大模型系列 - 集合通信

ZOMI

集合通信算法

大模型业务全流程

数据 & 模型算法

2. 数据处理

- 开源数据
- 数据预处理
- 向量数据库

3. 模型算法

- LLM模型架构
- 多模态一切皆Tokens

模型训练 & 微调

4. 模型训练

- 混合精度
- 梯度检查
- 梯度累积
- …

5. 分布式并行

训练集群稳定性

6. 模型微调

- 全参微调
- 低参微调
- 指令微调

模型验证 & 推理部署

7. 模型验证

- 下游任务
- 测评标准
- bench mark

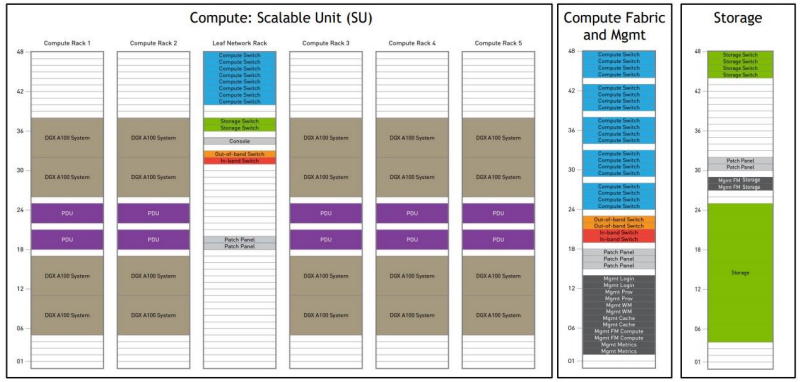
8. 推理与智能体

- 量化压缩
- 推理加速
- 9. Agent 智能体

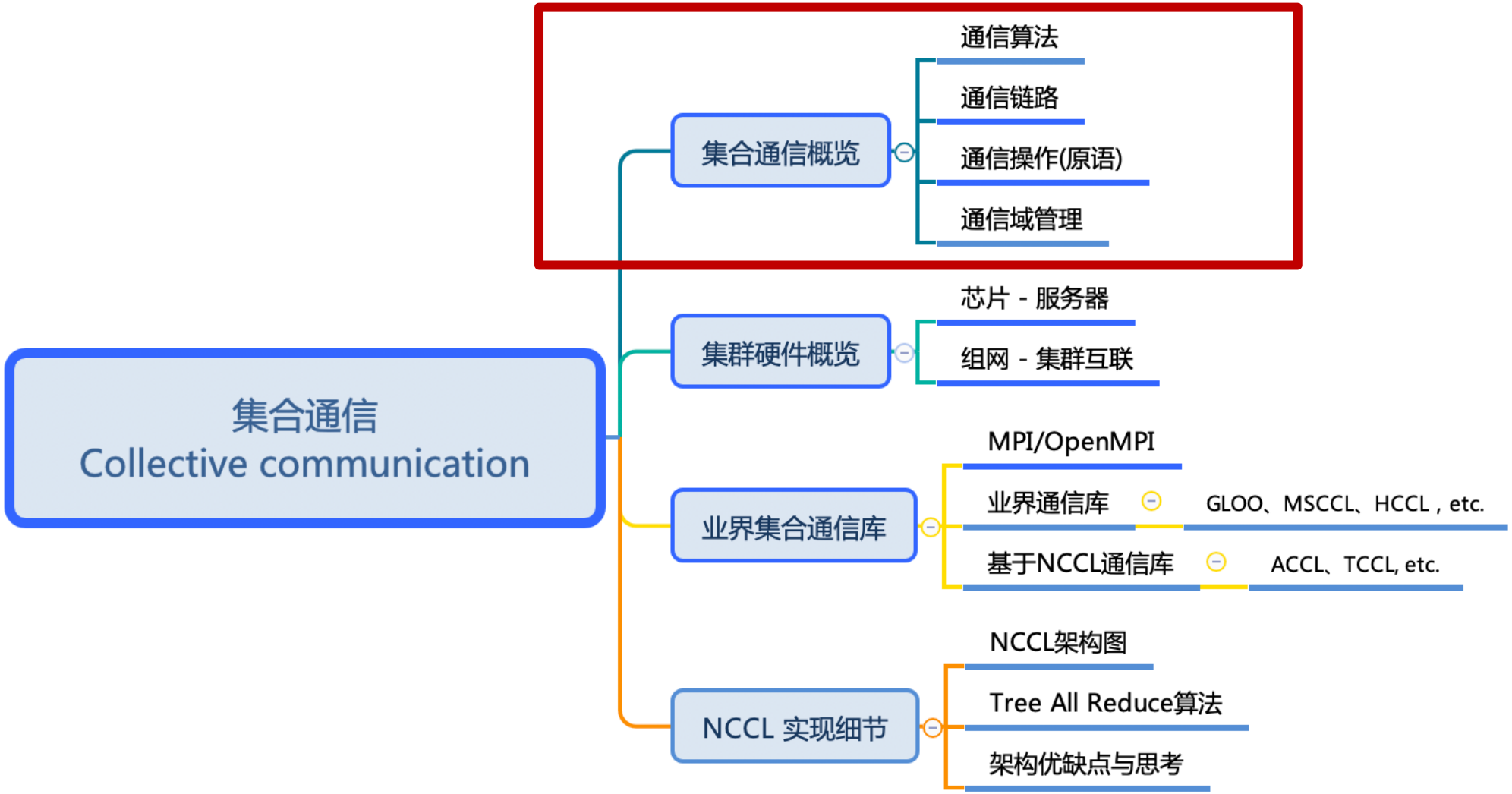
1. AI 集群建设

- 计算、存储、网络
- AI 集群机房建设
- AI 集群上线与运维

集群算力准备



思维导图 XMind


集合通信
Collective communication
集合通信概览
通信算法
通信链路
通信操作(原语)
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 实现细节
NCCL架构图
Tree All Reduce算法
架构优缺点与思考

集合通信概览

XCCL: XXXX Collective Communication Library

1. AI 与通信关系（AI 基础知识、训练推理、分布式并行）
2. XCCL 基本架构（HPC 通信架构 to XCCL 通信架构）
3. 集合通信原语（All Reduce, etc.）
4. 集合通信算法（Ring / Tree / Torus All Reduce etc.）
5. PyTorch 集合通信与计算并行

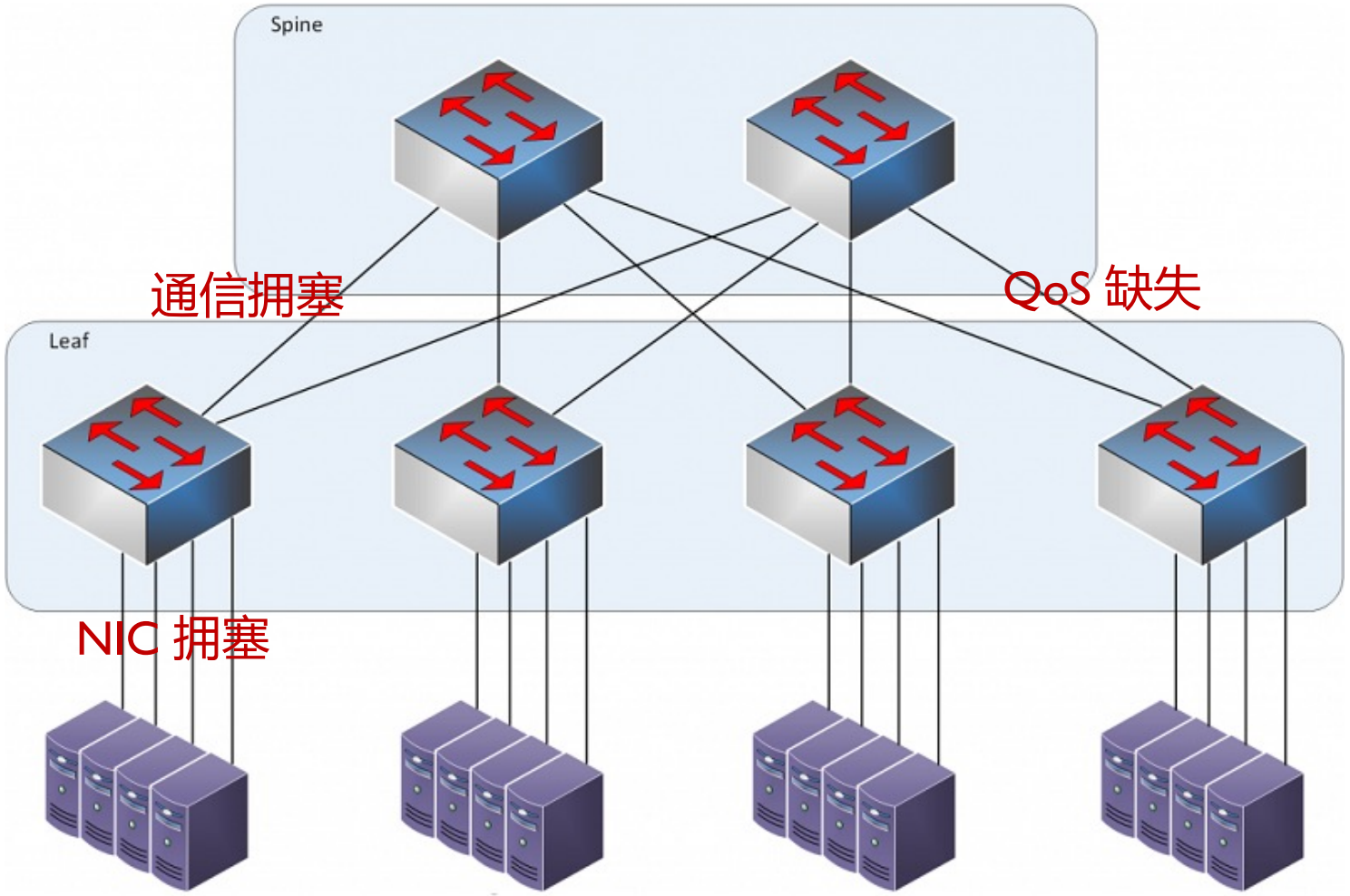
为什么需要集合通信算法?

- **通信算法：**XCCL 集合通信库实现流量统一规划，满足复杂物理拓扑中流量有序交换，最大化集群通信性能；
- **通信原语/操作：**提供不同 NPU 硬件上进程间的通信方式或者通信 API，即建立在通信算法上层概念，通过不同的通信算法来实现；

通信特性	HCCL	NCCL
通信算法	ring/mesh + ring/Hav-Doub/Pair-Wise，etc.	ring + Tree ring，etc.
通信操作	allreduce、broadcast、reduce、reduce scatter、allgather、all2all、send、recv	allreduce、broadcast、reduce、reduce scatter、allgather、all2all、send、recv

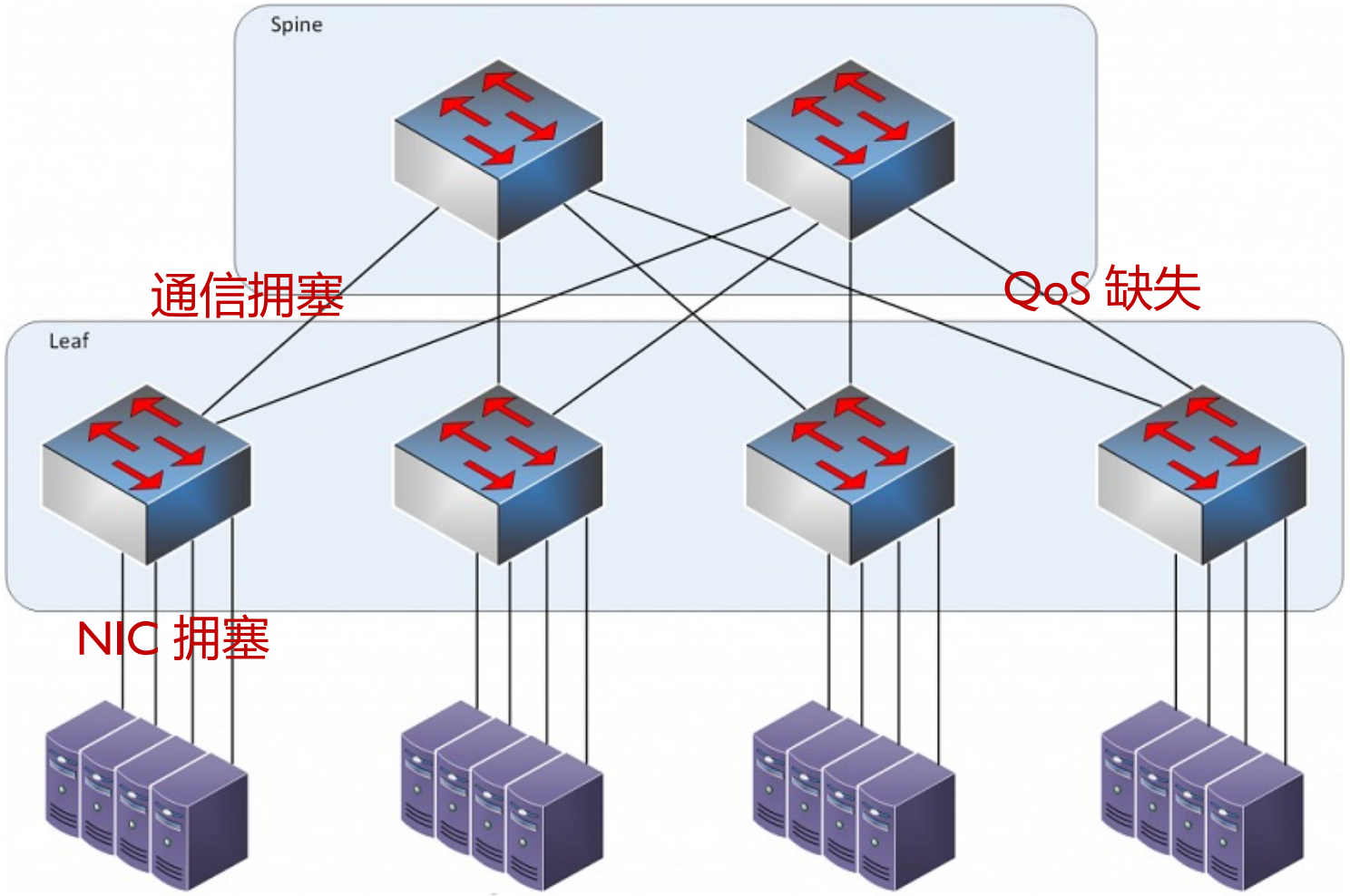
01. AI 对通信算法的诉求

传统 AI 服务器集群



- 传统服务器配备一张网卡用于节点间通信，为支持 AI 配置多个 NPU。
- AI 训练需要 NPU 间梯度同步，多 NPU 并发访问网络，网卡成为系统瓶颈。
- PCIe 链路带宽分配与路径长度密切相关，长路径获得带宽分配较低，跨 Socket 通信问题就变得严重。

