



# SyCCL: Exploiting Symmetry for Efficient Collective Communication Scheduling

**SIGCOMM'25**

Jiamin Cao<sup>†\*</sup>, Shangfeng Shi<sup>‡\*</sup>, Jiaqi Gao<sup>†</sup>, Weisen Liu<sup>‡</sup>, Yifan Yang<sup>‡</sup>, Yichi Xu<sup>†</sup>, Zhilong Zheng<sup>†</sup>, Yu Guan<sup>†</sup>, Kun Qian<sup>†</sup>, Ying Liu<sup>‡</sup>, Mingwei Xu<sup>‡</sup>, Tianshu Wang<sup>†</sup>, Ning Wang<sup>†</sup>, Jianbo Dong<sup>†</sup>, Binzhang Fu<sup>†</sup>, Dennis Cai<sup>†</sup>, Ennan Zhai<sup>†</sup>

<sup>†</sup>Alibaba Cloud

<sup>‡</sup>Tsinghua University



Presenter: 孙铂钛  
2026.1.5

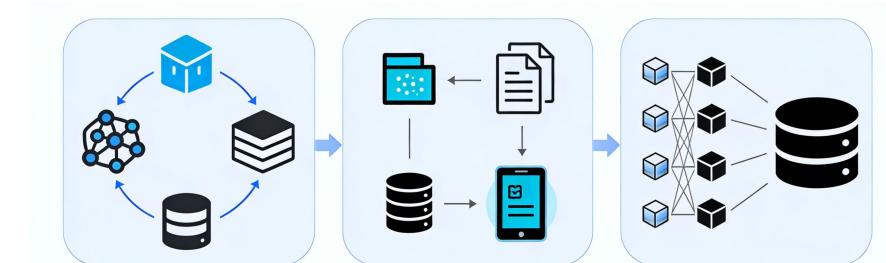
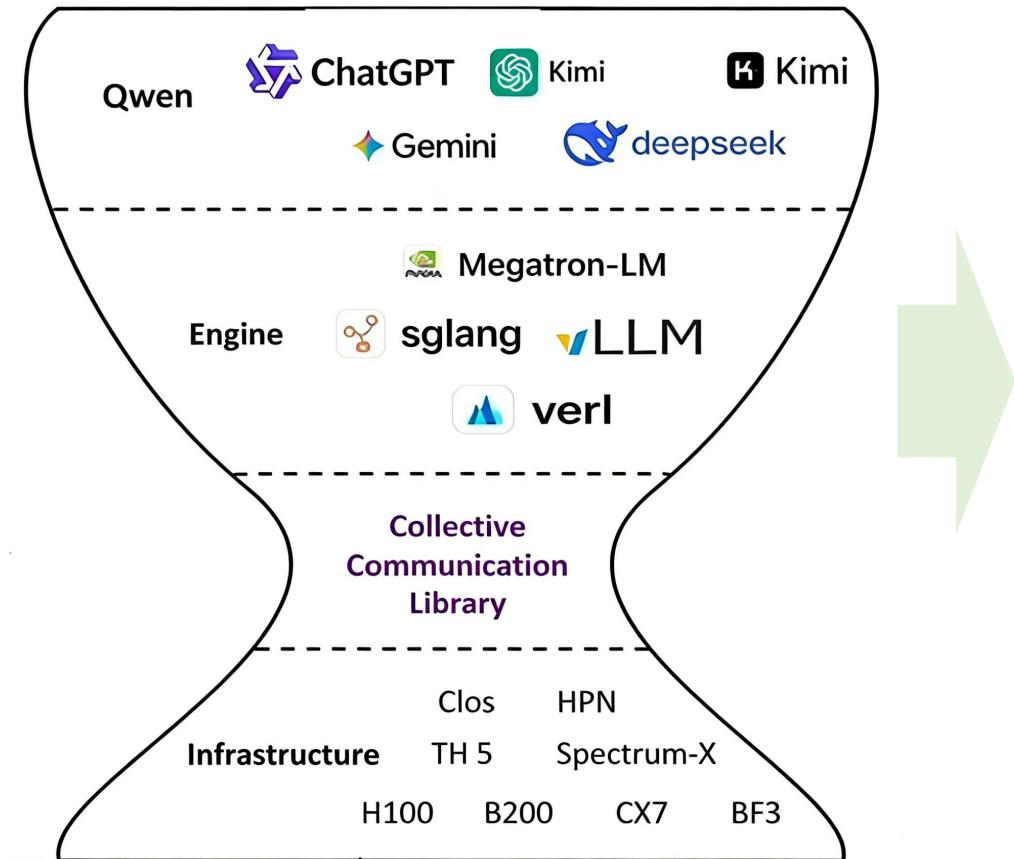
# Content

- **Background**
- **Motivation**
- **Related work**
- **Design**
- **Evaluation**
- **Conclusion**

# Background

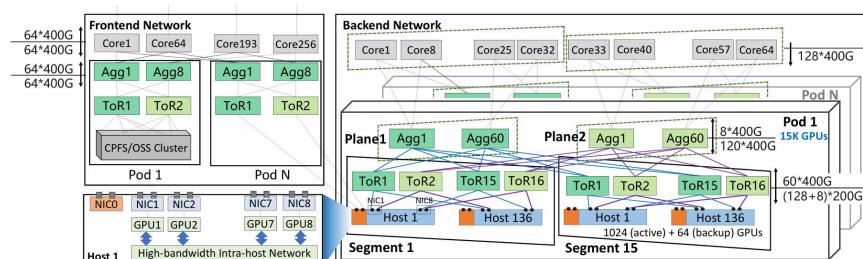
# 大规模ML训练和推理作业使用集体通信在GPU之间交换数据。

- 集合通信的性能至关重要，直接影响训练和推理效率。例如，在训练GPT-22B和LLaMa-7B模型时，通信时间占比超过30%。



预训练 微调 在线推理

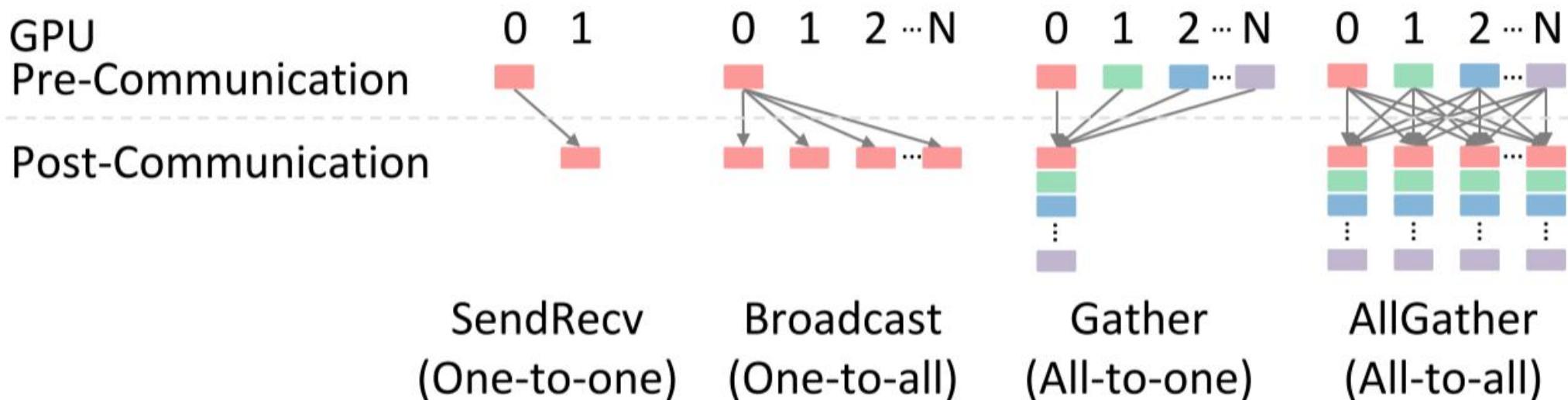
# Inter-GPU Communication



Aliabba HPN架构

# Background

四种的集合通信模式：

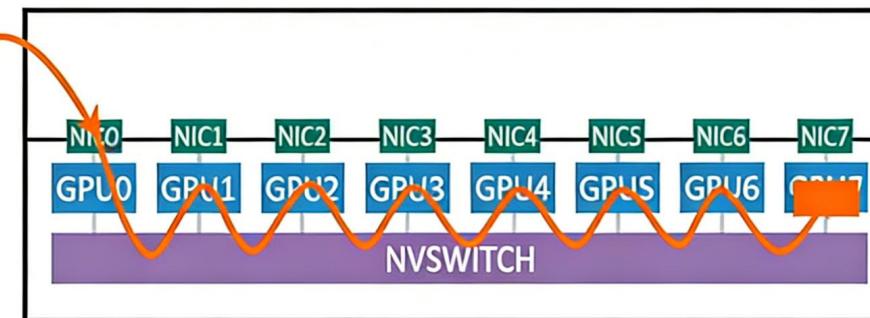
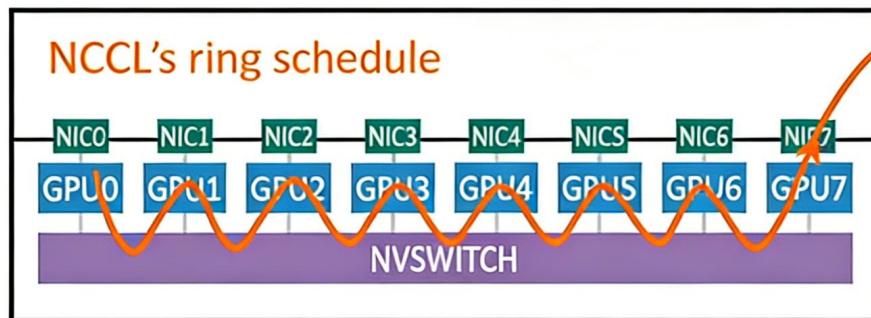


集合通信的性能很大程度上依赖于通信的调度策略。

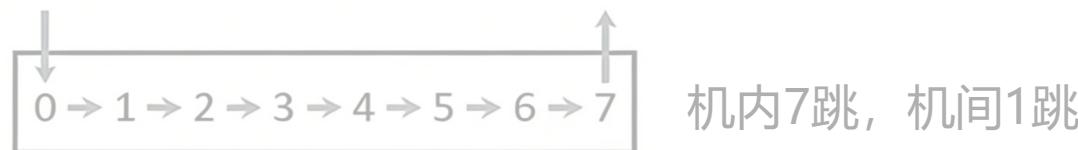
- 调度策略：定义了数据如何在GPU之间传输以满足通信需求。即在时间点  $t$ ，通过链路  $l$ ，发送数据块  $c$ 。

# Background

现有的集合通信库（如NCCL）对所有集合通信和拓扑使用固定的调度方案（如Ring, Double Binary Tree）。



大数据：性能取决于带宽



H800集群: NVlink: NIC=4:1  
存在带宽浪费!

