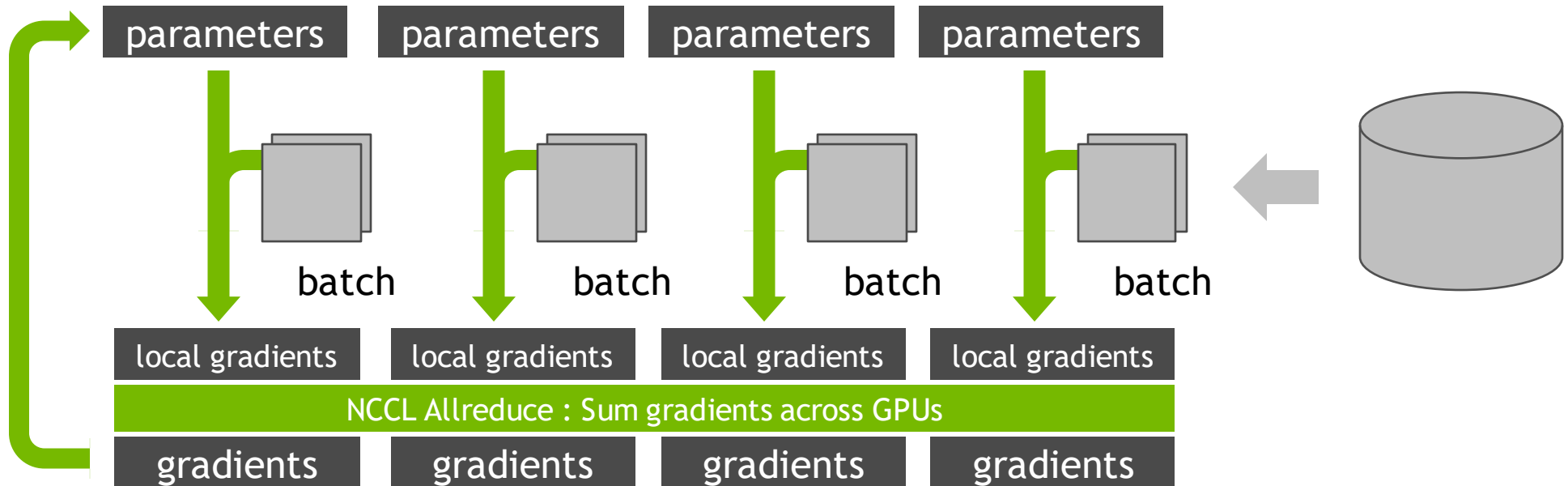
MULTI-GPU TRAINING

Single-GPU



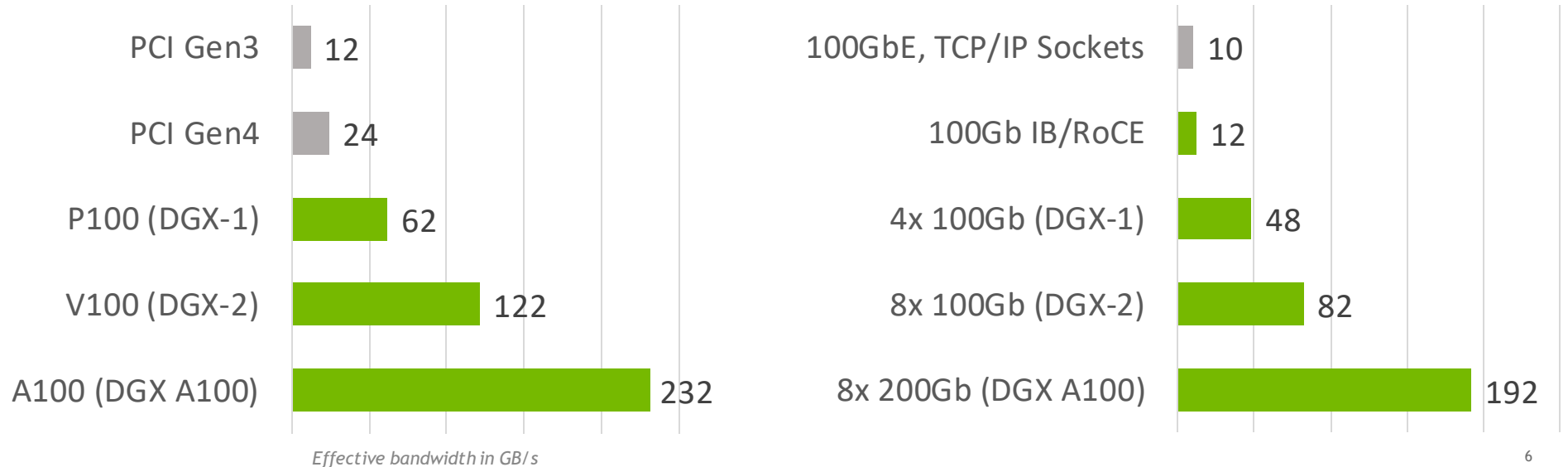
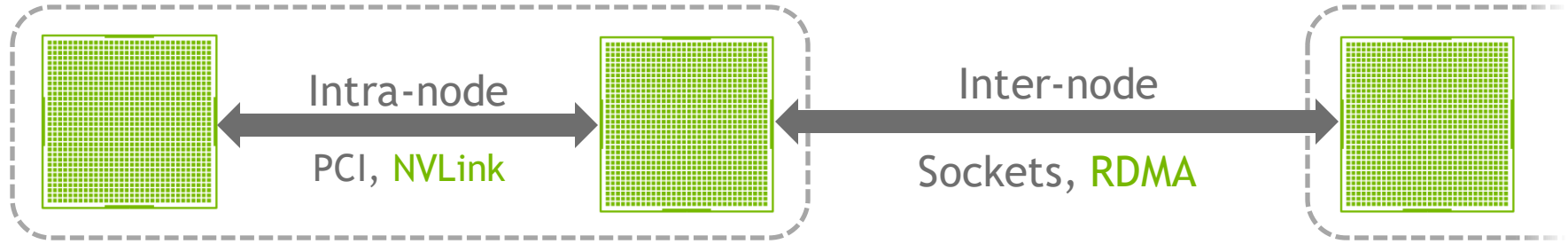
MULTI-GPU TRAINING

Data parallel



INTER-GPU COMMUNICATION

Intra-node and Inter-node



NCCL API

Overview

// Communicator creation

```
ncclGetUniqueId(ncclUniqueId* commId);  
ncclCommInitRank(ncclComm_t* comm, int nranks, ncclUniqueId commId, int rank);
```

// Communicator destruction / fault tolerance

```
ncclCommDestroy(ncclComm_t comm);  
ncclCommAbort(ncclComm_t comm);  
ncclCommGetAsyncError(ncclComm_t comm, ncclResult_t* asyncError);
```

// Collective communication

```
ncclAllReduce(void* sbuff, void* rbuff, size_t count, ncclDataType_t type, ncclRedOp_t op, ncclComm_t comm, cudaStream_t stream);  
ncclBroadcast(void* sbuff, void* rbuff, size_t count, ncclDataType_t type, int root, ncclComm_t comm, cudaStream_t stream);  
ncclReduce(void* sbuff, void* rbuff, size_t count, ncclDataType_t type, ncclRedOp_t op, int root, ncclComm_t comm, cudaStream_t stream);  
ncclReduceScatter(void* sbuff, void* rbuff, size_t count, ncclDataType_t type, ncclRedOp_t op, ncclComm_t comm, cudaStream_t stream);  
ncclAllGather(void* sbuff, void* rbuff, size_t count, ncclDataType_t type, ncclComm_t comm, cudaStream_t stream);
```

