CANN HCCL集合通讯算法介绍

作者：陈超

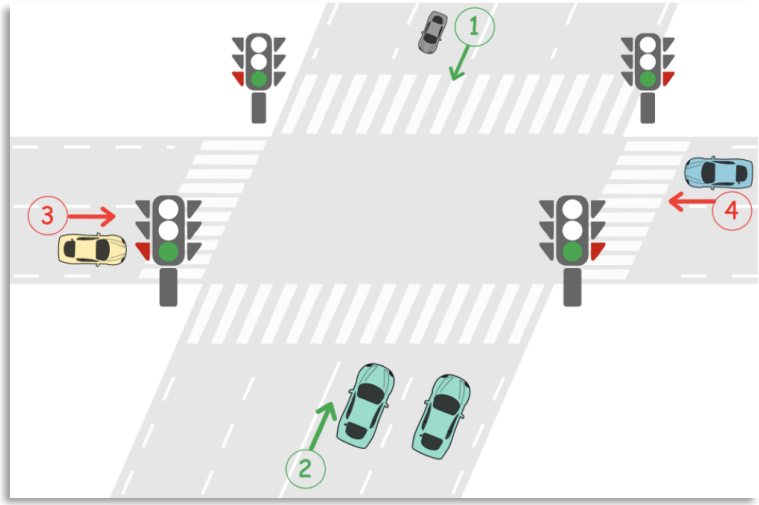
时间：2025/12/23

目录

为什么需要HCCL？－全局流量统一规划，有序完成通信

没有红绿灯：各种拥塞

需要交通枢纽的管控和规划



场景：1万颗芯片组成的计算集群，做一次全局求和通信，通信完成后继续计算

思考：

如果没有统一流量规划，会发生什么事情？

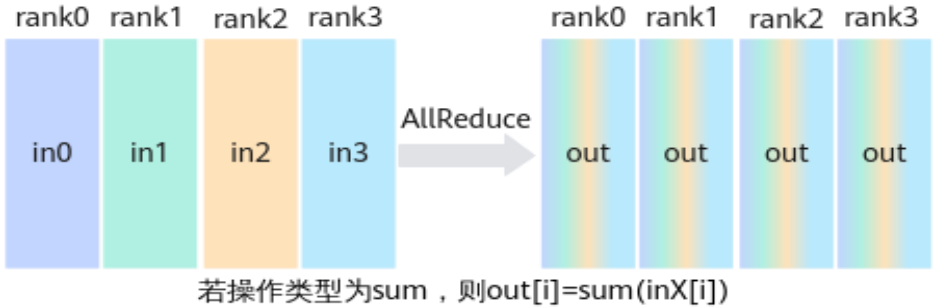
HCCL实现全局流量统一规划，满足复杂拓扑中流量有序交换，最大化集群通信效率

CANN

HCCL使用：集合通信算子举例

AllReduce：

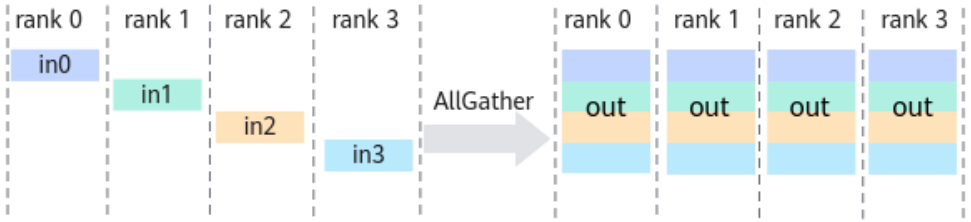
收集通信域内所有节点数据并做Reduce操作



若操作类型为sum，则out[i]=sum(inX[i])

