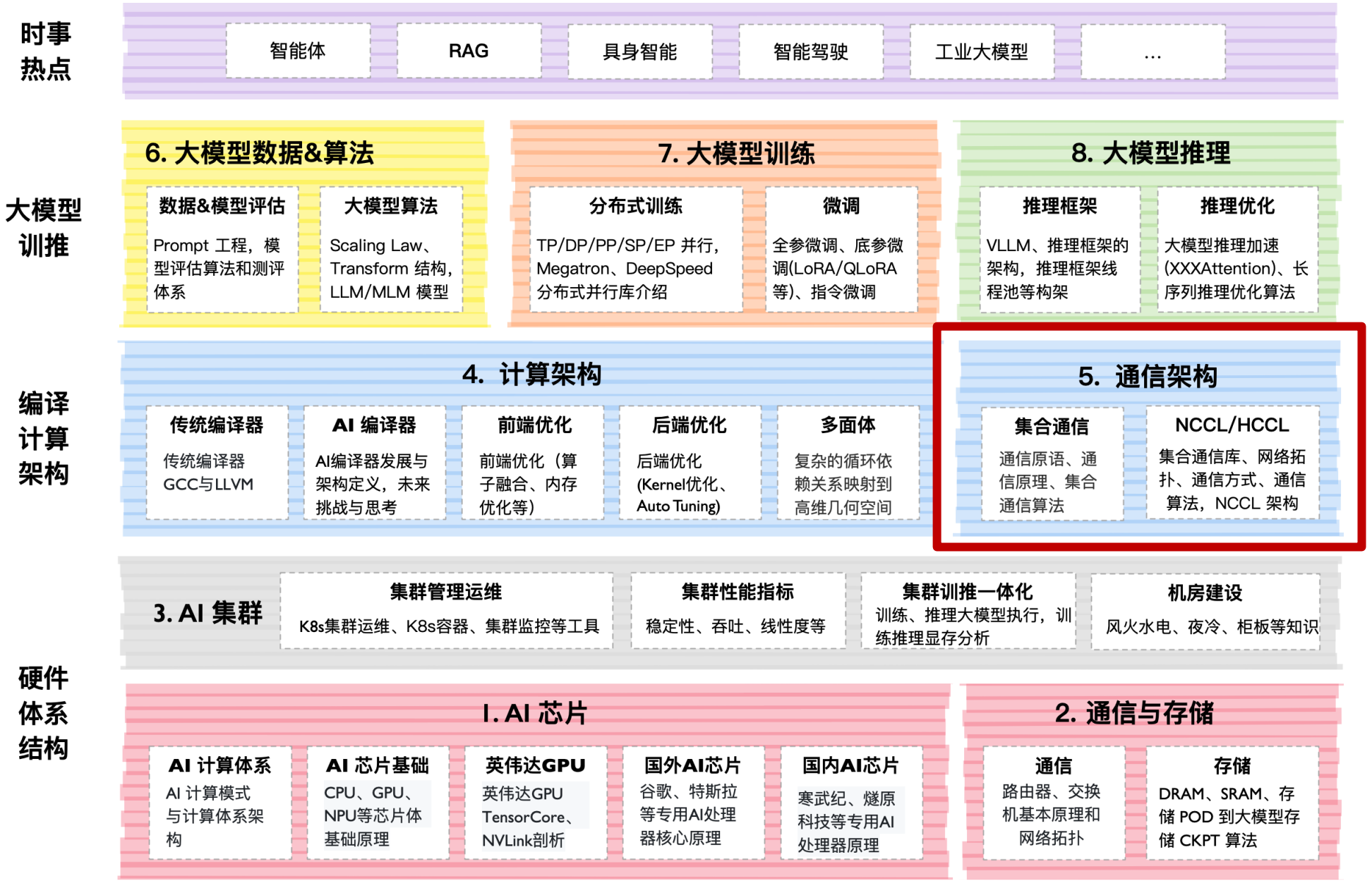
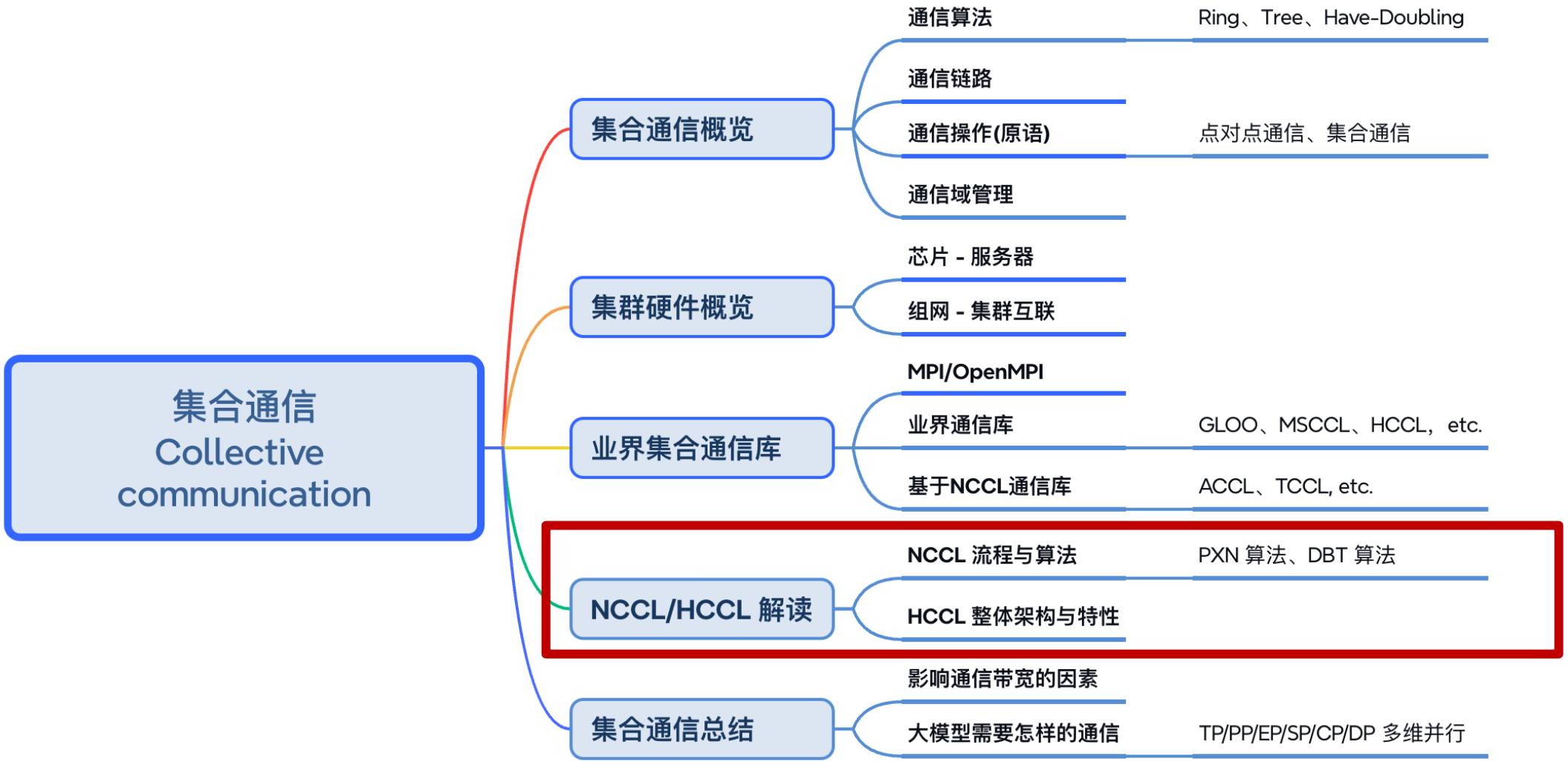
大模型系列 – 集合通信库