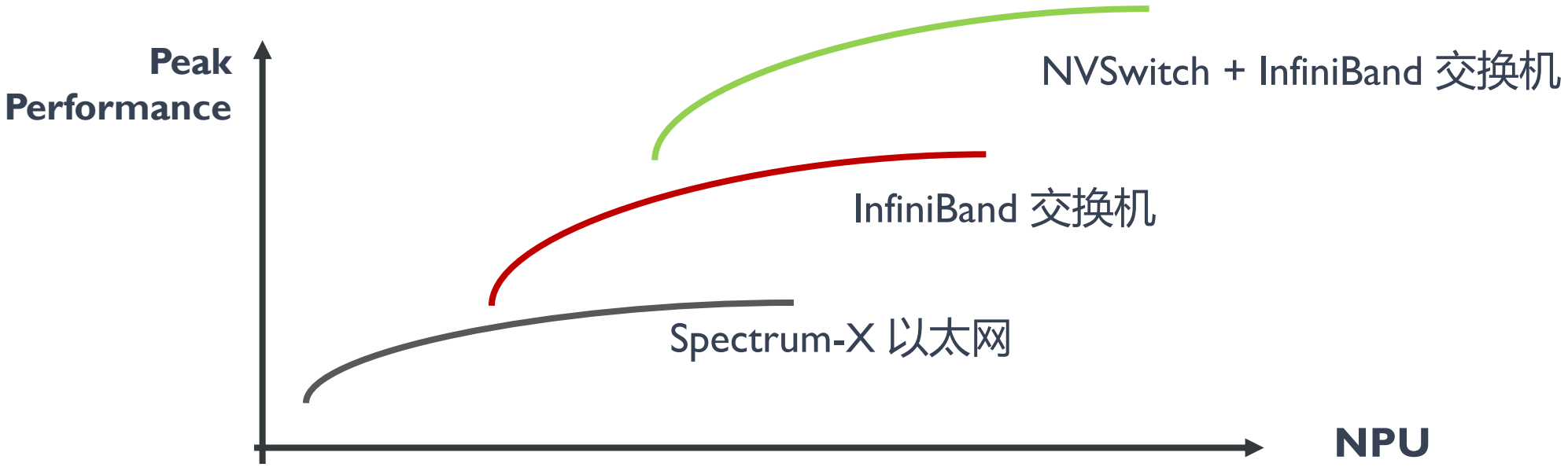
传统 AI 服务器集群



- 网络架构主要解决 AI 训练中同步通信导致短板效应。
- 拥塞控制算法：对两个碰撞流进行限速，使其尽快达到均分物理带宽的目的，不能解决 AI训练集群通信效率。
- AI 业务通信的同步性，每个通信最终性能决定于最慢的连接。均分带宽意味着事务完成时间的成倍提升，严重影响AI通信性能。

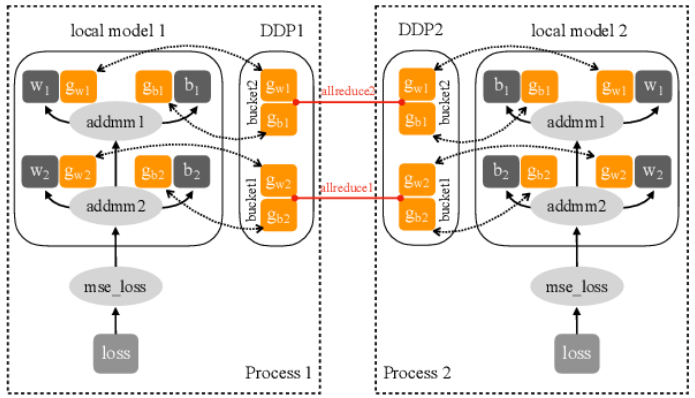
不同 AI 时期对通信的诉求

- **AI 集群：**
  - 单一化业务，整个 AI 系统只为大模型（LLM、LMM等）或者搜广推服务，几乎没有其他业务的复用性；
  - 用于超大规模的模型（百/千亿参数量）的训练、推理，L0 基础大模型算法研究的探索；
  - 训练大模型走极致性能优化路线 vs 虚拟化云服务和 AI 通用算力服务化走性价比路线；



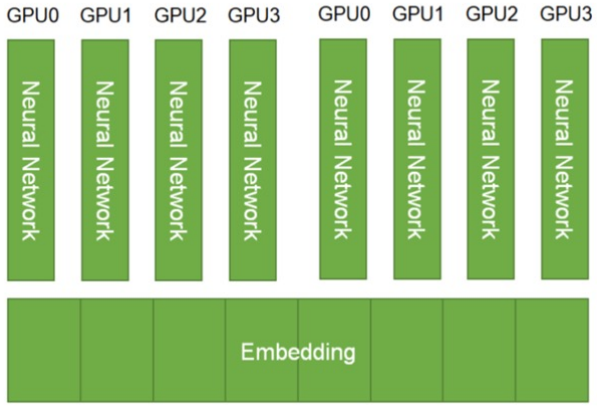
不同 AI 时期对通信的诉求

CV 模型--进入成熟期



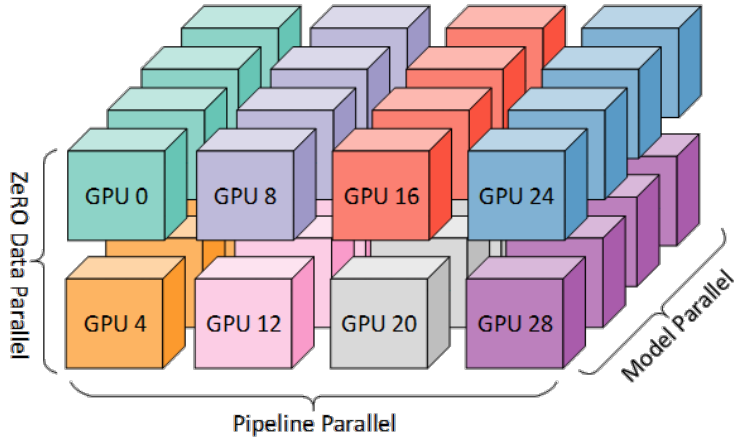
数据并行，模型参数少，All r
educe 为主，参数面规模<4K

推荐模型--大规模应用



Embedding 并行 All2All，参数
面规模<4K

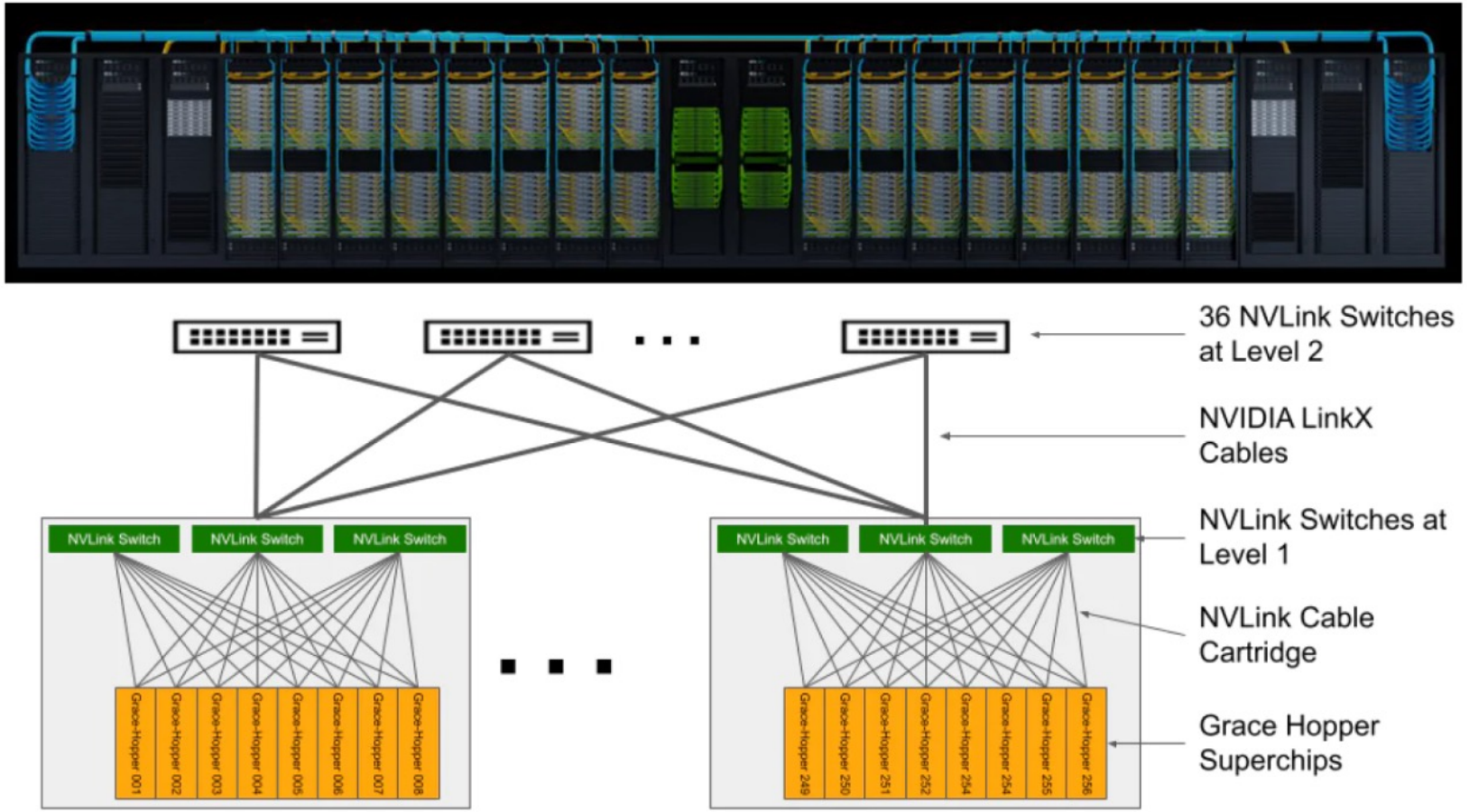
大模型—快速发展期



PTDES混合并行，集合通信都
有使用，参数规模<32K

组网形成不同的网络拓扑

- 基于不同的 AI 场景，组网需求不同，其中大模型对组网要求极高；



- 超节点 SuperPOD
  - 大规模组网
  - 通信拓扑结合
    - 高速互联

思考

- 大模型对 AI 集群训练的诉求?

1. 大带宽（大流量、高频次的通信）
2. 强同步（单卡故障拖慢全网、一卡慢整网慢）

大模型流量相对明确

- 大模型处于快速发展期，当前基于 Transformer 的模型结构固定，模型通信流量相对明确
- 面对超长序列、MOE 结构、低精度数据格式 FP8 等在通信流量仍然存在挑战和不确定性

并行模式	通信产生原理	通信操作
数据并行（纯数据并行）	反向梯度更新时需要进行 all reduce 将梯度聚合	All reduce
ZeRO1（优化器状态并行）	直能更新部分权重，需要多执行一次 all gather 将权重聚合	All reduce All Gather
ZeRO2（梯度并行）	每张卡只需要 reduce 部分梯度，更新部分权重后一样需要 all gather	Reduce Scatter All Gather
ZeRO3（权重并行）	每张卡只需要 reduce 部分梯度并更新部分权重，但是前向和反向时需要 all gather 将权重聚合	Reduce Scatter All Gather
Pipeline（层间并行）	前向和反向时需要夸 rank 传递激活值	Send Recv

大模型流量相对明确

- 大模型处于快速发展期，当前基于 Transformer 的模型结构固定，模型通信流量相对明确
- 面对超长序列、MOE 结构、低精度数据格式 FP8 等在通信流量仍然存在挑战和不确定性

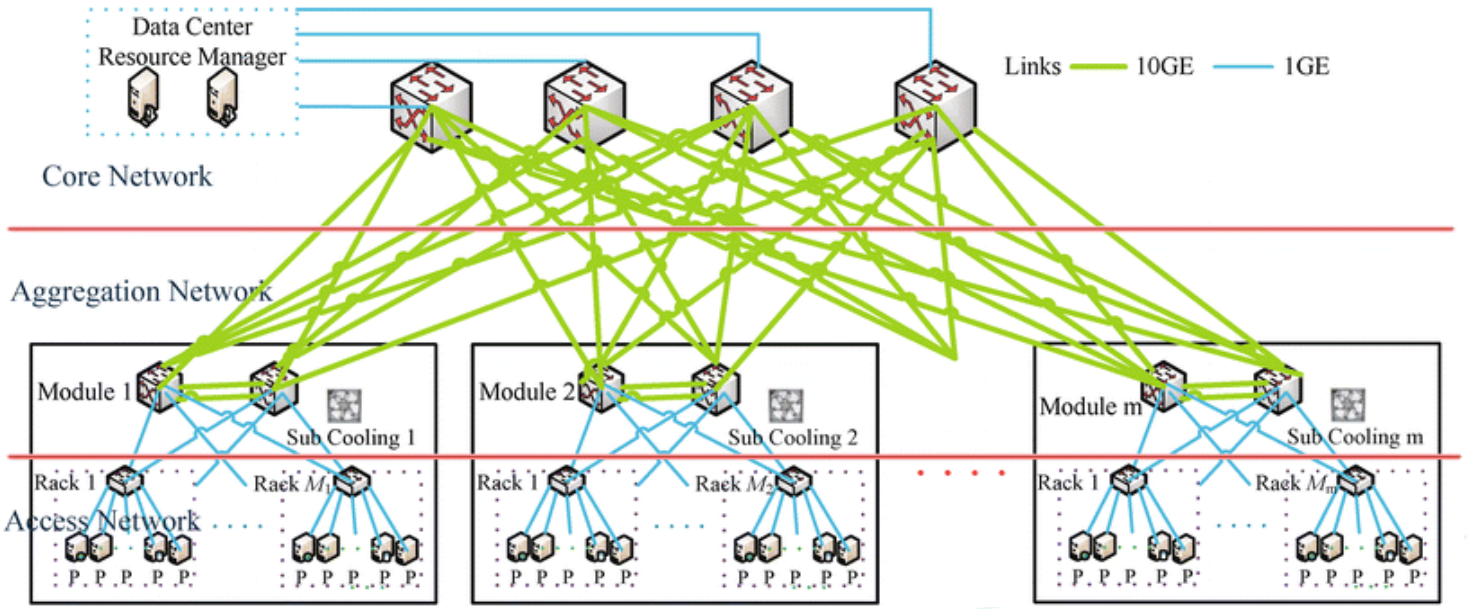
并行方式	通信操作	总通信量	单次通信量	算法	Rank 间关系
数据并行 张量并行	All Reduce	GB	25MB，PyTorch 可配置参数同步缓存 Bucket 大小	Ring	同序号卡通信
				HD	同序号卡通信
				Tree	同序号卡通信
MoE 并行	All2All	GB	按 Token 发送，KB 级别	Pair-wise	夸序号卡通信
				分层通信	同序号卡通信
流水并行	Send/Recv	MB	按 BSH 发送，MB 级别	P2P	夸序号卡通信

