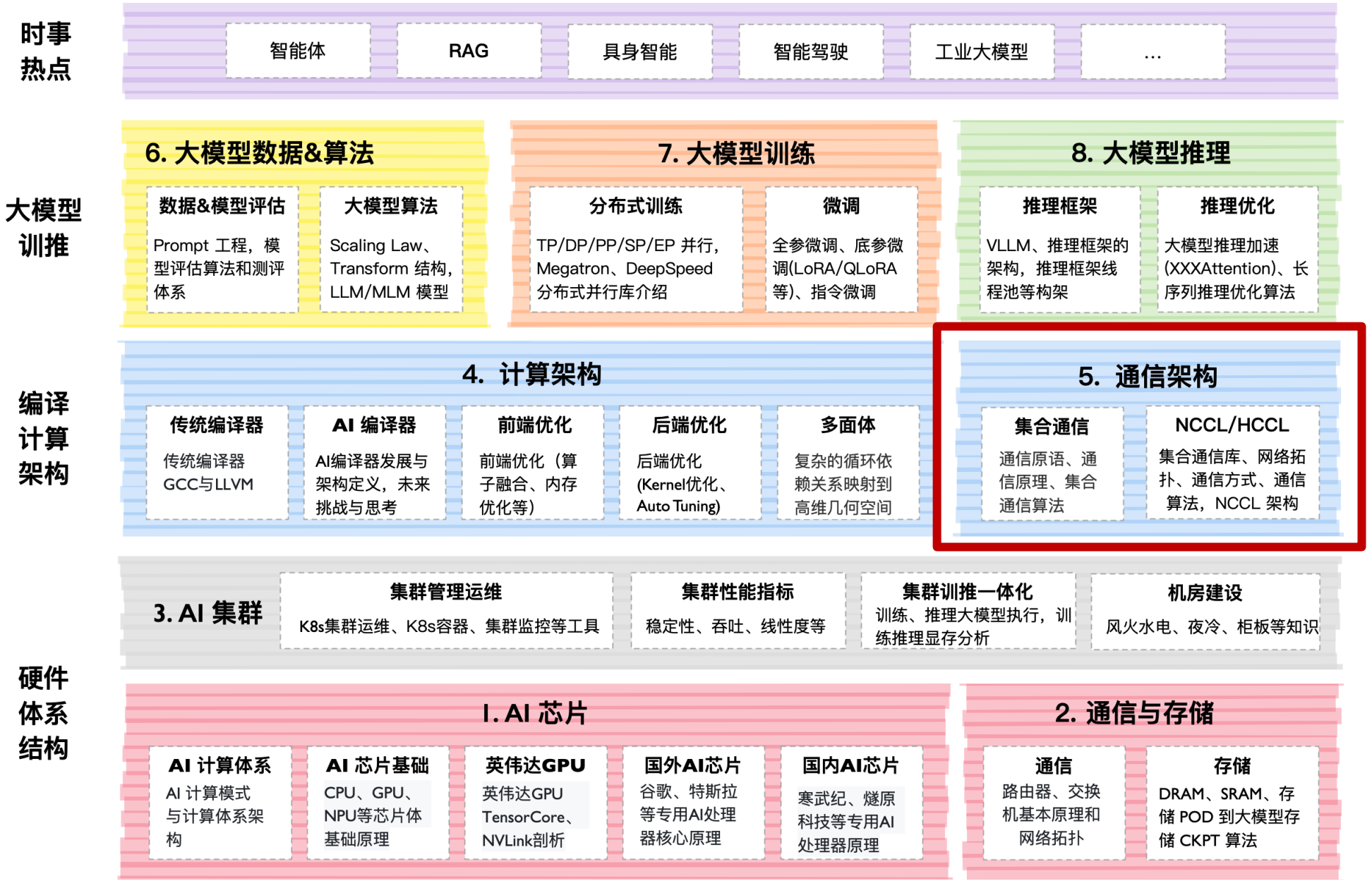
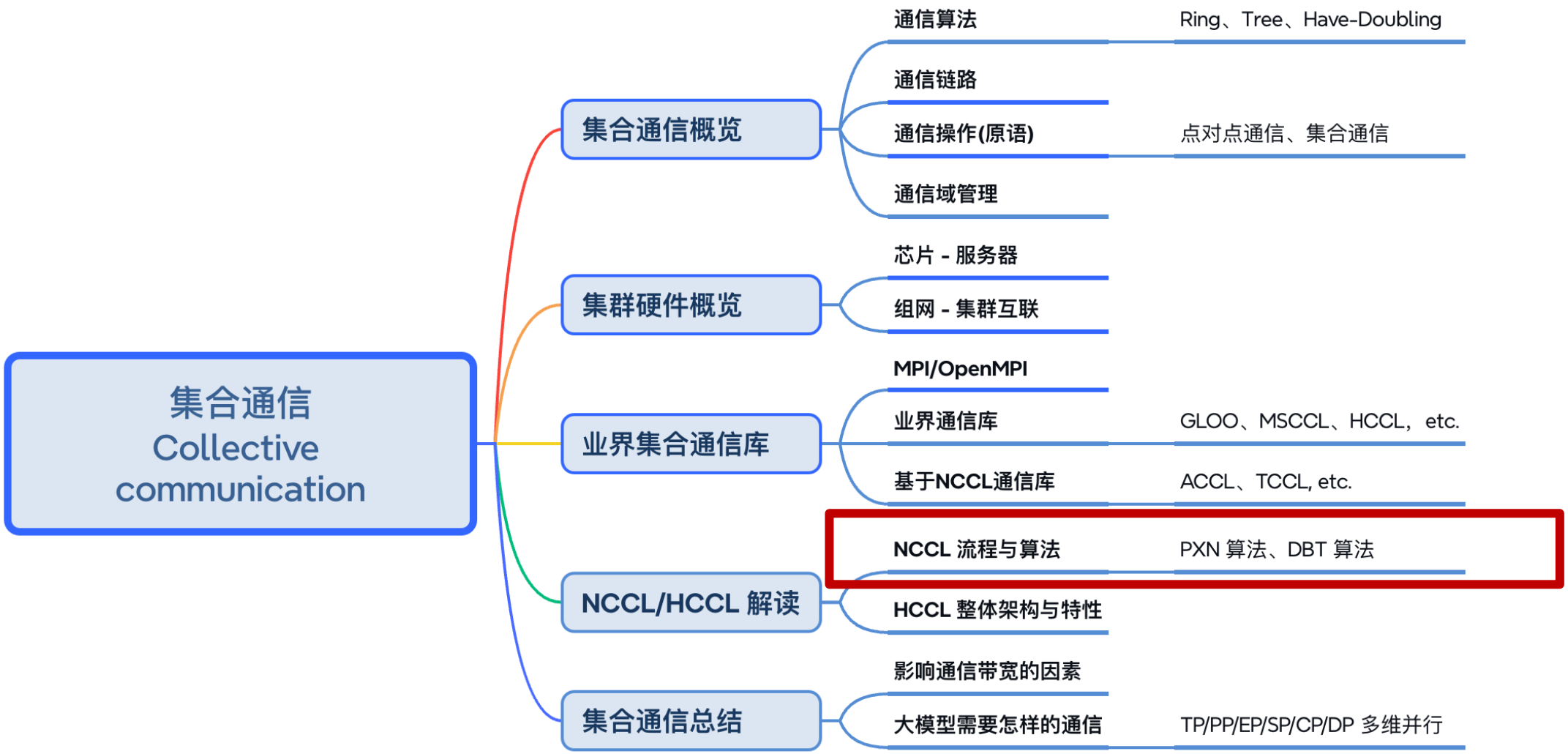
大模型系列 – 集合通信库

NCCL 基本介绍

ZOMI


时事热点
智能体
RAG
具身智能
智能驾驶
工业大模型
...
大模型训推
6. 大模型数据&算法
数据&模型评估
Prompt 工程，模型评估算法和测评体系
大模型算法
Scaling Law、Transform 结构，LLM/MLM 模型
7. 大模型训练
分布式训练
TP/DP/PP/SP/EP 并行，Megatron、DeepSpeed 分布式并行库介绍
微调
全参微调、底参微调(LoRA/QLoRA 等)、指令微调
8. 大模型推理
推理框架
VLLM、推理框架的架构，推理框架线程池等构架
推理优化
大模型推理加速(XXXAttention)、长序列推理优化算法
编译计算架构
4. 计算架构
传统编译器
传统编译器 GCC与LLVM
AI 编译器
AI编译器发展与架构定义，未来挑战与思考
前端优化
前端优化（算子融合、内存优化等）
后端优化
后端优化(Kernel优化、Auto Tuning)
多面体
复杂的循环依赖关系映射到高维几何空间
5. 通信架构
集合通信
通信原语、通信原理、集合通信算法
NCCL/HCCL
集合通信库、网络拓扑、通信方式、通信算法，NCCL 架构
硬件体系结构
3. AI 集群
集群管理运维
K8s集群运维、K8s容器、集群监控等工具
集群性能指标
稳定性、吞吐、线性度等
集群训推一体化
训练、推理大模型执行，训练推理显存分析
机房建设
风火水电、夜冷、柜板等知识
1. AI 芯片
AI 计算体系
AI 计算模式与计算体系架构
AI 芯片基础
CPU、GPU、NPU等芯片体基础原理
英伟达GPU
英伟达GPU TensorCore、NVLink剖析
国外AI芯片
谷歌、特斯拉等专用AI处理器核心原理
国内AI芯片
寒武纪、燧原科技等专用AI处理器原理
2. 通信与存储
通信
路由器、交换机基本原理和网络拓扑
存储
DRAM、SRAM、存储 POD 到大模型存储 CKPT 算法

思维导图 XMind


集合通信
Collective
communication
集合通信概览
通信算法
Ring、Tree、Have-Doubling
通信链路
通信操作(原语)
点对点通信、集合通信
通信域管理
集群硬件概览
芯片 - 服务器
组网 - 集群互联
业界集合通信库
MPI/OpenMPI
业界通信库
GLOO、MSCCL、HCCL，etc.
基于NCCL通信库
ACCL、TCCL, etc.
NCCL/HCCL 解读
NCCL 流程与算法
PXN 算法、DBT 算法
HCCL 整体架构与特性
集合通信总结
影响通信带宽的因素
大模型需要怎样的通信
TP/PP/EP/SP/CP/DP 多维并行

Question

1. 用 GPU/NPU 跑大模型（训练/云端推理），模型太大所以需要多卡通信，离不开 NCCL 通信库，那么 NCCL 到底有什么，是什么呢?