AllGather：

每个节点收集通信域内所有节点的数据



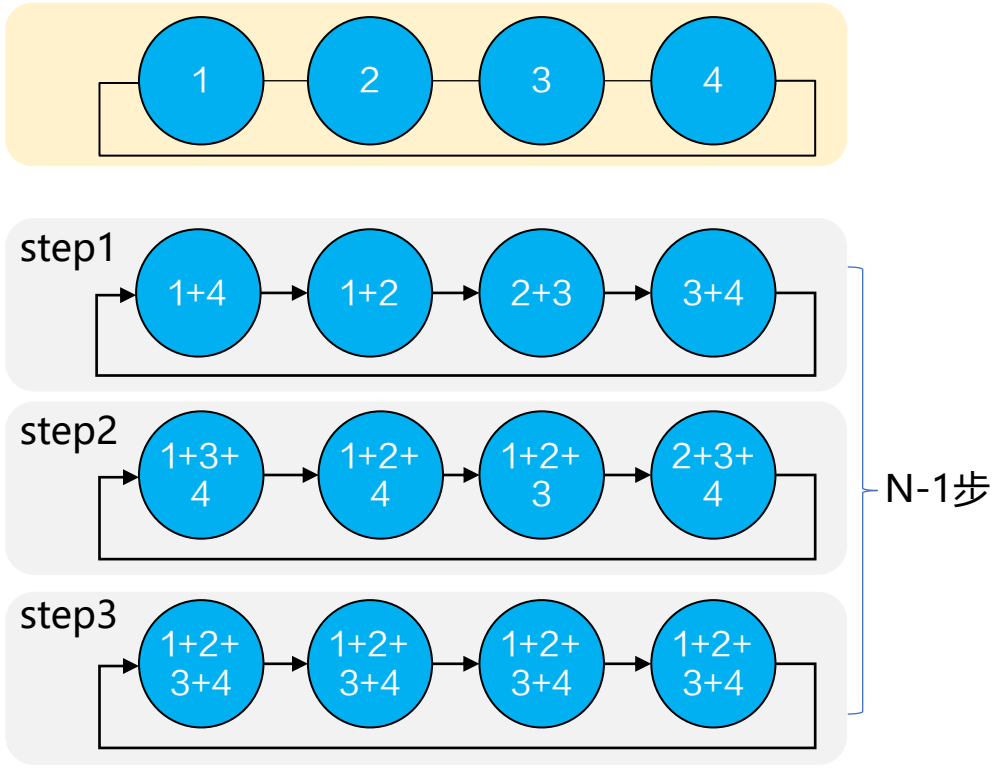
https://gitcode.com/cann

在AI框架层（pytorch等）调用通信算子接口

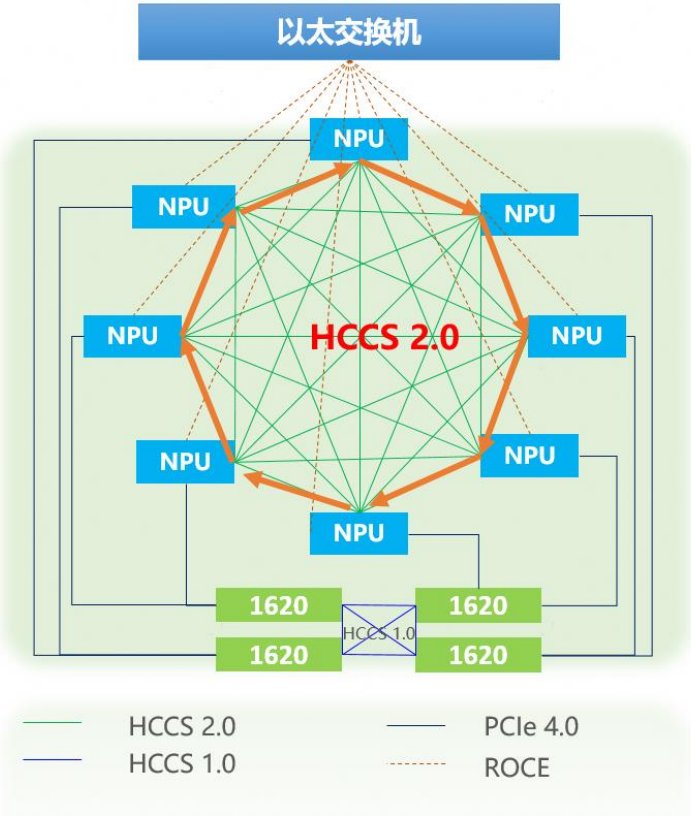
```
int Process(HcclRootInfo *rootInfo, int32_t deviceId, uint32_t devCount)
{
    // 设置当前线程操作的设备
    ACLCHECK(aclrtSetDevice(device_id));
    // 初始化集合通信域
    HcclComm hcclComm;
    HCCLCHECK(HcclCommInitRootInfo(devCount, ctx->rootInfo, ctx->device, &hcclComm));
    aclrtStream stream;
    ACLCHECK(aclrtCreateStream(&stream));
    // 申请集合通信操作的 Device 内存
    ...
    HCCLCHECK(HcclAllReduce(sendBuf, recvBuf, devCount, HCCL_DATA_TYPE_FP32,
HCCL_REDUCE_SUM, hcclComm, stream));
    // 阻塞等待任务流中的集合通信任务执行完成
    ACLCHECK(aclrtSynchronizeStream(stream));
    ...
}
int main()
{
    // 设备资源初始化
    ACLCHECK(aclInit(NULL));
    ...
    int rootRank = 0;
    ACLCHECK(aclrtSetDevice(rootRank));
    // 生成 Root 节点信息，各线程使用同一份 RootInfo
    void *rootInfoBuf = nullptr;
    ...
    HCCLCHECK(HcclGetRootInfo(rootInfo));
    // 启动线程执行集合通信操作
    std::vector<std::thread> threads(devCount);
    for (uint32_t i = 0; i < devCount; i++) {
        threads[i] = std::thread(Process, rootInfo, i, devCount);
    }
    for (uint32_t i = 0; i < devCount; i++) {
        threads[i].join();
    }
    ...
}
```