集合通信难点

- 集合通信难点：
  - 需要在固定网络互联结构（网络拓扑 Topology）约束下进行高效通信；
  - 集合通信算法与物理网络互联结构强相关，需要充分发挥网络通信效率；
  - 在效率与成本、带宽与时延间进行合理取舍。

网络结构和网络拓扑将会在网络一节重点展开

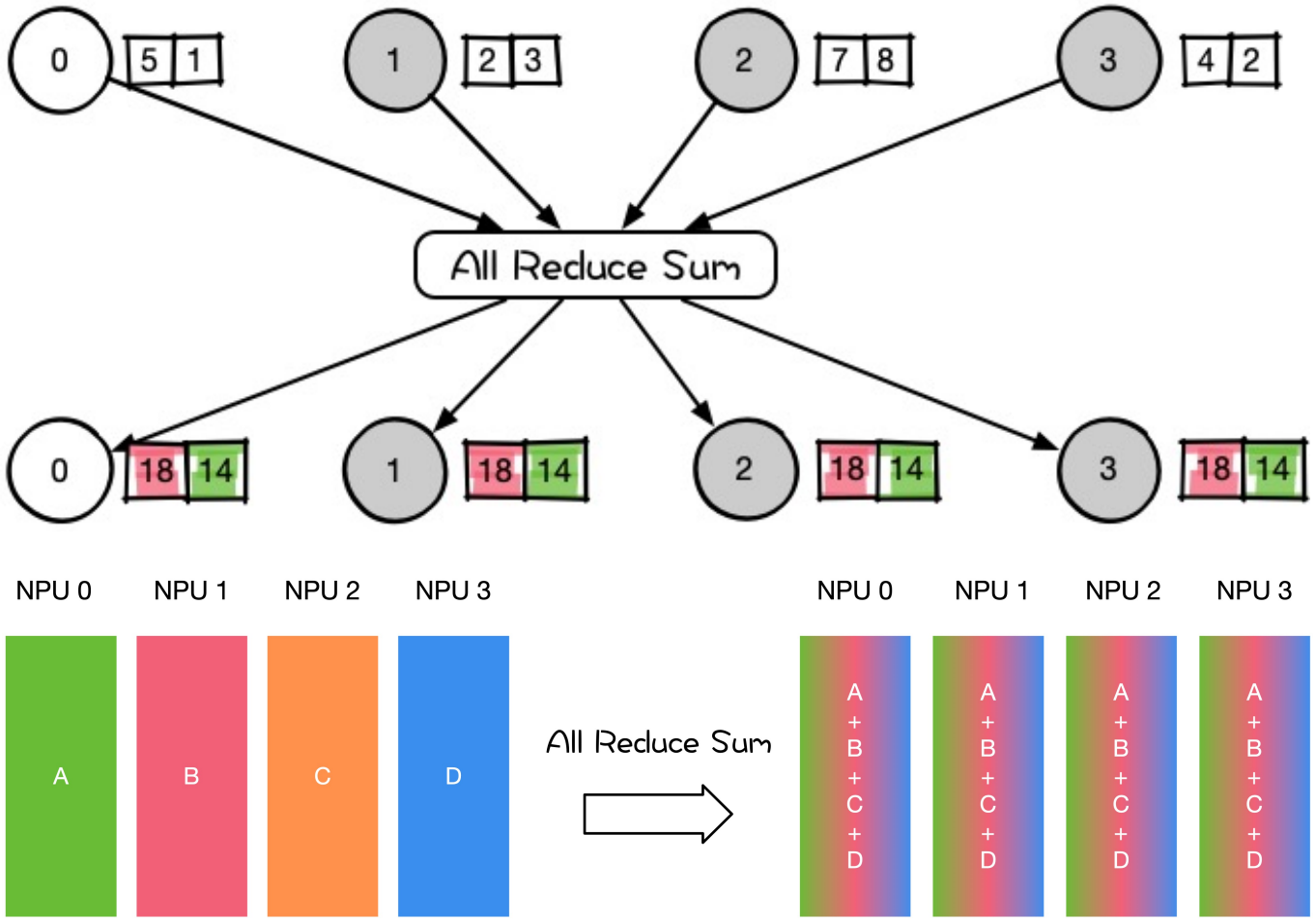
集合通信聚焦于拓扑之上的算法



02. All Reduce

All Reduce 操作是集合通信中最重要的操作，从所有 Rank 收集数据，以元素级方式聚合数据，并将聚合结果广播给其他 Rank

All Reduce



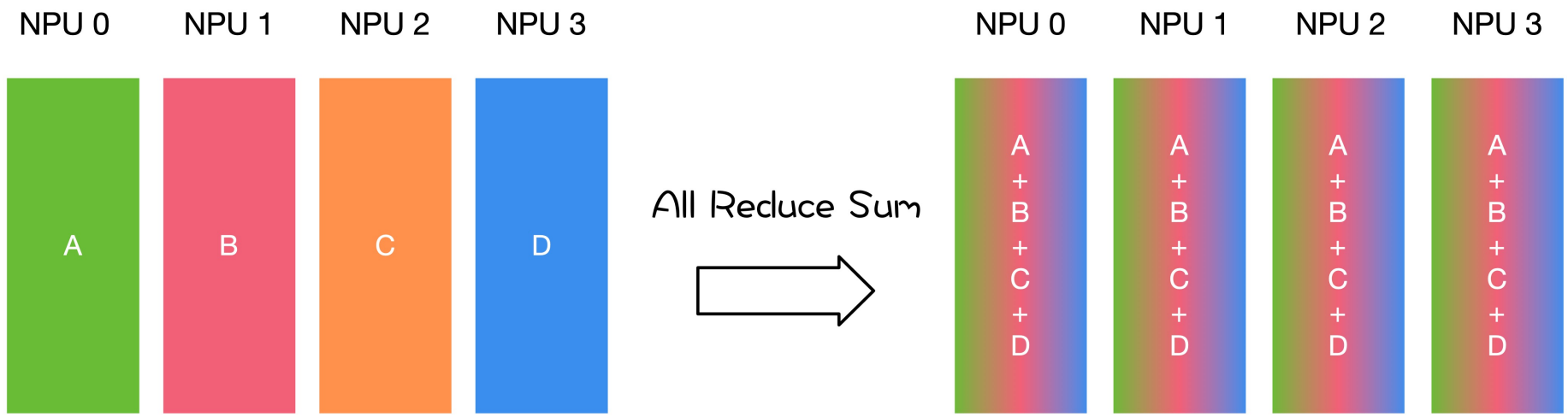
- 在所有 Rank 执行相同 Reduce 操作，将所有 Rank 数据规约运算得到的结果发送到所有 Rank

  - 在专家并行、张量并行、序列并行中大量地使用 All Gather 对权重和梯度参数进行聚合。
  - 数据并行 DP 各种通信拓扑结构比如Ring AllReduce、Tree AllReduce里的 AllReduce 操作；

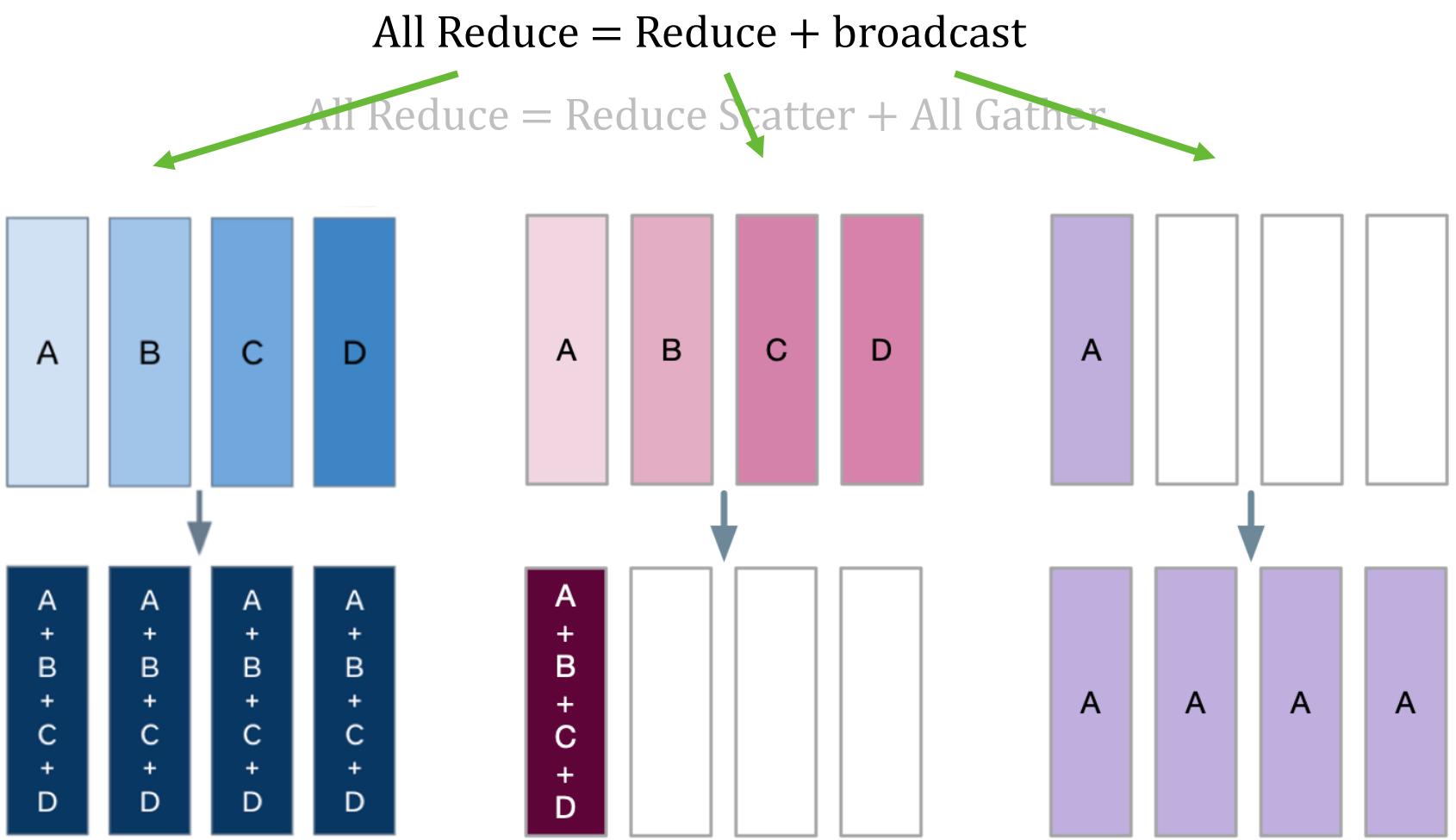
All Reduce

$$\text{All Reduce} = \text{Reduce} + \text{broadcast}$$

$$\text{All Reduce} = \text{Reduce Scatter} + \text{All Gather}$$

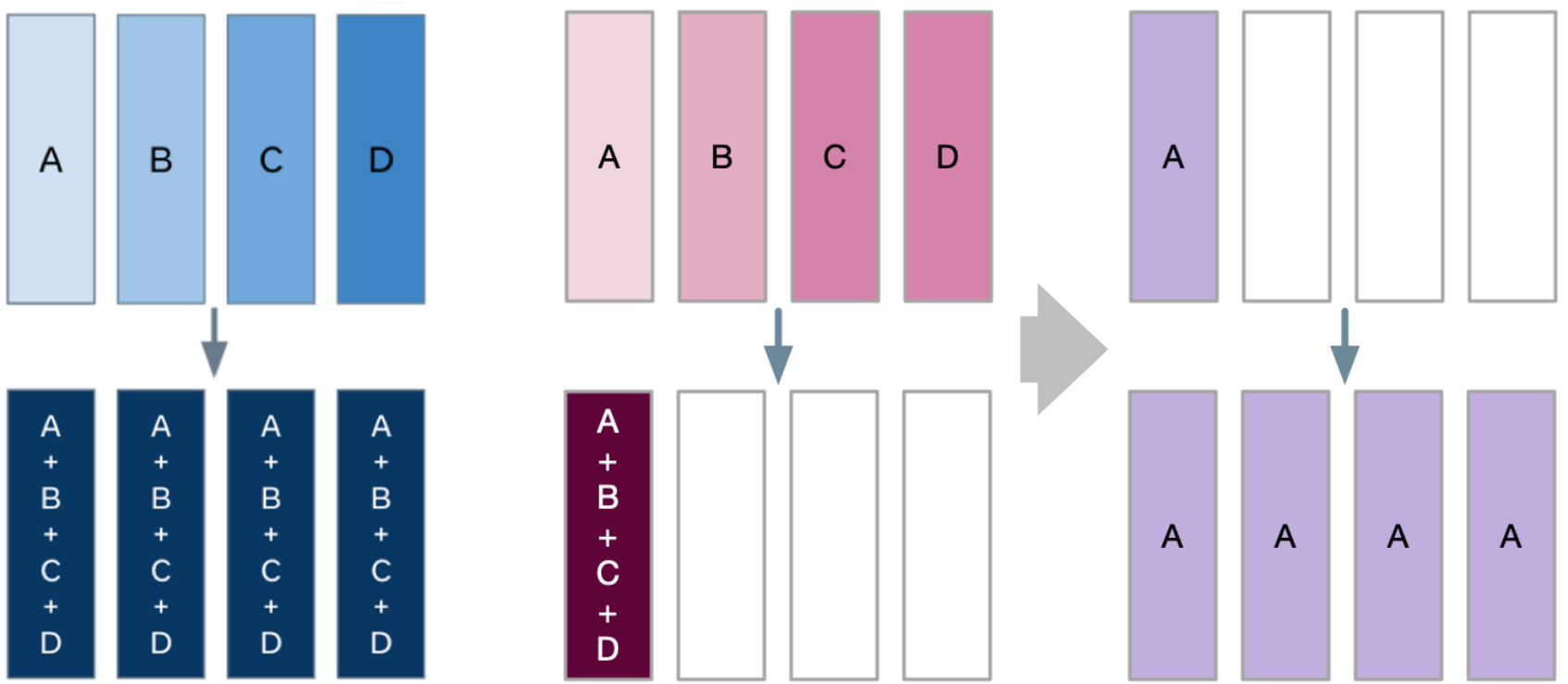


All Reduce = Reduce +broadcast



All Reduce = Reduce +broadcast

- 在 Reduce + broadcast 里， Reduce 先将 N 张 NPU 梯度 reduce sum 到 master Rank NPU0 上，再通过 broadcast 将 NPU0 中平均梯度复制到其他 NPU：