HCCL 拓扑算法

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

01. NCCL 与拓扑关系

XCCL 对通信操作支持情况

Table 2. Classic and xCCL's Collective Communication Algorithms

Category	Collective	Algorithm	Description on Suitability (e.g., Message Size, Number of Processes)
Classic	All-to-All	Bruck[42]	Short (e.g., < 32 B)
		Isend-Irecv[43]	Medium (e.g., 32 B to 32 KB)
		Pairwise-Exchange[44]	Long (2^n processes)
	All-Gather	Ring[43]	Long, medium (not 2^n processes)
		Recursive-Doubling[43]	Short, medium (2^n processes)
		Bruck[42]	Short (not 2^n processes)
	Broadcast	Binomial Tree[45]	Short (e.g., < 32 B)
		Van de Geijn[46, 47]	Long (e.g., > 32 KB)
	Reduce-Scatter	Recursive-Halving[44]	Short (commutative reduction)
		Recursive-Doubling[43]	Short (not commutative reduction)
		Pairwise-Exchange[43]	Long (e.g., $\geqslant$ 512 KB for commutative, $\geqslant$ 512 B for noncommutative)
		Binomial Tree and Linear Scatterv[43]	Medium

XCCL 对通信操作支持情况

- HCCL 开源了四种拓扑算法实现源码：

算法	描述
Mesh	Server 内通信算法，Mesh 互联拓扑基础算法
Ring	Server 内 & Server 间通信算法，是基于 Ring 环结构的并行调度算法。 Server 间通信场景下，适用于小规模节点数（<32机，且非2幂）& 中大规模通信数据量（≥ 256M）的场景
Pair Wise	Server 间通信算法，比较算法，仅用于 AlltoAll 与 AlltoAllV 算子，适用于数据量较小（≤1M x RankSize）的场景
Recursive Halving-Doubling（RHD）	Server 间通信算法，递归二分和倍增算法，当通信域内 Server 个数为 2 整数次幂时，算法具有较好的亲和性

XCCL 对通信操作支持情况

- HCCL采用 $a-\beta$ 模型（ Hockney）进行性能评估，算法耗时计算用到的变量定义如下:
  - α：节点间的固定时延
  - β：每 byte 数据传输耗时
  - n：节点间通信的数据大小，单位为 byte
  - γ：每 byte 数据规约计算耗时
  - ρ：通信域节点个数，影响通信步数
  - 其中单步传输并规约计算 n byte 数据的耗时为：$D = \alpha + n\beta + n\gamma$

Hockney 模型

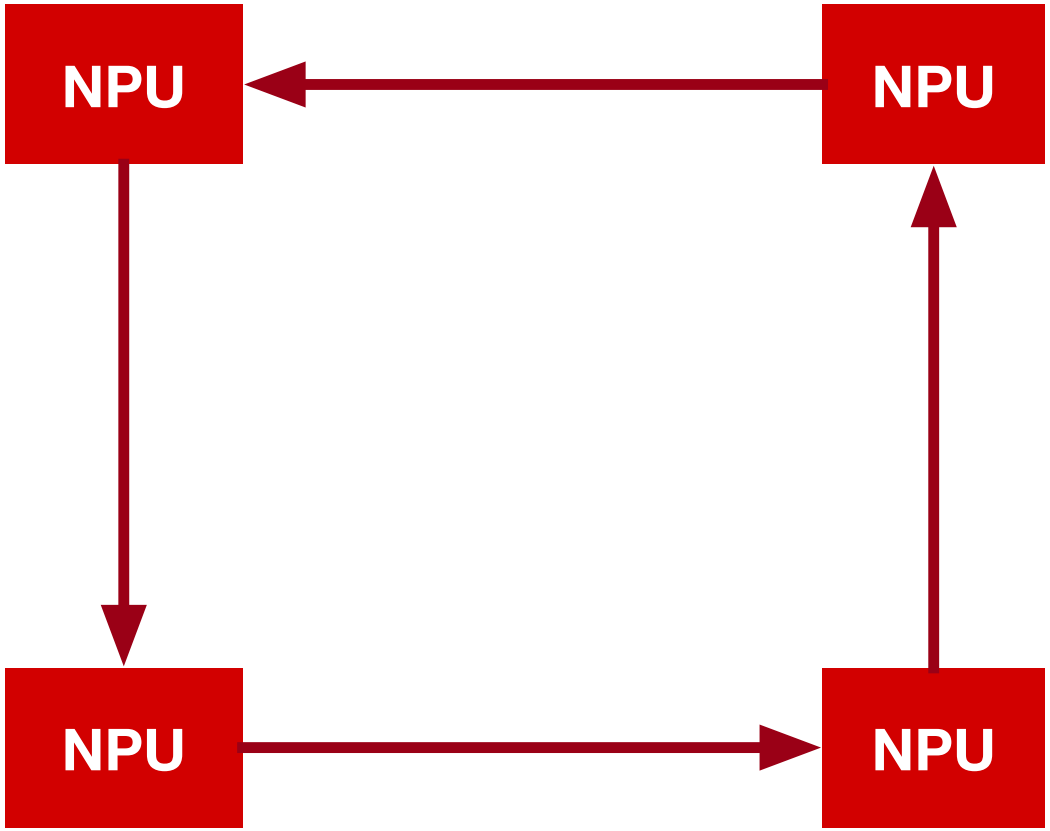
- Hockney 模型通常也被称为 $\alpha - \beta$ 模型，在 Postal 模型的基础上引入了带宽，并且限制每次传输的数据量。
- Hockney 模型一次点对点通信的成本 $T_{p2p}(n) = \alpha + n * \beta$。Hockney 模型假设一个节点可以同时执行一个发送操作和一个接收操作。