集合通信算法示意（以Ring为例）

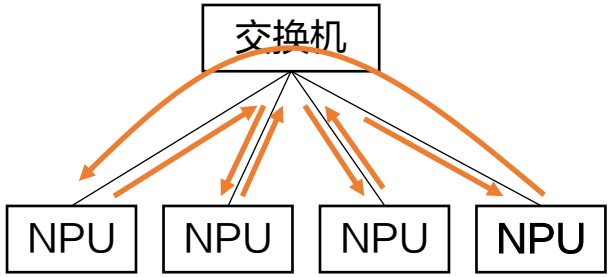
Ring算法逻辑步骤示意



Ring算法在昇腾A2节点内物理路径示意



Ring算法在CLOS(FatTree)组网物理路径示意



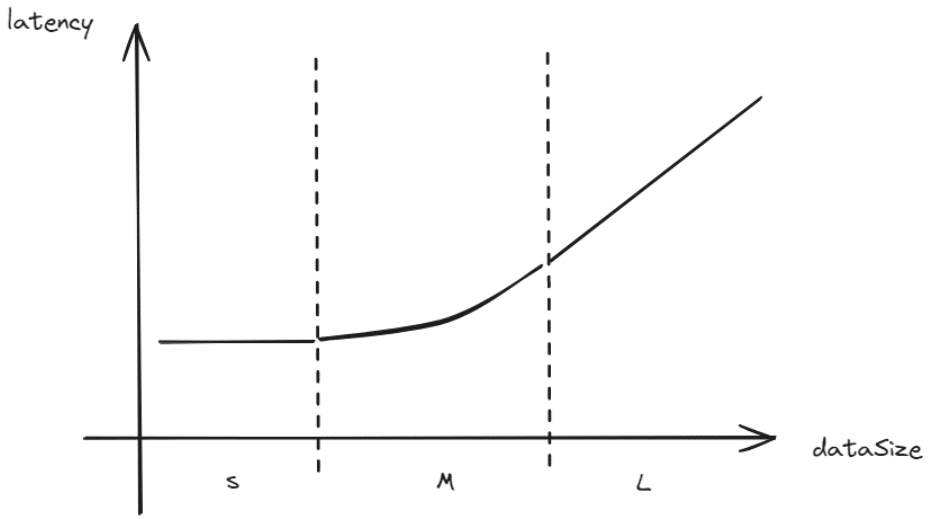
CANN

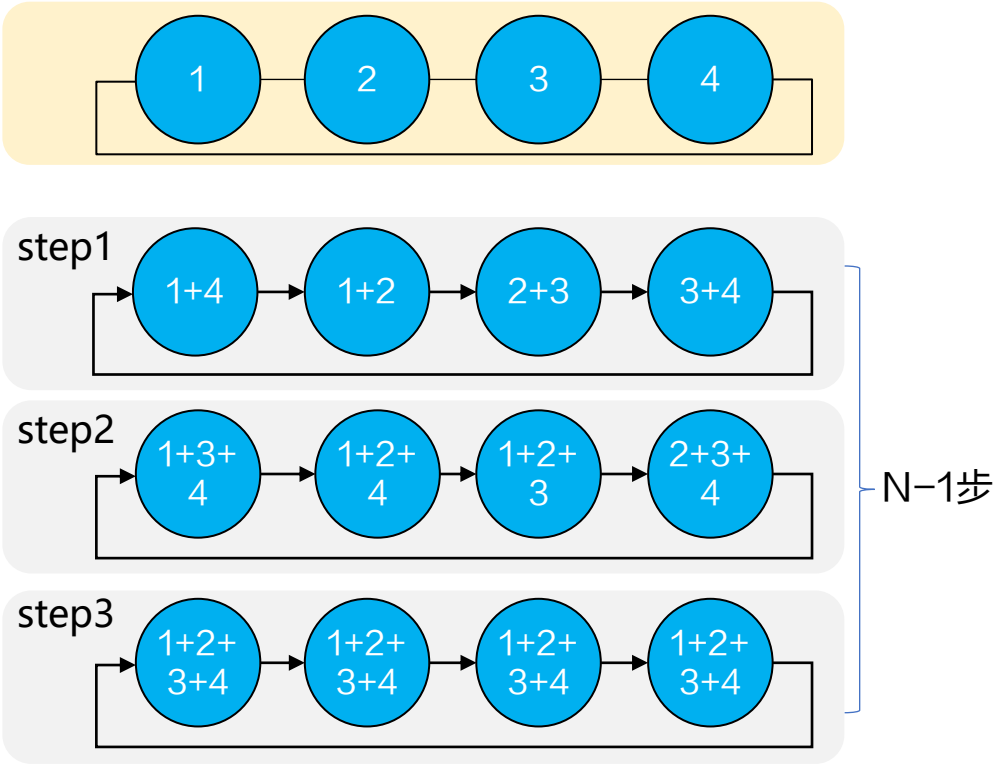
集合通信算法性能：基本性能建模，同步开销和带宽是关键因素

$$latency = A\alpha + B\beta$$

A: 通信步骤（轮次）
α: 同步一次的耗时，是固定开销
B：数据量
β：单位数据的传输耗时，带宽的倒数

对于确定的硬件系统及规模，α, β是固定的





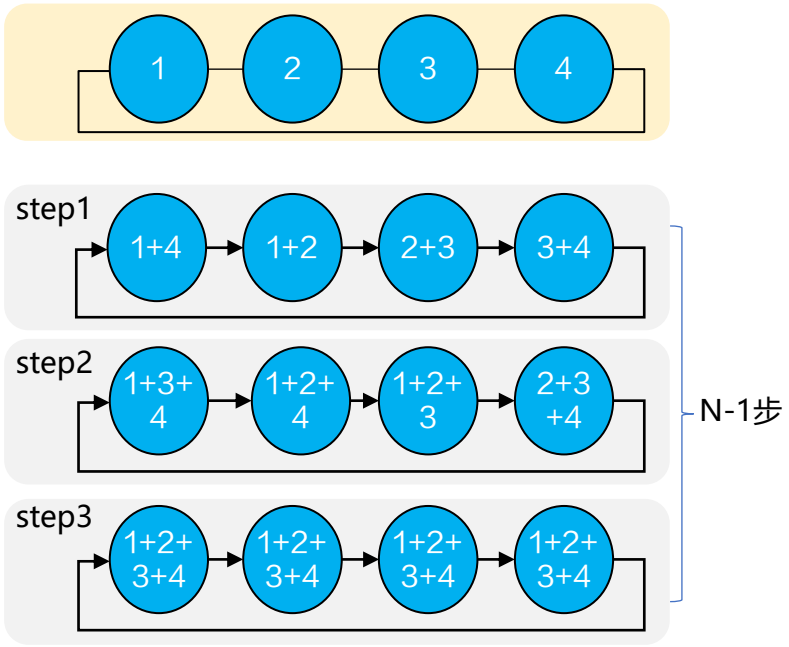
A = N−1，（N为总rank数）
B = inputData

CANN

集合通信性能：一个通信算子有不同通信算法实现，不同算法适用不同场景

Ring算法

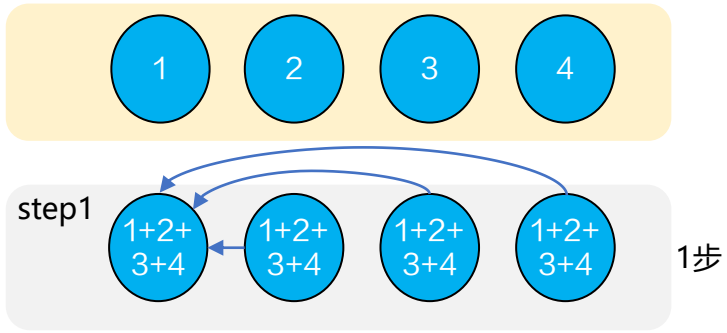
输入数据量：D, 规模N(N=4)