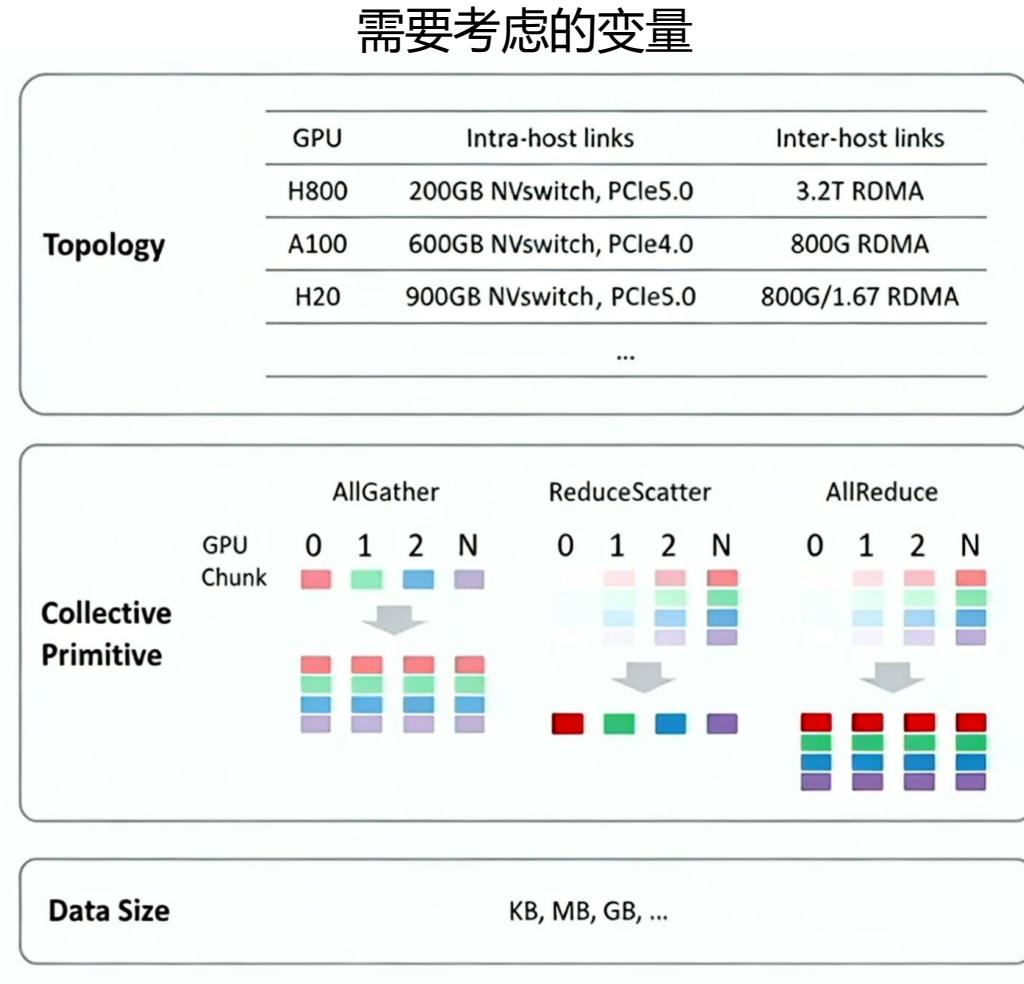
小数据：性能取决于延迟

$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7$        $O(n)$  跳

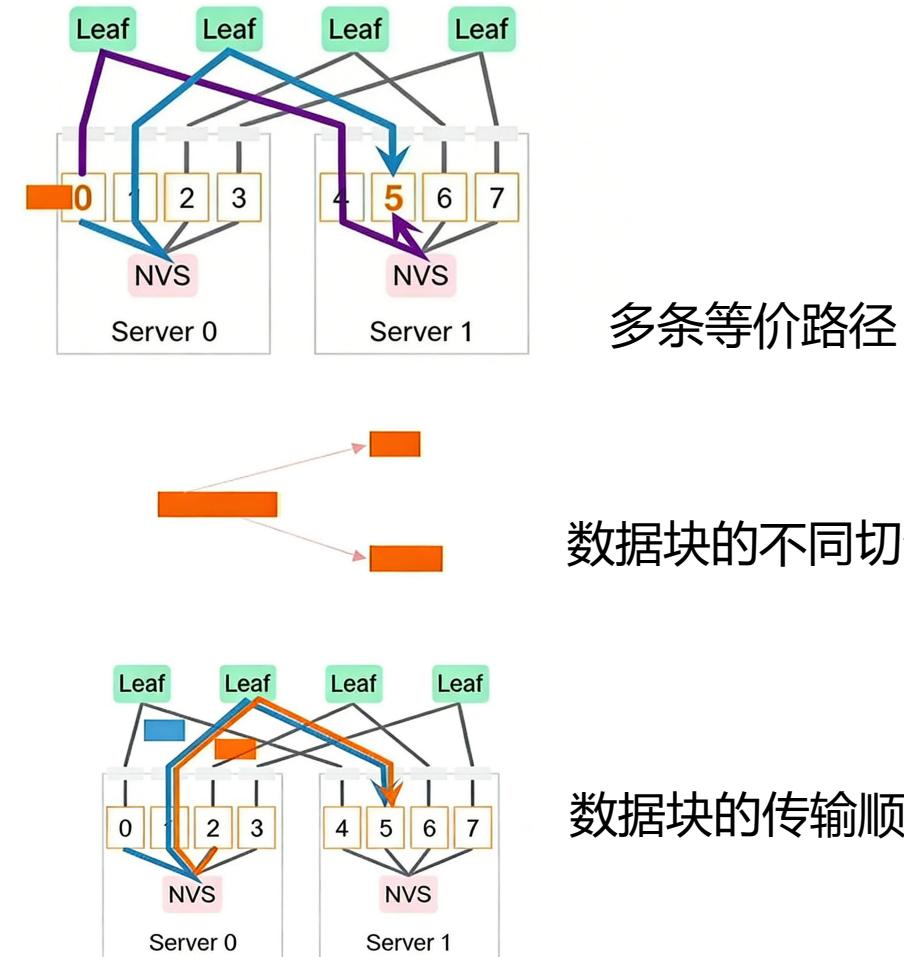


# Motivation

1、手动设计更优的调度策略是不切实际的。



2、自动生成？搜索空间巨大



# Related work

领域	代表工作	关注点	局限性
开源CCLs	NCCL(Nvidia), RCCL(AMD), Gloo	实现通用调度 (Ring/Tree), 运行时微调	不够灵活, 无法适应多样化的网 络架构
特定场景优化	Blink(MLSys'20), Themis(ISCA'22)	针对特定拓扑或集合通信优化	缺乏通用性
调度合成器	SCCL(PPoPP'21), TACCL(NSDI'23), TECCL(SIGCOMM'24)	为集合通信需求自动合成调度	在大规模网络下搜索空间爆炸

**SyCCL：也是一种合成器，但通过利用对称性缩小了搜索空间。**

# Related work

**最先进的合成器:** TACCL, TECCL将调度策略合成建模为混合整数线性规划(MILP)问题, 将整个通信需求和拓扑编码到一个整体公式中。



## 问题:

- 扩展性极差 (耗时数小时-数天)
- 变量随GPU数量指数级增长, 在大型GPU集群中无法运行

# Motivation

## SyCCL：快速生成近似最优调度策略

- 将调度策略生成建模为混合整数线性规划（MILP）问题
- 利用对称性简化搜索空间

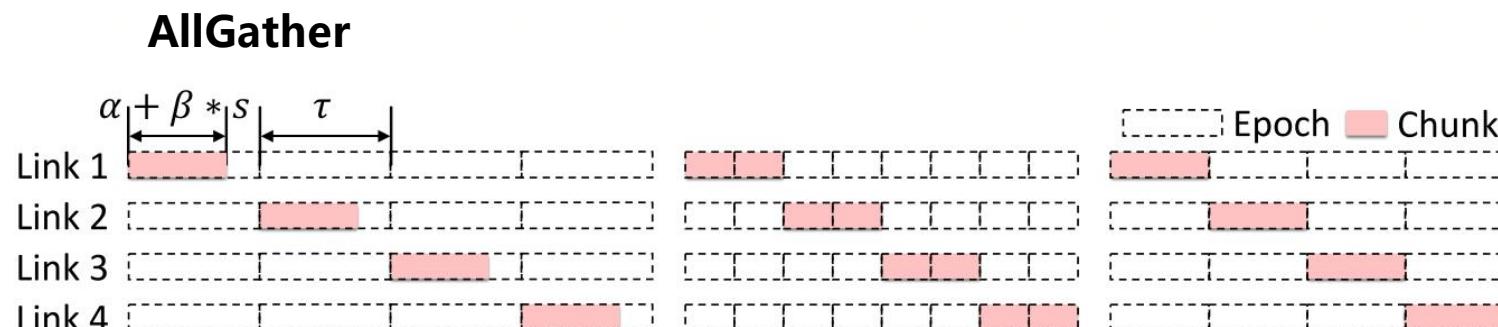
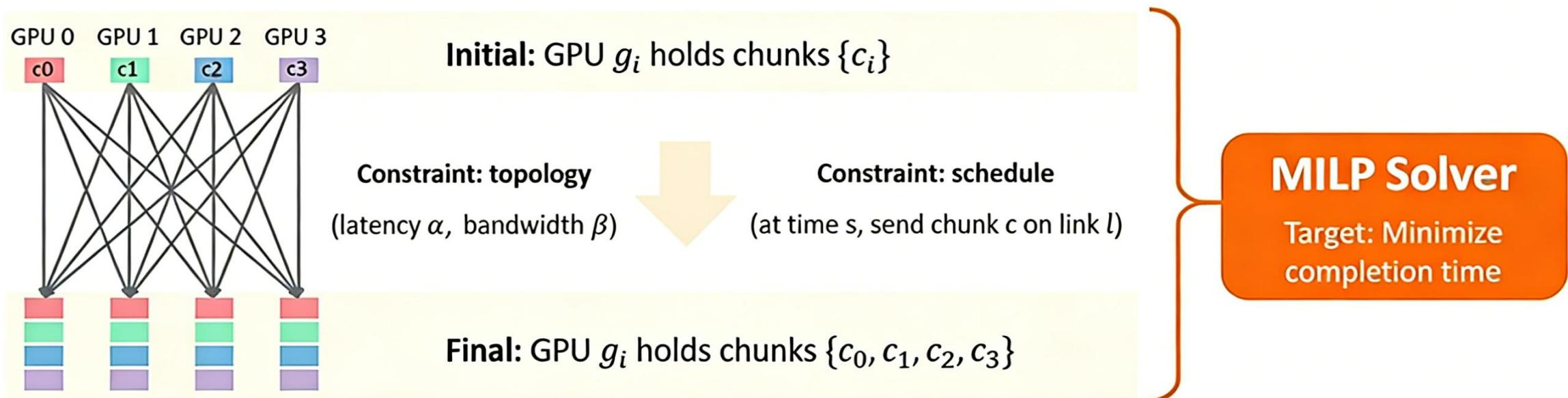
# Overview

## SyCCL: 快速生成近似最优调度策略

- 将调度策略生成建模为混合整数线性规划 (MILP) 问题
- 利用对称性简化搜索空间

# Overview

核心：将状态与事件表示为变量，并将所有要素转化为约束条件



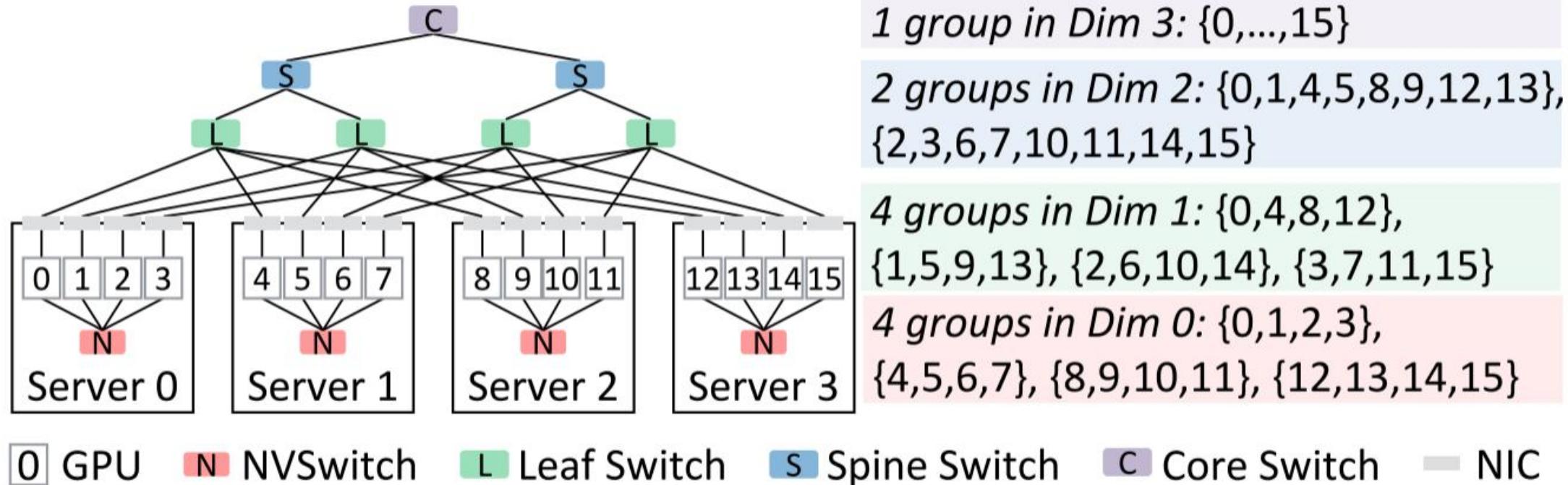
将时间划分为一个个固定的格子（Epoch），看作“装箱问题”。  
 $\Sigma$ (第 $t$ 个epoch中在传的数据量)  $\leq$  链路带宽