本节内容

1. NCCL 基本介绍 NCCL Introduction
2. NCCL 初始化 Init/Bootstrap
3. 通信架构 Communication Architecture
4. 节点间通信 Inter-node Communication
5. CUDA 执行核心 CUDA Kernels
6. 数据发送和接收 Send/Receive
7. 通信协议 Communication Protocol

1. NCCL 基本介绍

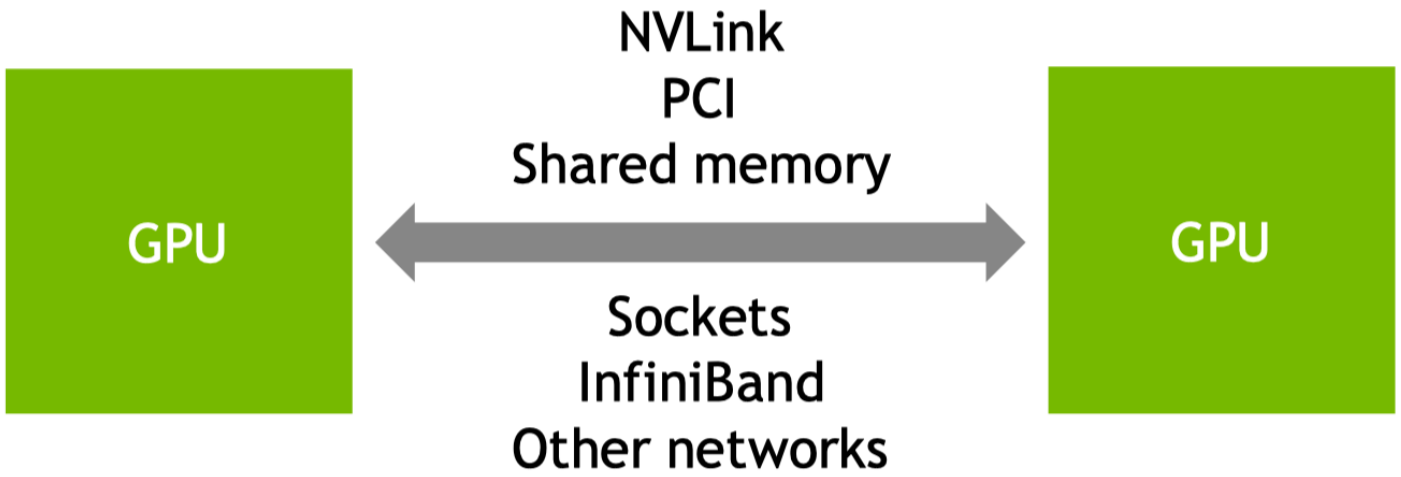
NCCL Introduction

https://developer.nvidia.com/nccl

OPTIMIZED INTER-GPU COMMUNICATION

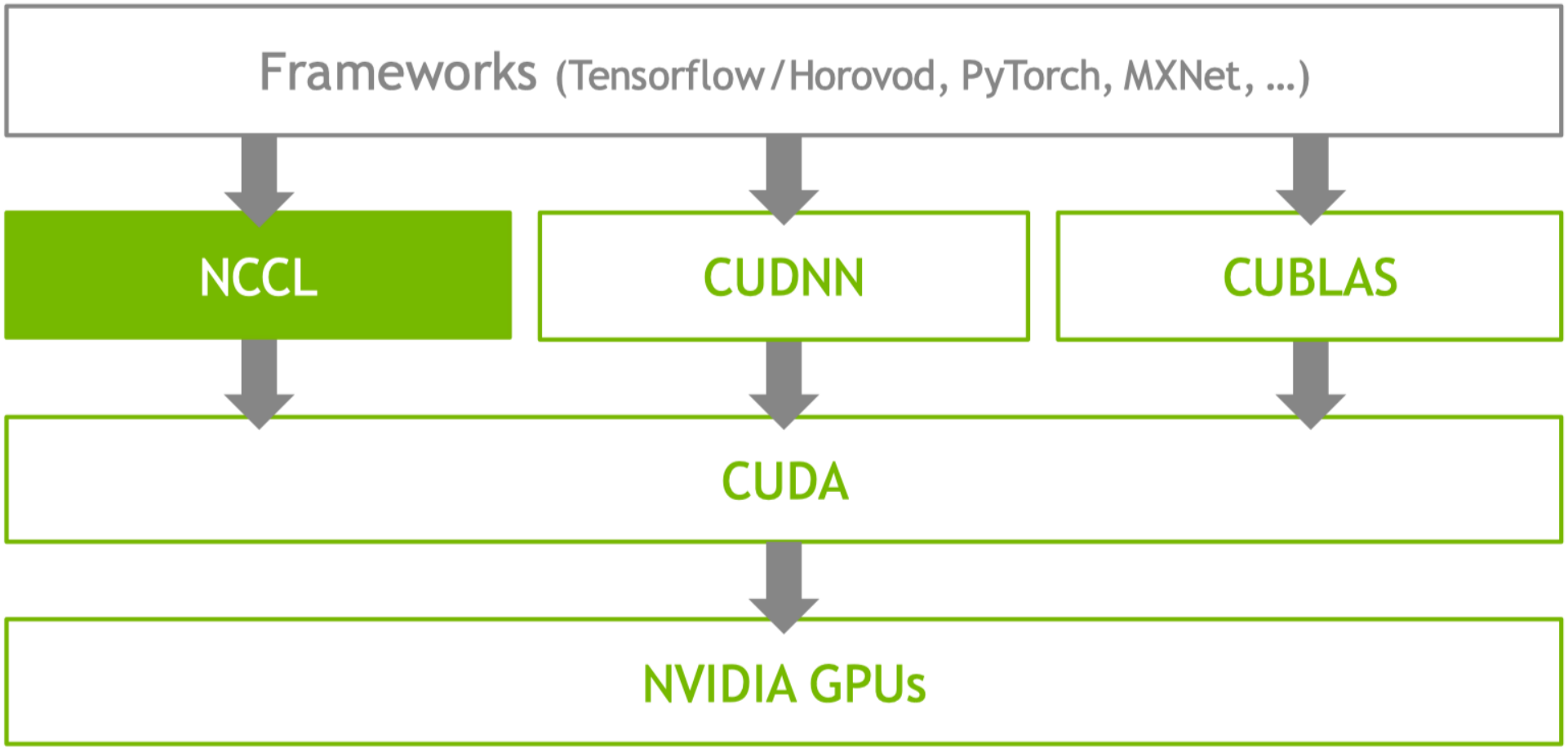
NCCL : NVIDIA Collective Communication Library

Communication library running on GPUs, for GPU buffers.

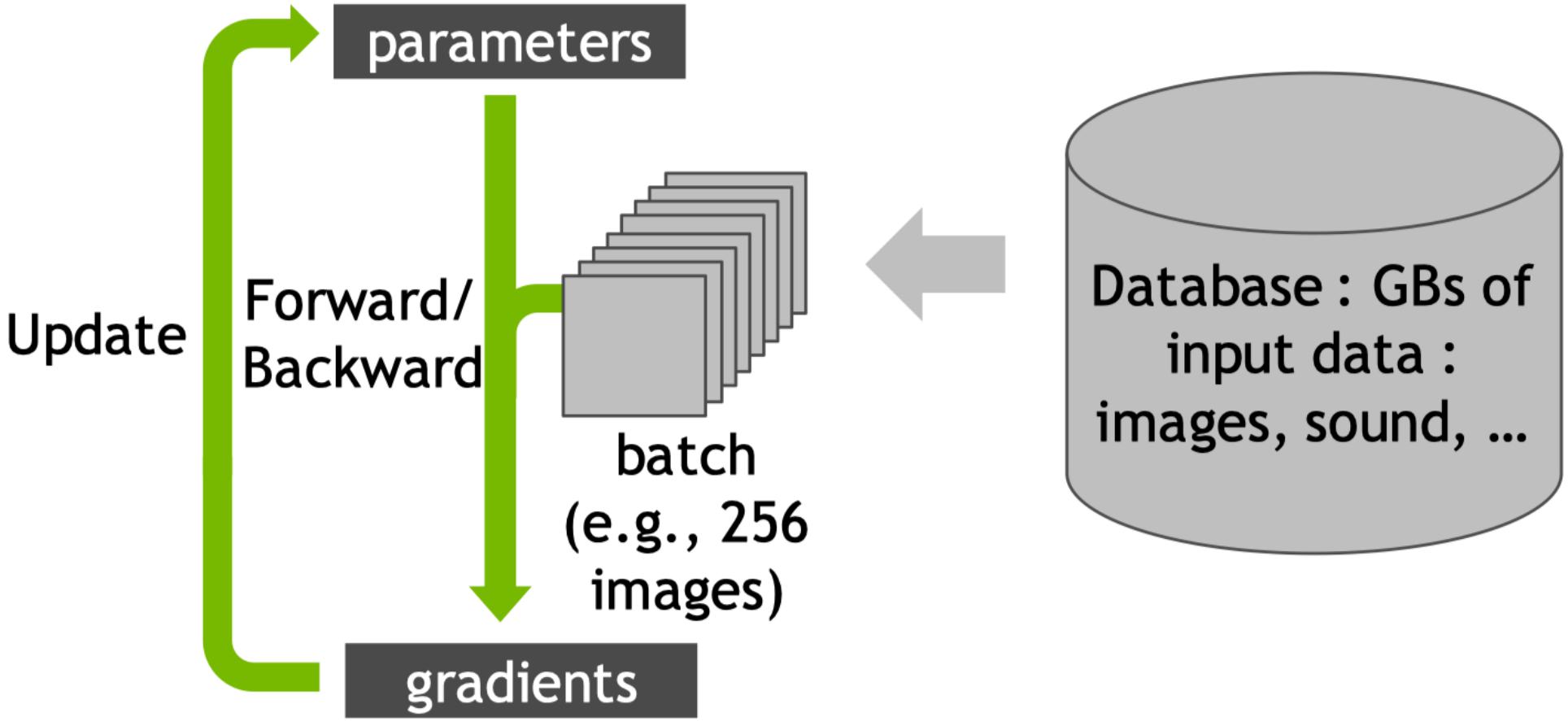


Binaries : https://developer.nvidia.com/nccl and in NGC containers
Source code : https://github.com/nvidia/nccl
Perf tests : https://github.com/nvidia/nccl-tests