$$latency = (N-1) * \alpha + D * \beta$$

- 步数N−1
- 能带宽用满，且不要求N为二次幂
- 额外好处：交换设备上流量不冲突

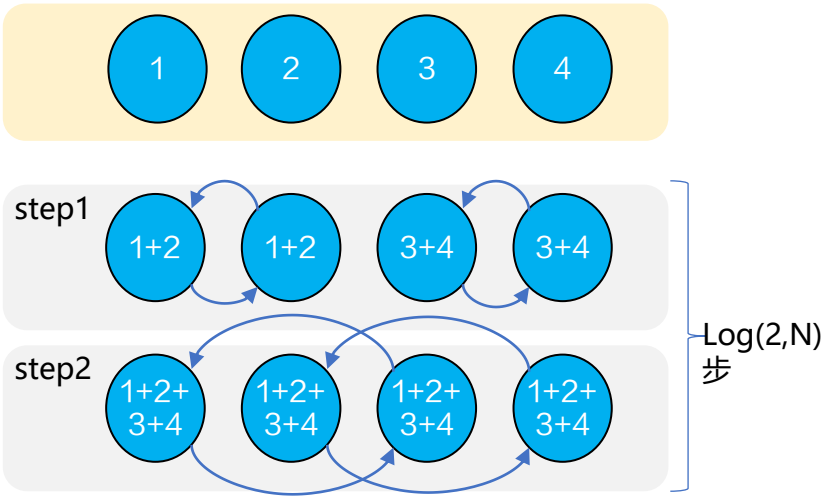
Mesh算法



$$latency = \alpha + D * \beta$$

- 步数1，适合小数据量
- 能带宽用满，但Clos组网容易冲突，适合硬件Mesh拓扑

Half doubling算法



$$latency = logN * \alpha + D * \beta$$

- 步数logN,适用大规模集群
- N不为二次幂，链路空闲

CANN

集合通信性能：根据硬件拓扑、数据量大小选择合适的算法

Mesh OneShot算法

	Rank0	Rank1	Rank2	Rank3
初始	A	B	C	D
step1	ABCD	ABCD	ABCD	ABCD

$$latency = \alpha + D * \beta$$

- 步数1，适合小数据量
- 数据量有冗余
- 能带宽用满，但Clos组网容易冲突，适合硬件Mesh拓扑

Mesh TwoShot算法

	Rank0	Rank1	Rank2	Rank3
初始	A0	B0	C0	D0
	A1	B1	C1	D1
	A2	B2	C2	D2
	A3	B3	C3	D3
step1	ABCD0	B0	C0	D0
	A1	ABCD1	C1	D1
	A2	B2	ABCD2	D2
	A3	B3	C3	ABCD3
step2	ABCD0	ABCD0	ABCD0	ABCD0
	ABCD1	ABCD1	ABCD1	ABCD1
	ABCD2	ABCD2	ABCD2	ABCD2
	ABCD3	ABCD3	ABCD3	ABCD3

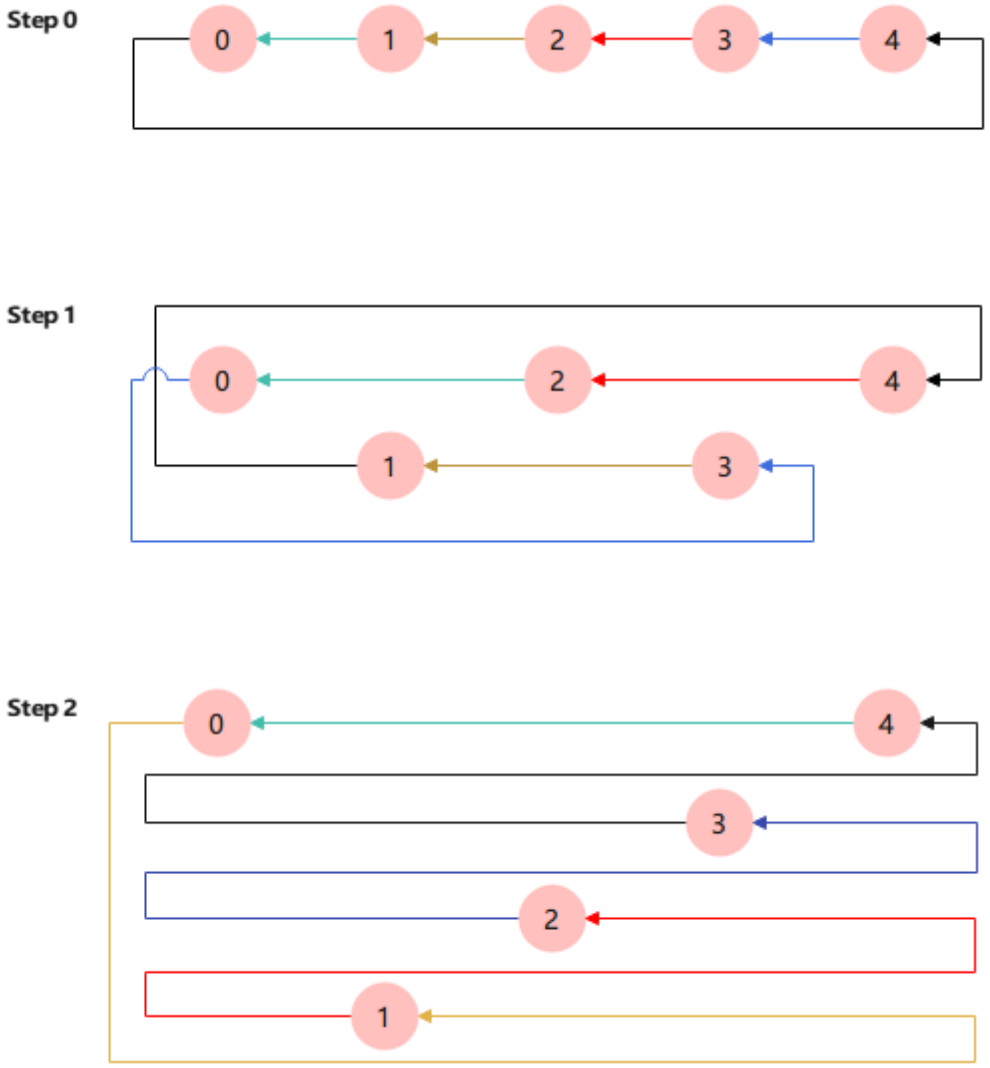
$$latency = 2 * \alpha + (\frac{D}{N}) * \beta$$