# Overview

## SyCCL：快速生成近似最优调度策略

- 将调度策略生成建模为混合整数线性规划 (MILP) 问题
- 利用拓扑、集合通信操作的对称性简化搜索空间

# Insight



## Observation 1: 现代GPU集群拓扑具有高度对称性。

- 将拓扑按维度(Dim)分解，将GPU划分为分组(Group)。在每个Dim下，各个Group结构完全一致（称为同构的GPU组）且相互隔离（Group间通信需要跨Dim）

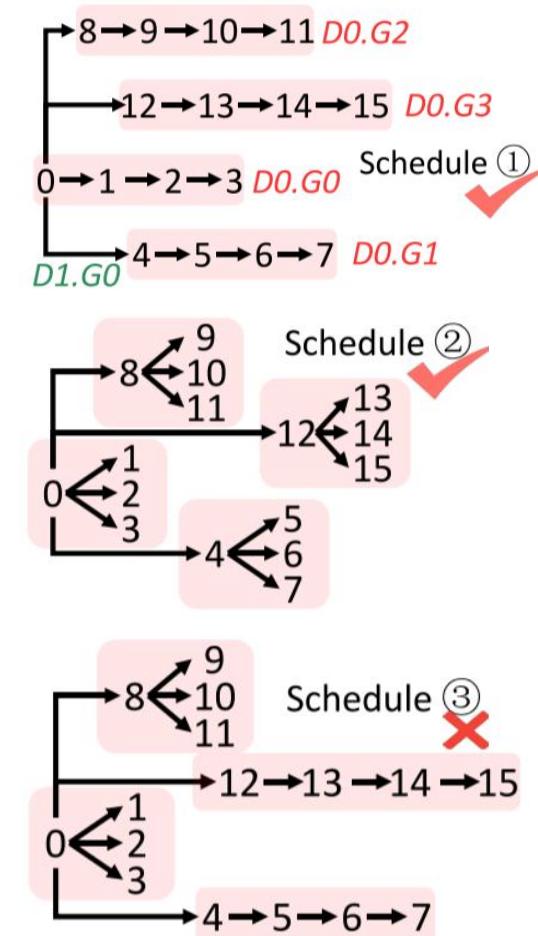
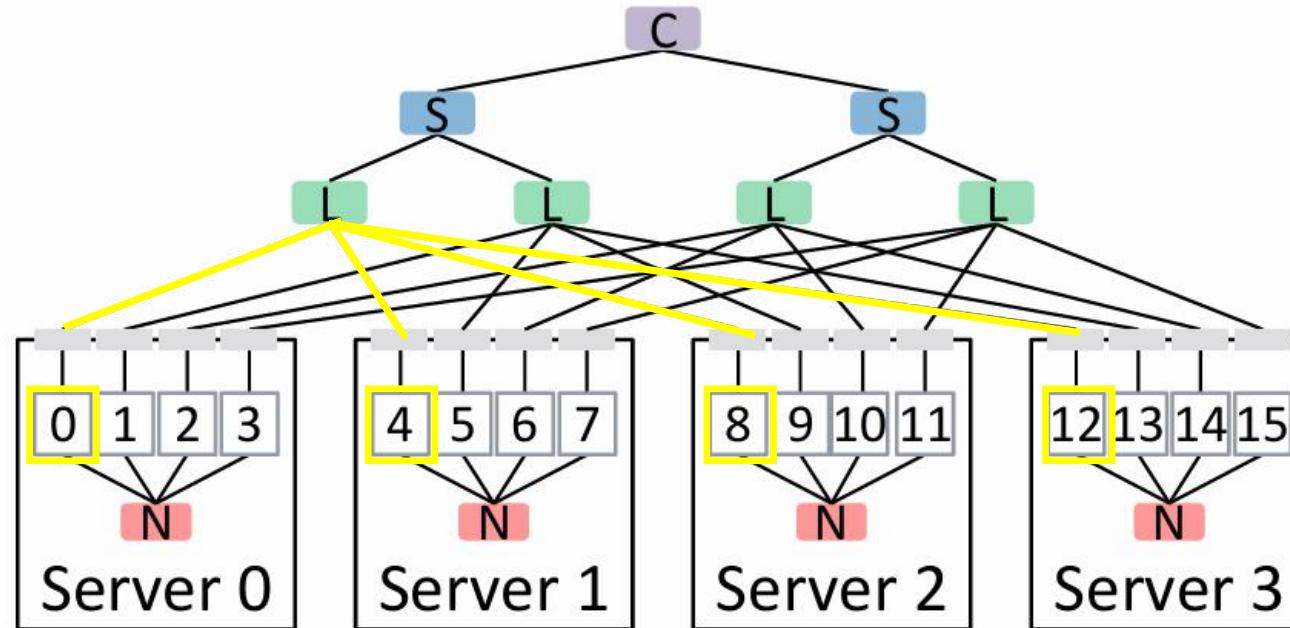
## Observation 2: 集合通信具有对称性。