// Point-to-point communication

```
ncclSend(void* sbuff, size_t count, ncclDataType_t type, int peer, ncclComm_t comm, cudaStream_t stream);  
ncclRecv(void* rbuff, size_t count, ncclDataType_t type, int peer, ncclComm_t comm, cudaStream_t stream);
```

// Aggregation/Composition

```
ncclGroupStart();  
ncclGroupEnd();
```



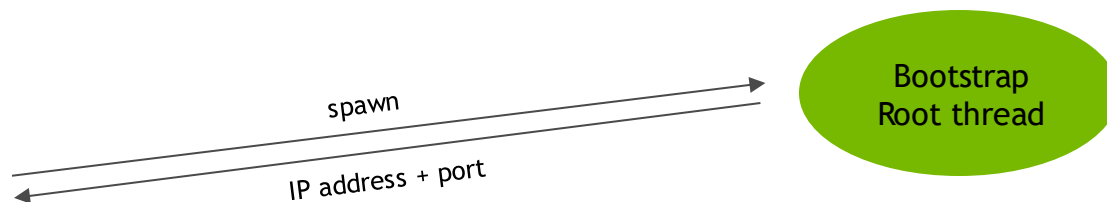
INIT / BOOTSTRAP

NCCL BOOTSTRAP

Principle

Once (typically on worker0):

```
ncclUniqueId id  
ncclGetUniqueId(&id);  
broadcast_to_all_ranks(&id);
```



On all parallel workers:

```
get_unique_id(&id);  
ncclCommInitRank(&comm, nranks, &id, rank);  
ncclAllReduce(..., comm);
```

NCCL BOOTSTRAP

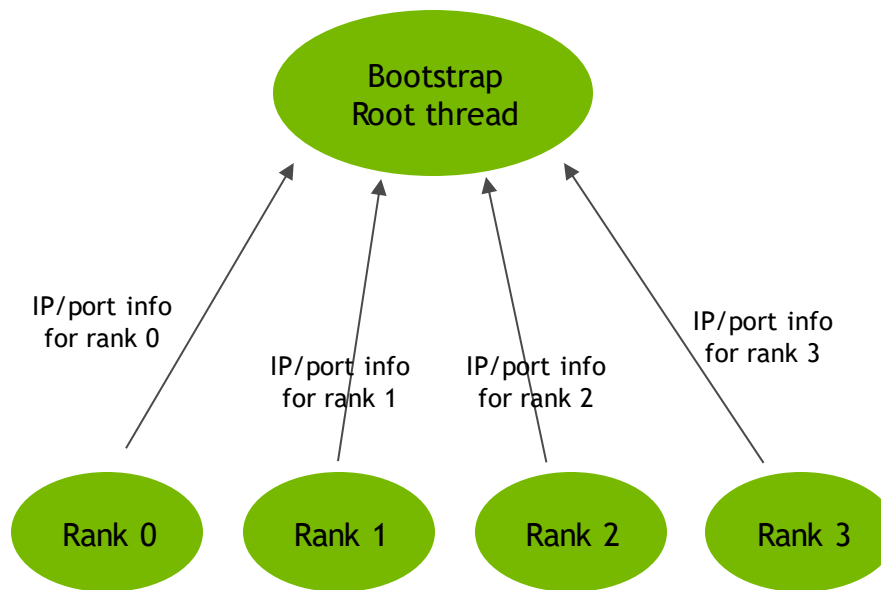
Principle

Once (typically on worker0):

```
ncclUniqueId id;  
ncclGetUniqueId(&id);  
broadcast_to_all_ranks(&id);
```

On all parallel workers:

```
get_unique_id(&id);  
ncclCommInitRank(&comm, nranks, &id, rank);  
ncclAllReduce(..., comm);
```



NCCL BOOTSTRAP

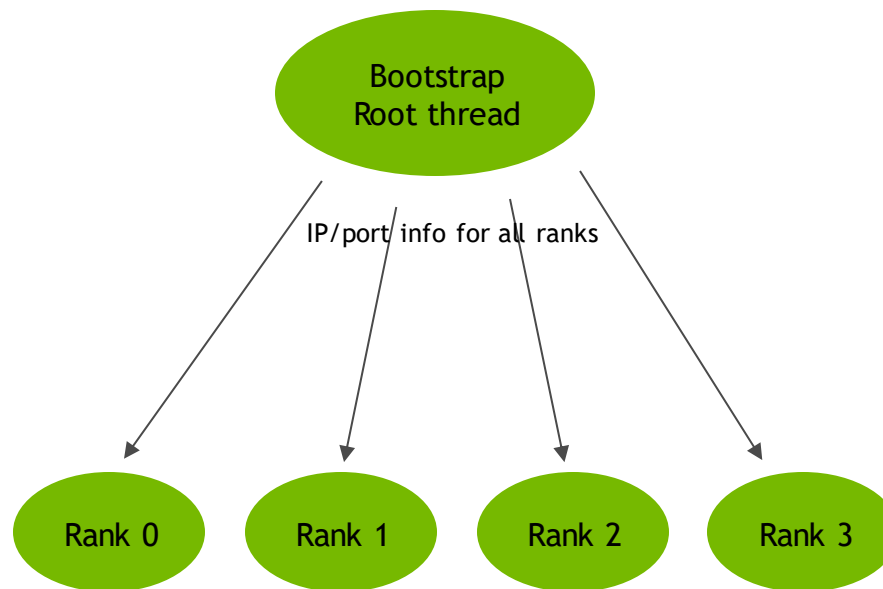
Principle

Once (typically on worker0):

```
ncclUniqueId id  
ncclGetUniqueId(&id);  
broadcast_to_all_ranks(&id);
```

On all parallel workers:

```
get_unique_id(&id);  
ncclCommInitRank(&comm, nranks, &id, rank);  
ncclAllReduce(..., comm);
```



NCCL BOOTSTRAP

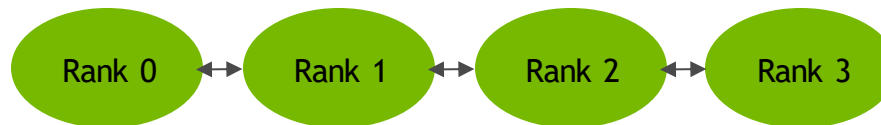
Principle

Once (typically on worker0):

```
ncclUniqueId id;  
ncclGetUniqueId(&id);  
broadcast_to_all_ranks(&id);
```

On all parallel workers:

```
get_unique_id(&id);  
ncclCommInitRank(&comm, nranks, &id, rank);  
ncclAllReduce(..., comm);
```



Allgather rank information
Allgather ring/tree information
Exchange connection information as
needed (ports, IB queue pair, ...)

NCCL BOOTSTRAP

Principle

NCCL Bootstrap uses plain TCP/IP sockets to connect ranks of the same job. It then provides an out-of-band channel to exchange information between ranks as needed.

Bootstrap operations are available during the entire life of the NCCL communicator. They are mostly used during init, but can also be used for dynamically connected send/recv operations.

There is currently no encryption, no security. Use `NCCL_SOCKET_IFNAME` to make sure NCCL uses a network interface which is private to your parallel job.