- 步数2，不适合小数据量；
- 数据量无冗余，适合大数据量

CANN

目录

NHR算法介绍：一种适用于非2的整数次幂拓扑的集合通信算法



https://gitcode.com/cann

NHR算法定义

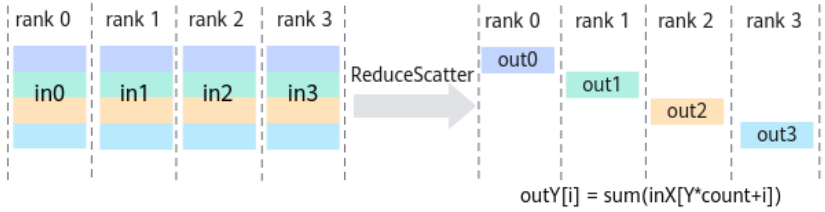
NHR算法是Nonuniform Hierarchical Ring的缩写，即非均衡的层次环算法，目前作为Server间AllReduce、ReduceScatter等通信算子的默认算法。

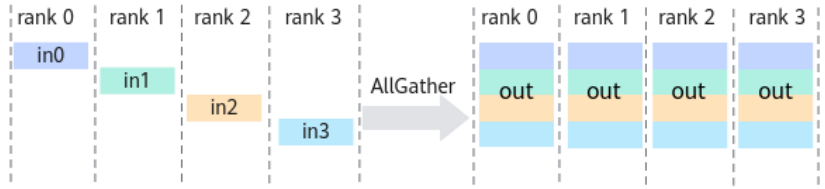
NHR算法优势

1. 可以有效支持2次幂和非2次幂规模集群，非2次幂场景下优于现有的所有算法，实现整体链路带宽最大化利用，是时间复杂度理论最优的通信模式；
2. 通信数据量最多的步骤发生在物理距离最相近的节点之间，可以有效利用物理近邻rank的高带宽优势，这样的跨交换机流量冲突最小；
3. 中小数据包传输时延优化，通过连续数据片合并或者不切分数据片的方式节省传输头开销。