- All-to-all通信可分解为同构的One-to-all通信

# Insight

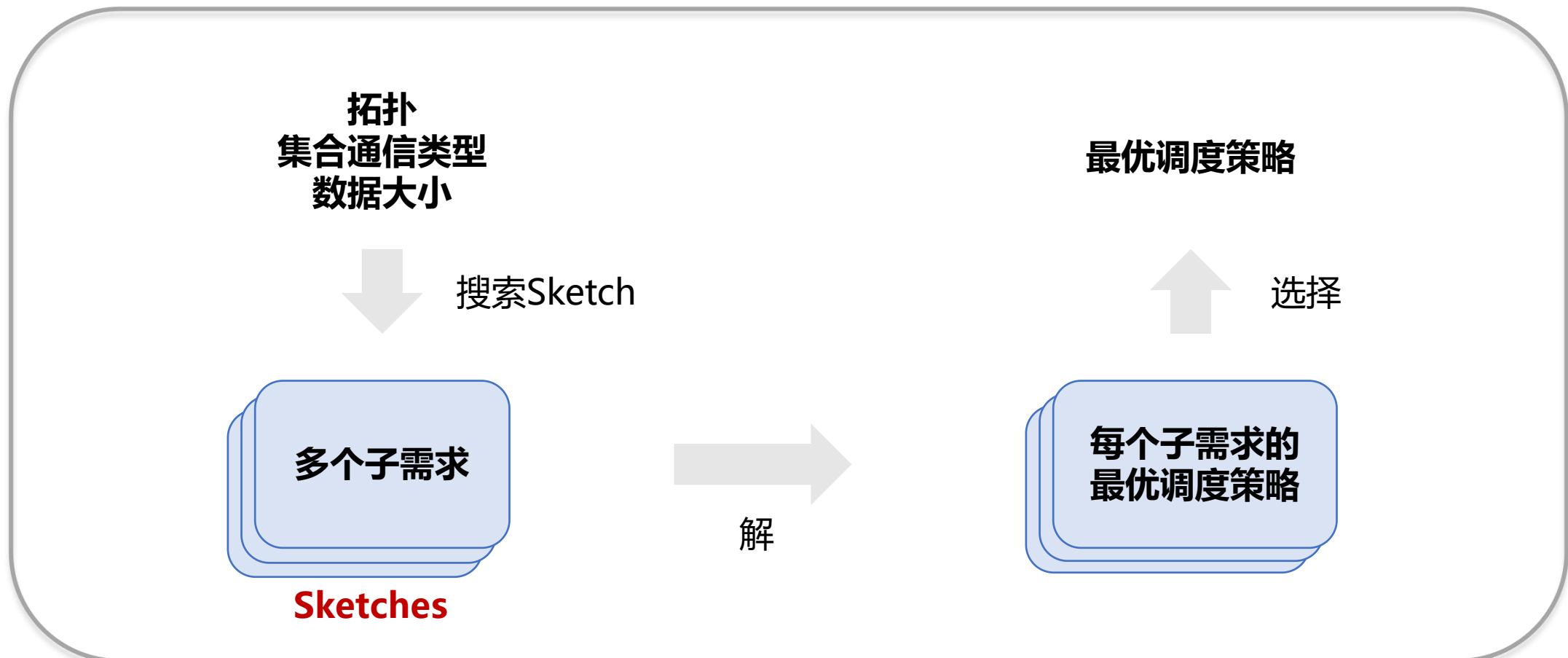
## Insight: 在每个Dim下，同构的Group内调度策略应保持统一

- 如果同构的组采用不一致的调度，会导致负载不均衡，从而变为次优解。
- e.g. GPU0向其余GPU做Broadcast通信



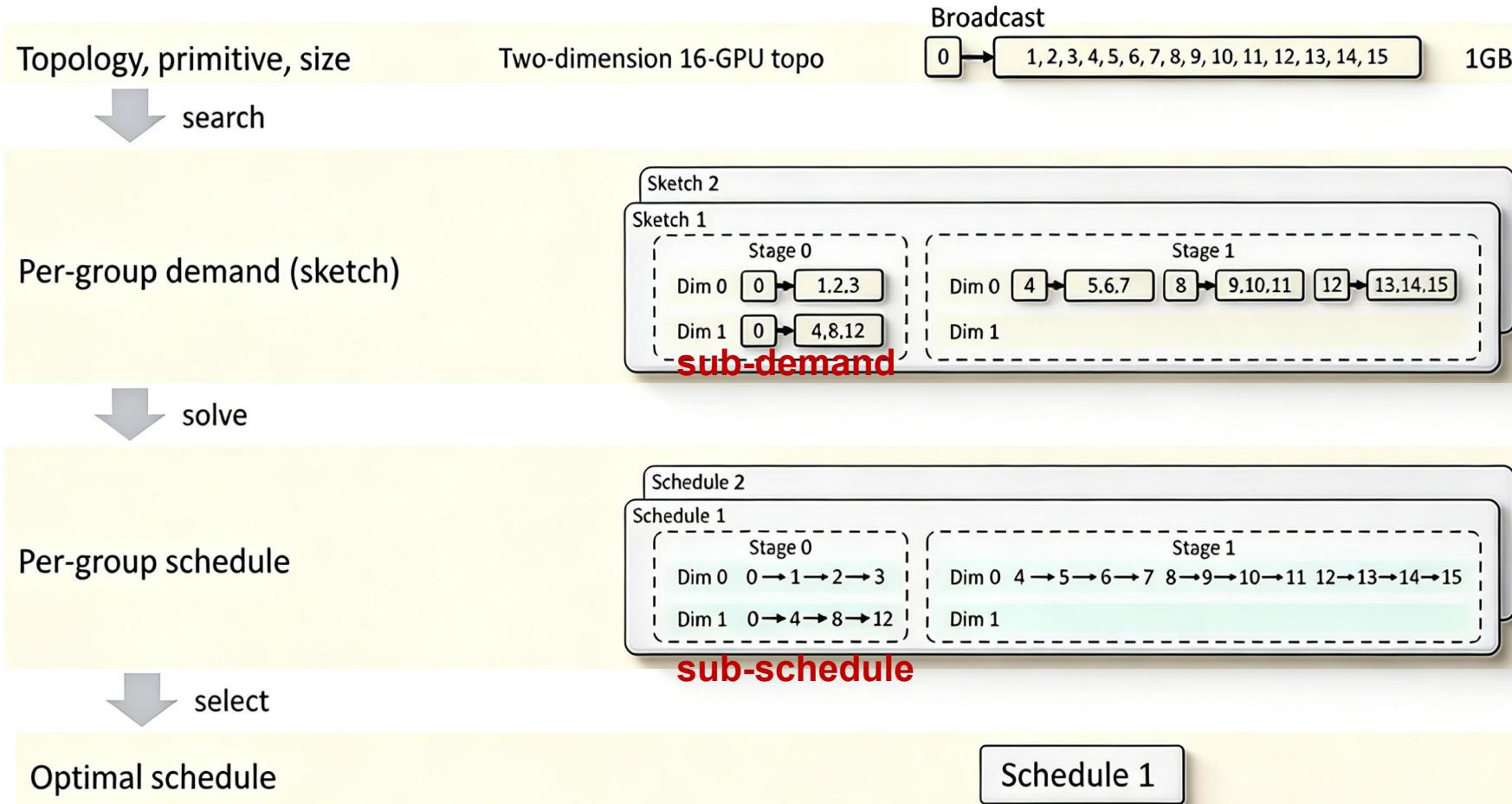
# Overview: SyCCL workflow

**Sketch:** 将集合通信需求划分为不同维度和时间阶段的较小的子需求 (谁发给谁)。



# Overview: SyCCL workflow

**Sketch:** 将集合通信需求划分为不同维度和时间阶段的较小的子需求 (谁发给谁)。



# Design1: Sketch Exploration

## SyCCL workflow:



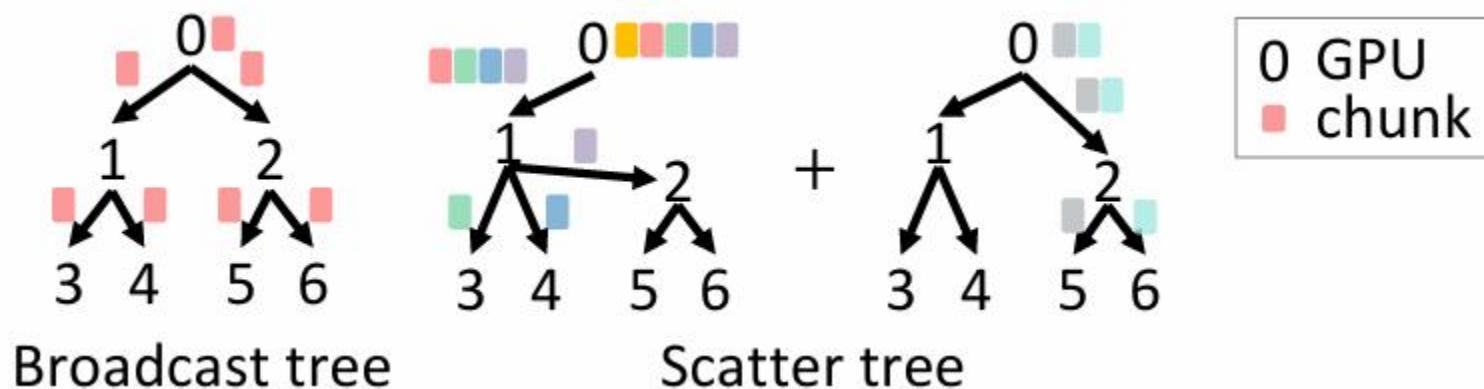
# Design1: Sketch Exploration

**适用范围:** 首先关注One-to-all(Broadcast、 Scatter)。

**树状结构:** Broadcast和Scatter的调度都可以表示为树的集合。因为避免防止浪费带宽，每个GPU (除了根) 应该只接收一次数据，即有且仅有一个父节点。

**基于枚举的搜索:** 将通行需求分解为k个阶段，每个阶段从维度、组、源和目的

**GPU枚举可能的通信子需求。**



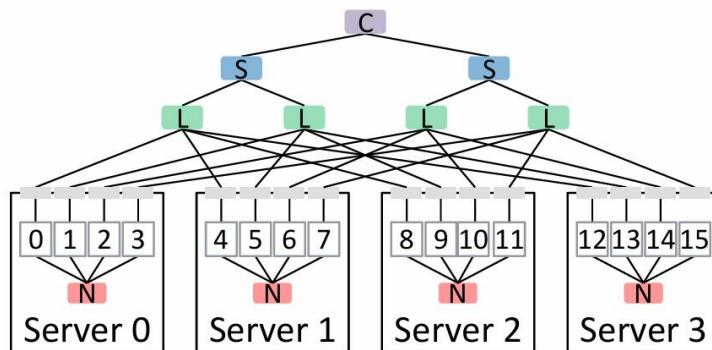
# Design1-1: Pruning Strategies

目的：过滤掉次优或冗余的Sketch。

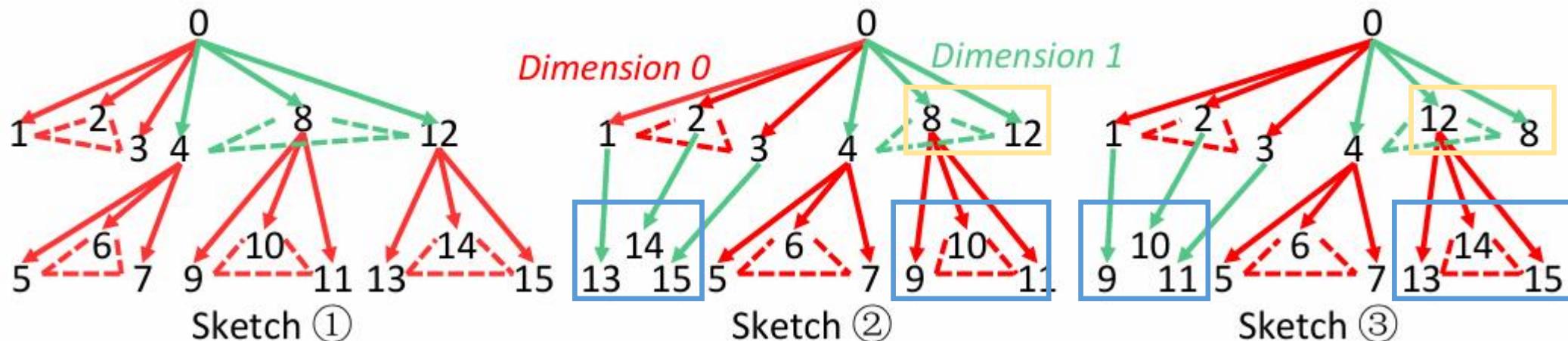
## Pruning 1: 同构检测

- 利用拓扑对称性，过滤掉同构的Sketch（只是交换了地位平等的GPU ID）。

拓扑：



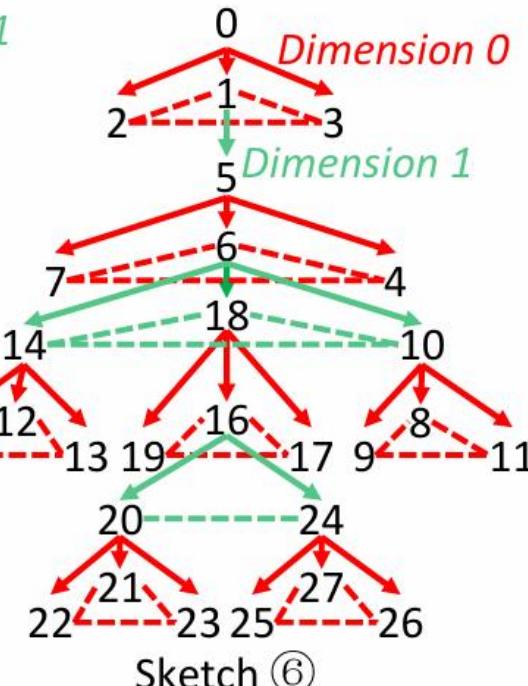
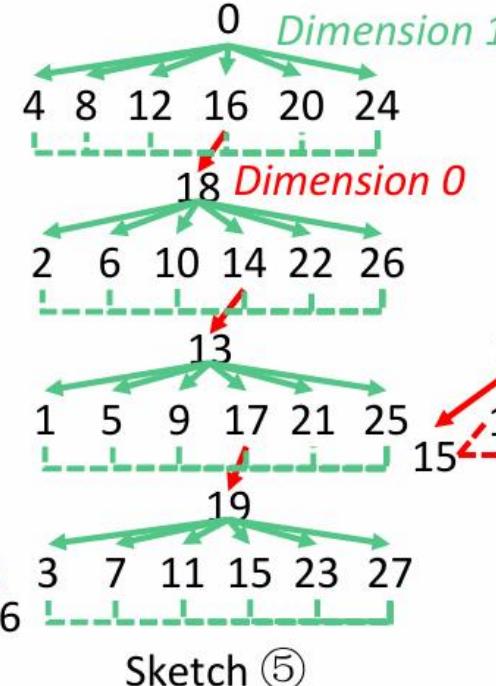
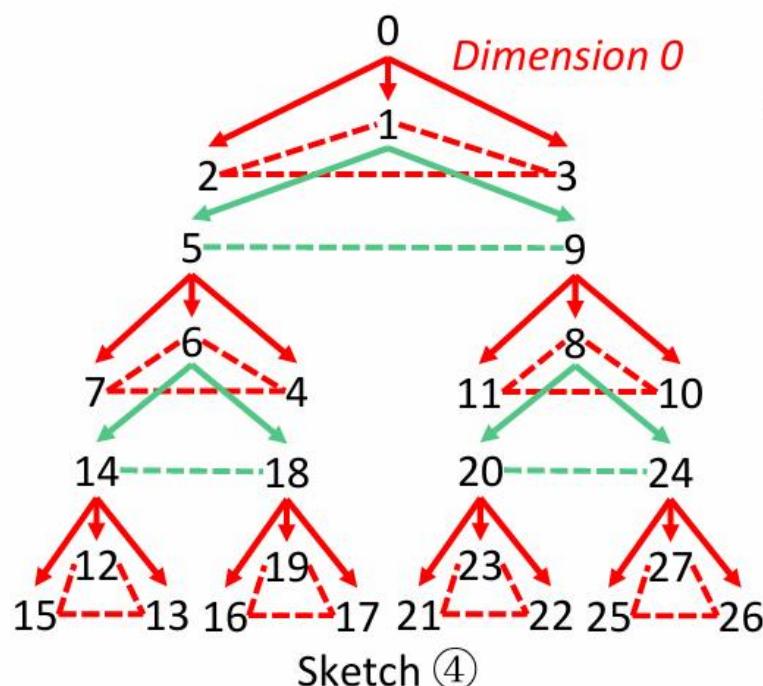
集合通信：GPU0向其他GPU广播