COMMUNICATION ARCHITECTURE

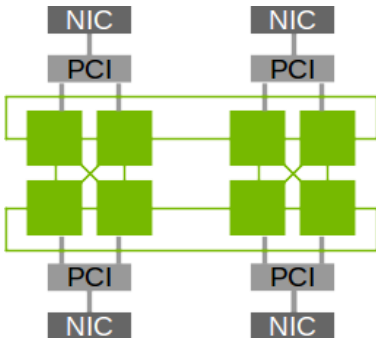
NCCL ARCHITECTURE

Optimized kernels for all platforms

Topology detection

Build graph with all GPUs, NICs, CPUs, PCI switches, NVLink, NVSwitch.

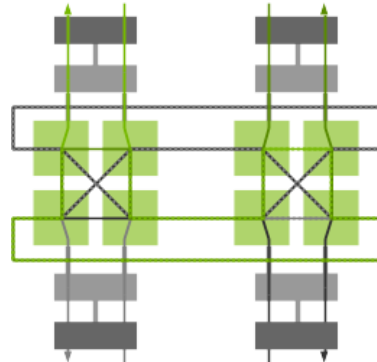
Topology injection for VMs.



Graph search

Extensive search to find optimal set of rings or trees.

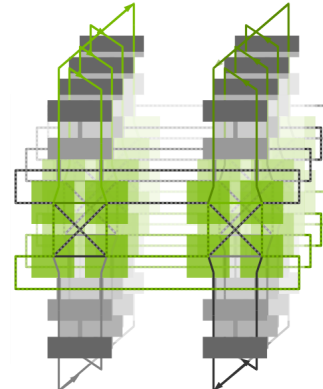
Performance prediction of each algorithm and auto-tuning.



Graph connect

Connect graphs between nodes.

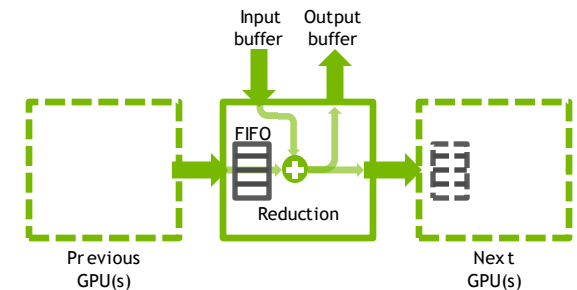
Connect GPUs with intermediate FIFOs, using PCI, NVLink, GPU Direct RDMA, ...



CUDA Kernels

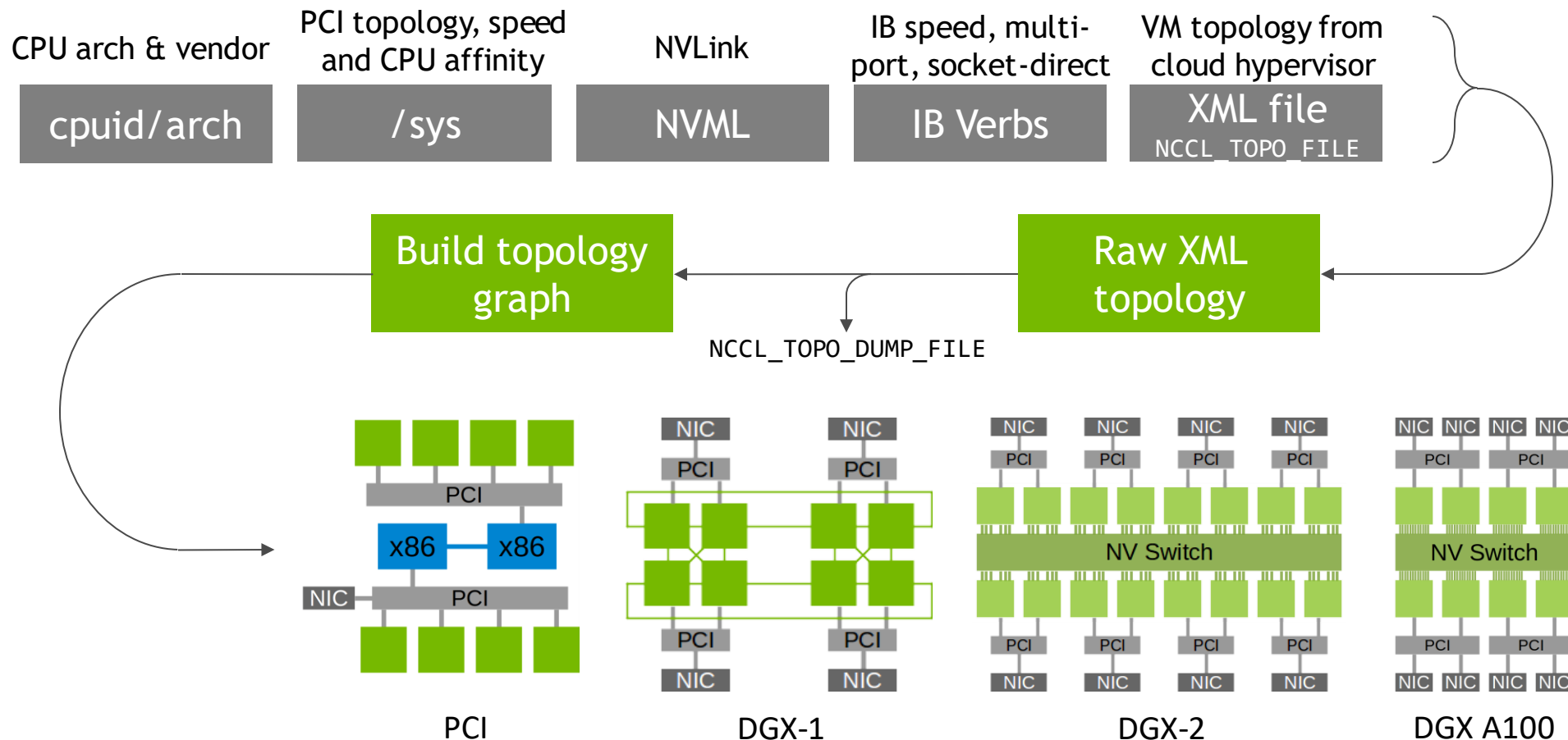
Optimized reductions and copies for a minimal SM usage.

CPU threads for network communication.