- 其中，
  - α：节点间的固定时延
  - β：每 byte 数据传输耗时

02. Ring 算法

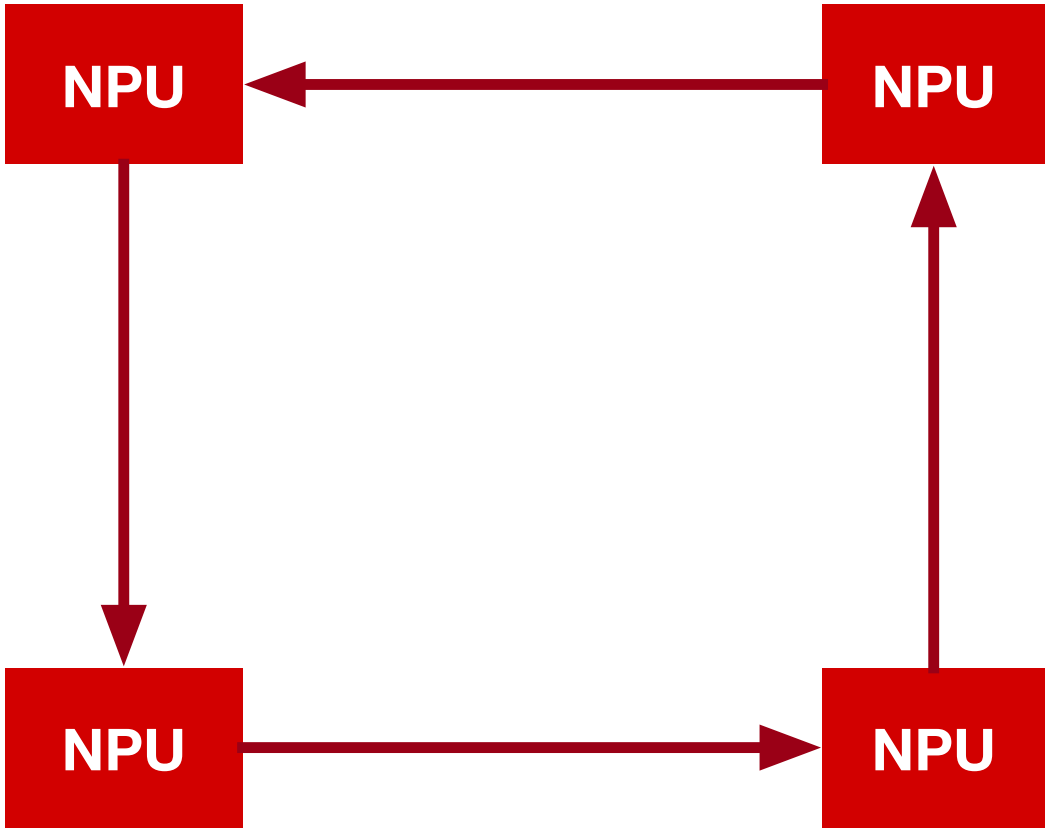
Ring 算法描述

- **Ring 算法：**所有 NPU 以环形相连，每张 NPU 都有左和右卡，一个负责数据接收，一个负责数据发送，循环完成梯度累加，再循环做参数同步。



Ring 算法描述

- Ring 算法适用于星型或胖树拓扑互联；
- **特点：**通过 Ring 环将所有 NPU 设备单端口双工链路串联起来；
- **复杂度：**时间复杂度是 $O(n-1)$



Ring 算法耗时计算

- 将所有参与节点构成环，每个节点只和左右节点通信。若节点数为 p，则需要通信次数为 $p-1$，每次交换 $1/p$ 数据。

操作	耗时
Scatter	$(p-1)(\alpha+\frac{n}{p}\beta)=(p-1)\alpha+\frac{p-1}{p}n\beta$
Gather	$(p-1)(\alpha+\frac{n}{p}\beta)=(p-1)\alpha+\frac{p-1}{p}n\beta$
Broadcast	$(p-1)(\alpha+n\beta)=(p-1)\alpha+(p-1)n\beta$
Reduce	$(p-1)(\alpha+n\beta+n\gamma)=(p-1)\alpha+(p-1)n\beta+(p-1)n\gamma$
ReduceScatter	$(p-1)(\alpha+\frac{n}{p}\beta+\frac{n}{p}\gamma)=(p-1)\alpha+\frac{p-1}{p}n\beta+\frac{p-1}{p}n\gamma$
Allgather	$(p-1)(\alpha+\frac{n}{p}\beta)=(p-1)\alpha+\frac{p-1}{p}n\beta$
AllReduce	实现为ReduceScatter + Allgather: $2(p-1)\alpha+2\frac{p-1}{p}n\beta+\frac{p-1}{p}n\gamma$