CANN

NHR算法实现（AllReduce实现）

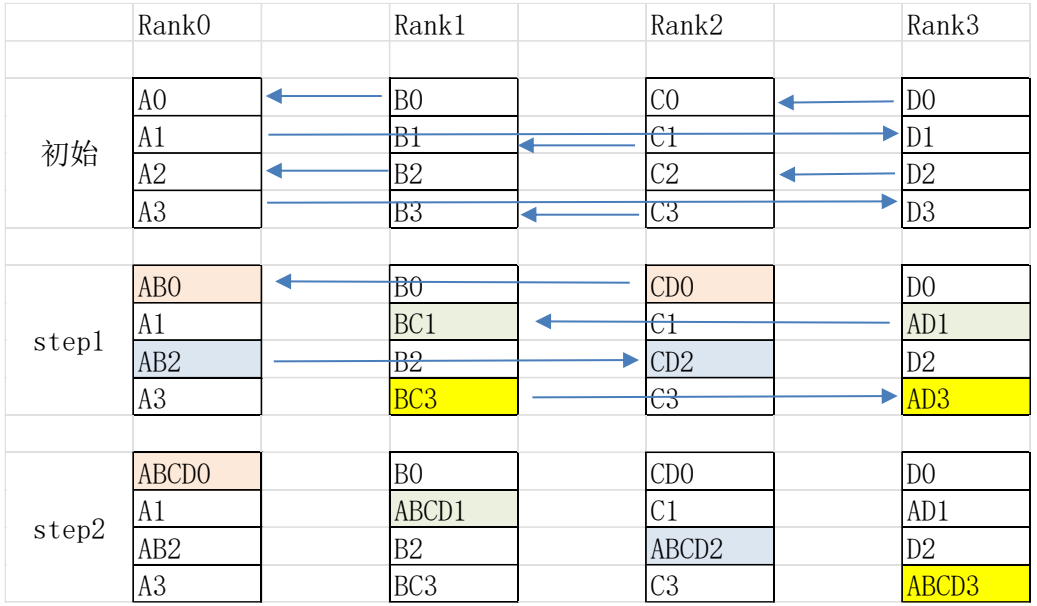


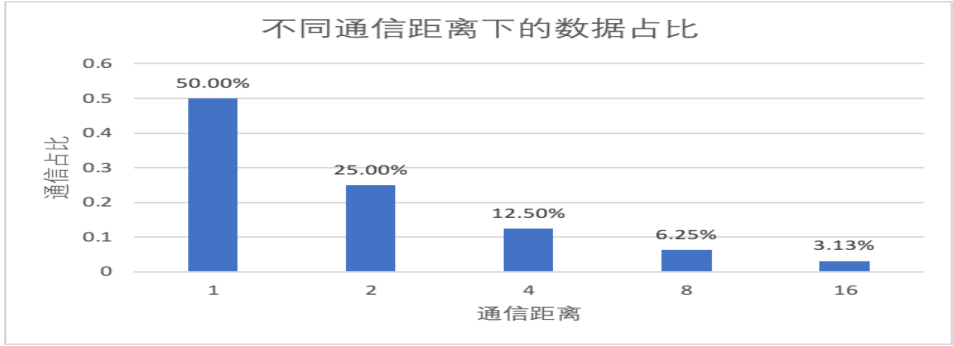


ReduceScatter		单步发送数据量
Step0	3->2[0,2], 0->3[1,3], 1->0[0,2], 2->1[1,3]	2S/4
Step1	2->0[0], 3->1[1], 0->2[2], 1->3[3]	S/4

Allgather		单步发送数据量
Step0	0->2[0], 1->3[1], 2->0[2], 3->1[3]	S/4
Step1	2->3[0,2], 3->0[1,3], 0->1[0,2], 1->2[1,3]	2S/4

表示方式说明：我们一般用Rank号代表通信域中每张卡的编号，用“3->2[0,2]”来简单地表示“Rank3给Rank2发送编号为0和2的数据片”这一步通信过程，箭头表示数据的流动方向为节点3发送给节点2，括号内的数字表示需要发送的数据切片的编号。