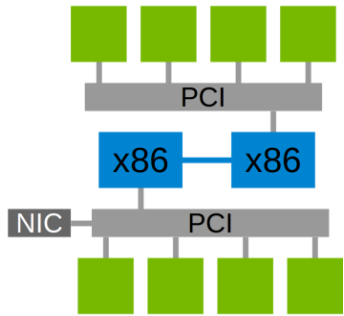
TOPOLOGY DETECTION

Extensive support for all platforms

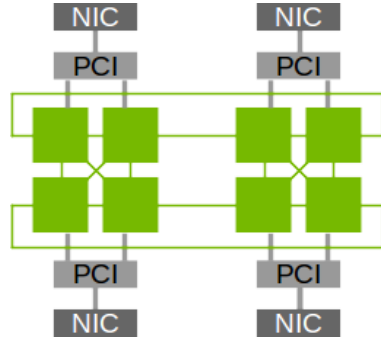


FROM TOPOLOGY TO GRAPHS

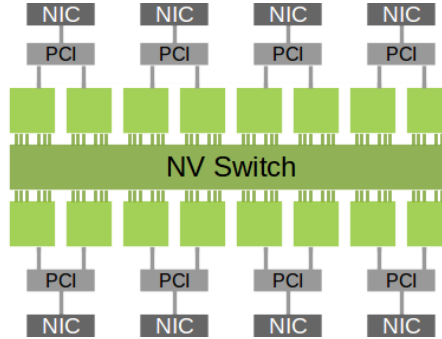
Intra-node graph search



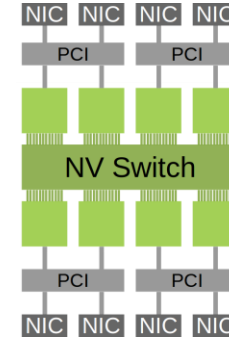
PCI



DGX-1

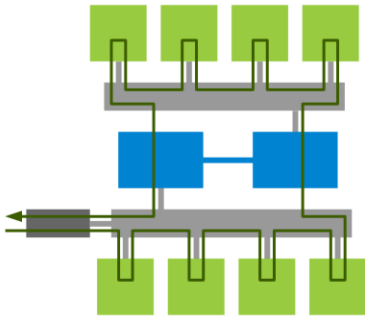


DGX-2

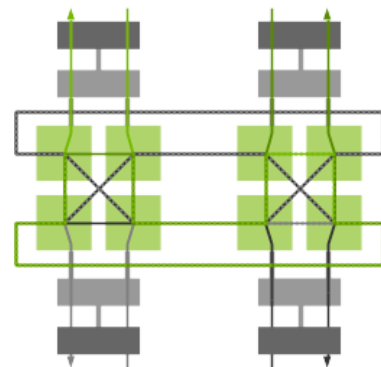


DGX A100

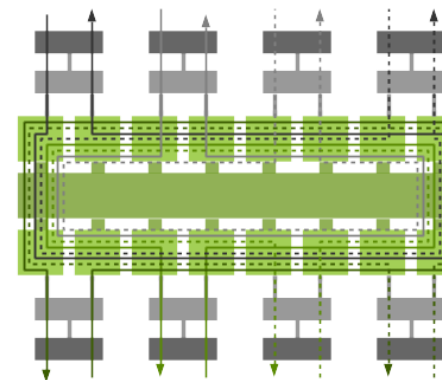
Multi-path search within the node to maximize intra- and inter- node bandwidth.



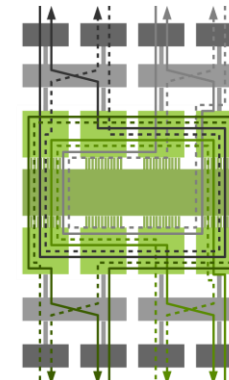
9 GB/s



48 GB/s



85 GB/s



192 GB/s

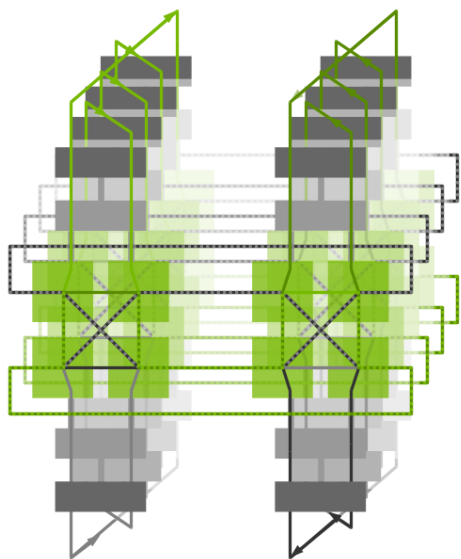
Model latency and bandwidth for each algorithm and protocol based on the number of channels and speed



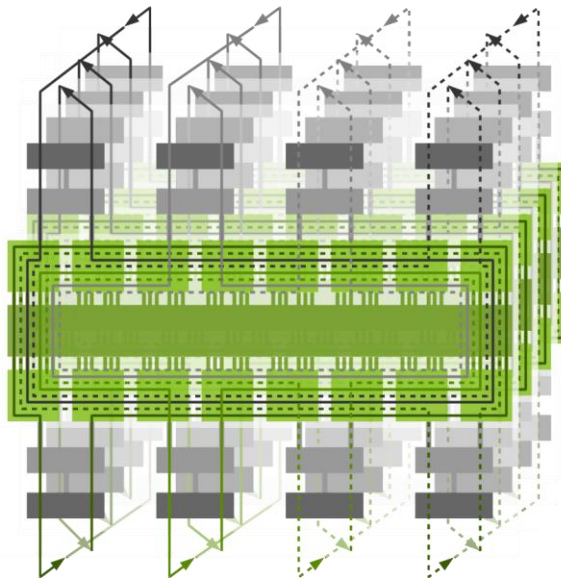
INTER-NODE COMMUNICATION

INTER-NODE COMMUNICATION

Connect rails together



DGX-1



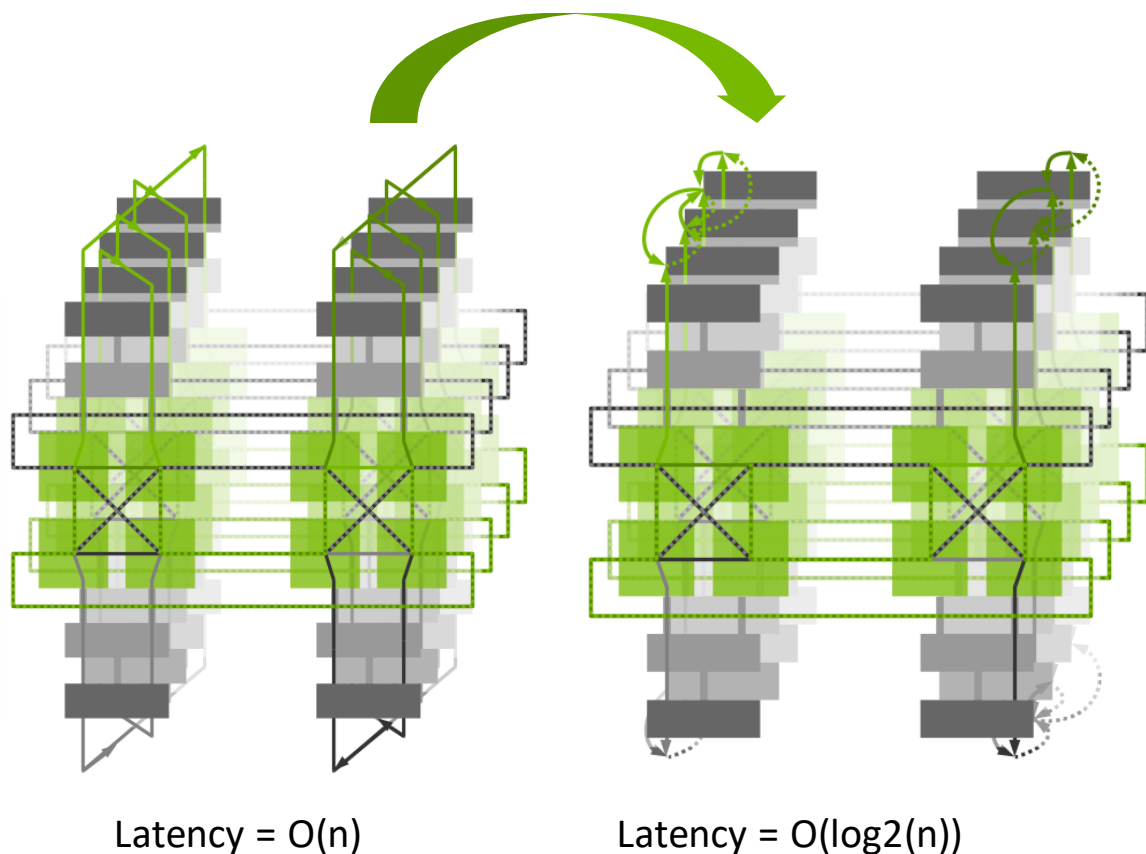
DGX-2

Rings (and trees) from each node are connected to each other.