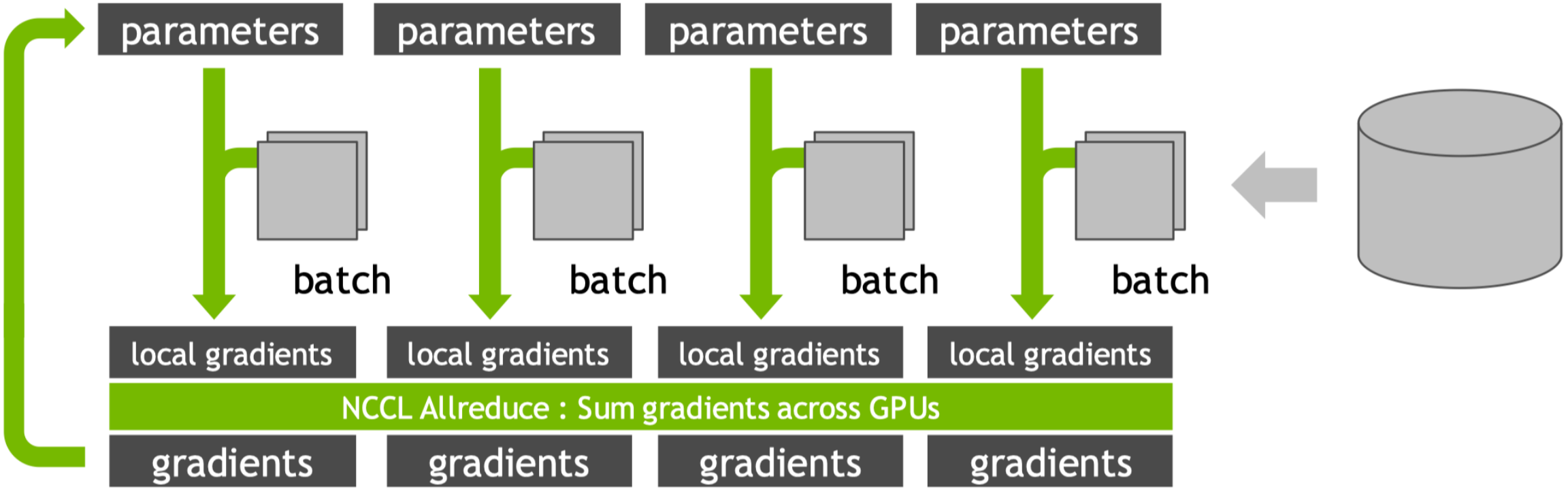
NCCL DL stack



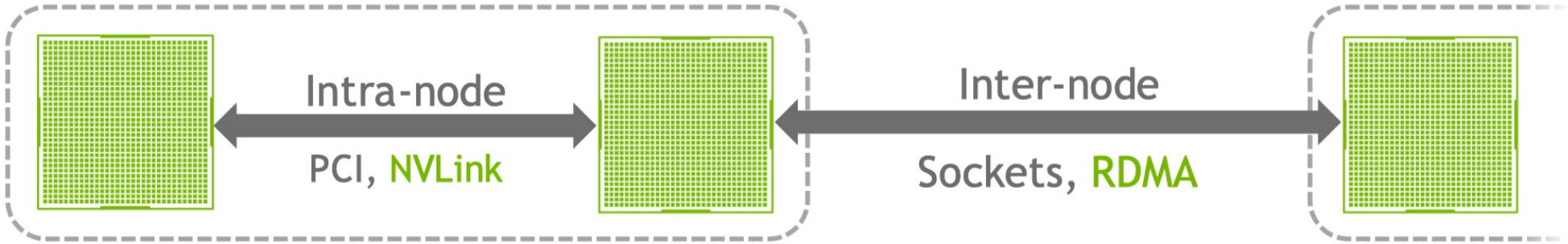
MULTI-GPU TRAINING: Single-GPU

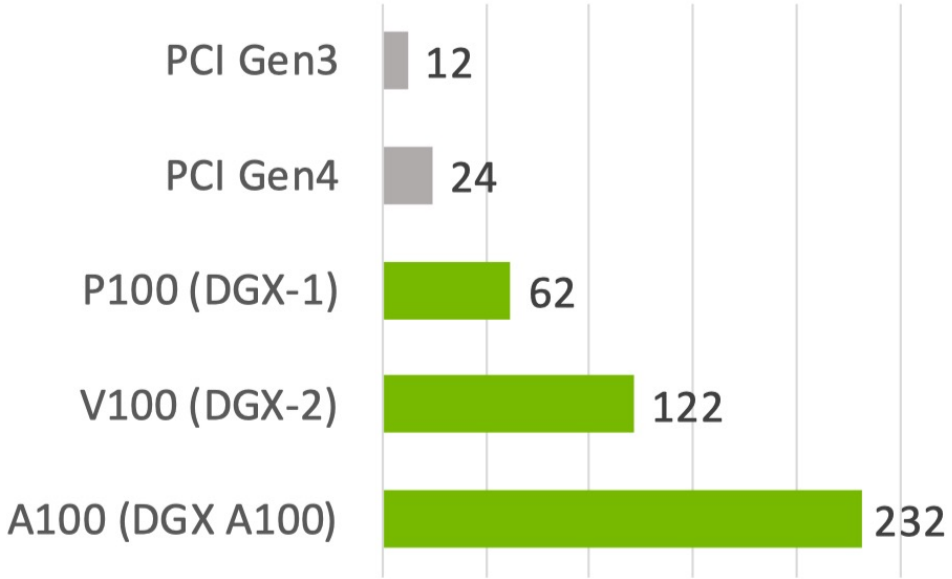


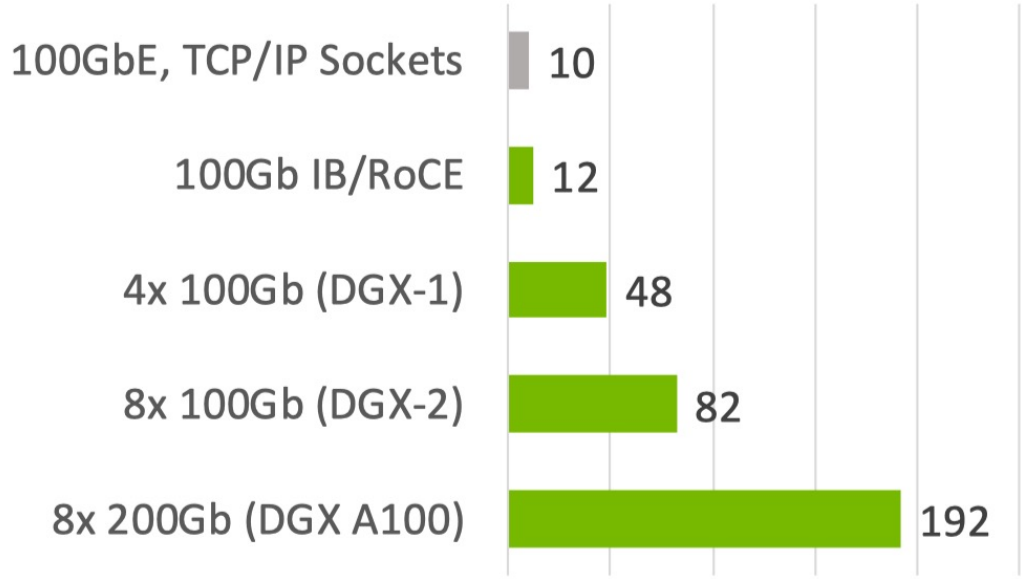
MULTI-GPU TRAINING: Data parallel



INTER-GPU COMMUNICATION







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();
```

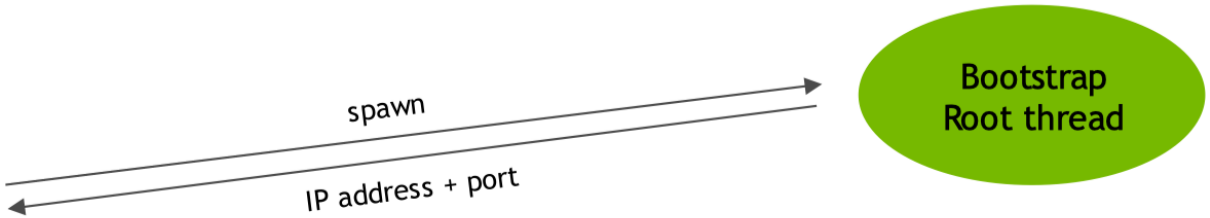
2. 初始化

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);
```