# Design1-1: Pruning Strategies

## Pruning 2: 强制一致性

- 基于Insight: 每个Dim下, 同构的Group内调度策略应保持统一。检查子需求中接收端与发送端GPU数量的比率, 若不一致则剪枝。



Per-stage ratios: 3,2,3,2,3

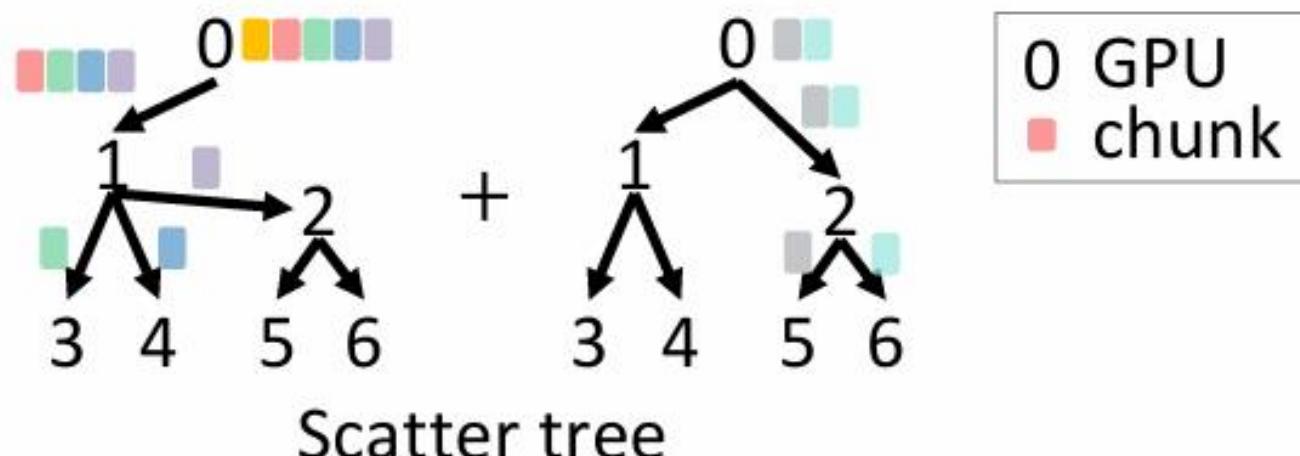
6,1,6,1,6,1,6

3,1,3,3,3,2,3

# Design1-1: Pruning Strategies

## Pruning 3: 限制中继

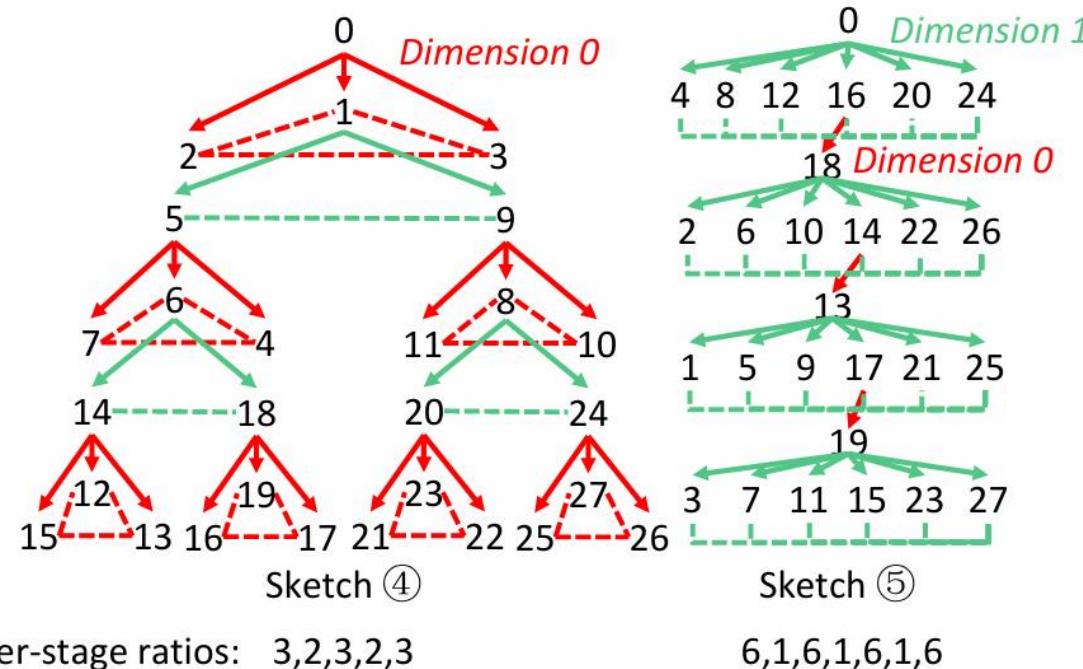
- 在Scatter操作中，如果一个中间节点有n个子节点，它不仅要接收自己的那份数据，还必须接收并转发n份额外的数据给子节点。如果树的层级越深，上游节点的通信过载就越严重，导致带宽瓶颈。
- 限制最大跳数不超过拓扑的维度数，**防止Scatter树过深导致负载过重。**



# Design1-2: Generating Sketch Combinations

问题: 单个Sketch通常只利用了部分带宽, 且可能导致负载不均衡。

- **组间不均衡:** 同一维度下的不同组工作负载不同。
- **维度间不均衡:** 不同维度的负载与其实际带宽能力不匹配。

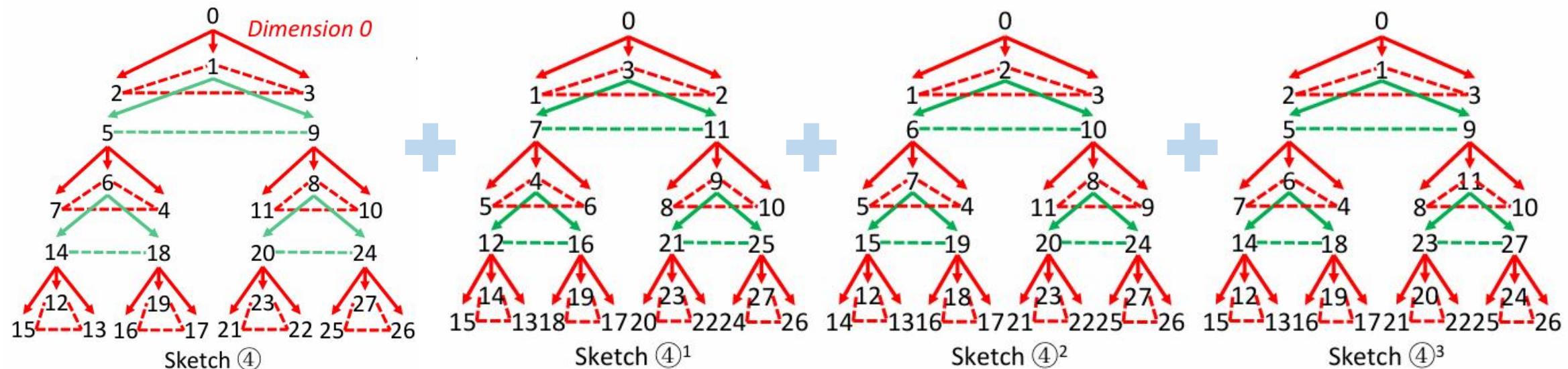


# Design1-2: Generating Sketch Combinations

问题：单个Sketch通常只利用了部分带宽，且可能导致负载不均衡。

- 组间不均衡：同一维度下的不同组工作负载不同。
- 维度间不均衡：不同维度的负载与其实际带宽能力不匹配。

解决方案：将数据块切分，通过不同的路径（sketches）传输，以最大化带宽利用率。具体来说，**先复制sketch再整合**。



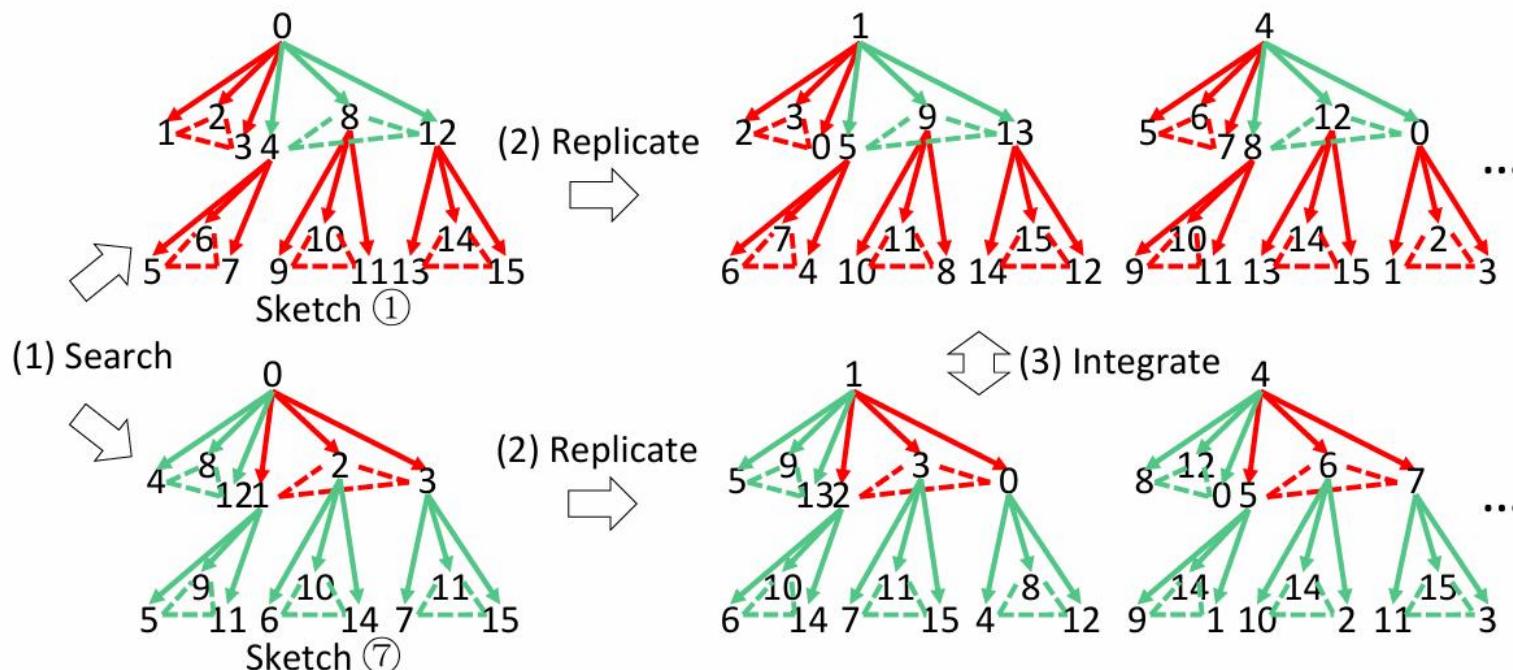
$$C_4 = \{\langle ④, 0.25 \rangle, \langle ④^1, 0.25 \rangle, \langle ④^2, 0.25 \rangle, \langle ④^3, 0.25 \rangle\}$$

# Design1-3: Extending to All-to-All

问题分解：N-GPU下，All-to-all看作N个同构的One-to-all。

生成方法：

- 1、先关注GPU0，**搜索**分解后的One-to-all集合通信的sketch。
- 2、利用拓扑对称性，将这些sketch**复制**并映射到其他N-1个源GPU上。
- 3、为了让不同维度的带宽充分利用，进行sketch**组合**生成最终的N-sketch组合。



# Design2: Schedule Synthesis

## SyCCL workflow:



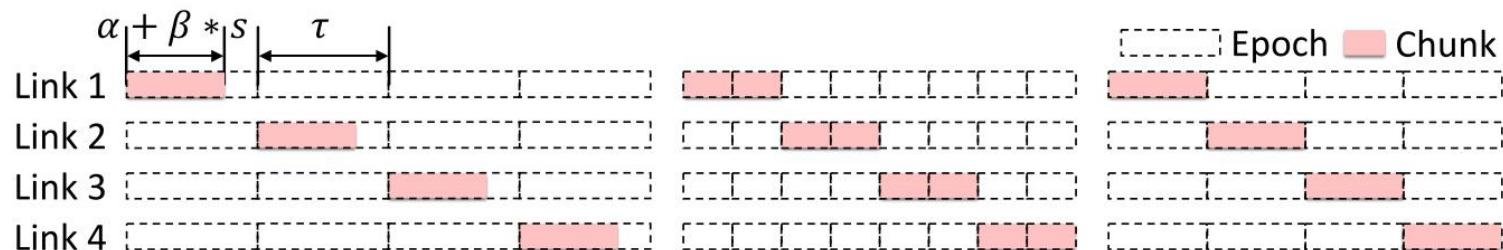
# Design2: Schedule Synthesis

Input: Sketch组合  
(多个子需求, 谁发给谁)