It is assumed that same NICs can communicate efficiently across nodes

INTER-NODE COMMUNICATION

Rings vs Trees



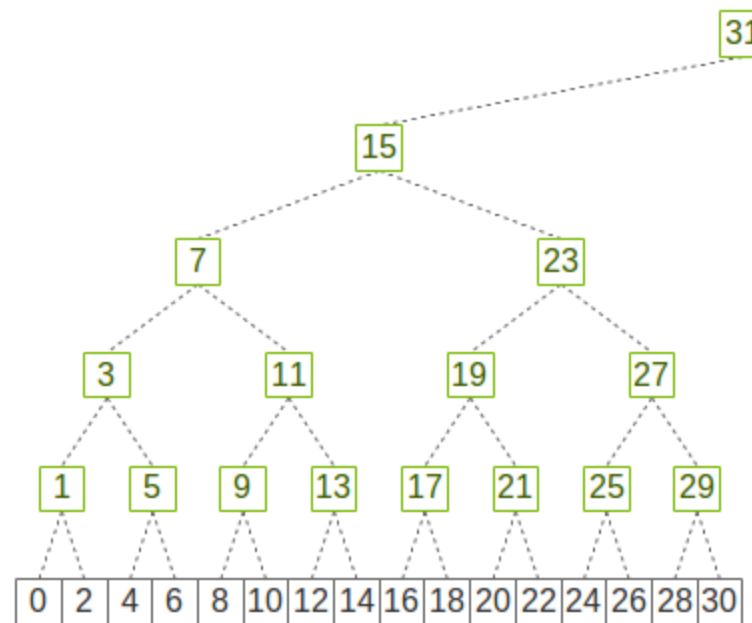
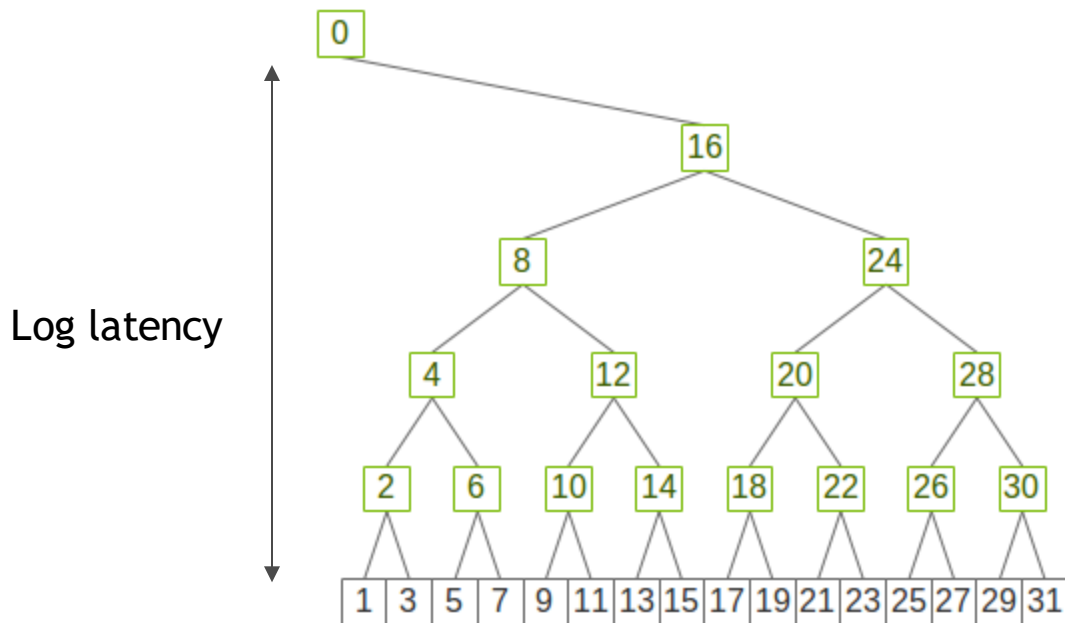
Similarly, trees are connected between nodes.

Intra-node still aggregates the bandwidth of multiple NICs.

NICs still communicate along planes.

INTER-NODE COMMUNICATION

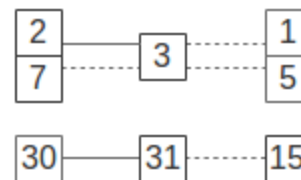
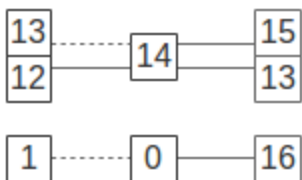
Dual binary tree