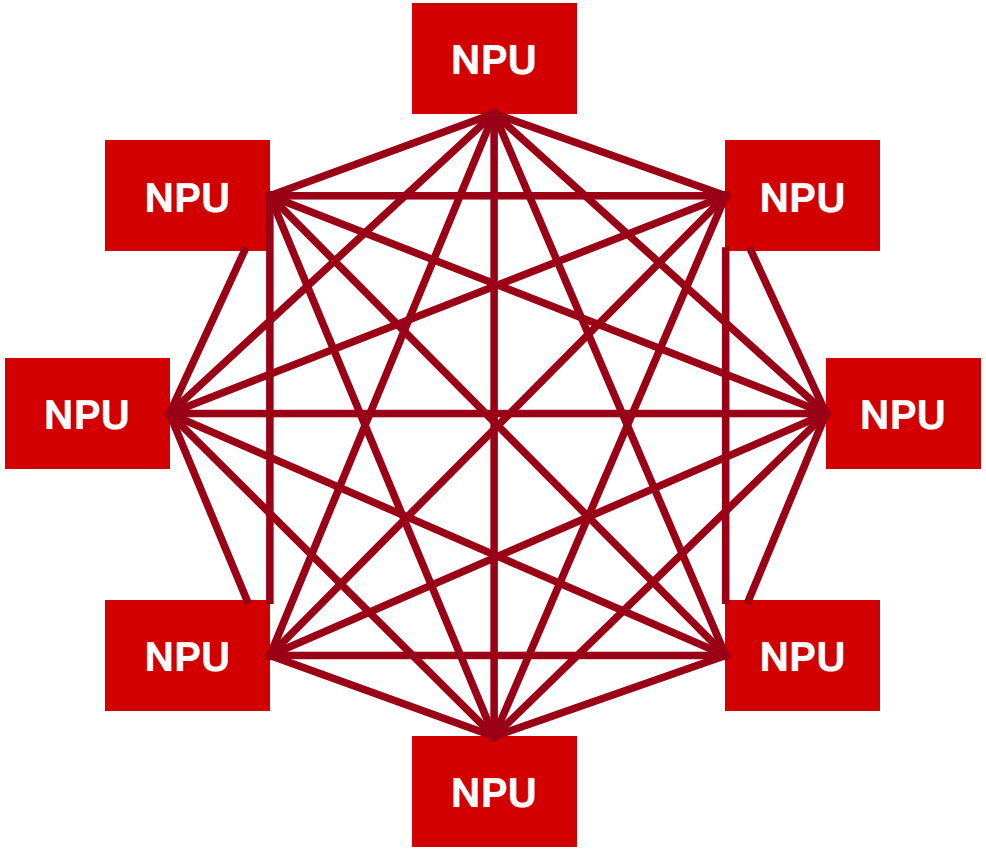
Ring 算法耗时计算

操作	耗时
Scatter	$(p-1)(\alpha+\frac{n}{p}\beta)=(p-1)\alpha+\frac{p-1}{p}n\beta$
Gather	$(p-1)(\alpha+\frac{n}{p}\beta)=(p-1)\alpha+\frac{p-1}{p}n\beta$
Broadcast	$(p-1)(\alpha+n\beta)=(p-1)\alpha+(p-1)n\beta$
Reduce	$(p-1)(\alpha+n\beta+n\gamma)=(p-1)\alpha+(p-1)n\beta+(p-1)n\gamma$
ReduceScatter	$(p-1)(\alpha+\frac{n}{p}\beta+\frac{n}{p}\gamma)=(p-1)\alpha+\frac{p-1}{p}n\beta+\frac{p-1}{p}n\gamma$
Allgather	$(p-1)(\alpha+\frac{n}{p}\beta)=(p-1)\alpha+\frac{p-1}{p}n\beta$
AllReduce	实现为ReduceScatter + Allgather: $2(p-1)\alpha+2\frac{p-1}{p}n\beta+\frac{p-1}{p}n\gamma$