MILP求解每个  
子需求

Output: 每个子需  
求的最优调度策略

建模：采用与TECCL类似的时延-带宽模型。将时间划分为Epochs。



优势：SyCCL只针对小拓扑子集中的具体传输任务（子需求）进行求解，而非整个大拓扑，极大降低了问题规模。

# Design2: Schedule Synthesis

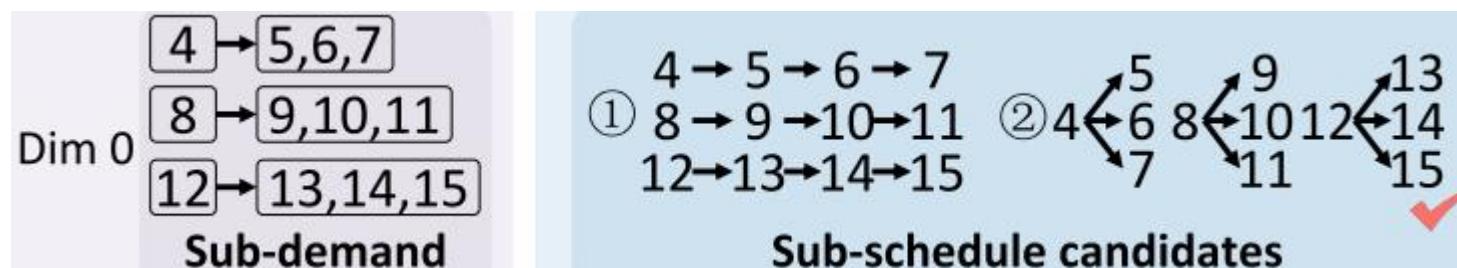
**挑战:** MILP Solver需要在精度和效率之间权衡。

## 优化 1: 两步合成

- (1) **粗粒度:** 使用较大的Epoch快速筛选出Top-K候选项。
- (2) **细粒度:** 对优选出的候选项使用较小的Epoch进行精确求解。
- 引入辅助变量  $E$  自动确定合适的 Epoch。

## 优化 2: 同构与并行

- 同构类: 将同构的子需求归类, 只需解一次, 映射结果。
- 并行计算: 独立的子需求多线程并行求解。



# Design3: Schedule Evaluation

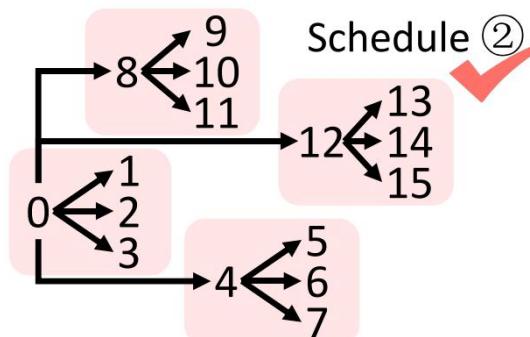
## SyCCL workflow:



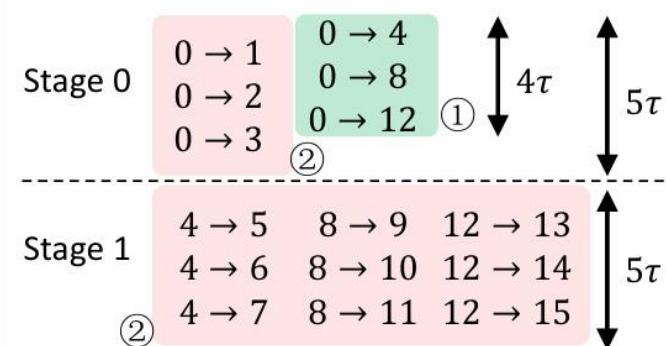
# Design3: Schedule Evaluation

**整合：**将子调度合并为完整的调度。但是简单的阶段时间相加是不准确的，因为不同阶段的通信可能流水线并行。

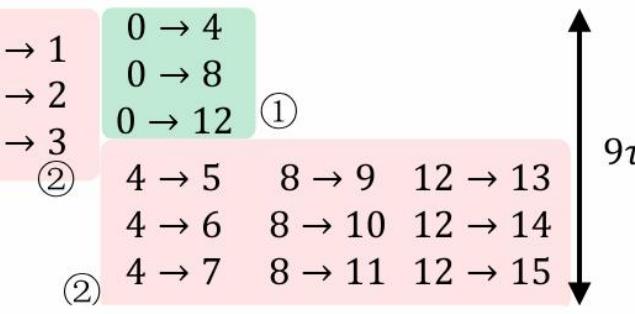
**调度模拟器：**基于ASTRA-sim开发。按时间顺序扫描通信事件，精确计算完成时间（时间复杂度低 $O(E)$ ， $E$ 是通信事件的数量）。最后评估所有候选选项，选择性能最好的调度策略。



GPU0做广播



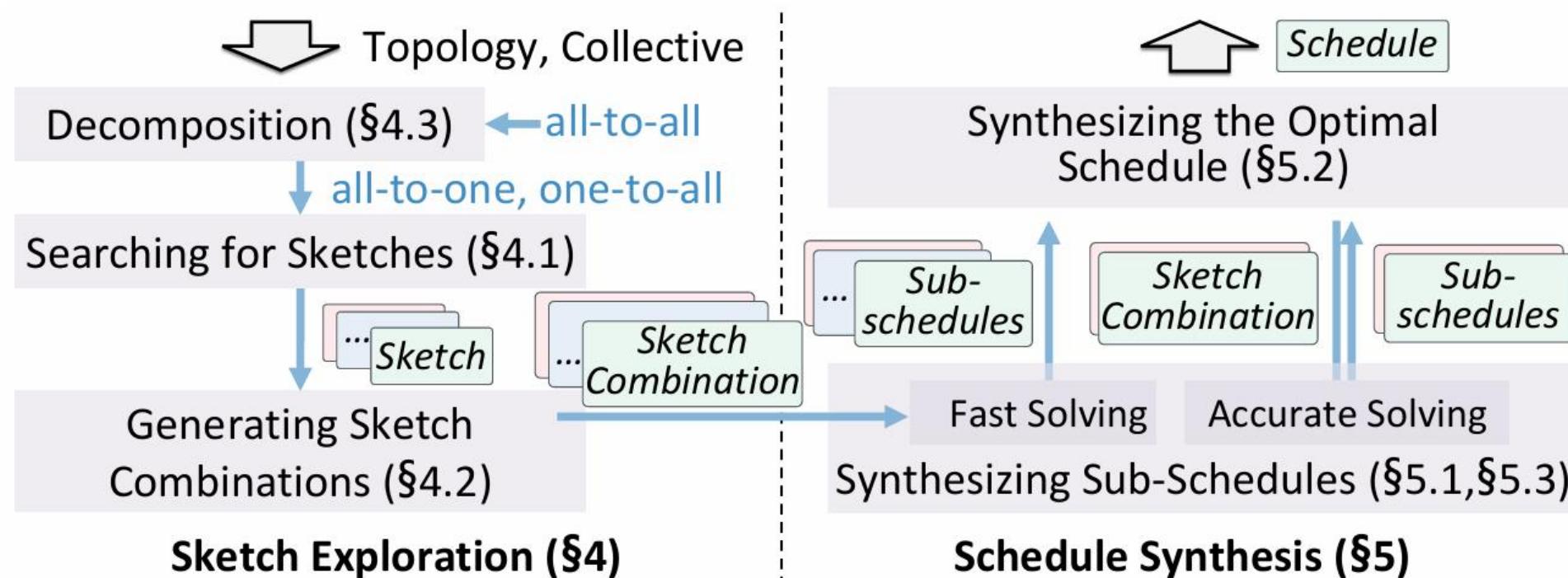
(a) A naive evaluation method



(b) SyCCL

# Implementation

- **Network Profiler:** 测量链路参数带宽和延迟。
- **Schedule Synthesizer:** 7000行C++代码, Sketch搜索、MILP合成、仿真。
- **Executor:** 将合成的调度转换为XML文件。通过MSCCL-executor在GPU上直接执行, 无需修改CUDA内核。



# Evaluation

## 拓扑:

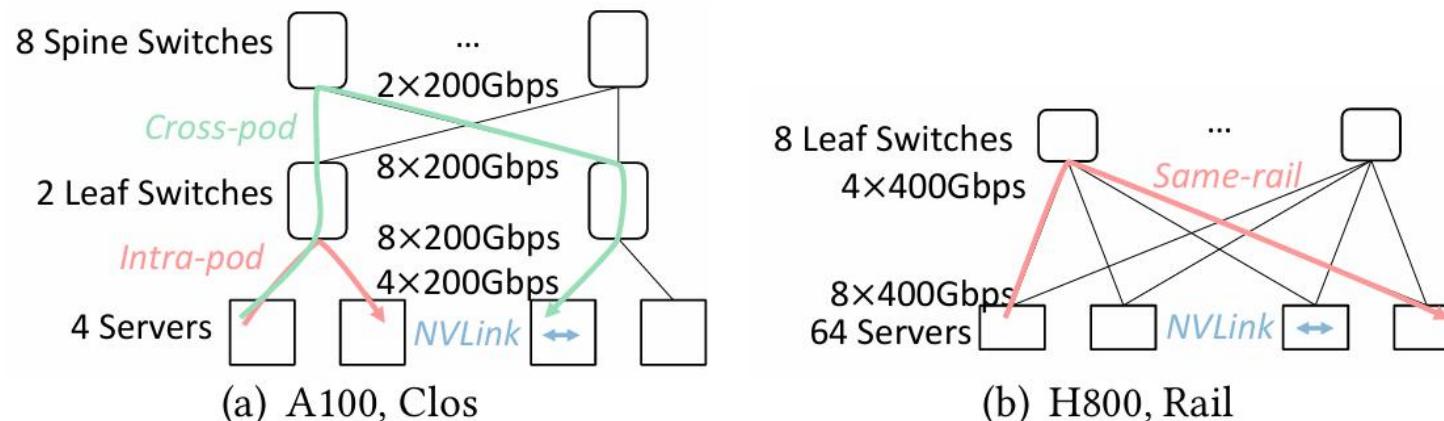
- A100 Cluster(实测): 4台服务器, 每台8x A800 GPU, 两层Clos网络。
- H800 Cluster(仿真): 64台服务器, 每台8x H800 GPU, Multi-rail网络。

## 对比基线:

- NCCL: 业界标准, 使用固定调度 (Ring/Tree)。
- TECCL: SOTA合成器, 基于MILP求解整个拓扑。

**指标:** Bus Bandwidth (busbw), 反映整体带宽的利用率。

**调度策略合成:** 192-core CPU, 1TB Memory.



# Evaluation: Schedule Performance - A100 (AllGather)

## 小数据 (<1MB):

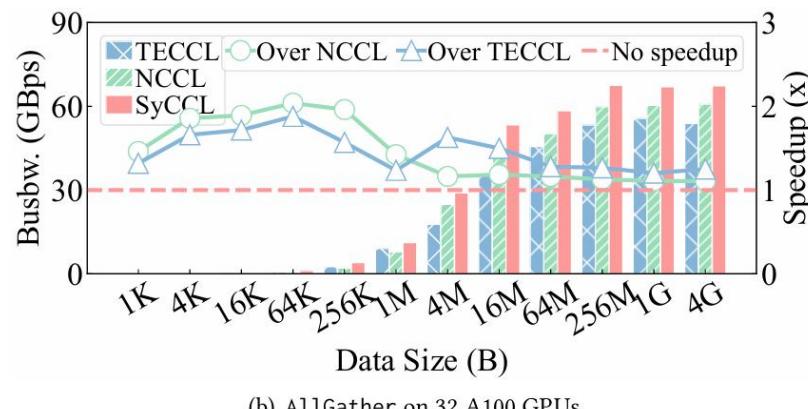
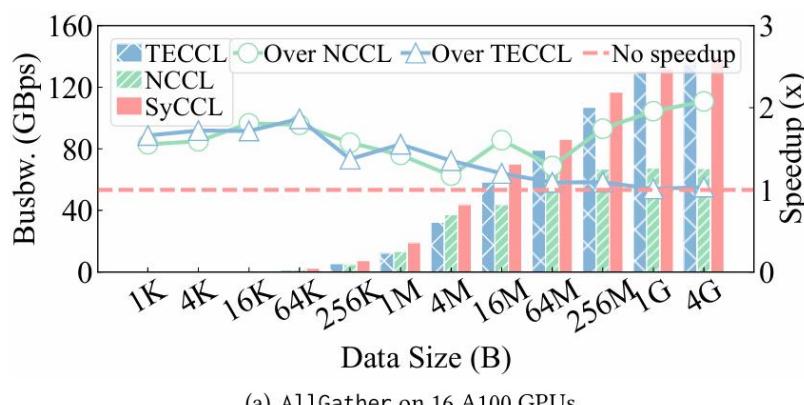
- SyCCL优于NCCL (0.43x-0.81x) 和TECCL。
- 原因： NCCL固定调度跳数多 (Ring需N-1跳) , 受链路延迟 ( $\alpha$ ) 影响大； SyCCL生成的树状调度跳数少。