Power of two
pattern

Maximizes local
communication

Half the data goes
through each tree,
achieving full bandwidth





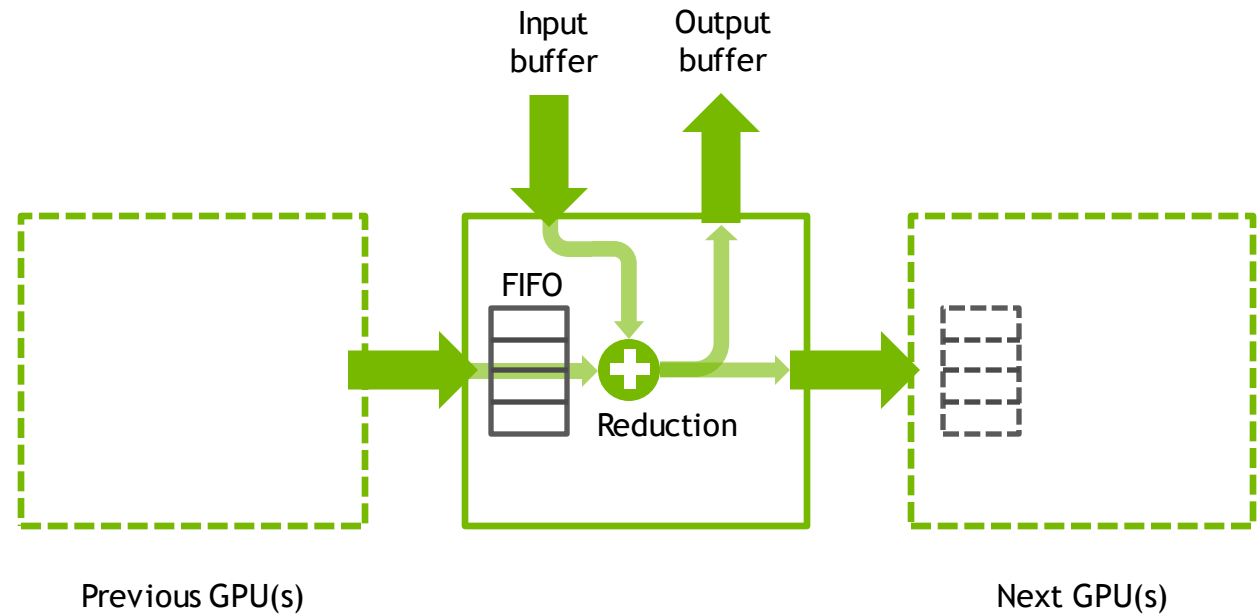
CUDA KERNELS

GPU COMMUNICATION KERNEL

Principle

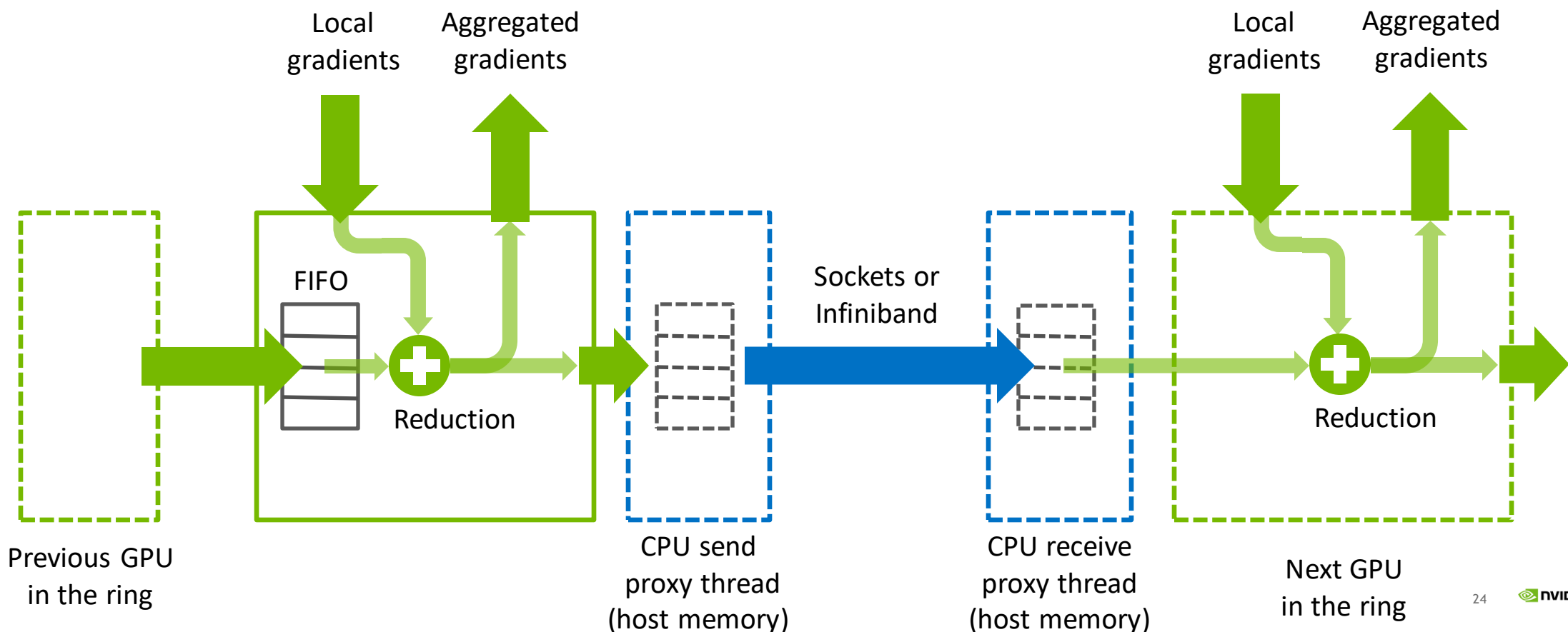
NCCL CUDA Kernel

- runs on the GPU
- receives and sends from other peers through internal FIFOs
- perform reductions and copies with local and remote buffers



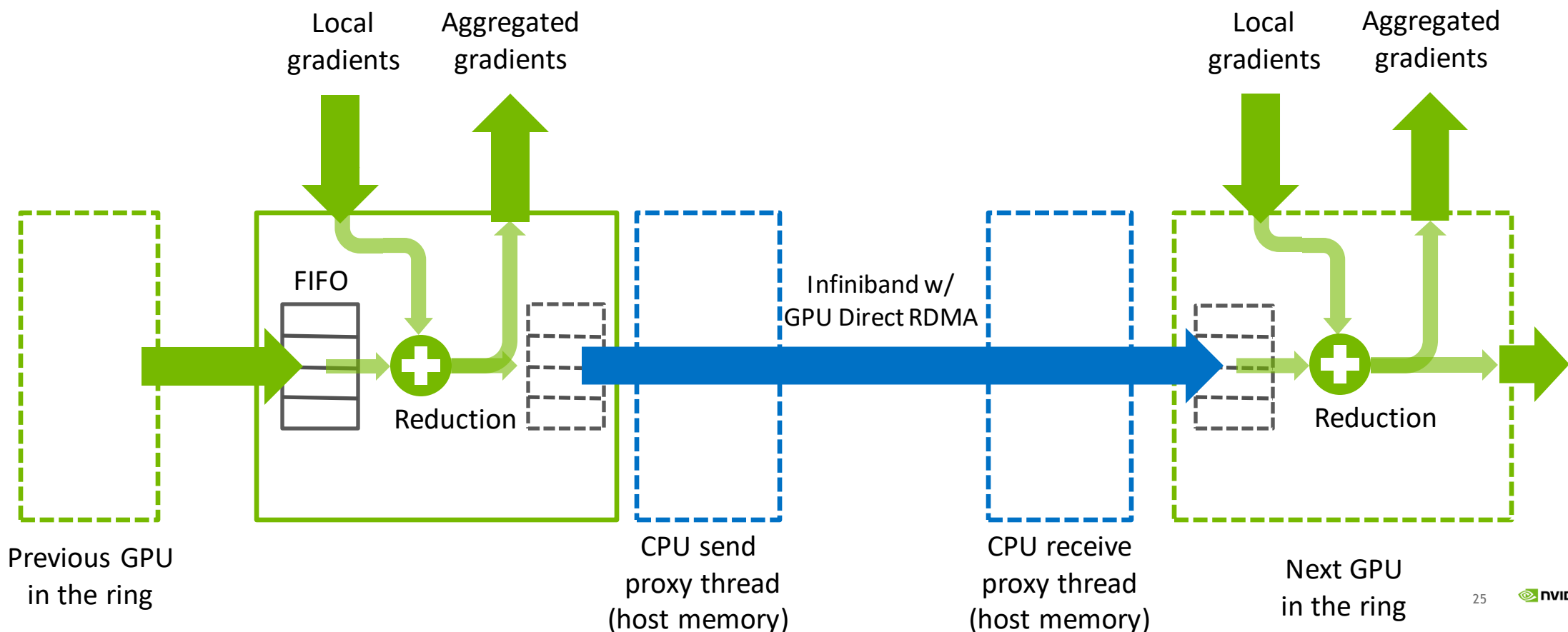
INTER-NODE COMMUNICATION

Network proxy



INTER-NODE COMMUNICATION

GPU Direct RDMA





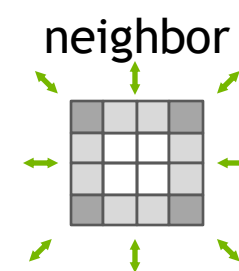
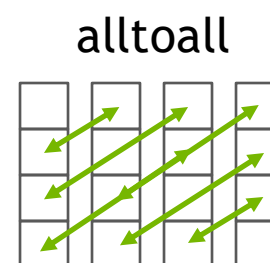
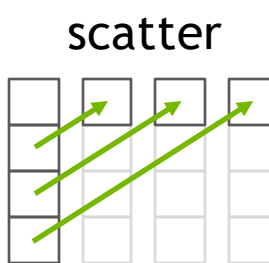
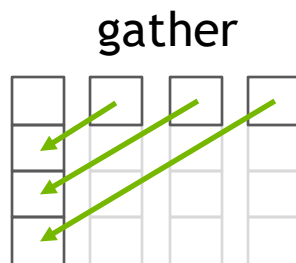
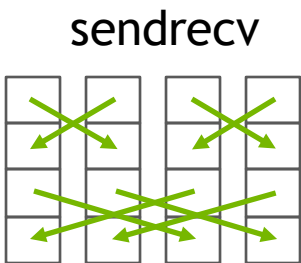
SEND/RECEIVE