03. Mesh 算法

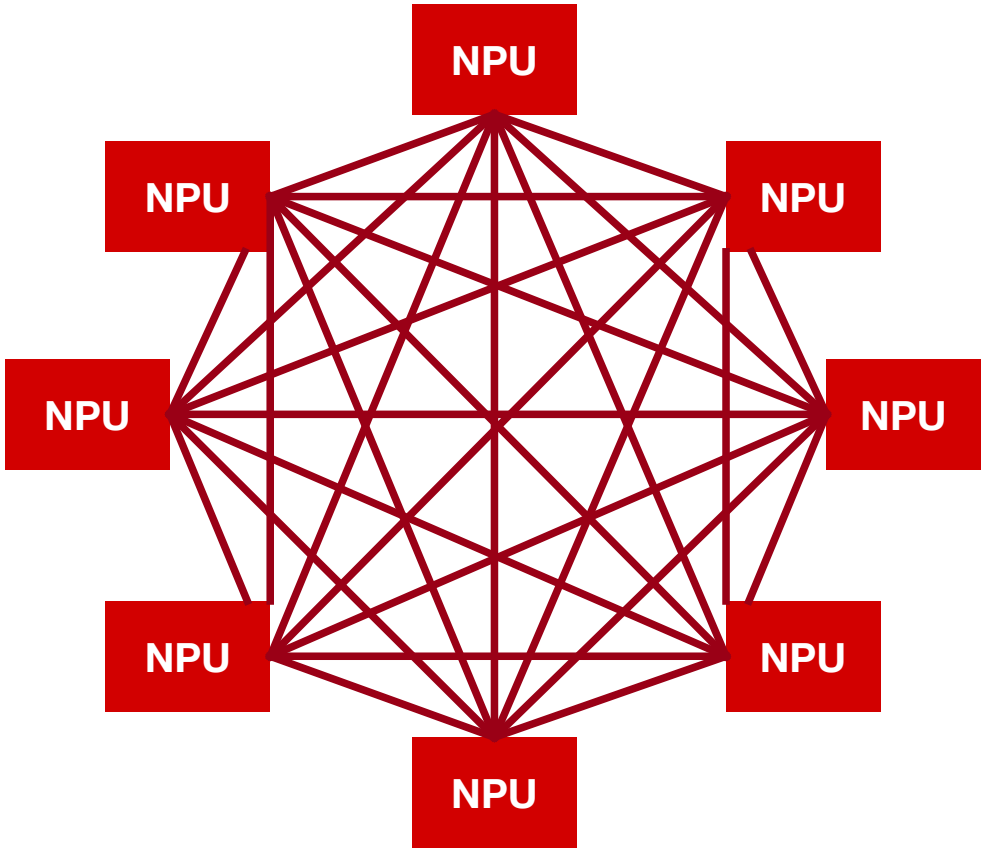
Mesh 算法描述

- **Mesh算法：** Full Mesh 互联拓扑内的基础算法，是 NPU 之间的全连接，任意两个 NPU 之间可以直接进行数据收发。



Mesh 算法描述

- Mesh算法实现原理：每个 NPU 并发的使用多路 HCCS 链路从对端读取或者写入数据，使双工互联链路的双向带宽同时得到利用。
- Mesh 算法时间复杂度是 O(1)。



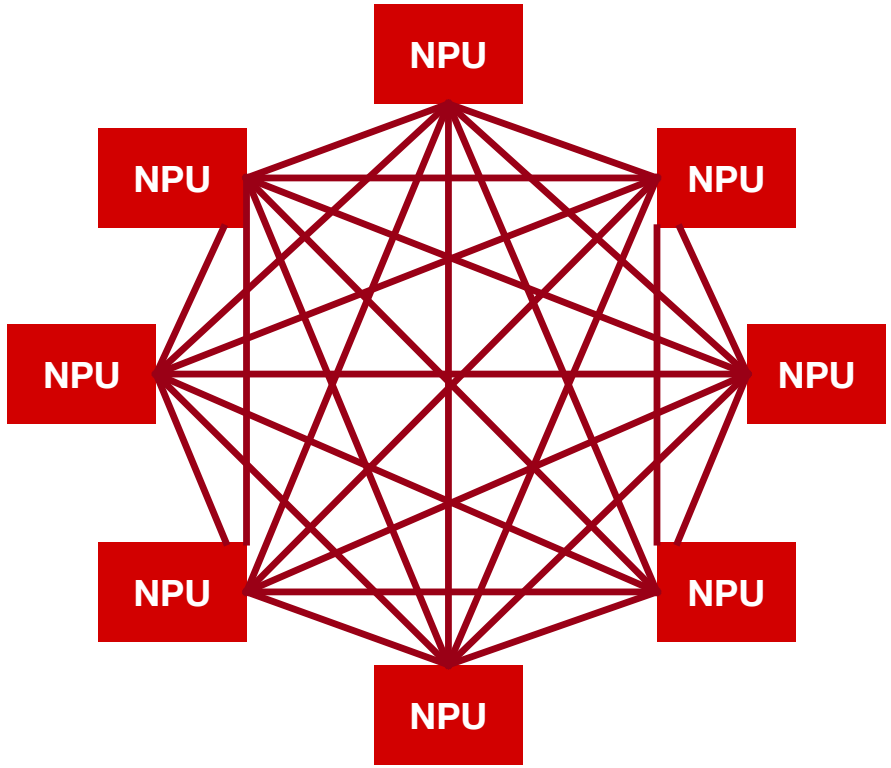
Mesh 算法耗时计算

操作	耗时
Scatter	Scatter算子是单流，耗时为: $(p-1)(\alpha+\frac{1}{p}n\beta)=(p-1)\alpha+\frac{p-1}{p}n\beta$
ReduceScatter	ReduceScatter算子是单流，耗时为： $(p-1)(\alpha+\frac{1}{p}n\beta+\frac{1}{p}n\gamma)=(p-1)\alpha+\frac{p-1}{p}n\beta+\frac{p-1}{p}n\gamma$
AllGather	Allgather算子是多流实现，耗时为： $(p-1)\alpha+\frac{1}{p}n\beta$