## 大数据 (>1MB):

- SyCCL相比NCCL提升0.17x-1.08x。
- 原因： NCCL的Ring调度固定了NVLink与Network带宽比为7:1，导致Network带宽瓶颈。 SyCCL能够利用14:1的带宽比，减少网络带宽消耗。

## 扩展性 (16 vs 32 GPUs):

- 随着规模增加，TECCL因模型过大牺牲精度导致性能下降，SyCCL仍能保持优势。



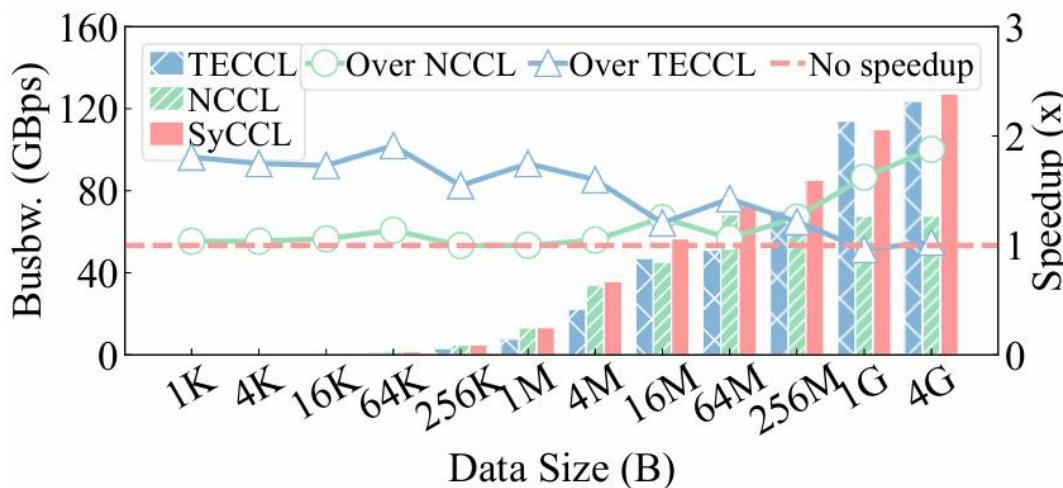
# Evaluation: Schedule Performance - A100 (ReduceScatter & Alltoall)

## ReduceScatter:

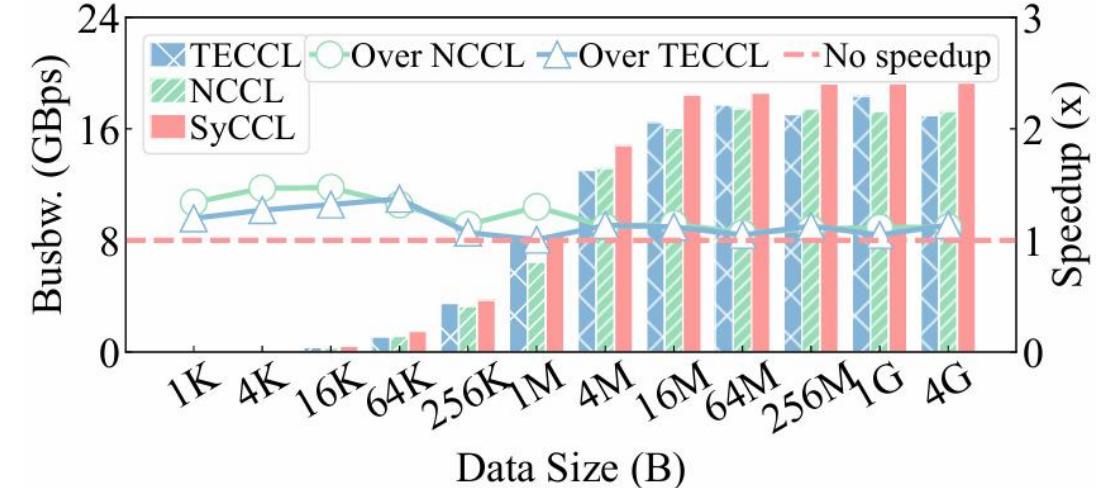
- SyCCL 仍优于 NCCL，但增益略小于 AllGather。
- 原因：MSCCL 运行时未像 NCCL 那样针对特定调度进行深度优化（如流水线重叠），限制了部分性能释放。

## Alltoall:

- SyCCL 相比 NCCL 提升高达 0.71x。
- 性能与 TECCL 相似，因为 Alltoall 分解为大量点对点传输，调度重排空间有限。



(c) ReduceScatter on 16 A100 GPUs



(d) AlltoAll on 16 A100 GPUs

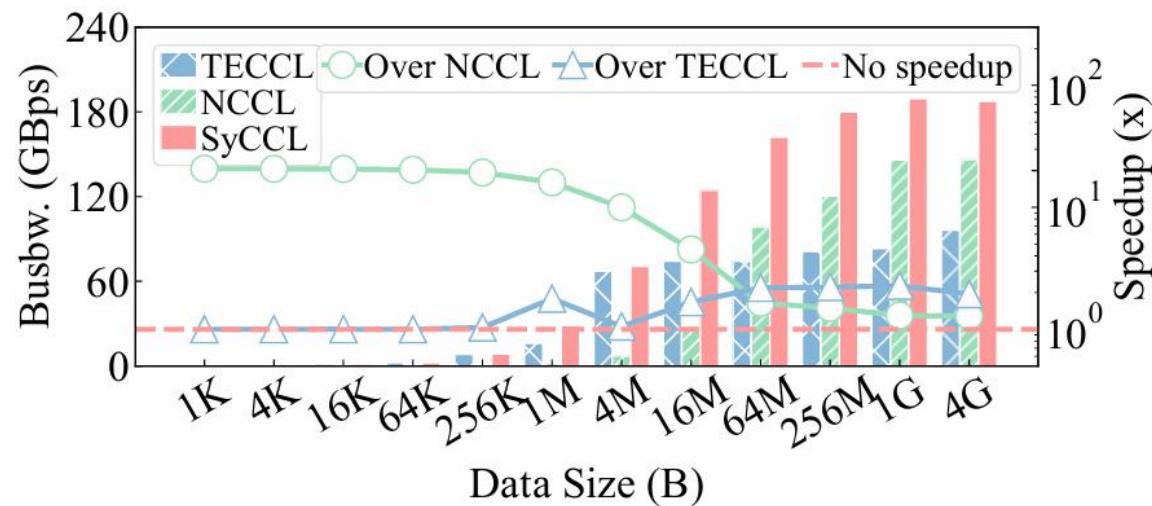
# Evaluation: Schedule Performance - H800 Large Scale

## 64 GPUs (AllGather):

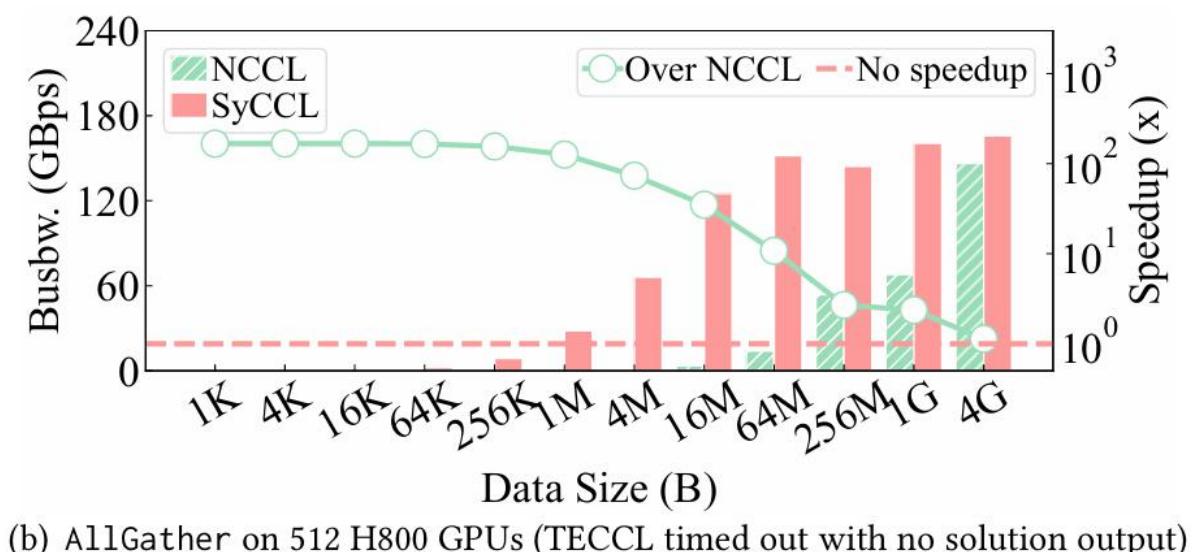
- SyCCL相比TECCL提升高达1.27x。
- SyCCL利用Sketch组合更有效地利用了带宽。

## 512 GPUs (AllGather):

- TECCL: 超时，无法生成结果。
- NCCL: 性能受限，因512个GPU的Ring调度需要511跳，延迟极高。
- SyCCL: 成功合成两维调度（先NVLink广播，再Rail间广播），显著提升性能。



(a) AllGather on 64 H800 GPUs

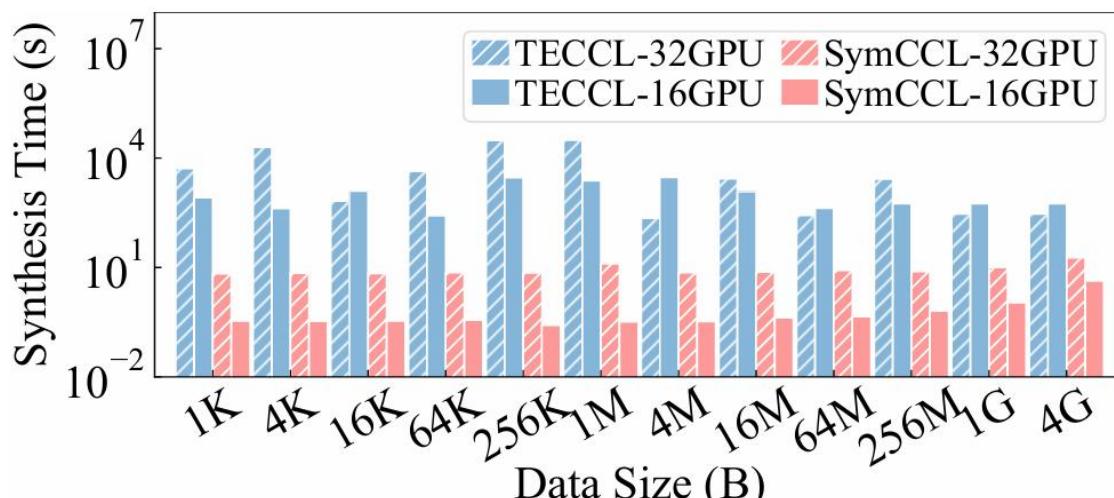


(b) AllGather on 512 H800 GPUs (TECCL timed out with no solution output)

# Evaluation: Synthesis Efficiency (Time)

## 对比结果:

- 16 GPUs: SyCCL (0.3s-2.6s) vs TECCL (4.4min-49.5min)。
- 32 GPUs: SyCCL (6.2s-19.1s) vs TECCL (3.8min-8.7h)。
- 在64 H800 AllGather场景下，加速比达到17286x。
- 总体加速：SyCCL将合成时间缩短了2到4个数量级。