SEND/RECEIVE

Communication semantics

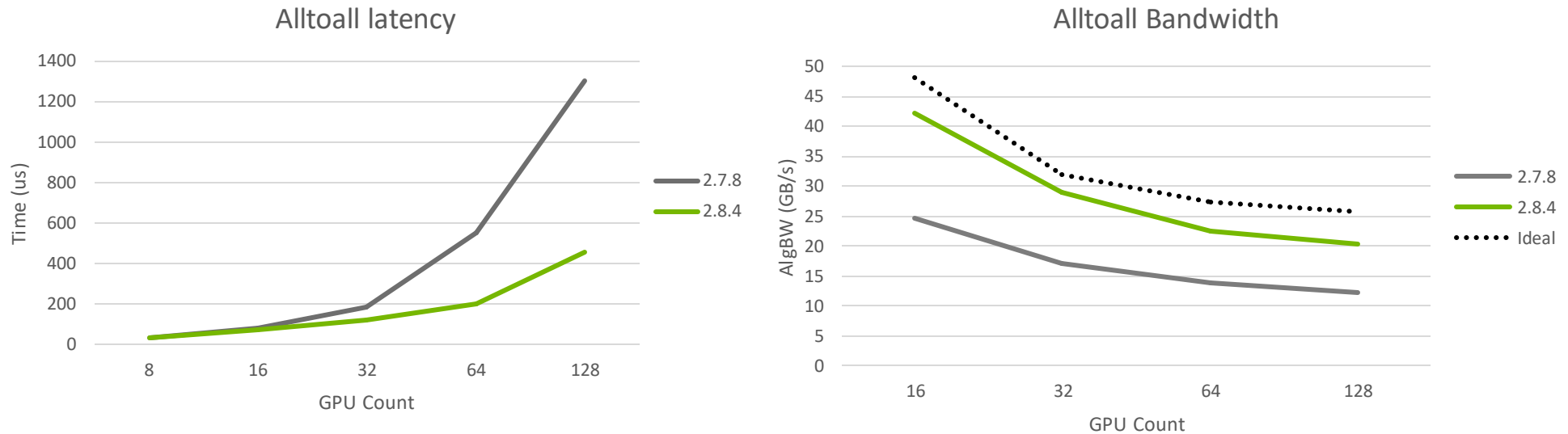
```
ncclGroupStart();  
ncclSend(sbuf, ssize, sdtype, peer);  
ncclRecv(rbuf, rsize, rdtype, peer);  
ncclGroupEnd();
```

Create any operation involving sending and receiving to/from different peers.
Grouping operation together is needed to guarantee forward progress and no deadlock.



SEND/RECEIVE

Optimizations (Cont'd)

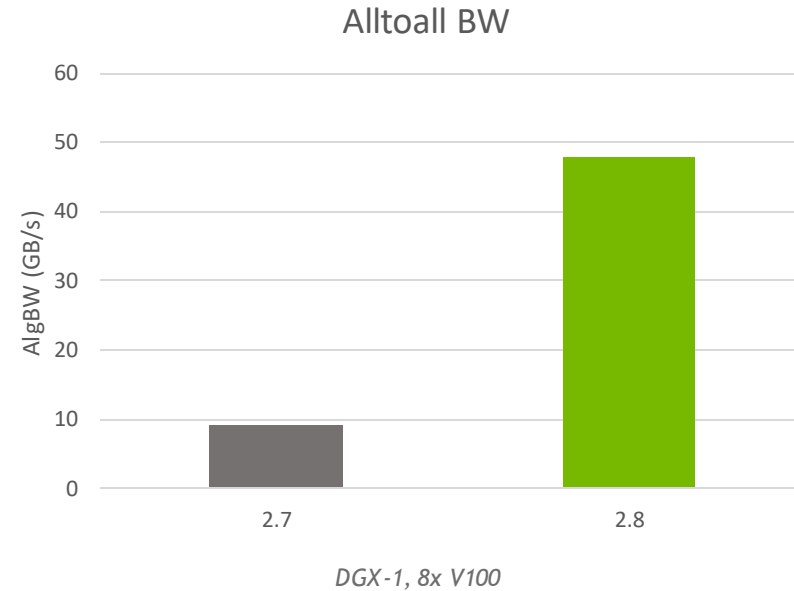
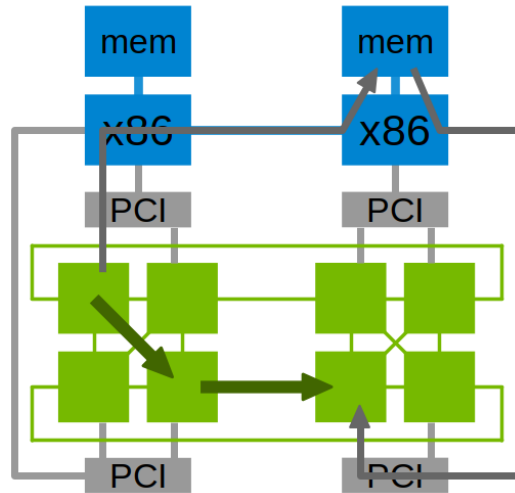


DGX A100, 8x A100, 8x IB HDR 200G, IB adaptive routing enabled

NCCL 2.8: increased send/receive parallelism and better NVLink/IB balancing.

SEND/RECEIVE

Optimizations



One hop NVLink for DGX-1 cube-mesh: alltoall >5x faster than communicating through shared memory.

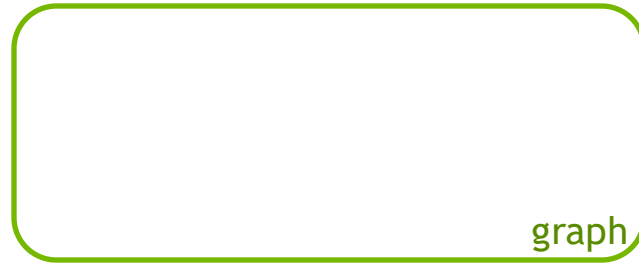


CUDA GRAPHS

CUDA GRAPHS

NCCL integration

```
cudaStreamBeginCapture(stream,  
    cudaStreamCaptureModeGlobal);  
cudaKernel1<<<..., stream>>>(...);  
ncclAllreduce(..., stream);  
cudaKernel2<<<..., stream>>>(...);  
cudaStreamEndCapture(stream,  
    &graph);  
  
cudaGraphLaunch(instance, stream);  
  
cudaStreamSynchronize(stream);
```