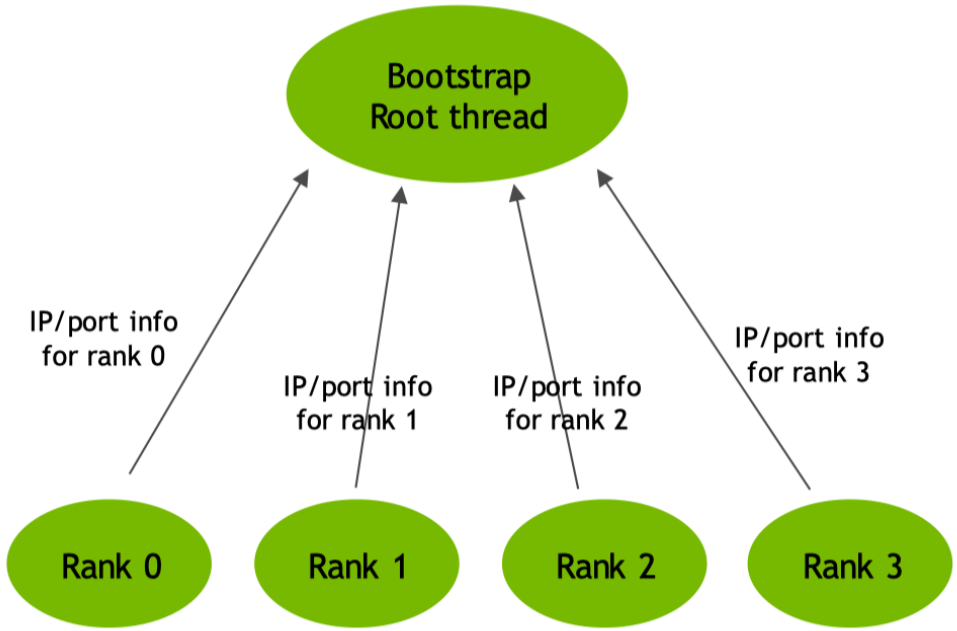
NCCL BOOTSTRAP: Principle

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);
```



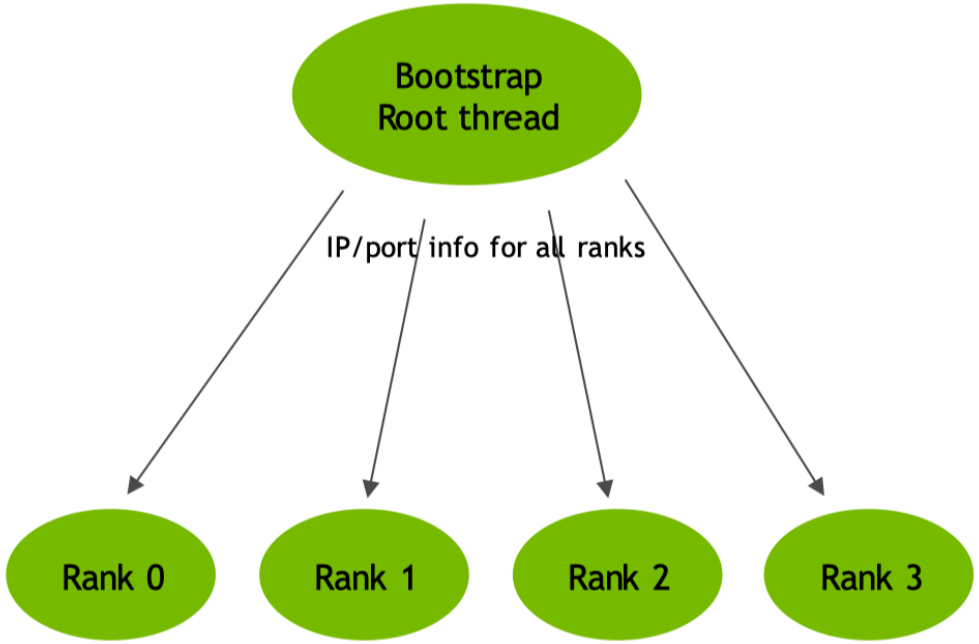
NCCL BOOTSTRAP: Principle

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);
```



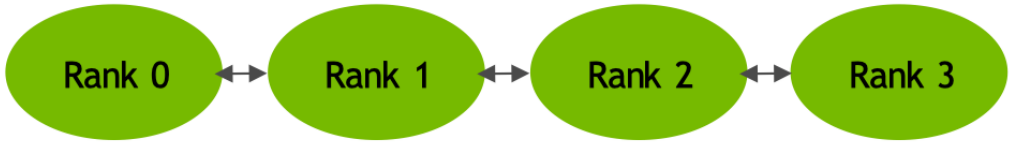
NCCL BOOTSTRAP: Principle

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 在同一个任务中使用 TCP/IP sockets 去连接不同的 Ranks，然后提供一个带外信道在 Ranks 间传输不同的数据。
- 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 操作在 NCCL 整个生命周期内都是可用的，不过主要用于初始化（Init）阶段，当然也可以用于动态连接时 send/recv 操作。
- 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.

03. 通信架构

COMMUNICATION ARCHITECTURE

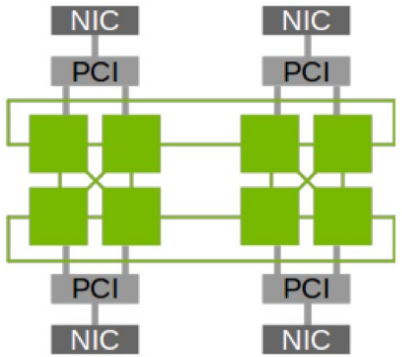
NCCL ARCHITECTURE: Optimized kernels for all platforms

拓扑检测

Topology detection

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

Topology injection for VMs.

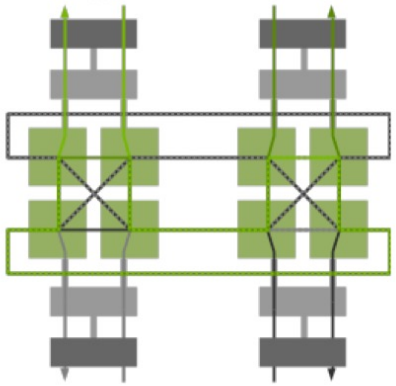


图搜索

Graph search

Extensive search to find optimal set of rings or trees.

Performance prediction of each algorithm and auto-tuning.

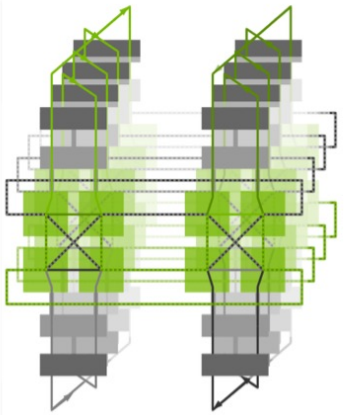


图连接

Graph connect

Connect graphs between nodes.

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

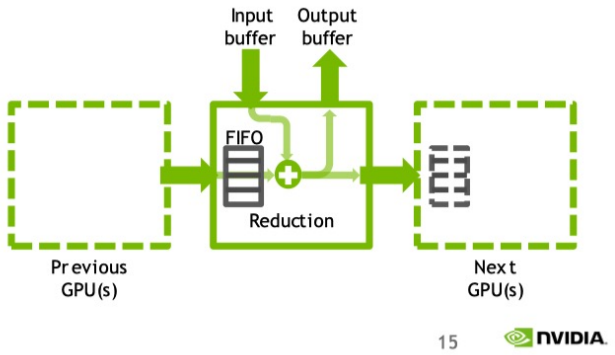


Kernel 执行

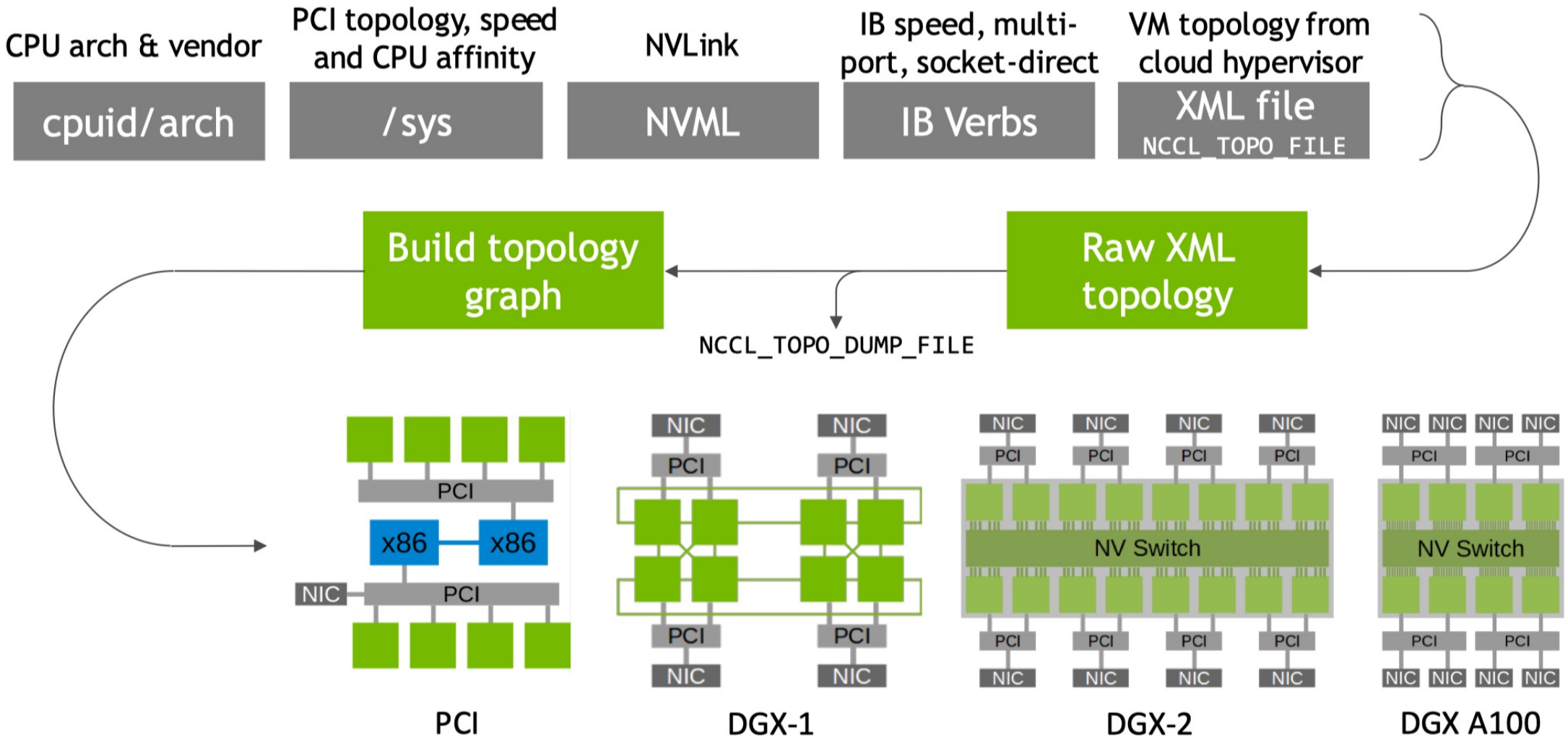
CUDA Kernels

Optimized reductions and copies for a minimal SM usage.