(a) Synthesis time of SyCCL vs. TECCL

Scenario	Min/Max/Mean Time (s)		Speedup
	TECCL	SyCCL	
16 A100, AG	266/2972/1193	0.3/4.3/0.8	1554×
16 A100, A2A	1042/26206/15759	1.4/8.3/3.6	4321×
32 A100, AG	226/31293/8200.2	6.7/18.6/9.0	915×
64 H800, AG	1225/67615/28200	0.4/10.8/1.6	17286×
64 H800, A2A	3698/59044/29371	1.3/29.8/5.7	5135×
512 H800, AG	Time Out	85.5/14146/2246	N/A

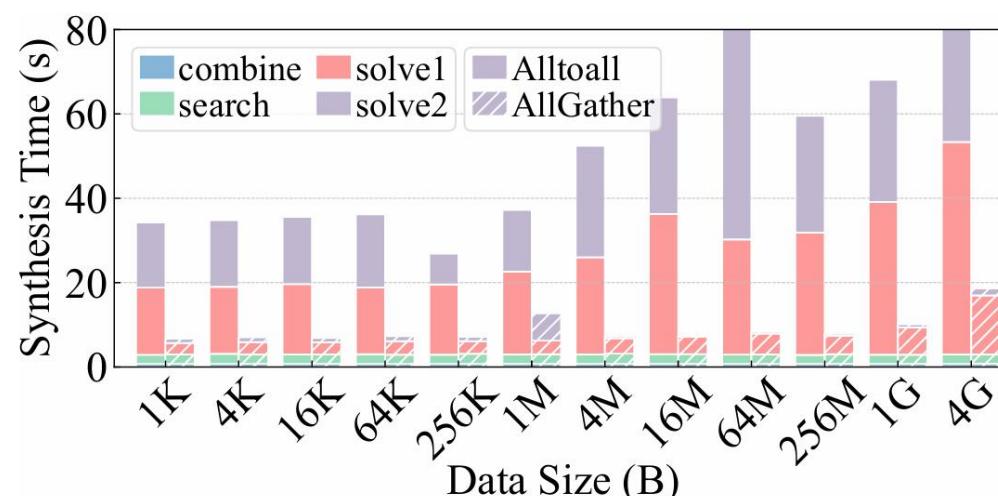
# Evaluation: Time Breakdown & Parallelism

## 时间分解:

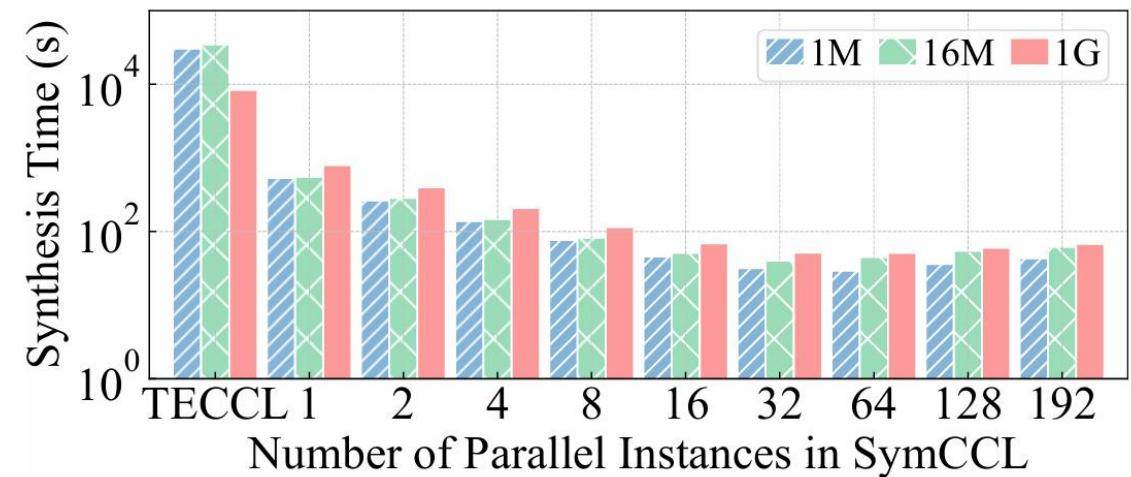
- Sketch搜索与组合的时间是恒定的。
- 主要时间消耗在于Schedule Synthesis (MILP求解) , 且随数据量增加而增加 (需要更复杂的调度) 。

## 并行加速:

- SyCCL利用多线程并行求解独立的子需求, 显著减少合成时间。



(b) Synthesis time breakdown for SyCCL (32GPU)



(c) Synthesis time with different numbers of threads

# Evaluation: Ablation Study - Synthesizing Policy

## Pruning 1 & 2 (去除冗余与一致性):

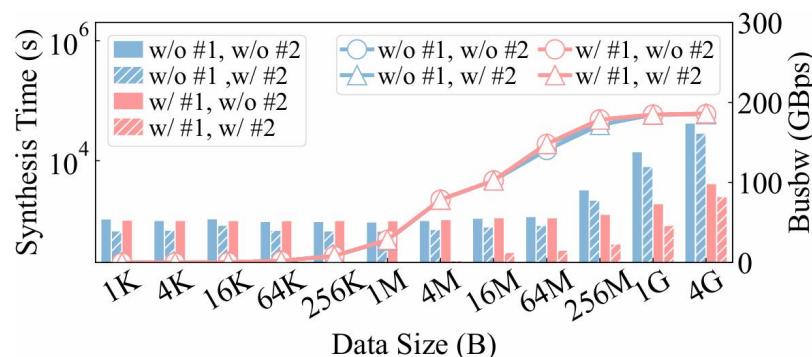
- 减少合成时间20.8% - 48.1%。
- 关键: 对最终调度性能几乎无影响 (证明了Pruning策略的安全性)。

## Pruning 3 (限制Stage数):

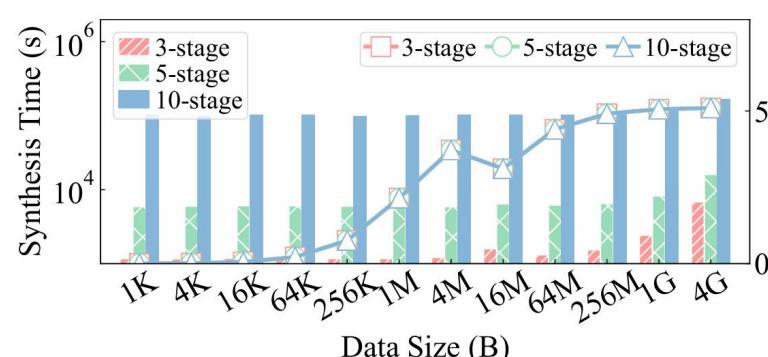
- 将Alltoall Stage限制为  $< 3$  可节省95%-97%的时间, 且不影响性能。

## Epoch Duration ( $E$ ):

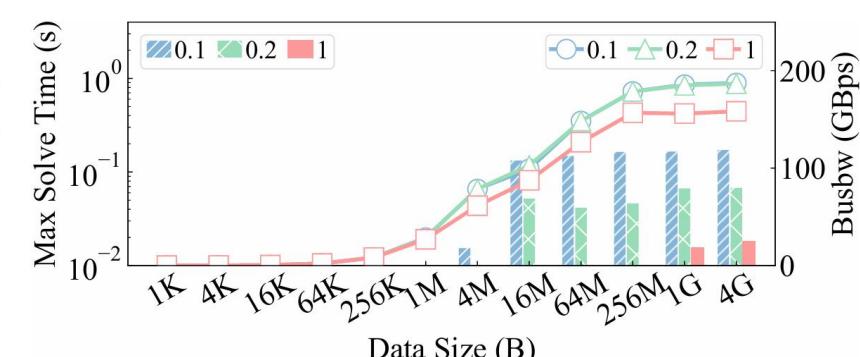
- 验证了  $E$  参数在精度与速度之间的权衡。 $E = 0.2$  是最佳平衡点。



(a) With and without Pruning #1 and #2 in §4.1



(b) With and without Pruning #3 in §4.1



(c) Various  $E_2$  value in §5.1

# Evaluation: End-to-end Training Performance

**场景:** 真实A100集群上的分布式训练。

**模型:** GPT-6.7B 和 Llama3-8B。

**并行策略:** DP和TP。

**结果:**

- 相比NCCL默认配置，SyCCL减少了6.3%的训练迭代时间。
- 相比TECCL减少了4.0%。

Model	NCCL	TECCL	SyCCL	vs NCCL	vs TECCL
GPT3-6.7B, DP16	672.4	653.0	630.0	6.3%	3.5%
GPT3-6.7B, TP16	200.0	197.7	192.5	3.8%	2.6%
GPT3-6.7B, TP32	219.4	216.5	209.7	4.4%	3.1%
Llama3-8B, DP16	1195.4	1153.8	1135.4	5.0%	1.6%
Llama3-8B, TP16	433.9	422.2	412.6	4.9%	2.3%
Llama3-8B, TP32	854.9	887.4	851.5	0.4%	4.0%

**Table 6: End-to-end training iteration time (ms) and speedup with NCCL, TECCL, and SyCCL.**

# Limitations

## 最优化:

- 由于将问题分解且依赖MILP，理论上不保证全局最优。
- 不能完全反映现实的所有方面，例如拥塞控制和抖动。
- 剪枝策略可能丢弃极少数潜在的更优解（但在实践中未发现）。

## 非对称负载:

- 对于MOE等非对称通信，对称性假设失效。

## 动态环境:

- 若网络频繁波动或发生故障导致拓扑非对称，SyCCL有效性降低。

## 异构集群:

- 在高度不规则的异构网络中，对称性假设失效。

# Conclusion

## 背景:

- 集合通信对于大规模ML训练&推理很重要。
- 固定的调度策略往往不是最优的。
- 自动合成调度策略搜索空间巨大。

## 设计:

- 提出sketch概念，利用拓扑和集合通信的对称性进行问题分解。
- 设计了高效的sketch搜索、剪枝、组合算法。
- 基于MILP求解调度策略。

## 结果:

- 相比现有SOTA方法，合成时间减少2-4个数量级。
- 调度性能提升高达127%。

## 一、能不能用到我们的场景？

**问题：作业放置。**在万卡集群中为一个需要1024卡的LLM训练任务寻找最优位置（不仅要有空闲GPU，还要通信距离最短、碎片最少）。例如需要同时满足TP域的高带宽、DP域的低跳数。如果任务的逻辑结构是对称的，物理资源的拓扑是对称的，那么放置策略也应该也是对称的。

**思考：**提取作业的通信模式。例如TP组对带宽要求高，需要紧耦合，DP/PP组可以松耦合，以此提取多个不同的通信模式。然后根据物理对称性，算出一种最优策略进行放置。最后扫描其他类似的作业和对称的子结构进行放置。

## 二、问题可以泛化吗？

**问题：**面向MoE模型。MoE的AlltoAll通信中数据量是动态且偏斜的（存在热点专家）。而SyCCL假设数据量均等，生成的调度在MoE上会负载不均衡。

**思考：**把AlltoAll的通信需求看作一个分布。不再生成一个静态的调度策略，而是生成一组备选sketch。运行时根据token的路由分布（通过简单的轻量级统计），动态选择最适合当前流量偏斜度的sketch。

## 三、能不能进一步提高？

**问题：**SyCCL的时间消耗主要在求解。能否替换掉求解器？

**思考：**将MILP Solver替换为一个基于GNN的强化学习Agent。用GNN编码GPU集群的拓扑结构，用RL Agent学习“在当前Step数据块应该往哪里跳”。应该会比MILP快，并且能适应拓扑的变化。

**问题：**推理系统对latency (首字延迟) 极其敏感，而不仅仅是带宽利用率。

**思考：**修改SyCCL的目标函数。从Minimize completion time (吞吐量导向) 修改为 Minimize first-chunk latency (延迟导向)。



# Q & A