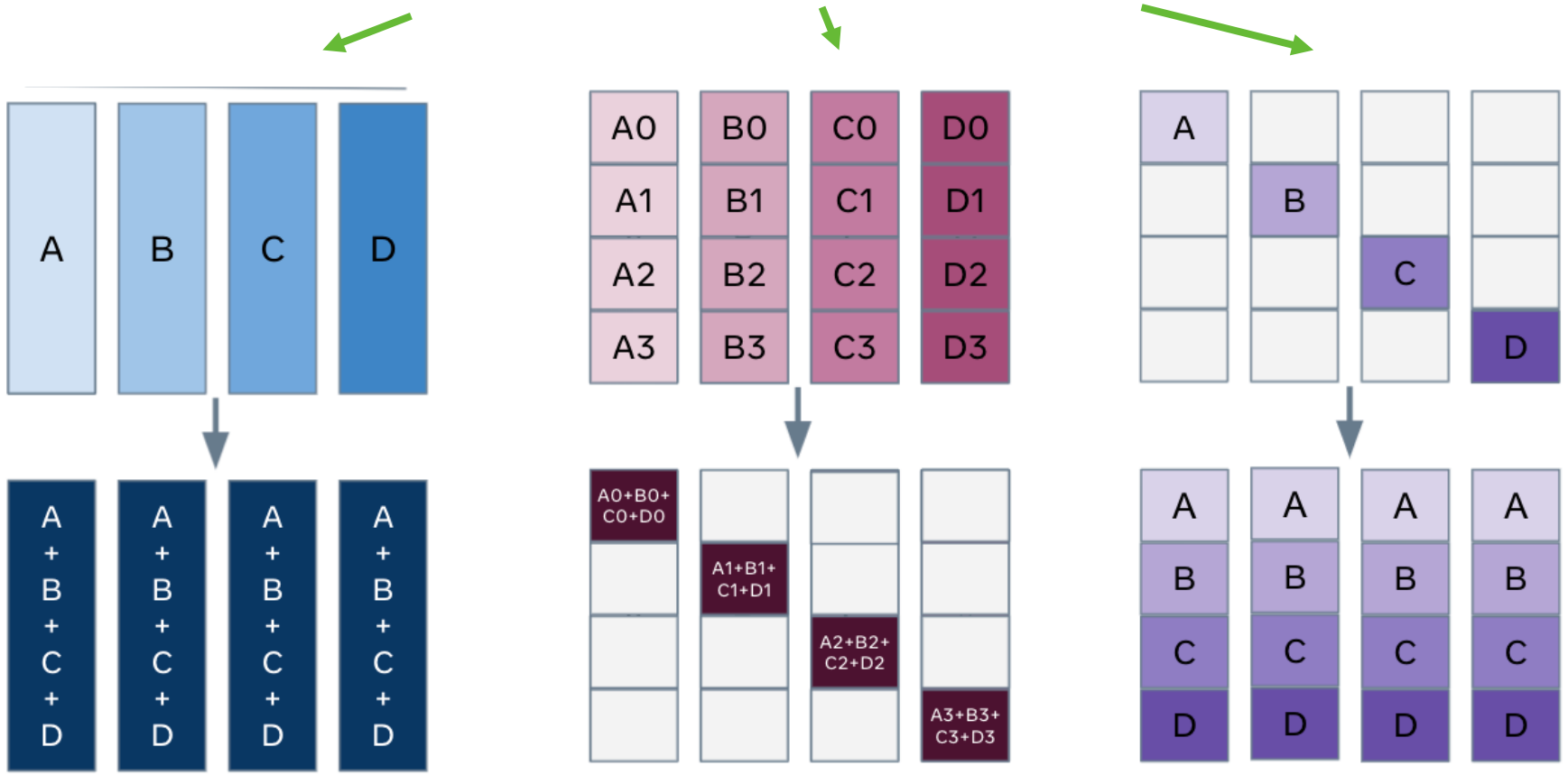
All Reduce = Reduce +broadcast

1. **通信数据：**N 个 Rank 数据 reduce sum 到一个 Rank。e.g. 假设为一个节点 8 个 Rank，每个 Rank 梯度 100MB，8 个 Rank 800MB，导致XPU0 在频繁发收数据，其余 Rank 空闲等待，集群效率低；
2. **通信带宽：**NPU0 网络带宽会成为瓶颈，所有 Rank 数据只能通过 NPU0 进行 reduce 和 broadcast，数据量较大场景 NPU0 带宽成为瓶颈。e.g. Tensor Parallelism；
3. **互联拓扑：**NPU 不一定两两全互联，N 个 Rank 数据一次 Reduce 或 broadcast，受限网络互联难实现，最终需要采用 ring/tree 策略进行 reduce 或 broadcast，集群效率低。

Reduce Scatter + All Gather

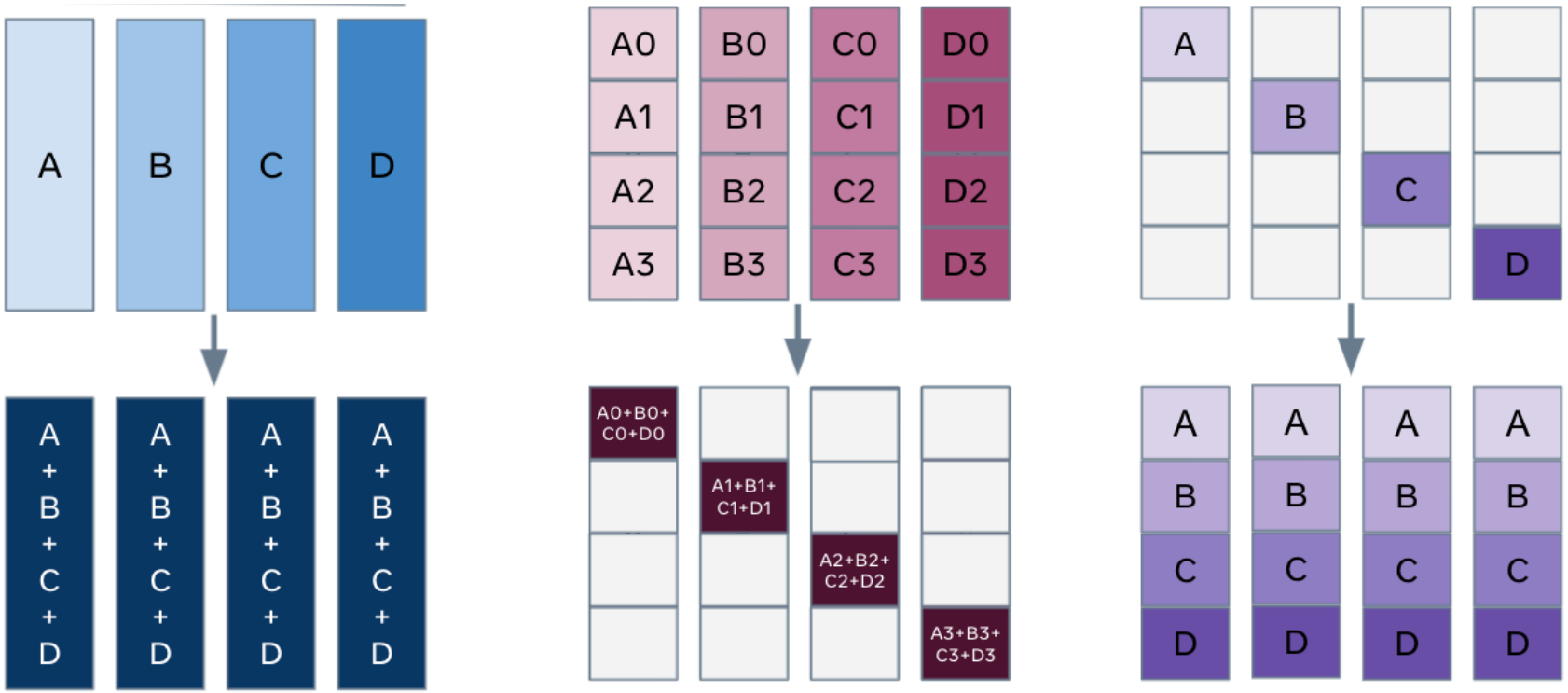
All Reduce = Reduce + broadcast

All Reduce = Reduce Scatter + All Gather



Reduce Scatter + All Gather

- 每个 NPU 只会从前向接受数据，并发送数据给后向，算法主要分为：
  - Reduce Scatter：先 scatter 拆分数据块再进行 reduce，每张 NPU 都会包括完整融合的同维梯度；
  - All Gather：进行全局 Gather 同步，最后所有 NPU 都会得到完整的梯度；



Reduce Scatter + All Gather

- 为什么大部分算法采用 Reduce Scatter + All Gather?

1. 充分考虑到 NPU 上梯度可能很大的情况，e.g. 一个梯度 400MB，reduce scatter 阶段会先被拆分成 NPU 个数份，假设节点内 NPU 数为 8，400MB 拆分 8 份（每份 50MB），从而减少 NPU 计算量以及节约带宽；
2. Reduce Scatter 通过将数据拆分成小块，同时并发进行 Reduce Scatter，从而将通信时间隐藏在计算时间内，进而提高 All Reduce 效率。

03. All Reduce

通信算法

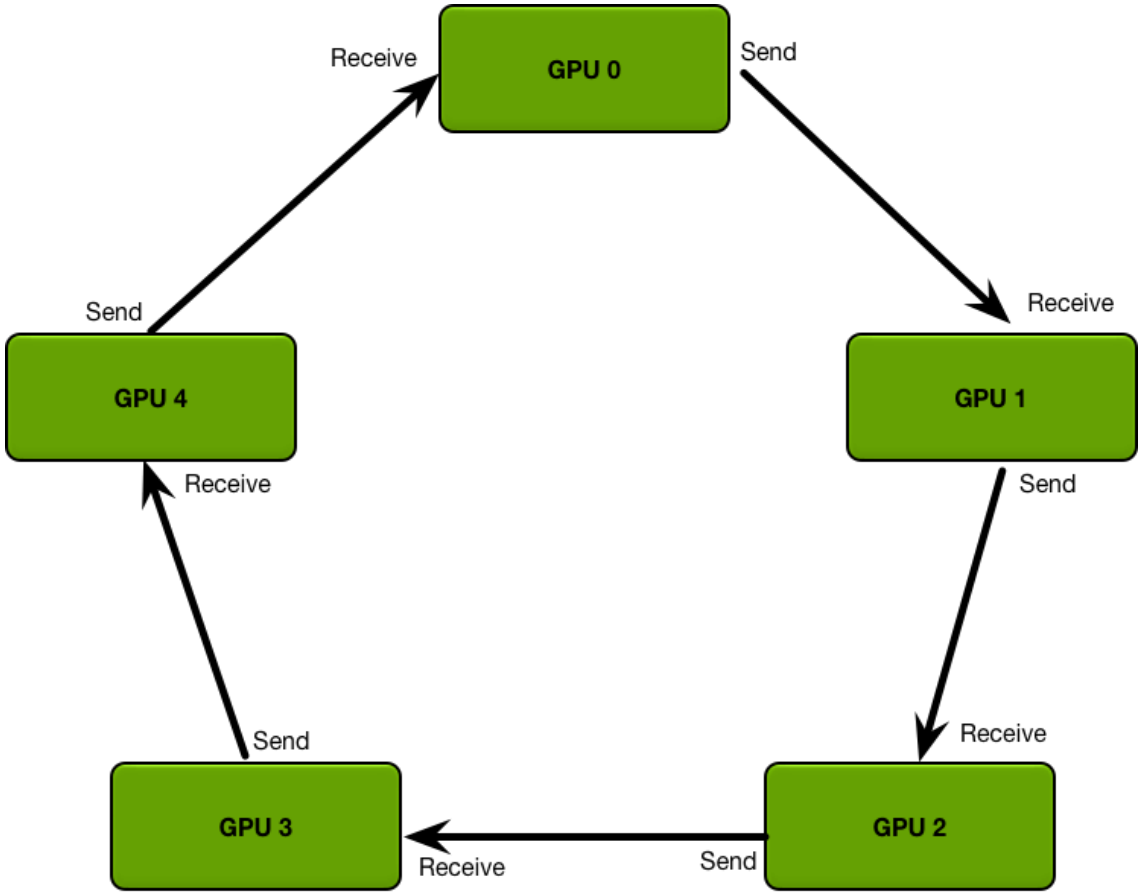
Ring 算法

- **基本介绍：**最常见实现算法基于 Ring All Reduce，NVIDIA NCCLv1.X通信库采用该算法，每次跟相邻的两个节点进行通信，每次通信数据总量的 1/N。

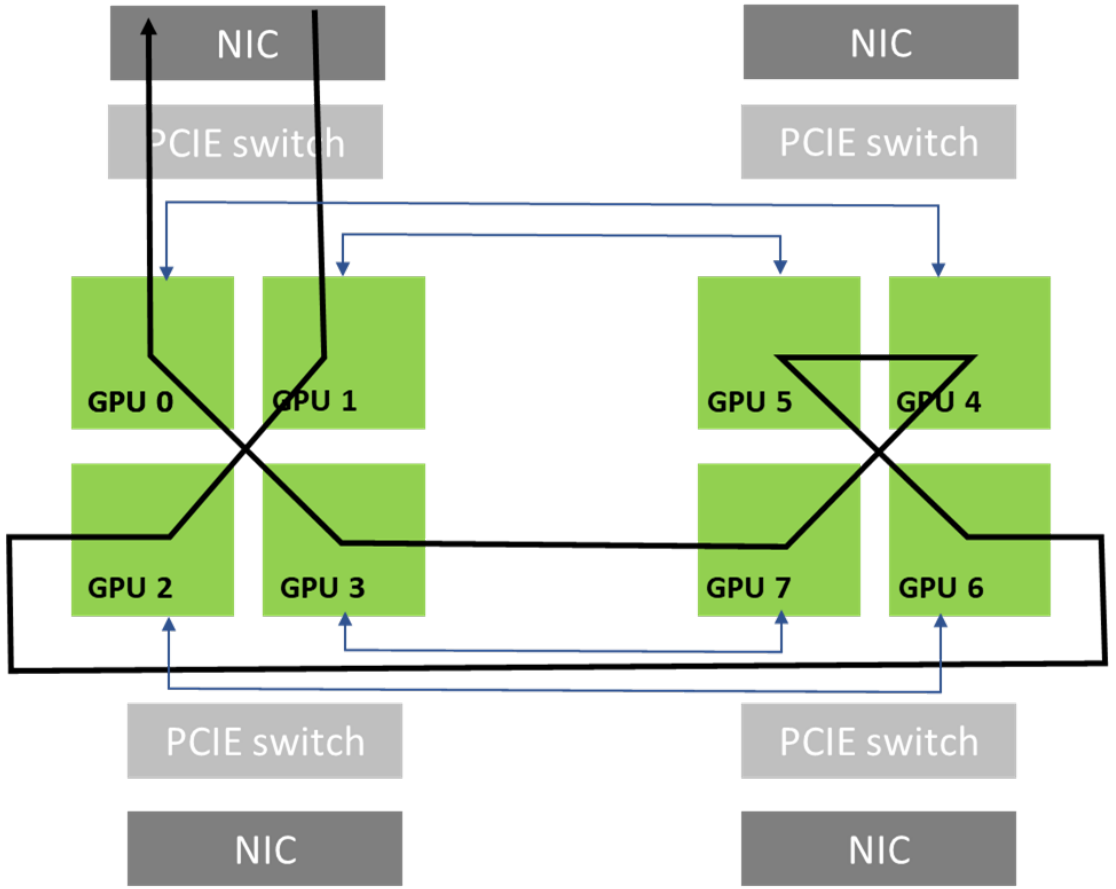
- **适用拓扑：**Star、Tree 等小规模集群；**通信步骤：**$2\times(N-1)$ Step；
- **优点：**实现简单，能充分利用每个节点的上行和下行带宽；
- **缺点：**通信延迟随着节点数线性增加，特别是对于小包延迟增加比较明显；Ring太大，Ring All Reduce 效率也会变得很低。