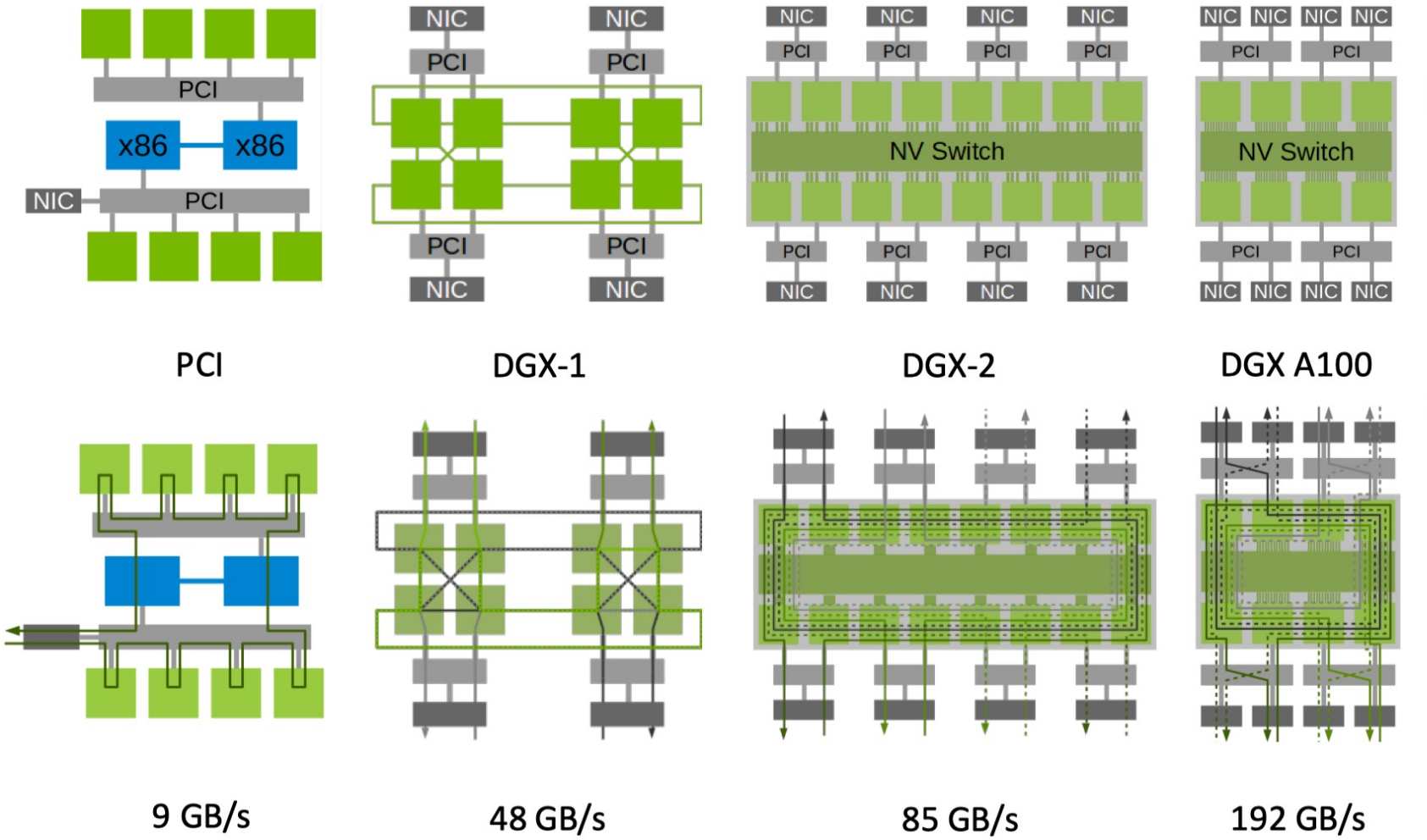
CPU threads for network communication.