- 吸取Ring算法的优势，环形同方向发送，交换机上冲突少
- 带宽能用满
- 较大的数据量传输发生在近距离rank间

CANN

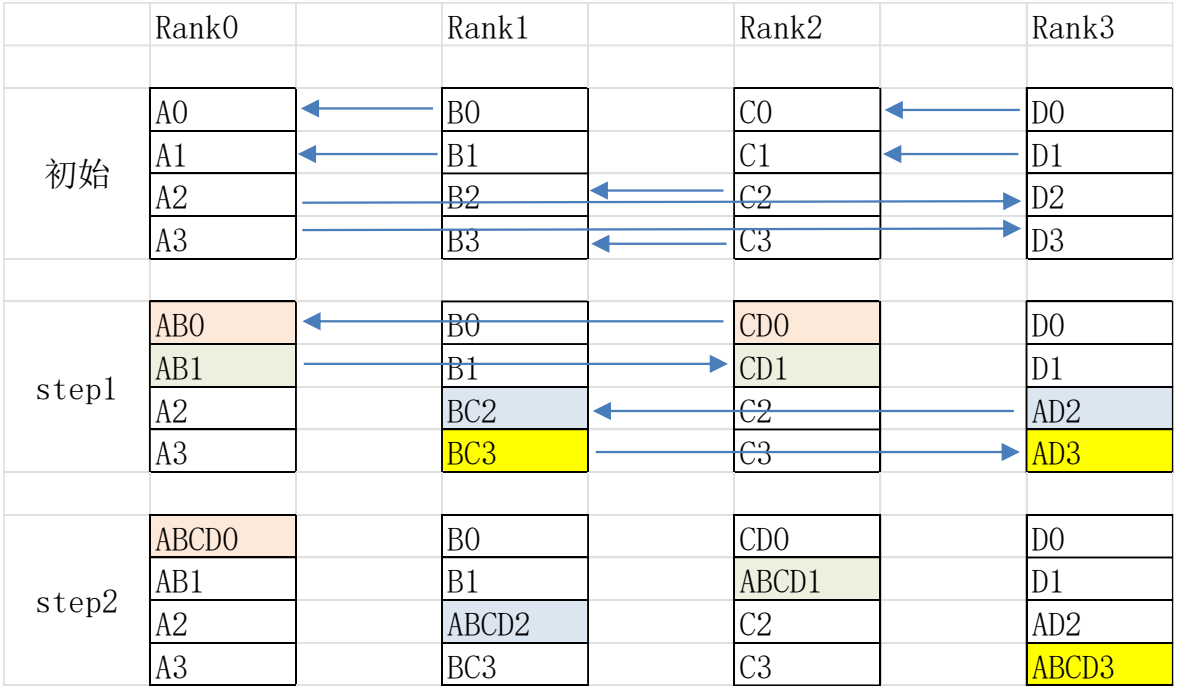
NHR算法实现优化（AllReduce数据重排）

前提：

- 发送同样的数据量，连续数据比非连续数据传输效率高

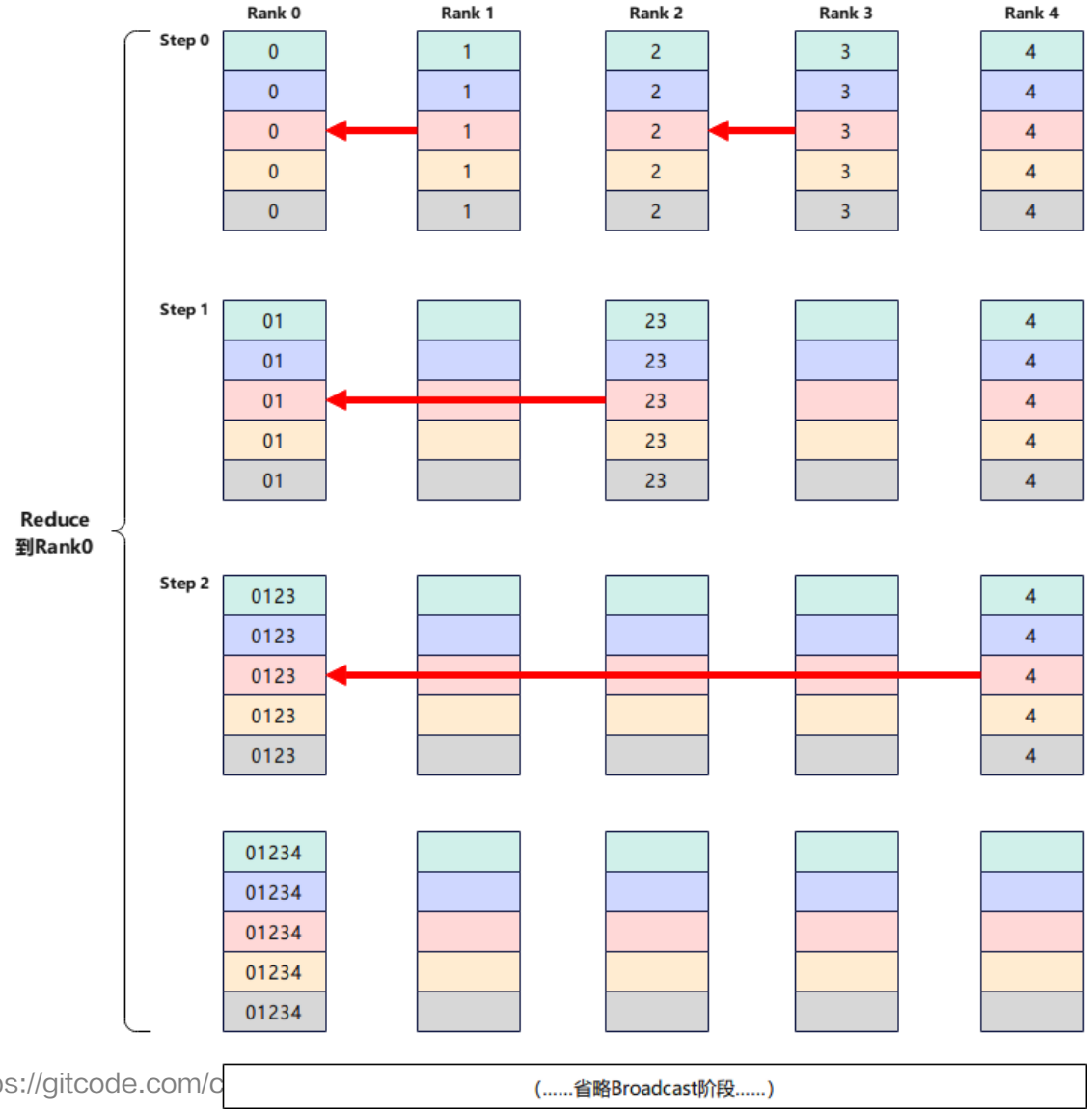
ReduceScatter		单步发送数据量
Step0	3->2[0,1], 0->3[2,3], 1->0[0,1], 2->1[2,3]	2S/4
Step1	2->0[0], 3->1[2], 0->2[1], 1->3[3]	S/4

Allgather		单步发送数据量
Step0	0->2[0], 1->3[2], 2->0[1], 3->1[3]	S/4
Step1	2->3[0,1], 3->0[2,3], 0->1[0,1], 1->2[2,3]	2S/4



- ReduceScatter的结果并不需要标准顺序
- 优化后，step1的数据传输变为连续，提升传输效率

NHR算法实现优化（AllReduce小数据量）



AllReduce = Reduce + Broadcast

算法特点：

1. 先从其他Reduce到Rank0，
   再从Rank0向其他Broadcast
2. 每一步通信都是收发全量数据，而非某一个切片
3. 虽然步数与ReduceScatter+AllGather的实现相同，但因为每一步都是单边通信，可以简化同步信号的使用以降低静态时延

CANN

算法总结：

$$D = round\left(\frac{N-1}{2^{k+1}}\right)$$

ReduceScatter算法总结：Tree[0,1,2,3,⋯,N−1]，slice[0,1,2,3,⋯,N−1]，数据等分为N份
Total Steps S= $[\log_2 N]$，

对于Step k (k=0,1,⋯,S−1)， Rank i

- 发送/接收的数据slice份数：$D = round\left(\frac{N-1}{2^{k+1}}\right)$
- 发送to Rank $(i - 2^k + N)\%N$，先算出$Treeid0 = (i - 2^k + N)\%N$；编号增量$\Delta = 2^{k+1}$，即向第$(i - 2^k + N)\%N$节点发送slice编号为slice[idx of $(Treeid0 - \Delta \cdot m)$ in Tree[]]的数据，$0 \le m < D$
- 接收from Rank $(i + 2^k)\%N$，先算出$Treeid0 = i$；编号增量$\Delta = 2^{k+1}$，即从第$(i + 2^k)\%N$节点接收slice编号为slice[idx of $(Treeid0 - \Delta \cdot m)$ in Tree[]]的数据，$0 \le m < D$

Allgather算法总结：Tree[0,1,2,3,⋯,N−1]，slice[0,1,2,3,⋯,N−1]，数据等分为N份
Total Steps S= $[\log_2 N]$，