Ring 算法

- NPUs arranged in a logical ring

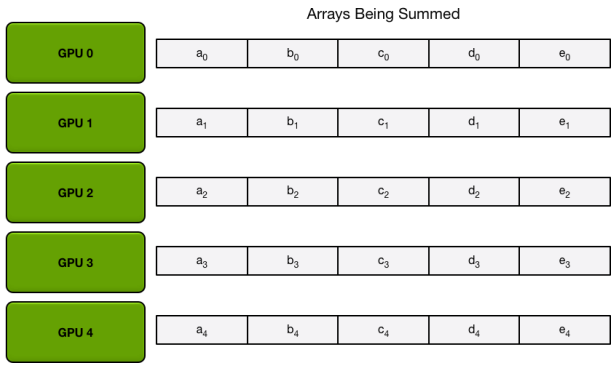


- NPUs Topology

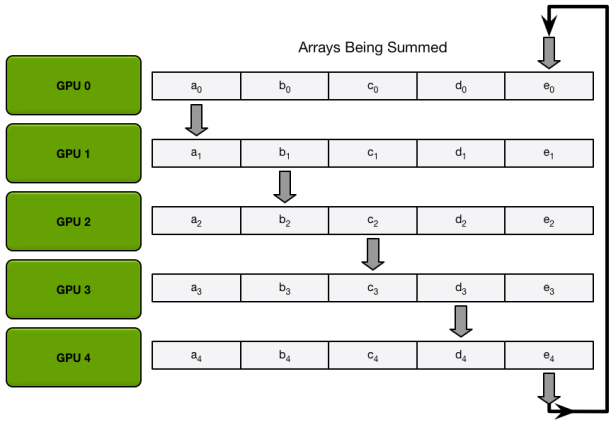


Ring 算法：Scatter-Reduce + All Gather

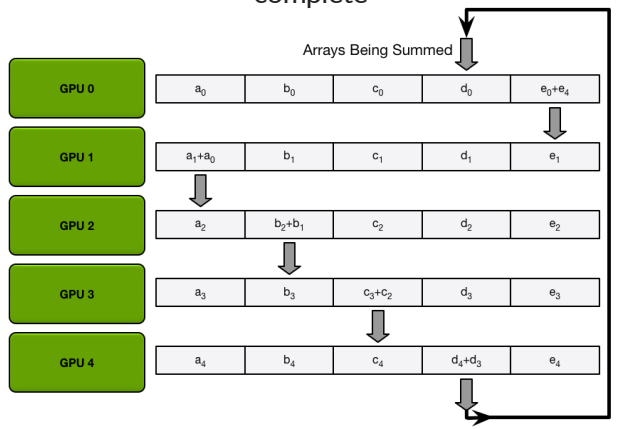
Partitioning of an array into N chunks



Data transfers in the first iteration of scatter-reduce

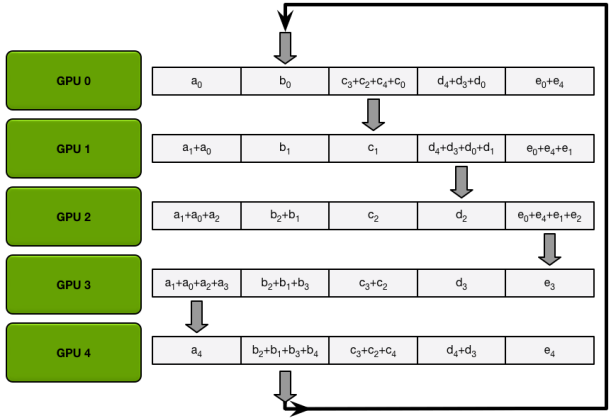


Itermediate sums after the first iteration of scatter-reduce is complete

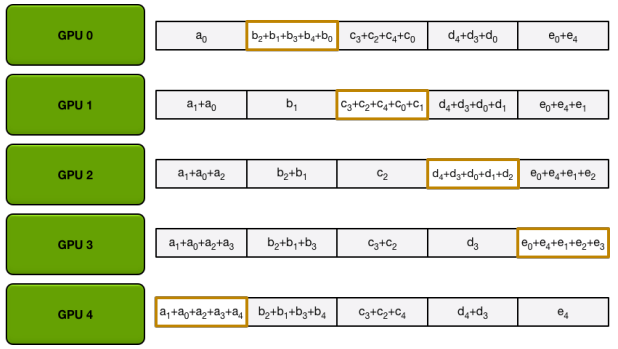


...

Scatter-reduce data transfers (iteration 4)

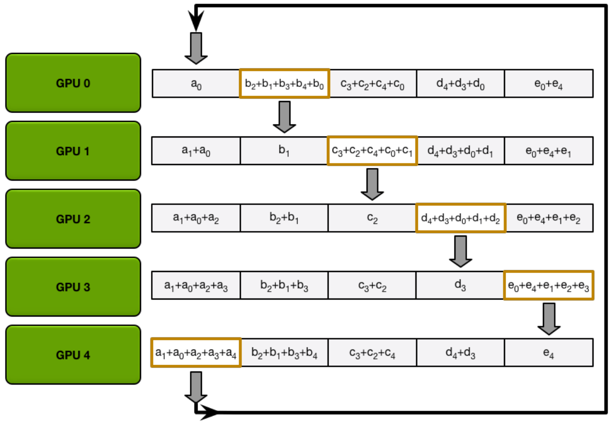


Final state after all scatter-reduce transfers

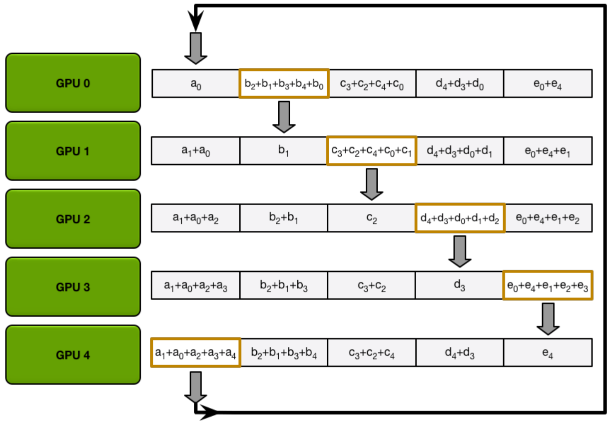


Ring 算法：Scatter-Reduce + All Gather

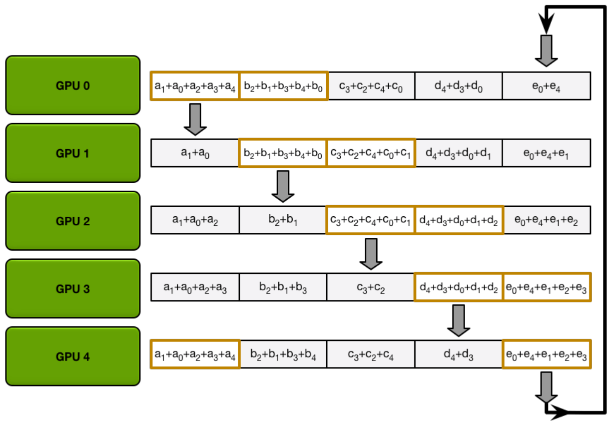
Data transfers in the first iteration of the allgather



Allgather data transfers (iteration 1)

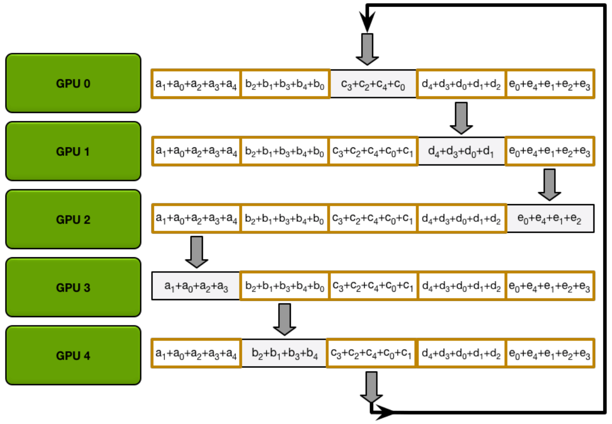


Allgather data transfers (iteration 2)

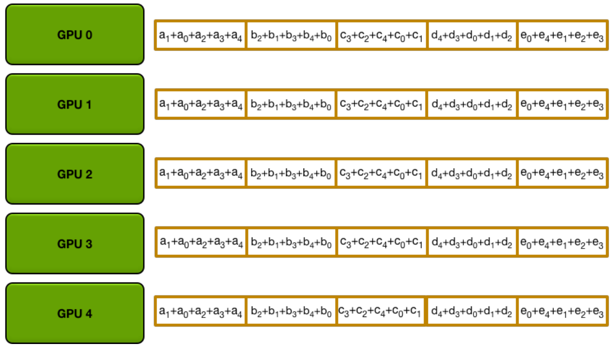


…

Allgather data transfers (iteration 4)

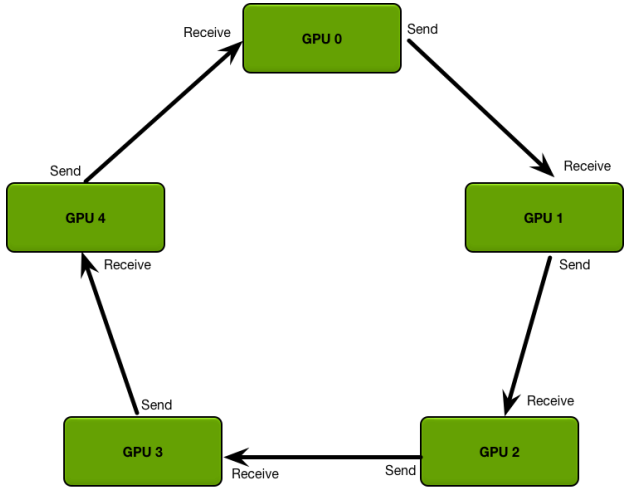


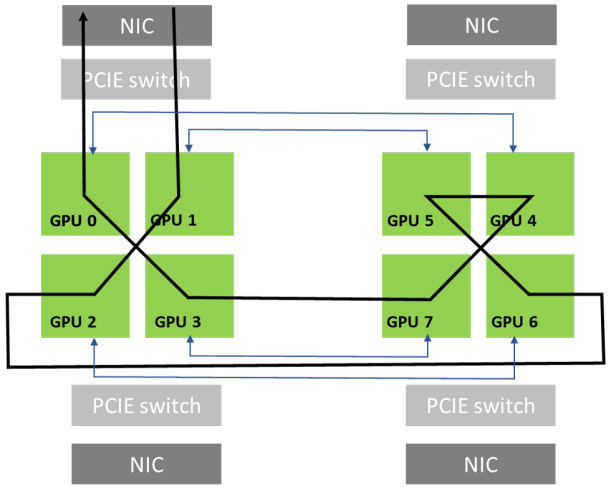
Final state after all allgather transfers



Ring 算法：数据量与通信步

- Ring 算法将进程组织成一个逻辑环（Ring），数据沿着环流动。该算法执行执行 $2(N-1)$ 步，每一步进程 r 向其邻居 $(r+1)\%P$ 发送 P/N 的数据，并接收来自进程 $(r-1)\%N$ 数据，聚合接收到数据，每个进程发送数据总量为 $2P(N-1)/N$。