04. Pair Wise 算法

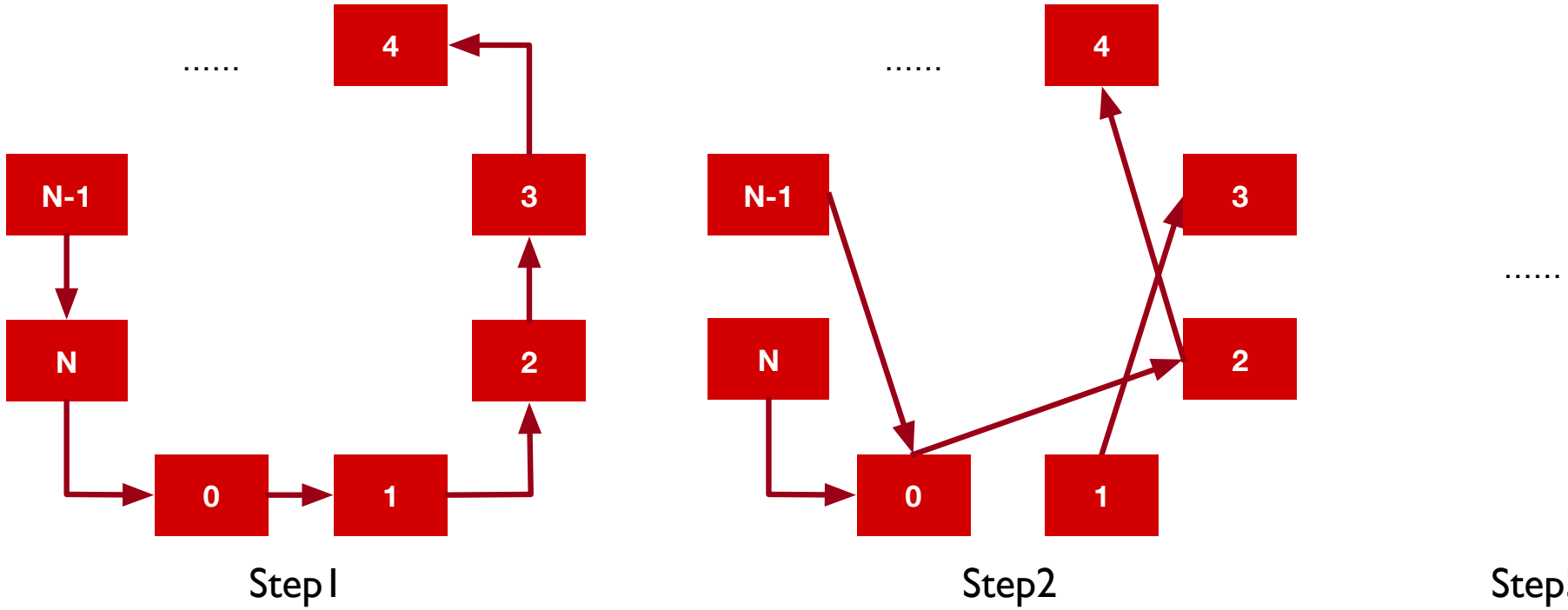
Pair Wise 算法描述

- 每个节点一个 RDMA 网口，若在 RDMA 链路使用 Mesh 算法实现 AllToAll，存在同时从多个节点接收数据、向多个节点发送数据“多打多”冲突，多个流在同一条链路上肆意争抢资源，从而导致整体性能下降。



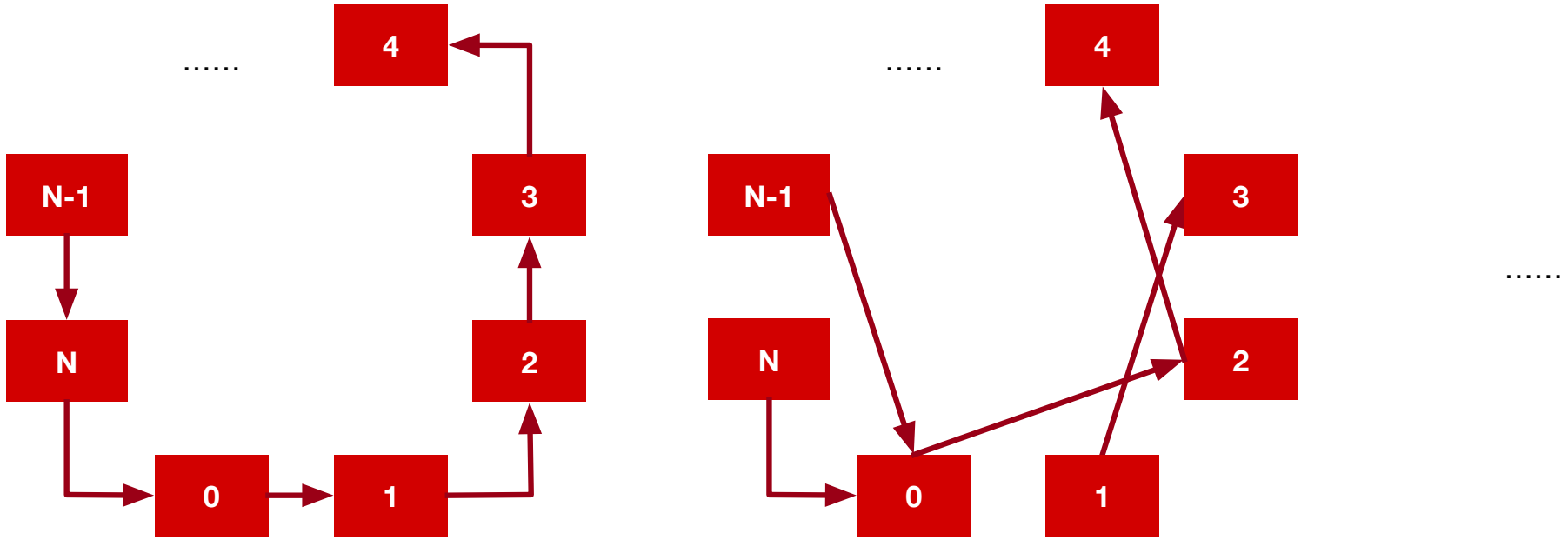
Pair Wise 算法描述

- Pair Wise 算法是 Mesh 算法分 Step 执行版：将通信分解成多 Step，每 Step 只从一个节点接收数据、向一个节点发送数据。



Pair Wise 算法描述

- 对于 Rank ID 为 i 节点，Step1 从 $i-1$ 节点接收数据，向 $i+1$节点发送数据；Step2 从 $i-2$ 节点接收数据，向 $i+2$ 节点发送数据，…… 以此类推。