对于Step t (t=0,1,⋯,S−1)， Rank i

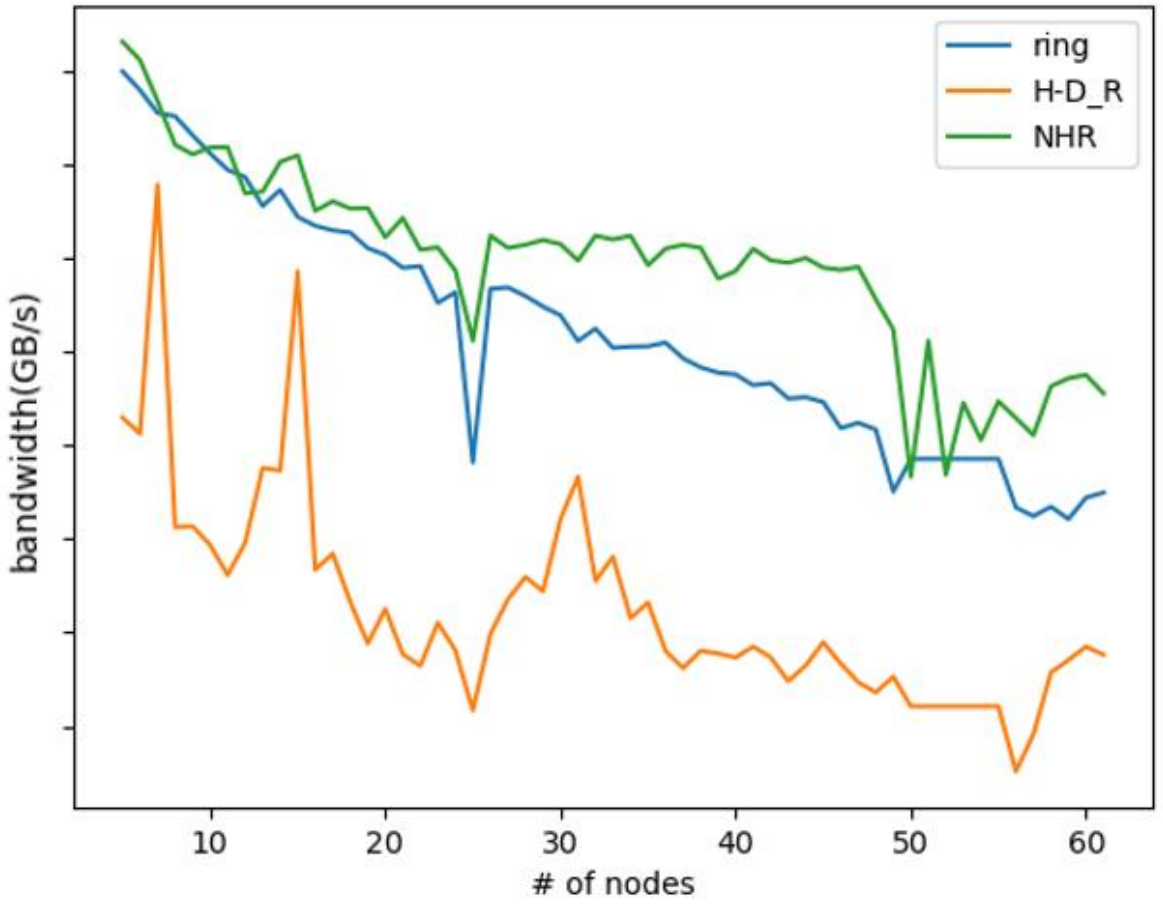
- k = S − 1 − t，发送/接收的数据slice份数：$D = round\left(\frac{N-1}{2^{k+1}}\right)$
- 发送to Rank $(i + 2^k)\%N$，先算出$Treeid0 = i$；编号增量$\Delta = 2^{k+1}$，即向第$(i + 2^k)\%N$节点发送slice编号为slice[idx of $(Treeid0 - \Delta \cdot m)$ in Tree[]]的数据，$0 \le m < D$
- 接收from Rank$(i - 2^k + N)\%N$，先算出$Treeid0 = (i - 2^k + N)\%N$；编号增量$\Delta = 2^{k+1}$，即从第$(i - 2^k + N)\%N$节点接收slice编号为slice[idx of $(Treeid0 - \Delta \cdot m)$ in Tree[]]的数据，$0 \le m < D$

D:发送份数	Step K=0	K=1	K=2	K=3
N=3	1	1		
N=4	2	1		
N=5	2	1	1	
N=6	3	1	1	
N=7	3	2	1	
N=8	4	2	1	
N=9	4	2	1	1
N=10	5	2	1	1
N=11	5	3	1	1
N=12	6	3	1	1
N=13	6	3	2	1
N=14	7	3	2	1
N=15	7	4	2	1
N=16	8	4	2	1

CANN

算法效果（集群实测效果）：

HCCL集合通信库中的AllReduce算子在A2不同集群规模下ring、H-D_R和NHR算法的实测性能趋势（通信数据量为32MB）



CANN

HCCL开源社区已上线，欢迎大家参与共创

开源代码仓

hccl（集合通信库）：
https://gitcode.com/cann/hccl

hcomm（通信基础库）：
https://gitcode.com/cann/hcomm

HCCL SIG组

https://gitcode.com/cann/community/tree/master/CANN/sigs/hccl

https://gitcode.com/cann

CANN

Thank you.

社区愿景：打造开放易用、技术领先的AI算力新生态

社区使命：使能开发者基于CANN社区自主研究创新，构筑根深叶茂、跨产业协同共享共赢的CANN生态

Vision: Building an Open, Easy-to-Use, and Technology-leading AI Computing Ecosystem

Mission: Enable developers to independently research and innovate based on the CANN community and build a win-win CANN ecosystem with deep roots and cross-industry collaboration and sharing.

上CANN社区获取干货

关注CANN公众号获取资讯

https://gitcode.com/cann

CANN