- 假设 N 为进程数，P 为需要聚合总数据量，数据被分成 N 块，一块为 P/N

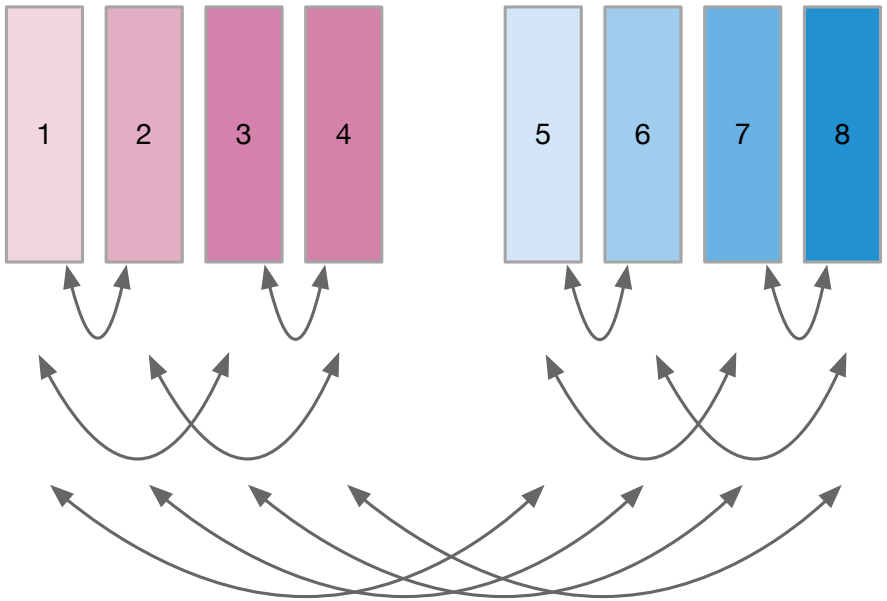
Halving-Doubling 算法

- **基本介绍：**每次选择节点距离倍增的节点相互通信，每次通信量倍减（或倍增），访问步长按照 2^N 增；如昇腾 HCCL，阿里 ACCL 均采用该算法；

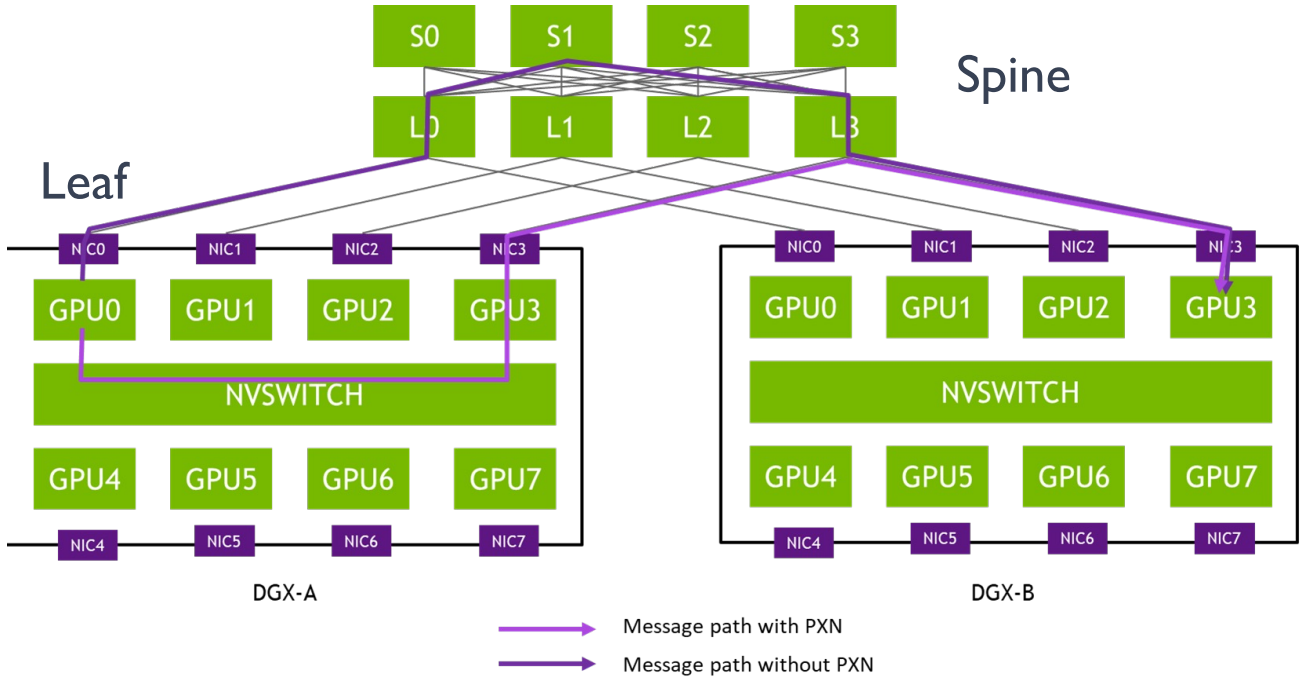
- **适用拓扑：**Fat-Tree 等；**通信步骤：**$\log_2 N$；
- **优点：**通信步骤较少，只有 $2\times\log_2 N$ 次（N 为参与通信 Rank 数）通信即可完成，有更低延迟；并且可以同时并发访问其他 NPU。
- **缺点：**固定并行算法（如 TP=8 否则性能劣化严重），每一个步骤相互通信 Rank 均不相同，链接来回切换会带来额外开销。

Halving-Doubling 算法

- Communication with doubling

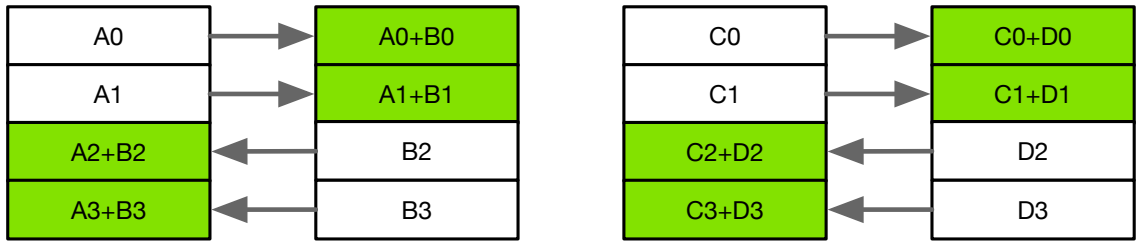


- NPUs Fat Tree Topology

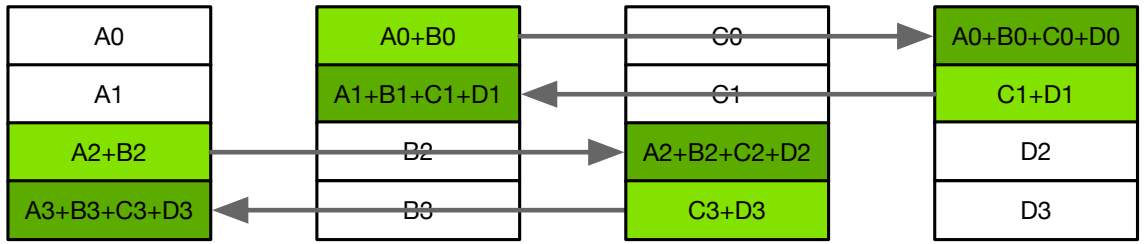


Halving-Doubling 算法

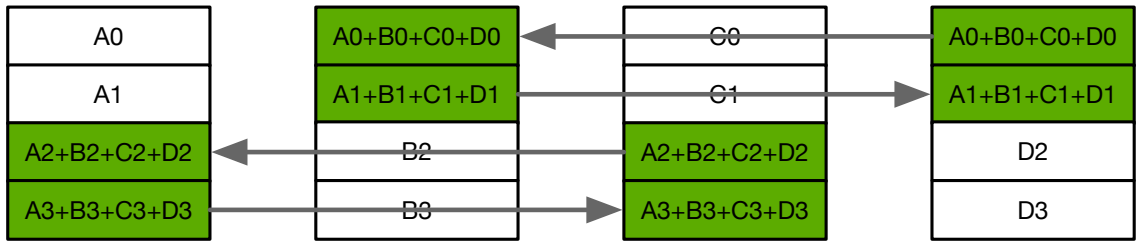
- Reduce Scatter



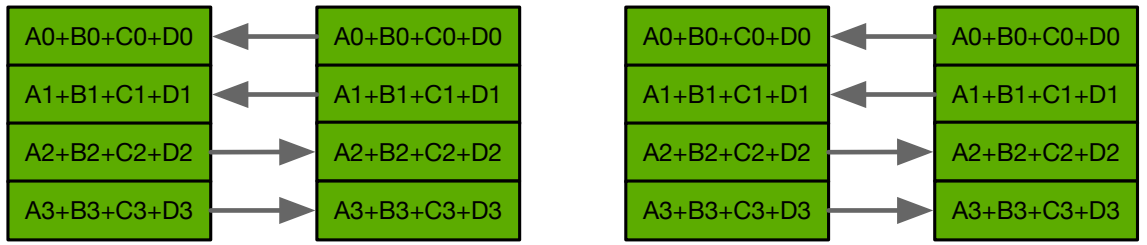
- Reduce Scatter



- All Gather



- All Gather



Halving-Doubling 算法：数据量与通信步

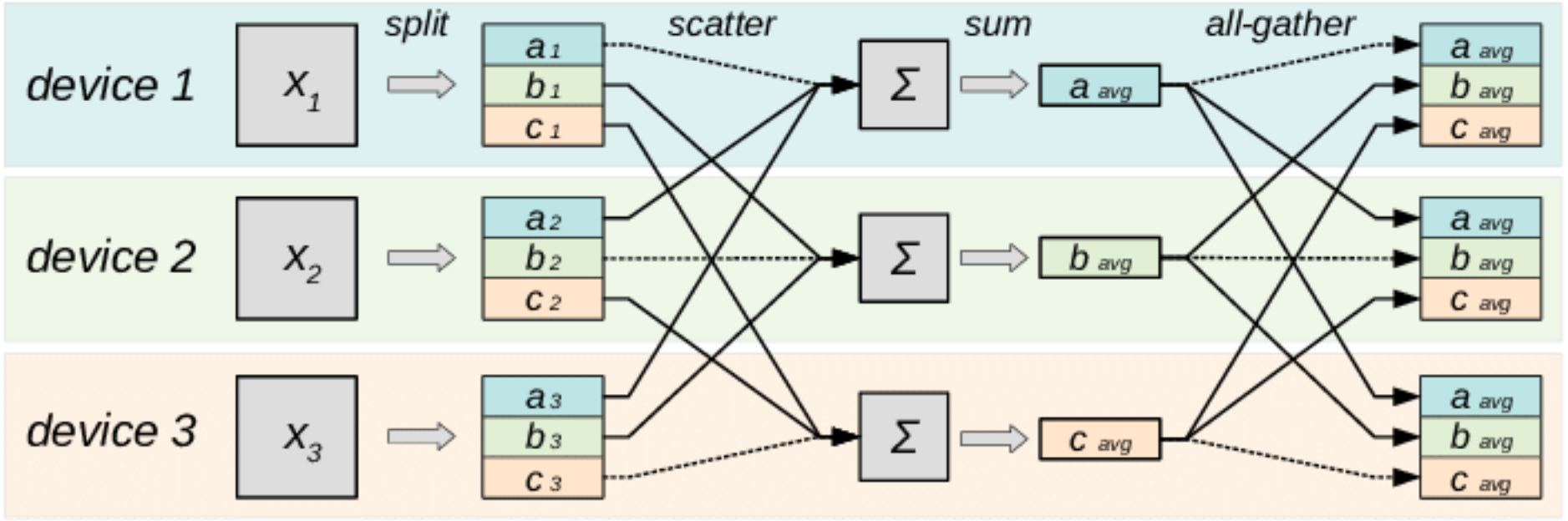
- Halving-Doubling 算法执行 $2\times\log_2 N$ 步，第 1 步距离为 1 两个进程交换 P/2 的数据，第 2 步距离为 2 的两个进程交换 $P/4$ 的数据，第 3 步距离为 4 的两个进程交换 P/8 的数据，以此类推，最后一步交换的数据量为 P/N。因此，每个进程发送数据总量为 $2(P/2+P/4+P/8+\cdots+P/N)=2P(N-1)/N$

对比

算法	步骤数	发送数据量	优缺点对比
Ring	$2\times(N-1)$	$2P(N-1)/N$	• 发送数据量少，聚合大数据性能好； • 用于小规模节点数时优选； • 每次通信域链接不用改变，较为固定； • 连接方式更加稳定；
Halving-Doubling	$2\times\log_2 N$	$2P(N-1)/N$	• 步骤数和发送数据量少，聚合大数据性能好； • 用于大规模节点数时较优； • 固定网络拓扑上可以做到全局无拥； • 通信链接在变化，网络拓扑未知下容易遇到拥塞；

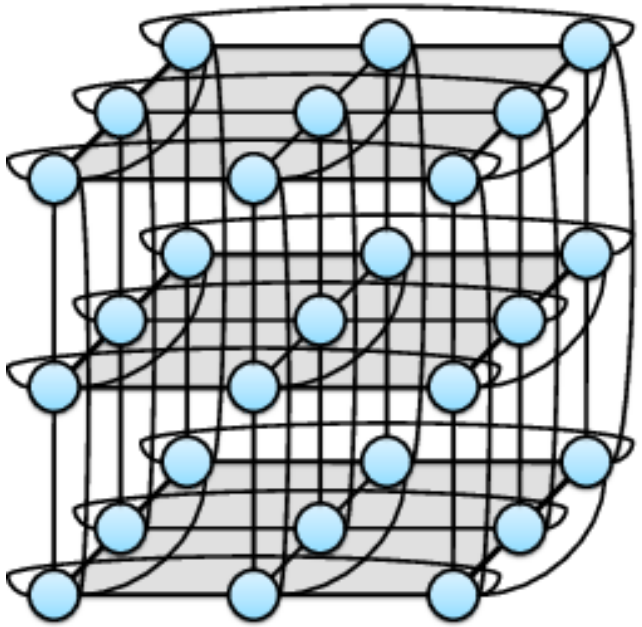
其他 All Reduce 算法

- Swing: Short-cutting Rings for Higher Bandwidth Allreduce
- A schematic illustration of Butterfly All-Reduce

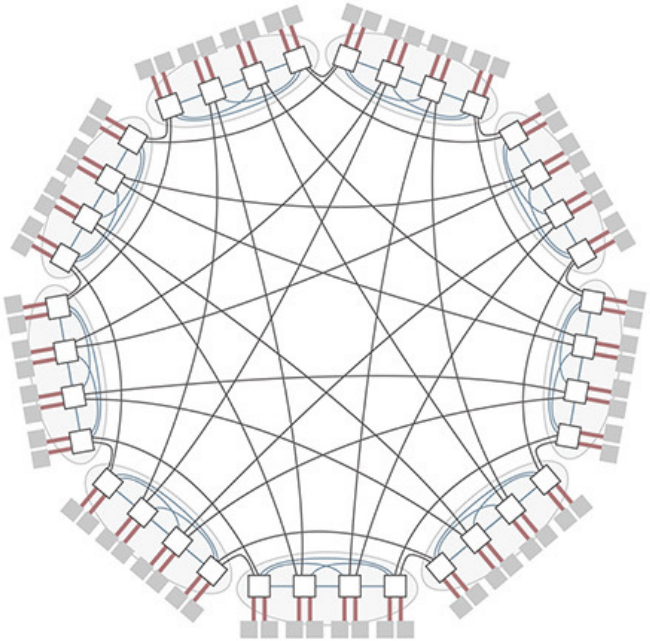


拓扑感知 All Reduce 算法

- 由于All reduce算法的优化空间越来越少，研究者兴趣逐渐转移到了拓扑感知 All Reduce算法（特定拓扑优化All Reduce算法，或者自动感知 All Reduce 算法）：
  - 例如 Torus 拓扑（用于 AI PS 架构），Dragonfly 拓扑（用于超算 HPC）。