Pair Wise 算法耗时计算

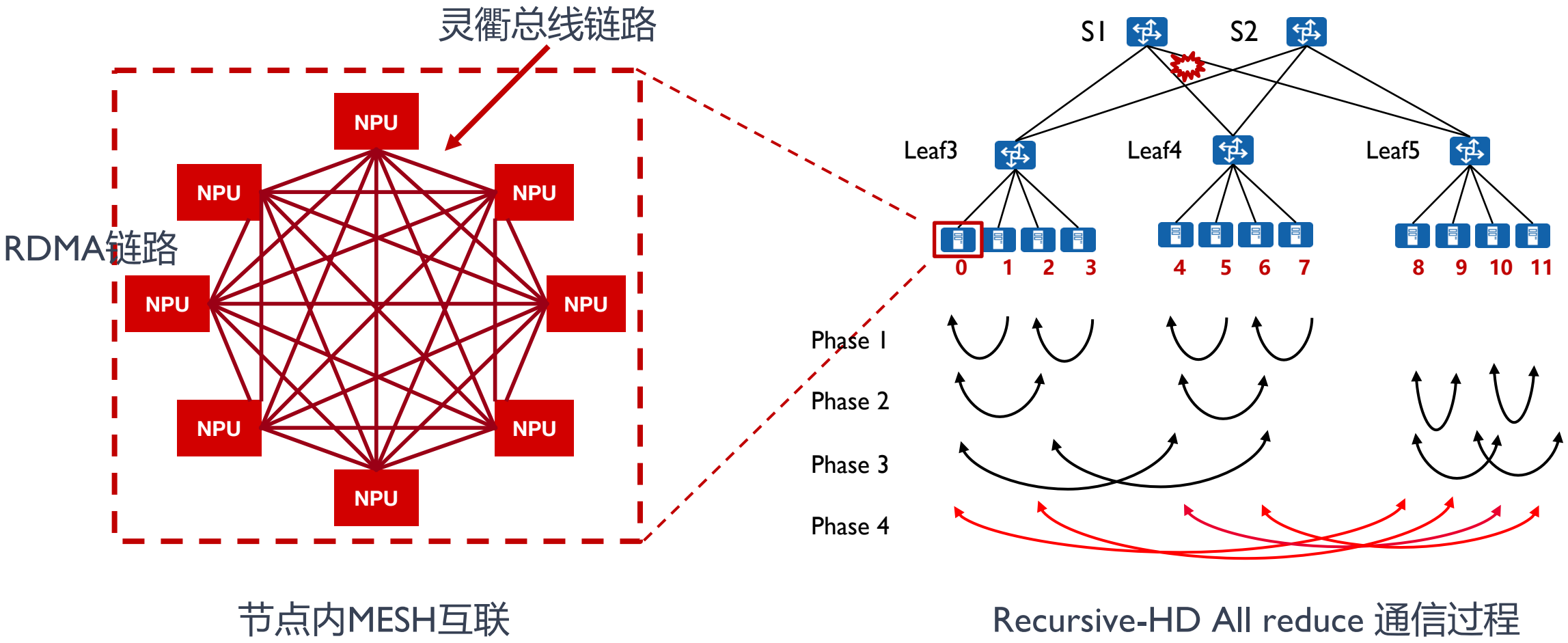
- 定义节点 i 需要给节点j发送的数据大小为n_{ij}
- 对第 k 步，节点 i 发送大小为 $n_{i,i+k}$ 数据给节点 $i+k$，则第 k 步耗时为 $\alpha + \beta \cdot \max_i(n_{i,i+k})$
- 那么，完成整个 Pair Wise 耗时为 $(\mathrm{p}-1)\alpha + \beta \cdot \sum_k max_i\,(n_{i,i+k})$

05. RHD 算法

RHD 算法描述

- 当组网增大时，如增大至 4K 场景，Mesh 算法很难组成 4K 全连接网络，且资源开销（链路资源，交换资源，同步资源）太大，可能存在算力和资源开销不匹配的问题。
- Ring 在这种情况下虽然节省资源（只用左手卡和右手卡进行一次收发），但是环内要做太多次，流转太慢。大型规模集群运算有服务器内数据量庞大、Ring 环极长，Ring 切分数据块方式就不再占优势。

RHD 算法描述

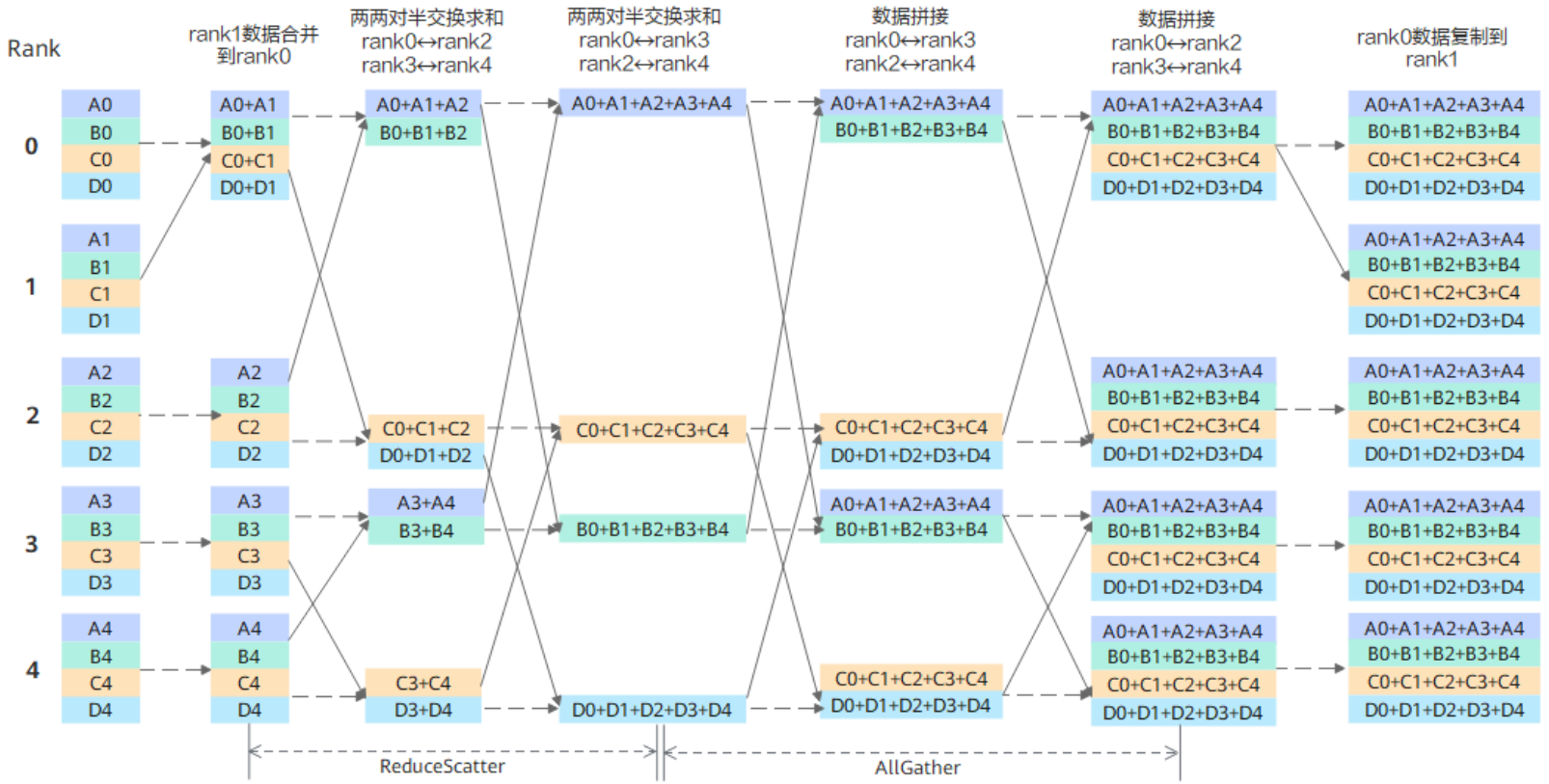


节点内MESH互联

Recursive-HD All reduce 通信过程

RHD 算法描述

- RHD（Recursive Halving-Doubling）算法通过递归加倍及递归折半方式完成NPU间的数据交换，相对Mesh资源消耗较小，相对Ring效率会更高。