拓扑感知：Extensive support for all platforms



从拓扑到构图：Intra-node graph search

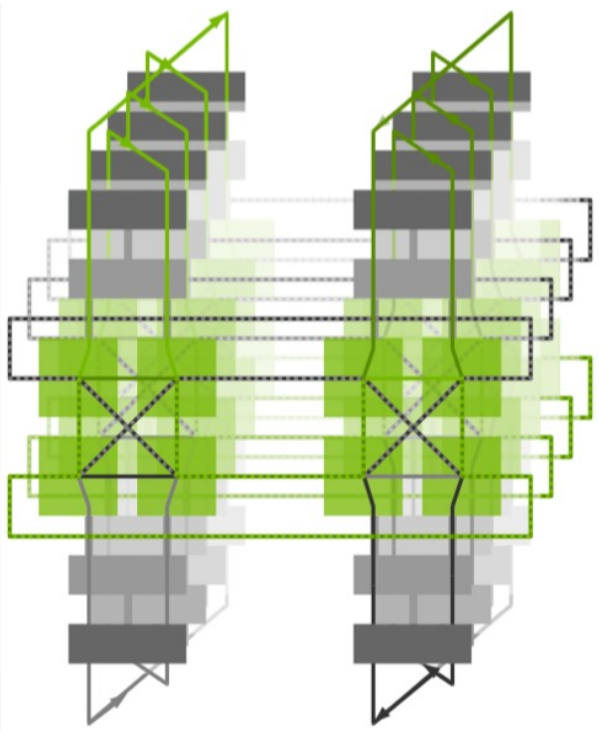


- 节点内进行多路径搜索，最大限度地提高节点内和节点间带宽
- 根据 Channel 数和传输速率对每种算法和协议的延迟和带宽进行建模

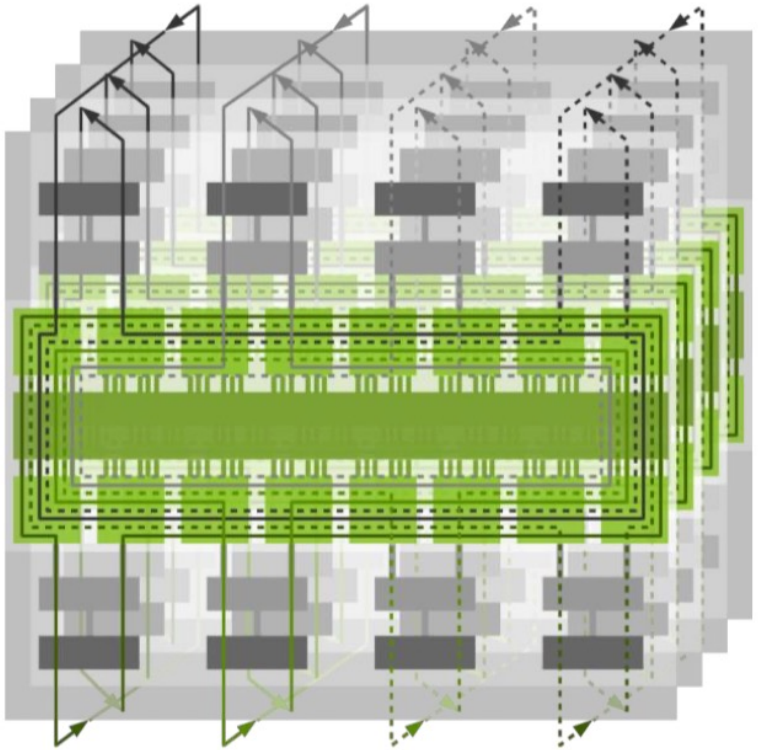
04. 节点间通信

INTER-NODE COMMUNICATION

节点间通信：Connect rails together 多轨连接



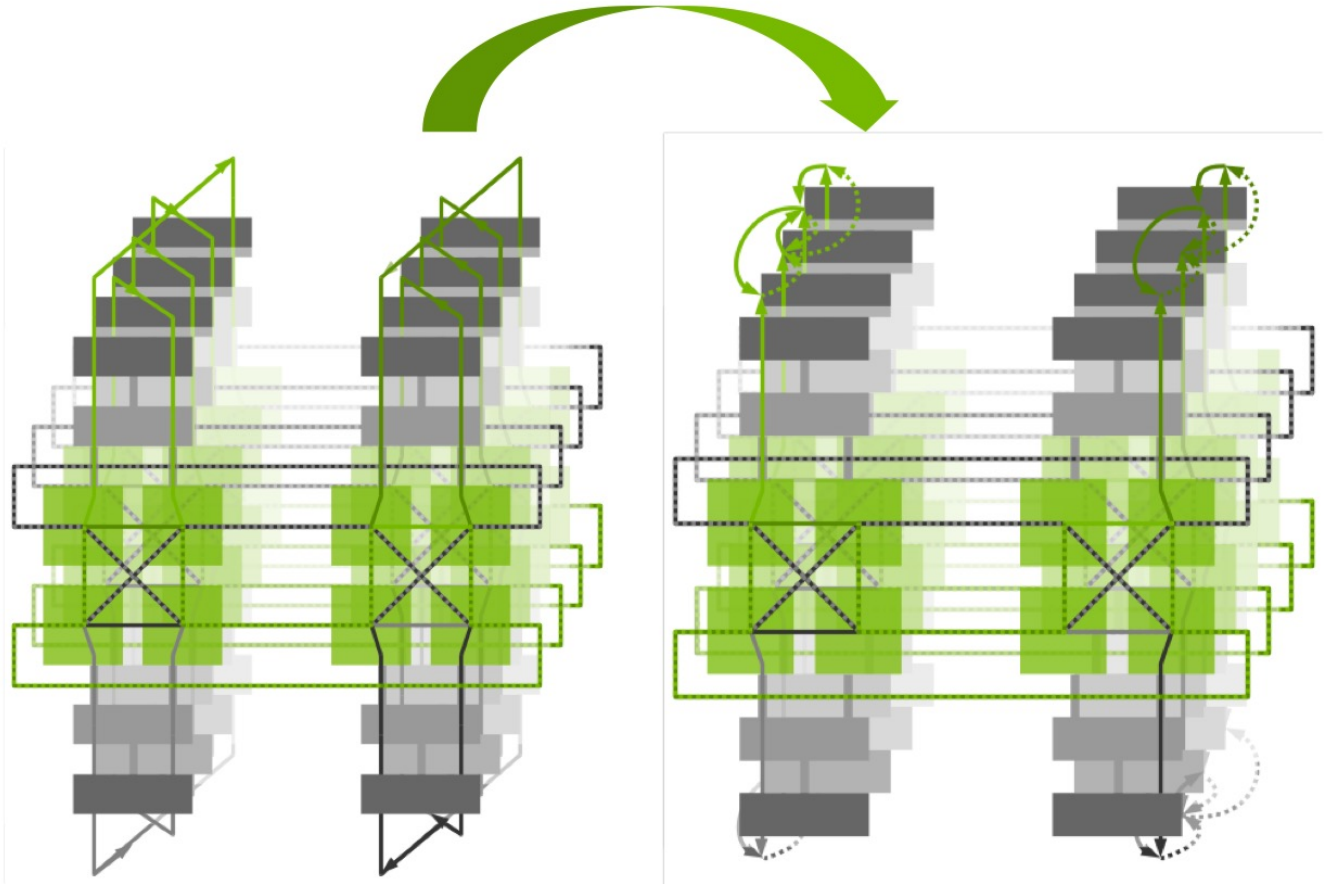
DGX-1



DGX-2

- 对每个节点的 Rings 进行相互连接
- 前提假设：NIC（网口）可以在节点间进行高效通信

节点间通信：Rings vs Trees

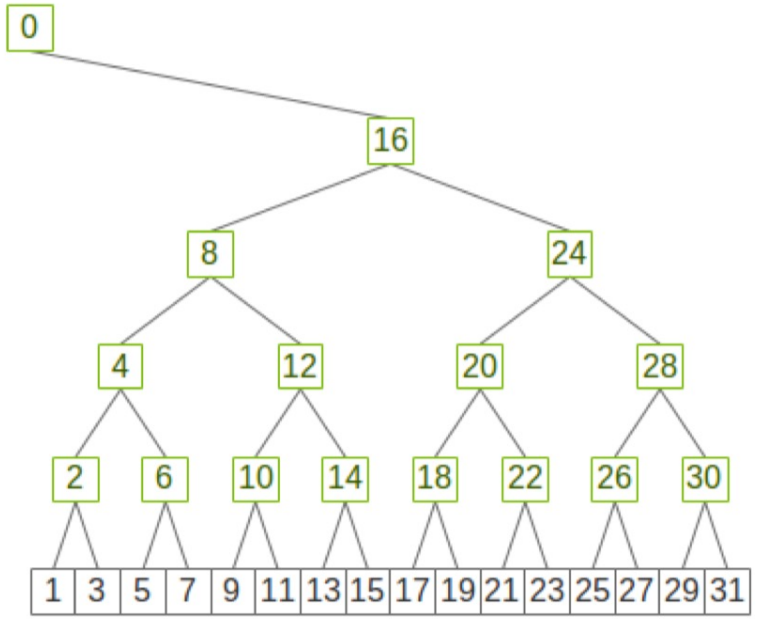


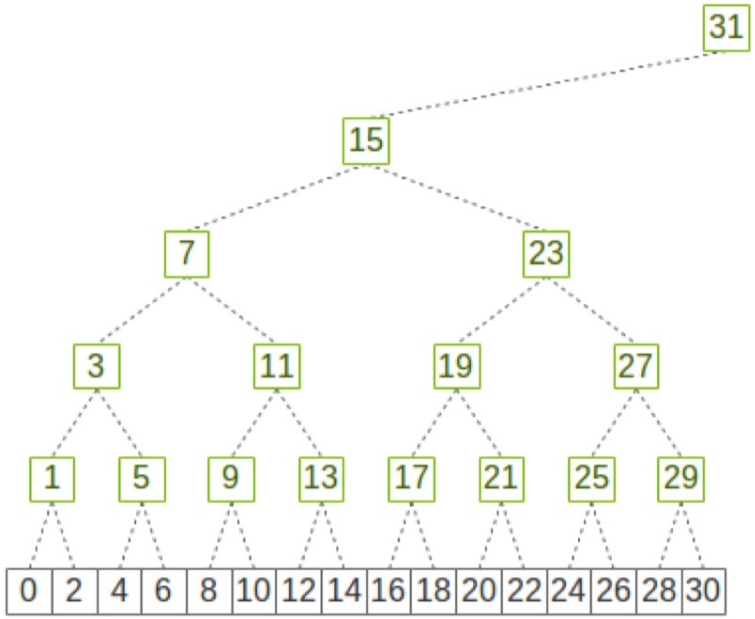
Latency = O(n)　　Latency = O(log2(n))

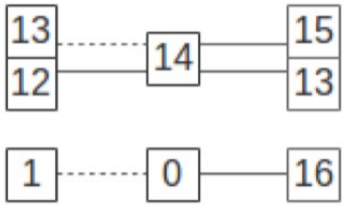
- 树通过节点之间进行连接
- 节点内通过多块 NIC 网卡聚合带宽

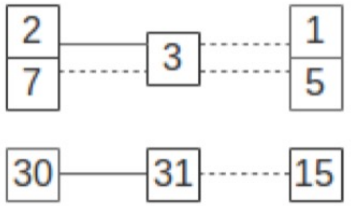
节点间通信：Dual binary tree

- 低时延



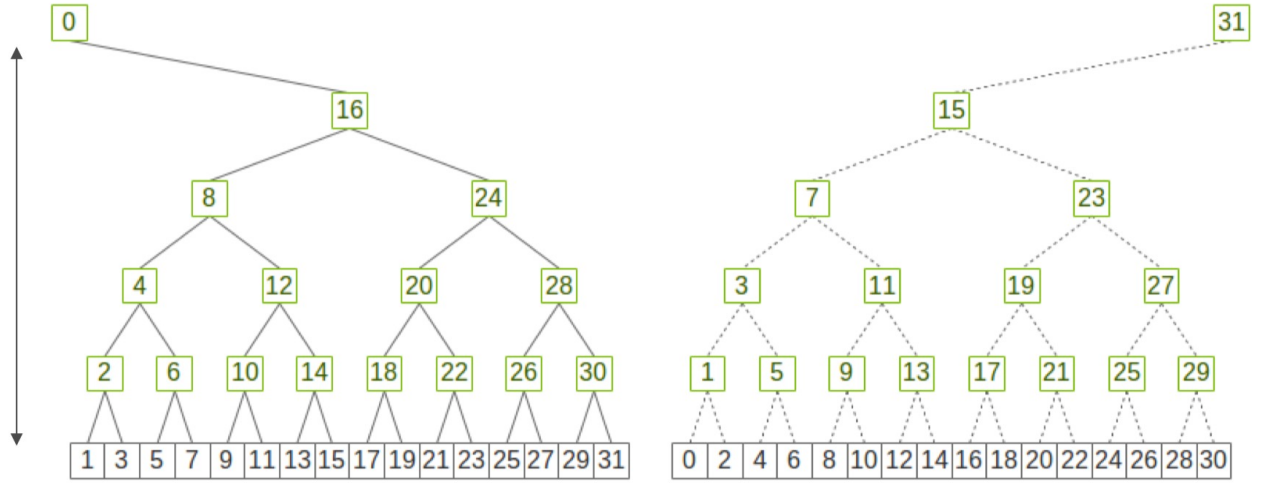


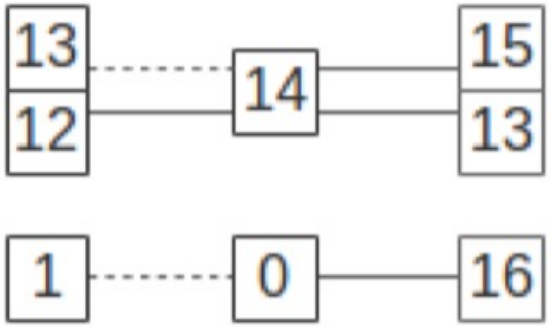




- 两棵树二叉树，两种模式
- 最大化局部通信

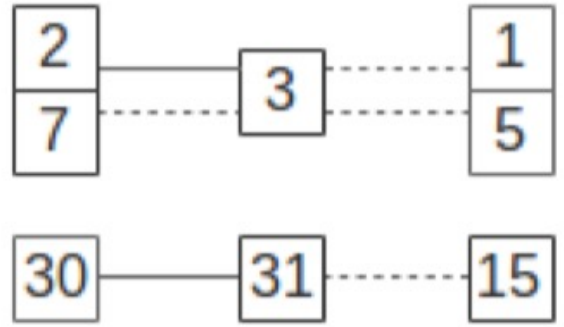
节点间通信：Dual binary tree





两棵互补二叉树，每一棵树处理一半的数据，充分利用网络带宽

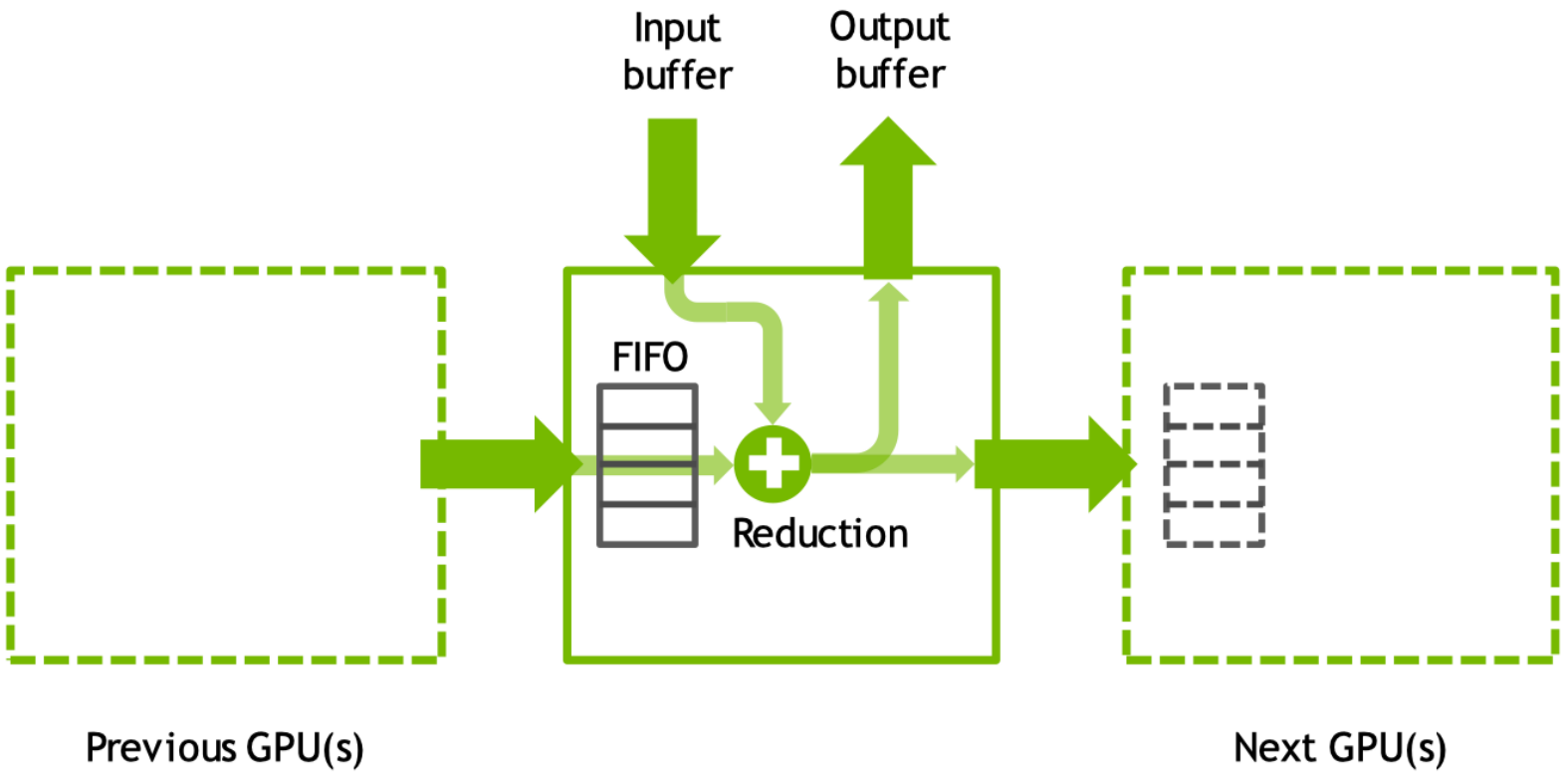
Half the data goes through each tre e, achieving full bandwidth



