Theseus

Motivation → Limitation → Gaps of SOTA → ideal → Solution

Problem: ① 只能在 init phase 定义、选择 schedule. 背景 evolving runtime condition (workload characteristic, hw health status).

Method: schedule level adapter.

Challenge: (1) Flexible (可变/增改), Informed (观测层) ← 基于一个 cluster-wide runtime.

(2) Schedule Consistency (分布一致, 要求快收敛) ⇒ Dynamic Selection

(3) 资源管理. (GPU 显存)

(4) Seamlessly 切换调度. (低时间性能 overhead) ⇒ Hot Swapping

Schedule 特征: ① 4层 schedule 分类: Global (Cluster) (communicator) → GPU → TB (buffer) → OP

② 资源: Scratch Buffer, Channels (MSCCL++核心抽象, RDMA资源), Semaphore.

PortChannel ↑ — QP ~ MR.

③ 4类 OP: (1) Local Op (2) Remote Op (3) Sync (4) Fusion.

4个问题场景: ① Shifting Load (MoE traing All to All) ② Fail-slow ↔ OPTCC

③ Slow Synthesis: Scalability问题: 大集群求解慢太大 TECCL VS NCCL

Comm Demands change frequently. MILP 32gpu α-β model tuning 13h

④ Suboptimal: 即使用 α-β 估测 → 仍局限于 off-the-shelf 估测而非实际 → 有偏差效果.

接口: endo attr NewAttr(·)

exo attr SetExo(·)

New Schedule NewSched(·) 要能支持 { usability: 用 0-1 快速 Filter 一些不可用方案 score

replacement → 轻量级更换 → 可试错/逐步进步

Persistent: Replicate Log

Context Consistency: ① Strawman 方法: { Global Barrier: overhead 太大了 Paxos / Raft: 多 stage 来收敛 Centralized: (1) hot spot (2) per rank tracking progress

② 利用 AllReduce 做并集 "∩" 以及 min 版本 → 因此确实需要 log 来 persist 所有版本

要求只达成 Agreement, 不要求最新版, 且是 Read Only

同时仍采用 RSM 对称实现 → 没有相对热点

Materialization: JSON() → 建连 + 发包 + 计算.

{ 可以接受 Fail-Slow.

不可接受 Fail-Stop → 要额外支持

资源树

L0 Communicator — { Scratch Buffer: 例如存中间结果用于 Reduce 的 → 不同 schedule 可以共享之.

L1 Schedule — FIFO queue: 指 NCCL 中 host 侧 proxy / progress 线程

— sync primitive (semaphore): 同步原语对调度中 "谁来解锁" 完全且强烈依赖

L2 Buffer — Buffer handle → 更换连接

Comm Channel → QP 的 ~~send~~ SQ / RQ. × 主要指远端节点 Buffer, 如 next → buf.

L3. Range — Low level Op → 数据分段如 0~1MiB, 1~2MiB.

Delta Migration：假设（无验证）新旧 schedule 之间很相似，有资源可以复用.

Lazy Migration：对于 RecTree 采用 只改 path ⊕ tag 的方式，不影响 子树其余分支.
- 范围小很多 → efficiency
- tag：低层级要知道上层换了而去更 → correctness.
  eg. buffer handle → sched 换了，peer 已不同 → rem buf 要更新

Tree Maintainence：实际遍历 tree ⊕ 复用资源.

(1) Hit：资源都有，按节点映射即可.

(2) Miss：从缺失处新建分支.

(3) tag 过期：迁移资源，更新 tag（Delta Mig）

(4) L0 变了：scratch 扩容.

有4个状态，且各自有 action → FSM 即可，即 L0 → L1 → L2 → L3 traversal

额外加一层 tag scratch 检查 ↔ 查找、建立 kernel（填写 template） ↔ 执行 kernel.

→ 2MB 跑 PoC：太小，处于 α-β 线性边界.

Eval：Metrics：① Init Time（CIT）：初始资源的 Materialization 步骤推到 Runtime.

↑ 扯淡 对比的 NCCL v2.19 不支持 Runtime ...

且从 S 级到 ms 级区别不会太大，就只跑一次而已.