RHD 算法分析

- 假设有 $5(2^2+1)$ 个rank，首先将rank1的数据合并到rank0，变成 $4(2^2)$ 个rank，然后将 4 个 rank 的数据两两对半交换数据并求和，即 Reduce Scatter 操作。
- 下一阶段，将这 4 个 rank 数据两两拼接，即 All Gather 操作。最后，将 rank0 数据复制到 rank 1，至此每个 rank 都具有所有 rank 全量数据之和。

- RHD 算法适用于“星型”或“胖树”拓扑互联，算法的时间复杂度是 $\log_2 N$。

RHD 算法耗时计算

- Recursive Havling-Doubling 为递归二分和倍增算法，对于2的整数次幂，使用Vector/Distance Havling/Doubling策略.
- 对于非 2 整数次幂，划分为 2r（part1）和 p-2r 两部分，先将 part1 合并为 r，使得剩余 rank 之和为 p-r（block），再执行 2 整数次幂 HD 算法，最后在 part1 部分恢复出 2r，得到最终结果。

RHD 算法耗时计算

操作	耗时
Broadcast	根据root rank的奇偶，决定part1部分参与block的为奇数rank还是偶数rank，在block内先执行Distance Havling，再向剩余rank发送一次，总耗时为： $\lceil \log(p) \rceil (\alpha + n\beta)$
ReduceScatter	使用Vector Doubling + Distance Havling（保证Scatter的顺序）。 2的整数次幂时，耗时计算公式为： $\log(p)\alpha + \frac{p-1}{p}n\beta + \frac{p-1}{p}n\gamma$ 非2的整数次幂时： 第一步（Reduce）：$\alpha + n\beta + n\gamma$ 第二步（非均匀分片的ReduceScatter，某些rank持有2份数据），需要做$k = \lfloor \log(p) \rfloor$次通信，每次交换的最大数据量为：$n_i = \lceil \frac{p}{2^{k-i+1}} \rceil \frac{n}{p} \quad i = 1, 2, \ldots k$，总耗时为： $\Sigma_{i=1}^{k}(\alpha + \frac{1}{p}\lceil \frac{p}{2^i} \rceil n\beta + \frac{1}{p}\lceil \frac{p}{2^i} \rceil n\gamma) = \lfloor \log(p) \rfloor \alpha + \frac{n\beta}{p}\Sigma_{i=1}^{k}\lceil \frac{p}{2^i} \rceil + \frac{n\gamma}{p}\Sigma_{i=1}^{k}\lceil \frac{p}{2^i} \rceil$ 该步计算比较复杂，这里尝试给出下限和上限： 下限：$k\alpha + (k + 2^k - 1)\frac{n\beta}{p} + (k + 2^k - 1)\frac{n\gamma}{p}$ 上限：$k\alpha + (2^{k+1} - 2)\frac{n\beta}{p} + (2^{k+1} - 2)\frac{n\gamma}{p}$ 第三步（Scatter）：$\alpha + \frac{1}{p}n\beta$

RHD 算法耗时计算

Allgather	耗时同ReduceScatter，无γ相关部分。
Allreduce	ReduceScatter + Allgather：这里的拆分是不完全的ReduceScatter和Allgather，不需要scatter到所有rank，并且可以采用Vector Havling + Distance Doubling（分层网络下耗时会小，但是无法保证顺序，拆分中也不需要保证顺序）。 2的整数次幂: $2\log(p)\alpha + 2\frac{p-1}{p}n\beta + \frac{p-1}{p}n\gamma$ 非2的整数次幂: 第一步（Reduce）: $\alpha + n\beta + n\gamma$ ReduceScatter: $\lfloor\log(p)\rfloor\alpha + \frac{p'-1}{p'}n\beta + \frac{p'-1}{p'}n\gamma, \quad p' = 2^{\lfloor\log(p)\rfloor}$ AllGather: $\lfloor\log(p)\rfloor\alpha + \frac{p'-1}{p'}n\beta, \quad p' = 2^{\lfloor\log(p)\rfloor}$ 最后一步: $\alpha + n\beta$ 总耗时: $(2\lfloor\log(p)\rfloor + 2)\alpha + (2\frac{p'-1}{p'} + 2)n\beta + (\frac{p'-1}{p'} + 1)n\gamma, \quad p' = 2^{\lfloor\log(p)\rfloor}$
Reduce	当前实现为ReduceScatter + Gather。 2的整数次幂: $2\log(p)\alpha + 2\frac{p-1}{p}n\beta + \frac{p-1}{p}n\gamma$ 非2的整数次幂: 第一步（Reduce）: $\alpha + n\beta + n\gamma$ ReduceScatter: $\lfloor\log(p)\rfloor\alpha + \frac{p'-1}{p'}n\beta + \frac{p'-1}{p'}n\gamma, \quad p' = 2^{\lfloor\log(p)\rfloor}$ Gather: $\lfloor\log(p)\rfloor\alpha + \frac{p'-1}{p'}n\beta, \quad p' = 2^{\lfloor\log(p)\rfloor}$ 总耗时: $(2\lfloor\log(p)\rfloor + 1)\alpha + (2\frac{p'-1}{p'} + 1)n\beta + (\frac{p'-1}{p'} + 1)n\gamma, \quad p' = 2^{\lfloor\log(p)\rfloor}$

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