05. CUDA执行核心

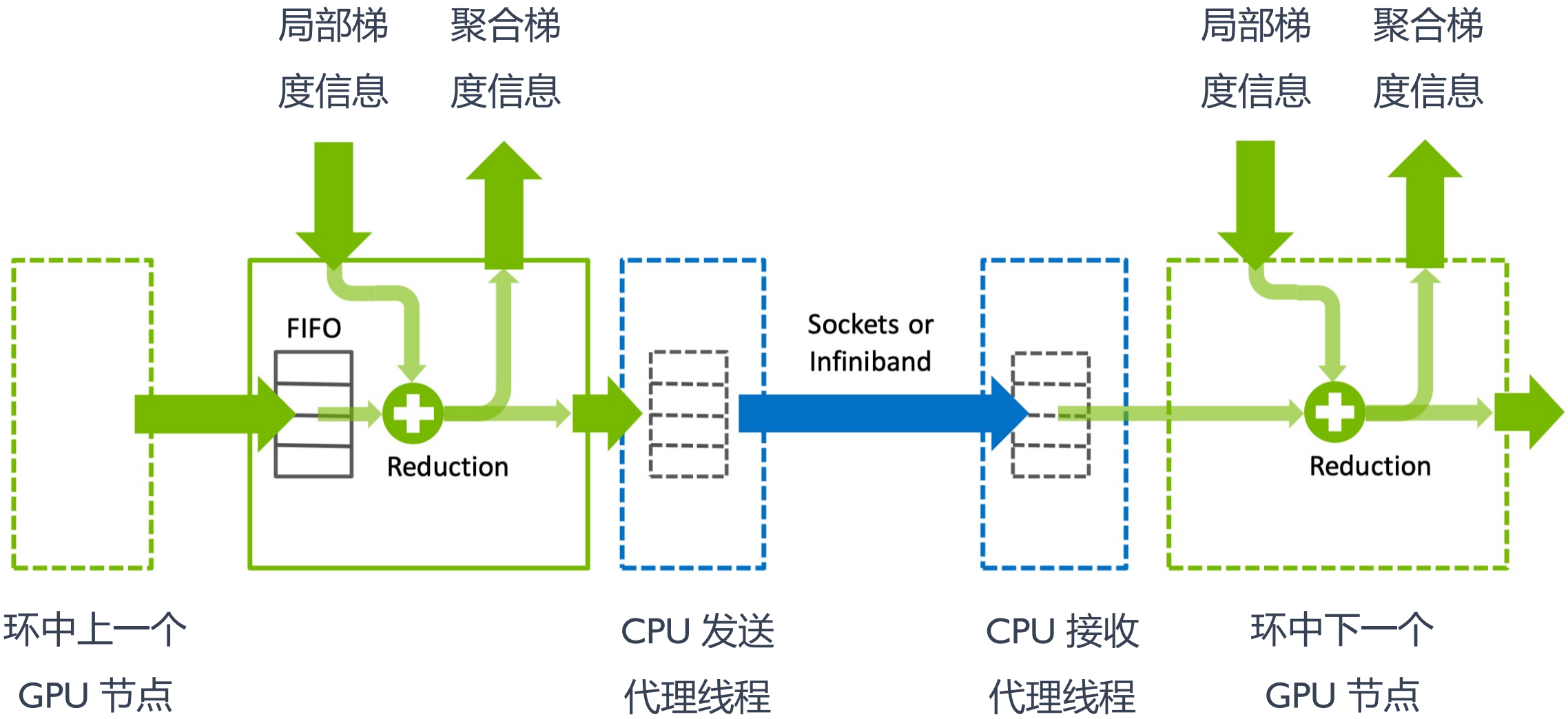
CUDA KERNELS

GPU 通信核心：Principle

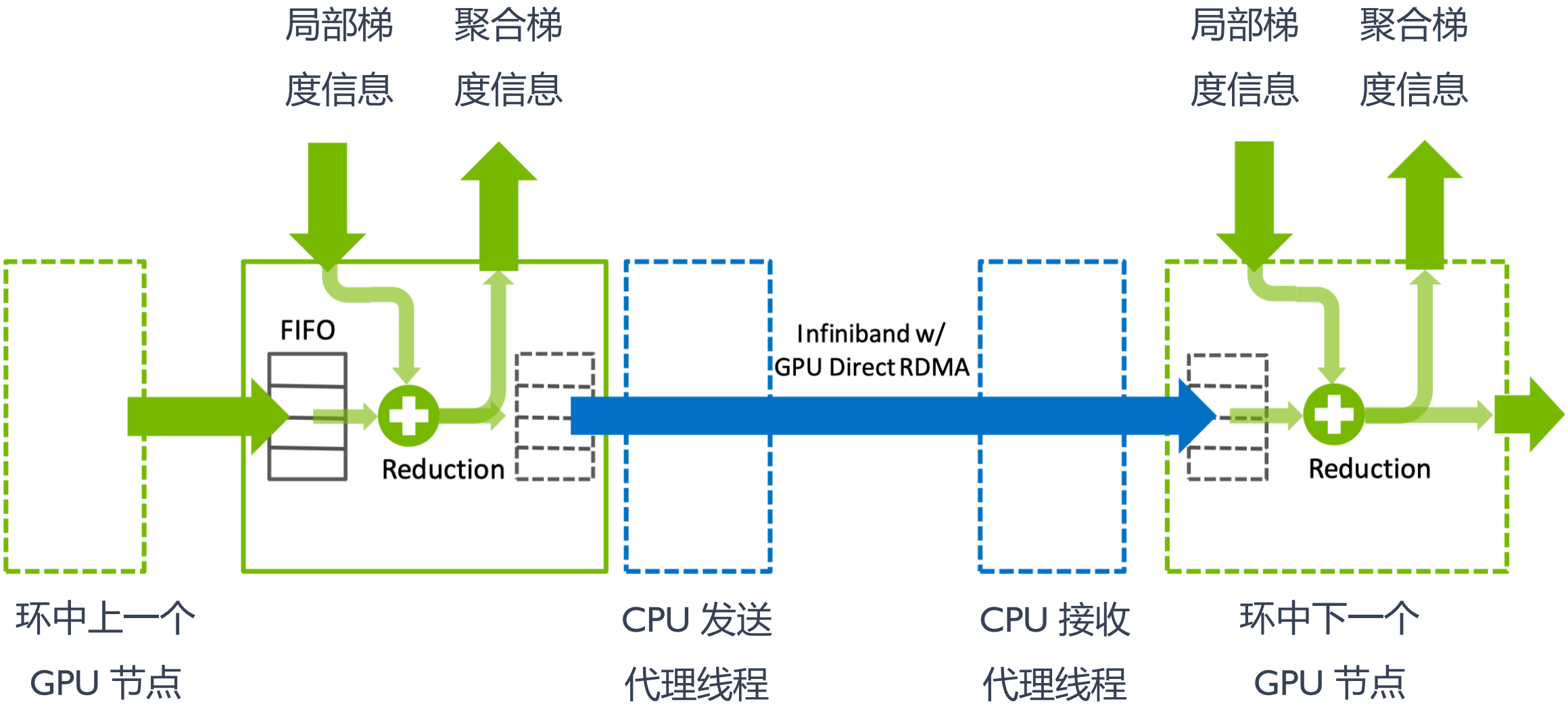


- GPU 核心内：
  - 通过 GPU 内 FIFO 队列从其他 GPU 中接收和发送数据
  - 使用本地和远程缓冲区执行 reductions 和 copy 操作

节点间通信：网络代理


局部梯度信息
聚合梯度信息
局部梯度信息
聚合梯度信息
FIFO
Reduction
Sockets or Infiniband
Reduction
环中上一个 GPU 节点
CPU 发送代理线程
CPU 接收代理线程
环中下一个 GPU 节点