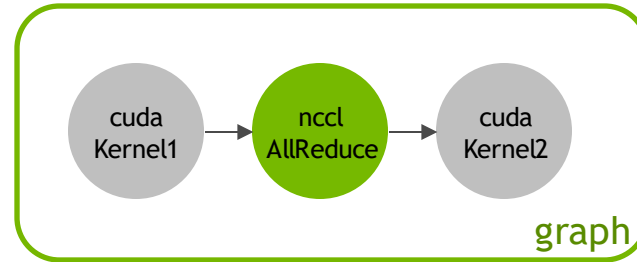


Requires CUDA 11.3

CUDA GRAPHS

NCCL integration

```
cudaStreamBeginCapture(stream,  
    cudaStreamCaptureModeGlobal);  
cudaKernel1<<<..., stream>>>(…);  
ncclAllreduce(..., stream);  
cudaKernel2<<<..., stream>>>(…);  
cudaStreamEndCapture(stream,  
&graph);  
  
cudaGraphLaunch(instance, stream);  
  
cudaStreamSynchronize(stream);
```



Requires CUDA 11.3

CUDA GRAPHS

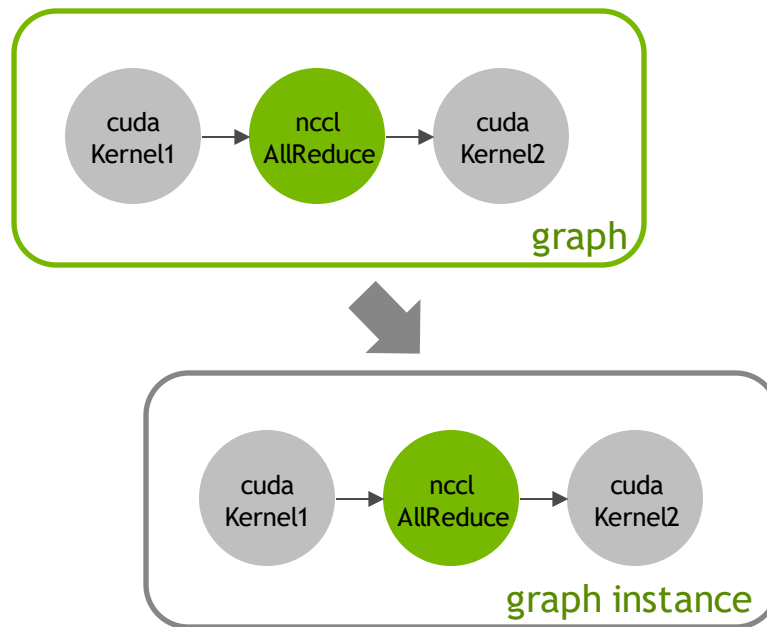
NCCL integration

```
cudaStreamBeginCapture(stream,  
    cudaStreamCaptureModeGlobal);  
cudaKernel1<<<..., stream>>>(...);  
ncclAllreduce(..., stream);  
cudaKernel2<<<..., stream>>>(...);  
cudaStreamEndCapture(stream,  
    &graph);
```

```
cudaGraphLaunch(instance, stream);
```

```
cudaStreamSynchronize(stream);
```

Requires CUDA 11.3



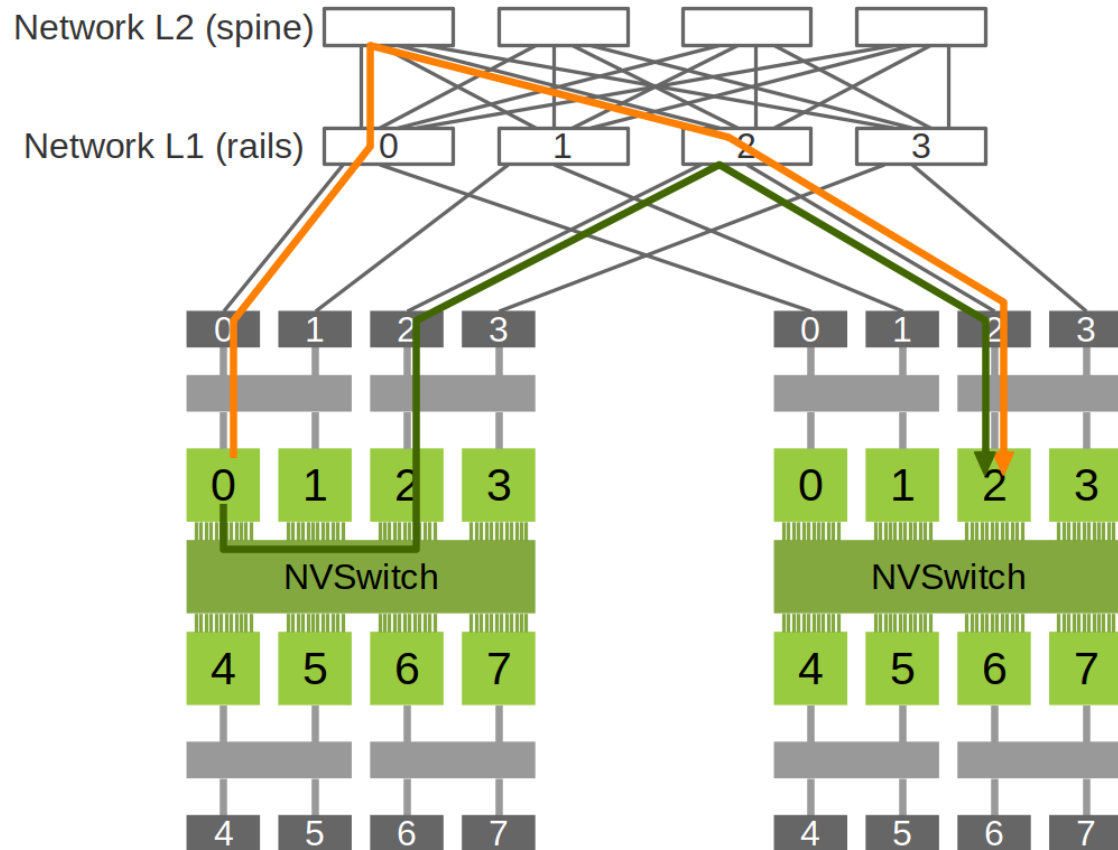


FUTURE

FUTURE

Rail-local alltoall

Use NVLink to avoid NIC crossing for send/receive operations.



FUTURE

ncclAvg

New “Average” operation, computes the sum divided by the number of ranks.

Allows to skip the buffer fusion and scaling step when combined with aggregation.

Fusion+Scaling
Single NCCL
AllReduce (Sum)



Grouped NCCL
AllReduce(Avg)

```
for (int b=0; b<nbuffers; b++) {  
    copy_and_scale(global_buffer+offset,  
                    buffers[b], nranks);  
}  
ncclAllReduce(global_buffer, ncclSum);
```

```
ncclGroupStart();  
for (int b=0; b<nbuffers; b++) {  
    ncclAllReduce(buffers[b], ncclAvg);  
}  
ncclGroupEnd();
```

FUTURE

Misc

Support for bfloat16.

Add `ncclGetLastError()` to get the `WARN()` error messages through the NCCL API.

Improved fault tolerance for `ncclCommInit*/ncclCommDestroy`.

Performance improvement for all platforms.



SUMMARY

NCCL

Optimized inter-GPU communication for CUDA applications

Optimized for all NVIDIA platforms, most OEMs and Cloud
Scales to 10,000s of GPUs.

Covers all communication needs for multi-GPU computing.

Requires only CUDA. Designed to easily integrate in any parallel environment (MPI or other).

Binaries : <https://developer.nvidia.com/nccl> and in NGC containers

Source code : <https://github.com/nvidia/nccl>

Perf tests : <https://github.com/nvidia/nccl-tests>

CWES1084: Multi-GPU Programming with CUDA, GPUDirect, NCCL, NVSHMEM, and MPI
CWES1186: Optimizing, Profiling, and Scaling Deep Learning Training