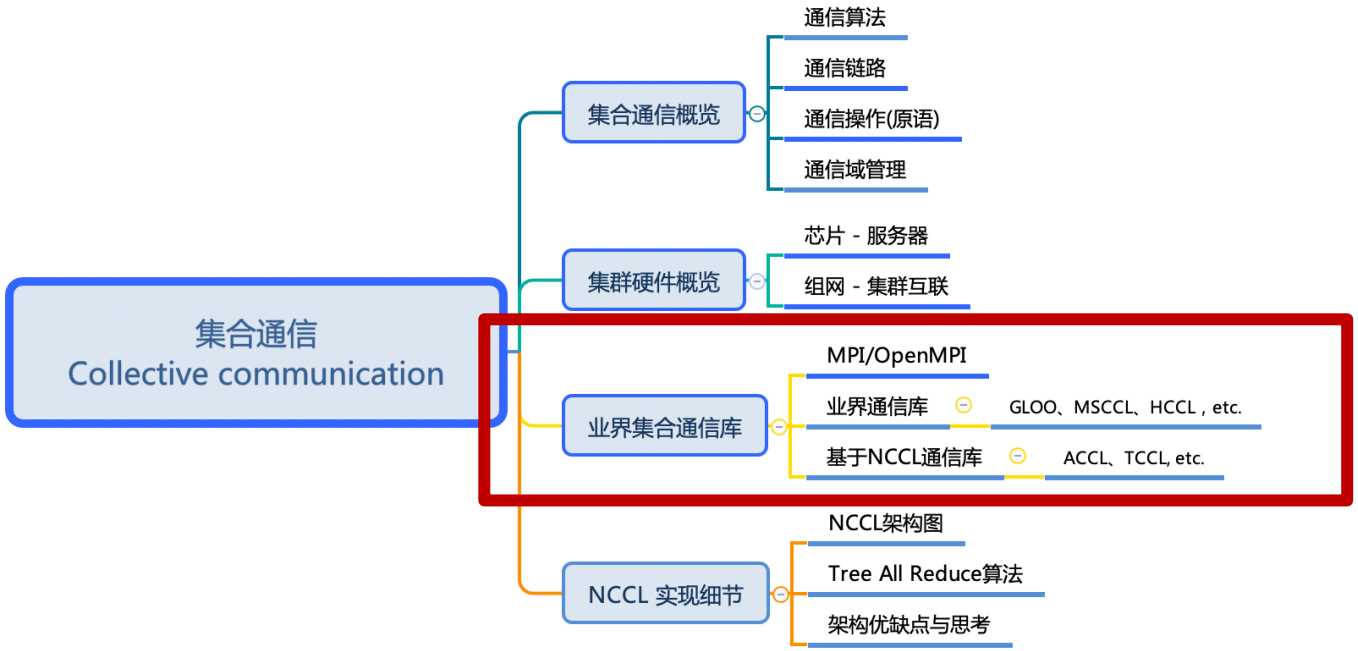
Torus 拓扑



Dragonfly 拓扑

All Reduce 算法

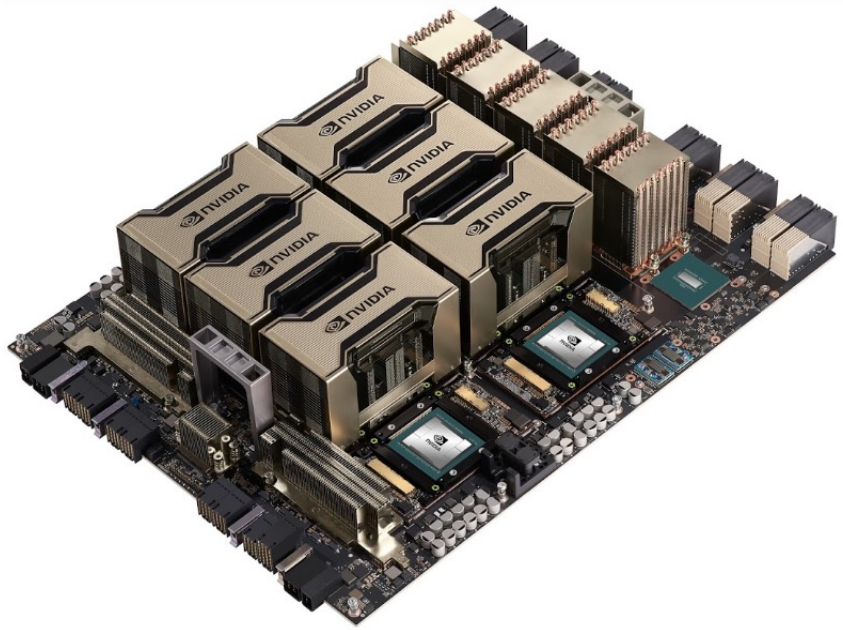
- 经典集合通信算法有 2D-Ring All Reduce、Hierarchical Ring All Reduce，Halving and Doubling All Reduce，Butterfly All Reduce，2D-Torus All Reduce，2D-Mesh All Reduce，Double Binary Tree等。
- 拓扑算法很多，但不是所有拓扑算法都能满足实际生产需求，需要具体问题具体分析、具体场景具体设计，因此出现了XCCL。

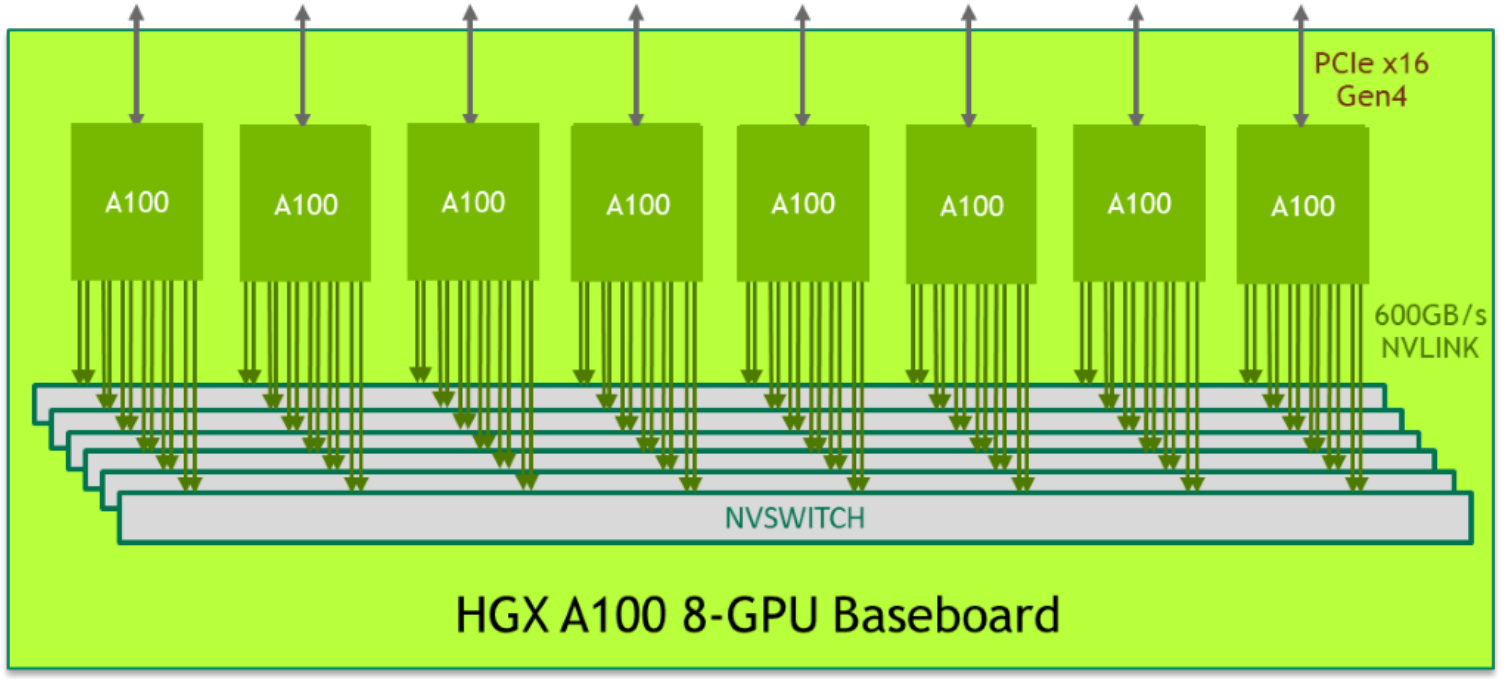


04. How About Nvidia vs Ascend

大模型集合通信性能分析

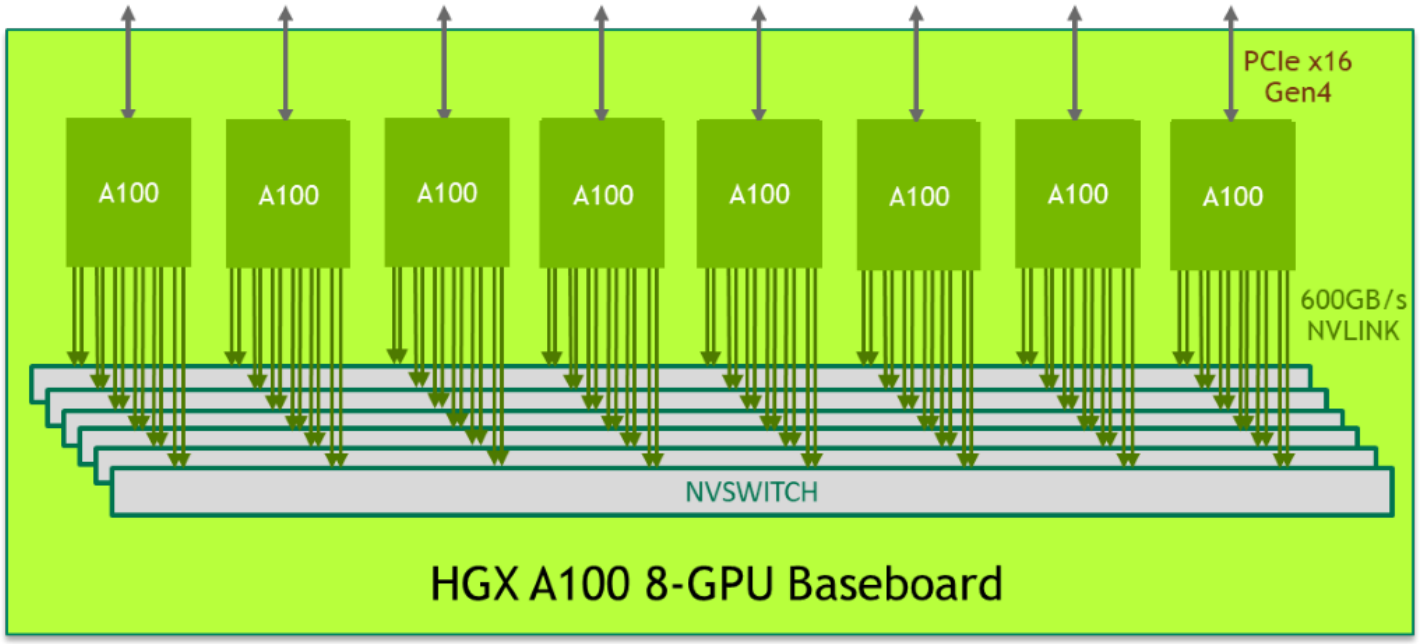
- 组网拓扑不同，并行配置不同，需要理解组网方式才能设置更好的并行策略；

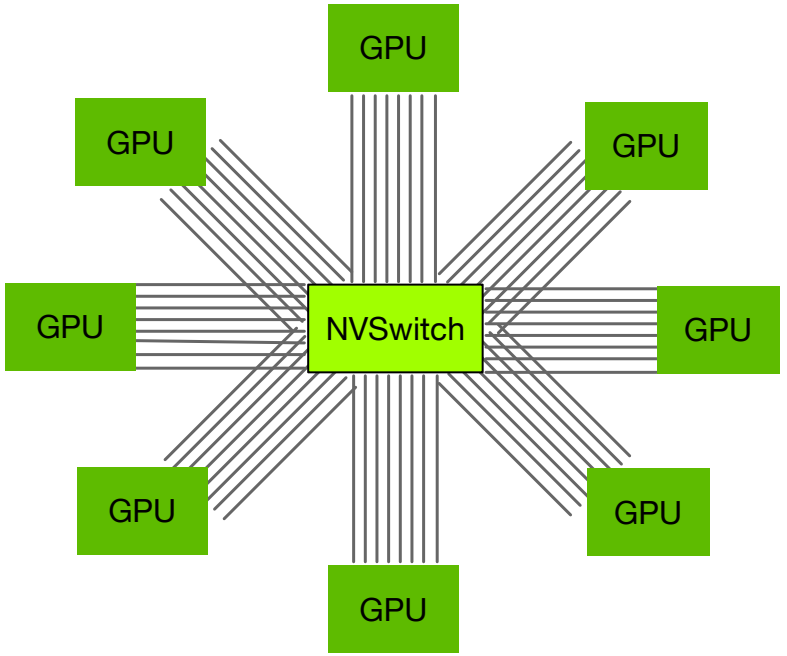




大模型集合通信性能分析

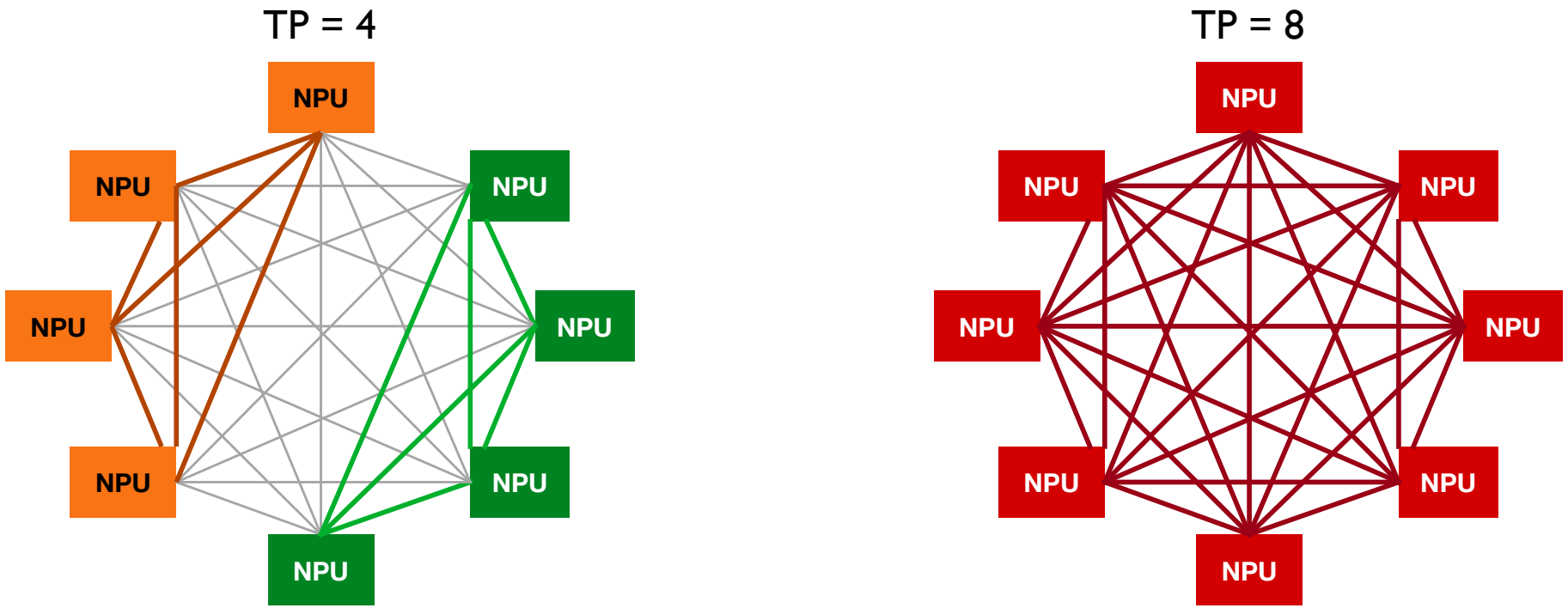
- 组网拓扑不同，并行配置不同，需要理解组网方式才能设置更好的并行策略；





大模型集合通信性能分析

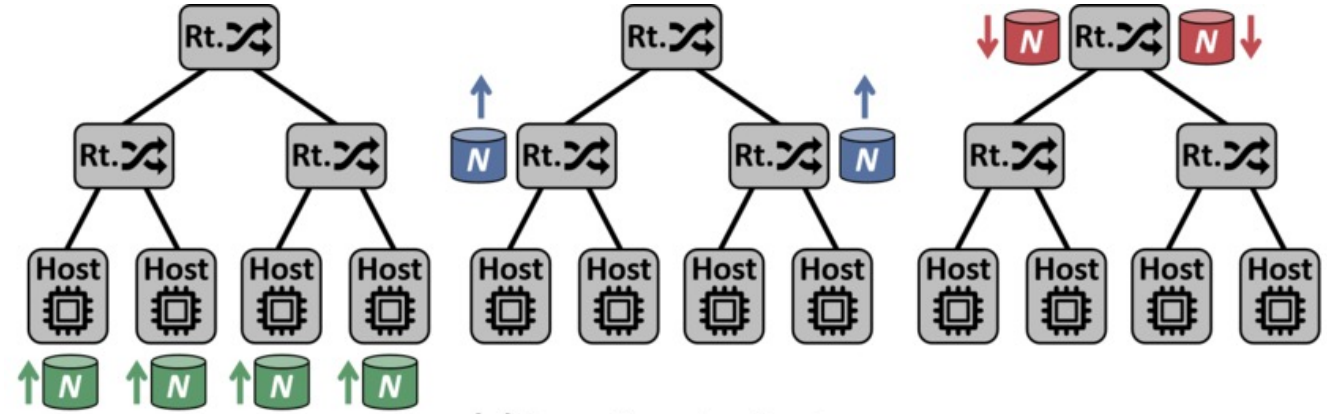
- Full-Mesh 拓扑：
  - TP=4，仅仅能利用 3/7 的 AI 节点内通信链路；存在仅能利用一条 AI 节点内通信链路的 PP 流量。
  - TP=8，可以将 AI 节点内通信链路用满；PP 通过 AI 节点 RDMA 链路通信。