节点间通信：GPU Direct RDMA



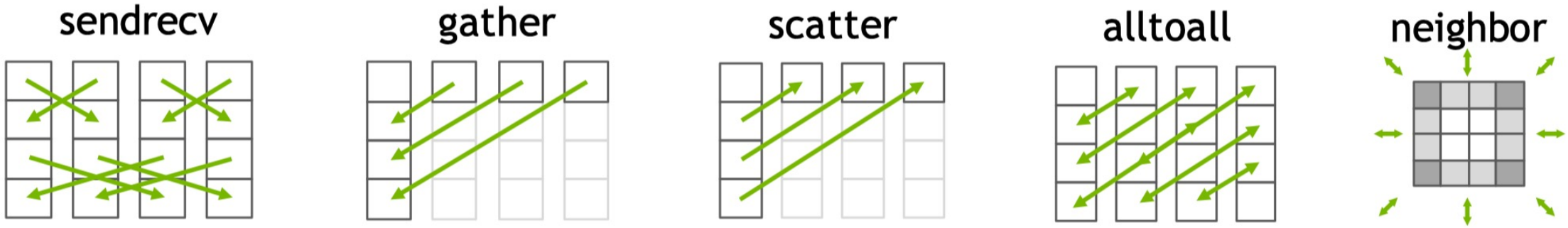
06. 数据发送和接收

SEND/RECEIVE

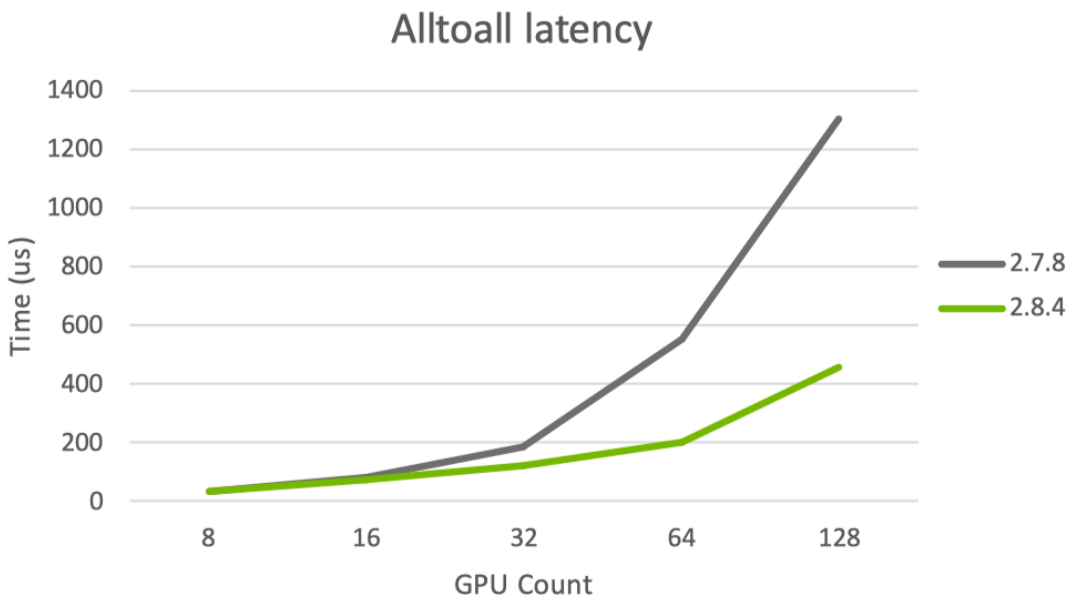
Communication semantics 通信语义

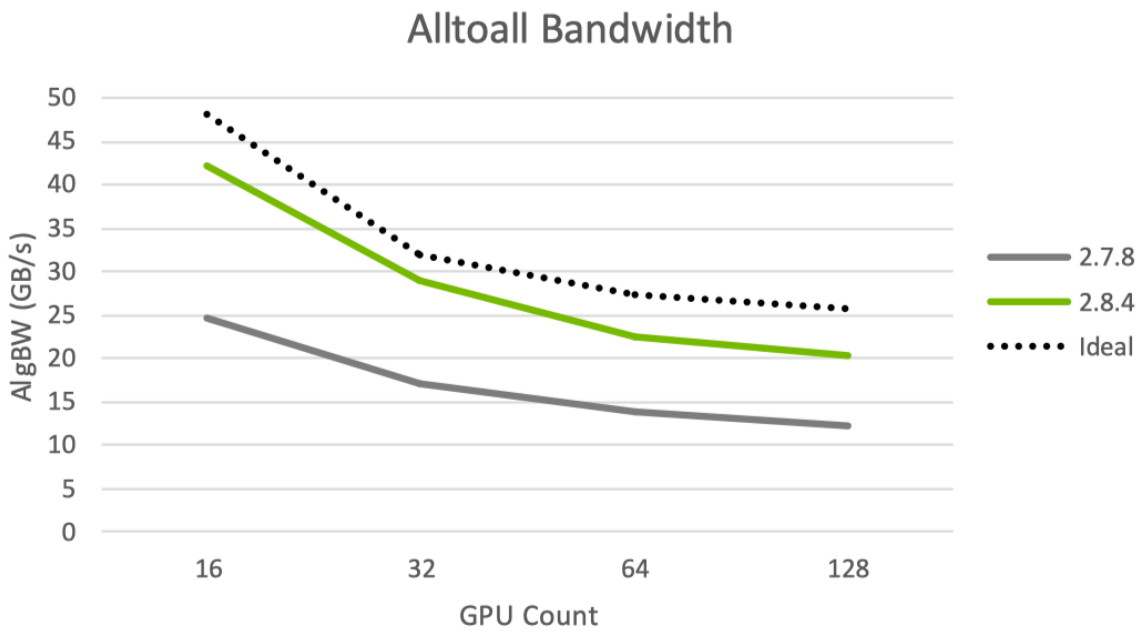
```
ncclGroupStart();
ncclSend(sbuf, ssize, sdtype, peer);
ncclRecv(rbuf, rsize, rdtype, peer);
ncclGroupEnd();
```

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



网络优化

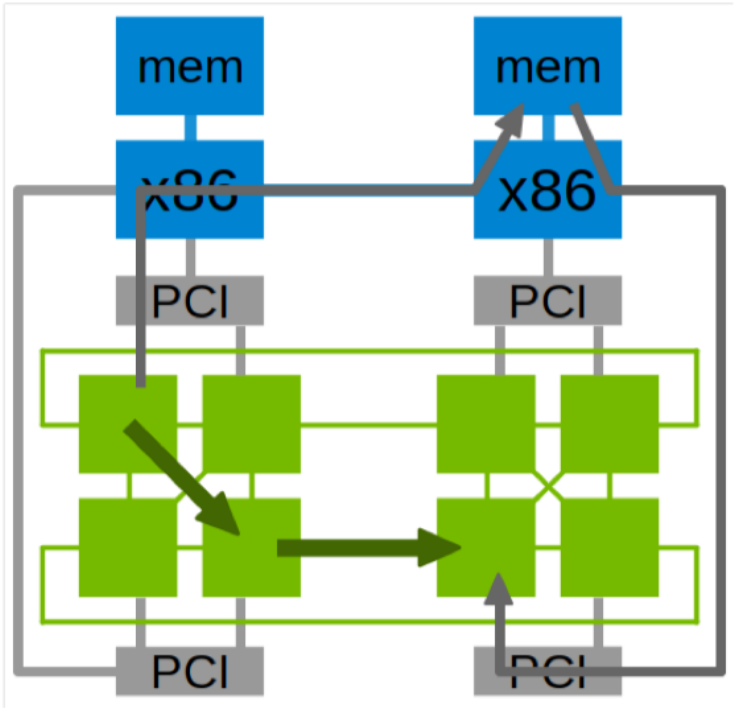


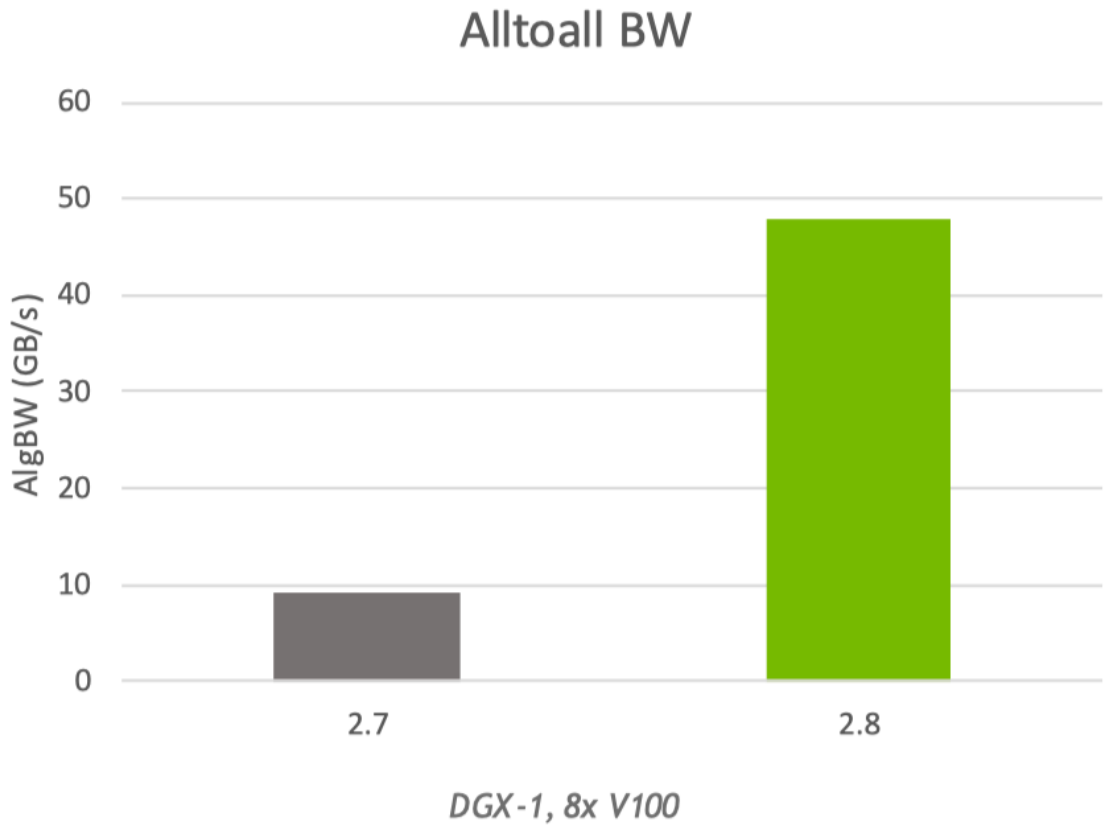


- 随节点数增加，网络时延也会增加
- 带宽理论值可以指引优化方向

网络优化

- 不同节点通信拓扑结构，很大程度影响通信带宽





07. 通信协议

Communication Protocol

NCCL 通信协议

- NCCL 使用 3 种不同的协议 LL、LL128 和 Simple：
  - 其分别具有不同延迟（~1us、~2us 和 ~6us）
  - 其分别具有不同带宽（50%、95% 和 100%）
  - 以及其他影响其性能的差异。

LL 协议

- **LL， Low Latency 协议：**
  - 优化小数据量传输：小数据量情况下，打不满传输带宽时，优化同步带来的延迟。
  - 提供低延迟：8 bit 原子存储操作，提供低延迟通信。

LL128 协议

- **LL128，Low Latency 128 协议依赖硬件 NVLink 实现：**
  - 默认协议：LL128 能够以较低延迟达到较大带宽率，NCCL 会在带有 NVLink 硬件上默认使用该协议。
  - 低延迟：128 字节原子存储。

Thank you

把AI系统带入每个开发者、每个家庭、每个组织，构建万物互联的智能世界

Bring AI System to every person, home and organization for a fully connected, intelligent world.

ZOMI

Course chenzomi12.github.io

GitHub github.com/chenzomi12/AIFoundation