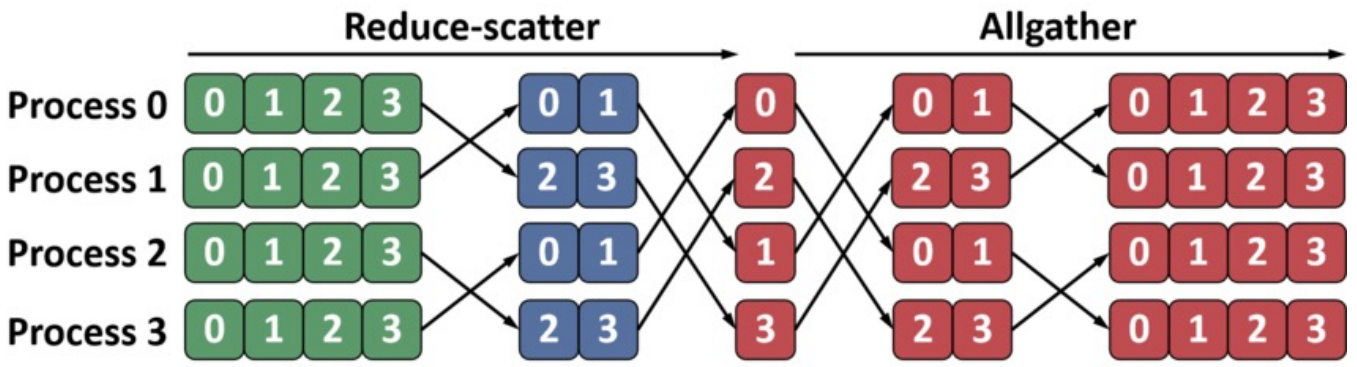
05. 在网计算

在网计算(In-network Computing)

- 把 All Reduce 操作放到交换机或者路由器中去执行。路由器收集并聚合孩子节点的数据，逐层向上进行，聚合的结果从根节点逐层向下广播。



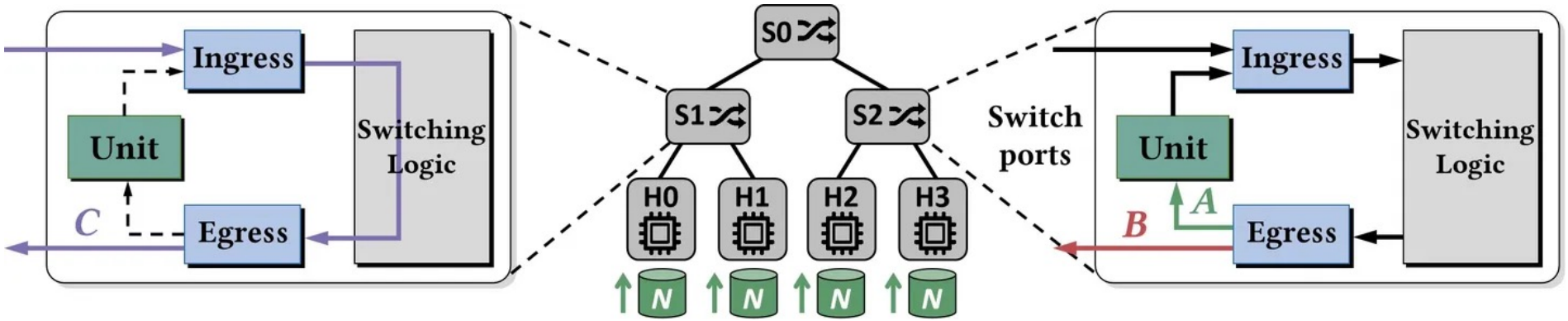
(a) In-network allreduce



(b) Rabenseifner algorithm for allreduce

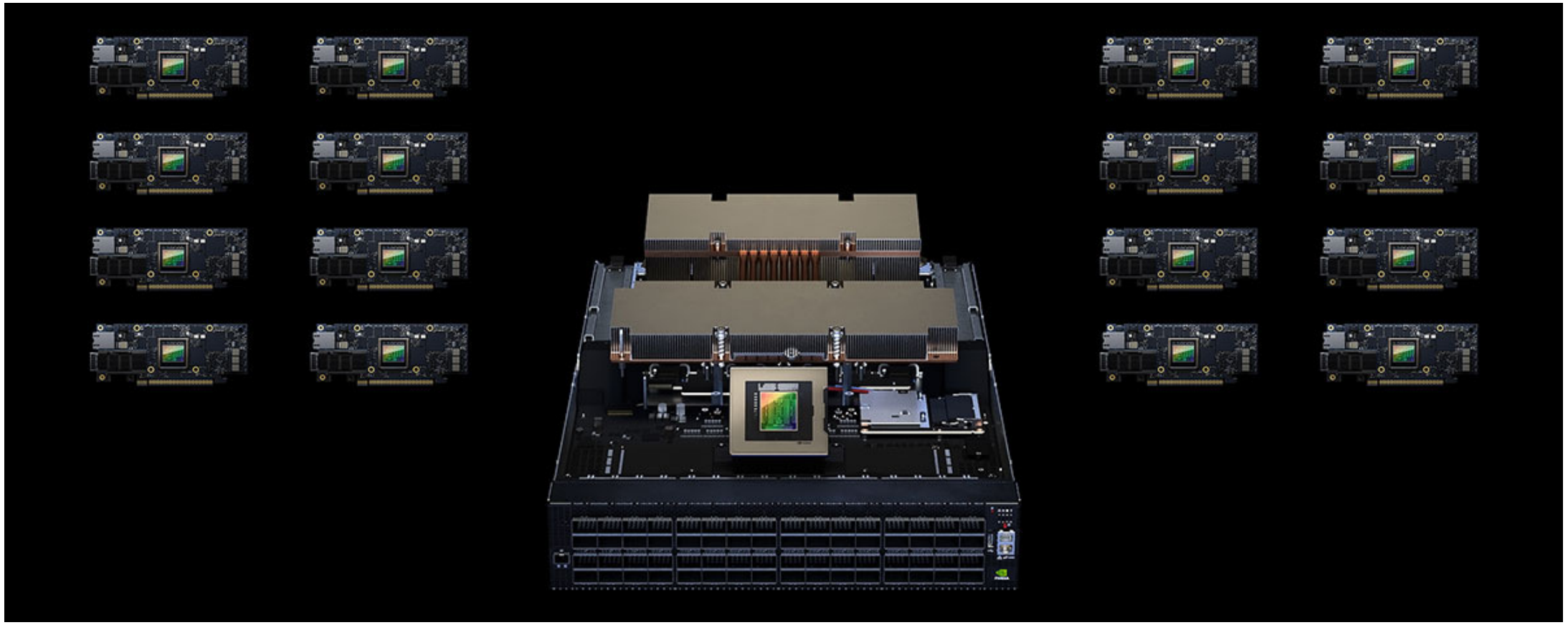
在网计算(In-network Computing)

- 相比传统的Allreduce算法，在网计算有以下几个优势：（1）2倍带宽提升：带宽最优的Allreduce算法要求每个进程发送总量为 $2P-1PN \approx 2N$ 的数据，在网计算只要求每个进程发送总量为N的数据；（2）网络中传输的报文大大减少，网络负载减轻，系统性能提升；（3）计算/通信重叠，主机不需要再去执行Allreduce，从而可以去做其他有用的任务。



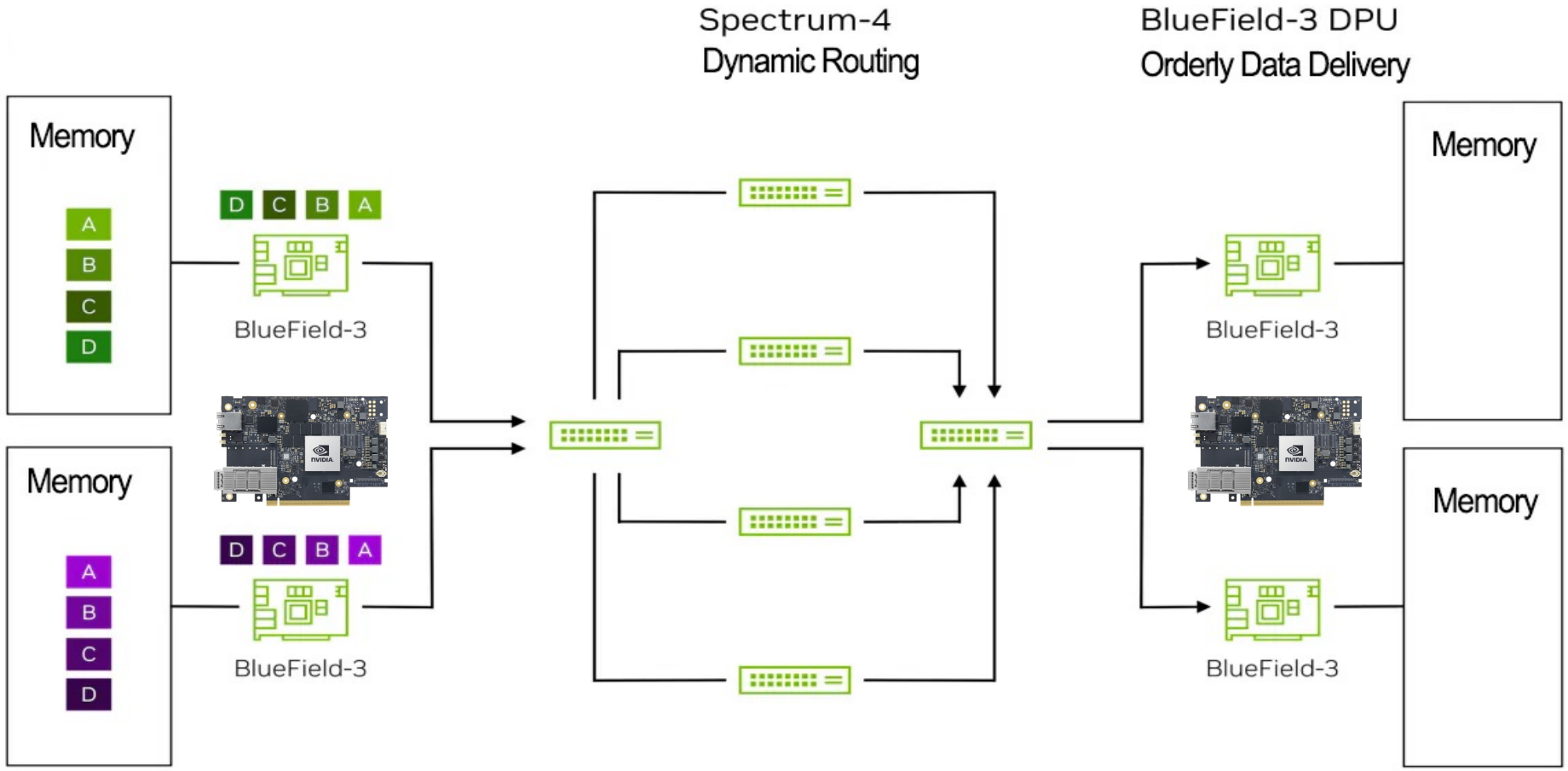
NVIDIA Spectrum-X

- Spectrum-X 包括 NVIDIA DPU、NVIDIA Spectrum-4 以太网交换机，NVIDIA LINK-X 网线，以及上面运行的软件，**是一套以太网端到端的解决方案。。**



NVIDIA Spectrum-X

- NVIDIA BlueField-3 DPU 是一个 400Gb/s 基础设施计算平台，可对软件定义的网络、存储和网络安全进行线速处理。



06. 小结与思考

小结与思考

了解完本内容后：

1. AI 神经网络模型学习/训练阶段为什么要通信（AI 基础知识、训练推理、分布式并行）
2. XCCL 在 AI 系统中的位置（HPC 通信架构 to XCCL 通信架构）
3. 集合通信原语（All Reduce, etc.）
4. 集合通信算法（Ring / Tree / Torus All Reduce etc.）
   - 通信算法在 Fat-Tree 网络拓扑架构下演进趋于稳定，NV 采用 DBT，Ascend 采用 HD/NHR；
   - 通信算法语内存语义结合，与计算融合是性能提升的方向（Flash Attention）；
   - 通信算法需要自适应，在不同规模（百卡/千卡/万卡规模）、不同拓扑（Fat-Tree/Tours）选择；
5. PyTorch 集合通信与计算并行

NCCL 与 HCCL 对比

对比项	NCCL	HCCL	对比
RoCE vs IB	RoCE v2 & IB	RoCE v2	以太兼容性好，生态较优
开源情况	NCCL 开源 (执行 CUDA 不开源)	HCCL Text 开源 (HCCL 代码为开源)	有限开源
节点内拓扑	NvSwitch 互联	Full-Mesh	TP8 下通信性能接近
Reduce 运算能力	通过 SHARP 解决规约 不占用计算核	DMA 引擎随路规约 不占用计算核	通信与计算并行 overlap
兼容三方交换机	IB 协议需要适用 NV 方案 RoCE 协议支持三方交换机	推荐华为参数面网络方案 兼容三方交换机	需要端网配合 减少流量冲突
直出网口	独立网卡	集成设备内部	调度硬化、减少抖动

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/AISystem